Computer Vision & AI

`I.BA_CVAI` — Zusammenfassung

Author(s) Dominic, Elias, Hannah, Laura
Date 30. May. 2026
Pages 240

Inhaltsverzeichnis

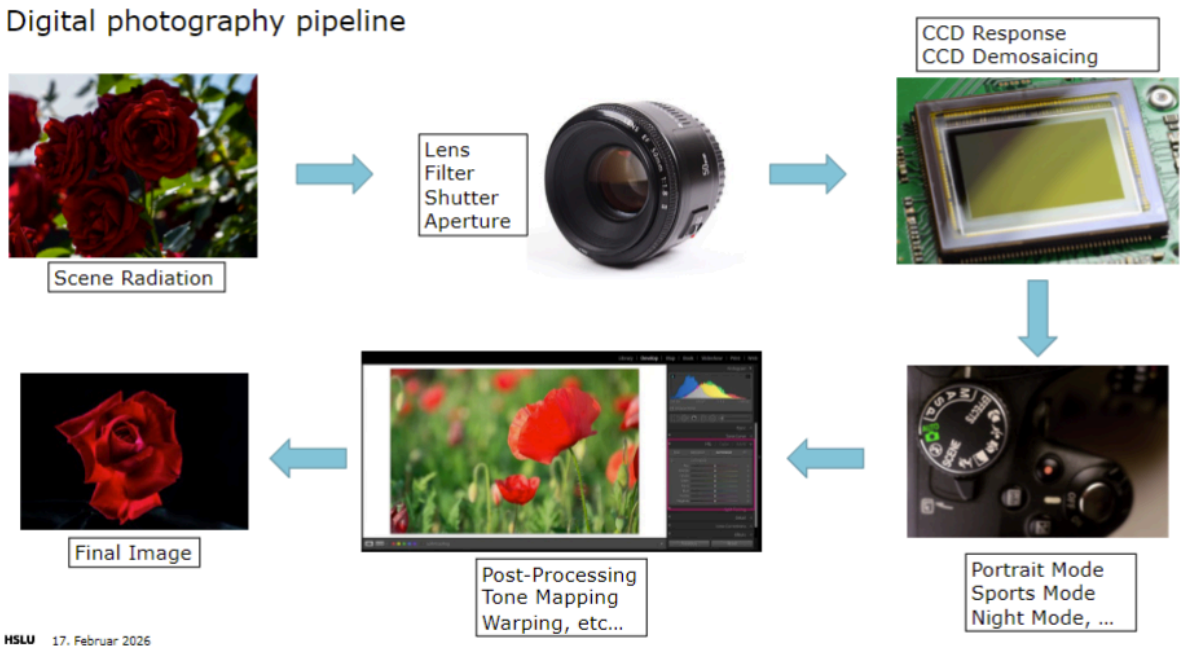
1. Digital Photograpy 📷

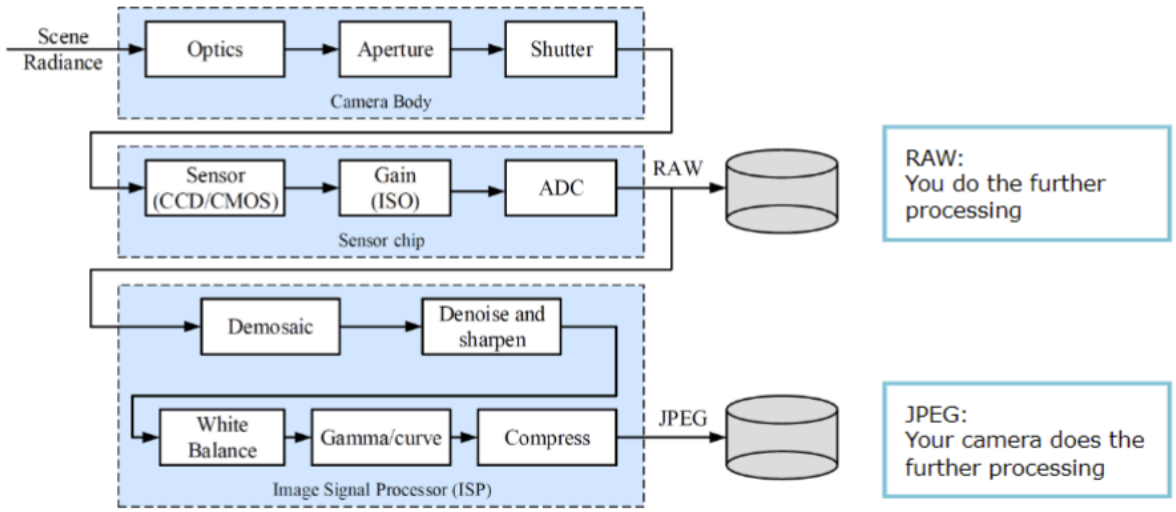
1.1. Instruction

1.1.1. Use of imaging

- Medizinische Bilder
- Satelliten Bilder
- Image Matching / Multi image processing
- HDR Imaging
- Shape from shading
- ...

1.1.2. Digital photography pipeline





RAW: you do the further processing

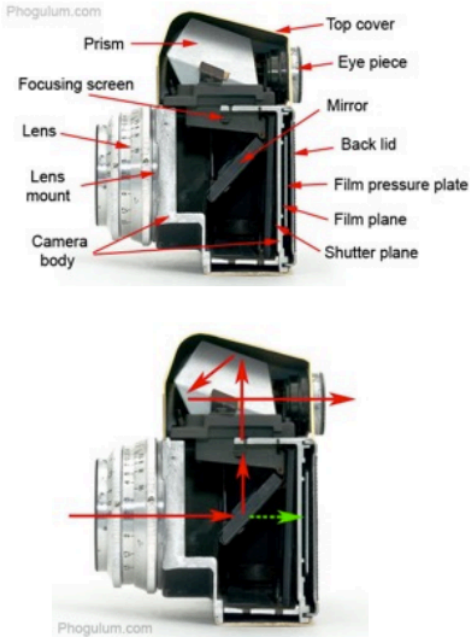
JPEG: your camera does the further processing

- JPEG ist nicht lossless

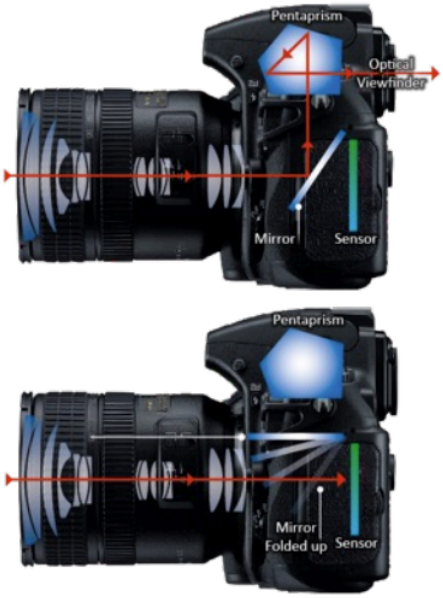
- Von JPEG zurück zum RAW geht nicht mehr, da JPEG verlustbehaftet ist (es wurde zu viel verändert). (Bsp. von 12Bit auf 8 Bit konvertiert)

1.1.3. SLR (Single-Lens Reflex) Cameras

SLR



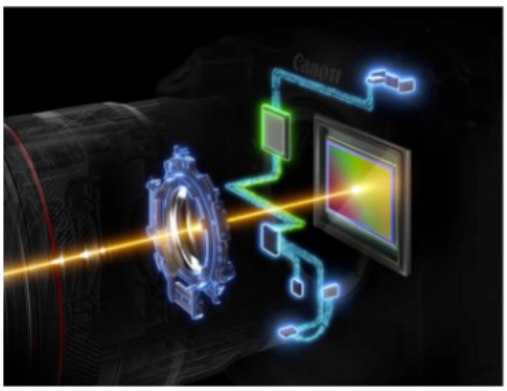
DSLR (Digital SLR)



wurde abgelöst von mirrorless cameras

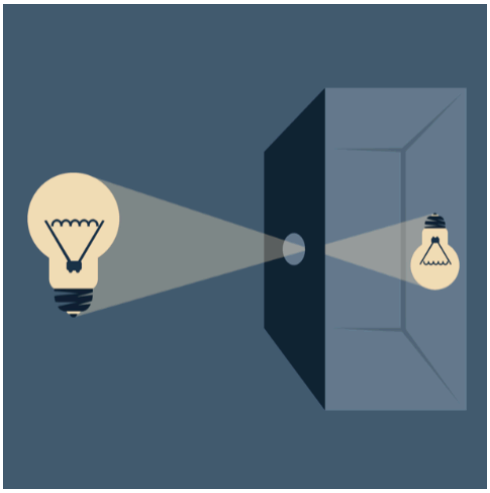
1.1.4. Mirrorless cameras

- einfacher (da kein Spiegel)
- kein Sucher (oder elektronischer Sucher)
- zeigt das vom Sensor empfangene Bild an
- kürzere Batterielebensdauer
- Nachteil: man kann nicht mehr durch das Objekt hindurchschauen



1.1.5. Pinhole Camera

- Für viele Anwendungen ist es okay
- gleiches Prinzip wie das menschliche Auge
- das Bild ist auf dem Kopf

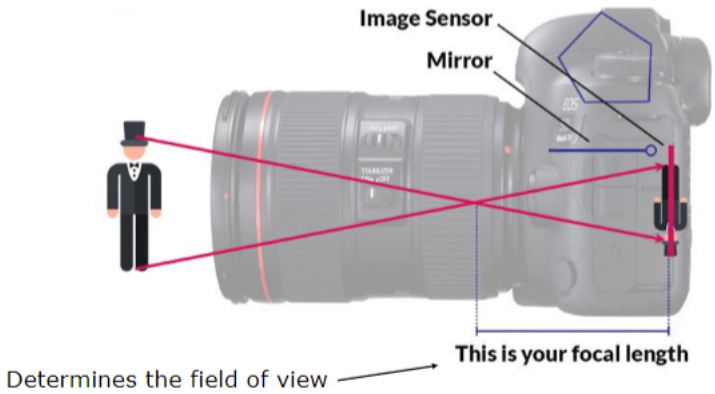


- **Ziel:** 1:1-Entsprechung zwischen Objektpunkt und Bildpixel.
- **Ohne Blende:** Mehrere Strahlen pro Pixel → **Bildunschärfe**.
- **Pinhole (Lochkamera):** Filtert Strahlen durch winzige Öffnung.
- **Effekt:** Nur ein Strahl pro Punkt erreicht das Medium → **Scharfes Bild**.

1.2. Camera Lense Basics

1.2.1. Focal Length

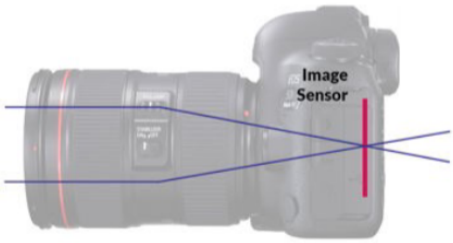
Abstand zwischen dem optischen Zentrum der Linse und dem Bildsensor



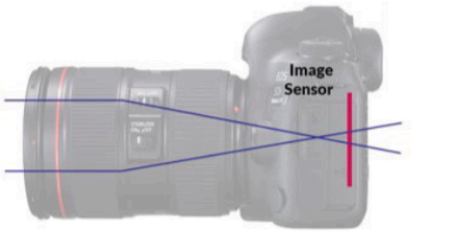
1.2.2. Focus

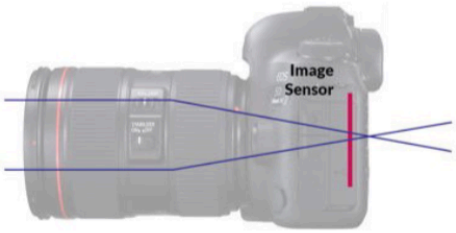
Linsen bündelt die Lichtstrahlen gezielt

scharfes Bild:



unscharfes Bild (Brechung müsste auf dem pinken Balken sein):





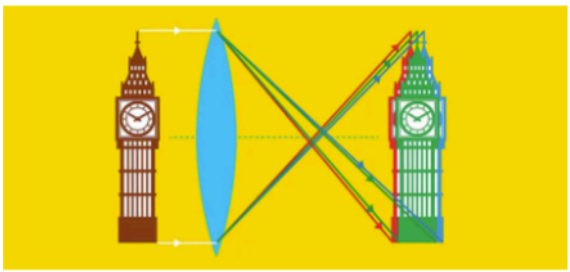
1.2.3. Mehrere Linsen

Die meisten Kameras haben mehrere Linsen:

- chromatic aberration
- lens distortion
- vignetting

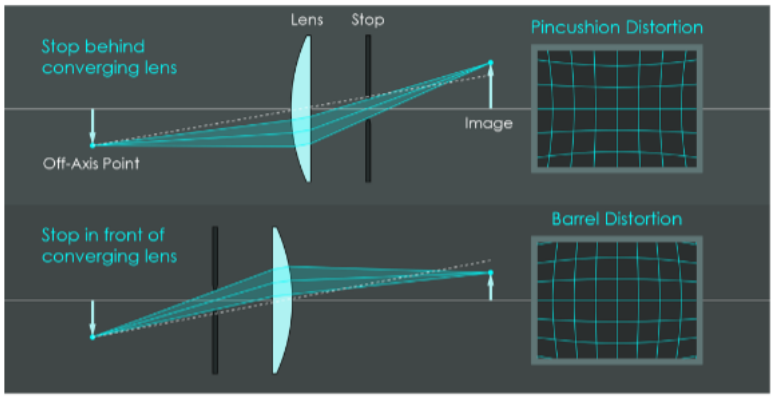
1.2.3.1. Chromatic Aberration

- Linsen sind wie Prisma -> unterschiedliche Farben, da unterschiedliche Wellenlänge, wird anders gebrochen
- das kann dazu führen, dass man beim hereinzoomen zum Beispiel violette Ränder hat



1.2.3.2. Lens Distortion (Linsenverzerrung)

- verzerrtes Bild
- führt vor allem bei Weitwinkel zu Verzerrungen



1.2.3.2.1. Das radiale Modell

- **Grundprinzip:** Verzerrungen treten auf, weil Bildpunkte proportional zu ihrem **radialen Abstand** (Entfernung vom Bildzentrum) verschoben werden.
- **Verschiebung:** Je weiter ein Punkt vom Zentrum entfernt ist, desto stärker ist in der Regel die Abweichung.

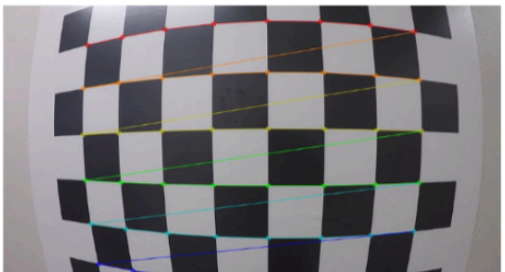
Man unterscheidet zwei Hauptformen, je nach Richtung der Verschiebung:

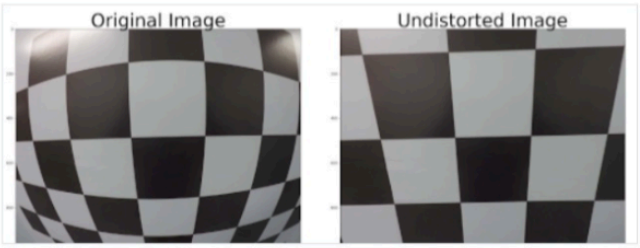
- **Tonnenverzerrung (Barrel distortion): Die Bildpunkte werden zur Mitte hin** verschoben. Gerade Linien krümmen sich nach aussen (wie bei einem Fass).
- **Kissenverzerrung (Pincushion distortion):** Die Bildpunkte werden vom **Zentrum weg** nach außen verschoben. Gerade Linien krümmen sich nach innen.

$\hat{x} = x(1 + \kappa_1)r_c^2 + \kappa_2 r_c^4)$

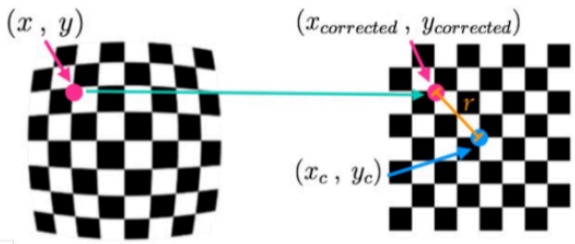
$\hat{y} = y(1 + \kappa_1)r_c^2 + \kappa_2 r_c^4)$

1.2.3.2.2. Parameter estimation





https://medium.com/analytics-vidhya/camera-calibration-with-opencv-f324679c6eb7



$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4)$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4)$$

$$r^2 = (x - x_c)^2 + (y - y_c)^2$$

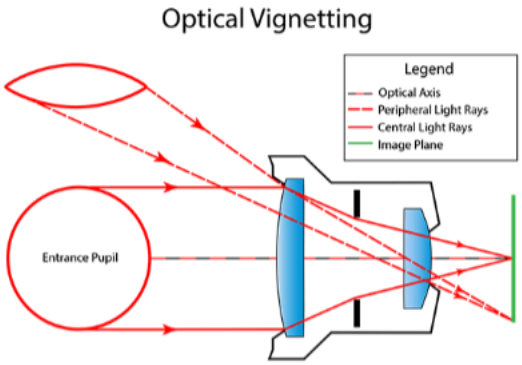
$$\begin{bmatrix} r_1^2 & r_1^4 \\ r_1^2 & r_1^4 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} x_1^{corrected}/x_1 - 1 \\ y_1^{corrected}/y_1 - 1 \end{bmatrix}$$

$$Ax = b$$

1.2.3.3. Vignetting

zum Teil durch dickere oder dünnere Linsen -> gewisse Stellen werden dunkler

-> z.B. alte Aufnahmen ins in den Ecken dunkler

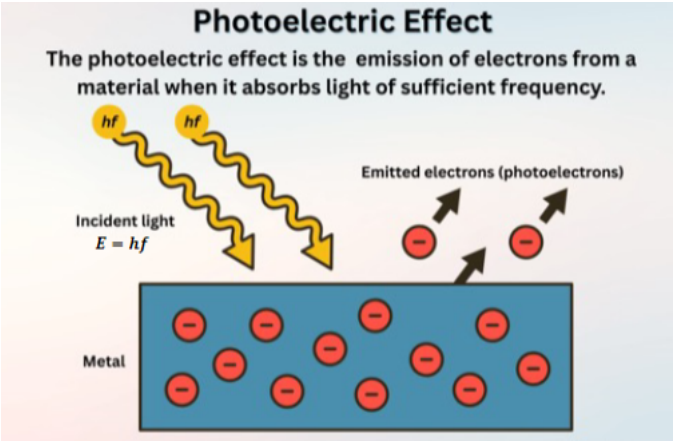


1.3. From light to pixel intensities

1.3.1. Photoelectic effect

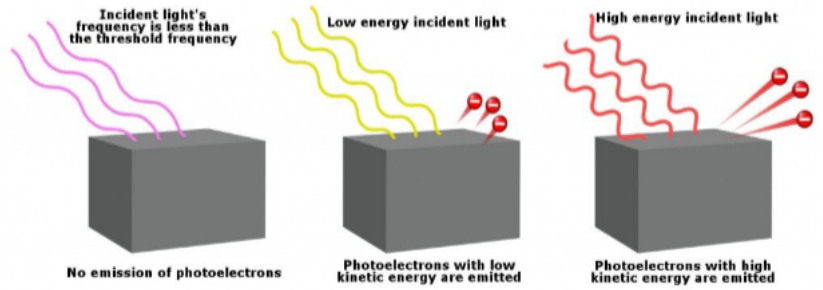
- **Definition:** Die Emission von Elektronen aus einem Material, wenn es Licht mit ausreichender Frequenz absorbiert.
  - Licht trifft auf Metalloberfläche und „schlägt“ Elektronen heraus
  - das passiert aber nicht bei jedem Licht, sondern es kommt auf die Energie des Lichtes darauf (nicht die Helligkeit)
- **Photoelektronen:** Die Elektronen, die durch das Licht herausgelöst werden.

Elektronen werden gemessen (nicht direkt Fotonen)



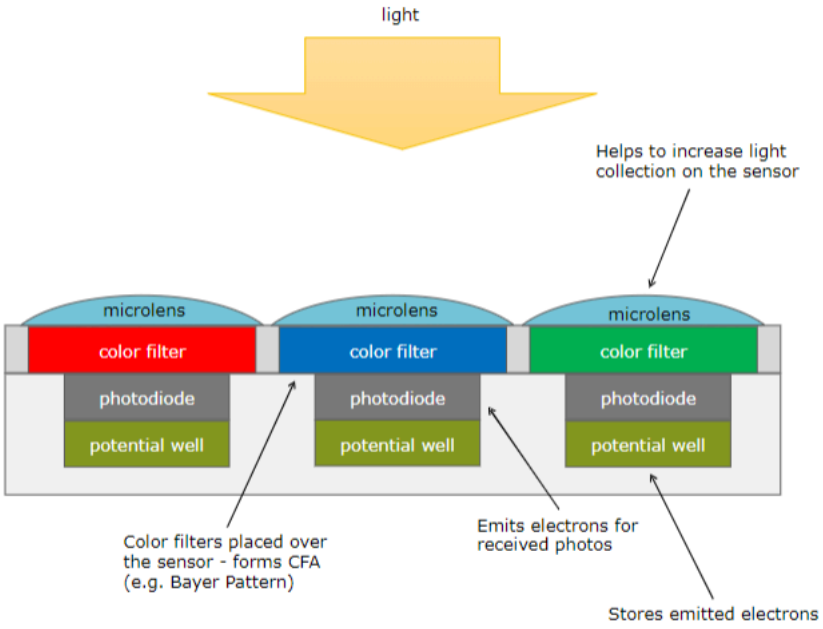
Drei Szenarien:

- **Zu niedrige Frequenz:** Wenn das Licht weniger Energie als die Schwellenenergie des Metalls hat, werden **keine** Elektronen emittiert.
- **Niedrige Energie (gelbes Licht im Bild):** Elektronen werden emittiert, haben aber eine **geringe kinetische Energie** (sie sind langsam).
- **Hohe Energie (rotes Licht im Bild/hohe Frequenz):** Elektronen werden mit **hoher kinetischer Energie** emittiert (sie sind sehr schnell).

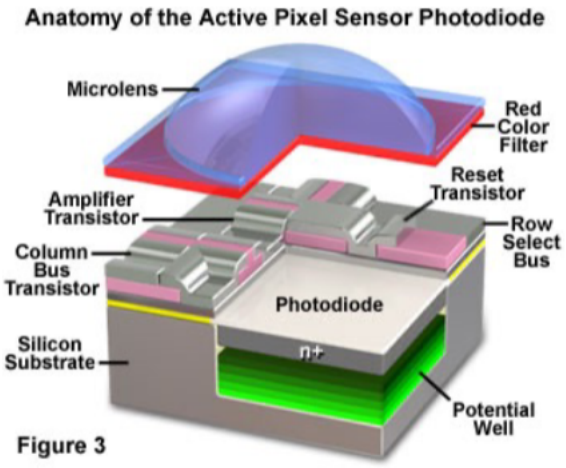


1.3.2. Imaging sensor

- auf Sensor / Fotodiode hat es einen Filter rot grün blau, diese Filter lassen nur diese Lichter rein (rot -> rotes Licht usw.)
- Elektroden werden dann herausgelesen



Potential well: Elektronen bleiben im Topf gefangen



1.3.2.1. Photodiode capacity

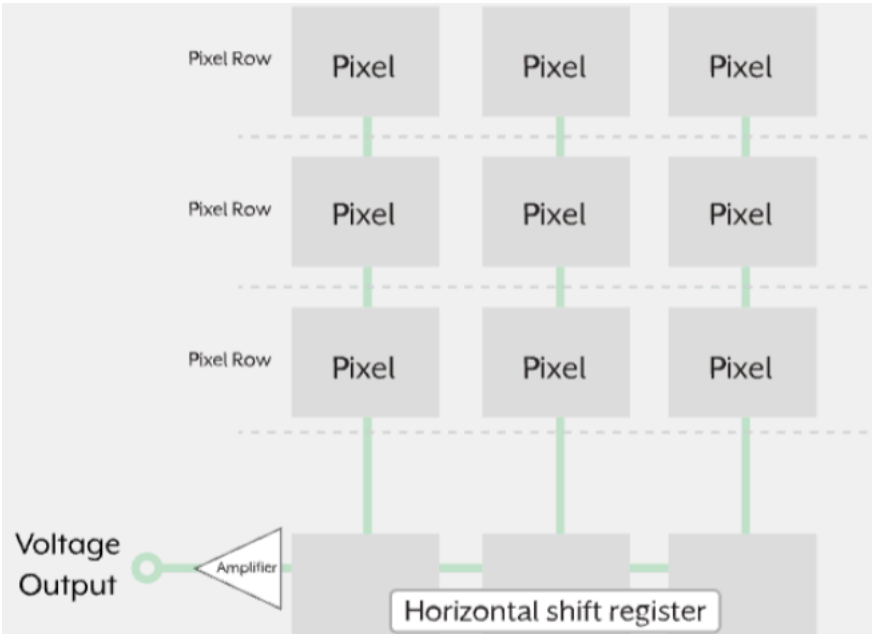
- Wenn die Kapazität aufgebraucht ist, kommt es zu blooming
- es wird einfach weiss

1.3.2.2. Aufbau Sensor

Auslesung von Pixel -> CCD, CMOS

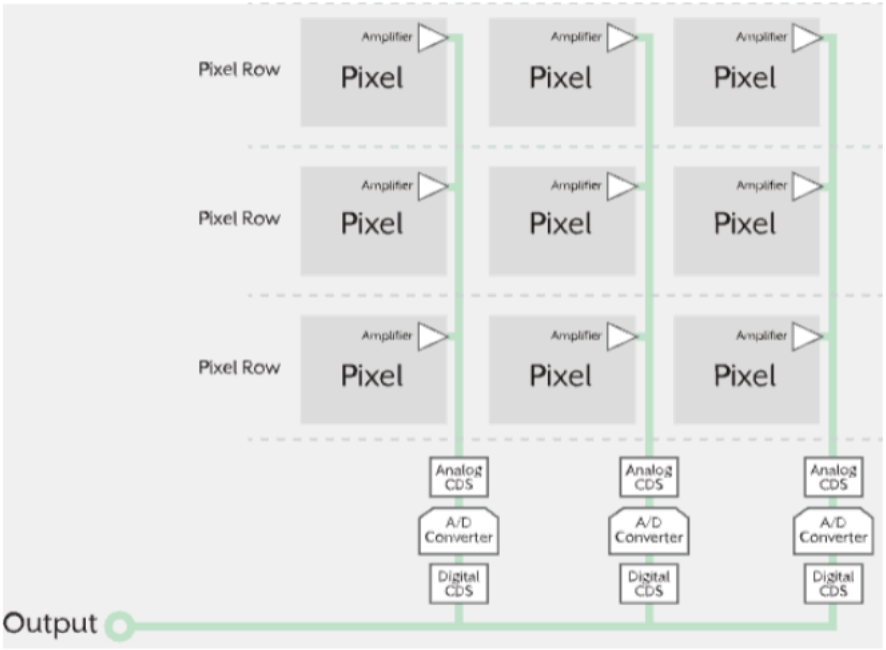
1.3.2.2.1. Charged Couple Device (CCD)

- Global shutter: stoppt und startet für alle Pixel zur selben Zeit
- Eigenschaften
  - low noise
  - high dynamic range
  - mittlere Möglichkeit um Bilder schnell hintereinander zu erstellen
  - anfällig für smearing und blooming



1.3.2.2.2. Complementary Metal-Oxide Semicondoctors (CMOS)

- Rolling shutter: Pixel werden nach einander ausgelesen
- Eigenschaften
  - low to very low noise
  - very high dynamic range
  - high frame rates
  - kein smearing



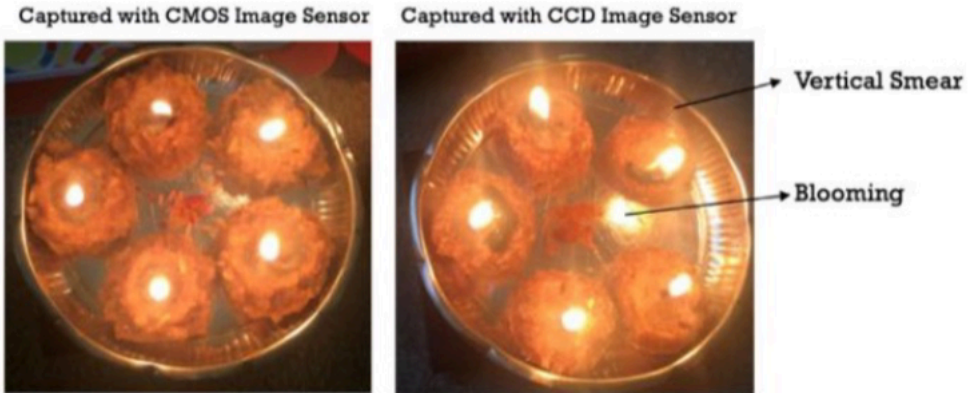
1.3.2.3. Smearing und blooming

Smearing

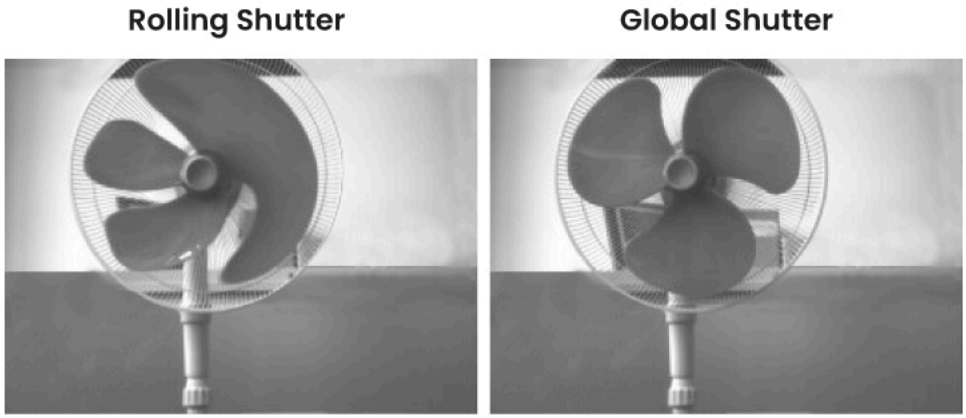
- unerwünschte helle Stellen
- aufgrund vertikalen Transferprozesses

Blooming

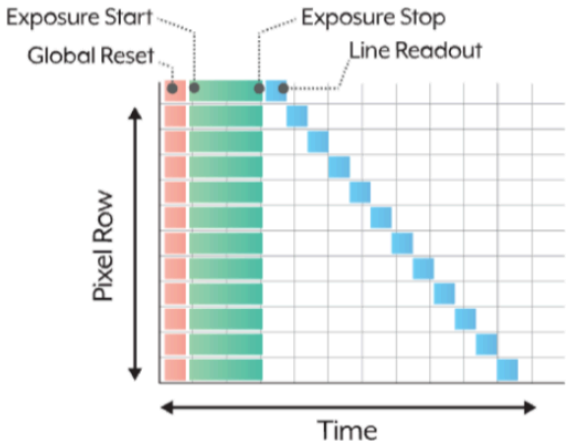
- unerwünschte helle Stellen
- Pixel, die in andere Pixel übergehen

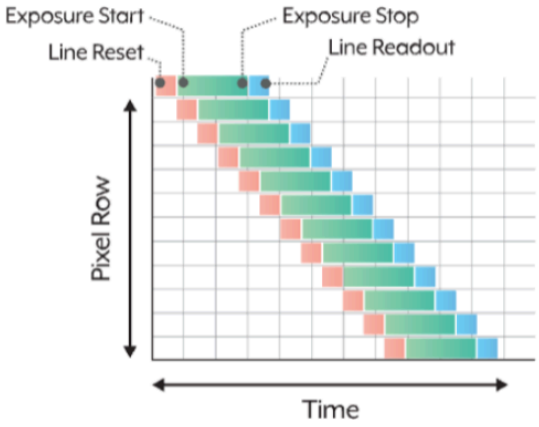


1.3.2.4. Global Shutter Timing vs. Rolling Shutter Timing



- links: Global Shutter
- rechts: Rolling Shutter

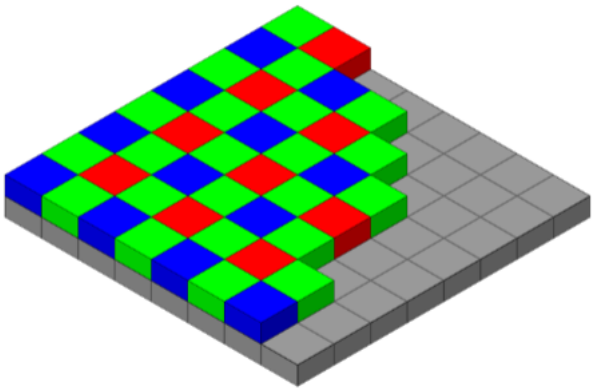




1.3.2.5. Color filter arrays (CFA)

- Bildfilter sind in color filter arrays (CFA) gegliedert
- der bekannteste array ist der Bayer array
- normalerweise in RGB, auch mit CMY (cyan magenta yellow -> z.B. bei Druckern)

Sensor:

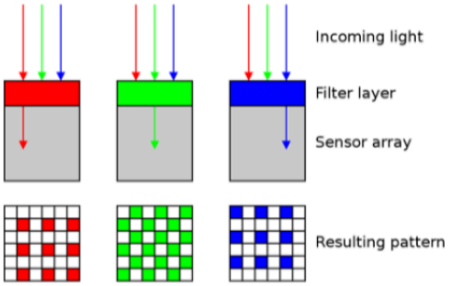


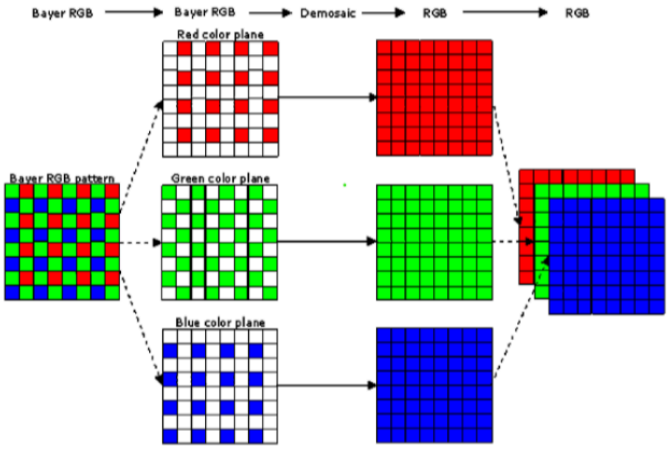
-> hat man gleiche viele Pixel von jeder Farbe? nein grün hat man am meisten, da das menschliche Auge grün am beste unterscheiden kann. Unterschiede in blau kann man weniger gut feststellen als grün

1.3.2.5.1. Demosaicing

(wird oft auch in RAW gemacht)

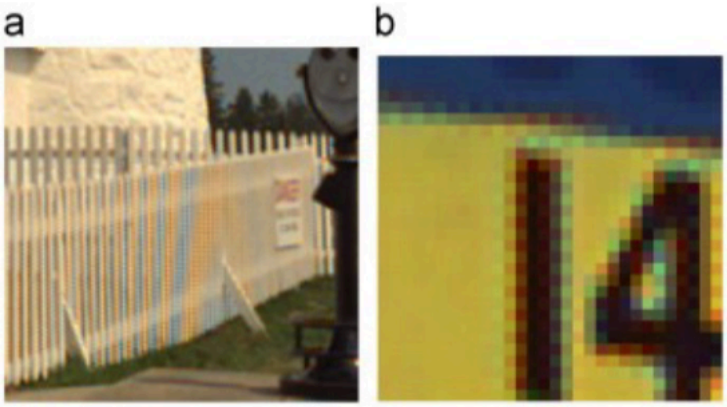
aus den Pixeln wird wieder ein Bild erstellt



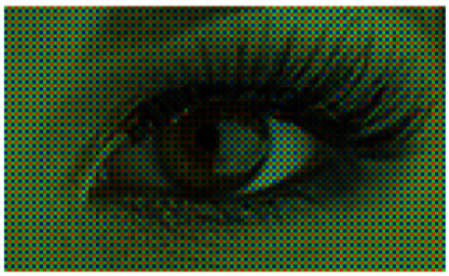


einfachste Methode: lineare interpolation

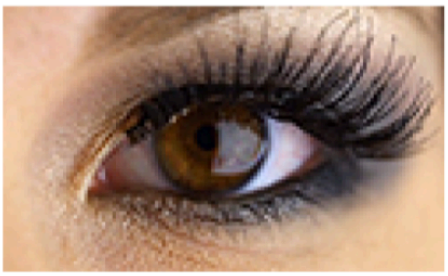
- Durchschnitt der umliegenden Pixel nehmen um den missing value zu berechnen
- Effekt: Unschärfe



Interpolation can lead to artifacts in the image.
- fix using e.g. Freeman method



Bayer image at 400%



Full color at 400%

Full color at 100%

1.4. Taking pictures

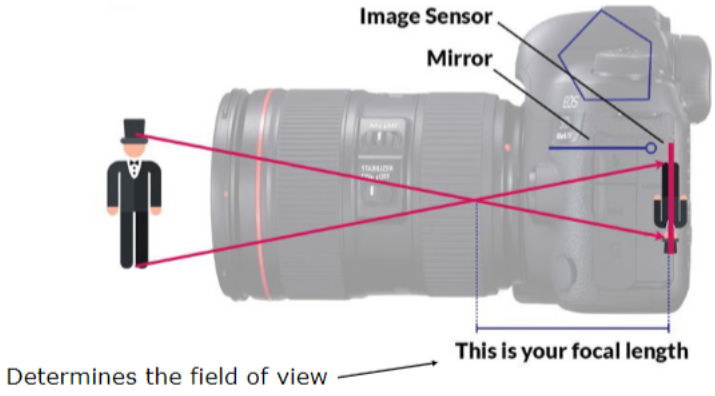
Nachfolgende Einstellungen für Kameras

1.4.1. Focal Length

1.4.1.1. Wide angele vs. Telephote

Weitwinkel: bringt vieles aufs Bild, grössere Winkel

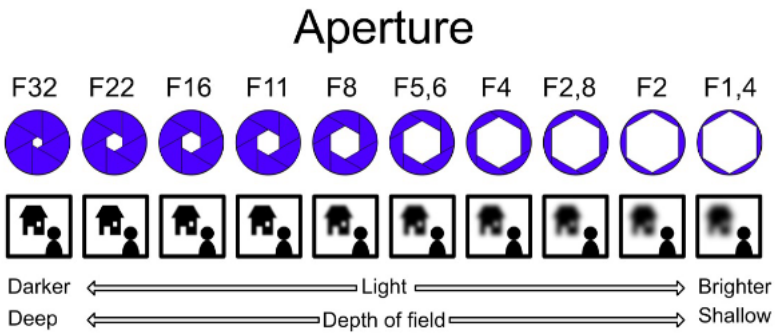
Telephoto: holt fernes nah



1.4.2. Aperture

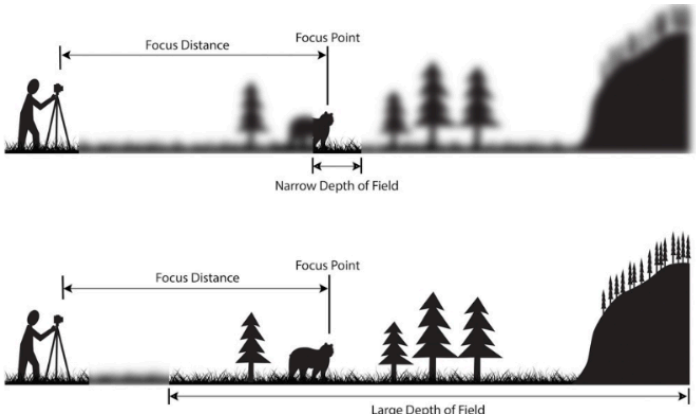
- wie viel Licht durch das Objektiv hindurchgeht
- Fokus ändert sich

je Teuer die Objekte desto kleiner die Zahl z.B. bis zu 1.2



1.4.2.1. Depth of field

je nach Blende ändert sich die Tiefenschärfe -> Frage ist wie viel mehr ist noch scharf kleine Tiefenscharf ist wenig scharf



1.4.3. shutter speed

- wie lange der shutter offen ist
- je länger es offen ist, je mehr Licht
- wenn sich etwas bewegt, kurze Öffnungszeit sieht man die Bewegung fast nicht

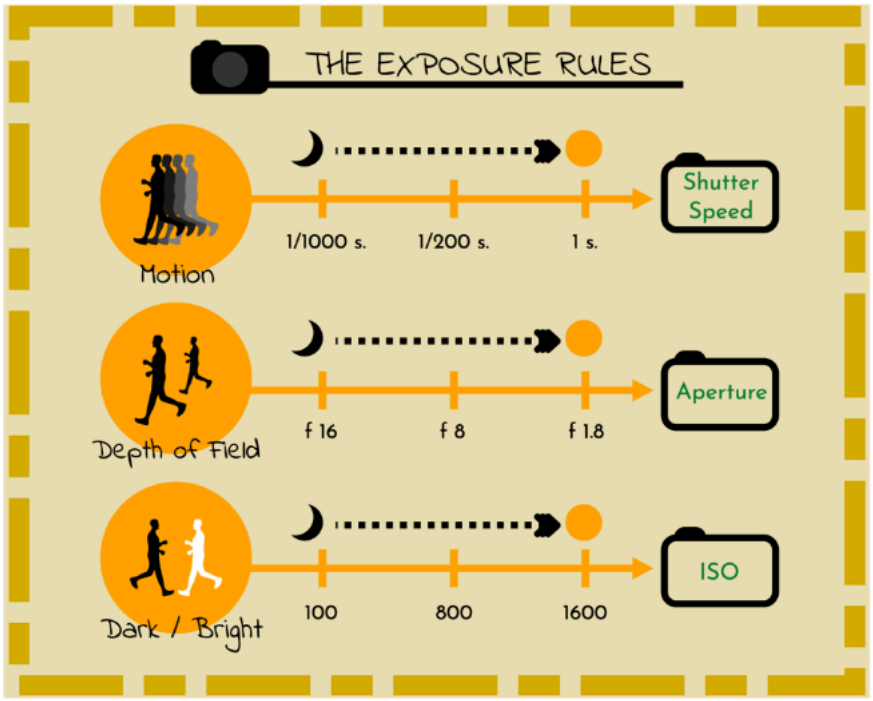
slow: 1/60 or lower

high:1/250 or higher

1.4.4. ISO

- Früher: Sensitivität von einer Filmrolle
  - für Dunkelheit höher wet, für Tag tieferer ISO wert
- Heute: Diodenspannung wird verstärkt -> führt zu mehr rauschen

1.4.5. Putting it all together



Motion und depth of field macht das Bild aber auch dunkler und heller

1.4.6. white balance

- das menschliche Gehirn korrigiert das Weiss automatisch
  - Weisse Wand: bei Tageslicht rötlicher, bei künstlichen Licht bläulicher
- Kameras machen das nicht, aus diesem Grund muss es korrigiert werden

1.4.6.1. manual vs. automatic white balancing

manual white balancing

- wird nachträglich gemacht, in einem Bildbearbeitungstool

- Objekt im Bild als Referenz nehmen und ganzes Bild normalisieren

automatic white balancing:
- wird von der Kamera gemacht
- Grey world assumption: die durchschnittliche Szene wird zwangsweise grau dargestellt
- white world assumption: das hellste Objekt wird zwangsweise weiss dargestellt
- ausgefeilte Histogrammbasisierte Algorithmen

1.4.7. Dynamic Range
- Ration zwischen dem dunkelten und dem hellsten Teil des Bildes
- higher dynamic range = grösseres Licht Spektrum welches eine Kamera aufnehmen kann
- under-exposure: zu dunkel
- correct-exposure
- over-exposure: zu hell

1.4.7.1. High Dynamic Range (HDR) - Merging RAW (linear) exposures
- Bilder müssen zusammengesetzt werden, damit es möglichst real ist
- für jedes Pixel

Vorgehensweise pro Pixel:
1. **Auswahl:** Prüfen, welche Bilder an dieser Stelle gültige Informationen liefern (nicht zu dunkel/verrauscht und nicht überbelichtet/ausgefressen).
2. **Gewichtung:** Festlegen, wie stark der Wert aus jedem passenden Bild einfliesst (optimale Belichtung zählt mehr).
3. **Berechnung:** Neuer Pixelwert ergibt sich aus der gewichteten Summe der Einzelpixel.

1.4.8. Wieso RAW?
- genauer
  - wir möchten wirklich messen wie viel Radiance es hat
- Bessere Bildbearbeitung

jedoch:
- braucht mehr Speicher
- mann muss sie noch fertig processen

2. Intensity Transformation

2.1. Bildverarbeitung & Computergrafik

Bildverarbeitung (image processing) hängt mit der Computer Grafik (Computer Graphics) zusammen:

- **Bildverarbeitung** -> Objekte und Szenen auf einem Bild erkenne
- **Computergrafik** -> Realistische Bilder von Objekten und Szenen generieren

So können realistische Daten generiert werden und verwendet werden für Machine Learning Applikationen.

2.2. Level der Bildverarbeitung

```
+----------------------+
|                      | Computer Vision
|  High Level Vision   |   Bilder interpretieren
|                      |   Objekte erkennen
+----------------------+
|   Mid Level Vision   | Image Processing
+----------------------+  Bildverbesserung
|                      |  (Kontrastkorrektur, Rauschunterdrückung, etc.)
|   Low Level Vision   |  Erkennen einfacher Merkmale
|                      |  (Kanten, Linien, etc.)
+----------------------+
```

2.3. Was ist ein Bild?

- **Bilddefinition**: Eine Matrix von Pixeln mit RGB-Werten.
- **Funktion**: Ein Input der Koordinaten (X, Y) wird auf eine Output-Intensität abgebildet.

2.4. Image Preprocessing

- Informationen verstärken
  - z.B. Skalieren bis die gesuchte Struktur erscheint
- Störung reduzieren
  - Noise, Belichtungsfehle, jitter, etc.

2.5. Bildverarbeitungsmethoden

Unterschied zwischen:

- **Pixeloperationen**:
  - eine einzelner Punkt wird bearbeitet
  - z.B. Intensität oder Farbe wird geändert
- **Nachbarschaftsoperationen**:
  - Resultat der Operation basiert auf den Pixel der Nachbarschaft
  - Nachbarschaft definiert in Blöcken (z.B. 3x3)

2.6. Intensitätstransformationen

Verändert der Intensität eines Pixel mithilfe von mathematischen Funktionen.

Beispiel:

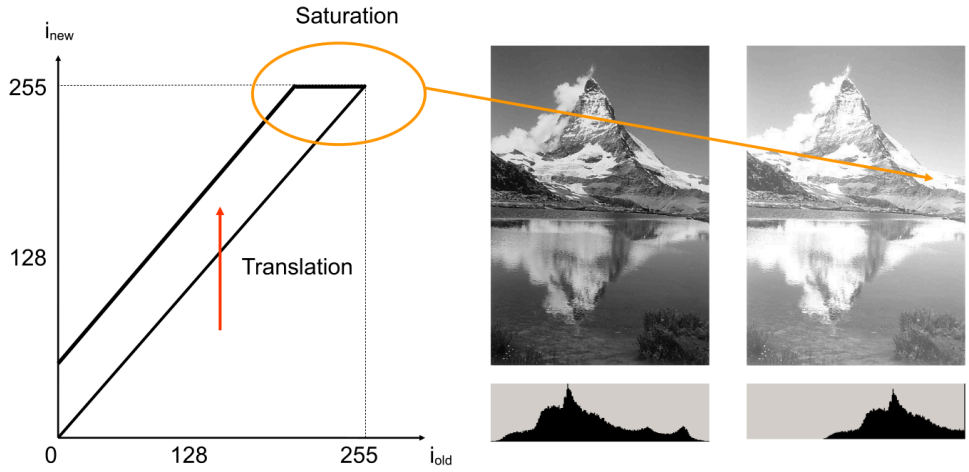
- Threshholding
- Helligkeit anpassen
- Kontrast anpassen
- Gamma Korrektion

2.6.1. Helligkeit

- Verschiebung der Transformationsfunktion nach oben (heller) oder unten (dunkler)
- Anpassung des gesamten Bildes oder spezifischer Teilbereiche möglich
- **Clipping**: Werte > 255 werden auf das Maximum von 255 begrenzt
  - So kommt es zu Übersättigung und Details des Bildes gehen verloren

BEISPIEL:

```py
def f(x):
  y = x + 128 # Transformation
  y[y>255] = 255 # Clipping auf 255
  return y
```

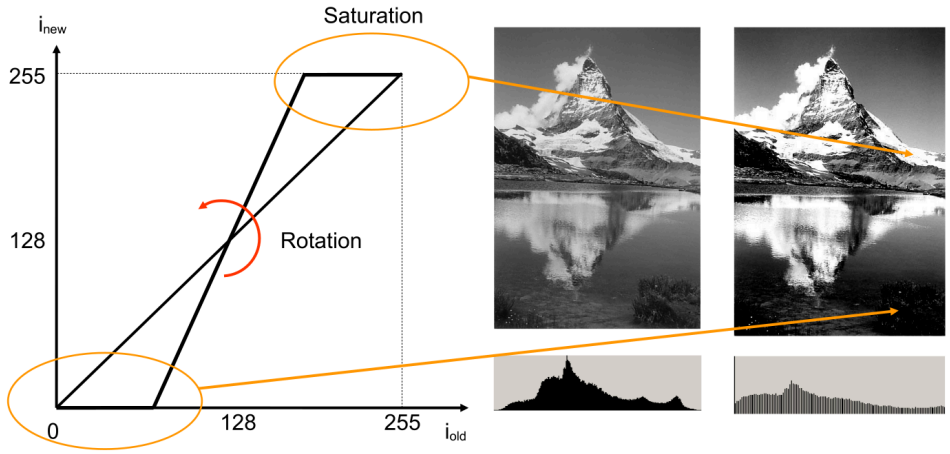


2.6.2. Kontrast

- **Kontrastverändern:** Anpassung von Startpunkt und Steigung der Funktion
- Macht helle Pixel heller und dunkle Pixel dunkler, wodurch das Spektrum gestreckt wird
- **Clipping**: Resultate auf Bereich 0–255 begrenzt

BEISPIEL:

```py
def f(x):
  y = 2*(x-64) # Transformation
  y[y>255] = 255 # Clipping oben auf 255
  y[y<0] = 0 # Clipping unten auf 0
  return y
```

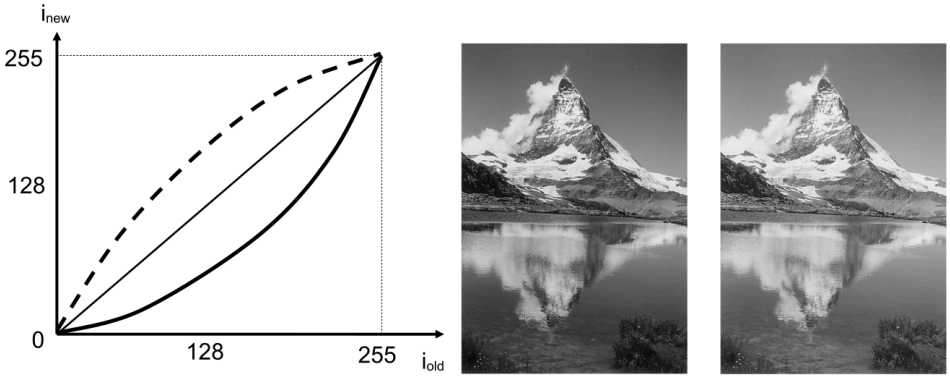


2.6.3. Gamma Korrektur

- Gezieltes Aufhellen oder Abdunkeln eines Bildes

- **Nicht-lineare Skalierung**: Dunkle Bildbereiche werden stärker aufgehellt als helle.
- **Anwendung**: Anpassen von Bildern auf einem Monitor an das menschliche Auge
- **Skale**: Häufig normiert auf $0 - 1$

$f(i) = i^{\gamma}$



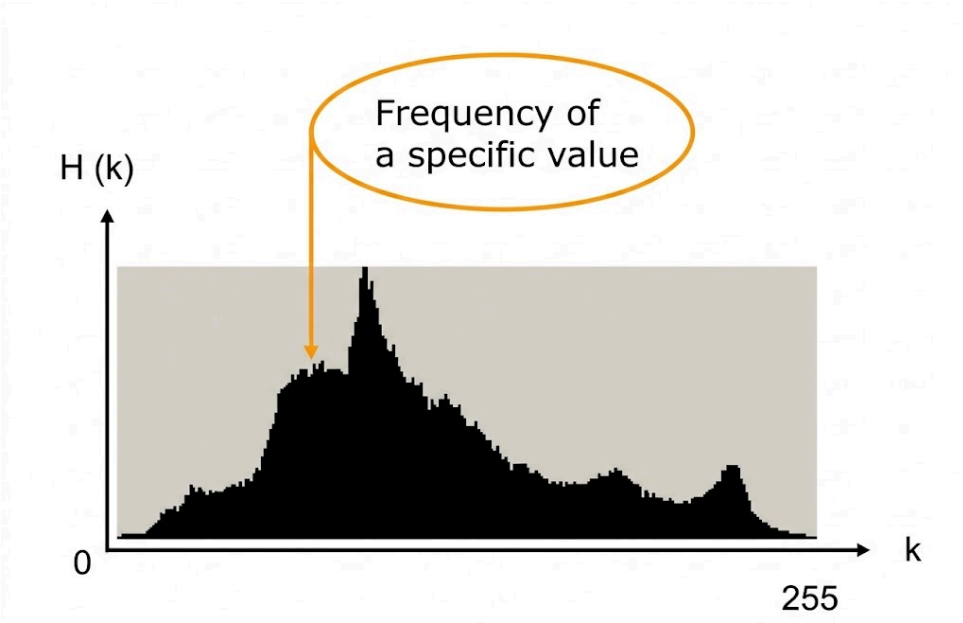
2.6.4. Histogramme

- Häufigkeit in der ein Farbwert/Intensität ins einem Bild auftaucht
- Dabei können die Werte in Bins aufgeteilt werden
  - z.B alle Rot Werte zwischen 0-10 befinden sich im selben Bin

$$H(k) = | \left\{ x_i \mid \underbrace{v_k}_{\text{Untergrenze}} \leq I(x_i) < \underbrace{v_{k+1}}_{\text{Obergrenze}} \right\} |$$

k : Anzahl Bins v_k : Intenstität der Bins

Beispiel:



Zusammenhang Intensitätstransformation mit dem Histogramm:

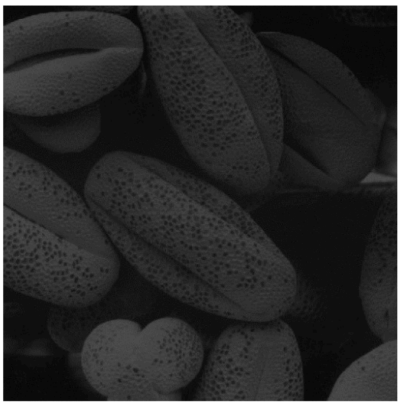
- **Intensitätstransformation**: $i_{\text{new}} = f(i_{\text{old}})$
- **Histogramm**: $H_{\text{new}(i)} = H_{\text{old}(i)} \cdot \frac{1}{f'(i)}$

2.6.4.1. Histogram Ausgleich

- **Ziel**: Grauwerte des Bildes so transformieren das das Histogram so ausbalanciert wie möglich ist

- **Wie?**: Grosse Werte stark verkleinert und kleine stark vergrössert
- **Anwendung**: Auto Kontrast Funktion in Bildbearbeitungstools

Beispiel:



Original Image

Image after histogram equalization

- **Bedingung**: Die neuen Histogrammwerte sind konstant

$H_{\text{new}(i)} = c = \text{const.}$

Das bedeutet:

$f'(i) = \frac{1}{c} H_{\text{old}(i)}$

$f(i) = \frac{1}{c} \int_0^i H_{\text{old}(j)} dj$

Python:

```py
cv2.calcHist() # Histogramm berechnen
cv2.equalizeHist() # Ausgleichen
```

2.6.5. Morphologische Bildverarbeitung

- **Verwendung:** Binärbilder Verarbeitung
- Binärbilder können mit **Threshholds** generiert werden
  - Häufig braucht es weitere Verarbeitungsschritte

Beispiel: Gesichter herausfiltern mithilfe von einem Skin Detector

Morphologische Operationen können mit Set Operationen repräsentiert werden

- Kombination von Bild und Strukturelement (Matrix)

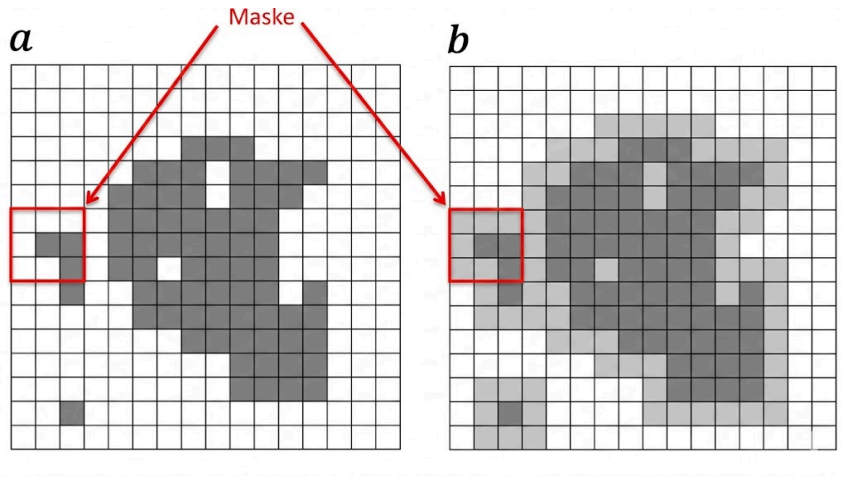
$$\text{Strukturelement(Maske)}: S = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

2.6.5.1. Dilatation (dilute)

Bei einer dilate Operation werden Pixel aufgeblasen:

- Löcher werden geschlossen
- Strukturen werden grösser
- `cv2.dilate(img, kernel, iterations=1)`

$$A \oplus B = \{x \mid A \cap S_x \neq \emptyset\}$$

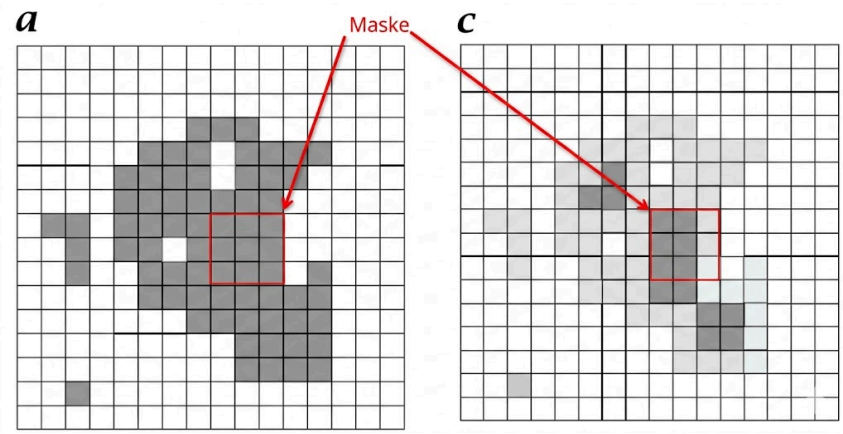


2.6.5.2. Erodieren (erode)

Pixel werden weggenommen

- Strukturen werden kleiner
- Objekte können voneinander getrennt werden
- Alles ausgefüllte bleibt erhalten
- `cv2.erode(img, kernel, iterations=1)`

$$A \ominus S_x = \left\{ x \mid \overline{A} \cap S_x = \emptyset \right\} = \{ x \mid A \supseteq S_x \}$$



2.6.5.3. Funktionen

- **Dilate**: Vergrössert Objekt
- **Erode**: Verkleinert Objekt
- **Closing**: Dilate -> Erode, entfernt Löcher, verbindet Objekte
  - `cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)`
- **Opening**: Erode -> Dilate, separiert Objekte
  - `cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)`

2.6.6. Connected Component Labeling

- Zählt Anzahl Objekte auf einem Binären Bild
- Misst die Grösse eines Objekts in Pixel
- Welche Regionen sind benachbart

Dabei kann mit einer 4-er oder eine 8-er Nachbarschaft gearbeitet werden:

	n	
n	p	n
	n	

4-neighborhood

n	n	n
n	p	n
n	n	n

8-neighborhood

Beispiel:

1	1	1	0	0	0	0	0
1	1	1	0	1	1	0	0
1	1	1	0	1	1	0	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	1	0	0
1	1	1	0	0	0	0	0

Original image (binary)

1	1	1	0	0	0	0	0
1	1	1	0	2	2	0	0
1	1	1	0	2	2	0	0
1	1	1	0	0	0	3	0
1	1	1	0	0	0	3	0
1	1	1	0	0	0	3	0
1	1	1	0	0	4	0	0
1	1	1	0	0	0	0	0

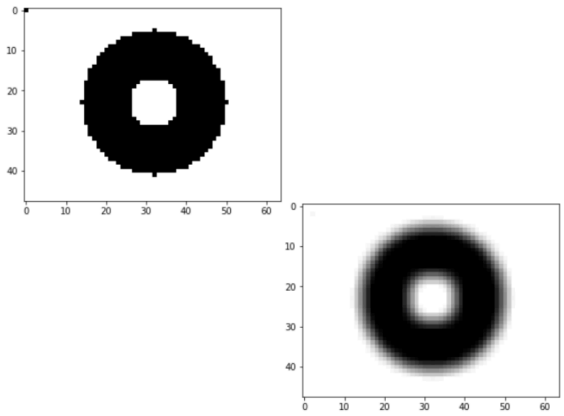
Detected regions with labels

2.7. OpenCV

- Bildverarbeitungslibrary
- Von Intel 1999 entwickelt
- Open-Source, Cross-Platfrom
- Geschrieben in C/C++ -> Schnittstellen zu Python und anderen Srpachen
- Funktionsbibliothek

3. Smoothing, Sharpening, Noise Reduction

3.1. Smoothing



- Lowpass Spatial Filters
- hohe Frequenzen rausfiltern
- tiefe Frequenzen durchlassen
- immer eine Kopie des Bildes erstellen

Anwendungen:
- **Antialiasing**: Reduzierung von Treppeneffekten an Kanten
- **Noise Reduction** (Rauschunterdrückung): reduziert Störungen auf einem Bild

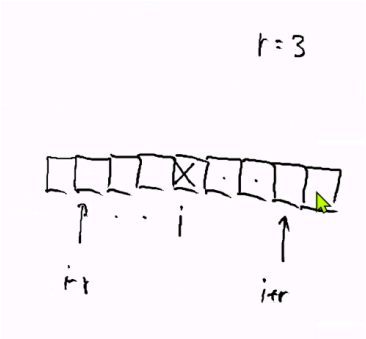
Üblicherweise Nachbarschaftsoperationen
- Durchschnitt der Nachbaren

3.1.1. Box Kernel

- auf einen Pixel wird eine Box gelegt
- alle Pixel in der Nachbarschaft werden gleich gewichtet
- MIttelwert dieser Pixel berechnet

```python
def BlurBox (img, r):
  res = np.ones_like (img)
  m, n = res.shape
  for i in range (r, m - 1 - r):
    for k in range (r, n - 1 - r):
      res [i, k] = np.mean (img [i - r : i + r + 1, k - r : k + r + 1])
  return res
```

je grösser der Radius `r` desto mehr wird geblurt



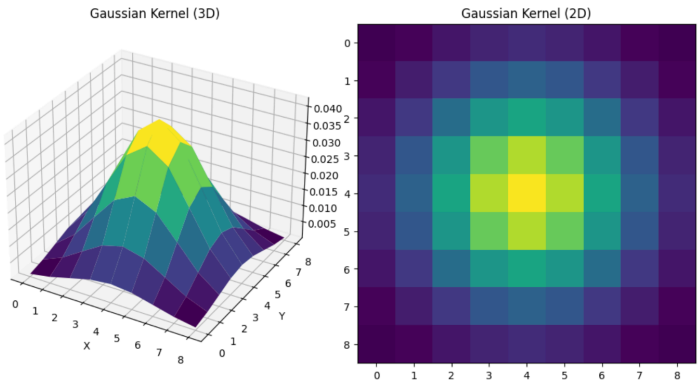
Ein Box Blur erzeugt oft störende horizontale und vertikale Streifen (Artefakte), besonders bei starken Weichzeichnungen.

-> Daher Gauss-Kernel verwenden

- Von innen nach aussen immer weniger gewichten

3.1.2. Gaussian Kernel

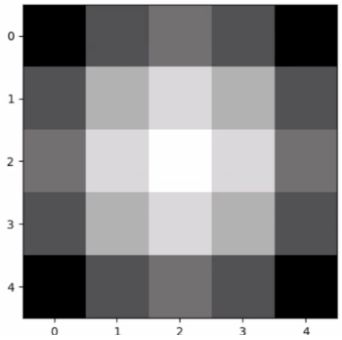
- nahe Nachbaren werden grösser gewichtet
- ferne Nachbaren werden geringer gewichtet
- gelb = starke Gewichtung



3.1.2.1. Code

```python
def Gauss2D (sigma, x, y):
  return 1 / (2 * np.pi * sigma ** 2) * np.exp (- (x ** 2 + y ** 2) / (2 * sigma ** 2))
```

```python
def GaussKernel (sigma, radius):
  x = np.arange (- radius, radius + 1)
  xx, yy = np.meshgrid (x, x)
  zz = Gauss2D (sigma, xx, yy)
  return zz / np.sum (zz)


Kernel = GaussKernel (3, 2)
print (Kernel)
Show (Kernel)
```



Anwenden:

```python
def Apply (kernel, image):
  irows, icols = image.shape
  krows, kcols = kernel.shape
  res = np.ones ((irows, icols))
  drow = krows // 2
  dcol = kcols // 2

  for row in range (0, irows - krows):
    for col in range (0, icols - kcols):
      res [row + drow, col + dcol] = np.sum (image [row : row + krows, col : col + kcols]
* kernel)

  return res


Show (Apply (GaussKernel (3, 2), Image))
```

3.1.2.2. OpenCV

```python
size = 21
blur = cv.GaussianBlur (gray, (size, size), 0)
```

- `size` breite und höhe des Filters (muss meistens eine ungerade Zahl sein), je grösser der Wert ist, je mehr wird das Bild weichgezeichnet

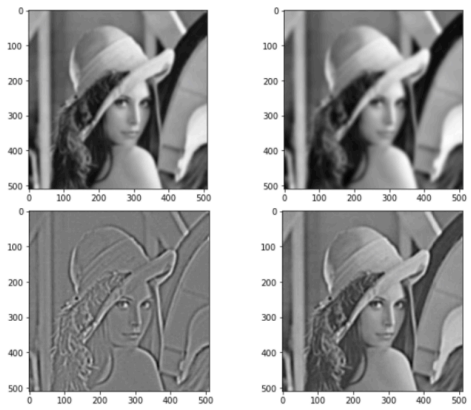
3.2. Sharpening

- Highpass Spatial Filters
- tiefe Frequenzen werden gehemmt
- hohe Frequenzen werden durchgelassen

Methoden:

- Unsharp Masking (with Highboost Filtering)
- Sharpening mit der Heat Equation

3.2.1. Unsharp Masking



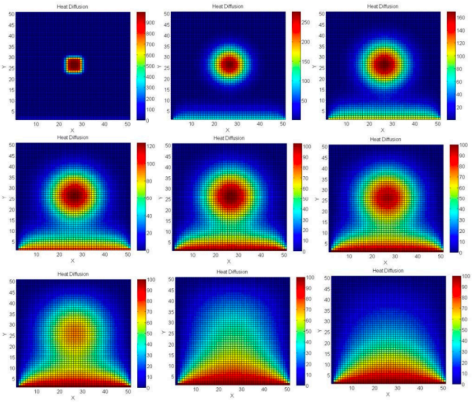
1. Original Image (unscharf)
2. lowpass: Noch weiter blurren
3. Mask: highpass = original - lowpass
   - Edges sind die hohen Frequenzen

```python
mask = image - blur
```

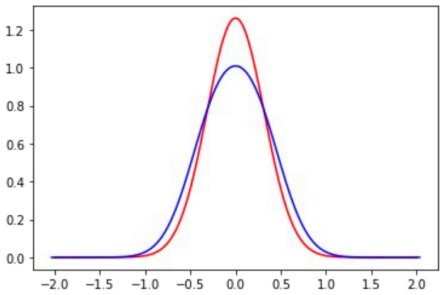
4. Sharpend Image
   - highpass auf Original Bild anwenden (multiplizieren)
   - mit Faktor z.B. 3

```python
sharp = image + 3 * mask
```

3.2.2. Heat Equation



- Beschreibt, wie sich Wärme/Energie über die Zeit (t) im Raum verteilt.
- Wärme fliesst stets so, dass sich Temperaturunterschiede mit der Zeit ausgleichen (diffundieren)



- rot = initiale Zeit, blau = gewisse Zeit später
- **Starke Krümmung (Kanten/Spitzen):** Schnelle Temperaturänderung (Wärme fliesst schnell ab).
- **Schwache Krümmung (Flächen):** Langsame bis keine Änderung.

Anwendung in Computer Vision

- **Diffusion (Vorwärts in der Zeit):** Scharfes Originalbild (u_s) + Wärmeausbreitung = Unscharfes Bild (u_b). Kanten verlaufen.
  - Stellen mit hohem Temperaturunterschied gleichen sich schneller an der Umgebung an -> verrauschen mehr mit der Umgebung / blurren
- **Sharpening (Der Trick / Rückwärts):** Wir simulieren die „Rückabwicklung“ der Zeit.

3.2.2.1. Sharpening in OpenCV

```python
Delta = cv.Laplacian (image, cv.CV_64F, cv.BORDER_DEFAULT)
```

```python
c = 3
sharp = image - c * Delta
```

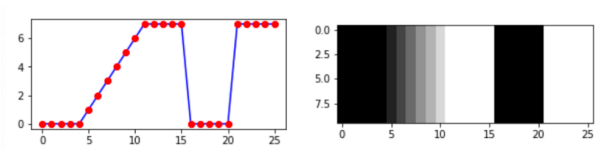
- `-` => Rückabwicklung (Schärfen)
  - „geht in der Zeit zurück“
  - beim Bluren wäre es `+`

3.3. Noise Reduction

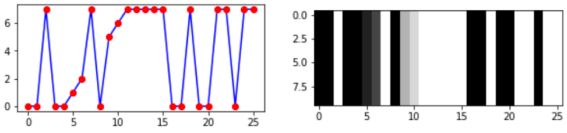
3.3.1. 1D Median Filter

- Inspektion mit Nachbarschaft
  - z.B. 2 Pixel links und 2 Pixel rechts betrachten
- Werden der Intensität nach sortiert
- Nach Sortierung mittleren Wert nehmen

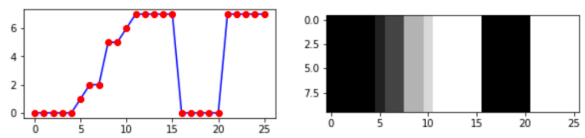
Original Bild:



Noise hinzufügen für Demonstration:



Median:



Beispiel Pixel 3 (Bild mit Noise):

- Original Bild: `[low, low, high (Pixel 3), low, low]`
- links 2 niedrig
- rechts 2 niedrig
- sortiert: `[low, low, low, low, high]`
- Mittelwert nehmen -> `[low]`
- Anwendung Median: Pixel 3 -> `[low]`

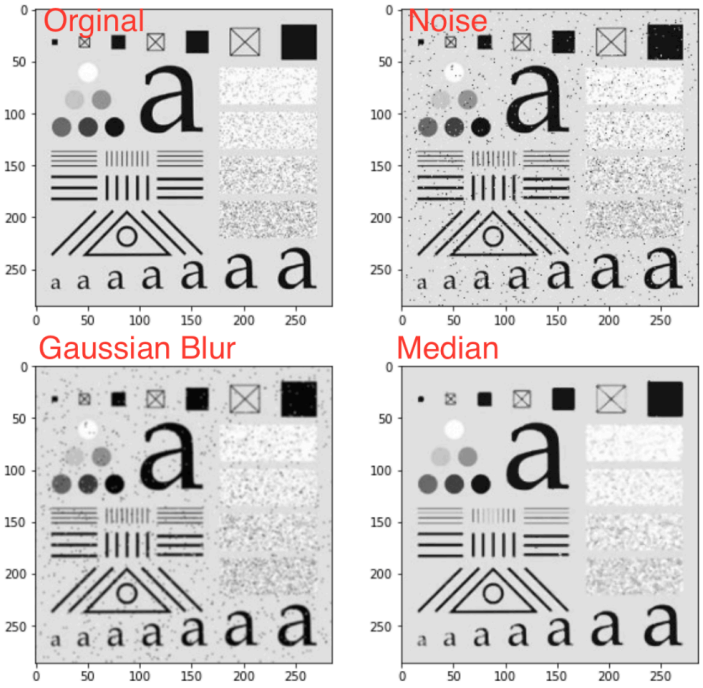
3.3.1.1. Denoise in OpenCV

```python
kernelSize = 3
img_median = cv.medianBlur(img2, kernelSize)
```

3.3.2. Gaussian Blur

Gaussian Blur unterscheidet nicht zwischen Rauschen und wichtigen Details.

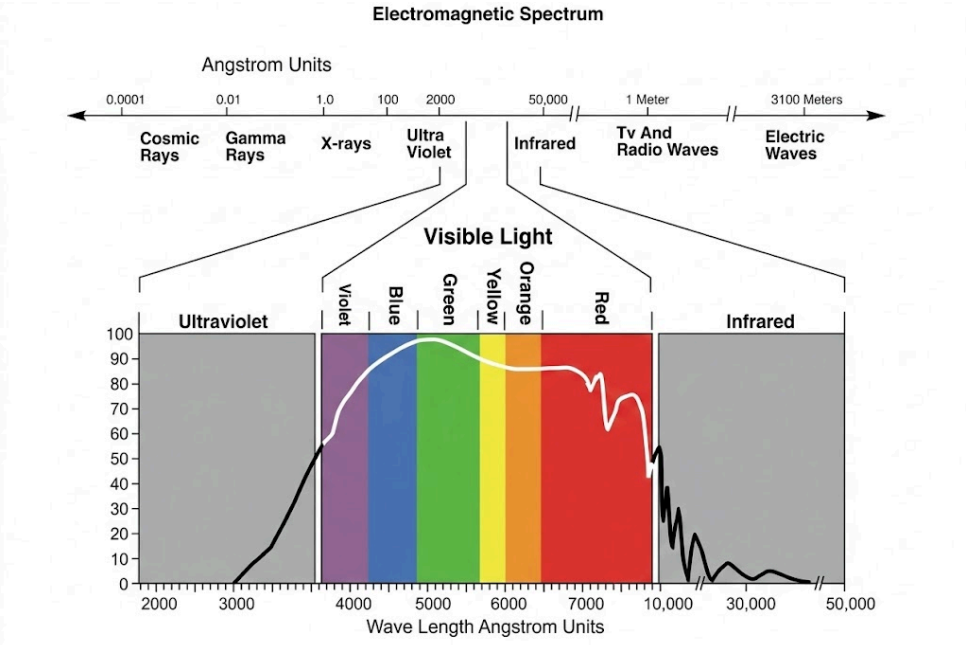
- **Zerstört Kanten:** Er macht das gesamte Bild unscharf, wodurch feine Strukturen (Haare, Texturen, Konturen) verloren gehen.
- **Verschmiert nur:** Das Rauschen wird nicht entfernt, sondern nur zu einem „matschigen“ Schleier verteilt.



4. Color

DEFINITION: Visuelle Wahrnehmung elektromagnetischer Strahlung (sichtbares Licht)

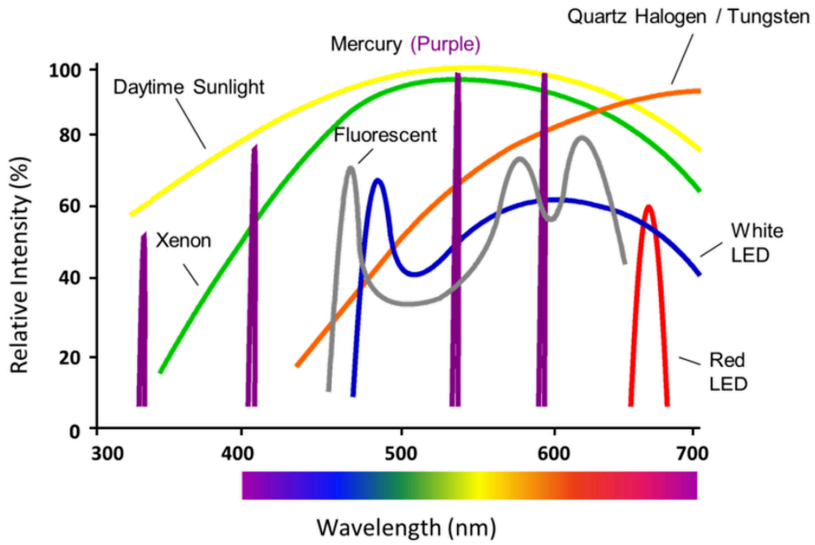
- Beschreibung durch spektrale Verteilung oder Farbmodelle (RGB, CMYK, HSV)



- **Einheiten für Wellenlänge:**
  - **Ångström**: Nicht-SI-Einheit, 1 Å = 10^(-10) m
  - **Nanometer**: SI-Einheit, 1 nm = 10^(-9) m
  - **Umrechnung**: 1 nm = 10 Å

4.1. Lichtarten

- **Spektrale Verteilung:** Zeigt die individuelle Lichtintensität einer Quelle pro Wellenlänge

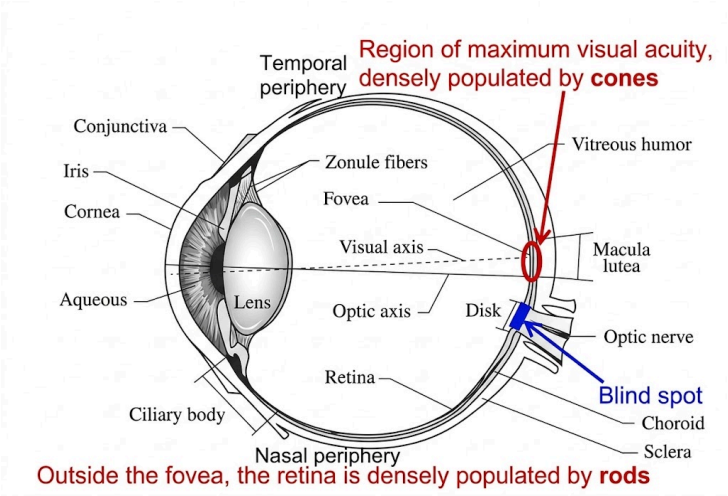


4.2. Auge

- **Iris / Pupille:** Blende, steuert die Menge des einfallenden Lichts
- **Linse:** Fokus, verformt sich zur Scharfstellung des Bildes
- **Retina (Netzhaut):** Bildsensor, erfasst das Licht mit Fotorezeptoren

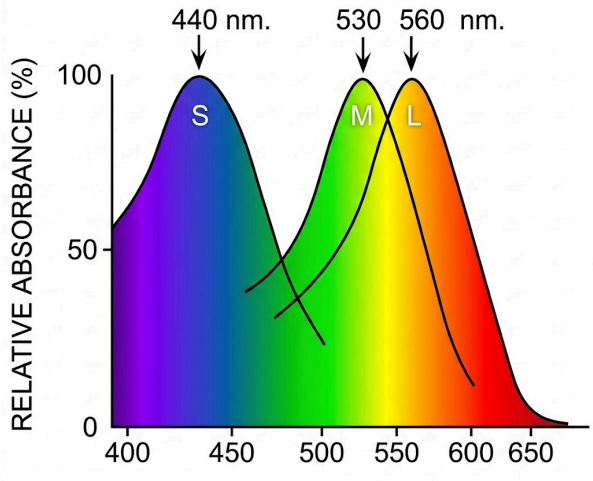
- **Fovea:** Bereich des schärfsten Sehens im Zentrum, enthält ausschliesslich Zapfen
- **Blinder Fleck:** Austrittsstelle des Sehnervs, besitzt absolut keine Fotorezeptoren

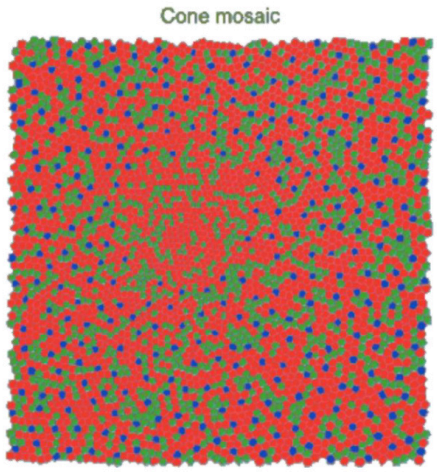
- **Fotorezeptoren**
  - Wandeln Licht in elektrische Impulse um
  - **Stäbchen (Rods):** ca. 75 bis 100 Millionen, zuständig für die Hell-Dunkel-Wahrnehmung
  - **Zapfen (Cones):** ca. 6 bis 7 Millionen, zuständig für die Farbwahrnehmung



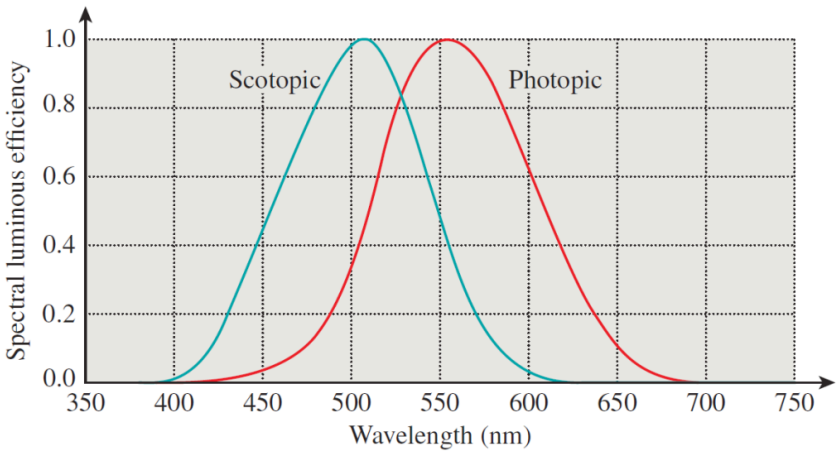
4.2.1. Farbwahrnehmung

- **Farbwahrnehmung (3 Zapfentypen):**
  - **S (Short)**: Blau-Bereich, Peak bei ca. 455 nm
  - **M (Medium)**: Grün-Bereich, Peak bei ca. 534 nm
  - **L (Long)**: Rot-Bereich, Peak bei ca. 563 nm
  - Räumliche Verteilung der drei Zapfentypen (S, M, L) auf der Retina zur detaillierten Farberfassung.

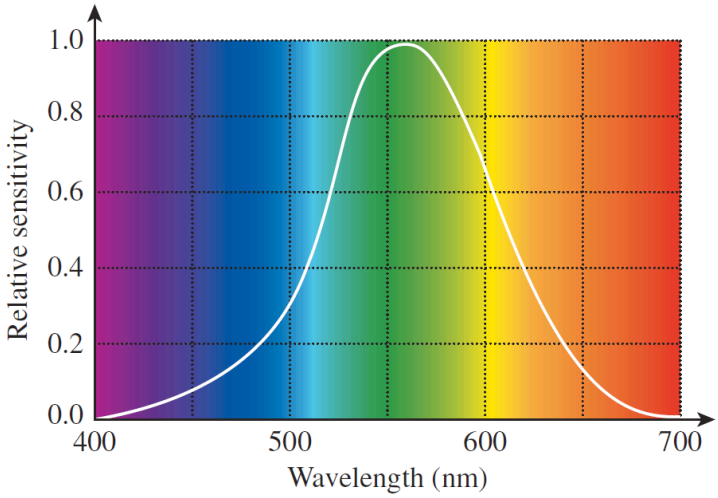




- **Wahrnehmungskette:**
  - Lichtquelle -> Auge (Zapfenreaktion) -> Gehirn (Interpretation)
- **Helligkeitsempfindlichkeit:**
  - Photopisch (Tag): Zapfen-aktiv, Maximum bei ca. 555 nm (Gelb-Grün)
  - Skotopisch (Nacht): Stäbchen-aktiv, Maximum verschiebt sich zu ca. 507 nm



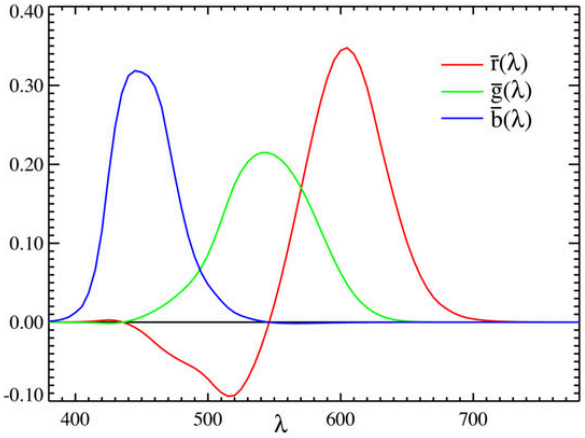
- **Erkenntnis:** Farben haben je nach Wellenlänge eine unterschiedliche wahrgenommene Intensität

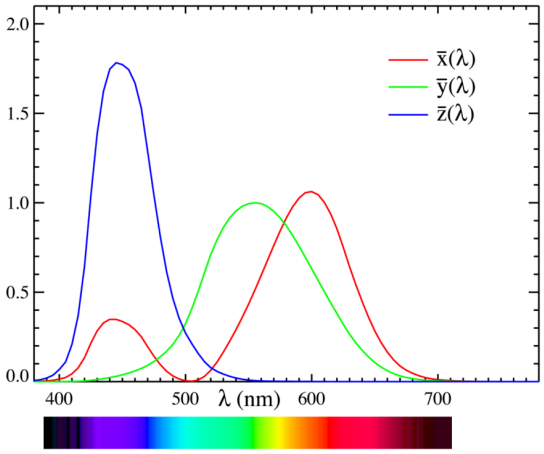


4.3. CIE RGB Farbmischung

- **Setup**: Testperson 2 Blickwinkel, vergleicht Testfarbe mit drei Primärlichtern (p_1, p_2, p_3).
- **Prozess**: Manuelle Intensitätsregelung der Primärfarben bis zur visuellen Farbe.
- **Einschränkung**: Bestimmte Farben sind nicht durch rein additive Mischung erreichbar.
- **Lösung**: Licht wird zur Testfarbe gemischt, zählt mathematisch als „Minus-Wert“ auf der anderen Seite
- **Ergebnis**: Farbanpassungsfunktionen $\overline{r}(\lambda), \overline{g}(\lambda), \overline{b}(\lambda)$.
- **Merkmal**: $\overline{r}(\lambda)$ zeigt negative Werte im Bereich um 500 nm.
- **XYZ-Transformation**: Lineare Transformation für rein positive Werte

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 2.768892 & 1.751748 & 1.130160 \\ 1 & 4.590700 & 0.060100 \\ 0 & 0.056508 & 5.594292 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$





4.3.1. Color Space

- Ermittlung durch Integration der spektralen Verteilung $P(\lambda)$:

$$X = k \int_{\Lambda} P(\lambda)\overline{x}(\lambda)d\lambda$$

$$Y = k \int_{\Lambda} P(\lambda)\overline{y}(\lambda)d\lambda$$

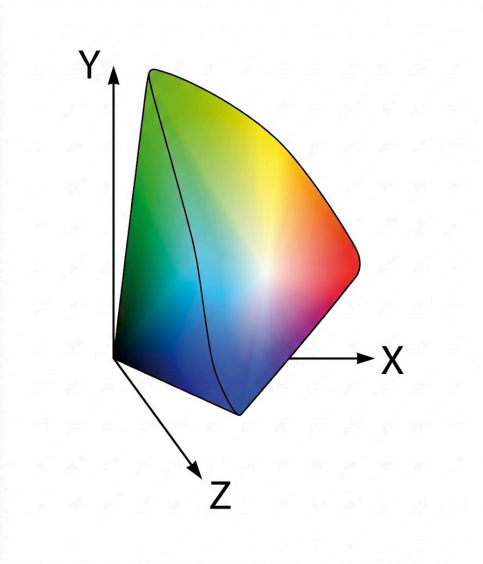
$$Z = k \int_{\Lambda} P(\lambda)\overline{z}(\lambda)d\lambda$$

$$\Lambda = [380\ \text{nm}, 780\ \text{nm}]$$

- Ermöglicht das Rechnen mit Werten aus dem RGB-Experiment

4.3.1.1. 3D-Koordinatensystem

- Enthält alle theoretischen Farben
- **Zentrum**: Sonnenlicht (Weiss)
- **Aussenrand**:
  - Höchste Sättigung
  - Untere Kante (Purpurlinie) kommt nicht als reine Wellenlänge im Sonnenlicht vor

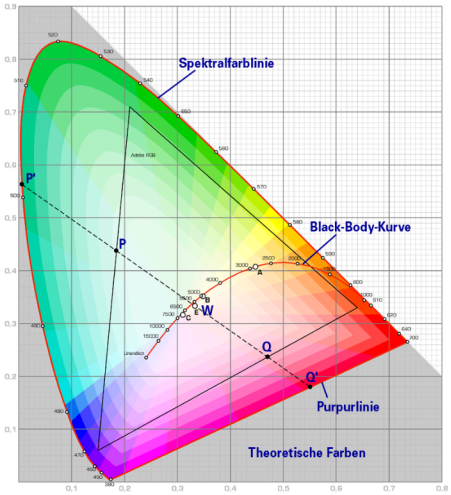


4.3.1.2. Chromatizitätsdiagramm (2D)

- Schnittebene durch $(1,0,0)$, $(0,1,0)$, $(0,0,1)$ im XYZ-Raum
- Nutzt keine vollen XYZ-Koordinaten (Helligkeit fehlt, Projektion auf 2D

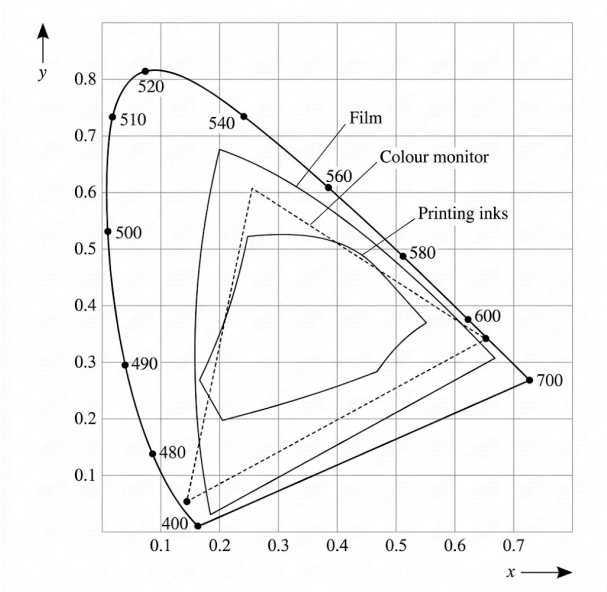
Merkmale

- **Spektralfarben:** Auf der gebogenen Randkurve
- **Purpurlinie:** Verbindet Violett (380 nm) und Rot (780 nm)
- **Farbmischung:**
  1. Linie zwischen den zwei Farbpunkten ziehen
  2. Position auf der Linie bestimmt das Mischverhältnis (0 bis 1)
  3. Exakte Mitte (0.5) entspricht der exakt gleichen Menge beider Farben
- **Dominante Wellenlänge:**
  1. Strahl vom Weisspunkt durch die Zielfarbe ziehen
  2. Schnittpunkt am oberen gebogenen Spektralrand ablesen
- **Komplementärfarbe:**
  1. Punkt der Ausgangsfarbe direkt am Weisspunkt spiegeln
  2. Gespiegelter Punkt entspricht der Komplementärfarbe



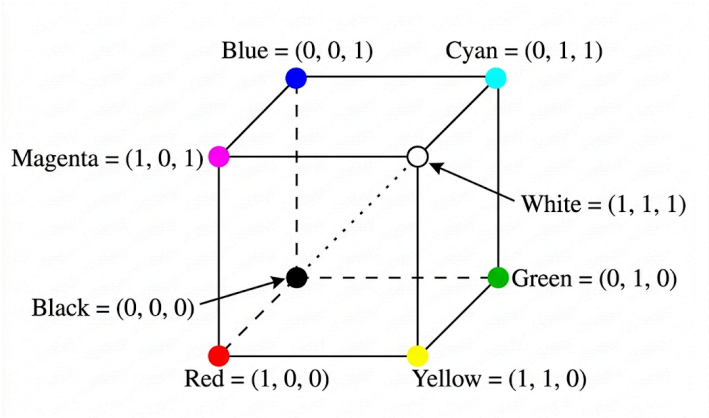
4.4. Farbmodelle

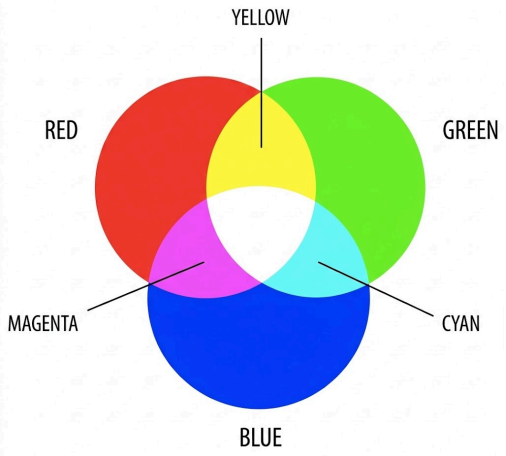
- Farbräume sind Geräteabhängig



4.4.1. RGB

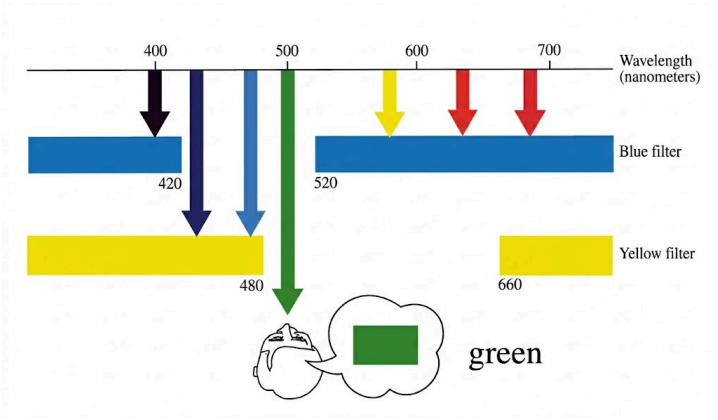
- Red, Green, Blue im Bereich $[0, 1]^3$
- Standard für Computer-Monitore
- **sRGB**: Genormter RGB-Standard, für konsistente Farben auf Monitoren, Druckern und im Web
- Additve Farbmodus
  - Benachbarte Farbpunkte verschmelzen aus der Distanz zur Mischfarbe
- Können als Wüfel dargestellt werden
  - Gesättigte Farben in der Mitte (mühsamer für Berechnungen)

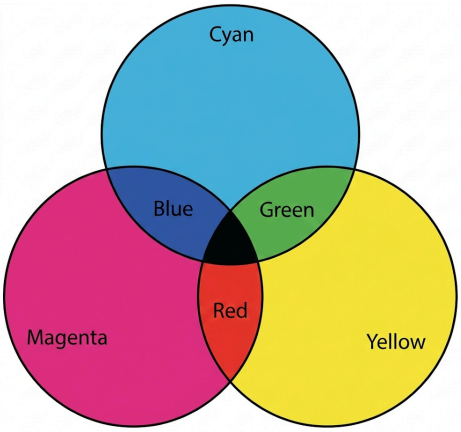




4.4.2. CMY(K)

- Cyan, Magenta, Yellow, Key (black)
- Key kann auch durch mischen von C + M + Y erreicht werden (teuer)
- Für Farbdrucker
- Subtraktiver Farbmodus
  - Farbfilter absorbieren bestimmte Lichtwellenlängen und lassen den Rest als neue Mischfarbe durch.





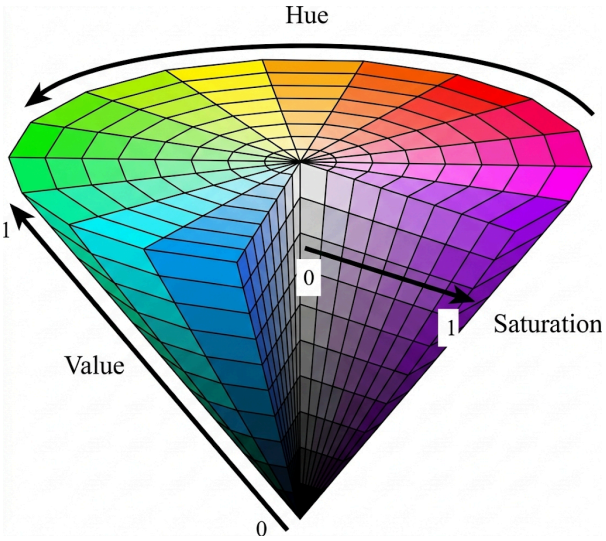
$$\text{RGB zu CMY: } \begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

4.4.3. YIQ

- **Y**: Helligkeit (luminance)
- **I**: Differenz zwischen Cyan und Orange
- **Q**: Differenz zwischen Magenta und Grün
- Alte Fernseher

4.4.4. HSV

- **H (Hue)**
  - Farbton
  - $0° - 360°$
- **S (Saturation)**:
  - Sättigung/Weiss-Anteil
  - $0 - 1$
  - $0 = \text{Weiss}, 1 = \text{kein Weiss/reine Farbe}$
- **V (Value)**:
  - Intensität/Hellwert
  - $0 - 1$
  - $0 = \text{Schwarz}, 1 = \text{maximale Intensität}$
- Anwenderorientiertes Farbmodell



Farbe	H	S	V	R	G	B
Schwarz	–	–	0 %	0 %	0 %	0 %
Rot	0°	100 %	100 %	100 %	0 %	0 %
Gelb	60°	100 %	100 %	100 %	100 %	0 %
Braun	24,3°	75 %	36,1 %	36 %	20 %	9 %
Weiß	–	0 %	100 %	100 %	100 %	100 %
Grün	120°	100 %	100 %	0 %	100 %	0 %
Dunkelgrün	120°	100 %	50 %	0 %	50 %	0 %
Cyan	180°	100 %	100 %	0 %	100 %	100 %
Blau	240°	100 %	100 %	0 %	0 %	100 %
Magenta	300°	100 %	100 %	100 %	0 %	100 %
Orange	30°	100 %	100 %	100 %	50 %	0 %
Violett	270°	100 %	100 %	50 %	0 %	100 %

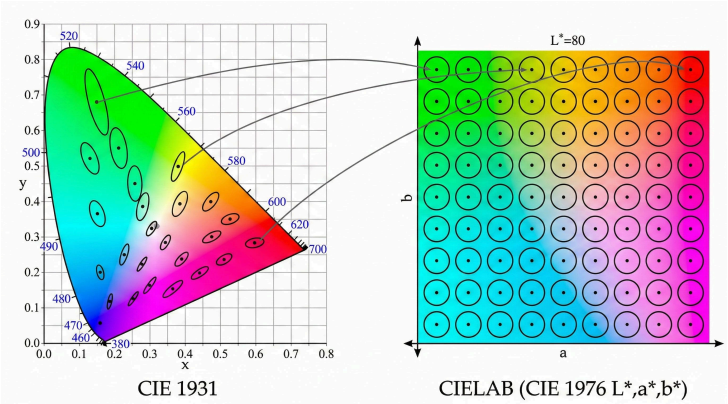
4.4.5. Composite Video YUV

- **Y**: Helligkeit (luminance)
- **UV**: Chrominanz (Farbanteile)
  - Farbton (Hue) + Sättigung
- Oft mit reduzierter Auflösung übertragen für Videokompression
- **YPbPr**: Skalierte Version von YUV

$$\text{RGB zu YUV:}$$
$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$
$$U = \frac{0.436 \cdot (B - Y)}{1 - 0.114}$$
$$V = \frac{0.615 \cdot (R - Y)}{1 - 0.299}$$

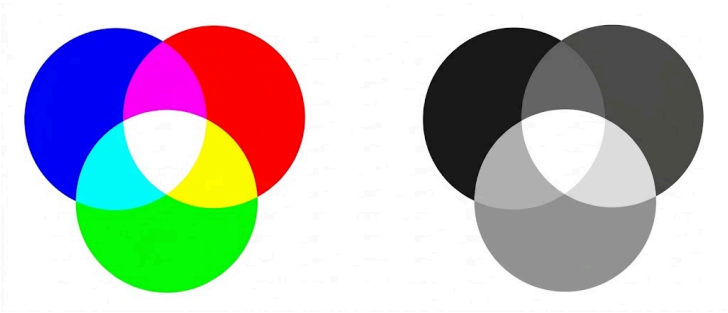
4.4.6. CIE Lab

- Welche Farben haben gleiche Helligkeit, werden gleich wahrgenommen?
- **MacAdam-Experiment**: Identifikation von Ellipsen, in denen Farbunterschiede für das Auge nicht unterscheidbar sind.
- **Nutzen**: Absolutes, geräteunabhängiges System, bei dem die mathematische Distanz zwischen zwei Werten exakt der visuell wahrgenommenen Farbdifferenz entspricht
- **Transformation**: Geometrische Verzerrung des Farbraums, um Ellipsen in Kreise für eine gleichmässige Wahrnehmung zu verwandeln
- **Attribute**:
  - **L***: Luminanz (Helligkeit)
  - **a***: Rot-Grün-Achse
  - **b***: Gelb-Blau-Achse



CIE 1931 CIELAB (CIE 1976 L*,a*,b*)

4.5. Umwandlung in Grauwerte



$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

5. Image Compression

5.1. Anwendung

5.1.1. Vorteile

- Reduziert Datenvolumen
- Weniger Speicher
- Weniger Übertragungszeit

5.1.2. Anwendungsbereiche

- Bilder
  - Digitale Fotografie (JPEG)
  - Vidoes (MPEG)
- Audio (MP3)
- Daten
  - Speicher
  - Kommunikation
    - Übertragung
    - Empfang

5.2. Software

5.2.1. Verlustfreie Kompression (Lossless)

Diese Verfahren erlauben eine exakte Rekonstruktion der Originaldaten.

- **RLE (Run-Length Encoding)**
  - *Anwendung:* BMP (Windows Bitmap), FAX
- **LZ77 (Lempel-Ziv 77)**
  - *Anwendung:* GZIP, ZIP
- **LZ78 (Lempel-Ziv 78)**
  - *Anwendung:* GIF
  - GIF heute **LZW** (eine Weiterentwicklung von LZ78)
- **LZW (Lempel-Ziv-Welch)**
  - *Anwendung:* Unix-Befehl `compress`
- **Huffman-Kodierung**
  - *Anwendung:* Unix-Befehl `pack`

5.2.2. Verlustbehaftete Kompression (Lossy)

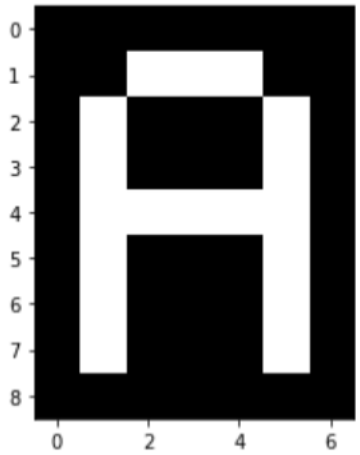
Diese Verfahren komprimieren stärker, indem sie kaum wahrnehmbare Details weglassen.

- **DCT (Diskrete Kosinustransformation)**
  - *Anwendung:* **JPEG** (Standbilder)
  - *Anwendung:* **MPEG** (Video/Audio)

5.3. Run Length Encoding

- **Verlustfrei**: Bezieht sich auf das Binärbild
- Muss ein Binärbild sein (schwarz / weiss)
  - $128 \leq$ pixel $\rightarrow$ Weiss
  - $128 >$ pixel $\rightarrow$ Schwarz
- **Funktionsweise**: Speichert die Anzahl aufeinanderfolgender, identischer Pixel (Lauflänge) anstelle einzelner Binärwerte
  - Zeilen- oder spaltenweiser Durchlauf (muss für Kodierung und Dekodierung identisch sein)
- **Effizienz**: Je länger die Pixelsequenzen, desto besser die Kompression (Worst-Case: Schachbrettmuster)
- **Dekodierung**: Die Originalgrösse muss bekannt sein, um Zeilenumbrüche richtig zu setzen
- **Invertieren**: Ein vertauschtes Startbit (0 statt 1) invertiert automatisch das Bild

5.3.1. Beispiel




```
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 0, 0,
0, 1, 0, 0, 0, 1, 1, 0,
0, 1, 0, 0, 0, 0, 1, 0,
0, 1, 1, 1, 1, 1, 1, 0,
0, 1, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0
```

- $9 \cdot 7 = 63$ pixels

Sequenz von identischen Pixel (Zeilenweise)

$9, 3, 3, 1, 3, 1, 2, 1, 3, 1, 2, 5, 2, 1, 3, 1, 2, 1, 3, 1, 2, 1, 3, 1, 8$

Speicher: 25 Längen (statt 63 Pixel)

5.4. Dictionary (Wörterbuch)

- Buchstaben im Alphabet werden als Diagramm zusammengefasst
- Es werden so viele Diagramme wie möglich gebildet
- Dictionary wird für den Empfänger mitgeschickt (Absprache nötig)
- Grosser Aufwand dies zu programmieren

5.4.1. Beispiel

string: `abracadabra` $11 \cdot 8$ Bit $= 88$ Bit

Dictionary

- Alphabet
  - 0 (000): a
  - 1 (001): b
  - 2 (010): c
  - 3 (011): d
  - 4 (100): r
- Diagram (freie Plätze nutzen für Kombinationen)
  - 5 (101): ab
  - 6 (110): ac
  - 7 (111): ad

`ab r ac ad ab r a` = `101 100 110 111 101 100 000`

$7 \cdot 3$ Bit $= 21$ Bit

5.5. Lempel-Ziv Compression

5.5.1. LZ77: Compress (Encode)

- **Einsatz**: Basis für Formate wie ZIP oder PNG.
- **Adaptiv**: Passt sich an die jeweiligen Sprachen des Textes an
- **Vorteil**: Wörterbuch / Dictionary muss nicht mitgeschickt werden
- **Nachteil**: search buffer kann zu klein sein
- **Funktionsweise**: Kommt eine Zeichenfolge erneut vor, wird sie durch einen Verweis (Index) auf die vorherige Stelle ersetzt.
- **Komponenten**:
  - Search Buffer: Bereits verarbeitete Zeichen.
  - Look-ahead Buffer: Als nächstes zu kodierende Zeichen.
  - Vorgang: Zeichen wandern vom Look-ahead- in den Search-Buffer.
- **Code**:
  - `m` : index (0 = no string)
  - `n` : Länge
  - `c` : Buchstabe nach der Kopie

5.5.1.1. Beispiel

`abracabrabra`

1. Kein string, `a`
2. Kein string, `b`
3. Kein string, `r`
4. Beim index 3, 1 Zeichen, `a`, dann `c`
5. Beim index 5, 3 Zeichen, `abr`, dann `a`
6. Beim index 3, 3 Zeichen `bra`

search buffer						look-ahead buffer				code
6	5	4	3	2	1					
						a	b	r	a	(0, 0, a)
					a	b	r	a	c	(0, 0, b)
				a	b	r	a	c	a	(0, 0, r)
			a	b	r	a	c	a	b	(3, 1, c)
	a	b	r	a	c	a	b	r	a	(5, 3, a)
a	c	a	b	r	a	b	r	a		(3, 3, -)

5.5.2. LZ77: Decompress (Decode)

- Zeichen werden anhand des Codes aus dem buffer ausgelesen

5.5.2.1. Beispiel

`abracabrabra`

1. Kein string, `a`
2. Kein string, `b`
3. Kein string, `r`
4. Beim index 3, 1 Zeichen, `a`, dann `c`
5. Beim index 5, 3 Zeichen, `abr`, dann `a`
6. Beim index 3, 3 Zeichen `bra`

input		buffer						output			
#	code	6	5	4	3	2	1				
1	(0, 0, a)							a			
2	(0, 0, b)						a	b			
3	(0, 0, r)					a	b	r			
4	(3, 1, c)				a	b	r	a	c		
5	(5, 3, a)		a	b	r	a	c	a	b	r	a
6	(3, 3, -)	a	c	a	b	r	a	b	r	a	

5.5.3. LZ78: Compress (Encode)

- **Vorteil**: kann nicht zu klein werden (kein search buffer mehr)
- **Nachteil**: Dictionary muss übergeben werden
- Ähnlich wie LZ77
- **code** `(m,c)`
  - `m`: index (0 = no string)
  - `c`: nächster Buchstabe

5.5.3.1. Beispiel

`abbcbcababcaabcaab` -> `a b bc bca ba bcaa bcaab`

1. `a` nicht im Wörterbuch, `a` hinzufügen
2. `b` nicht im Wörterbuch, `b` hinzufügen
3. `b` im Wörterbuch (Index 2), `bc` nicht, `bc` hinzufügen
4. `bc` im Wörterbuch (Index 3), `bca` nicht, `bca` hinzufügen
5. `b` im Wörterbuch (Index 2), `ba` nicht, `ba` hinzufügen
6. `bca` im Wörterbuch (Index 4), `bcaa` nicht, `bcaa` hinzufügen
7. `bcaa` im Wörterbuch (Index 6), `bcaab` nicht, `bcaab` hinzufügen

dictionary		code	
index	string	#	output
1	a	1	(0, a)
2	b	2	(0, b)
3	bc	3	(2, c)
4	bca	4	(3, a)
5	ba	5	(2, a)
6	bcaa	6	(4, a)
7	bcaab	7	(6, b)

5.5.4. LZ78 Decompress (Decode)

- Code wird übergeben
- Wird aus dem Wörterbuch ausgelesen

5.5.4.1. Beispiel

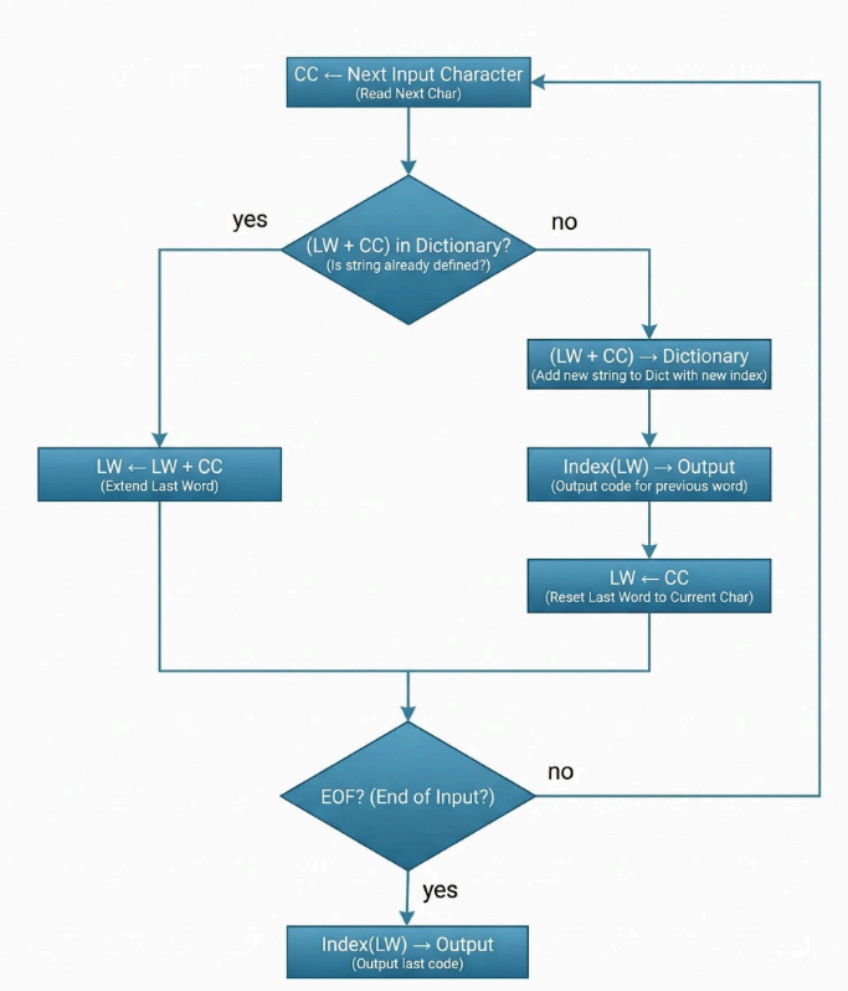
`a b bc bca ba bcaa bcaab` -> `abbcbcababcaabcaab`

1. Kein string, `a`
2. Kein string, `b`
3. `b` beim Index 2, dann `c`
4. `bc` beim Index 3, dann `a`
5. `b` beim Index 2, dann `a`
6. `bca` beim Index 4, dann `a`
7. `bcaa` beim Index 6, dann `b`

input		dictionary		output
#	code	index	string	
1	(0, a)	1	a	a
2	(0, b)	2	b	b
3	(2, c)	3	bc	bc
4	(3, a)	4	bca	bca
5	(2, a)	5	ba	ba
6	(4, a)	6	bcaa	bcaa
7	(6, b)	7	bcaab	bcaab

5.5.5. LZW: Compress (Encode)

- Initial Wörterbuch vorhanden
  - Alle ASCI Zeichen im Dictionary
- **Vorteil**: Wörterbuch muss nicht mitgeschickt werden
- **LW**: longest word
  - Letzte Wort bzw. die Zeichenkette, die der Algorithmus sich gerade ansieht
- **CC**: current character
  - Nächste einzelne Zeichen, das aus dem Text gelesen wird
- **Dict**: Dictionary
  - Code Tabelle
  - Beginnt bei 256 (nach ASCI)
  - Zählt einfach hoch
- **Out**: Output
  - Tatsächlich komprimierte Datei
  - In ASCI Code (0 bis 255)



5.5.5.1. Beispiel

`rabarbarbarbara`

LW	CC	LW+CC	Dict	Out
- / r	r	r		
r / a	a	ra	256: ra	114
a / b	b	ab	257: ab	97
b / a	a	ba	258: ba	98
a / r	r	ar	259: ar	97
r / b	b	rb	260: rb	114
b / ba	a	ba		
ba / r	r	bar	261: bar	258
r / rb	b	rb		
rb / a	a	rba	262: rba	260
a / ar	r	ar		
ar / b	b	arb	263: arb	259
b / ba	a	ba		
ba / bar	r	bar		
bar / a	a	bara	264: bara	261
a	EOF	-	-	97

- `r` : ist bereits im Dictionary (da Initial ASCI Code vorhanden)
- Zu `r` wird `a` (nächster Input) hinzugefügt -> `ra`
  - Noch nicht im Dictionary vorhanden
  - Wird zu Dictionary hinzugefügt (erster Code nach 255)
  - Output vorherigen Zeichen (`r`) wird geschrieben

5.5.6. LZW: Decompress

- Wörterbuch wird nicht benötigt
  - Wird Stück für Stück wie beim Compress aufgebaut

```
code <- NextCode
w <- Dict [code]
w -> Out

while (code <- NextCode) {
    if (code ∈ Dict) {
        tmp <- w
        w <- Dict (code)
        tmp + w[0] -> Dict
    } else {
        w + w[0] -> Dict
        w <- Dict (code)
    }
    w -> Out
}
```

5.5.6.1. Beispiel

code	tmp	w	Dict	Out
114		r		r
97	r	a	256: ra	a
98	a	b	257: ab	b
97	b	a	258: ba	a
114	a	r	259: ar	r
258	r	ba	260: rb	ba
260	ba	rb	261: bar	rb
259	rb	ar	262: rba	ar
261	ar	bar	263: arb	bar
97	bar	a	264: bara	a

5.5.7. Huffman Coding

- Von einem Wald zu einem Baum
- **Vorteil**: Häufige Zeichen kommen ganz oben im Baum (schneller)
- **Nachteil**: Der ganze Code muss komplett übertragen werden

Vorgehen:

1. **Häufigkeit ermitteln**: für jedes Zeichen wird gezählt, wie oft es im Text vorkommt
2. **Wald erstellen (Start-Knoten)**: Für jedes **Zeichen** wird ein Start-Knoten erstellt
3. **Baum aufbauen**:
   - Die zwei Knoten mit den geringsten Häufigkeiten suchen
   - Diese beiden zu einem neuen Baum zusammenfassen
   - Root-Knoten stellt die Häufigkeit der beiden Kind-Knoten dar (Summe)
   - Solange wiederholen, bis ein Baum vorhanden ist
4. **Kanten beschriften**:
   - Pfad links: o
   - Pfad rechts: 1
5. **Code ablesen**: Pfad folgen von Wurzel bis zum gewünschten Knoten

5.5.7.1. Beispiel

`aeebafabacecffcdefeff` -> 21 Byte = 168 Bit

Häufigkeiten:

- a: 4
- b: 2
- c: 3
- d: 1
- e: 5
- f: 6

Schritte:

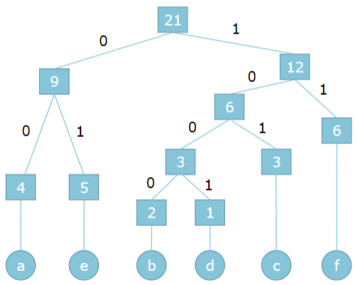
- Start: d(1), b(2), c(3), a(4), e(5), f(6)
- d+b=3 ⟶ c(3), db(3), a(4), e(5), f(6)
- c+db=6 ⟶ a(4), e(5), f(6), cdb(6)
- a+e=9 ⟶ f(6), cdb(6), ae(9)
- f+cdb=12 ⟶ ae(9), fcdb(12)
- ae+fcdb=21 ⟶ Wurzel(21)

Codes:

- a: 00
- b: 1000
- c: 101
- d: 1001
- e: 01
- f: 11

Compressed: 00 01 01 1000 00 11 00 1000 00 101 01 101 11 11 101 1001 01 11 01 11 11 (51 Bit)

Decompressed: a e e b a f a b a c e c f f c d e f e f f f



5.6. Discrete Cosine Transform

Verlustbehaftet

5.6.1. 1D

- **Klang vs. Rauschen:**
  - Ein reiner Klang lässt sich als Mischung (Synthese) von verschiedenen Sinus- und Kosinuswellen darstellen.
- **Fourier-Analyse:**
  - Der Prozess, um in einem Signal (z.B. Audio) zu analysieren, aus welchen exakten Frequenzen (Ober- und Untertönen) es besteht.
- **Synthese (Klang rekonstruieren):**

$$f(x) \approx \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos(kx) + b_k \sin(kx)$$

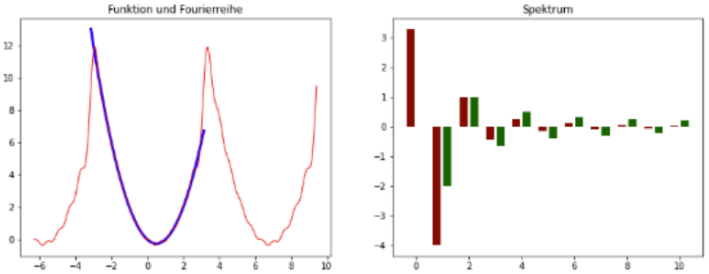
  - **Cosine amplitudes**:

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx) \ \mathrm{dx}$$

  - **Sine amplitudes**:

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(kx) \ \mathrm{dx}$$

Vorgehen: Signal nehmen → analysieren → als Spektrum abspeichern. Aus dem Spektrum kann man durch Fourier-Synthese das Signal wieder rekonstruieren.

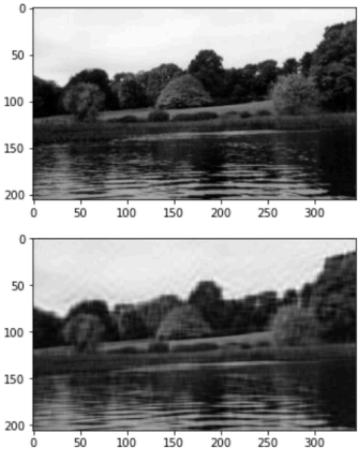


5.6.1.1. Wieso nicht die normale Fourier-Transformation?

- Vermeidung von komplexen Zahlen
- Sinus Funktionen werden weggelassen
  - Es wird nur noch cosine Wellen verwendet
- Statt $[-pi, pi]$ nur noch $[0, pi]$

5.6.2. 2D (Bilder und Kompression)

- Bei Bilder DCT zuerst zeilenweise, und dann spaltenweise
- DCT wandelt das Bild vom Ortsraum (Pixelkoordinaten x,y) in den Frequenzraum um
  - Frequenzbild zeigt an, wie stark Frequenzen im Bild vertreten sind
- Hohe Frequenzen werden weggelassen oder auf null gesetzt um Speicher zu sparen
- **Kanten-Problem:** Hohe Frequenzen sind oft scharfe Kanten
  - Darstellungsfehler entstehen


0
50
100
150
200
0
50
100
150
200
250
300
0
50
100
150
200
0
50
100
150
200
250
300

6. Filtering Convolution Correlation

6.1. Filter

DEFINITION: Bilder verändern oder bestimmte Informationen daraus zu gewinnen.

- Extraktion von Details
  - Kanten
  - Ecken
- Noise Reduction
- Modifikation vom Bild

6.2. Convolution 1D

$$\text{conv}(x,y)_k = \sum_{i=-\infty}^{\infty} x_i y_{k-i} = \sum_{i=-\infty}^{\infty} x_{k-i} y_i$$

6.2.1. Rechnen

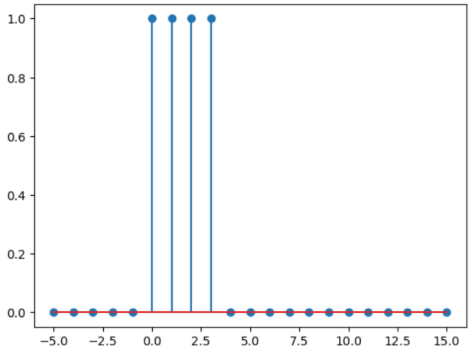
i	-3	-2	-1	0	1	2	3	4	5	6	
x_i	0	0	0	1	1	1	1	0	0	0	first sequence
y_i	0	0	0	1	2	1	0	0	0	0	second seqence
y_{0-i}	0	1	2	1	0	0	0	0	0	0	reflected ...
y_{1-i}	0	0	1	2	1	0	0	0	0	0	right shifted ...
y_{2-i}	0	0	0	1	2	1	0	0	0	0	...
y_{3-i}	0	0	0	0	1	2	1	0	0	0	...
y_{4-i}	0	0	0	0	0	1	2	1	0	0	...
y_{5-i}	0	0	0	0	0	0	1	2	1	0	....
y_{6-i}	0	0	0	0	0	0	0	1	2	1	...

- zuerst Spiegelung einer der Reihen machen
- danach vorlaufend nach rechts verschieben
- so lange machen, bis die erste Folge mit der zweiten Folge nicht mehr überschneiden
- spaltenweise multiplizieren und dann addieren

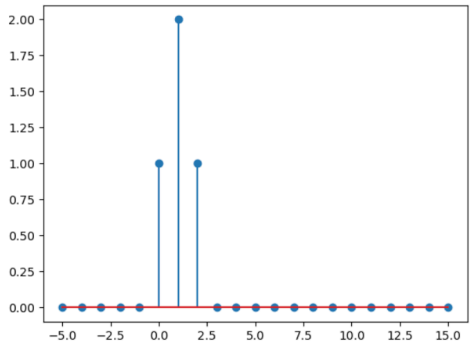
$$\begin{aligned}
z_k &= \text{conv}(x,y)_k \\
z_k &= 0 \quad \text{for} \quad k < 0. \\
z_0 &= 1 \cdot 1 = 1 \\
z_1 &= 1 \cdot 2 + 1 \cdot 1 = 3 \\
z_2 &= 1 \cdot 1 + 1 \cdot 2 + 1 \cdot 1 = 4 \\
z_3 &= 1 \cdot 1 + 1 \cdot 2 + 1 \cdot 1 = 4 \\
z_4 &= 1 \cdot 1 + 1 \cdot 2 = 3 \\
z_5 &= 1 \cdot 1 = 1 \\
z_k &= 0 \quad \text{for} \quad k > 5.
\end{aligned}$$

6.2.2. Python bzw. Graphisches Beispiel

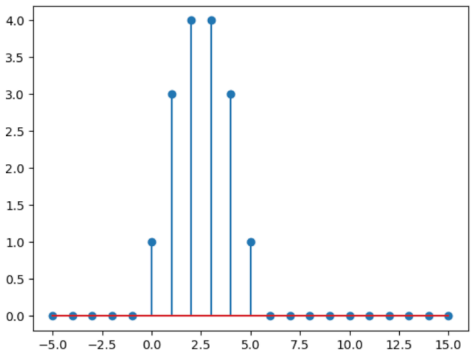
- erster Operand eine Box



- zweiter Operand ein Pixel



- Als Resultat kommt eine geglättete Reihe raus



6.2.3. Rechenregeln

Assoziativ

$$\text{conv}(\text{conv}(x, y), z) = \text{conv}(x, \text{conv}(y, z))$$

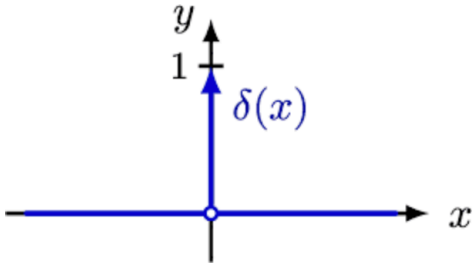
Kumulativ

$$\text{conv}(x, y) = \text{conv}(y, x)$$

Distributiv

$$\text{conv}(x + y, z) = \text{conv}(x, z) + \text{conv}(y, z)$$

6.3. Delta Sequence



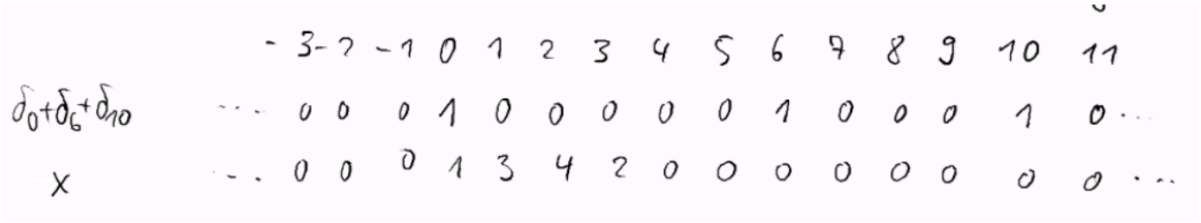
$$\delta_k = \begin{cases} 1 \text{ if } \ k = 0 \\ 0 \text{ else} \end{cases}$$

- überall 0 bis auf einer Stelle, dort ist sie 1
- δ^0 = an der Stelle 0 ist 1, überall sonst 0
- δ^6 = an der Stelle 6 ist 1, überall sonst 0

6.3.1. Beispiel

$x = (1, 3, 4, 2)$

$\text{conv}(x, \delta^0 + \delta^6 + \delta^{10})$



6.4. 2D Convolution

$$I'(u,v) = \sum_{(i,j)\in R} I(u-i, v-j) \cdot \tilde{H}(i,j)$$

Oder umgeschrieben mit der rotierten Matrix H

$$I'(u,v) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} I(u+i, v+j) \cdot H(i,j)$$

- I = Originalbild
- $\tilde{H}$ = Filtermatrix (Filter Kernel)
- H = Filtermatrix, um 180° rotiert
- R = Wertebereich (Range) der Indizes von $\tilde{H}$
- I' = Gefiltertes Bild (Faltung: $\text{conv}(I, \tilde{H})$)

—

- Im Unterschied zur einfachen Korrelation wird die Filtermatrix w bei der Faltung (Convolution) zuerst um 180° rotiert.
- Danach wird der Filter über das Bild geschoben, pixelweise multipliziert und aufsummiert.

6.4.1. Low-pass Filter

Ein typisches Beispiel für einen linearen 2D-Filter ist der Low-Pass Filter (Mittelwertfilter)

$$\tilde{H} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Die Berechnung der Faltung vereinfacht sich bei diesem $3 * 3$ Filter zu

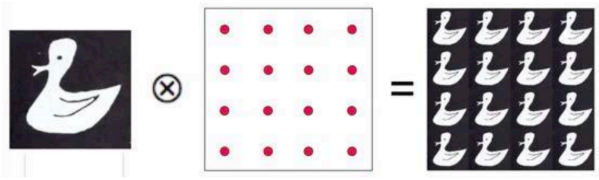
$$I'(u,v) = \frac{1}{9}\sum_{i=-1}^{1}\sum_{j=-1}^{1} I(u+i, v+j) \cdot \tilde{H}(-i,-i)$$

$$= \frac{1}{9}\sum_{i=-1}^{1}\sum_{j=-1}^{1} I(u+i, v+j)$$

- **Effekt:** Das Originalbild wird durch die Durchschnittsbildung der benachbarten Pixel geglättet
- Als Resultat erhält man ein *weichgezeichnetes* (smoothed) Bild

6.4.2. Placing Objects

Delta Funktion nehmen



6.5. Cross-Correlation

Misst **zwei verschiedene** Signale miteinander

- eines der Signale wird gegenüber dem anderen zeitlich verschoben

Wird für folgendes verwendet:

- Finden eines Signals in einem anderen
- Messen einer Zeitverzögerung (Delay)
- **Pattern Erkennung in 1D oder 2D**
- **Verwende einen Filterkernel ähnlich wie Muster**

$$r_{\mathrm{xy}(k)} = \sum_{n=-\infty}^{\infty} x(n) \cdot y(n+k)$$

$$\mathrm{corr}(x,y)_k = \sum_{i=-\infty}^{\infty} x_i \cdot y_{k+i}$$

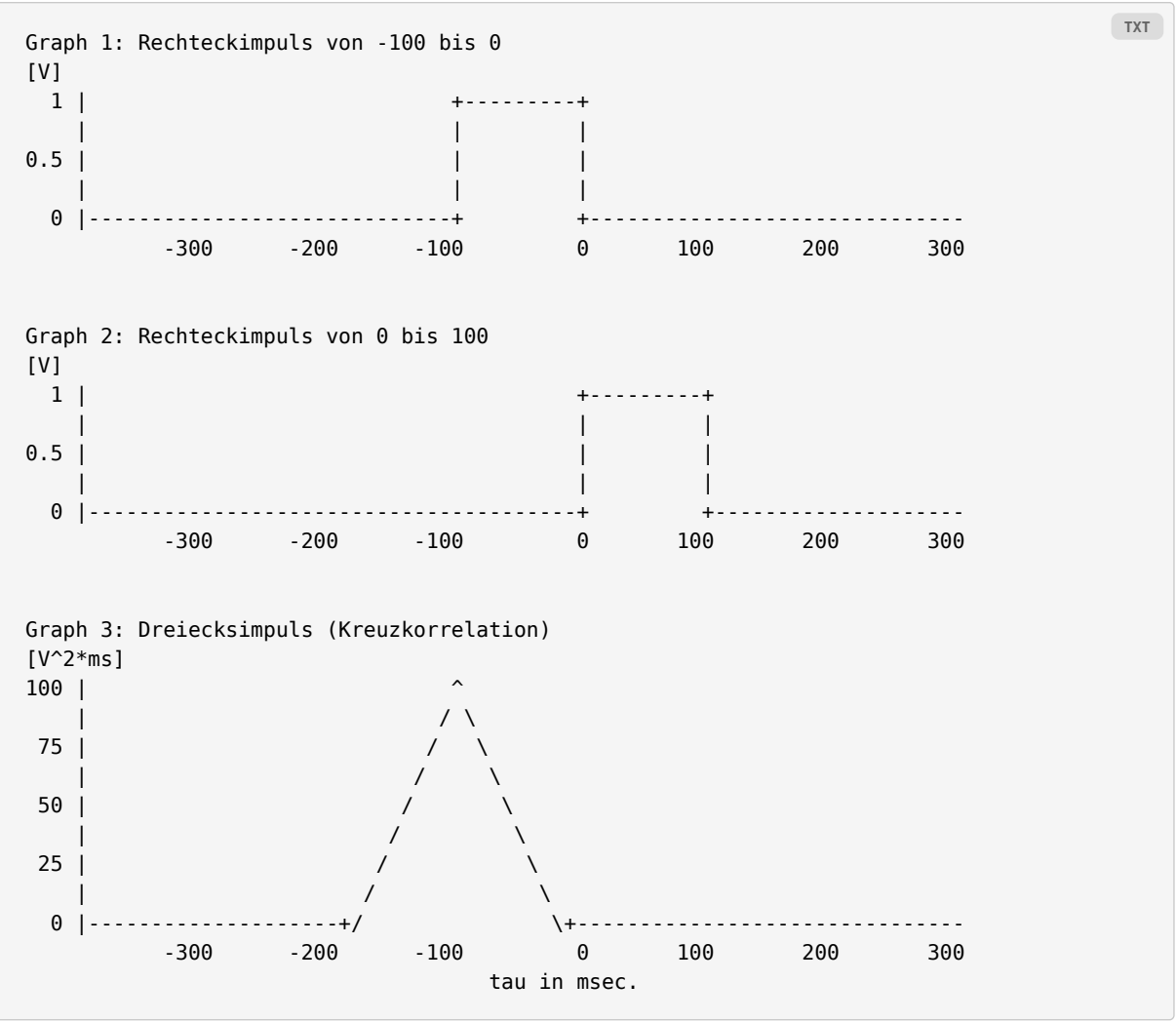
6.5.1. Normalized Cross Correlation

$$\mathrm{ncc}(x,y)_k = \frac{\sum_{i=-\infty}^{\infty} x_i \cdot y_{k+i}}{\sqrt{\left(\sum_{i=-\infty}^{\infty} x_i^2\right) \cdot \left(\sum_{i=-\infty}^{\infty} y_{k+i}^2\right)}}$$

- **1:** Perfekte Übereinstimmung (die Signale sind identisch).
- **−1:** Perfekte Gegenphasigkeit (das genaue Gegenteil/invertiert).
- **0:** Unkorreliert (die Signale haben keine Ähnlichkeit miteinander).

6.5.2. Shapes

6.5.2.1. Rechteck



6.5.2.2. Rechteck unterschiedlicher Länge

TXT

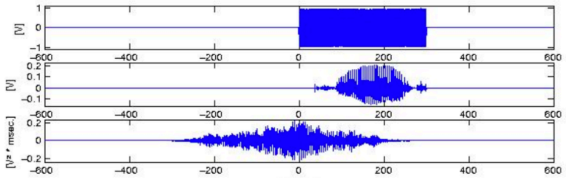
```
Graph 1: Rechteckimpuls von -100 bis -50
[V]
  1 |                             +------+
    |                             |      |
0.5 |                             |      |
    |                             |      |
  0 |-----------------------------+      +-------------------------------------
         -250    -200    -150    -100     -50       0      50     100     150


Graph 2: Rechteckimpuls von 0 bis 100
[V]
  1 |                                           +--------------+
    |                                           |              |
0.5 |                                           |              |
    |                                           |              |
  0 |-------------------------------------------+              +--------------
         -250    -200    -150    -100     -50       0      50     100     150

Graph 3: Trapezimpuls (Kreuzkorrelation)
[V^2*ms]
 50 |                      +------+
    |                    /          \
 25 |                  /              \
    |                /                  \
  0 |--------------+                      +---------------------------------------
         -250    -200    -150    -100     -50       0      50     100     150
                                                    tau in msec.
```

6.5.2.3. Dreieck & Rechteck

TXT

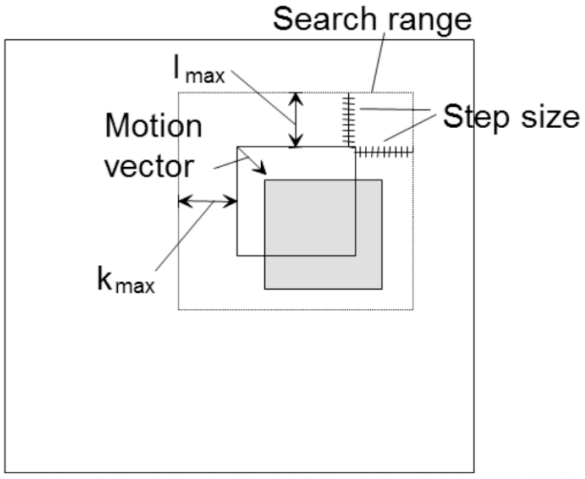
```
Graph 1: Dreiecksimpuls von -100 bis -50 (Spitze bei -75)
[V]
  1 |                               ^
    |                              / \
0.5 |                             /   \
    |                            /     \
  0 |----------------------------+       +-------------------------------------
         -250    -200    -150    -100    -50     0      50     100     150


Graph 2: Rechteckimpuls von 0 bis 100
[V]
  1 |                                           +--------------+
    |                                           |              |
0.5 |                                           |              |
    |                                           |              |
  0 |-------------------------------------------+              +--------------
         -250    -200    -150    -100    -50     0      50     100     150


Graph 3: Faltung/Kreuzkorrelation (abgerundetes Trapez)
[V^2*ms]
 25 |                      .-----------.
    |                    /               \
12.5|                  /                   \
    |                /                       \
  0 |--------------'                           `---------------------------------
         -250    -200    -150    -100    -50     0      50     100     150
                                                   tau in msec.
```

6.5.3. Sprach & Noise Signal



6.5.4. Feature Matching 2D

- den optimalen Verschiebungsvektor finden
- einen Block um ein Feature-Point herum
- basierend auf einem similarity measure



$$E_{\mathrm{WSSD}(u)} = \sum_i w(x_i)[I_1(x_i + u) - I_0(x_i)]^2$$

6.6. Auto-Correlation

Misst die Ähnlichkeit eines Signals **mit sich selbst**
- oft zeitlich verschoben (Lag / Verzögerung)

Wird für folgendes verwendet:
- Finden von Periodizitäten
- Mustern oder Rhythmen in einem Datensatz

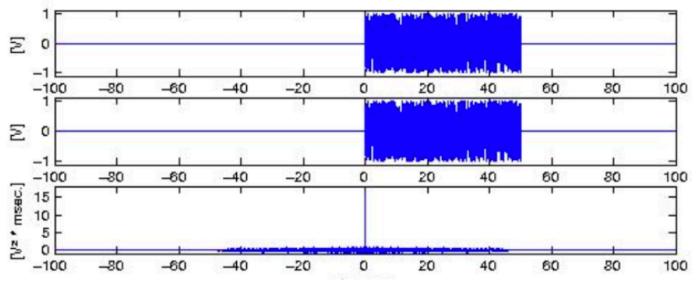
Discrete Auto-Correlation-Function

$$r_{xx(k)} = \sum_{n=-\infty}^{\infty} x(n) \cdot x(n+k)$$

Normalized Auto-Correlation-Function

$$\rho_{xx(k)} = \frac{r_{xx(k)}}{r_{xx(0)}}$$

6.6.1. White Noise



- **Delta-Funktion ($\delta(\tau)$):** Rauschen korreliert ausschliesslich dann mit sich selbst, wenn es exakt unverschoben ist ($\tau = 0$).
- **Vollständige Auslöschung:** Jede zeitliche Verschiebung ($\tau \neq 0$) führt sofort dazu, dass es keinerlei Übereinstimmung mehr gibt.

6.6.2. Shapes

6.6.2.1. Rechteck

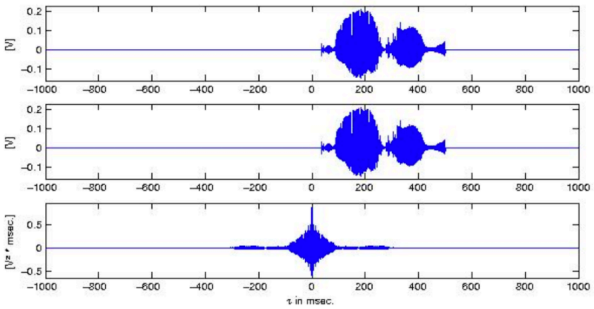
TXT

```
Graph 1: Rechteckimpuls von 0 bis 100
[V]
  1 |                                     +---------+
    |                                     |         |
0.5 |                                     |         |
    |                                     |         |
  0 |-------------------------------------+         +-----------------
            -200        -100       0        100       200


Graph 2: Rechteckimpuls von 0 bis 100 (identisch)
[V]
  1 |                                     +---------+
    |                                     |         |
0.5 |                                     |         |
    |                                     |         |
  0 |-------------------------------------+         +-----------------
            -200        -100       0        100       200


Graph 3: Dreiecksimpuls (Autokorrelation)
[V^2*ms]
100 |                                       ^
    |                                      / \
 75 |                                     /   \
    |                                    /     \
 50 |                                   /       \
    |                                  /         \
 25 |                                 /           \
    |                                /             \
  0 |-------------------------------+               +-----------------
            -200        -100       0        100       200
                                          tau in msec.
```

6.6.2.2. Dreieck

```
Graph 1: Dreiecksimpuls von 0 bis 100
[V]
  1 |                                                  ^
    |                                                /   \
0.5 |                                              /       \
    |                                            /           \
  0 |--------------------------------------------------------+             +-----
             -200    -150    -100     -50       0      50     100     150


Graph 2: Dreiecksimpuls von 0 bis 100 (identisch)
[V]
  1 |                                                  ^
    |                                                /   \
0.5 |                                              /       \
    |                                            /           \
  0 |--------------------------------------------------------+             +------
             -200    -150    -100     -50       0      50     100     150


Graph 3: Glockenartige Kurve (Autokorrelation / kubische Splines)
[V^2*ms]
 33 |                                                  _--_
 30 |                                                /    \
    |                                              /        \
 20 |                                            /            \
    |                                          /                \
 10 |                                        /                    \
    |                                    .-'                        `-.
  0 |-----------------------------------'                              `-----
             -200    -150    -100     -50       0      50     100     150
                                                 tau in msec.
```

6.6.2.3. Sprach Signal



6.6.3. Feature Extraction

$$E_{\mathrm{AC}(\Delta u)} = \sum_i w(x_i)[I_0(x_i + \Delta u) - I_0(x_i)]^2$$

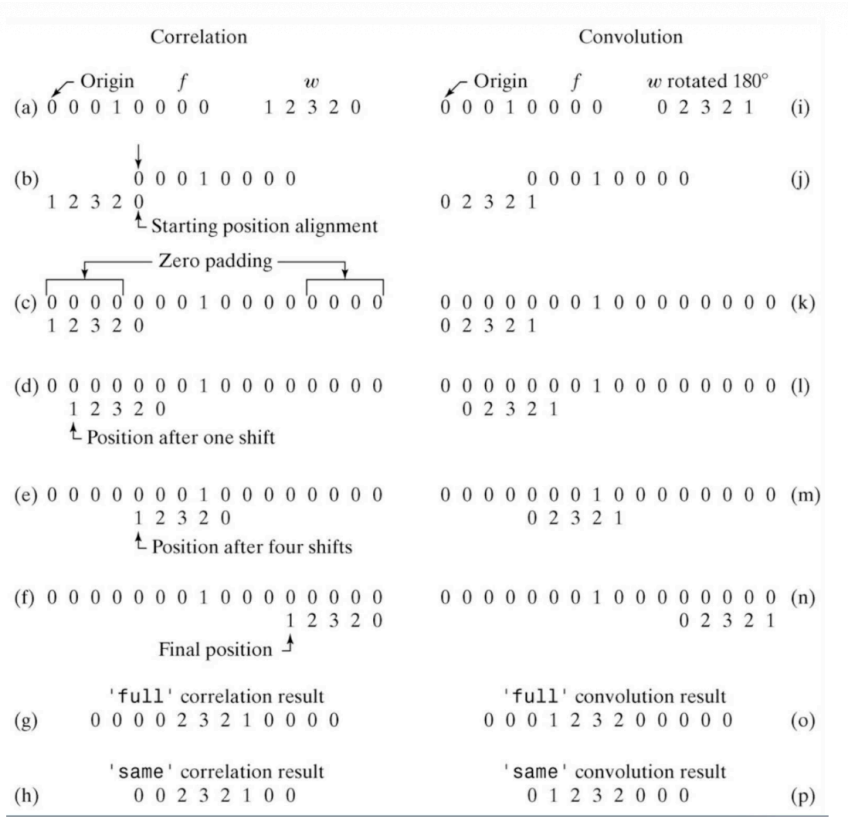
- Original-Bild wird um einen Vektor verschoben
- Differenz Original zu Verschiebung wird berechnet
  - negativ, geht nicht, deshalb wird es quadriert
- Punkt ist dann markant, wenn eine Verschiebung in jede beliebige Richtung zu einer grossen Änderung in der Fehlerfunktion führt
- **Feature = Räumliches Rauschen:** Ein markanter Bildpunkt (z. B. eine Ecke) verhält sich räumlich exakt so, wie sich weisses Rauschen zeitlich verhält

6.7. Konvolution vs Korrelation

Eigenschaft	Konvolution (Faltung)	Korrelation
Hauptziel	**Signal verändern** (Filtern oder Effekt anwenden)	**Muster erkennen** (Ähnlichkeit finden)
Mechanik	**Umdrehen**, Schieben & Multiplizieren	Schieben & Multiplizieren
Computer Vision	**Bildbearbeitung**: Weichzeichner-Filter über ein Foto legen.	**Bilderkennung**: Gesicht einer Person in einem Gruppenfoto suchen.

6.7.1. 1D

- Convolution
  - Filter wird um 180° gedreht
  - reflektiert kernel (filter) auf dem ersten Pixel
- Correlation
  - verschiebt den kernel nach links bis die Überlappung 1 Pixel beträgt



full: Bild wird so gross wie filter **same**: wird so gross wie original Bild

6.7.2. 2D

5x5 Bild f, 3x3 kernel w

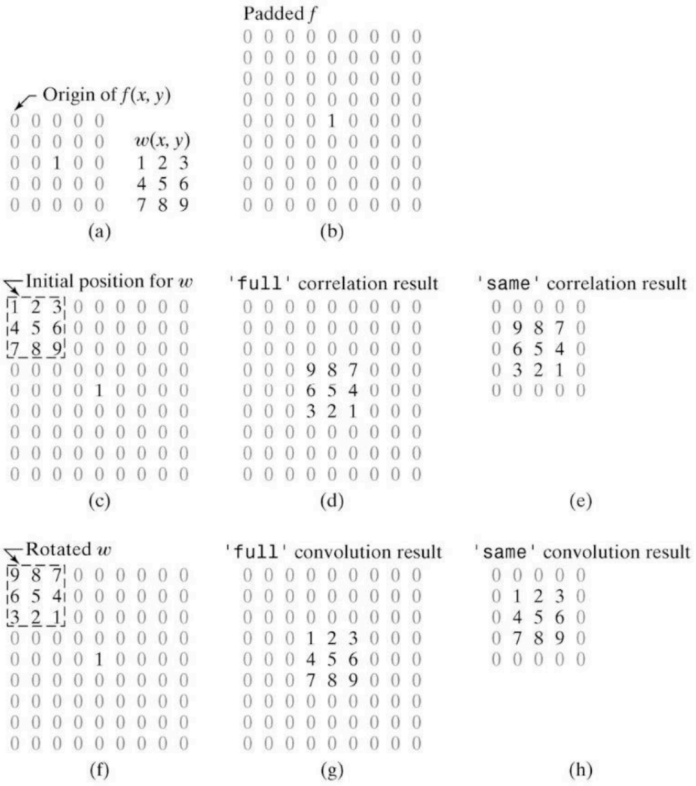
- correlation (c, d, e)
- convolution (f, g, h)

same (e, h)

- Grösse des Results ist gleich wie das Bild f

full (d, h)

- Bild f: $m \cdot n$
- Kernel w: $p \cdot q$
- Resultat: $(n + p + 1) \cdot (m + q + 1)$



6.8. Linear System

- wandelt ein Eingangssignal in ein Ausgangssignal um.

6.9. Linear Filters

6.9.1. Low Pass Filters (Tiefpass)

Tiefpassfilter dienen der Glättung und Rauschunterdrückung.

6.9.1.1. Mean Filter

Berechnet das arithmetische Mittel der Nachbarschaftspixel (arithmetic mean of neighborhood).

$$\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$$

6.9.1.2. Gaussian Filter

Approximation der Normalverteilung. Erzeugt weniger Unschärfe im Vergleich zum Mean Filter (less blur compared to mean filter).

$$\frac{1}{16}\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{pmatrix}$$

6.9.2. High Pass Filters (Hochpass)

Hochpassfilter betonen Kanten und Details durch die Berechnung von Helligkeitsunterschieden.

6.9.2.1. Prewitt Operator

Kantenfindungs-Filter für vertikale (P_x) und horizontale (P_y) Kanten.

$$P_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad P_y = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

6.9.2.2. Sobel Operator

Ähnlich wie Prewitt, gewichtet aber die direkten Nachbarn stärker.

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

6.9.2.3. Sobel Operator (Diagonal)

Angepasst, um diagonale Kantenverläufe zu betonen.

$$S_x = \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix} \quad S_y = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

6.9.2.4. Laplace Filter

Identifiziert Pixel mit der höchsten „Krümmung“ (pixels with highest „curvature“) und reagiert stark auf feine Details oder Rauschen. Vier verschiedene Kernel-Varianten:

$$L_1 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad L_2 = \begin{pmatrix} 0 & -1 & -1 \\ -1 & 6 & -1 \\ -1 & -1 & 0 \end{pmatrix}$$

$$L_3 = \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix} \quad L_4 = \begin{pmatrix} -1 & -2 & -1 \\ -2 & 12 & -2 \\ -1 & -2 & -1 \end{pmatrix}$$

6.9.2.5. Compass Filters

Durch die Anwendung von 45°-Rotationen auf die Kernel P_x (Prewitt) und S_x (Sobel) ergeben sich für jeden Typ jeweils 8 Variationen, um Kanten in alle Himmelsrichtungen (Kompass) zu detektieren.

7. Filter for Edge Detection

7.1. Segmentation

- **Ziel**: Objekterkennung
- **Ansätze**:
  - **Randbasiert**: Grenzt Objekte durch ihre Aussenkanten ab → erfordert zwingend Kantenerkennung
  - **Regionenbasiert**: Grenzt Objekte durch ihre Aussenkanten ab → erfordert zwingend Kantenerkennung

7.2. Ursprung von Kanten

- Richtungsänderung der Oberflächennormalen (z.B. eine scharfe Würfelkante)
- Aneinandergrenzende Farben (z.B. der rote Aufdruck auf einer weissen Dose)
- Änderung der Tiefe (z.B. der Rand eines Objekts vor einem entfernten Hintergrund)
- Änderung der Beleuchtung (z.B. die Grenze eines harten Schlagschattens)

7.3. Kantenerkennung

DEFINITION: Starke Änderungen der Bildintensität

7.3.1. 1D Kantenerkennung

- **Formen**:
  - **Step**: Abrupte Steigung (ansteigende Kante)
  - **Ramp**: Allmählicher Anstieg
- **Kantenstärke**: Je höher der Peak der Ableitung, desto extremer die Kante
- **Problem**: Ideale Stufen sind nicht differenzierbar (benötigt numerische Näherung)

7.3.1.1. Ansätze zur Kantenfindung

- **Differenzieren der Bildintensität**: Helligkeitsunterschiede ausrechnen
  - Reicht alleine nicht aus (verstärkt das Rauschen)
- **Optimale Filter** nach Qualitätskriterien (z.B. Canny-Filter)
- **Matched Filtering**: Eine „Schablone“ verwenden, um das Signal der Kante zu maximieren und Rauschen zu minimieren

Praktische Umsetzung (1D)

- Differenzen zeigen die Intensitätssprünge auf
- Masken-Berechnung zwingend über Korrelation
  - Faltung (Convolution) ist hier falsch herum
  - Korrelation bewahrt Ausrichtung der Pixel und Vorzeichen

7.3.2. Kantenerkennung durch Differentiation

- **Problem**: Bilder bestehen aus einzelnen Pixeln statt aus fliessenden Kurven. Klassisches Ableiten ist daher unmöglich.
- **Lösung**: Man berechnet stattdessen einfach den Helligkeitsunterschied (Minus-Rechnen) zwischen benachbarten Pixeln.

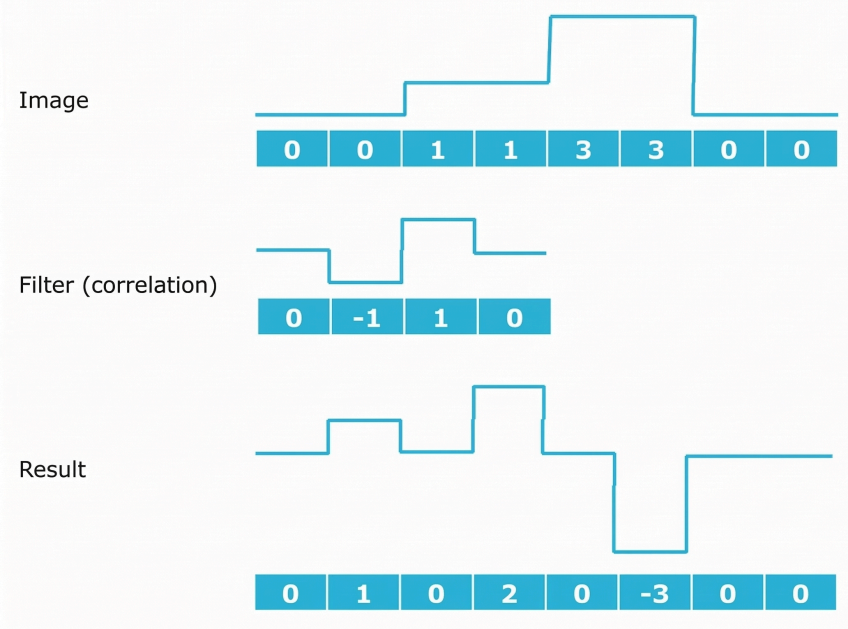
7.3.2.1. Einfache / Vorwärts Differenz

- Entsprechender Filter: $[-1, 1]$

- Zieht den aktuellen Pixelwert vom rechten Nachbarn ab

$$\frac{\partial I}{\partial x} \approx I(x+1, y) - I(x, y)$$

Beispiel:



1. **Schritt** $[0, 0]$: $(-1 \times 0) + (1 \times 0) = 0$
2. **Schritt** $[0, 1]$: $(-1 \times 0) + (1 \times 1) = 1$ (Kante erkannt)
3. ...

7.3.2.2. Zentrale Differenz

- Entsprechender Filter: $\left[-\frac{1}{2}, 0, \frac{1}{2}\right]$
  - Vergleicht nur linken und rechten Nachbarn, Mitte wird mit 0 ignoriert und erhält das Ergebnis

$$\frac{\partial I}{\partial x} \approx I(x+1, y) - I(x-1, y)$$

Beispiel:

- **Bildsignal**: $[0, 0, 1, 1, 3, 3, 0, 0]$
- **Filter**: $\left[-\frac{1}{2}, 0, \frac{1}{2}\right]$

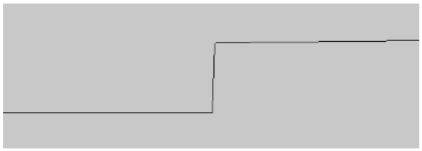
1. **Schritt** $[0, 0, 1]$: $\left(-\frac{1}{2} \times 0\right) + (0 \times 0) + \left(\frac{1}{2} \times 1\right) = 0.5$
2. **Schritt** $[0, 1, 1]$: $\left(-\frac{1}{2} \times 0\right) + (0 \times 1) + \left(\frac{1}{2} \times 1\right) = 0.5$
3. ...

- **Resultat**: $\left[0, \frac{1}{2}, \frac{1}{2}, 1, 1, -\frac{3}{2}, -\frac{3}{2}, 0\right]$

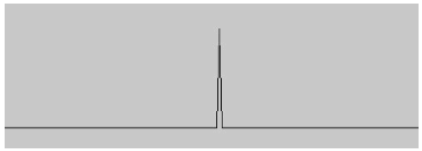
7.3.3. Rauschen

DEFINITION: Besteht aus vielen steilen Flanken (Hochfrequenz)

- **Empfindlichkeit**: Differentiation reagiert auf jeden kleinsten Intensitätssprung
- **Verstärkung**: Ableitung macht das Rauschen im Kantenbild deutlich schlimmer
- **Resultat**: Wahre Kanten gehen in einer Vielzahl falscher „Kanten-Peaks“ unter



image



edge

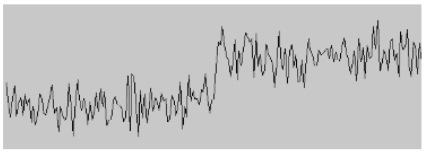
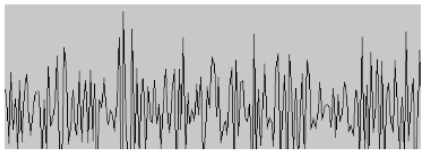


image with noise



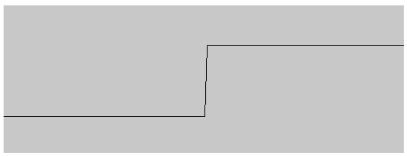
problem: many edges

7.4. Canny Filter

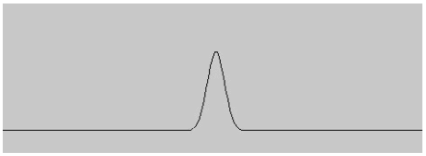
- **Ziel**: Optimale Kantenfindung durch drei Kriterien
  - **Erkennung**: Filter reagiert stark auf echte Kanten
  - **Lokalisierung**: Erkannte Kante liegt exakt auf der Originalkante
  - **Eindeutigkeit**: Nur eine einzige Kante pro Übergang (keine Mehrfachantworten)

7.4.1. Mathematische Optimierung

- Wir leiten nicht das Bild ab, sondern den Gauss-Filter
- Durch das Assoziativgesetz gilt, $D * (G * I) = (D * G) * I$
  - D: Differentiation (Ableitung)
  - G: Gauss-Glättung
  - I: Bild (Image)
- **Vorteil**: Rechenaufwand wird halbiert. Ein einziger Filter-Durchlauf erledigt Glättung und Kantensuche gleichzeitig.
- Dadurch saubereres Kantenbild mit deutlich weniger Rechenzeit



image



edge

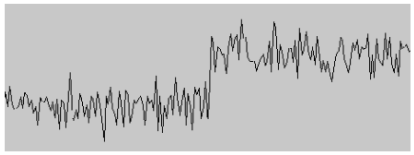
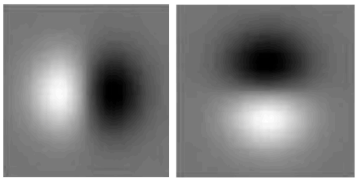


image with noise

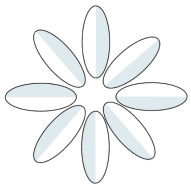
edge

7.4.2. Canny-2D

- Der Filter glättet das Bild entlang der Kante und berechnet die Ableitung (Differenz) senkrecht dazu.



- **Richtungsabhängigkeit**:
  - Kanten können horizontal, vertikal oder diagonal verlaufen
  - Der Filter wird in verschiedene Richtungen angewendet (oft 4 Richtungen)
    - theoretisch 8, aber die jeweils gegenüberliegenden Kanten können ignoriert werden



  - Die stärkste Antwort des Filters gibt die Gradientenrichtung an
- **2D-Masken**:
  - **Horizontal**: Erkennt senkrechte Kanten
  - **Vertikal**: Erkennt waagrechte Kanten
  - Bekannte Operatoren: Prewitt (basiert auf Box-Filter) und Sobel (basiert auf Gauss-Filter)

Sobel filter

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

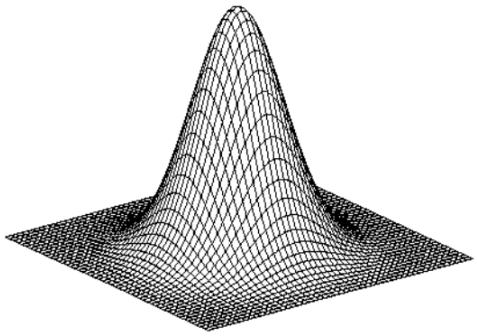
Prewitt filter

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

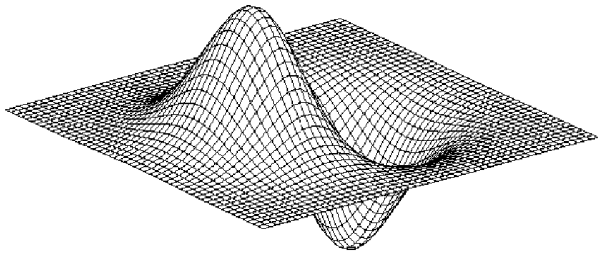
7.4.2.1. Berechnung und Mathematik

- **Ansatz**: Ein normaler Kanten-Filter ist anfällig für Rauschen. Wir verschmelzen Weichzeichner und Kanten-Filter mathematisch zu einem einzigen Filter.
- **1. Glättung (2D-Gauss)**: Eine 3D-Glockenkurve zeichnet das Bild weich.



$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

- **2. Kantenerkennung (Ableitung)**: Die Ableitung der Glockenkurve erzeugt eine sanfte Welle mit Berg (positiv) und Tal (negativ).



$$\frac{\partial}{\partial u} h_\sigma$$

- **Resultat**: Ein 3D-Filter, der Helligkeitsunterschiede über einen breiteren, geglätteten Bereich misst anstatt nur starr zwei Einzelpixel zu vergleichen. Er unterdrückt feines Rauschen und schlägt nur bei echten Kanten aus.

7.4.3. 2D Edge Detection Recipe

Beschreibt die Implementierung eines isotropen (richtungsunabhängigen) Kantendetektors

1. **Glättung**: 2D-Gauss-Filter unterdrückt Rauschen
2. **Ableitung**: Berechnung der Gradienten-Komponenten dx und dy (horizontale und vertikale Änderungen)
3. **Kantenstärke (Magnitude)**: Kombination der Ableitungen zur Gesamthelligkeit der Kante
4. **Sigma (σ)**: Filterbreite bestimmt Detailgrad (klein = fein, gross = grob)

7.4.3.1. Berechnung der Kantenstärke

- **Präzise (teuer)**: $\sqrt{dx^2 + dy^2}$
- **Effizient (günstig)**: $|dx| + |dy|$ oder $\max(|dx|, |dy|)$

7.4.3.2. Interpretation der Polarität

- **Hell zu Dunkel**: Abfallende Flanke, wird oft als schwarze Kante dargestellt
- **Dunkel zu Hell**: Ansteigende Flanke, wird als weisse Kante dargestellt

7.4.4. Canny-Filter: Letzte Schritte

7.4.4.1. 1. Kantenrichtung (Winkel)

- **Zweck**: Exakte Richtung für die Ausdünnung bestimmen
- **8 Richtungen**: Berücksichtigt alle Himmelsrichtungen

- **Berechnung**: $\varphi = \text{atan2}(dy, dx)$
- **Vorteil**: Verhindert Division durch Null und deckt alle vier Quadranten ab

7.4.4.2. 2. Non-Maximum Suppression (NMS)

- **Ziel**: Kanten auf exakt 1 Pixel Breite ausdünnen
- **Methode**: Behält nur das lokale Maximum senkrecht zum Kantenverlauf, der Rest wird gelöscht (auf 0 gesetzt)
- **4 Achsen reichen**: 0°, 45°, 90°, 135° genügen für die Suche, da die Suchrichtung auf der Achse (links/rechts) egal ist

7.4.4.3. 3. Hysteresis Thresholding

- **Ziel**: Rauschen entfernen und zusammenhängende Kanten stabilisieren
- **Parameter-Tuning**: Die perfekten Schwellenwerte müssen je nach Bild (Kontrast, Helligkeit) und gewünschtem Detailgrad durch Ausprobieren ermittelt werden
- **Prinzip**: Nutzt zwei Schwellenwerte („High“ und „Low“)
  - $\text{Kante} > \text{High} \rightarrow$ Starke, sichere Kante
  - $\text{Kante} < \text{Low} \rightarrow$ Keine Kante, wird sofort gelöscht
  - $\text{Low} < \text{Kante} < \text{High} \rightarrow$ Schwache Kante, wird nur behalten, wenn sie eine starke Kante berührt

7.5. Zusammenfassung Canny-Pipeline

Der komplette Ablauf des Canny-Filters lässt sich in vier aufeinanderfolgende Kernschritte unterteilen:

1. **Glättung (Rauschunterdrückung)**: Das Bild wird mit einem Gauss-Filter weichgezeichnet, um feines Rauschen zu entfernen.
2. **Gradientenberechnung (Kantenstärke & Richtung)**: Bestimmung der Intensitätsänderung und des exakten Winkels der Kante durch Ableitung.
3. **Non-Maximum Suppression (Ausdünnung)**: Die Kanten werden senkrecht zu ihrer Verlaufsrichtung auf exakt ein Pixel Breite verdünnt.
4. **Hysteresis Thresholding (Kantenverfolgung)**: Endgültige Filterung durch zwei Schwellenwerte, um echte Kanten zu verbinden und Rauschen endgültig zu löschen.

8. Hough Tranform & Feature Detection

8.1. Object Recognition Using Hough Transform

Zweck von Hough Transform ist das detectieren von Formen in Bildern:

- Linien
- Kreise
- andere Formen mit general Hough Tranform

Die Formen werden durch Parameter beschrieben:

- Linien:
  - Distanz zum Nullpunkt
  - Winkel zwischen Linie und X-Achse
- Kreise
  - x Koordinate des Zentrums
  - y Koordinate des Zentrums
  - Radius

Vorgehen:

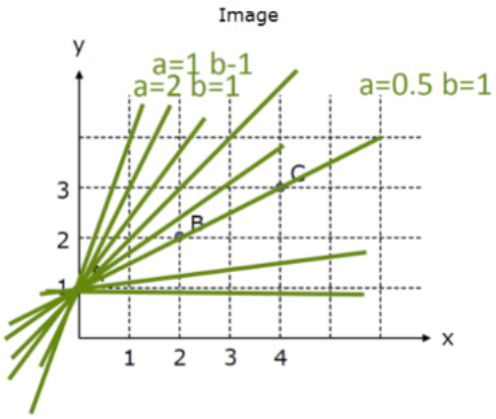
- Linien Erkenne (z.B. Canny)
- Für jeden Punkt auf einer Linie:
  - Parameter aller Formen welche diesen Punkt enthalten
  - Parameter die beide vielen Punkten gleich berechnet wurden bezeichnen zusammengehörende Formen

8.2. Hough Transforma

Ziel: Linien / Geraden / andere Formen in einem Bild finden

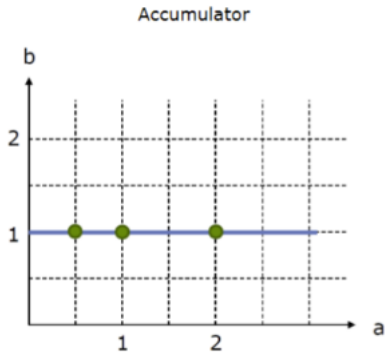
$$y = ax + b$$

Bildraum:



Akkumulator:

- ist ein Zähler
- pro Treffer wird der Count hochgezählt
- dort wo sich die Geraden schneiden (im Bild auf dem grünen Punkten) ist eine Kante
- Steigung a und Achsenabschnitt b
- jede Linie entspricht genau einem Punkt im Akkumulator
- y-Achsenabschnitt $\rightarrow$ b
- Steigung $\rightarrow$ a
- ein Punkt im Bild wird zu einer Linie im Akkumulator



Verwendung des Akkumluators:
- Akkumulator diskretisieren
- Erhöhen des Akkmulator für jeden Punkt
  - $+1$
  - Proportional zu der Kantenstärke
- Suche das Maximum im Parameterraum

8.2.1. Vorgehen

Steigung a:

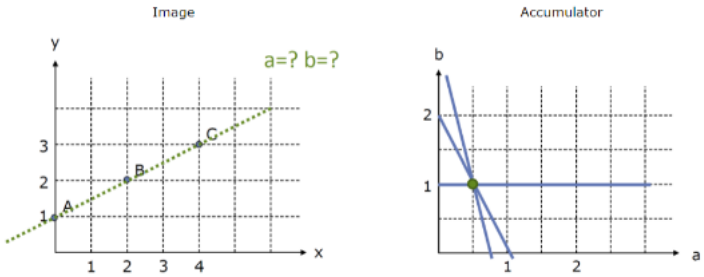
$$a = \frac{\Delta y}{\Delta x} = \frac{\text{Unterschied in der Höhe}}{\text{Unterschied in der Breite}}$$

Gleichung umformen:

$y = ax + b$ nach b umformen $\rightarrow b = -xa + y$

oder direkt:
- y-Achsenabschnitt $\rightarrow$ b
- Steigung $\rightarrow$ a

8.2.1.1. Beispiel 1



Steigung a:

$$a = \frac{\Delta y}{\Delta x} = \frac{\text{Unterschied in der Höhe}}{\text{Unterschied in der Breite}} = \frac{1}{2} = 0.5$$

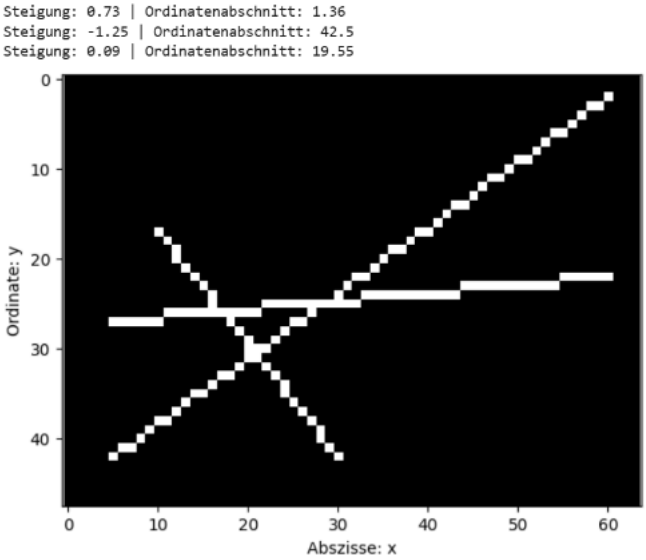
Punkt A(0,1)
- $b = -0c \cdot a + 1$
- y-Achsenabschnitt $\rightarrow$ b = 1
- Steigung: 0

Punkt B(2,2)
- $b = -2c \cdot a + 2$
- y-Achsenabschnitt $\rightarrow$ b = 2
- Steigung: -2

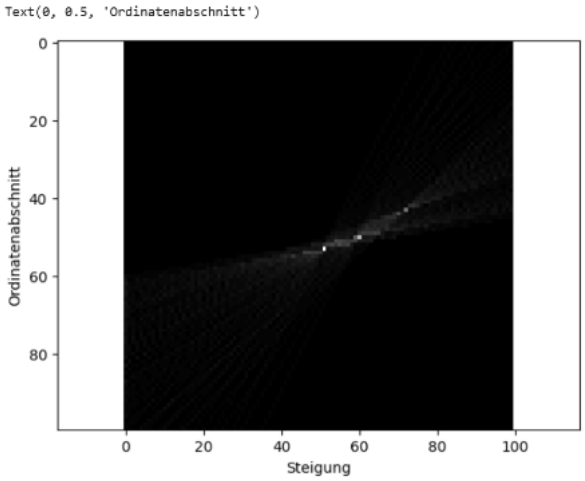
Punkt C(4,3)

- $b = -4c \cdot a + 3$
- y-Achsenabschnitt → b = 3
- Steigung: -4

8.2.1.2. Beispiel 2



Akkumulator:
- an drei Stellen gibt es einen maximalen Punkt
  - somit die drei Linien vom Bild oben



→ man hat immer Abweichungen, da man sehr grob differenziert
- Abweichung geringer zu machen → Akkumulator und Bild vergrössern (Rechenaufwand wird auch grösser)

8.3. Canny vs. Hough

- Canny-Filter findet die Farben
  - gibt zurück an welchen Koordinaten eine Kante liegt
  - findet oft zu viele Kanten
- Hough Transformation findet die Formen mathematisch
  - Pixel die zusammen eine Gerade bilden und somit eine Kante ist
  - kann ungleiche Linien finden
    - Bsp. Spurmarkierung Autobahn
  - filtert unklare Kanten aus, da er bei Rauschen keine klare Formen erkennt

8.4. Polar Koordinaten

- Vertikalen Linien haben eine Steigung ins Unendliche
- **Lösung**: Polar Koordinaten verwenden

$$x \cos \varphi + y \sin \varphi = r$$

Dadurch bekommt der Akkumulator Sinuswellen.

8.5. Implementation von Hough Transform

1. Akkumulator initialisieren
2. für jeden Pixel (x, y)
   - für jede Richtung φ
     - berechne $r = x \cos \varphi + y \sin \varphi$
     - Inkrementiere Akkumulator an der Stelle (r, φ)
3. Maximum im Akkumulator suchen

8.5.1. Edge Direction

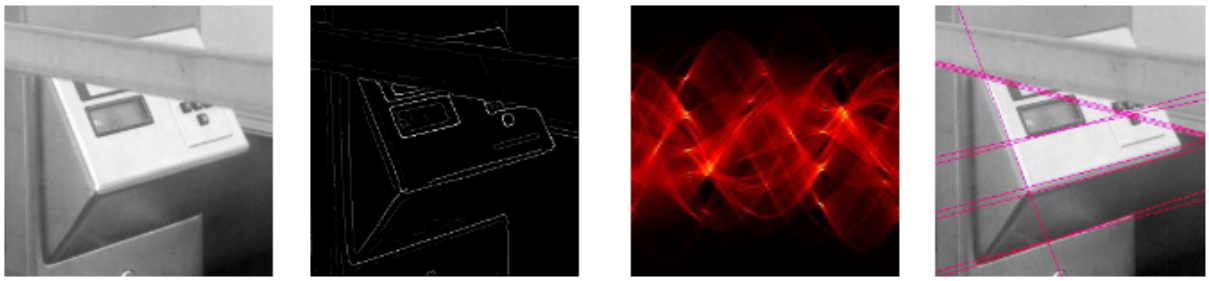
Gebraucht man die Edge Direction kann die Operation kleinflächiger und weniger rechen intensiv durchgeführt werden.

Dafür wird das bild zuerst mit einem Edge Detection Algorithmus bearbeitet (z.B. Canny)

Dadurch erhält man die mögliche Richtung der Kanten (edge normal)

1. Akkumulator initialisieren
2. für jeden Pixel (x, y) mit edge normal $\left(n_x, n_y\right)$
   - berechne $r = x n_x + y n_y$
   - brechne $\varphi = \arctan\left(\frac{n_y}{n_x}\right)$
   - Inkrementiere Akkumulator an der Stelle (r, φ)
3. Maximum im Akkumulator suchen

Resultat von Hough Transform:



8.5.2. Kreis Erkennung

- Ähnlich wie die Line Detection
- Alle Kreise welche eine Edge Pixel enthalten werden berechnet
- Parameterraum ist dreidimensional

8.6. Features und Correspondences

Bei zwei Bildern mit gleichem Inhalt ist es möglich Punkte zu finden die einander entsprechen, d.h. zum gleichen Objekt gehören

Je nachdem wie sich die beiden Bilder unterscheiden, müssen hierbei andere Verfahren angewendet werden:

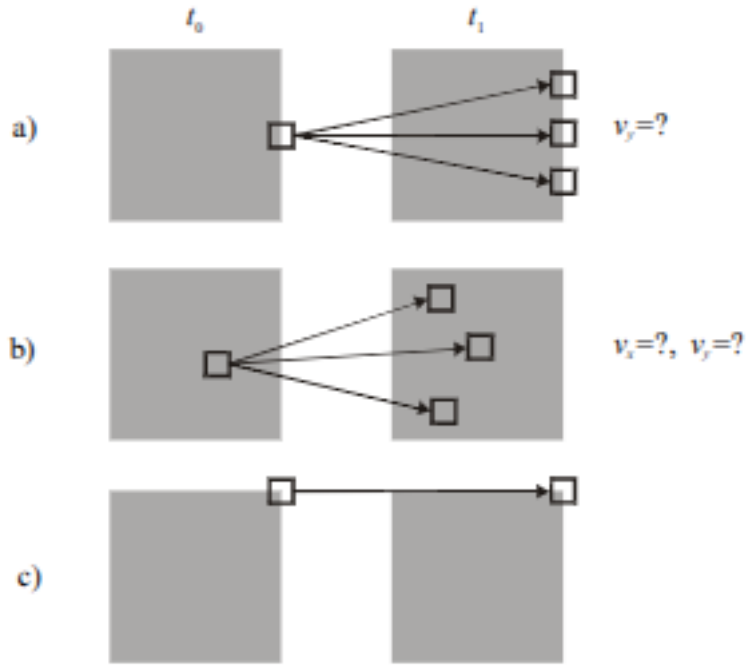
- Zwei Bilder aus der selben Videosequenz:
  - **Verfahren:** Motion Estimation, Optical Flow
- Zwei Bilder aus unterschiedlichen Blickwinkeln
  - **Verfahren:** Disparity Estimation
- Zwei Bilder aus einer rotierten Kamera

- **Verfahren:** Homography Estimation

8.6.1. Problem

Corresponcende kann nur für eine Nachbarschaft um einen Punkt herum eingeschätzt werdem
- Ambivalenzen entstehen

Eignung eines Features:
- Kante -> mittelmässig geeignet
- Fläche -> nicht geeignet
- Ecke -> gut geeignet



Zusätzliche Problematik: Schattenwurf
- Kontur des Schattens unterscheidet sich nicht vom Original
- Schatten sieht je nach Blickwinkel anders aus

8.6.2. Feature Extraction

Messwert um die Eignung von Punkten als Features zu ermitteln:
$E_{AC}(\Delta u) = \sum w(x_i)[I_0(x_i + \Delta u) - I_0(x_i)]^2$

Häufig genutzte Feature Detection Algorithmen:
- Harris
- Lacas Kanade
- Shi Tomasi

→ Hauptquelle von Information sind Image gradients;
- d.h. die Änderung in der Intensität in x und n_y

Formel:
$\frac{\delta^2 I(x,y)}{\delta x^2} \cdot \frac{\delta^2 I(x,y)}{\delta y^2} - \frac{\delta^2 I(x,y)}{\delta x \delta y}$

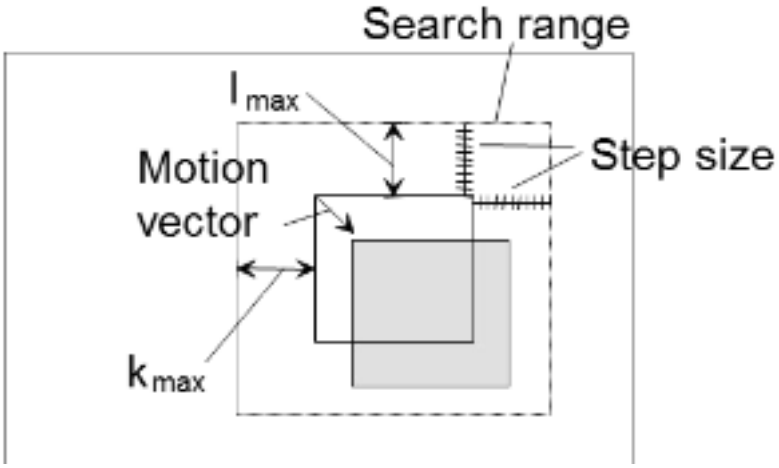
Für die gleichmässige Verteilung von Features kann Adaptive non-maximal suppressions (ANMS) verwendet werden.
- dafür wird ein suppression radius r verwendet

8.6.3. Feature Matching

Ziel: Optimaler Displacement Vektor für einen Block um einen Feature Point herum finden

Basiert auf eine Similarity Measure $E_{\text{WSSD}}(u) = \sum w(x_i)[I_0(x_i + u) - I_0(x_i)]^2$

- Block in einem Suchbereich umher schieben
- Bei jedem Step wird die Similarity Measure berechnet
- Displacement Vektor ist der Vektor von der jetzigen Position zu der Position im Suchbereich mit der besten Similarity Measure



Unter den folgenden Bedingungen funktioniert das nicht:
- Grosse displacements
- Grossen Boxen
- Rotation
- Skalierung
- Andere Perspektiven
- Andere Varianzen
  - Licht, Reflexionen, Noise

Lösung:
- Nicht direkt Pixel values verwenden für das Matching
- Informationen über die Bilder herausfinden
  - Skalierung, Rotation, Perspektive
  - verwendet in der Regel Image Gradiente
  - Repräsentation dieser Information heisst descriptor
- Matching im Feature Raum

9. Image Classification

9.1. Introduction

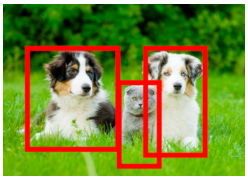
Image Classification / Object Recognition

- Image -> `Class`

```python
def classify(image) -> int:
  # do magic
  return class_label
```

Object Detection

- Image -> `List[(Region, Class)]`



Semantic Segmentation

- Image -> `Array[Class]`

Image Generation

- Text -> Image

9.1.1. Herausforderungen

Background Clutter

- Subjekt von Hintergrund unterscheiden

Illumination

- z.B. overexposed
- Menschen sind gut in Informationen füllen
- erkennen Augen in einem Sack, wissen aber das eine ganze Katze drin ist

Occlusion

- Objekt wird verdeckt

Deformation

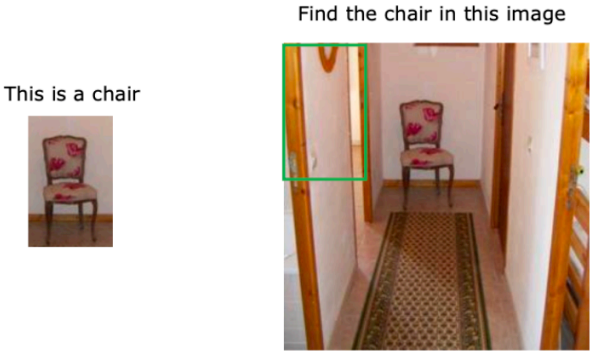
- Formen / Positionen, die nicht erwartet wurden
- auch für neuere Methoden schwierig, da nicht mit solchen Daten trainiert wurden

Intraclass variation

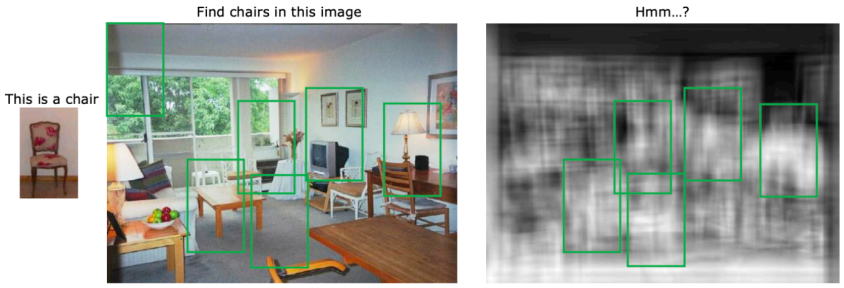
- viele Varianten von „Katze“
- gehören aber in die gleiche Klasse

9.1.2. Object recognition - Klassischer Weg

- Korrelation berechnen
- schiebt Ziel bild über das Bild



- Korrelation funktioniert nur wenn sie gleich oder sehr ähnlich ist:



- Daher verwendet man heutzutage **Data Driven Approaches**

9.2. Data Driven Approaches

Grundprinzip: Sammeln einer Bilddatenbank mit Labels, Training eines Klassifikators und Evaluation mittels Testbildern.

Workflow-Struktur:

- `train`-Phase (Modellbildung)
- `predict`-Phase (Vorhersage von Labels).

Klassische ML-Methoden (ohne neuronale Netze):

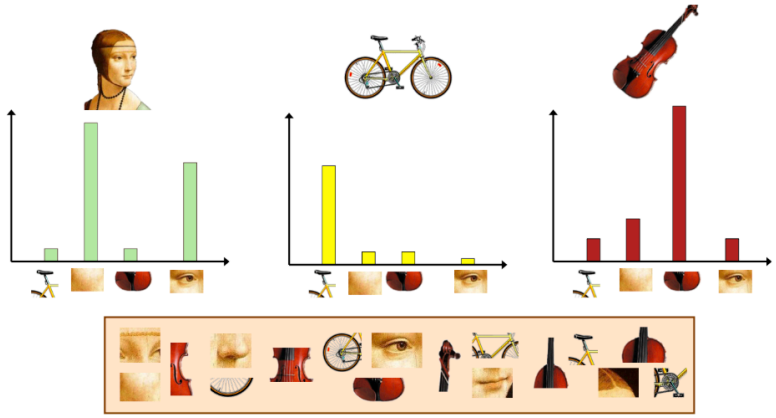
- **Bag-of-Words-Verfahren:** Nutzung von „Code Words“ zur Klassifizierung.
- **Merkmalsextraktion:** Einsatz von Features wie SIFT, SURF, ORB oder HOG.
- **Algorithmen:** Kombination dieser Merkmale mit Clustering, linearen Klassifikatoren, SVMs oder Entscheidungsbäumen.

Moderner Deep-Learning-Ansatz:

- tiefer neuronaler Netze für die simultane Merkmalsextraktion und Klassifizierung.

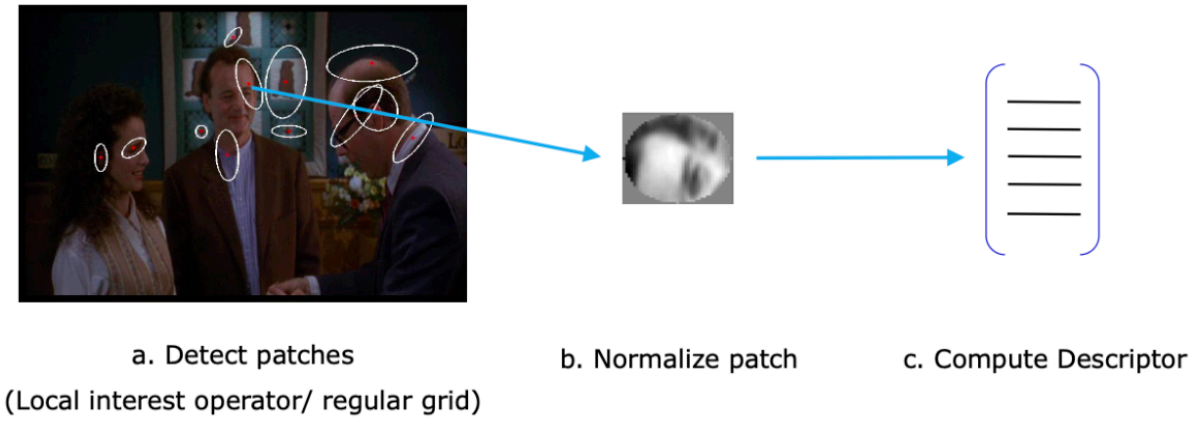
9.3. Bag of Words

- **Zerlegung** eines **Objekts in unabhängige Merkmale** (visuelle Wörter)
  - Gesicht: Auge, Mund, Nase, usw.
- Ignorieren der räumlichen Anordnung der Bildelemente
- Erstellung eines **Histogramms** basierend auf der Häufigkeit dieser Merkmale
- Vergleich von Histogrammen zur **Klassifizierung** verschiedener Objekte



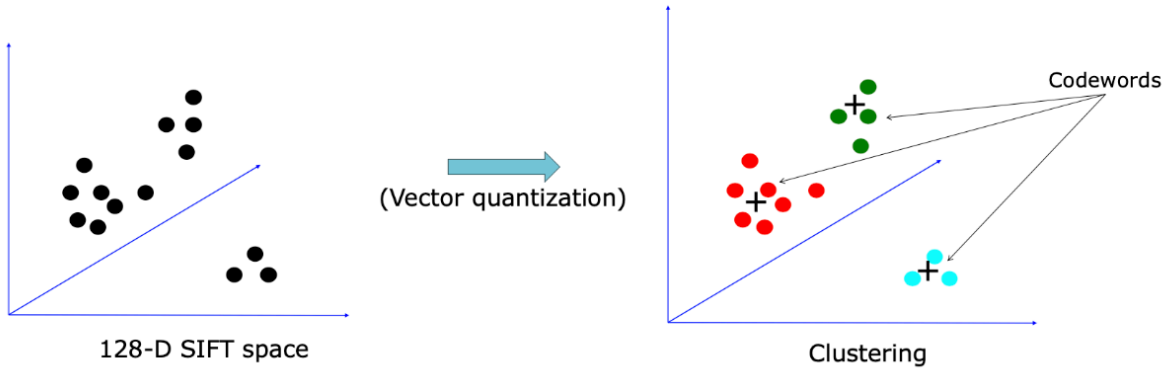
9.3.1. Feature Detection & Representation

- **Identifizierung lokaler interessanter Bereiche** im Bild (z. B. Kanten, Ecken oder Raster-Patches).
- **Normalisierung**
  - Skalierung und Ausrichtung der Bildausschnitte für eine einheitliche Verarbeitung.
- Umwandlung der Patches in mathematische **Vektoren** (z. B. im 128-dimensionalen SIFT-Raum).



9.3.2. Codewords dictionary formation

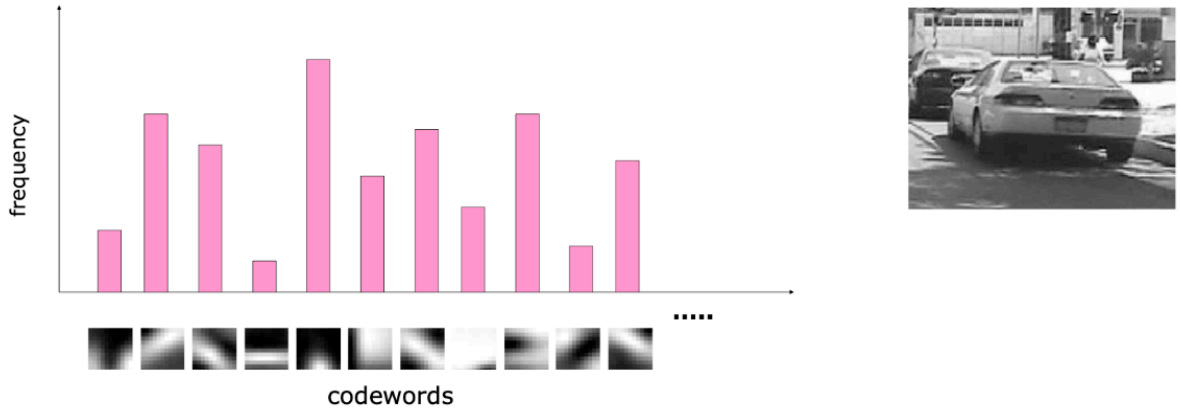
- Abbildung aller **extrahierten Deskriptoren als Punkte** in einem hochdimensionalen Raum.
- **Vektorquantisierung**
  - Zusammenfassung ähnlicher Deskriptoren
  - Viele Werte mit wenigen Werten repräsentieren



- Cluster-Zentren bilden die „visuellen Wörter“ des Wörterbuchs

9.3.3. Image Representation - Histogramm

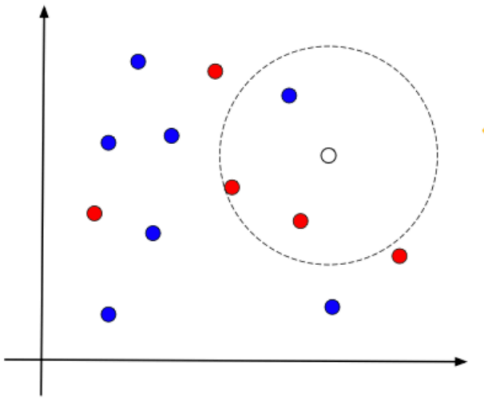
- Jedes Merkmal eines neuen Bildes wird dem ähnlichsten Codeword aus dem Wörterbuch zugeordnet.
- Zählen der Häufigkeit, mit der jedes Codeword im Bild vorkommt.



- Histogramm kann später verwendet werden, um zu klassifizieren was auf dem Bild ist

9.4. Classification: Nearest Neighbor

- Klassifizierung erfolgt durch **Mehrheitsentscheid** (Majority Vote) der k nächsten Nachbarn.
  - Problem k=2? -> derjenige mit der kürzesten Distanz



- **Lernphase:** Besteht lediglich aus dem Speichern (Auswendiglernen) aller Trainingsdaten.

Komplexität

- **Training**: $O(1)$

- keine Modellparameter werden berechnet
- **Vorhersage (Prediction)**: $O(N)$
  - Abstände zu allen N Beispielen werden berechnet.

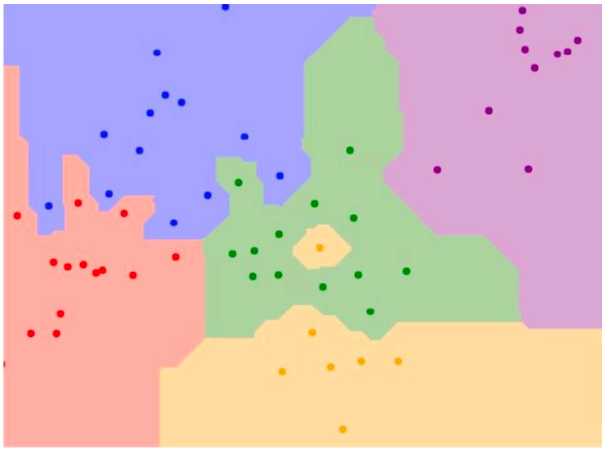
9.4.1. Distanz Funktionen

je nach Anwendungsfall ist L1 oder L2 besser

9.4.1.1. L1 Manhattan

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

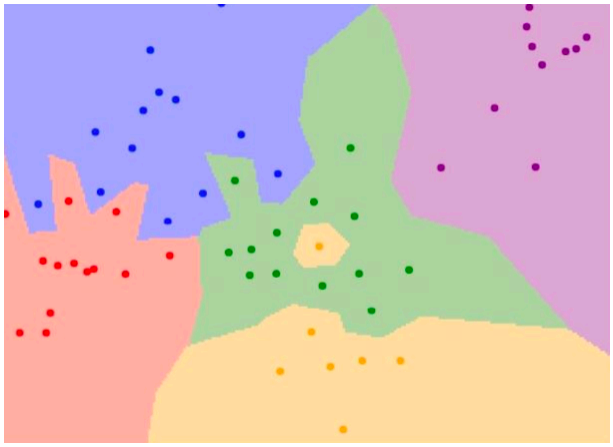
- wenn es kann, macht es gerade Linien



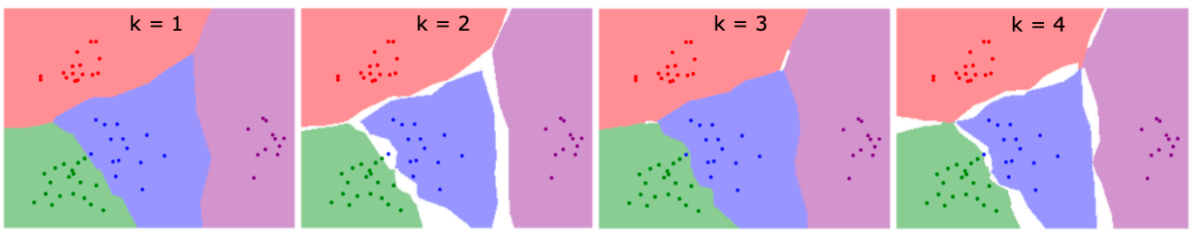
9.4.1.2. L2 Euklidische Distanz

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

- ist runder



9.4.2. Auswirkung von k



die weissen Bereiche:

- Überschneidungen von zwei Bereichen

gerade Nummer:

- mehr Fälle mit Überschneidungen

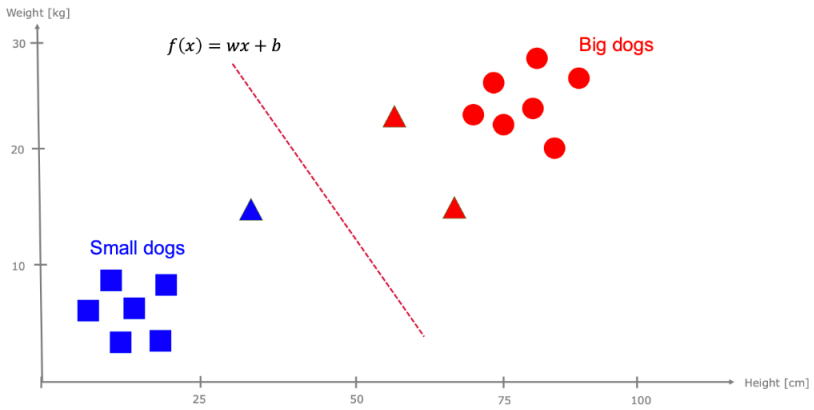
9.4.3. Training - Hyperparameter Evaluation

Wie wählt man k & die Distanzmetrik am besten? -> Hyperparameter Evaluation

Best Practices

- **Trainingsdaten**: Zum „Lernen“ (Speichern der Beispiele).
- **Validierungsdaten**: Zum Vergleichen verschiedener Hyperparameter und zur Auswahl der besten Kombination.
- **Testdaten**: Ausschliesslich für die finale Leistungsbewertung (einmalige Anwendung), um echte Generalisierung zu prüfen.

9.5. Linear classifier



Verwendung einer linearen Funktion um zwei Klassen voneinander zu trennen.

- 2D -> Gerade
- allgemein **Hyperplane**
  - hat eine Ebene weniger als Original Space

$$f(x) = wx + b$$

- w: Gewichtsvektor
  - bestimmt die Ausrichtung der Grenze
- b: Bias
  - verschiebt die Grenze vom Ursprung weg

Generalisierung Der Klassifikator versucht, eine Grenze zu finden, die auch für neue, unbekannte Datenpunkte (im Bild durch Dreiecke dargestellt) korrekt funktioniert.

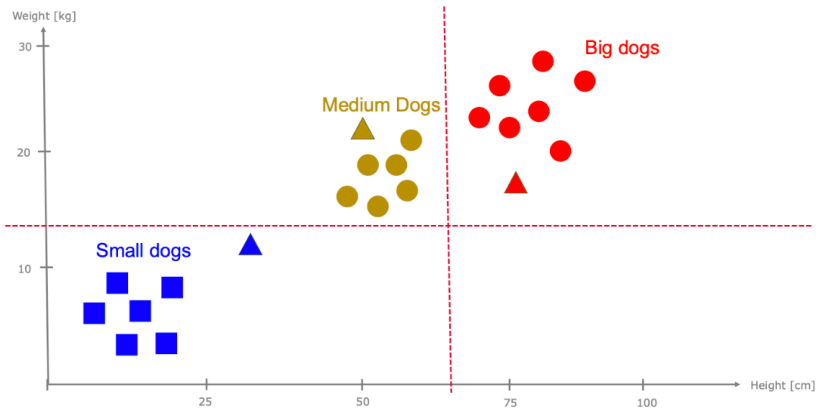
Vorteil: Im Gegensatz zu k-NN (das alle Daten speichert) muss ein linearer Klassifikator nach dem Training nur die Parameter w und b speichern.

Nachteil: ist auf zwei Klassen beschränkt

9.6. Decision Tree Classifier

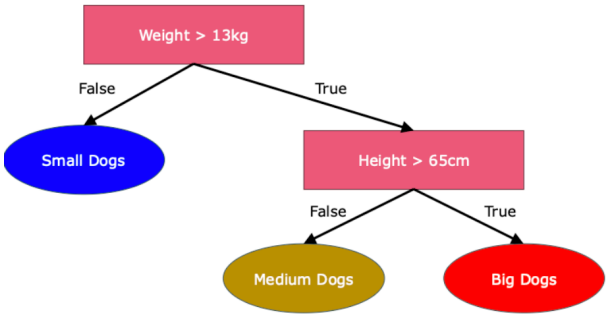
Unterscheidung von mehr als zwei Klassen

Iterative Teilung: Der Merkmalsraum wird schrittweise durch horizontale oder vertikale Linien (Splits) unterteilt.

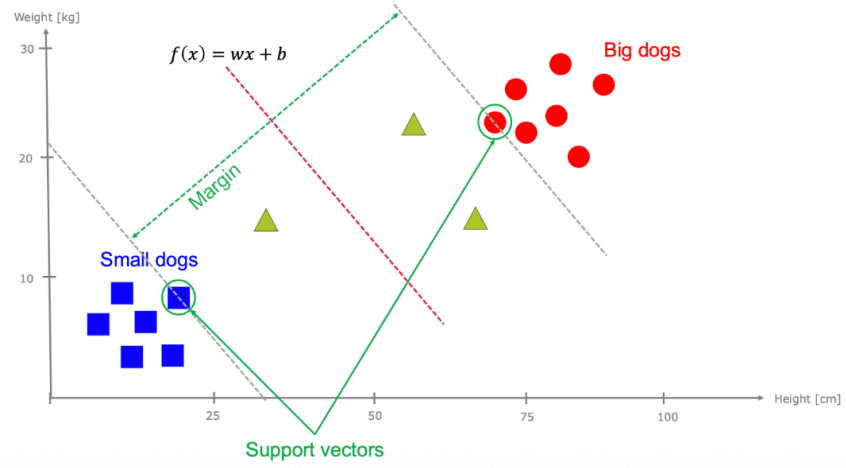


Minimierung der Unreinheit: Das Ziel jedes Splits ist es, die „Impurity“ (z. B. **Gini Impurity**) der resultierenden Gruppen zu senken, sodass die Klassen möglichst sauber getrennt werden.

Baumstruktur



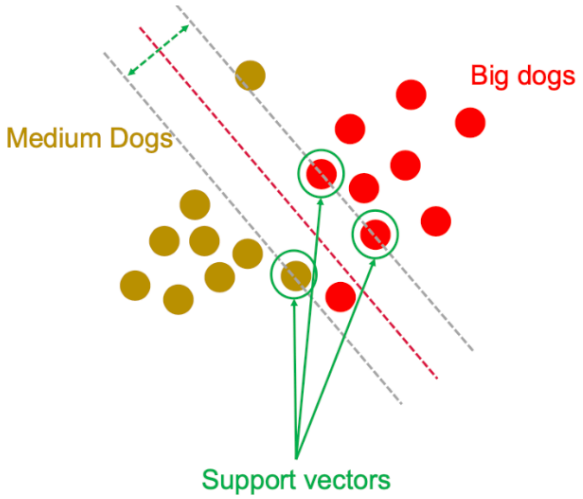
9.7. Support Vector Machine (SVM)



Sucht die Trennlinie mit dem grösstmöglichen Abstand (Margin / Gap) zu beiden Klassen

Support Vectors: Datenpunkte die der Linie am nächsten sind (oder auf ihr liegen)

Soft Margin: Wenn Daten nicht perfekt linear trennbar sind, erlaubt der Soft Margin bewusste Fehlklassifizierungen, um eine stabilere Grenze zu finden.

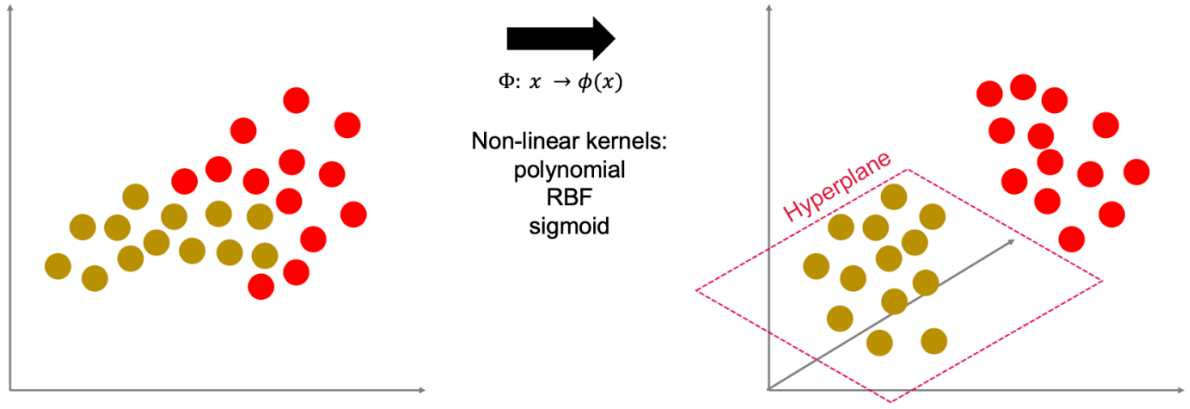



```python
from sklearn import svm
# hard margin (C -> infinity)
svm.SVC(kernel='linear', C=1000)
# soft margin
svm.SVC(kernel='linear', C=1.0)
```

9.8. Nonlinear Support Vector Machine

Klassen sind im ursprünglichen Merkmalsraum (z. B. 2D) vermischt und können nicht durch eine gerade Linie getrennt werden.

Kernel-Trick: Mathematische Methode, um Daten in einen **höherdimensionalen Raum** zu transformieren $\Phi : x \rightarrow \varphi(x)$.



In neuem, hochdimensionalem Raum, können Daten durch **Hyperplane getrennt werden**

```python
from sklearn import svm
# RBF kernel
svm.SVC(kernel='rbf', C=1.0)
# Polynomial kernel
svm.SVC(kernel='poly', C=1.0, degree=3)
```

9.9. Boosting

Kombination vieler einfacher Modelle (**Weak Classifiers**) zu einem hochpräzisen Gesamtmodell (**Strong Classifier**).

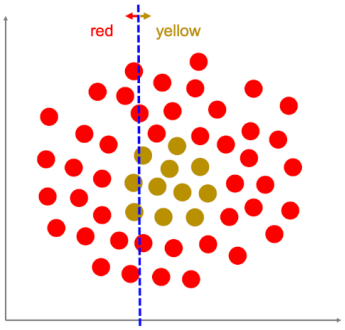
$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + ...$$

- $f_{i(x)}$: einzelner schwacher **Klassifikator**
- α_i: **Gewicht des Klassifikators**, wird beim **Training bestimmt**

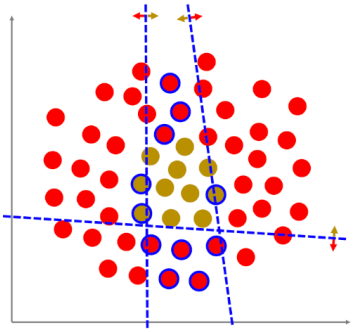
9.9.1. Iterativer Trainingsprozess

- **Initialisierung**: Zu Beginn haben alle Datenpunkte das gleiche Gewicht (initialisiert auf 1).
- **Fehlersuche**: Ein schwacher Klassifikator wird trainiert (z. B. die erste Trennlinie). Er ist oft nur geringfügig besser als zufälliges Raten.
- **Gewichts-Update**:
  - **Falsch klassifizierte Punkte**: Erhalten ein **höheres Gewicht**, damit der nächste Klassifikator sich stärker auf sie konzentriert.
  - **Richtig klassifizierte Punkte**: Erhalten ein **niedrigeres Gewicht**.
- **Wiederholung**: Eine neue Linie wird gezogen, die versucht, die nun schwerer gewichteten Fehler des Vorgängers zu korrigieren.

Iteration 1:



Iteration 3:



9.9.1.1. Abbruchkriterien

- **Feste Anzahl**: Nach einer vordefinierten Anzahl von Durchläufen (X Iterationen).
- **Validierungsmetrik**: Wenn sich der Fehler auf den Validierungsdaten nicht mehr verbessert (Early Stopping), um Overfitting zu vermeiden.

9.10. Viola Jones - Face Detection

- erkennen ob es Gesichter hat
- aber nicht welches Gesicht

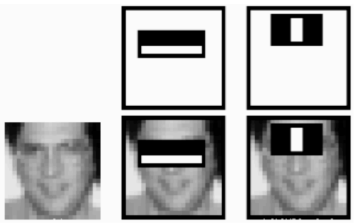
Hauptidee, eine Kombination aus:

- **Haar** Features

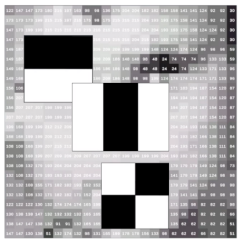
- **Boosting**
- **Cascaded Classifiers**

9.10.1. Haar Features

- **Einfache Filter**: Anstatt komplexer mathematischer Operationen werden einfache schwarz-weisse Masken (Rechtecke) verwendet.
- **Pixel-Differenz**: Das Feature berechnet den **Unterschied** zwischen der Summe der Grauwerte im schwarzen Bereich und der Summe im weissen Bereich.
  - **Hoher Kontrast**: Grosse Differenzwerte deuten auf ein Merkmal hin (z. B. Augenbrauen sind dunkler als die Stirn).
  - **Neutraler Bereich**: In Bereichen ohne Struktur heben sich schwarze und weisse Pixel gegenseitig auf (Differenz approx 0).

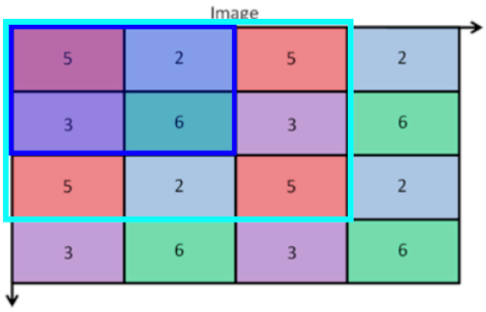


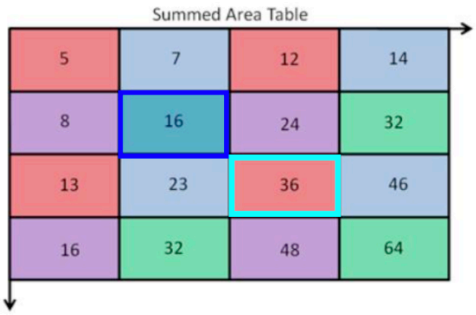
Features mit verschiedenen Grössen und Grössen and verschiedenen Stellen ausprobieren:



- **Low-Resolution**: Das Bild wird oft auf eine geringe Auflösung gebracht (Subsampling).
- **Sliding Window**: Ein Fenster von **24x24 Pixeln** gleitet über das Bild, um Features zu extrahieren.
- **Kombinationsvielfalt**: Allein in einem 24x24 Fenster gibt es ca. **160.000 mögliche Kombinationen** von Filtern (verschiedene Formen, Grössen und Positionen).
  - Berechnung muss daher effizient sein: **Integral Image**

9.10.1.1. Integral Image

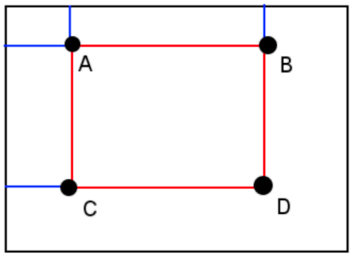




- Summe aller Pixelwerte oben und links, inklusiv des eigenen Pixelwerts
  - blau: $5 + 2 + 3 + 6 = 16$
  - türkis: $5 + 3 + 5 + 2 + 6 + 2 + 5 + 3 + 5 = 36$
- Das Integral Image wird **nur einmal** für das gesamte Bild berechnet.

Es ermöglicht die **Berechnung der Pixelsumme** eines **beliebigen rechteckigen Bereichs in konstanter Zeit**:

- unabhängig von der Grösse des Rechtecks



$$\text{Sum} = D - B - C + A$$

Beispiel

Original Image

0	1	3	5	2	7	10	7	10	9
3	7	6	9	3	8	1	8	5	8
1	11	0	7	13	2	14	2	13	1
14	3	2	7	1	0	9	7	2	12
1	5	15	3	6	6	5	1	10	6
8	1	2	6	7	3	2	11	0	15
7	7	6	0	9	5	10	3	8	1
12	5	6	10	11	3	6	7	9	1

Integral Image

0	1	4	9	11	18	28	35	45	54
3	11	20	34	39	54	65	80	95	112
4	23	32	53	71	88	113	130	158	176
18	40	51	79	98	115	149	173	203	233
19	46	72	103	128	151	190	215	255	291
27	55	83	120	152	178	219	255	295	346
34	69	103	140	181	212	263	302	350	402
46	86	126	173	225	259	316	362	419	472

$$\text{Sum} = D - B - C + A = 215 - 72 - 80 + 20 = 83$$

9.10.2. AdaBoost

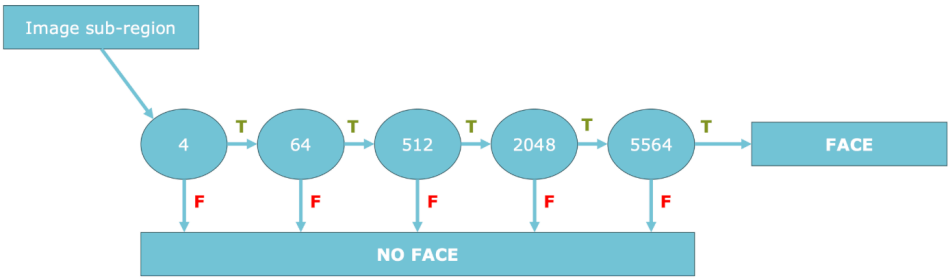
aus riesigen Menge nur die **relevantesten Merkmale** herausfiltern

Vorgehensweise: Jedes einzelne Haar-Feature wird als ein **Weak Classifier** (schwacher Klassifikator) betrachtet. AdaBoost wählt schrittweise diejenigen aus, die den Klassifizierungsfehler am stärksten reduzieren.

-> AdaBoost erkennt automatisch anatomische Konstanten in Gesichtern

9.10.3. Cascaded Classifiers

- Man verschwendet **keine Rechenzeit auf Hintergrundbereiche** (Sub-Windows), in denen offensichtlich kein Gesicht ist.
- Mehrere Classifier werden hintereinander geschaltet.
  - **Negative Match**: Erkennt ein früher Classifier „kein Gesicht“, wird der Bereich sofort abgelehnt und die Berechnung für dieses Fenster gestoppt (**Exit Computation**).
  - **Positive Match**: Nur wenn ein Classifier „Gesicht“ meldet, wird das Fenster an die nächste, komplexere Stufe weitergereicht.

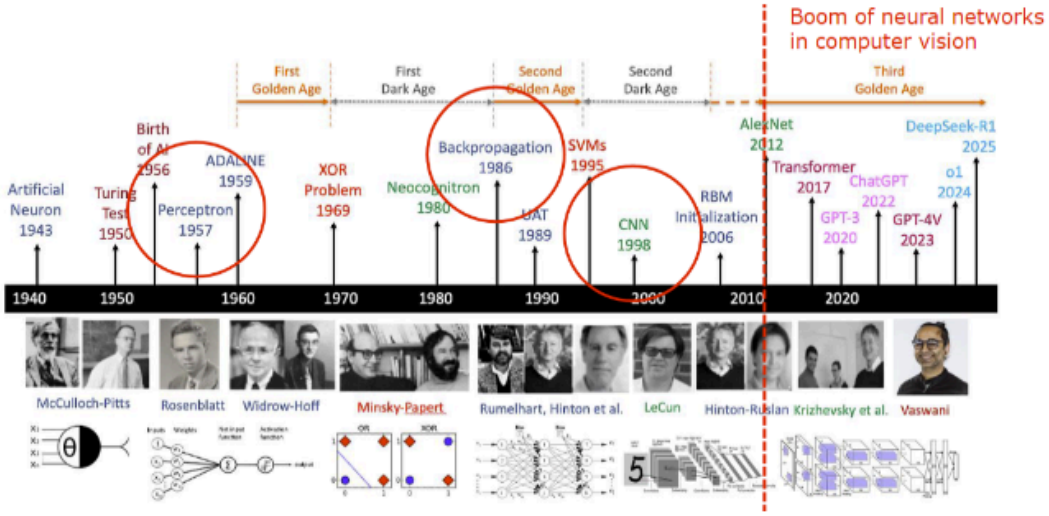


- Jede Stufe hat einen **Threshold**, der beim Training festgelegt wird.

10. Convolutional Neural Networks for Image Classification

10.1. History

2012: Boom of neutral networks in computer vision



Von 2000 bis 2010 hat sich die Hardware stark verbessert, dass Modelle auch trainiert werden konnten.

Ideen und Möglichkeiten waren schon vorher entwickelt aber aufgrund der zu schwachen Hardware konnte das nicht wirklich angewandt werden.

10.2. Classic ML Approaches

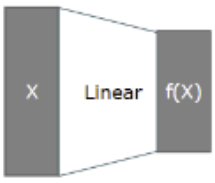
Problem mit linearer Klassifikation: Methoden sind nicht immer optimal und schwierig umzusetzen

10.3. Neural Networks

Eine einfache Methode, um zu lernen, wie man stark nichtlineare Funktionen darstellt.

10.3.1. Linear (Dense, Fully-Connected) Layers

- **Building Block** von neural networks
- auch **Lineare Funktion**
- Input mit n Dimensionen in einen Output mit d Dimensionen
- jedes Element vom Input ist mit jedem Element vom Output verbunden
  - deshalb ist es nicht optimal so Bilder zu analysieren



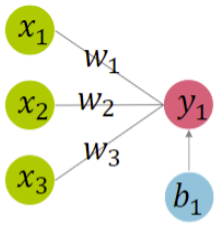
$$f(x) = W_x + b$$

Input vector
$xinR^n$

weight matrix
$WinR^{\{n \times d\}}$

bias vector
$binR^d$

Berechnung Wert Output y_1 mit allen Inputs x_1, x_2, x_3 und bias b_1:



10.3.1.1. PyTorch

```python
class torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None)
```

incoming data: $y = xA^T + b \rightarrow A^T = W$

Parameter	Typ	Beschreibung
`in_features`	`int`	Grösse jedes Eingabemusters (Eingangsdimension).
`out_features`	`int`	Grösse jedes Ausgabemusters (Ausgangsdimension).
`bias`	`bool`	Wenn `False`, lernt die Schicht keinen additiven Bias (Standard: `True`).
`device`	`torch.device`	Das Gerät, auf dem die Parameter gespeichert werden (z. B. ‚cpu' oder ‚cuda').
`dtype`	`torch.dtype`	Der Datentyp der Parameter (z. B. `torch.float32`).

`bias=True`: default (es gibt aber Fälle in dem man den bias (Offset) nicht haben möchte, z.B. bei Feature-learning, z.B. Blurring eines Bildes)

Beispiel:

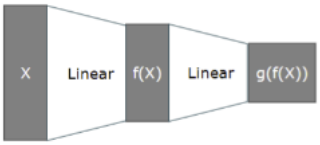
```python
import torch
import torch.nn as nn

# Definition der linearen Schicht: 20 Eingangsmerkmale, 30 Ausgangsmerkmale
m = nn.Linear(20, 30)

# Erstellung eines zufälligen Input-Tensors (Batch-Größe 128, 20 Merkmale)
input = torch.randn(128, 20)

# Anwendung der Schicht auf den Input
output = m(input)
```

```
# Ausgabe der resultierenden Größe
print(output.size())
# Erwartetes Ergebnis: torch.Size([128, 30])
```

10.3.2. Two Linear Layers

- **mehrere lineare layers**
- Input mit n Dimensionen in einen Output mit d Dimensionen
- **Vorteil**: noch kompliziertere Funktionen möglich

$$g(f(x)) = W_2(W_1 x + b_1) + b_2$$



Input vector
$xinR^n$

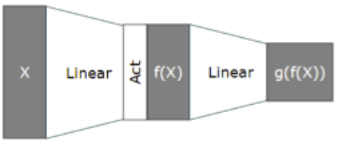
weight matrix
$W_1 inR^{\{n \times h\}}, W_2 inR^{\{h \times d\}}$

bias vector
$b_1 inR^h, b_2 inR^d$

10.3.2.1. Activation function

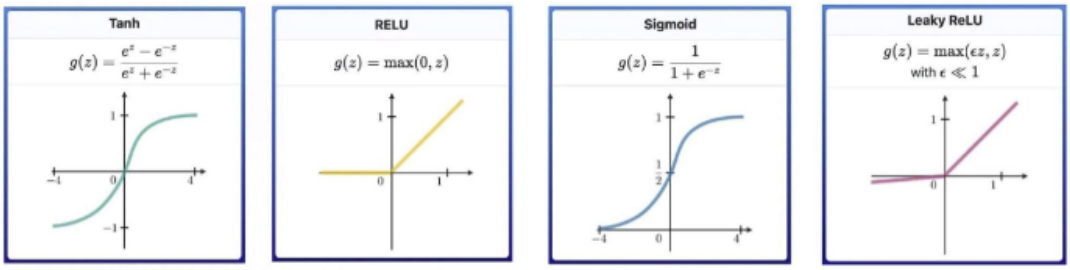
- bringen etwas non-lineares in die Formel
- **machen das Modell non-linear**

$$g(f(x)) = W_2 \ \mathrm{Act}(W_1 x + b_1) + b_2$$



10.3.2.1.1. Beispiele

(nicht abschliessend)



RELU

- linear für positive Inputs
- 0 bei negativen Inputs
- kann nicht abgeleitet werden / kein Gradient

Leaky ReLU

- kann während Training auch negative Werte zu haben

- verglichen zu RELU

10.3.2.1.2. PyTorch

Rectified Linear Unit Funktion (ReLu):

```python
class torch.nn.ReLU(inplace=False)
```

LeakyReLU-Funktion:

```python
class torch.nn.LeakyReLU(negative_slope=0.01, inplace=False)
```

Tanh:

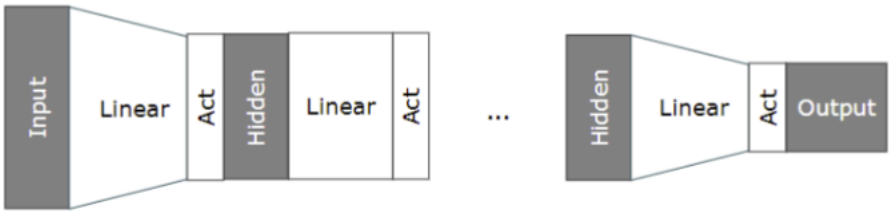
```python
class torch.nn.Tanh(*args, *kwargs)
```

Sigmoid:

```python
class torch.nn.Sigmoid(*args, *kwargs)
```

10.3.2.2. Multi Layer Perceptron (MLP)

- mehrere Lineare Layers nacheinander mit Activation Functions
- **Anwendung**: für Klassifikation



10.3.2.2.1. MLP: Applications

Problem:

- spatial structur wird zerstört
  - z.B. Gesichtserkennung: das Gesicht muss der Struktur entsprechen
- Lernbare Parameter explodieren bei grossen Bildern

→ Lösung Convolution

10.4. Convolutional Neural Networks (CNNs)

Meist implementiert als Korrelation

- muss bei der Anwendung nicht immer geflippt werden

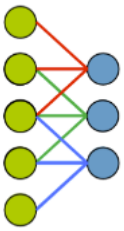
Wieso kann man einfach Korrelation anstatt Convolution verwenden?

- Flipping kann bereits beim Trainieren trainiert werden
- braucht keinen extra Step

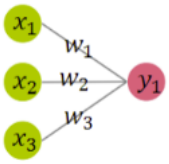
Vorteile

- Spatial Structure bleiben bestehen
- Parameter Anzahl bleibt konstant
- Nicht jeder Input braucht eine Connection mit jedem Output
- Receptive Field

Output ist nur abhängig von einem Teil vom Input



Berechnung Output y_1 mit Inputs x_1, x_2, x_3:



10.4.1. CNN vs. MLP

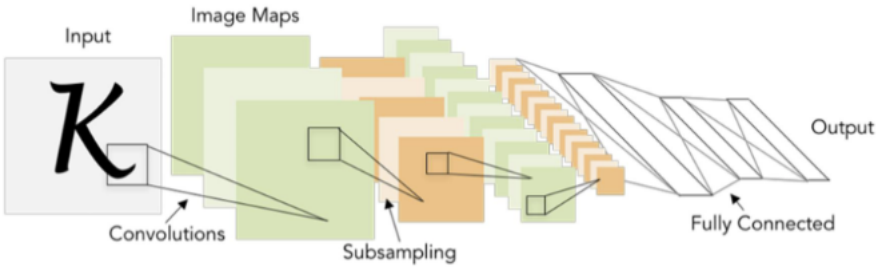
CNN

- lose Verbindungen (sparse connections)
- Beispielbild: 3 gemeinsame Parameter
- hat rezeptives Feld

MLP

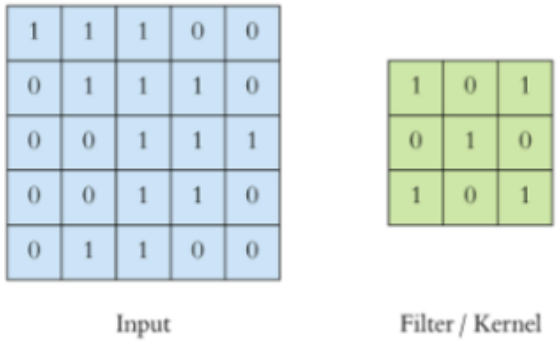
- vollständige Verbindungen
- Beispielbild: 15 Parameters
- jeder Output ist abhängig von allen Inputs

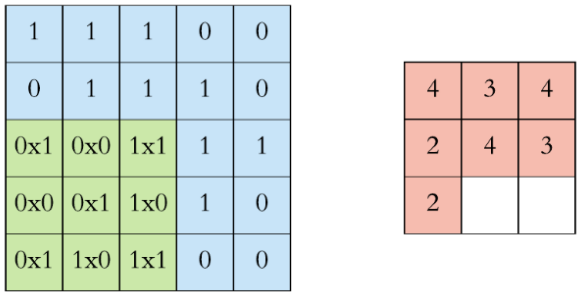
10.4.2. Typischer Aufbau von CNN

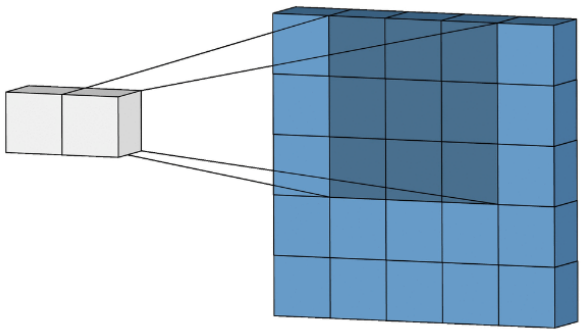


10.4.3. Convolutional Layer

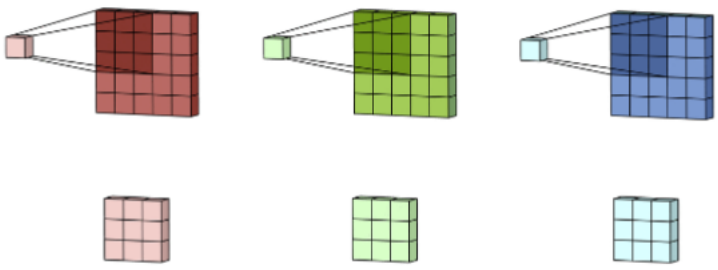
- Erhaltung der räumlichen Struktur in Bildern
- Filter: dunkelblau
- Output wird kleiner
  - Filtern kann für die Randpixel die Pixel nicht zentrieren







Mehrkanalbilder haben für jeden Kanal einen anderen Kernel:



10.4.3.1. Dimensionen

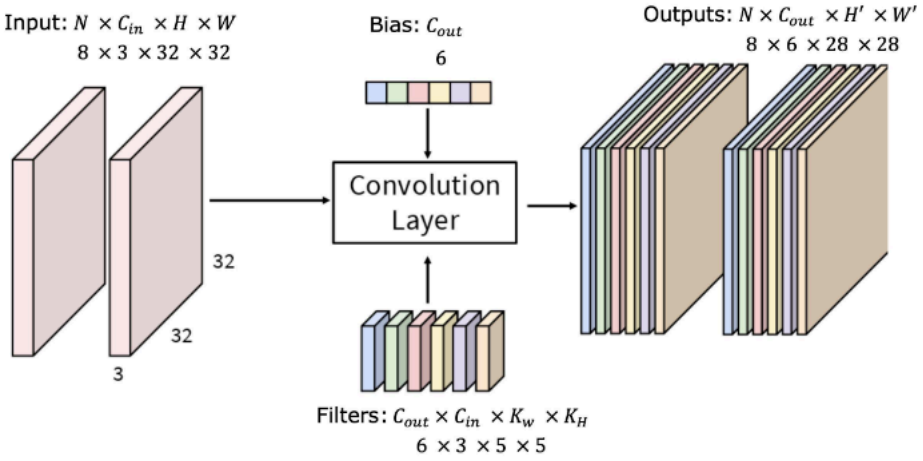
Input: $N \times C_{\in} \times H \times W$

- N: badge size
- $C_{\in}$: Input Channel (3 bei RGB Bild)
- $H \times W$: Grösse

Filters: $C_{\text{out}} \times C_{\in} \times K_W \times K_H$
- C_{out}: frei wählbar
  - definiert Anzahl Dimensionen / Filter des Outputs
- $K_W \times K_H$: Kernel Grösse (Breite x Höhe)

Outputs: $N \times C_{\text{out}} \times H' \times W'$
- $H' \times W'$: ohne padding wird der Output kleiner

Bias: C_{out}



Idee von Neural Networks: Filter C_{out} muss nicht definiert werden, wird gelernt, z.B. man definiert einfach Sobel für die edge detection

10.4.3.2. PyTorch

```python
class torch.nn.Conv2d(
    in_channels,
    out_channels,
    kernel_size,
    stride=1,
    padding=0,
    dilation=1,
    groups=1,
    bias=True,
    padding_mode='zeros',
    device=None,
    dtype=None
)
```

2D Kreuzkorrelation: $\text{out}\left(N_i, C_{\text{out}_j}\right) = \text{bias}(C_{\text{out}_j} + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}\left(C_{\text{out}_j}, k\right) \star \text{input}(N_i, k)$

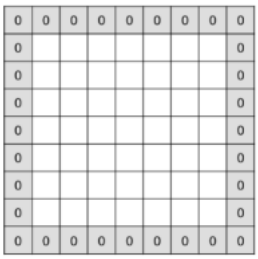
- $\boldsymbol{N}$: Batch-Größe
- $\boldsymbol{C}$: Anzahl der Kanäle (Channels)
- $\boldsymbol{H}$: Höhe der Bildebene in Pixeln
- $\boldsymbol{W}$: Breite der Bildebene in Pixeln
- $*$: 2D Kreuzkorrelations-Operator

```python
import torch
import torch.nn as nn

# Mit quadratischen Kernels und gleichem Stride
```

```python
m = nn.Conv2d(16, 33, 3, stride=2)

# Nicht-quadratische Kernels, ungleicher Stride und mit Padding
m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))

# Beispiel-Input: (Batch=20, Channels=16, Height=50, Width=100)
input = torch.randn(20, 16, 50, 100)
output = m(input)

print(output.size())
```

10.4.4. Padding

- damit das Output Bild nicht mit jedem Layer kleiner wird
- ein Padding (Zahlenrahme) um das Bild herum → meistens 0, kann aber auch eine andere Zahl sein



Allgemeine Regeln:

- Input: W
- Filter: K
- Padding: P
- Output: $W - K + 1 + 2P$

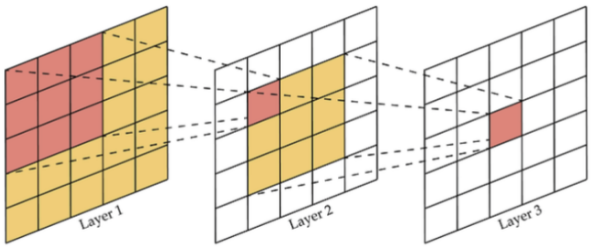
Berechnung Padding:

$$P = \frac{K - 1}{2}$$

10.4.5. Receptive field

Der Bereich des Eingabebildes, den ein einzelnes Neuron einer Schicht „sieht“ (Kontext für Vorhersagen).

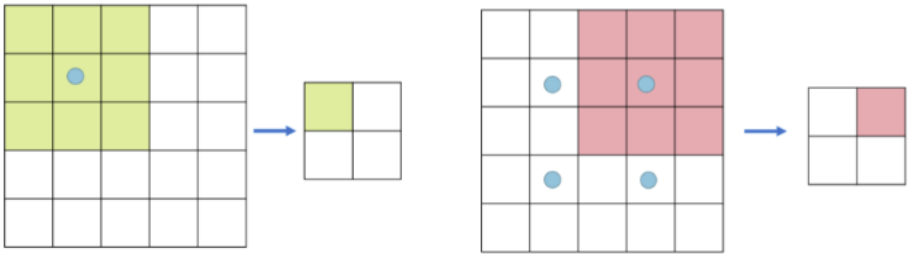
Durch das Stapeln (**Stacking**) mehrerer Layer mit kleinen Filtern (z. B. 3x3) vergrössert sich das Receptive Field in die Tiefe.



Parametereinsparung: Mehrere kleine Filter hintereinander (z. B. zwei 3x3) haben ein ähnlich grosses Receptive Field wie ein einzelner grosser Filter (z. B. 5x5), benötigen aber deutlich **weniger Parameter**.

10.4.6. Downsampling - Strided convolutions

- Datenmenge reduzieren
- strided definiert wie viele Pixel der Filter überspringt
- Input wird verkleinert (aufgrund des Überspringens) → als Resultat wird auch der Output kleiner

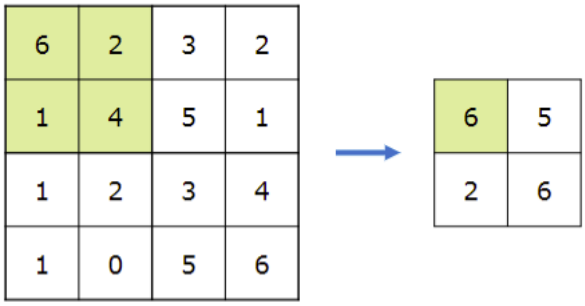


stride = 2

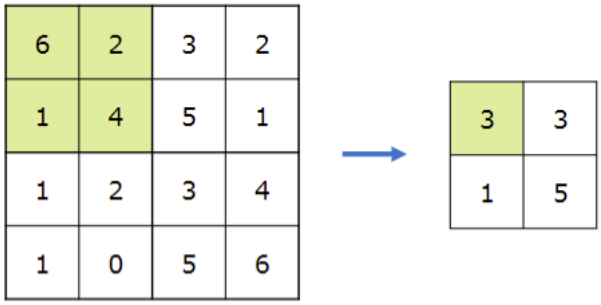
10.4.7. Downsampling - Pooling

- maximal oder Durchschnittswert werden für den output übernommen
- stride und pooling können kombiniert werden

Max Pooling: maximalen Wert nehmen



Average pooling: Mittelwert nehmen



10.4.7.1. PyTorch

```python
class torch.nn.MaxPool2d(
    kernel_size,
    stride=None,
    padding=0,
    dilation=1,
    return_indices=False,
    ceil_mode=False
)
```

$$\text{out}(N_i, C_j, h, w) = \max_{\{m=0,\cdot,\text{ kH-1}\}} \max_{\{n=0,\cdot,\text{ kW-1}\}} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

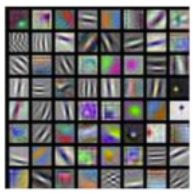
- $\mathbf{kH}, \mathbf{kW}$: Höhe und Breite des Pooling-Fensters (`kernel_size`).
- $\boldsymbol{N}, \boldsymbol{C}, \boldsymbol{H}, \boldsymbol{W}$: Batch-Größe, Kanäle, Höhe und Breite.

```python
import torch
import torch.nn as nn

# Beispiel 1: Pooling mit quadratischem Fenster der Größe 3 und Stride 2
m = nn.MaxPool2d(3, stride=2)

# Beispiel 2: Pooling mit nicht-quadratischem Fenster
m = nn.MaxPool2d((3, 2), stride=(2, 1))

# Beispiel-Input: (Batch=20, Channels=16, Height=50, Width=32)
input = torch.randn(20, 16, 50, 32)
output = m(input)

print(output.size())
```

10.4.8. Wieso convolutional layers?

Von Daten lernen abhängig vom Datensatz

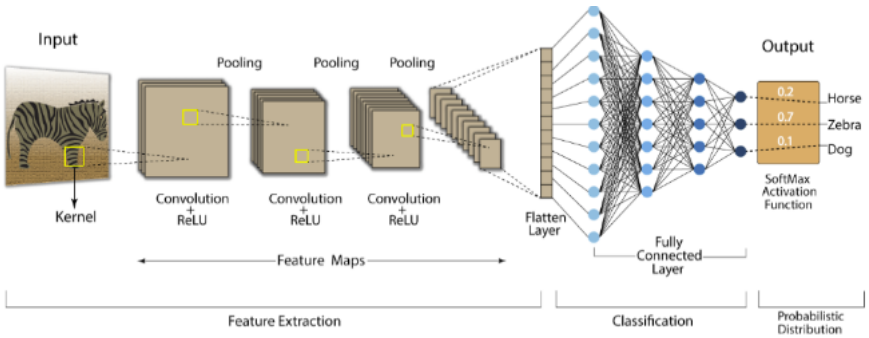
Erste Layer sind meistens wie Edge-Filter



Tiefer im Netzwerk, gibt es kompliziertere / spezifischere Filter

- Könnte von Hand nicht gemacht werden
- z.b Augen
- Buchstaben

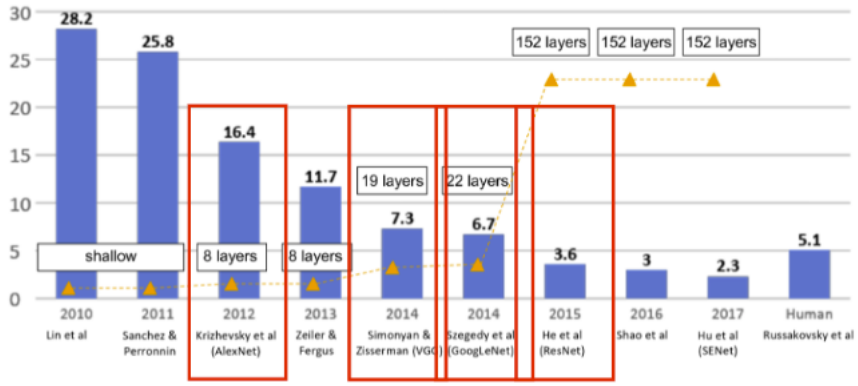
10.4.9. CNNs - Summary



10.5. CNN Architekturen

Tiefe Netzwerke: Tiefere Netzwerke erlauben es immer detaillierte Dingen zu lernen (z.B. Layer 1: Kantenerkennung, mittlerer Layer erkennt Formen, letzter Layer erkennt Objekte wie Autos, Gesichter)

Meilensteine:



10.5.0.1. ImageNet

Image Net ist ein Benchmark

Image Net schaut Top 5 an

- 16 % Fehler bei AlexNet

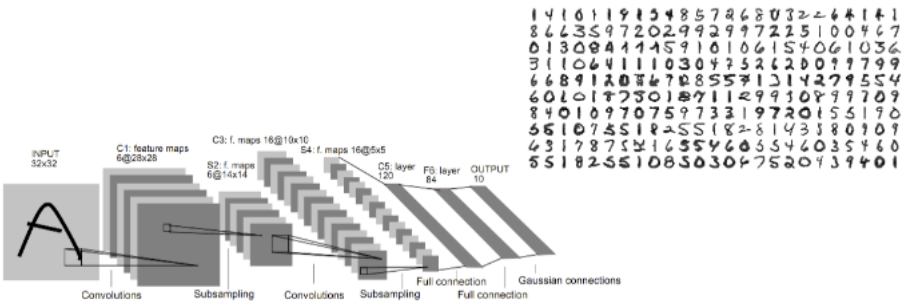
Top 1: input dog → 1. cat, 2. dog, 3. auto → FALSE

Top 2: input dog → 1. cat, 2. dog, 3. auto → TRUE

schaut die höchsten Wahrscheinlichkeiten an

10.5.1. LeNet (1998)

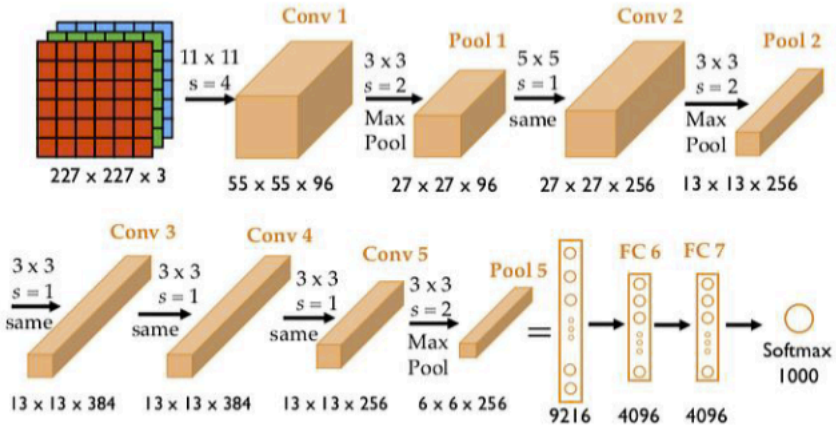
- 1998
- Handgeschriebene Digit Recognition mit Back-Propagation Network, Le Cun et al.
- erste Convolutional network
- 2 conv layers, 2 pooling (averaging) layers
- Problem
  - nicht skalierbar
  - keine Hardware zum trainieren



10.5.2. AlexNet (2012)

- 2012
- deep learning
- 5 Convolution Layers
- max-pool layers
- 3 fully connected layers am Ende
- RELU activation
- dropout
- 60 Millionen Parameter

- GPU wurden trainiert mit 3GB Memory → das hat das Feld von KI deutlich verändert

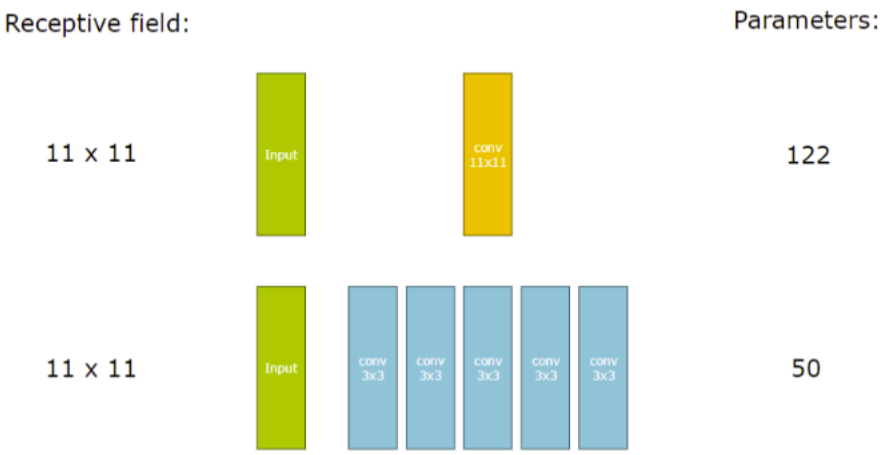


10.5.3. VGG (2014)

- 2014
- Netzwerk ist zweimal so tief wie AlexNet
- verwendet kleinen 3 x 3 Filter und pooling
- Effekt von depth of networks

smaller filters: parameters werden kleiner

Vergleich:

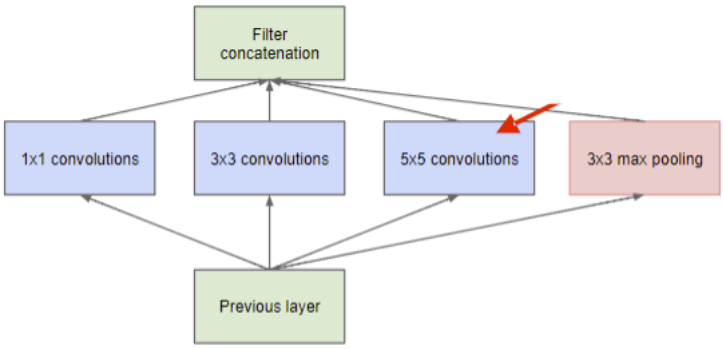


Problem:

- Netzwerke können nicht unendlich tief werden
- Gradienten werden irgendwann sehr klein
- tiefe Modelle werden immer schwieriger zu trainieren

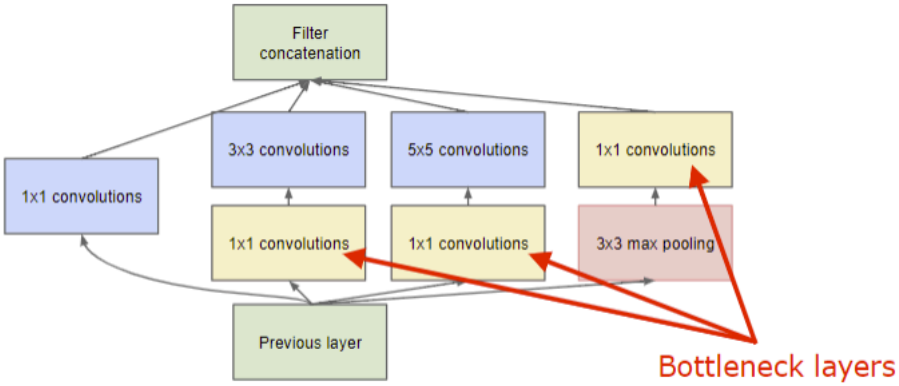
10.5.4. GoogLeNet - Inception Architecture (2014)

- Parameter sparen, ohne dass die Netzwerke immer tiefer werden
- Netzwerke gehen in Breite



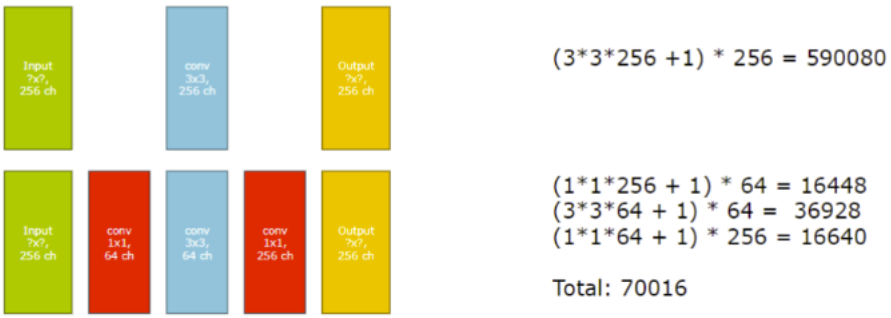
Aber, grössere Filter wieder mehr Parameter

Trick: Bottleneck layer



10.5.4.1. Bottleneck layers

- 1 x 1 convolutional layer
- reduziert die Kanäle von einer höheren Anzahl
- oder erhöht die Kanäle von einer geringeren Anzahl
- damit der Output Kanal gleich bleibt

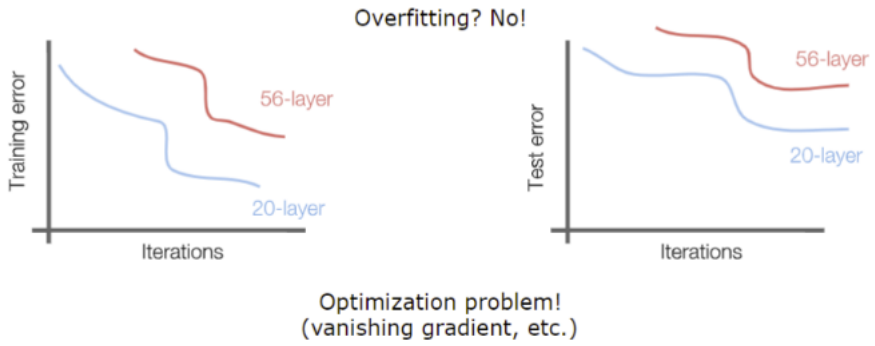


Berechnung: conv :}($\{\text{Höhe}\} \times \text{Breite} \times \text{Input} + 1) \times \text{Output}$

- wird für jeden Layer durchgeführt

10.5.5. ResNet (2015) - Residual Networks

- Früher waren tiefere Modelle (z. B. 56 Layer) schlechter als flachere (z. B. 20 Layer) – sowohl beim Test als auch beim Trainingsfehler.
- **Kein Overfitting**: Da auch der Trainingsfehler beim tieferen Modell höher liegt, ist es kein Overfitting, sondern ein **Optimierungsproblem**

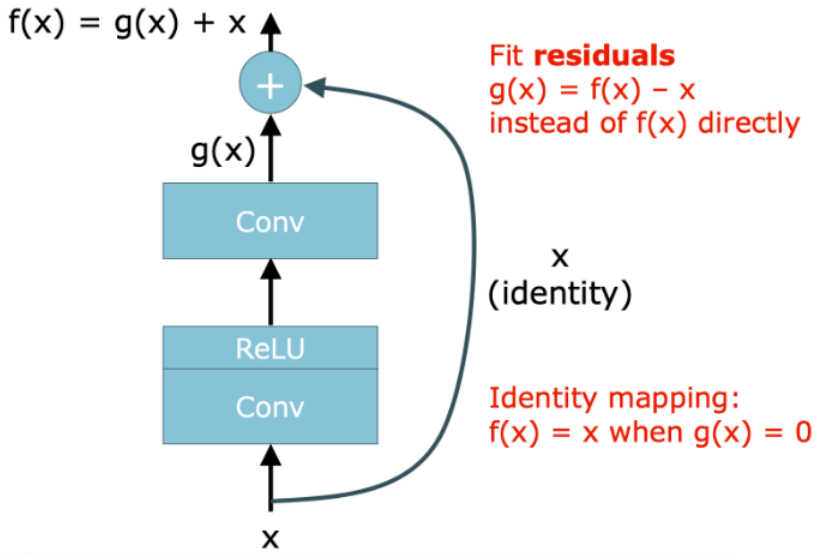


- **Vanishing Gradient**: Bei sehr tiefen Netzwerken wird der Gradient (das Lernsignal) beim Zurückfliessen durch die Schichten immer kleiner, bis er gegen Null geht – das Netzwerk hört auf zu lernen.

Lösung: **Identity Function** / Residual Connections
- Wenn eine Schicht das Ergebnis schlechter machen würde, kann diese Übersprungen werden
- in einem normalen Netzwerk muss jede Schicht eine neue Abbildung lernen
- bei ResNet muss nur noch das Residum (Rest) gelernt werden
- kann mit bottleneck layers kombiniert werden

$$f(x) = g(x) + x$$

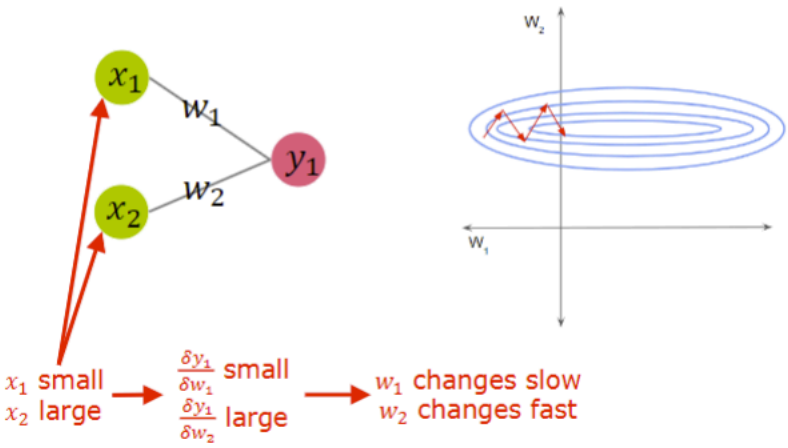
- x: ursprüngliche Input (identity)
  - wird einfach an die nächste Schicht weitergereicht
- $g(x)$: was die Schichten tatsächlich lernen müssen
- wenn $g(x)$ keine Verbesserung bringt, gilt $g(x) = 0$

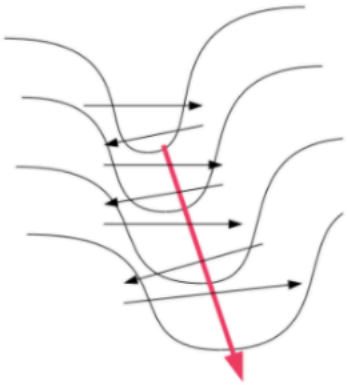


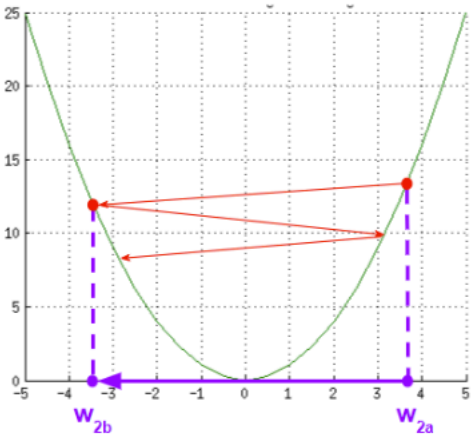
Problem: immer noch Vanished Gradients

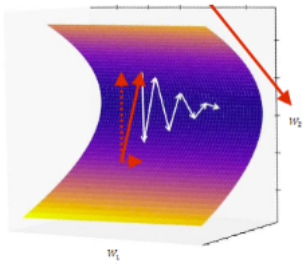
10.5.5.1. Batch Normalization

Keine Normalisierung des Inputs:
- aufgrund des Gradienten (gross / klein) wird nicht der direkteste Weg abgearbeitet



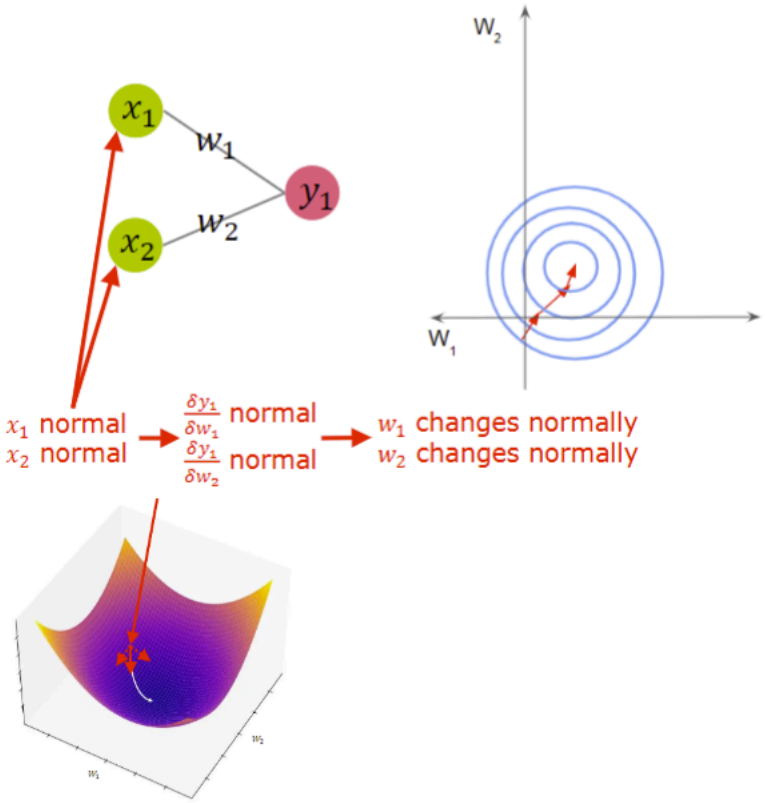






- ein Gewicht hat grosser Gradient → beginnt zu bouncen von einer zur anderen Seite
- ein anderes Gewicht hat ein kleiner Gradient → geht in die richtige Richtung aber sehr langsam

Normalisierung:
- konvergiert schneller
- direktere Weg wird abgearbeitet



10.5.5.1.1. PyTorch

```python
class torch.nn.BatchNorm2d(
    num_features,
    eps=1e-05,
    momentum=0.1,
    affine=True,
    track_running_stats=True,
    device=None,
    dtype=None
)
```

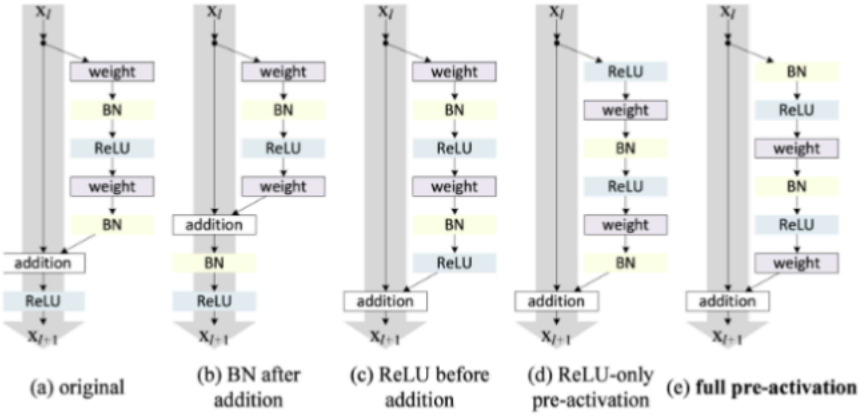
`num_features` : Anzahl Kanäle im convolution layer

$$y = \frac{x - E[x]}{\sqrt{\mathrm{Var}[x] + \varepsilon}} * \gamma + \beta$$

- $\boldsymbol{E[x]}$: Erwartungswert (Mittelwert) des Mini-Batches.
- $\mathbf{Var}[\boldsymbol{x}]$: Varianz des Mini-Batches.
- ε (`eps`): Ein kleiner Wert im Nenner zur numerischen Stabilität.
- γ: Lernbarer Skalierungsfaktor (Scale).
- β: Lernbarer Verschiebungfaktor (Shift/Bias).

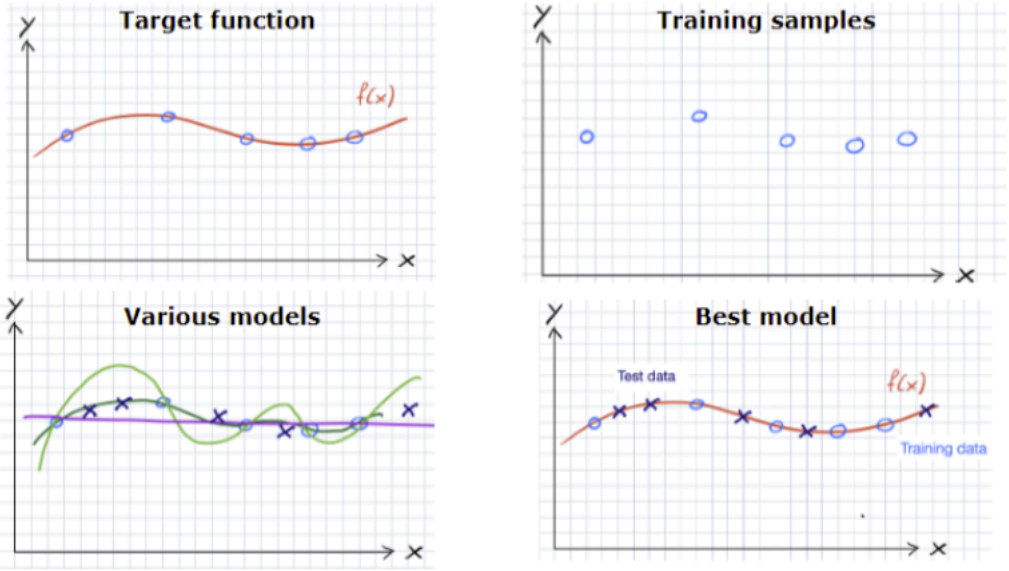
10.5.5.1.2. Residual Connections and Batch Normalization

case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**



10.6. Training Deep Networks

Ziel: Funktion finden, die die Daten am besten repräsentiert



10.6.1. Errors

Trainingsdaten

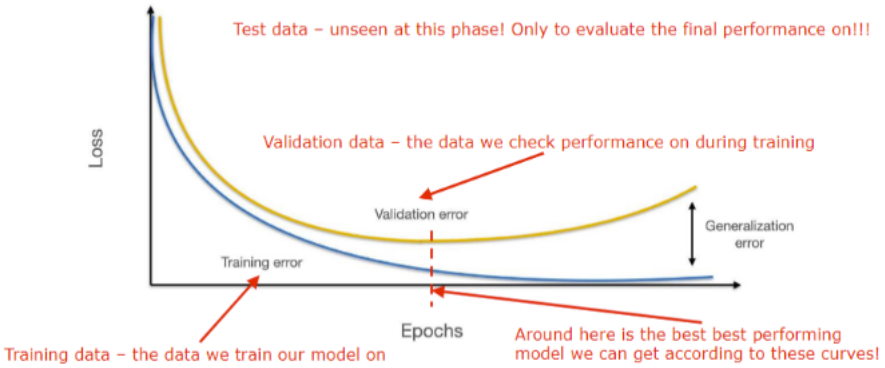
- Daten, mit denen das Model trainiert

Validierungsdaten

- Daten, die die Performance während dem Training überprüfen

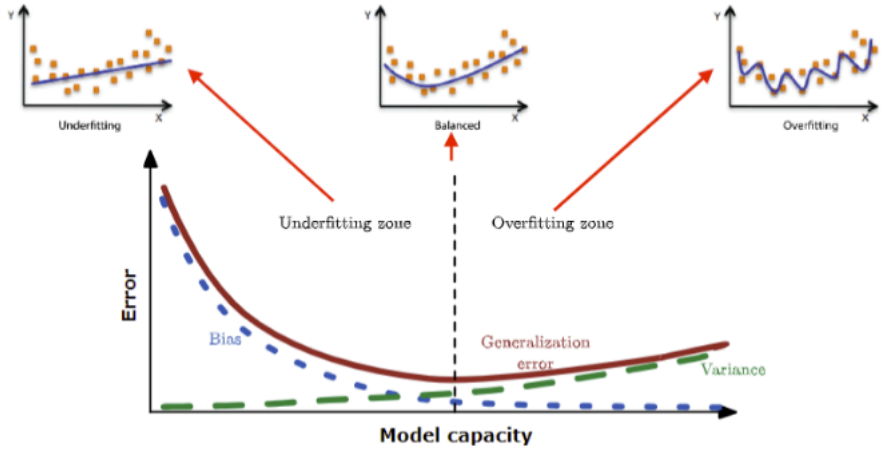
Testdaten

- nur um die finale Performance zu validieren
- darf nicht zum Training verwendet werden



Länger zu trainieren bedeutet nicht, dass das Model auch immer besser wird.

10.6.2. Optimal Capacity



Underfitting: Model ist nicht komplex genug, um die Daten zu lernen

Balanced: Model lernt gut genug, kann mit neuen Daten umgehen

Overfitting: hat die Daten „perfekt“ gelernt, wird aber bei neuen Daten scheitern (Model hat die Daten auswendig gelernt)

- Lösung: es ist nicht wichtig, alles perfekt zu lernen. In ML werden Einschränkungen eingebaut, damit sich das Model auf die wichtigsten Muster konzentriert

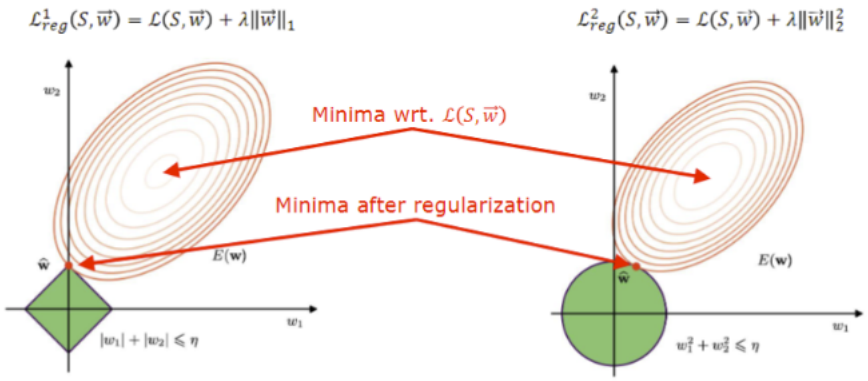
10.6.3. Regularization

L1 Regularization:

- setzt einige Parameter auf Null
- Summe von absoluten Werten
- Graphisch: Viereck
- Minimum: ist genau auf einem Ecken, somit Null

L2 Regularization:

- setzt einige Parameter auf kleine Werte
- Summe von quadrierten Werten
- Graphisch: Kreis
- Minimum: schiebt sich Richtung Null
  - je grösser λ ist, desto näher an Null



10.6.3.1. L2 Regularization

- Überanpassung vermeiden
- grosse Parameter werden zugunsten kleineren Parameter bestraft

$$\mathcal{L}_{\text{reg}}(S, \vec{w}) = \mathcal{L}(S, \vec{w}) + \lambda \, |\vec{w}|_2^2$$

- **λ (Lambda):** Regler für die Strenge. Je höher λ, desto härter die Bestrafung.
- **$\| \vec{w} \|_2^2$:** Quadrat der Gewichte

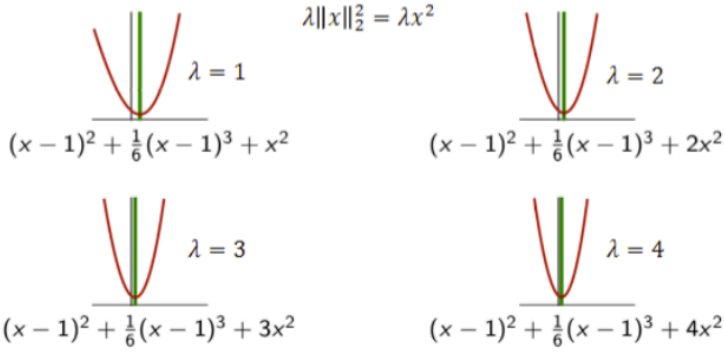
Effekt: Model wird folgendes tun:

1. Den Fehler in den Daten minimieren.
2. Die Gewichte so klein wie möglich halten.

10.6.3.1.1. Beispiel

$\mathcal{L}(x, \{x\}) = (x-1)^2 + \frac{1}{6}(x-1)^3$

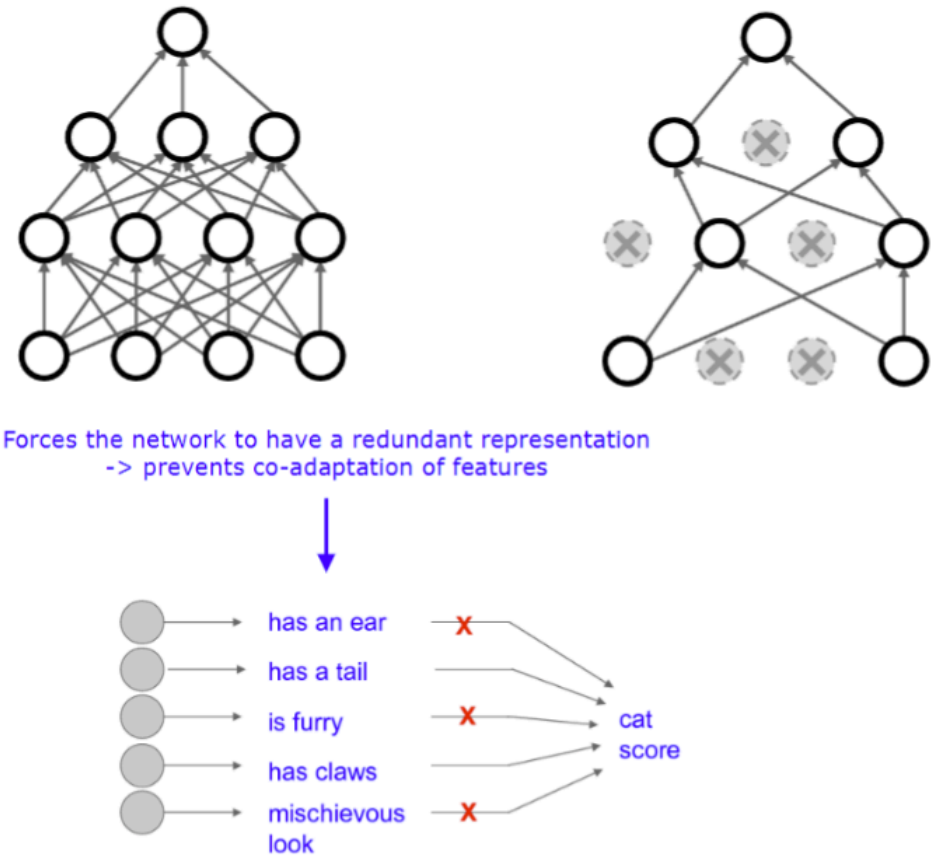
$\lambda\|x\|_2^2 = \lambda x^2$



- je Grösser λ, je stärker wird die rote Kurve Richtung Null gezogen
- **Ergebnis:** Die grüne Linie wandert immer weiter nach links zur **Null**. Das Modell wird „konservativer“.

10.6.3.2. Dropout

- Bei jedem Vorwärtsdurchlauf werden einige Neuron zufällig auf Null gesetzt.
- Intuitiv gesehen trainiert Dropout ein grosses Ensemble von Modellen, die sich Parameter teilen.
- Man geht davon aus, dass man Features hat, die Ähnliches beschreiben und man diese somit nicht benötigt.
- **Macht das Modell robuster und genereller**. (verhindert overfitting)



Während dem **Training**:
- Zufällige einigen **Input auf Null** setzen (ausschalten)
- Wahrscheinlichkeit ist p (Dropout ratio)
- **Effekt**: Netzwerk darf sich nicht auf bestimmte Merkmale verlassen, es muss die Vorhersage auch ohne die abgeschalteten Merkmale machen können

Während dem **Test**:
- **keine** Neuron mehr auf Null schalten
- **Problem**: Gewichte sind durch das Dropout zu laut
- **Lösung**: Gewichte mit $1 - p$ multiplizieren
- Alternative Lösung: Gewichte bereits während dem Training skalieren mit $\frac{1}{1-p}$
  - Standard in Pytorch oder TensorFlow

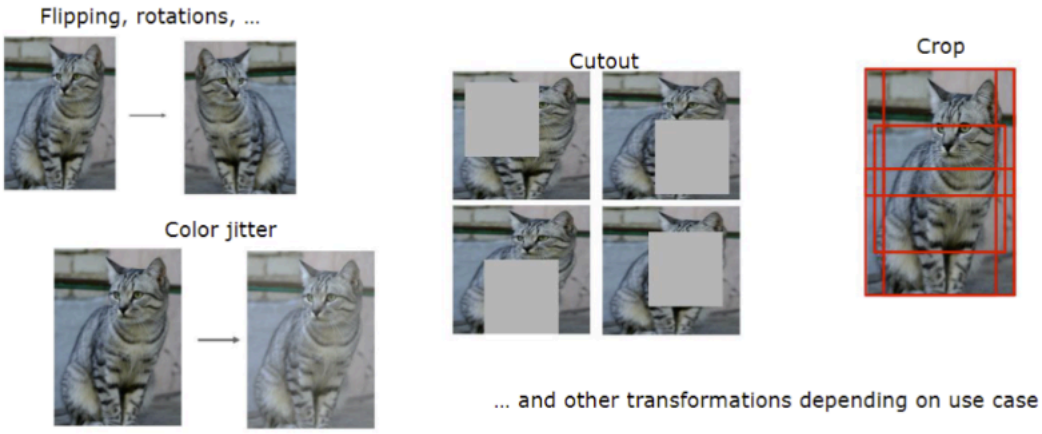
10.6.3.3. Data Augmentation

Anzahl von Training-Daten erhöhen mit verschiedenen Transformationen

→ macht das Modell **robuster**

nur auf Trainings-Daten anwenden

Transformationen:
- Flipping, Rotieren
- Farbanpassungen
- Ausschnitte
- Zuschneiden
- ...



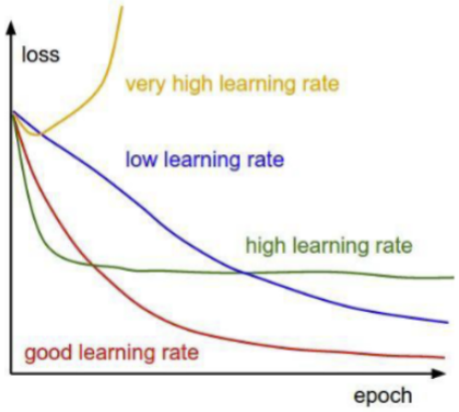
Validierungsdaten und Testdaten müssen immer gleich bleiben!!

10.6.4. Hyperparameter Optimization

Ziel: die besten Hyperparameter finden

Vorgehen:

- **Grob-Suche / coarse**
  - verschiedene Werte nur für ca. 1-5 epochs kurz ausprobieren
  - overfit provozieren
  - LR finden, der den Fehler minimiert
- **Fein-Suche / finer**
  - genauer und länger trainieren



- very high (gelb): Schritte sind zu gross, Fehler wird sehr gross
- high (grün): lernt am Anfang schnell, bleibt aber auf einem schlechten Niveau
- low (blau): Schritte sind zu klein, Modell braucht viele Epochen um an das Ziel zu kommen
- good (rot): Fehler sinkt schnell, und bleibt auf einem tiefen Niveau

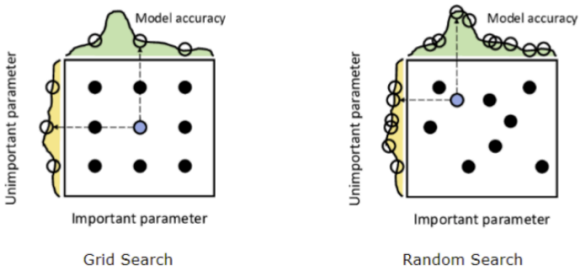
10.6.4.1. Grid vs. Random Search

Grid Search:

- vordefinierter Orte wo man sucht (selbst gewählt)
- kann dazu führen, dass man einen Ort mit sehr hohe Accuracy verpasst
- wird aber dennoch noch oft verwendet

Random Search:

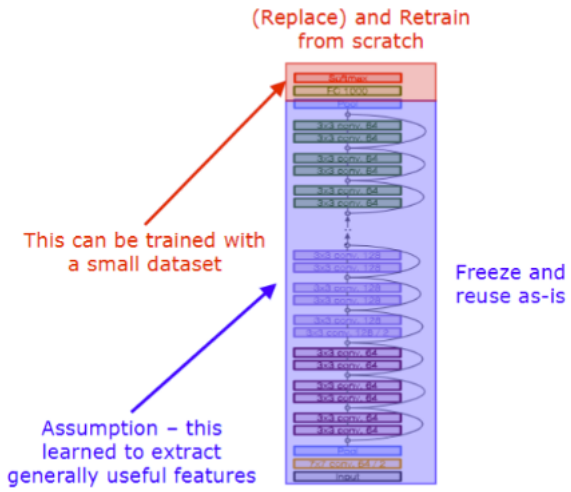
- Zufällige Orte wählen
- Wahrscheinlich höher einen guten Wert finden
- aufwändiger zum Berechnen



10.6.5. CNNs in Practice

10.6.5.1. Transfer Learning

- **Einfrieren (Freezing):** Ein grosser Teil des Netzwerks bleibt fix und wird nicht verändert (Reuse as-is).
- **Anpassung (Replace):** Der letzte Teil des Netzwerks (meist der Classification-Head/Softmax) wird ersetzt.
- **Training:** Nur die neuen oder die obersten Schichten werden mit dem kleinen, spezifischen Datensatz trainiert.
- **Effizienz:** Deutlich geringerer Rechenaufwand und weniger Datenbedarf als ein Training „from scratch“.



10.6.5.2. Fine Fine-tuning

- **Grundidee:** Nutzung von Modellen, die bereits auf riesigen Datensätzen (z. B. ImageNet mit 14 Mio. Bildern) trainiert wurden & gewisse Parameter „fine-tunen“
- **Vorteil:** Ermöglicht das Lösen komplexer Aufgaben auch mit sehr kleinen eigenen Datensätzen.
- **Voraussetzung:** Das vortrainierte Modell muss „generelle“ Features (Kanten, Formen, Texturen) gelernt haben, die auch für die neue Aufgabe nützlich sind.

10.6.5.2.1. torchvision

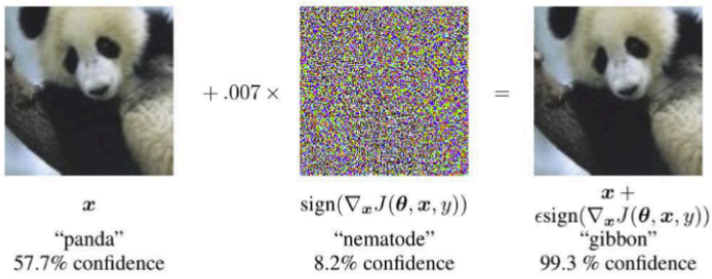
Model	`Acc@1`	`Acc@5`
VGG-16 with batch normalization	73.360	91.516
VGG-19 with batch normalization	74.218	91.842
ResNet-18	69.758	89.078
ResNet-34	73.314	91.420

ResNet-50	76.130	92.862
ResNet-101	77.374	93.546
ResNet-152	78.312	94.046

```python
from torchvision.io import decode_image
from torchvision.models import resnet50, ResNet50_Weights

img = decode_image("test/assets/encode_jpeg/grace_hopper_517x606.jpg")

# Step 1: Initialize model with the best available weights
weights = ResNet50_Weights.DEFAULT
model = resnet50(weights=weights)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = preprocess(img).unsqueeze(0)

# Step 4: Use the model and print the predicted category
prediction = model(batch).squeeze(0).softmax(0)
class_id = prediction.argmax().item()
score = prediction[class_id].item()
category_name = weights.meta["categories"][class_id]

print(f"{category_name}: {100 * score:.1f}%")
```

10.6.6. CNNs: Probleme

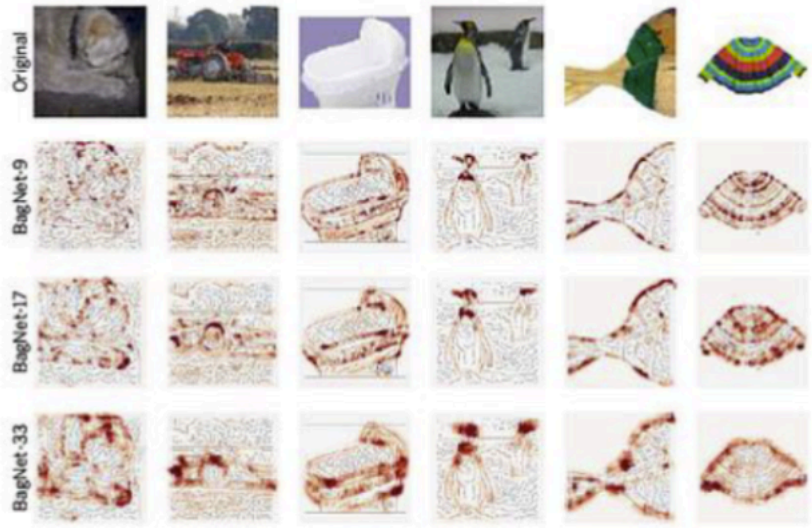
Kleine Änderung (von Auge nicht sichtbar) am Bild, kann zu ganz anderen Resultaten im Modle führen:



CNNs schauen zu sehr auf die **Struktur**, statt auf die Form:

- Katzenform mit Elephanthaut ergibt ein Elephant

CNNs sind ähnlich wie Bags of Words



11. Vision Transformer

11.1. Introduction

- Transformers wurde mit dem Paper „Attention is all you need“ bekannt
- aus dem Jahr 2017
- von Google und University of Toronto

Classic Deep Learning Landscape: eine Architektur pro community

Transformer werden heutzutage für alle möglichen Arten von neuronalen Netzen verwendet:

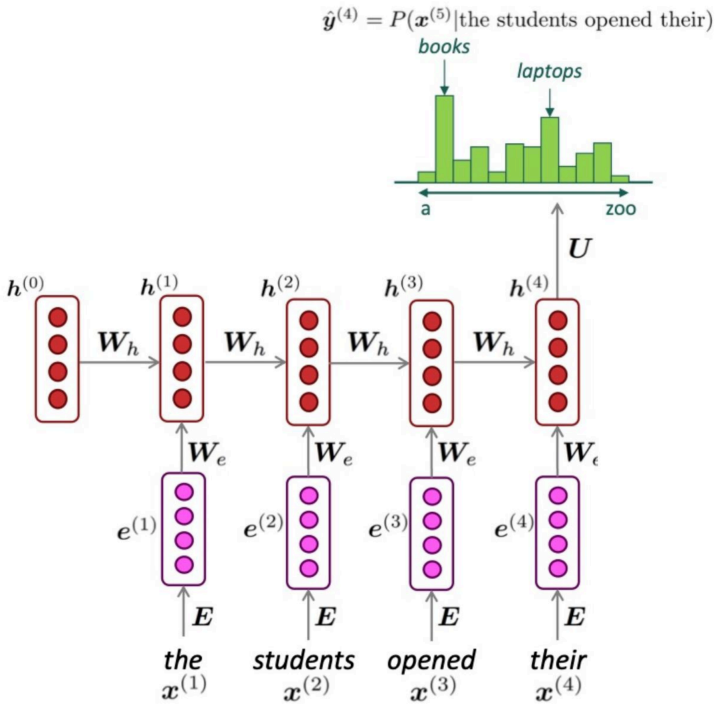
- Computer Vision
- Natural Language Processing
- Reinforcement Learning
- Speech
- Translation
- Graphs / Science

11.1.1. Attention Mechanismus

DEFINITION: Modellen ermöglichen, Zusammenhänge zwischen verschiedenen Modalitäten zu erlernen

11.2. Simple RNN Language Model

RNN: Recurrent neural networks



- **Zweck:** Vorhersage des jeweils nächsten Wortes in einer Sequenz (iteratives Lesen).
- **Textrepräsentation:** Umwandlung von Wörtern in Vektoren oder Matrizen.
- **Word Embeddings ($e^{(t)}$):** Transformation des Inputs $x(t)$ durch die Matrix E in einen dichten Vektorraum.
- **Hidden State ($h^{(t)}$):** Fungiert als „Speicher“ des Netzwerks für bereits gesehene Informationen.

Vorgehen:

- **Input:** Ein Wort nach dem anderen wird eingegeben.
- **Kontext:** Das RNN aktualisiert bei jedem Wort seinen internen Zustand (h).
- **Output:** Es gibt eine Wahrscheinlichkeitsverteilung für das nächste Wort aus.

Training und Logik

- Transformation ermöglicht Kombination von Vektoren unterschiedlicher Grösse (z. B. 24 und 32 Dimensionen).
- Bestimmung des wahrscheinlichsten nächsten Wortes aus einem Dictionary (z. B. „books“ oder „laptops“).
- Optimierung des Modells durch **Cross-Entropy Classification**.

11.2.1. Mathematische Funtionsweise

- **Zustandsaktualisierung:** Berechnung des neuen Hidden States durch:

$$h^{(t)} = \sigma\big(W_h h^{(t-1)} + W_e e^{(t)} + b_1\big)$$

- **Gewichtsmatrizen**
  - W_h: Filtert relevante Informationen aus dem vorherigen Speicher.
  - W_e: Transformiert das Word Embedding in die Dimension des Hidden States (Anpassung unterschiedlicher Vektorbereiche).
- **Output-Berechnung:** Erzeugung einer Wahrscheinlichkeitsverteilung ($\hat{y}^{(t)}$) über das gesamte Vokabular $|V|$ mittels Softmax-Funktion:

$$\hat{y}^{(t)} = \text{softmax}\big(U h^{(t)} + b_2\big)$$

- **Word Embeddings**

$$e^{(t)} = E x^{(t)}$$

$$x^{(t)} \in \mathbb{R}^{|V|}$$

11.2.2. Vorteile

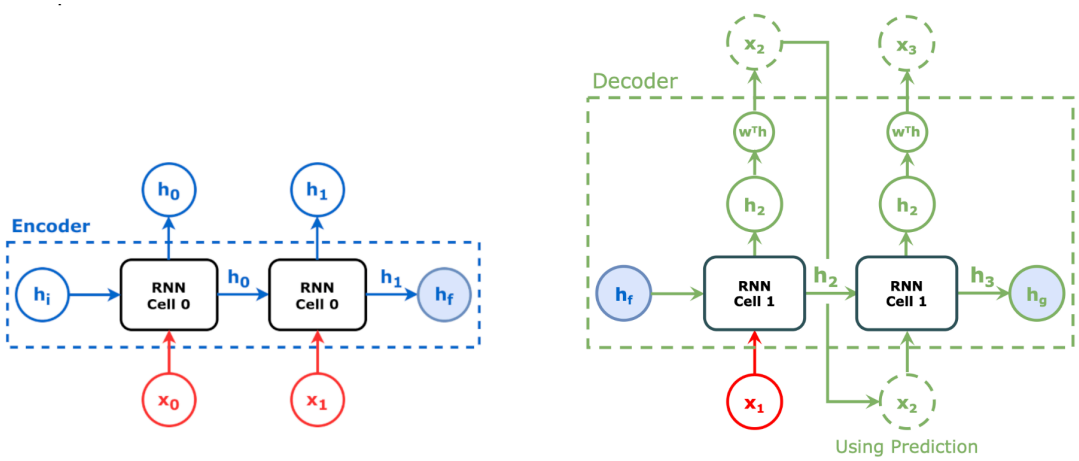
- **Variable Sequenzlänge**
  - Verarbeitung von beliebig langen Eingaben möglich
- **Konstante Modellgrösse**
  - Parameter (W_h, W_e, U) werden geteilt und wachsen nicht mit der Eingabelänge
- **Gedächnis**
  - Information aus frühen Schritten kann für spätere Berechnungen genutzt werden.

11.2.3. Nachteile

- **Sequentielle Berechnung**
  - Rechenschritte müssen nacheinender erfolgen
- **Fehlende Parallelisierung**
- **Eingeschränktes Gedächtnis** / Context Window Problem
  - Schwierigkeit, Informationen über weite Distanzen („viele Schritte zurück“) stabil zu halten.
  - Vanishing Gradient

11.3. Seq2Seq

- **Zweck**: Sequenz-zu-Sequenz-Generierung (z. B. Übersetzungen)
- **Architektur**: Kombination aus zwei RNNs (Encoder und Decoder)



Encoder:

- Input: Wort-Sequenz $(x_0, x_1, ...)$
- Prozess: Iterativer RNN-Durchlauf
- Output: Finaler Hidden State (h_f) (Vektor mit komprimierter Satzinformation)

Decoder

- Nutzt Encoder-Output (h_f) als initialen Hidden State
- Benötigt Start-Token $(< \text{START} >)$ als Erst-Eingabe
- Output aus Schritt t wird Input für Schritt $t + 1$
- Generierung eines End-Tokens $(< \text{END} >)$

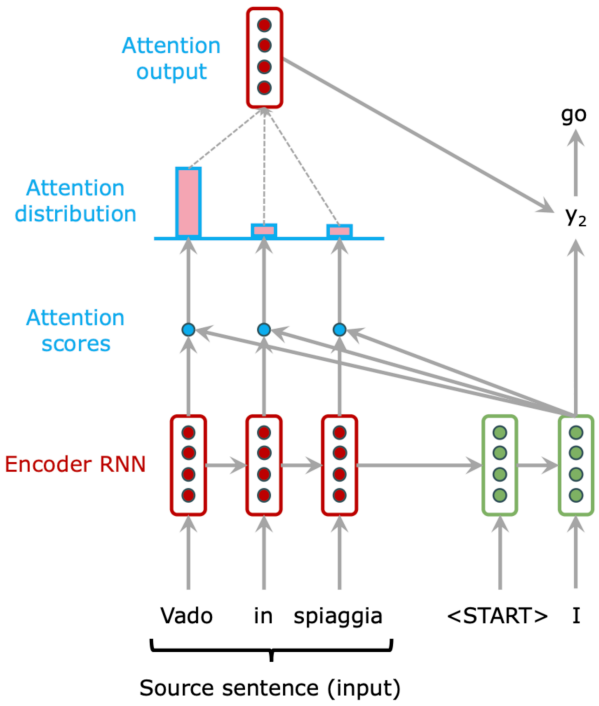
Schwachstelle: Information Bottleneck

- Gesamter Input-Satz muss in einem einzigen Vektor (h_f) Platz findenI
- Zu viele Details in zu kleinem Speicher („Bottleneck“)
- Sinkende Übersetzungsqualität bei langen Sätzen

Lösung **Attention Mechanismus**

11.3.1. Attention

- **Encoder-Zustände:** Liefert einen Vektor *pro Input-Wort*.
- **Scoring:** Vergleicht den aktuellen Decoder-Zustand mit allen Input-Wörtern.
- **Gewichtung:** Wandelt die Scores in Wahrscheinlichkeiten (0 bis 1) um.
- **Dynamischer Kontext:** Multipliziert die Input-Vektoren mit den Gewichten und addiert sie. Wichtige Wörter dominieren.
- **Output:** Nutzt diesen massgeschneiderten Kontextvektor zur Generierung des exakt nächsten Wortes.



11.3.1.1. Mathematik

Hidden State des Encoders: $h_1, ..., h_N \in \mathbb{R}^h$

Hidden State des Decoders zum Zeitpunkt t: $s_t \in \mathbb{R}^h$

Attention-Scores:

$$e^t = [s_t^T h_1, ..., s_t^T h_N] \in \mathbb{R}^N$$

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

Gewichtete Summe der Hidden-States des Encoder:

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

Verkettung der Attention-Ausgabe mit dem HIdden-State des Decoders:

$$[a_t; s_t] \in \mathbb{R}^{2h}$$

11.3.2. Score

Die Ausrichtungsfunktion a bewertet, wie gut die Eingaben um die Position j und die Ausgaben um die Positionen i übereinstimmen

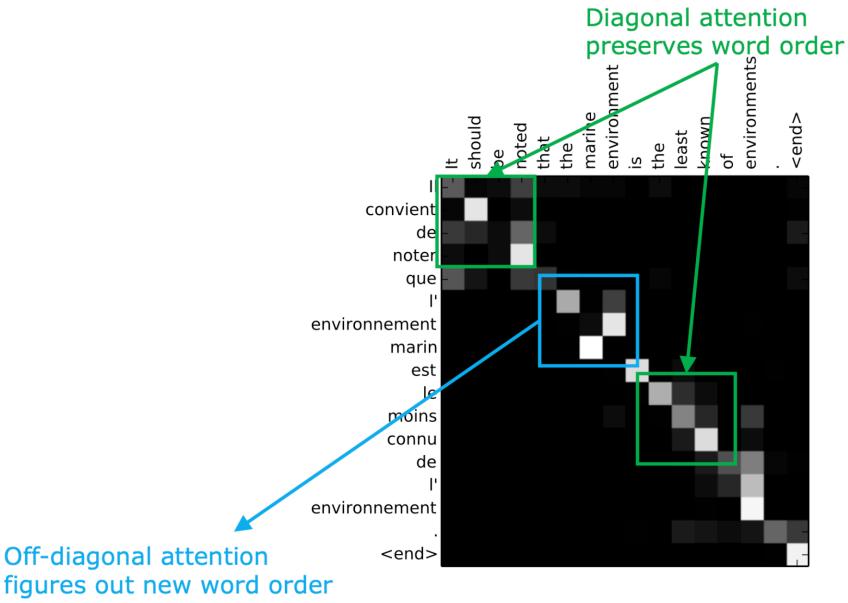
- parametrisiert als Feedforward-Neuralnetz
- gemeinsam mit den anderen Komponenten des Modells trainiert

$$\alpha_{ij} = \frac{e^{\text{score}(s_{t-1}, h_i)}}{\sum_{j=1}^{n} e^{\text{score}(s_{t-1}, h_j)}}$$

- s_{t_1}: Decoder hidden state
- h_i: Encoder hidden state

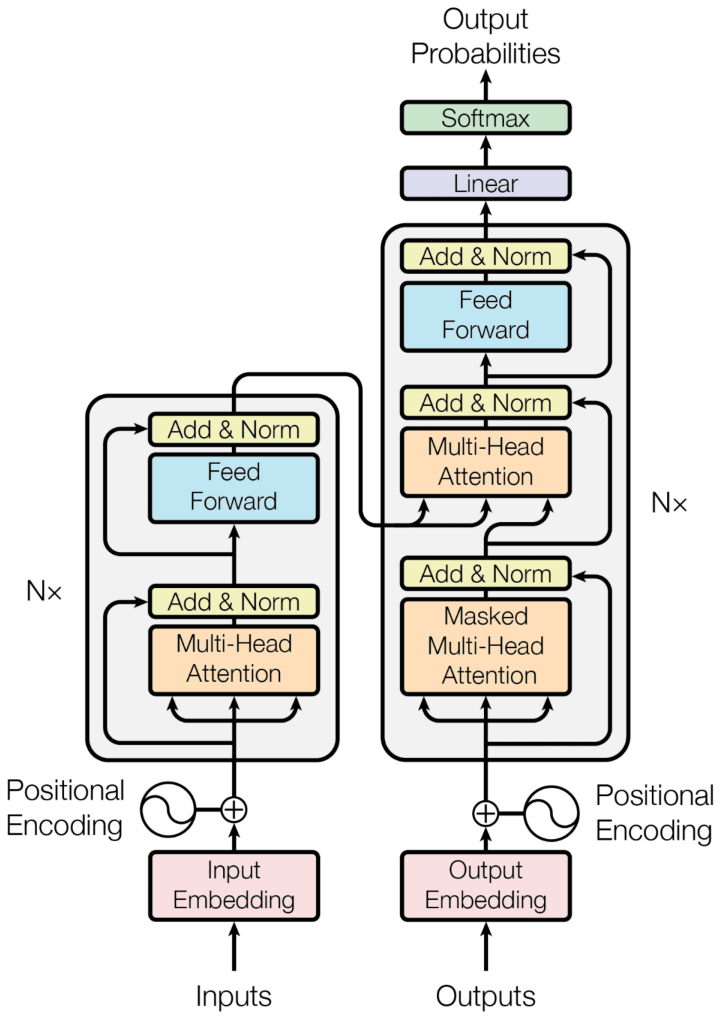
$$\text{score}(s_t, h_i) = v_a^T \tanh(W_a[s_t; h_i])$$

- rechts: Bewertungsfunktion parametrisiert als MLP



11.4. Transformer

- Encoder-Decoder model
- Bearbeiten sequenzielle Daten
- müssen NICHT den Input sequenziell abarbeiten
- können parallelisiert werden



Encoder:

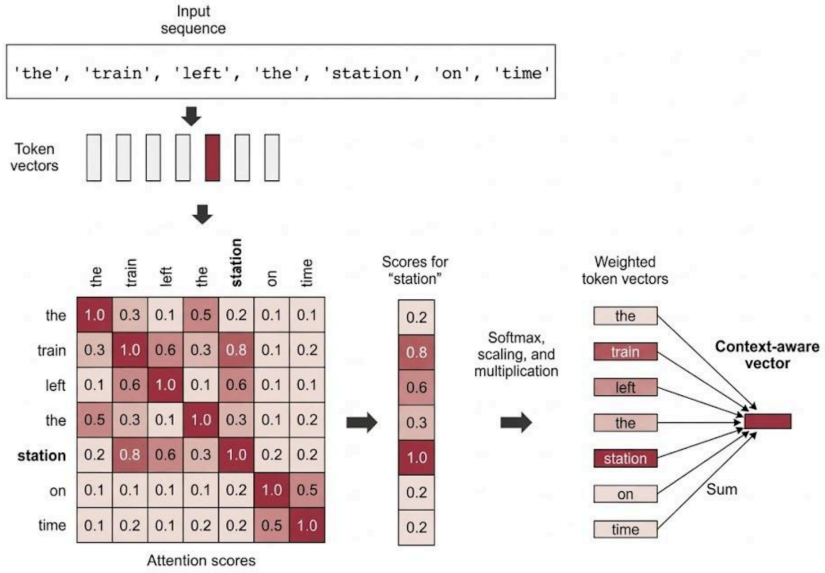
1. **Multi-Head Attention:** *Self-Attention* wird mehrfach gleichzeitig durchgeführt (aus verschiedenen „Blickwinkeln“)
2. **Add & Norm:** Ein Kontrollschritt, der hilft, die Daten stabil zu halten und Informationen aus vorherigen Schichten nicht zu vergessen (Residual Connections).
3. **Feed Forward:** Ein neuronales Netzwerk, das die gewonnenen Informationen pro Wort weiter verarbeitet.

11.4.1. Self-Attention

- Core Building Block eines Transformers
- Erlaubt jedem Wort im Satz, auf alle anderen Wörter Bezug zu nehmen.
- Hilft dabei, kontextbezogene Embeddings von Wörtern zu erstellen

Ablauf

- **Eingabe:** Ein Text wird in einzelne Bausteine (Wörter oder Wortteile, sogenannte **Tokens**) zerlegt.
- **Vektorisierung:** Jedes Token wird in eine Zahlenreihe (*Vektor*) übersetzt, die das Modell verarbeiten kann.
- **Scoring (Beziehungen messen):** Das Modell berechnet für **jedes** Wort, wie stark es sich auf **jedes andere** Wort im selben Satz bezieht.
- **Softmax (Normieren):** Diese Beziehungswerte werden mathematisch geglättet, sodass sie wie prozentuale Anteile funktionieren.
- **Gewichtung:** Die Vektoren aller Wörter werden mit diesen errechneten Prozentwerten multipliziert. Wichtige Kontext-Wörter erhalten ein hohes Gewicht, unwichtige ein niedriges.
- **Summierung:** Alle so gewichteten Vektoren werden zusammengezählt.
- **Ausgabe:** Das Resultat ist ein neuer, „schlauerer" Vektor. Er steht nicht mehr nur für das isolierte Wort, sondern für das Wort **inklusive seines genauen Bedeutungskontextes** in diesem Satz.



11.4.1.1. Vectorized / Calculation

1. Mit in X gestapelten Embeddings, berechne Queries, Keys und Values:

$$Q = XW^Q \quad K = XW^K \quad V = XW^V$$

2. Berechne Attention Scores zwischen Queries und Keys:

$$E = QK^T$$

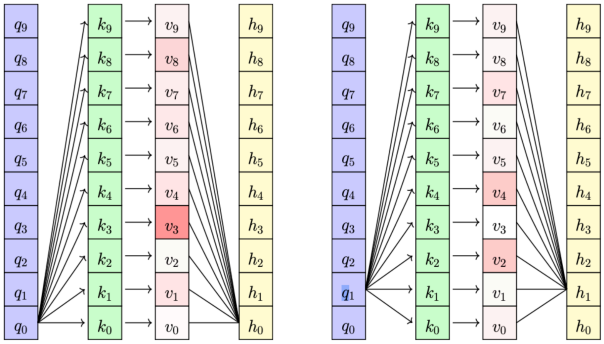
3. Wende Softmax an (normalisiere Attention Scores):

$$A = \text{softmax}(E)$$

4. Berechne die gewichtete Summe der Values und skaliere:

$$\text{Output} = \text{softmax}\left(Q\frac{K^T}{\sqrt{d_k}}\right)V$$

11.4.1.2. Queries, Keys & Values



- **Query (q):** aktuelles Wort
- **Key (k):** die anderen Wörter im Satz
- **Value (v):** Wert der weitergegeben wird

$$\text{outputs} = \sum(\text{values} * \text{pairwise\_scores}(\text{queries}, \text{keys}))$$

11.4.2. Attention is (not quite) all you need

Attention kann keine nicht-lineare Transformation machen

→ Daher **MLP** verwenden

Transformer können sehr tief werden beim Trainieren, daher braucht es zusätzlich:

- **Residual Connectinos**
- **Layer Normalizaiton**
- **Scaled Dot Product Attention**

11.4.2.1. Residual Connections

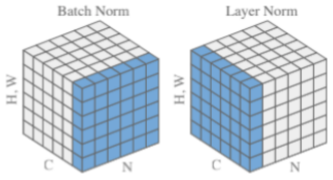
- Siehe @Residual_Networks

11.4.2.2. Layer Normalization

- **Kernproblem:** Schwierige Parameter-Optimierung, da sich die Eingabeverteilungen tieferer Schichten während des Trainings ständig verschieben
- **Lösung:** Normalisierung der Aktivierungen auf einen Mittelwert von Null und eine Standardabweichung von Eins.
- **Verfahren:** Reduzierung der Variation innerhalb jeder Schicht für jeden einzelnen Datenpunkt.

Vergleich Batch Norm vs. Layer Norm:

- **Batch Normalization:** Normalisiert über die Batch-Dimension (alle Datenpunkte für ein Merkmal).
  - Durchschnittsnote *einer* bestimmten Übung über *alle* Studierenden hinweg.
- **Layer Normalization:** Normalisiert über die Feature-Dimension (alle Merkmale eines einzelnen Datenpunktes).
  - Durchschnittsnote *eines* bestimmten Studierenden über *alle* seine Übungen hinweg.



11.4.2.3. Scaled Dot Product Attention

- **Ausgangslage:** Elemente nach der Layer Normalization haben Mittelwert 0 und Varianz 1.
- **Problem:** Das Skalarprodukt (QK^T) neigt bei hohen Dimensionen d_k zu extrem grossen Werten.
- **Lösung (Scaling):** Division des Skalarprodukts durch $\sqrt{d_k}$, um die Varianz wieder auf 1 zu bringen.

- **Ziel:** Verhindern, dass die Softmax-Funktion in Regionen mit extrem geringen Gradienten gerät, was das Training stabilisiert.

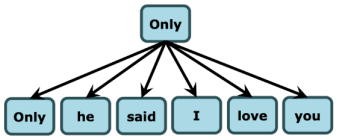
$$\text{Output} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

11.4.3. Ordering

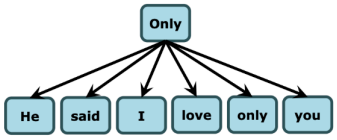
$$\text{Output} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Problem:

- Für die **Formel macht es keinen Unterschied**, in welcher Reihenfolge die Wörter (Vektoren) eingegeben werden.
- Die **Position eines Wortes ist entscheidend für die Bedeutung** eines Satzes.



He said, “I love you.” (Nice thought.)
Only he said, “I love you.” (No one else said it.)
He **only** said, “I love you.” (He said nothing else.)
He said, “**Only** I love you.” (No one else does.)
He said, “I love **only** you.” (He doesn’t love any one else.)
He said, “I love you **only**.” (His love is exclusive.)



Lösung: Positional Encoding

11.4.4. Positional Encoding

Grundidee: Jedem Wort-Vektor wird ein **Positions-Vektor** (p_i) hinzufügen.

$$p_i \in \mathbb{R}^d, \text{ for } i \in \{1, 2, ..., T\}$$

Vorgehen: Man addiert diesen Positions-Wert direkt zu den Werten für **Values (v)**, **Queries (q)** und **Keys (k)**.

$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

Beispiel:

- `[Only]` + 0, `[he]` + 1, `[said]` + 2, `[I]` + 3, `[love]` + 4, `[you]` + 5
- `[He]` + 0, `[said]` + 1, `[I]` + 2, `[love]` + 3, `[only]` + 4, `[you]` + 5

Probleme:

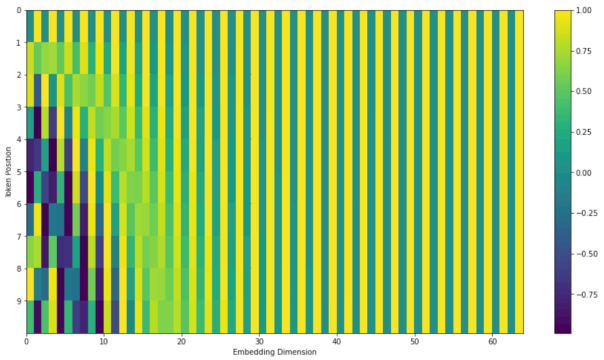
- Zahlen explodieren in langen Sätzen
- Eine Normalisierung auf Werte zwischen 0 und 1 bringt nichts
  - Variable Längen der Sequenzen möglich

Lösung: Sinusoids

11.4.4.1. Sinusoids

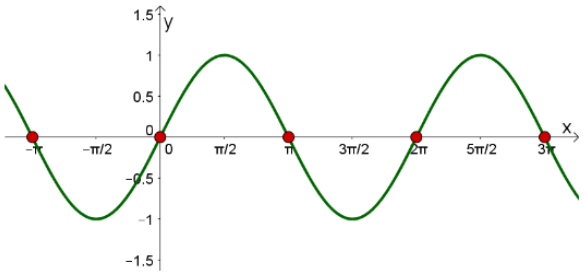
- Alle Werte liegen stabil zwischen **0 und 1** (bzw. −1 und 1).
- **Das Problem:** Eine einzelne Welle wiederholt sich
  - verschiedene Positionen hätten den **gleichen Wert (Duplikate)**.
- **Die Lösung:** Kombination vieler Wellen mit **unterschiedlichen Geschwindigkeiten** (Frequenzen).

$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin\left(\frac{\mathrm{pos}}{10000^{2\frac{i}{d_{\mathrm{model}}}}}\right)$$
$$\mathrm{PE}_{(\mathrm{pos},2i+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{2\frac{i}{d_{\mathrm{model}}}}}\right)$$



- Jede Position bekommt so ein **einmaliges Muster**.
- **Relativer Abstand:** Das Modell lernt durch die Wellenbewegungen, wie weit Wörter **voneinander entfernt** sind.

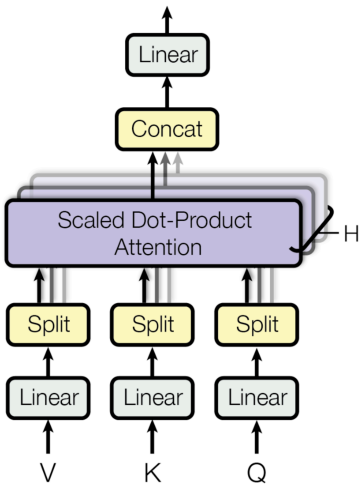
Problem: nur eine harmonische Funktion: es gibt Duplikate (rote Punkte)



mit mehreren Funktionen, kann man diese Duplikate entfernen

11.4.5. Multi-headed Self-Attention

DEFINITION: Anstatt nur einmal Attention zu berechnen, nutzt man mehrere Heads gleichzeitig.



- Jeder Head nutzt eigene Q-, K- und V-Matrizen und arbeitet unabhängig.

Vorteil: Unterschiedliche Heads können sich auf **verschiedene Zusammenhänge** im Satz konzentrieren:

- **Grammatik:** z. B. Subjekt-Verb-Beziehung („*She*“ -> „*sells*“).
- **Semantik:** z. B. logische Verknüpfungen zwischen Nomen („*sea*“ -> „*shore*“).
- **Referenzen:** Wortwiederholungen oder Bezüge erkennen.

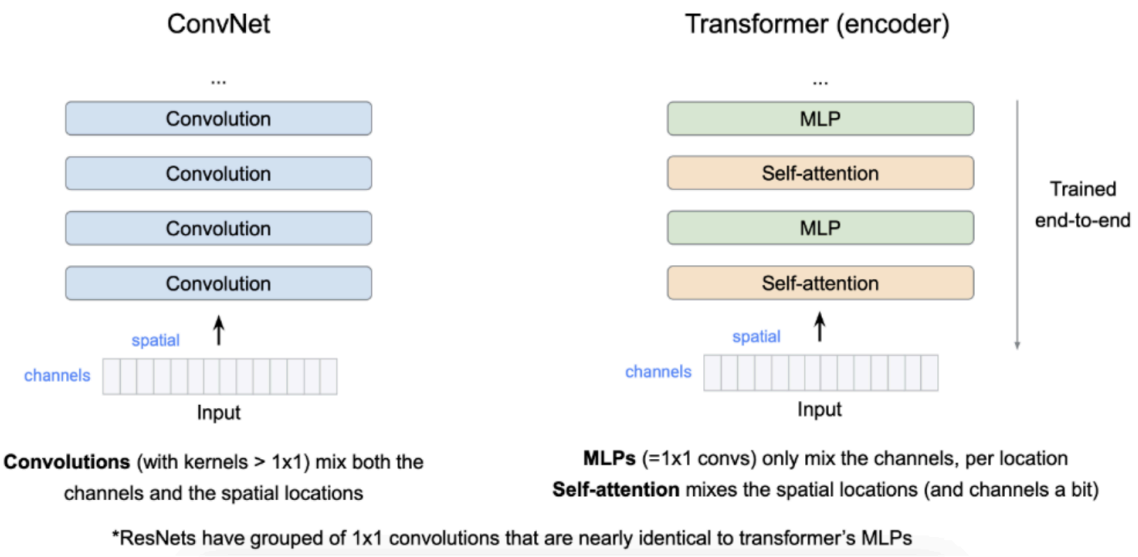
Ablauf

- Jeder Head berechnet Attention eigenständig
- Outputs von allen Heads werden kombiniert

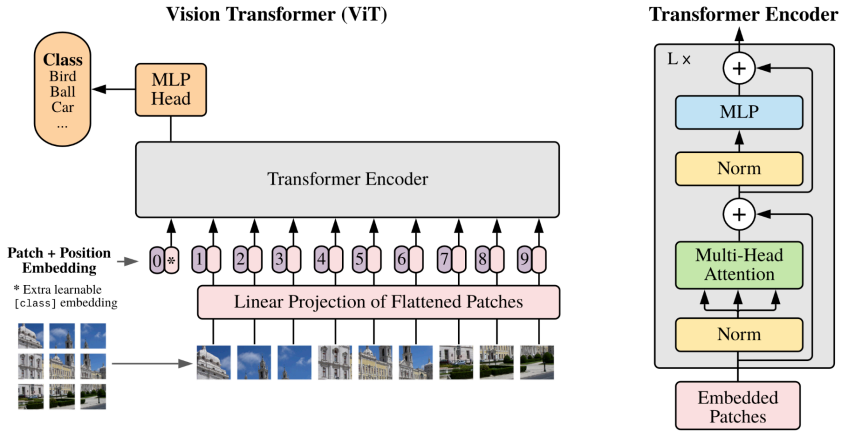
$$\text{output} = Y[\text{output}_1; \ldots; \text{output}_h], \text{ where } Y \in \mathbb{R}^{d \times d}$$

- Das Ergebnis mit einer trainierten Weight-Matrix multiplizieren und das Ergebnis auf die Länge Outputs abbilden.

11.4.6. ConvNets vs. Transformers



11.5. Vision Transformer - ViT

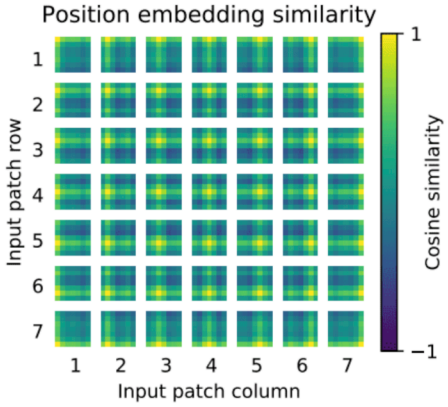


Der Decoder-Teil des Standard-Transformers wird weggelassen; man nutzt nur den **Encoder zur Feature Extraction**.

- **Input-Verarbeitung:** Das Bild wird in quadratische **Patches** (Teilstücke) zerlegt, flattened und in Vektoren (Embeddings) umgewandelt.
- **Klassifizierung:** Anstatt Text zu generieren, wird am Ende ein **MLP-Head** verwendet, um die Bildklasse vorherzusagen.

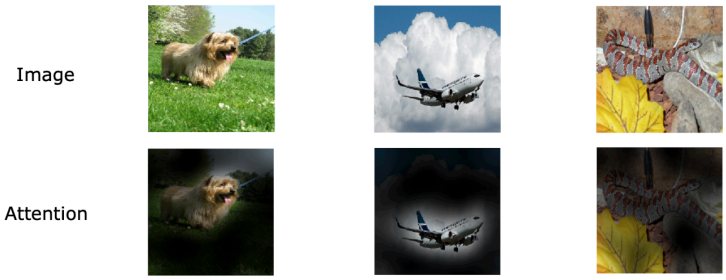
11.5.1. Positional Encoding

- **Spatial Information:** Obwohl die 2D-Struktur beim Zerlegen zunächst verloren geht, lernt das Modell durch **Positional Encodings** die relative Lage der Patches wieder her.
- **Globale Attention:** Ein Patch (bzw. „Pixel-Gruppe“) kann mit **allen anderen Patches** im Bild gleichzeitig verglichen werden.



11.5.2. Mean Attention Distance

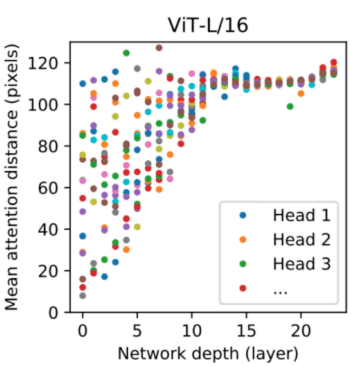
DEFINITION: Beschreibt, wie weit entfernte Bildbereiche ein Attention-Head gleichzeitig sehen kann (ähnlich dem Receptive Field bei CNNs).

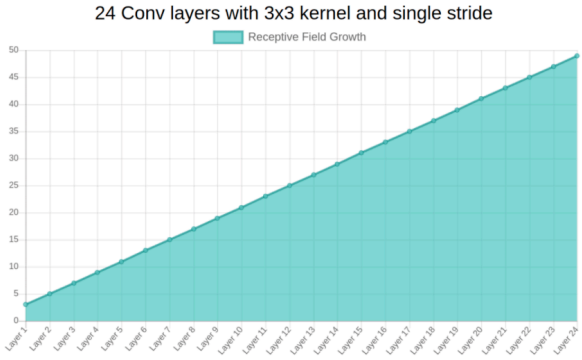


- **Attention** konzentriert sich automatisch auf **wichtigen Objekte** im Bild und ignoriert unwichtigen Hintergrund.
- Mit **zunehmender Tiefe ViT** nutzen **fast alle Heads grosse Distanzen**, um **komplexe globale Zusammenhänge** zu verstehen

ViT vs. CNN:

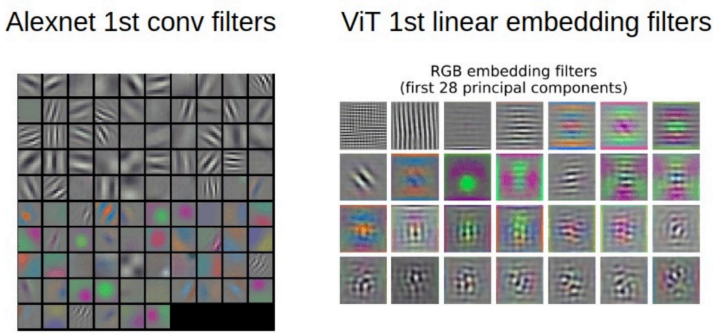
- **CNN:** Das Sichtfeld wächst nur **langsam und linear** mit jeder tieferen Schicht (lokaler Fokus).
- **ViT:** Schon in den **untersten (frühen) Schichten** gibt es Attention-Heads, die das **gesamte Bild** erfassen (globaler Fokus).





11.5.3. Learned Filters

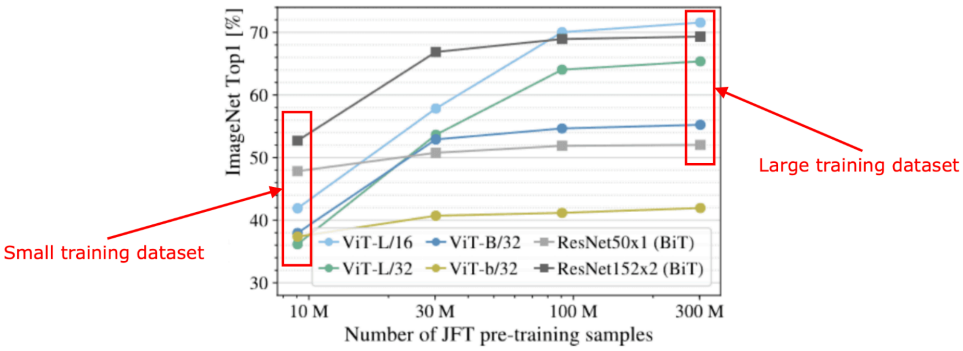
- **Vergleich:** CNN (AlexNet) vs. Vision Transformer (ViT).
- **Beobachtung:** Beide lernen in der ersten Schicht identische Filter (Kanten, Texturen, Farben).



- **Besonderheit:** CNNs sind dafür gebaut. ViTs nicht (sie verarbeiten nur einfache Pixel-Patches).
- **Fazit:** ViTs lernen völlig selbstständig, dass Kantenerkennung der beste erste Schritt ist, um Bilder zu „verstehen“.

11.5.4. Training Dataset Size - Data-Hungry

DEFINITION: Vision Transformer (ViT) sind extrem datenhungrig.



- **Mittlere Datensätze (ImageNet):** Mässige Leistung. Traditionelle Modelle (BiT, graue Linie) schneiden hier besser ab.
- **Riesige Datensätze (14M - 300M Bilder):** Exzellente Leistung. ViTs überholen traditionelle Modelle deutlich.
  - Traditionelle Modelle profitieren kaum noch

11.6. Translation Equivariant

DEFINITION: Gleiche Vorhersage bei Verschiebung des Objekts

- Bei CNN ja
- Transformer nein

11.7. Inductive Bias

DEFINITION: Wie flexibel ist mein Modell

- Wie viel ist schon in meinem Modell definiert
- z.B. am Anfang definieren der Filter ist weniger flexibel
- werden Filter selber gelernt, ist es flexibler
- mehr inductive Bias => weniger flexibel

Am meisten Inductive Bias:
1. Bag of Words
2. CNN
3. Transformer

11.8. CoAtNet

DEFINITION: Kombination von Convolution & Attention

Fokus auf die Balance zwischen **Genauigkeit** (Accuracy) und **Effizienz**.

Problemstellung bei Transformers in der Computer Vision:

- Rückstand gegenüber SOTA CNNs bei geringen Datenmengen
- Schlechtere Generalisierung trotz hoher Modellkapazität.
  - not translation equivariant
  - lack of inductive bias

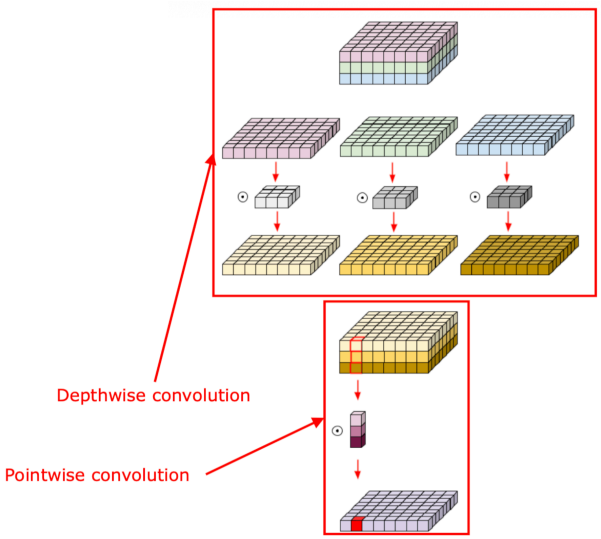
11.8.1. Depthwise-separable Convolution

Zweigeteilter Prozess

- Depthwise Convolution
- Pointwise Convolution

Depthwise Convolution: Räumliche Faltung erfolgt unabhängig für jeden einzelnen Eingangskanal.

Pointwise Convolution: Einsatz einer 1×1 Faltung zur Kombination der Ergebnisse aus dem ersten Schritt.



Vorteile:

- Hohe Recheneffizienz.
- Geringerer Bedarf an Rechenressourcen.

11.8.2. Combination of Convolution and Attention

11.8.2.1. Depthwise Convolution

- Abhängigkeit der Kernel-Gewichte nur von relativer Position (i, j)
- Translationsinvarianz zur Verbesserung der Generalisierung
- Eingabeunabhängiger Kernel w

$$y_i = \sum_{j \in \mathcal{L}(i)} w_{i-j} \odot x_j \quad \text{(depthwise convolution)}$$

- $\mathcal{L}(i)$: local neighborhood von i (z.B. 3x3)

11.8.2.2. Attention

- **Dynamische Abhängigkeit** der Aufmerksamkeitsgewichte von der Eingabe
- Vereinfachtes Erfassen komplexer relationaler Interaktionen zwischen räumlichen Positionen
- **Globales Rezeptives Feld** für mehr Kontextinformationen
- Deutlich höherer Rechenaufwand

$$y_i = \sum_{j \in \mathcal{G}} \underbrace{\frac{\exp\left(x_i^T x_j\right)}{\sum_{k \in \mathcal{G}} \exp\left(x_i^T x_k\right)}}_{A_{i,j}} x_j \quad \text{(self-attention)}$$

- $\mathcal{G}$: Global neighborhood (ganzes Bild)

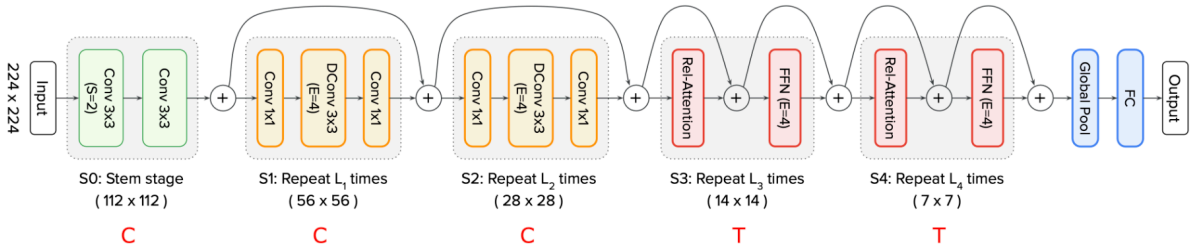
11.8.2.3. Combination

- **Attention-Weights:** Kombination aus eingabeadaptivem Wert $x_i^T x_j$ und globalem Kernel-Wert w_{i-j}
- Bekannt als **Relative Self-Attention**

$$y_i^{\text{pre}} = \sum_{j \in \mathcal{G}} \frac{\exp\left(x_i^T x_j + w_{i-j}\right)}{\sum_{k \in \mathcal{G}} \exp\left(x_i^T x_k + w_{i-k}\right)} x_j$$

11.8.3. Vertical Design

- Relative Attention auf Pixelebene nicht praktikabel ($O(n^2)$ Komplexität)
- **Downsampling:** Notwendigkeit der Bildverkleinerung
  - z. B. durch Strided Convolutions
- **CNN-Architektur-Prinzip:** Unterteilung in verschiedene Phasen mit Downsampling zu Beginn jeder Phase bei gleichzeitiger Erhöhung der Kanalanzahl
- **Optimale Konfiguration:**
  - Bestwerte für Generalisierung und Transferfähigkeit erzielt durch
    - **3 Convolution Blocks**
    - gefolgt von **2 Transformer Blocks**



12. Detecting Multiple Objects

DEFINITION: Object Detection kombiniert die Klassifizierung und Lokalisierung von mehreren Objekten in einem Bild.

12.1. Cascaded Classifier Approach (Viola Jones)

- **Problem**:
  - Viele Positionen im Bild auf mögliche Gesichter prüfen
  - Das Scannen dieser vielen Fenster möglichst effizient machen
  - Geeignete Merkmale für die Erkennung finden
- **Grundideen**:
  - Boosting mit AdaBoost zur Auswahl und Kombination guter Klassifikatoren
  - Einfache Features zur kompakten Beschreibung von Bildinhalten
  - Kaskadierte Klassifikatoren zur frühen Verwerfung negativer Fenster

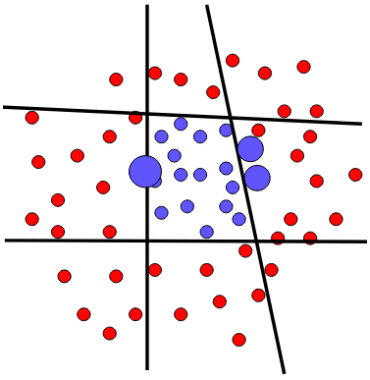
12.1.1. Boosting

DEFINITION: Boosting kombiniert mehrere schwache Klassifikatoren zu einem starken Klassifikator, indem schwer klassifizierbare Beispiele im Training schrittweise stärker gewichtet werden.

- **Ziel**:
  - Einen Klassifikator finden, der die Klassen möglichst gut trennt
- **Idee**:
  - Mehrere schwache Klassifikatoren zu einem starken Klassifikator kombinieren
- **Formel**:
  - $F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + ...$
  - $f_{i(x)}$ = schwache Klassifikatoren
  - α_i = deren Gewichte
- **Ergebnis**:
  - durch die gewichtete Kombination vieler schwacher Klassifikatoren entsteht ein deutlich stärkerer, oft nichtlinearer Klassifikator
- Unterschrift Endbild:
  - Kombination mehrerer schwacher Entscheidungsgrenzen ergibt eine starke Gesamtregel

12.1.1.1. Ablauf

1. Trainingsdaten erhalten anfangs Gewichte
2. Ein schwacher Klassifikator wird trainiert
3. Der Klassifikator bewertet die Trainingsbeispiele
4. Falsch klassifizierte Beispiele bekommen höheres Gewicht
5. Korrekt klassifizierte Beispiele bekommen geringeres Gewicht
6. Danach wird der nächste schwache Klassifikator auf den angepassten Gewichten trainiert
7. Am Ende werden alle schwachen Klassifikatoren gewichtet zu einem starken Klassifikator kombiniert

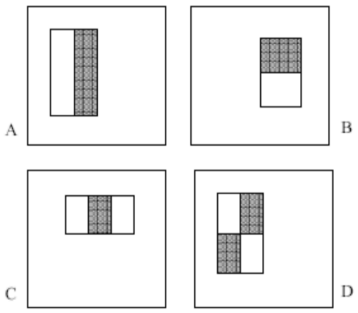


Kombination mehrerer schwacher Klassifikatoren ergibt eine starke nichtlineare Entscheidungsgrenze, die die Klassen deutlich besser trennt

12.1.2. Features

DEFINITION: **Haar-Features** sind einfache rechteckige Merkmale, die Helligkeitsunterschiede mit **+1** und **−1** erfassen

- Statt aufwendiger allgemeiner Bildfilter werden einfache rechteckige Merkmale verwendet
- Die Haar-Features werden auf vielen Subfenstern des Bildes berechnet
- Für ein 24×24-Subbild gibt es sehr viele mögliche Feature-Kandidaten $(O(n^4))$
- **AdaBoost** wählt die Haar-Features aus, die z.B. Gesichter am besten von Nicht-Gesichtern unterscheiden
- Ein ausgewähltes Feature wird mit einer einfachen Schwelle zu einem **schwachen Klassifikator**



12.1.3. Integral Image

DEFINITION: **Integral Image** ist eine Vorverarbeitung des Bildes, bei der jeder Wert die Summe aller Pixel im links oberen Rechteck bis zu dieser Position speichert

- Ermöglicht die schnelle Berechnung von Rechteckssummen
- Wird auch **Summed Area Table** genannt
- Macht die Berechnung von Haar-Features deutlich effizienter

12.1.3.1. Ablauf

1. Das Integral Image wird einmal über das ganze Bild berechnet
2. Jeder Eintrag speichert die kumulierte Summe aller Pixel links oben von dieser Position
3. Die Summe eines beliebigen Rechtecks kann dann mit wenigen Werten aus dem Integral Image berechnet werden
4. Dadurch lassen sich Feature-Werte sehr schnell bestimmen

12.1.3.2. Beispiel

Original Image

0	1	3	5	2	7	10	7	10	9
3	7	6	9	3	8	1	8	5	8
1	11	0	7	13	2	14	2	13	1
14	3	2	7	1	0	9	7	2	12
1	5	15	3	6	6	5	1	10	6
8	1	2	6	7	3	2	11	0	15
7	7	6	0	9	5	10	3	8	1
12	5	6	10	11	3	6	7	9	1

Integral Image

0	1	4	9	11	18	28	35	45	54
3	11	20	34	39	54	65	80	95	112
4	23	32	53	71	88	113	130	158	176
18	40	51	79	98	115	149	173	203	233
19	46	72	103	128	151	190	215	255	291
27	55	83	120	152	178	219	255	295	346
34	69	103	140	181	212	263	302	350	402
46	86	126	173	225	259	316	362	419	472

- Die Summe eines Rechtecks wird nicht direkt im Originalbild berechnet
- Stattdessen werden vier Werte aus dem Integral Image verwendet:
- Die Rechteckssumme ergibt sich dann mit:

$\sum = D - B - C + A = 215 - 80 - 72 + 20 = 83$

12.1.4. Classification

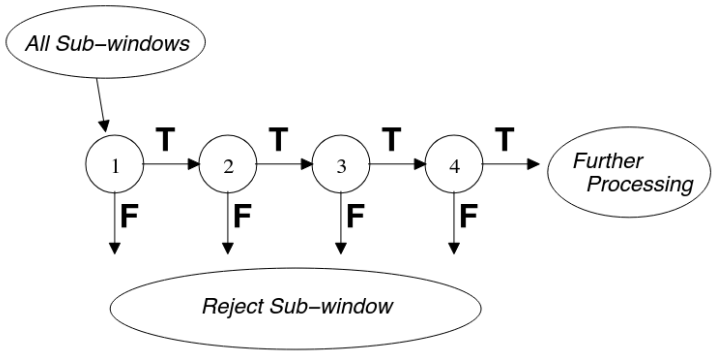
- **Problem**:
  - Die Auswertung von 160′000 möglichen Features pro Subfenster ist rechnerisch zu aufwendig
- **Idee**:
  - Jedes einzelne Feature wird als schwacher Klassifikator behandelt
- **Lösung**:
  - AdaBoost wird verwendet, um gezielt die relevantesten Features auszuwählen und zu trainieren
- **Beispiel AdaBoost**:
  - 1. Feature: Misst den Intensitätsunterschied zwischen der Augenregion und dem Gesicht darunter
  - 2. Feature: Vergleicht die Intensität der Augenregion mit dem Nasenrücken

12.1.4.1. Efficient Face Detection

- Ein Sliding Window Detektor muss tausende Positionen und Skalierungen prüfen
- Gesichter sind auf Bildern jedoch selten (meist nur 10-12 pro Bild)
- **Ziele**:
  - Sehr hohe True Positive Rate (85-95%)
  - Extrem tiefe False Positive Rate ($10^{\{-5\}}$ bis $10^{\{-6\}}$)
- **Idee**:
  - Bildbereiche ohne Gesichter so früh und schnell wie möglich verwerfen

12.1.4.2. Cascaded Classifiers Approach

- Einsatz einer Sequenz (Kaskade) von einzelnen Klassifikatoren
- True Positive Raten und False Positive Raten multiplizieren sich über die verschiedenen Stufen
- **Ablauf**:
  1. Ein Sub-Window wird nacheinander durch die Stufen der Kaskade geschickt
  2. Sobald ein Klassifikator „False“ meldet, wird die Region sofort verworfen und nicht weiter getestet
  3. Nur wenn alle Stufen „True“ melden, wird das Fenster weiterverarbeitet



- **Beispiel**:
  - Eine Kaskade aus 10 Stufen, wobei jede Stufe mit AdaBoost und 20 unterschiedlichen Features trainiert wird

12.2. Neuronale Netze: Two-Stage Detectors

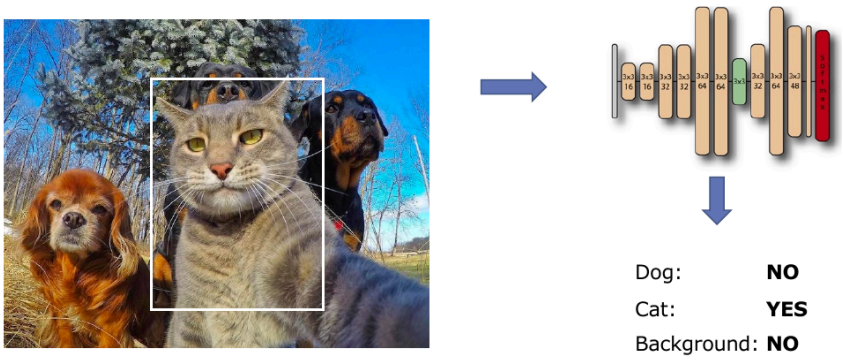
DEFINITION: Objekterkennung in zwei Schritten: 1. Region Proposals erstellen, 2. Regionen klassifizieren und Bounding Boxes anpassen.

12.2.1. Classification and Localization

- Das Erkennen mehrerer Objekte wird in zwei Teilaufgaben unterteilt:
  - **Classification** (Klassifikation): Die korrekte Klasse des Objekts bestimmen (z.B. Katze oder Hund)
  - **Localization** (Lokalisierung): Die beste Bounding Box um das Objekt finden

12.2.1.1. Sliding Windows

- **Idee**: Ein CNN wird auf viele verschiedene Sub-Fenster eines Bildes angewendet, um Merkmale zu extrahieren und zu klassifizieren
- **Problem**: Es ist extrem rechenaufwendig, das CNN auf eine riesige Anzahl von Positionen und Grössen anzuwenden



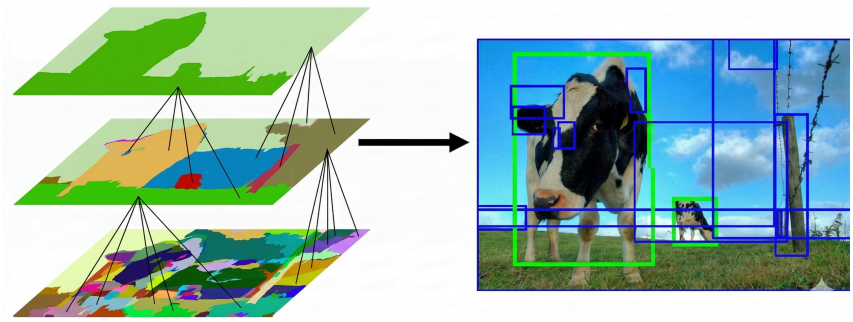
12.2.1.2. Region Proposals

- **Ziel**: Finden von fleckigen (blobby) Bildregionen, die mit hoher Wahrscheinlichkeit Objekte enthalten
- **Vorteil**: Relativ schnell, liefert z. B. ca. 1000 Vorschläge in wenigen Sekunden auf der CPU

12.2.1.2.1. Ablauf Selective Search

- Ein Bottom-up Ansatz zur Segmentierung, der Regionen über mehrere Skalen hinweg zusammenfügt

1. Das Bild wird initial in viele kleine Teilregionen segmentiert (Subsegmentierung)
2. Ähnliche Regionen werden rekursiv zu grösseren Regionen kombiniert (basierend auf Farbe, Textur, Grösse)
3. Aus diesen zusammengefügten Regionen werden schliesslich die Bounding Boxes (Candidate Proposals) abgeleitet

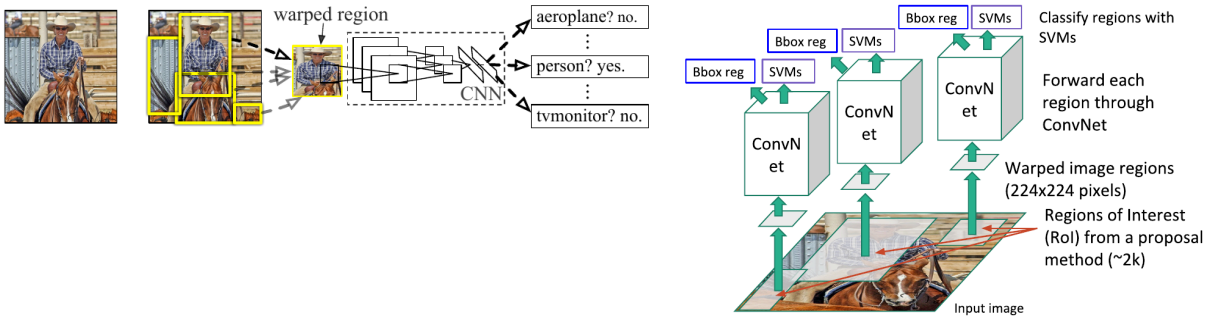


12.2.2. R-CNN Architektur

DEFINITION: R-CNN (Regions with CNN features) ist ein zweistufiges Verfahren, das zuerst mögliche Objektregionen vorschlägt und diese anschliessend einzeln durch ein neuronales Netz (CNN) bewertet und klassifiziert.

- **Problem**: Klassisches R-CNN ist zu langsam, da das CNN für jeden der 2000 Vorschläge einzeln berechnet werden muss

12.2.2.1. Ablauf



1. **Input**: Bild in das System laden
2. **Region Proposals**: Ca. 2000 potenzielle Objektregionen generieren (z. B. mit Selective Search)
3. **Warping**: Alle 2000 ausgeschnittenen Regionen auf eine feste Pixelgrösse (z. B. 224x224) skalieren
4. **Feature Extraction**: Jede Region einzeln durch ein CNN leiten, um die Merkmale zu extrahieren
5. **Klassifikation**: Die Objektklasse oder den Hintergrund anhand der Merkmale mit Klassifikatoren bestimmen
6. **Bounding Box Regression**: Die Koordinaten der Rahmen anhand der Merkmale exakt an das Objekt anpassen

Details

- Regionen werden auf 224x224 skaliert
- Das CNN extrahiert 4096 Features pro Region
- Verwendet wird AlexNet mit 5 Convolutional und 2 Fully Connected Layers
- Zuerst erfolgt Pre-Training auf ILSVRC2012
- Danach folgt Fine-Tuning auf den Region Proposals
- Pro Klasse wird ein linearer SVM trainiert
- Regionen mit 30% Overlap gelten als positive Beispiele
- Die Bounding Box wird über Regression verfeinert

12.2.2.2. Bounding Box Regression

DEFINITION: Bounding Box Regression verfeinert eine zuerst grobe Bounding Box, indem Positions- und Grössenkorrekturen relativ zu einer Ground-Truth-Box gelernt werden.

- Gegeben sind eine vorhergesagte Box $p = (p_x, p_y, p_w, p_h)$ und eine Ground-Truth-Box $g = (g_x, g_y, g_w, g_h)$
- Gelernt wird eine Transformation $d(p)$, die die Box an das Objekt anpasst
- Die verfeinerte Box ergibt sich durch:

$$\hat{g}_x = p_w d_{x(p)} + p_x$$

$$\hat{g}_y = p_h d_{y(p)} + p_y$$

$$\hat{g}_w = p_w e^{d_{w(p)}}$$

$$\hat{g}_h = p_h e^{d_{h(p)}}$$

- Die Zielwerte für das Training sind:

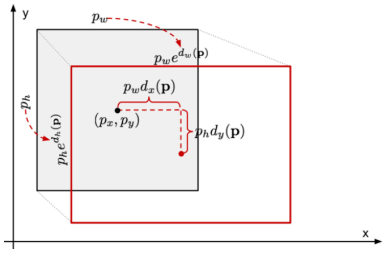
$$t_x = \frac{g_x - p_x}{p_w} \qquad t_w = \log\left(\frac{g_w}{p_w}\right)$$

$$t_y = \frac{g_y - p_y}{p_h} \qquad t_h = \log\left(\frac{g_h}{p_h}\right)$$

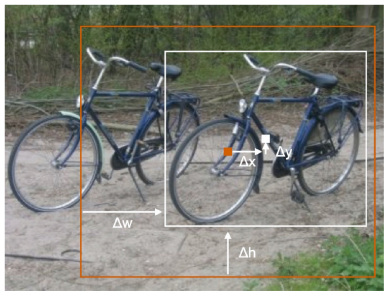
- Trainiert wird mit einer Regressions-Loss:

$$L_{\text{reg}} = \sum_{i \in \{x,y,w,h\}} \left(t_i - d_{i(p)}\right)^2 + \lambda \|w\|^2$$

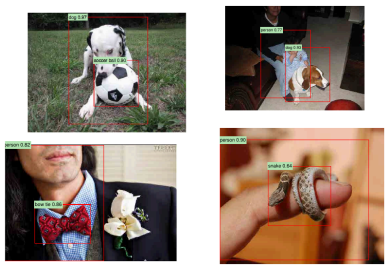
- So lernt das Modell, Position und Grösse der Bounding Box zu korrigieren



Bounding-Box-Regression als Transformation



Beispiel für die Korrektur einer Bounding Box



Beispiel R-CNN Resultat

12.2.2.3. Post-Processing

DEFINITION: Beim Post-Processing werden mehrfach erkannte, stark überlappende Bounding Boxes auf die beste Vorhersage reduziert

- Nach Klassifikation und Bounding Box Regression entstehen oft mehrere überlappende Bounding Boxes für dasselbe Objekt
- Diese mehrfachen Vorhersagen werden mit **Non-Max Suppression (NMS)** bereinigt

12.2.2.4. Non-Max Suppression

1. Wähle die Bounding Box mit dem höchsten Score
2. Entferne alle stark überlappenden Boxen derselben Klasse
3. Wiederhole den Schritt mit den übrigen Boxen
4. Am Ende bleibt pro Objekt meist nur die beste Bounding Box übrig

- **Ziel**:
  - Mehrfacherkennungen vermeiden
  - Pro Objekt nur die plausibelste Box behalten

12.2.2.5. Probleme mit R-CNN

- R-CNN ist sehr langsam, weil für jede Proposal-Region ein eigener CNN-Forward-Pass berechnet werden muss
- Bei rund 2000 Regionen pro Bild ist das rechnerisch sehr aufwendig
- Die Architektur berechnet viele Convolutionen mehrfach für überlappende Bildbereiche
- Dadurch ist R-CNN für praktische Anwendungen oft zu langsam

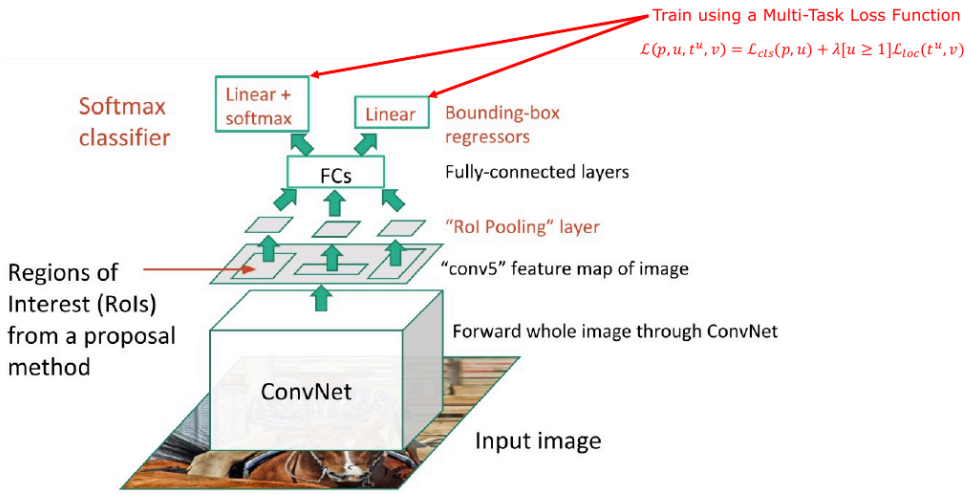
12.2.3. Fast R-CNN

DEFINITION: Fast R-CNN beschleunigt R-CNN, indem das Bild nur einmal durch das CNN verarbeitet wird. Die Regionen werden erst danach aus der gemeinsamen Feature Map entnommen

- **Idee**:
  - Bei R-CNN wird jede Proposal-Region separat durch das CNN geschickt
  - Fast R-CNN berechnet die Convolutionen nur einmal für das ganze Bild
  - Dadurch entfallen viele doppelte Berechnungen
- **Wichtig**:
  - Die Region Proposals kommen meist weiterhin von einem externen Verfahren wie **Selective Search**
  - Neu ist vor allem, dass die Merkmale für alle Regionen aus **einer gemeinsamen Feature Map** gelesen werden
- **Zusätzlich**:
  - Statt eines separaten SVM-Klassifikators verwendet Fast R-CNN direkt einen **Softmax-Klassifikator**
  - Auch die Bounding Box Regression ist direkt im Netz integriert
- **Vorteil**:
  - Deutlich schneller als R-CNN
  - Training und Vorhersage sind stärker in einem einzigen Netz zusammengefasst

12.2.3.1. Ablauf

1. Das gesamte Eingabebild wird einmal durch ein CNN geleitet
2. Dadurch entsteht eine gemeinsame **Feature Map** für das ganze Bild
3. Externe **Region Proposals** markieren interessante Bildbereiche (**Regions of Interest**, RoIs)
4. Für jede RoI wird der passende Bereich aus der Feature Map entnommen
5. Mit **RoI Pooling** wird jede Region auf eine feste Grösse gebracht
6. Die so vereinheitlichten Merkmale werden durch Fully Connected Layers weiterverarbeitet
7. Das Netz gibt für jede Region zwei Dinge aus:
   - Die Objektklasse mittels **Softmax**
   - Eine verfeinerte Bounding Box mittels **Bounding Box Regression**



- **Warum ist das schneller?**
  - Die aufwendigen Convolutionen werden nicht mehr für jede Region einzeln berechnet
  - Stattdessen teilt sich jede Region dieselbe bereits berechnete Feature Map
- **Training**:
  - Fast R-CNN wird mit einer **Multi-Task-Loss** trainiert
  - Dabei werden **Klassifikation** und **Bounding Box Regression** gleichzeitig gelernt

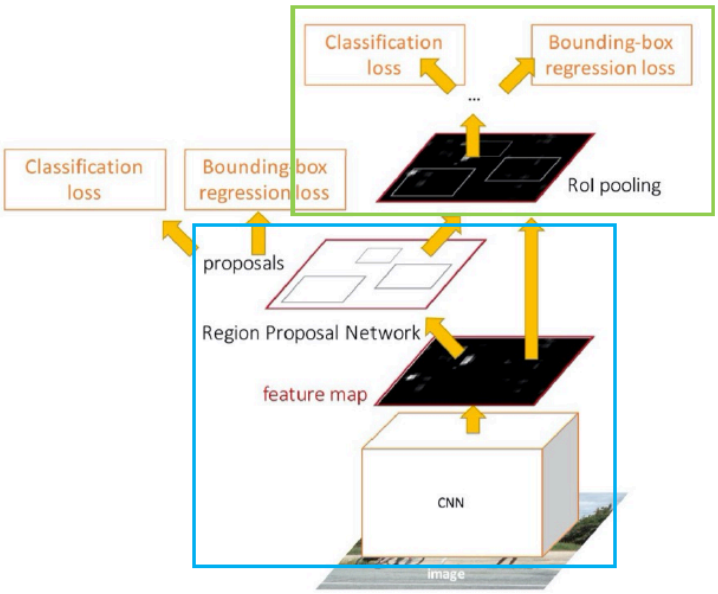
12.2.4. Faster R-CNN und Region Proposal Networks (RPN)

DEFINITION: Faster R-CNN ist ein Two-Stage-Detector, bei dem die Region Proposals nicht mehr von einem externen Verfahren kommen, sondern direkt von einem neuronalen Netz erzeugt werden.

- **Unterschied zu Fast R-CNN**:
  - Region Proposals werden nicht mehr extern berechnet, sondern direkt vom **RPN** erzeugt
- **Vorteil**:
  - Das ganze System arbeitet mit gemeinsamen CNN-Features
  - Region Proposals und Objekterkennung werden in einem verbundenen Modell berechnet

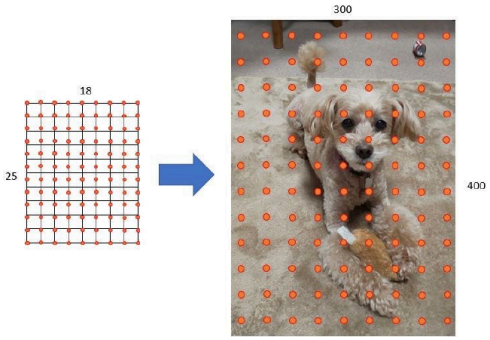
12.2.4.1. Ablauf

1. Das Eingabebild wird einmal durch ein Backbone-CNN geleitet
2. Dadurch entsteht eine gemeinsame **Feature Map**
3. In der **ersten Stufe** erzeugt das **RPN** Region Proposals auf dieser Feature Map
4. Dazu bewertet es viele Anchor Boxes als **Objekt** oder **kein Objekt** und passt ihre Lage grob an
5. Die besten Vorschläge werden als **RoIs** an die **zweite Stufe** weitergegeben
6. Für jede RoI werden die Merkmale aus der Feature Map entnommen
7. Mit **RoI Pooling** oder **RoI Align** werden sie auf eine feste Grösse gebracht
8. Die zweite Stufe bestimmt die genaue Objektklasse
9. Gleichzeitig verfeinert sie die Bounding Box weiter



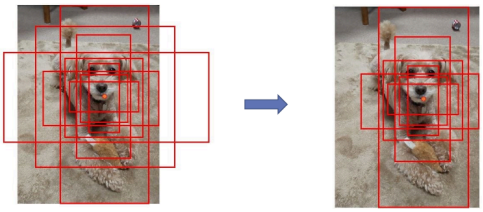
12.2.4.2. Anchor Points

- Auf der Feature Map werden regelmässig **Anchor Points** gesetzt, z. B. alle 16 Pixel
- Jeder Anchor Point dient als Ausgangspunkt für mehrere Box-Vorschläge
- Bei einem Bild von 300x400 Pixeln ergeben sich so z. B. 18x25 = 450 Anchor Points



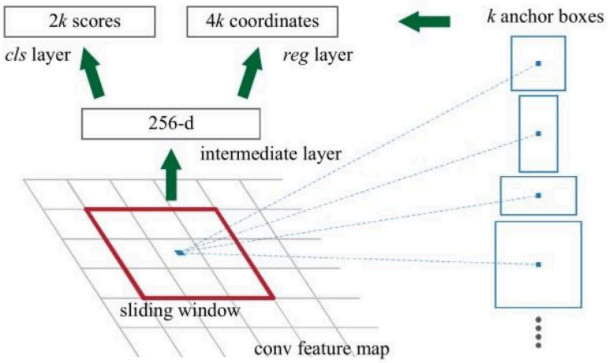
12.2.4.3. Anchor Boxes

- Zu jedem Anchor Point werden mehrere **Anchor Boxes** mit unterschiedlicher Grösse und Form definiert
- Sie unterscheiden sich in:
  - **Skala** (z. B. klein, mittel, gross)
  - **Seitenverhältnis** (z. B. quadratisch, hoch, breit)
- Beispiel:
  - 3 Skalen und 3 Seitenverhältnisse ergeben 9 Anchor Boxes pro Anchor Point
- So lassen sich Objekte verschiedener Grössen und Formen besser erfassen



12.2.4.4. Region Proposals

- Das RPN betrachtet die Feature Map mit einem kleinen Sliding Window
- Für jede Anchor Box sagt es voraus:
  - **Objekt oder kein Objekt**
  - **Korrektur der Bounding Box**
- Daraus entstehen die **Region Proposals**
- Unwahrscheinliche oder stark überlappende Vorschläge werden verworfen

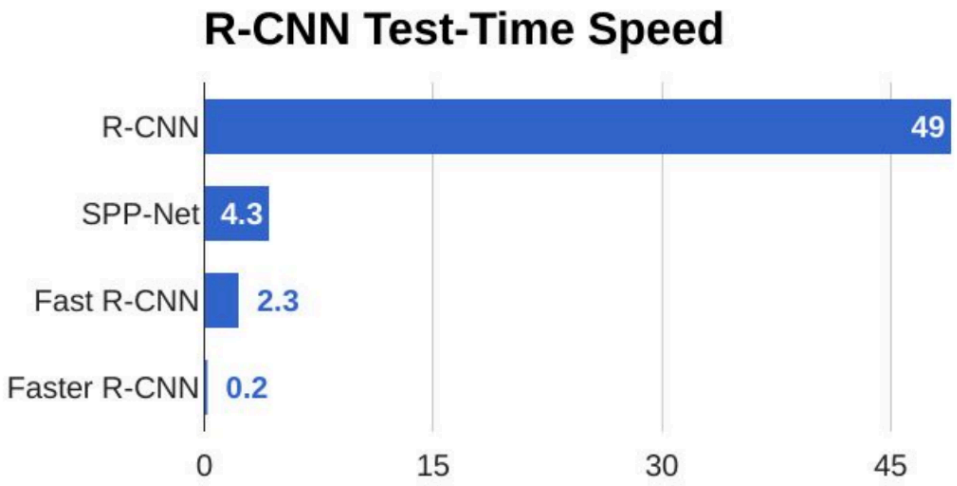


12.2.4.5. Training

- Faster R-CNN lernt mehrere Aufgaben gleichzeitig:
  - Klassifikation im RPN: Objekt / kein Objekt
  - Bounding-Box-Regression im RPN

- Endgültige Objektklassifikation in der zweiten Stufe
- Endgültige Bounding-Box-Verfeinerung
- Beide Stufen nutzen dabei dieselbe Feature Map

12.2.5. Geschwindigkeitsvergleich



12.3. Neuronale Netze: One-Stage Detectors

DEFINITION: One-Stage-Detectors sagen Objektklassen und Bounding Boxes in einem einzigen Schritt direkt aus dem Bild voraus.

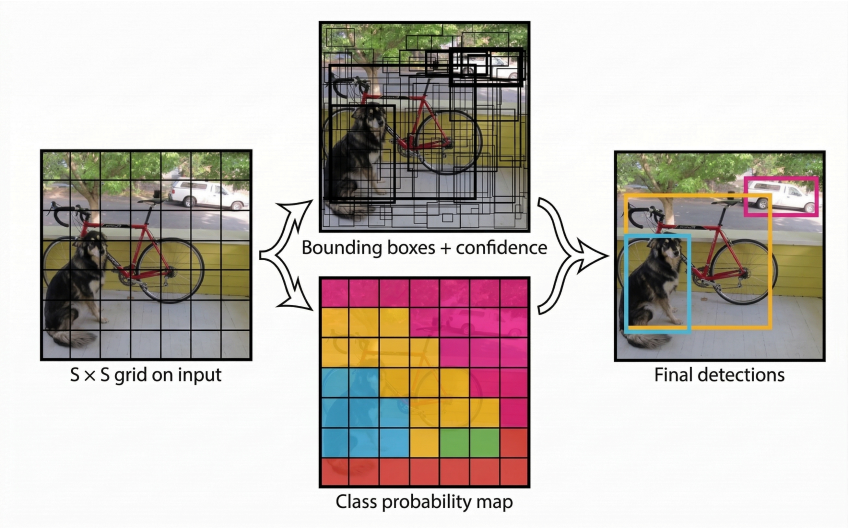
12.3.1. YOLO (You Only Look Once) Architektur

DEFINITION: YOLO teilt das Bild in ein Gitter und sagt für jede Zelle direkt Bounding Boxes, Konfidenzen und Klassenwahrscheinlichkeiten voraus.

- **Idee**:
  - Das ganze Bild wird in einem Schritt verarbeitet
  - Es werden keine separaten Region Proposals erzeugt

12.3.1.1. Ablauf

1. YOLO verarbeitet das ganze Bild in einem Schritt
2. Das Eingabebild wird in ein $S \times S$-Gitter aufgeteilt
3. Jede Gitterzelle ist für Objekte zuständig, deren Mittelpunkt in dieser Zelle liegt
4. Für jede Gitterzelle sagt das Netz mehrere Bounding Boxes voraus
5. Zu jeder Box werden Lage, Grösse und ein **Confidence Score** vorhergesagt
6. Zusätzlich sagt jede Zelle Klassenwahrscheinlichkeiten voraus
7. Confidence Score und Klassenwahrscheinlichkeiten werden kombiniert
8. So entstehen die endgültigen Objekterkennungen im Bild

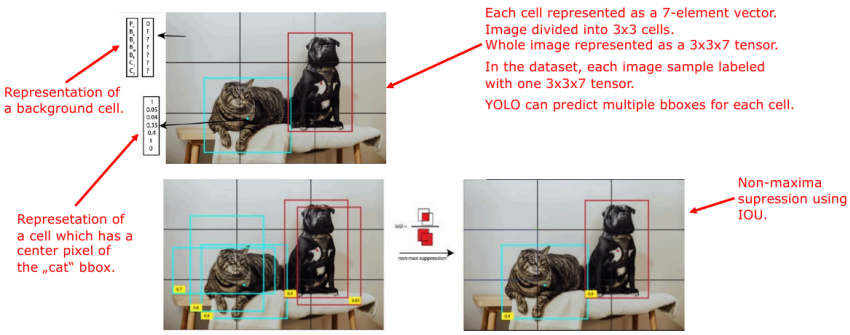


12.3.1.2. Non-Max Suppression

- YOLO erzeugt oft mehrere überlappende Bounding Boxes für dasselbe Objekt
- Deshalb wird **Non-Max Suppression** angewendet

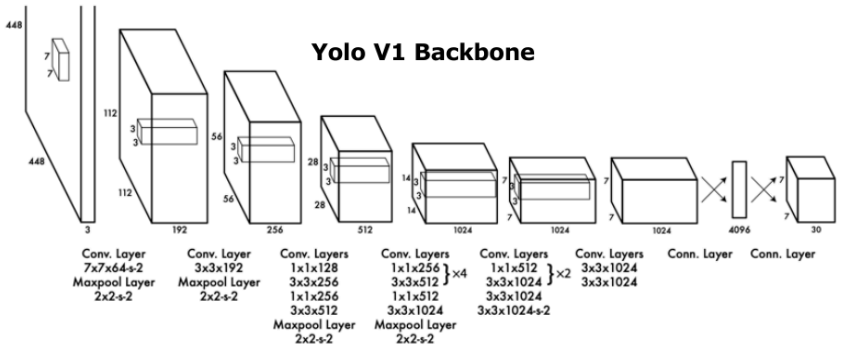
1. Behalte die Box mit dem höchsten Score
2. Entferne stark überlappende Boxen derselben Klasse
3. Wiederhole dies für die verbleibenden Boxen

- So bleibt pro Objekt meist nur die beste Bounding Box übrig



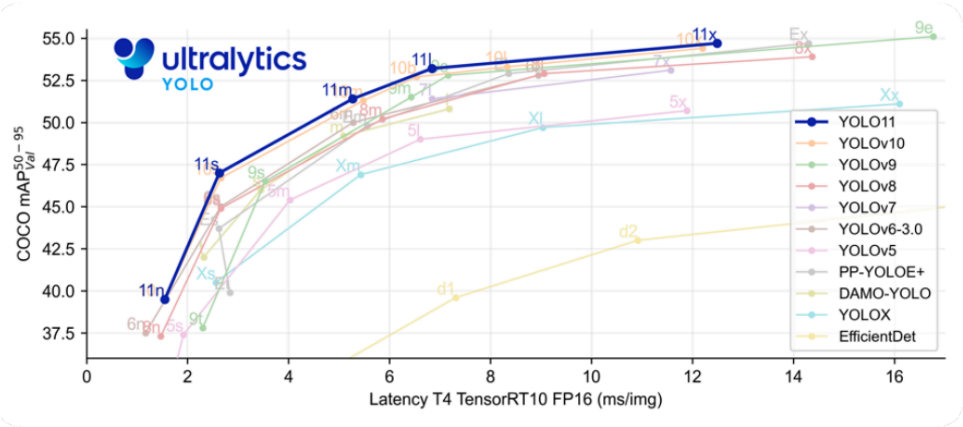
12.3.1.3. Backbone

- Das **Backbone** ist der Teil des Netzes, der aus dem Eingabebild Merkmale extrahiert
- Bei **YOLOv1** wird dazu ein Convolutional Neural Network verwendet
- Es reduziert das Bild schrittweise und erzeugt eine kompakte Feature-Repräsentation
- Auf diesen Merkmalen werden danach Bounding Boxes und Klassen vorhergesagt

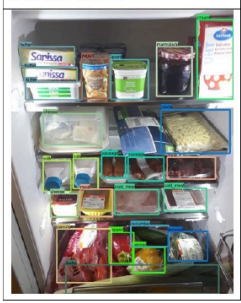


12.3.1.4. YOLO Varianten (V1 bis V11)

- Über die Zeit wurden verschiedene YOLO-Versionen entwickelt
- Dabei wurden vor allem Backbone, Aktivierungsfunktionen, Loss-Funktionen und Effizienz verbessert
- Neuere Varianten sind meist schneller, robuster und genauer als frühe Versionen



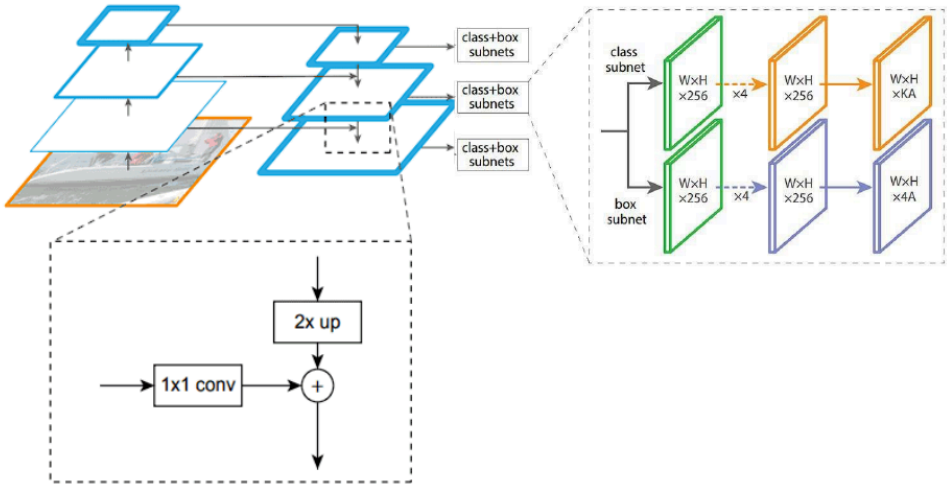
12.3.1.5. Beispiel Fridge Analyzer

	Validation Set		
Ground Truth	**SSD**	**R-FCN**	**YOLO**
			

- Der Vergleich zeigt, dass sich die Modelle vor allem in Vollständigkeit und Genauigkeit der Detektion unterscheiden
- **SSD** erkennt eher weniger Objekte, während **R-FCN** und **YOLO** mehr Vorhersagen liefern
- Dafür können bei dichteren Vorhersagen auch mehr überlappende oder ungenaue Bounding Boxes entstehen

12.3.2. RetinaNet und Focal Loss

DEFINITION: RetinaNet ist ein One-Stage-Detector, der mit *Focal Loss* schwierige Trainingsbeispiele stärker gewichtet als einfache.

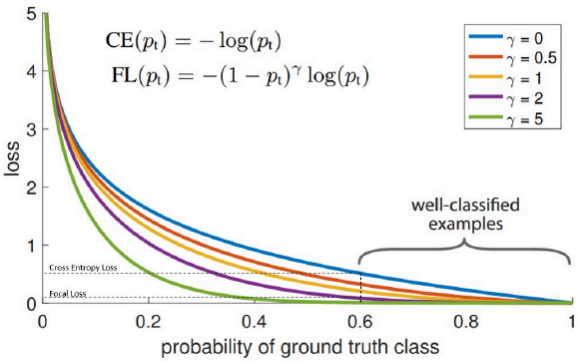


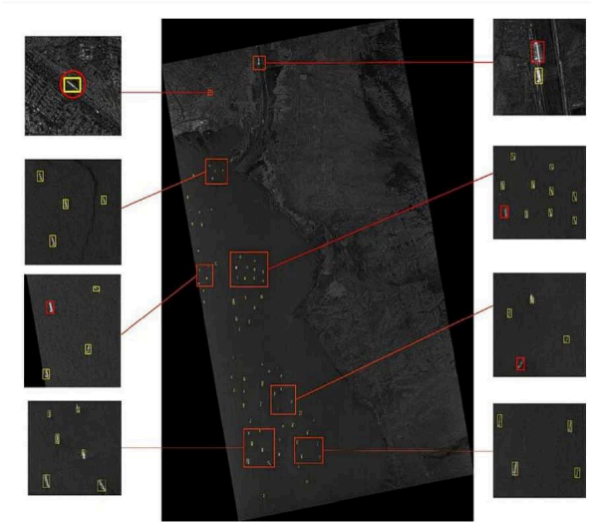
- RetinaNet arbeitet mit mehreren Feature-Ebenen und erkennt so Objekte verschiedener Grössen
- Ein zentrales Problem ist das Ungleichgewicht zwischen vielen Hintergrundbeispielen und wenigen Objekten
- **Focal Loss** reduziert den Einfluss leicht klassifizierbarer Beispiele und lenkt das Training auf schwierigere Fälle

12.3.2.1. Focal Loss

$$\mathrm{FL}(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

- p_t ist die Wahrscheinlichkeit für die wahre Klasse
- α_t gleicht Klassenungleichgewichte aus
- γ reduziert den Beitrag einfacher Beispiele
- Der Faktor $(1 - p_t)^{\gamma}$ sorgt dafür, dass gut klassifizierte Beispiele weniger stark in den Loss eingehen





12.4. Evaluierung von Object Detection Modellen

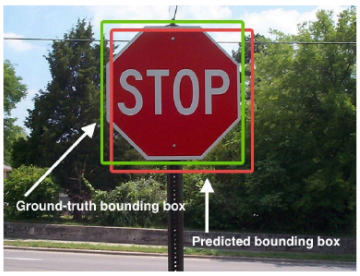
12.4.1. Intersection over Union (IoU)

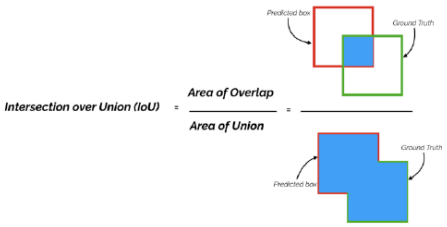
DEFINITION: Intersection over Union (IoU) misst, wie stark sich eine vorhergesagte Bounding Box mit der Ground-Truth-Box überlappt.

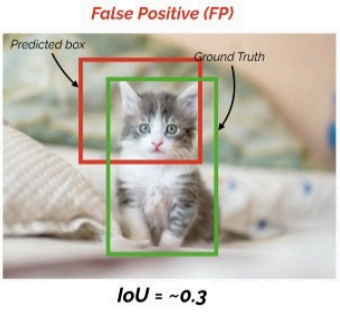
- Die IoU ist definiert als:

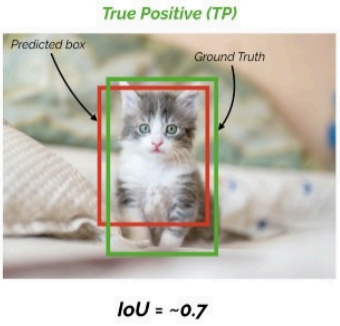
$$\text{IoU} = \frac{\text{Fläche der Überlappung}}{\text{Fläche der Vereinigung}}$$

- Eine hohe IoU bedeutet, dass die vorhergesagte Box gut zur Ground-Truth passt
- Oft wird ein **IoU-Threshold** verwendet, z. B. 0.5:
  - $\text{IoU} \geq 0.5 \longrightarrow$ **True Positive**
  - $\text{IoU} < 0.5 \longrightarrow$ **False Positive**









12.4.2. Precision und Recall

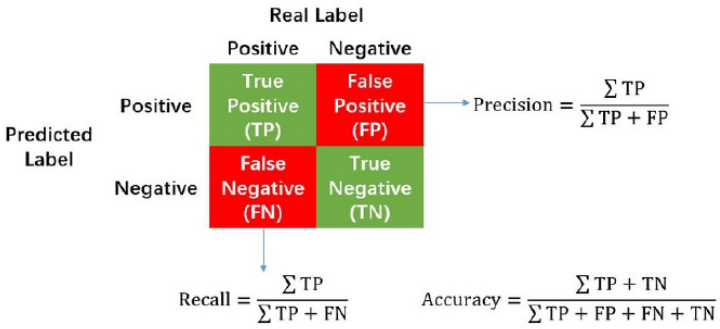
DEFINITION: Precision und Recall beschreiben, wie zuverlässig und wie vollständig ein Modell Objekte erkennt.

- **Precision**:
  - Von allen vorhergesagten Objekten, wie viele sind korrekt?

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall**:
  - Von allen tatsächlich vorhandenen Objekten, wie viele wurden gefunden?

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

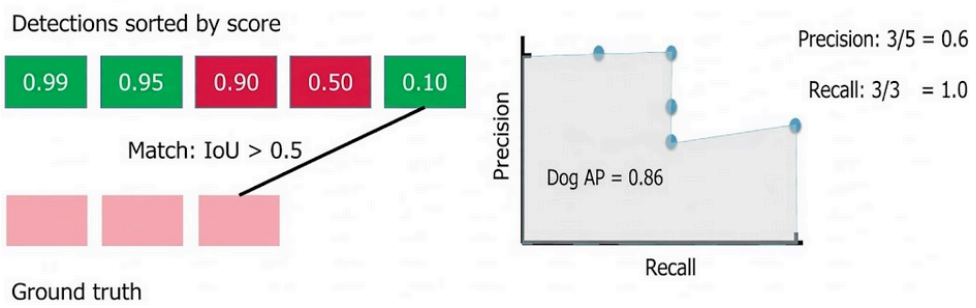


12.4.3. Average Precision (AP)

DEFINITION: Average Precision (AP) ist die Fläche unter der Precision-Recall-Kurve für eine einzelne Klasse.

12.4.3.1. Ablauf

1. Die Vorhersagen werden nach ihrem Score sortiert
2. Jede Vorhersage wird mit der Ground-Truth verglichen
3. Wenn eine Vorhersage zu einer Ground-Truth-Box mit genügend hoher IoU passt, gilt sie als positiv
4. Andernfalls gilt sie als negativ
5. Daraus entstehen Punkte auf der **Precision-Recall-Kurve**
6. Die Fläche unter dieser Kurve ergibt die **Average Precision (AP)** der Klasse



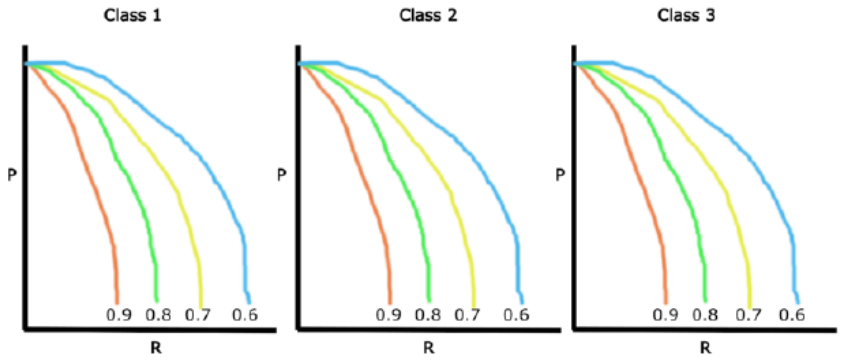
12.4.4. Mean Average Precision (mAP)

DEFINITION: Mean Average Precision (mAP) ist der Mittelwert der Average Precision über alle Klassen.

- Für jede Klasse wird zuerst eine **AP** berechnet
- Danach wird über alle Klassen gemittelt:

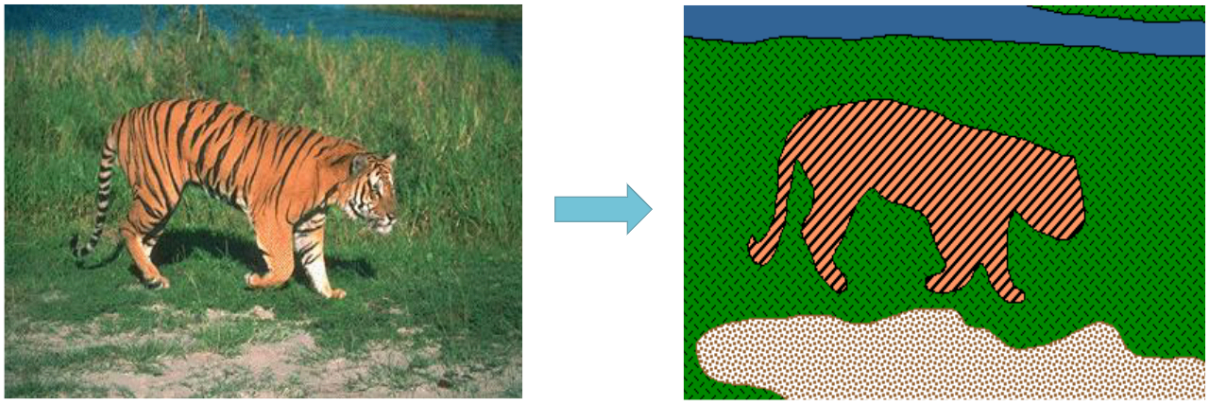
- $\text{mAP} = \frac{1}{n} \sum_{k=1}^{n} \text{AP}_k$

- Dabei ist:
  - AP_k: Average Precision der Klasse k
  - n: Anzahl Klassen
- Häufig wird mAP bei einem festen IoU-Wert angegeben, z. B. **mAP@0.5**



13. Segmentation

DEFINITION: Segementation teil ein Bild auf in klar definierte Zonen welche etwas gemeinsam haben, z.B. die Farbe oder Intensität. Mit Segementierung können auch Objekte in einem Bild herausgelesen werden um diese anschliessend mit „Object Recognition“ zu erkennen.



Einfache Segementation kann mit folgenden Methoden durchgeführt werden:
- Threshholding
- Histogramme
- Edge Detection

13.1. Region Based Segmentation

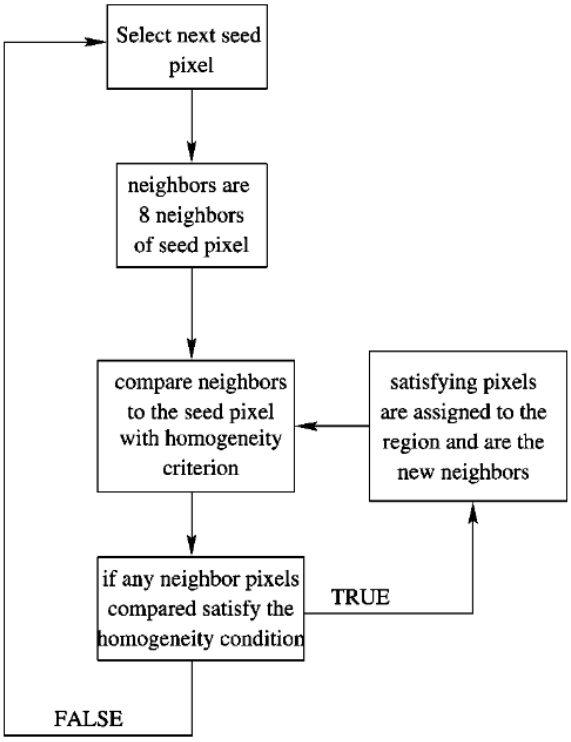
Mit folgenden Methoden kann komplexere Segmentation durchgeführt werden:

- **Klassische Methoden**:
  - Region Growing
  - Split and Merge
  - Watershed
  - Oversegmentation / SLIC
  - Graph-Cuts
- **Unsupervised Learning Methoden**:
  - k-Means Clustering
  - Mean-Shift Algorithm

13.1.1. Region Growing

- Ähnliche Pixel werden rekursiv zu Gruppen zusammengefügt
- Viele verschieden Varianten, je nachdem wie Ähnlichkeit definiert wird
- Kann mathematisch ausgedrückt werden
- Funktioniert auch in 3D (Image-Layers)
- Wird of interaktiv genutzt, da so einfach das Zentrum festgelegt werden kann

Ablauf:



Anwendungsbeispiele:
- Photoshop Magic Wand
- Medizinische Bildern

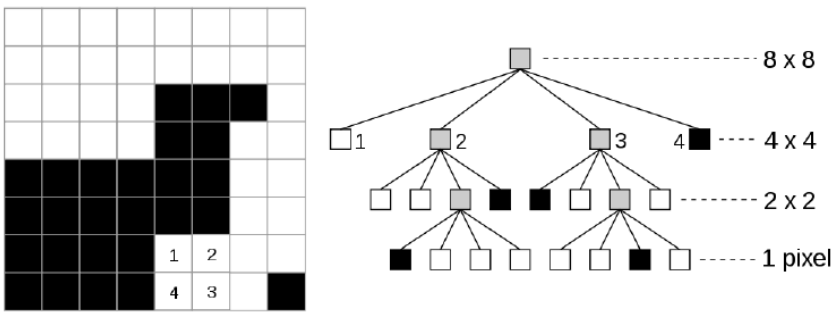
13.1.2. Split and Merge

Idee:

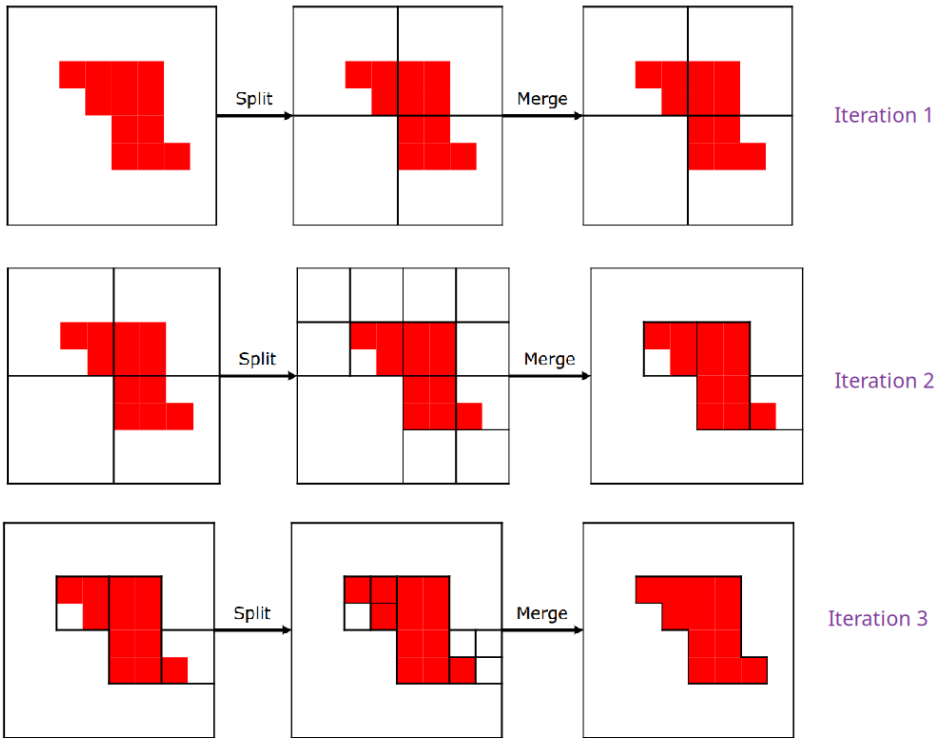
- Nicht uniforme Regionen des Bildes werden rekursiv aufgesplittet (**Split**)
- Benachbarte Regionen werden zusammengefügt (**Merge**)
- **Kriterium definieren**: Wann ist eine Region uniform?

Split:
1. Ganzes Bild ist eine Regionen
2. Aufsplitten in 4 Subregionen
3. Subregionen werden rekursiv gesplittet

-> Implementation über Quadtree



Ablauf:



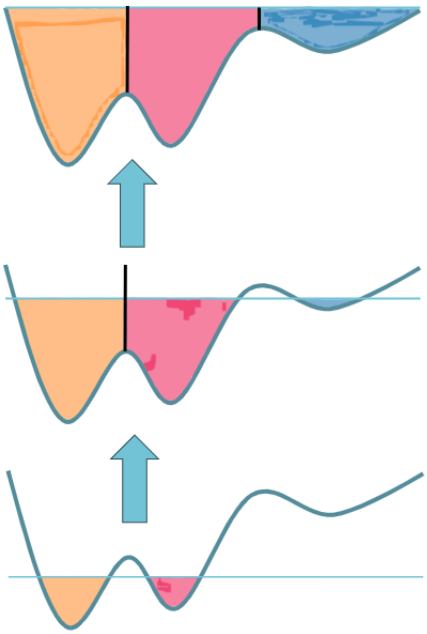
13.2. Watershed

Idee:

- Bild wird als Höhenmap interpretiert
- Bild wird mit Wasser gefüllt
- Linie des Watershed berechnet

Parameter mit welchem die Höhenmap gebildet wird ist frei wählbar (z.B. Intensität)

Prinzip:



13.2.1. k-Means Clustering

Über einen ML Algorithmus werden eine feste Anzahl k Cluster gefunden.

Problem:
- k Cluster in einem n-Dimensionalen Raum finden
- k ist gegeben

Cluster:
- jeder Cluster hat eine Zentrum
- jedes Sample gehört zum Cluster mit dem nächsten Zentrum

Optimale Cluster:
- Summe aller Distanzen von Punkten innerhalb eines Clusters sollten minimal Seitenverhältnis
- NP Problem (**zu grosser Aufwand**)

13.2.1.1. Lloyds Algorithmus

1. k Cluster initialisieren
2. Berechnen welches Sample zu welchem Cluster gehört
3. Zentum jedes Clusters Berechnen
4. Cluster zum neuen Zentrum hin verschieben
5. Punkt 2-4. so lange wiederholen bis sich die Cluster nicht mehr ändern

Beispiel:

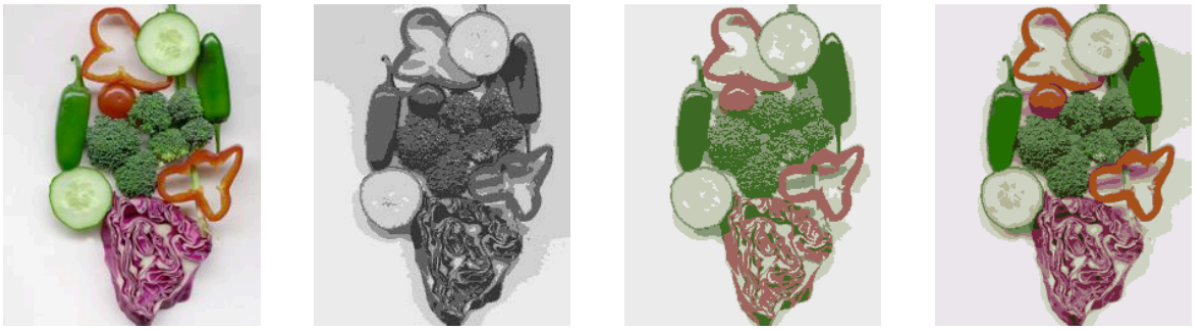


Image | K=5 Cluster on grayvalue image | K=5 Cluster on color image | K=11 Cluster on color image

13.2.1.2. Clustering Attribute

Als Basis für das Clustering können beispielsweise folgende Attribute verwendet werden:
- RGB Werte
- HSV Werte
- HS Werte
- Positionen
- Farben und Positionen

WICHTIG: Werden Werte vermischt wie z.B. Farbe und Positionen, sollten diese Normalisiert werden damit sie sich auf eine vergleichbaren Skala befinden

13.2.1.3. Problem mit K-Means

- Herkömmliche Cluster nach Lloyds sind rund
- Häufig sind reale Cluster aber nicht rund

Lösung: Mean Shift Clustering

13.2.2. Mean Shift Clustering

Idee: Form der Cluster ist variable und nicht definiert über die Methoden

Algorithmus:

- Suchen nach dem Maximum in Feature Density an einer gegebenen Position
- **Cluster**: eine Menge an Positionen welche zum selben Maximum konvergieren
- **Problem**: die Feature Distribution ist nicht direkt verfügbar
- **Lösung**: Kernel Density Estimitation wird für die Verteilung verwendent (z.B. Gauss Kernel)

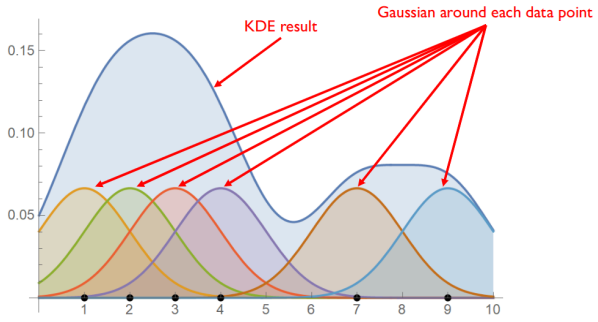
Formel:

$$\hat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Dabei ist K (der Gauss Kernel) definiert als:

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{\left(-\frac{(x-x_i)^2}{2h^2}\right)}$$

Grafische Darstellung einer KDE:



Um das Maximum zu berechnen wird mit Gradient Ascent gearbeitet:

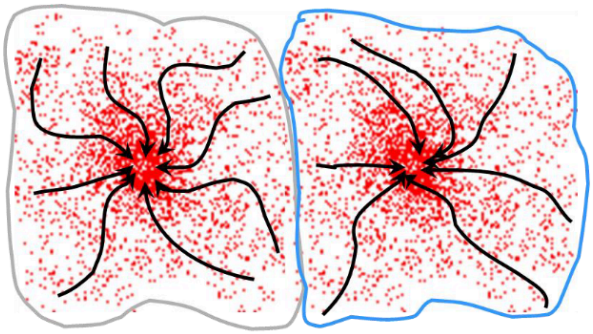
- Mean Shift Vektor $m(x)$ einer Nachbarschaft $N(X)$ wird berechnet
- x wird als mean shift vector $m(x)$ gesetz
- Iterieren bis die Shifts kleiner sind als eine definierter Threshhold

Formel:

$$m(xi) = \frac{\sum_{x_j \in N(x_i)} K(x_j - x_i) x_j}{\sum_{x_j \in N(x_i) K(x_j - x_i)}} - x_i$$

13.2.2.1. Attraction Basins

Als Attraction Basin wird eine Region bezeichnet bei der das Trajectorie (die Bahn) zum selben Modus führen.

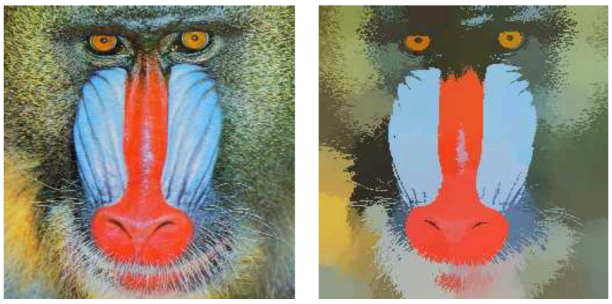


13.2.2.2. Gesamtablauf von Mean Shift Clustering

1. Features berechnen (oft inklusive der Pixel Location)
2. Mean Shift bei jedem Pixel initialisieren
3. Mean Shift durchführen bis er konvergiert
4. Positionen zusammenführen welche zum selben Peak gehören

Der Algorithmus ist sehr aufwändig und dementsprechend langsam, führt aber zu guten Resultaten

Ergebnis von Mean-Shift Clustering:



13.2.3. Graph Cuts

Idee: Bild wird als Graph interpretiert

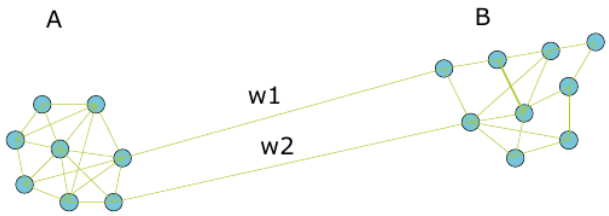
- jeder Pixel ist ein Knoten
- Kanten zwischen Pixeln (z.B. benachbarte Knoten)
- Affinitätgewicht für jede Kante -> beschreibt wie ähnlich die Pixel sind
  - je ähnlicher desto grösser das Gewicht
  - **Beispielswerte**: Farbe & Position
- Für grosse Graphen

Ziel: So durch den Graph zu schneiden dass das Bild segementiert ist

- Graph $G = (V, E)$
- Kante (u, v) hat Gewicht $w(u, v)$ welches die Ähnlichkeit repräsentiert
- Graph G wird in zwei disjunkte Graphen welche aus den Knoten Mengen A und B bestehen zerteilt
- Alle Kanten zwischen A und B werden entfernt

Der dem Cut wird folgendermassen berechnetes value assigned:

$$\text{cut}(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

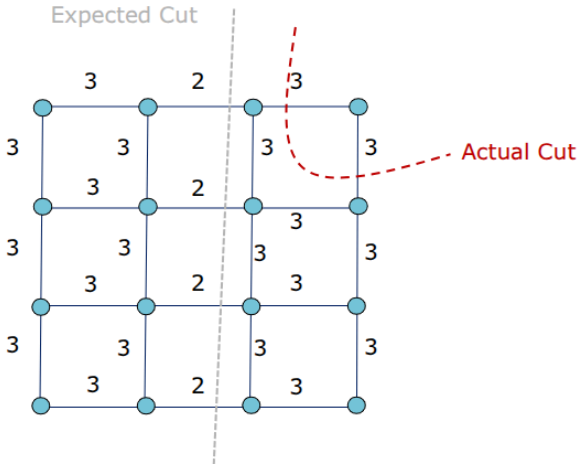


Es bestehen verschiedene Methoden den Cut durchzuführen

13.2.3.1. Min Cut

Cut mit dem minimalten values aussuchen

Problem: Cut mit dem minimalsten Value enthält nur einen Knoten



13.2.3.2. Normalized Cut

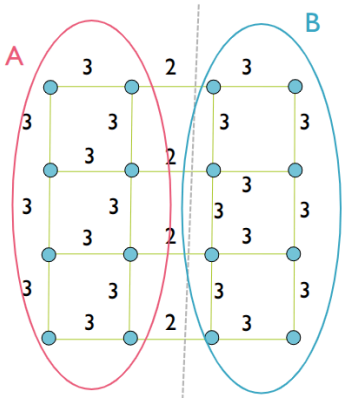
Der Normalized Cut löst das Problem des Min Cut.

$$\text{Ncut}(A,B) = \frac{\text{cut}(A,B)}{\text{assoc}(A,V)} + \frac{\text{cut}(A,B)}{\text{assoc}(B,V)}$$

assoc ist definiert als die Summe der Gewichte aller Kanten welche mit einem Konten in A verbunden sind:

$$\text{assoc}(A,B) = \sum_{u \in A, t \in V} w(u,t)$$

Beispiel:



$$assoc(A,V) = 38$$
$$assoc(B,V) = 38$$
$$cut(A,B) = 8$$
$$Ncut(A,B) = \frac{16}{38}$$

13.2.3.3. Superpixel

Der Normalized Cut ist sehr aufwändig, da er viele Berechnungen erfordert

Um den Aufwand zu reduzieren werden Superpixel erstellt
- Superpixel sind ähnliche Pixel welche gruppiert werden
- Effizient, jedoch entsteht oft das Problem der **Oversegmentation**

Der Normalized Cut wird dann auf den Superpixel durchgeführt

13.2.4. SLIC

Simple Lineare Iterative Clustering ist eine Methode um Superpixel zu berechnen

Idee:
- Superpixel haben keine internen Kanten
- es gibt eine bekannte ungefähren Anzahl Superpixel
- Cluser Zentren werden regelmässig initialisiert (Grid)
- ein lokaler Kluster befindet sich im Raum (L, a, b, x, y)
  - Farbe (L, a, b) und Position (x, y)

Die Distanz wird folgendermassen definiert:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$

$$D_S = d_{lab} + \frac{m}{S} d_x y$$

Hierbei ist $S = \sqrt{\frac{N}{K}}$ wobei N die Grösse des Bildes ist und K die vorbestimmte Anzahl an Superpixeln ist

13.2.4.1. Ablauf von SLIC:

1. Cluster Zentren initialisieren in einem Grid welche die Punkte S Pixel Entfernung haben
2. Cluster Zentren werden in 3x3 Nachbarschaften dorthin verschoben wo der Gradient am niedrigsten ist
   - 3x3 Nachbarschaften um zu verhindern das ein Cluster Zentrum auf einer Kante ist

Berechnung des Gradienten:

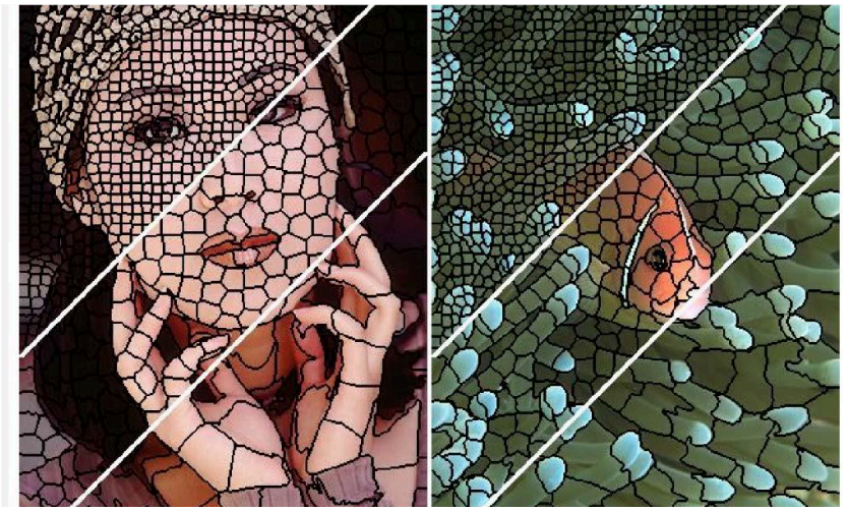
$$G(x,y) = \|I(x+1,y) - I(x-1,y)\|^2 + \|I(x,y+1) - I(x,y-1)\|^2$$

Wiederholen solange bis Fehler E kleiner ist als ein definierter Threshhold:

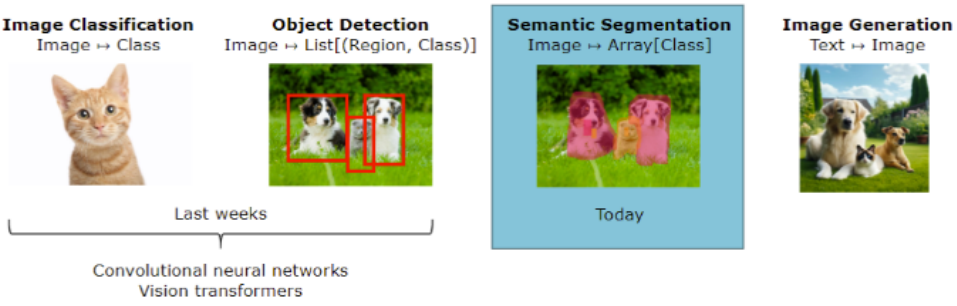
- Für jedes Cluster Zentrum:
  - Für jeden Pixel in einer 2Sx2S Nachbarschaft
  - Weise den Pixel dem nahesten Cluster Zentrum zu
- Update Cluster Zentren
- Berechnen Fehler E als Differenz zwischen dem alten und dem neuen Cluster Zentrum

Am Schluss werden verwaiste Pixel noch verarbeitet mit Connected Component Analysis um sicherzustellen das alle Cluster verbunden sind

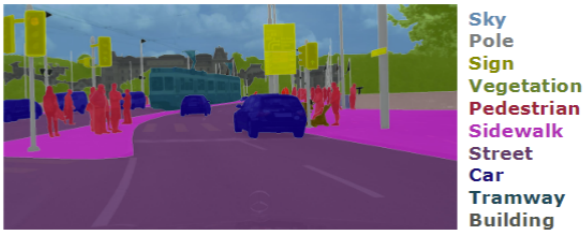
Beispiel SLIC:



14. Semantic Segmentation



Semantic Segmentation: man erhält eine Klasse pro Pixel



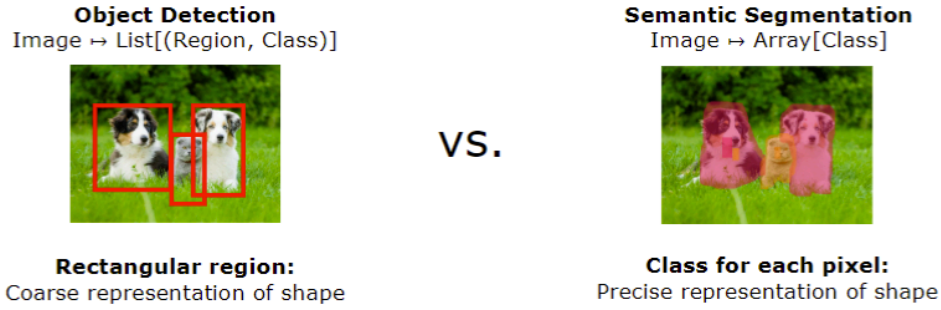
Classes: Each region corresponds to one class

Anwendungen:

- Robotik (Bsp. zum Greifen von Objekten)
- Selbstfahrende Autos (Bsp. Erkennung von Fahrspurkrümmungen)
- Medizin (Bsp. Analyse von Blutgefäßen)

Segmentation vs. Semantic Segmentation

- Segmentation: definiert relevante Bildregionen, z.B. Hintergrund / Vordergrund, verschiedene Farbregionen
- Semantic Segmentation: Klasse für jedes Pixel



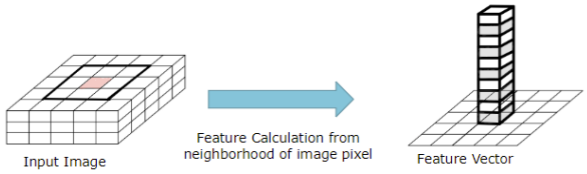
14.1. Classical Approaches

Früher wurden Merkmale (Features) manuell definiert, um Pixel zu klassifizieren. Ähnlich wie klassische Ansätze in der Bildklassifikation [[CVAI - 05.1 - Image Classification]]:

- **Local Binary Patterns (LBP):** Analysiert die Nachbarschaft eines Pixels und weist einen Binärcode zu.
- **Grey Level Co-occurrence Matrices (GLCM):** Analysiert zusätzlich die Häufigkeit der Grauwerte
- **Filter Banks & Textons:** Texturen werden durch statistische Verteilungen von Filterantworten beschrieben.
- Ein **Klassifikator** (wie eine **SVM**) entscheidet dann basierend auf diesen Merkmalen über die Klasse des Pixels.

Semantic segmentation **Merkmale**:

- Features sollen von einem **lokalen Nachbarn** berechnet werden
  - die Werte eines einzelnen Pixels helfen nicht um diesem Pixel eine Klasse zuzuweisen
  - anhand der Nachbarn kann gelernt werden, wie sich das Pixel verhält
- Häufig führt die semantische Segmentierung dazu, dass Materialien oder Texturen segmentiert werden.



Welche Eigenschaften sollten Merkmale aufweisen? Invarianz gegenüber:

- Translation (in der Regel durch Filter vorgegeben)
- Rotation (je nach Aufgabe)
- Skalierung
- Änderungen der Beleuchtung (Farbe)
- ...

14.1.1. Local Binary Patterns (LBP)

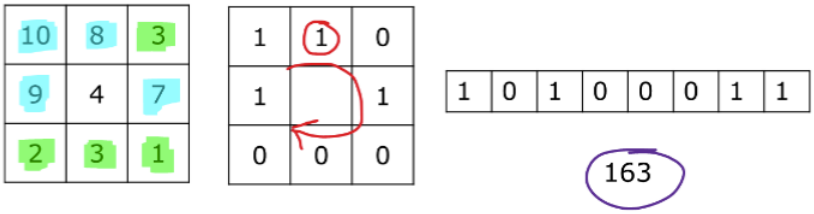
Idee:

- Analyse der Nachbarschaft eines Pixels (typisch in einem 3 x 3 Block)
- Kontrast (Helligkeitsunterschiede) zwischen seinen Pixel und der Nachbarschaft in einer einfachen Zahl zusammenzufassen

Ziel: lokales Muster erkennen

Vorgehen:

- Grauwert eines Pixels mit den acht Nachbarn vergleichen (bei einem 3x3 Block)
  - Nachbarn >= Zentrum → 1
  - Nachbarn < Zentrum → 0
- Binärcode erstellen: Einsen und Nullen in einer bestimmen Reihenfolge, z.B. Uhrzeigersinn als binär Zahl zusammenreihen
- LBT Code erstellen: Dezimalzahl vom binär Code



Vorteile:

- **Beleuchtungsinvarianz**: Wenn das Bild heller oder dunkler wird, bleibt das Ergebnis gleich (greyscale invariance)
- **Effizienz**: Berechnung ist schnell

- **Texturerkennung**: Unterschiedliche LBP-Werte stehen für unterschiedliche Mikrostrukturen wie Kanten, Ecken, glatte Flächen oder Linien.

14.1.1.1. Erweiterungen

Methoden robuster gegenüber von Veränderungen in der Bildgrösse, Rotation und Beleuchtung machen.

14.1.1.1.1. Rotationsinvarianz

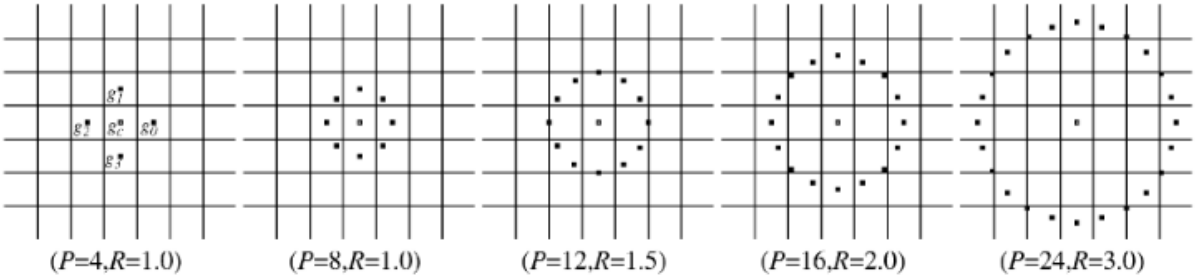
Problem: Wenn sich das Bild dreht, ändert der Binärcode

Lösung: Binärcode wird rotiert, bis der kleinstmögliche Zahlenwert erreicht ist

Beispiel: Ein Muster `00000011` (Wert 3) wäre bei einer Drehung vielleicht `11000000` (Wert 192). Durch das Rotieren werden beide zum selben „minimalen" Wert (3) zusammengefasst. So erkennt das System die Textur, egal ob sie waagerecht oder schräg im Bild liegt.

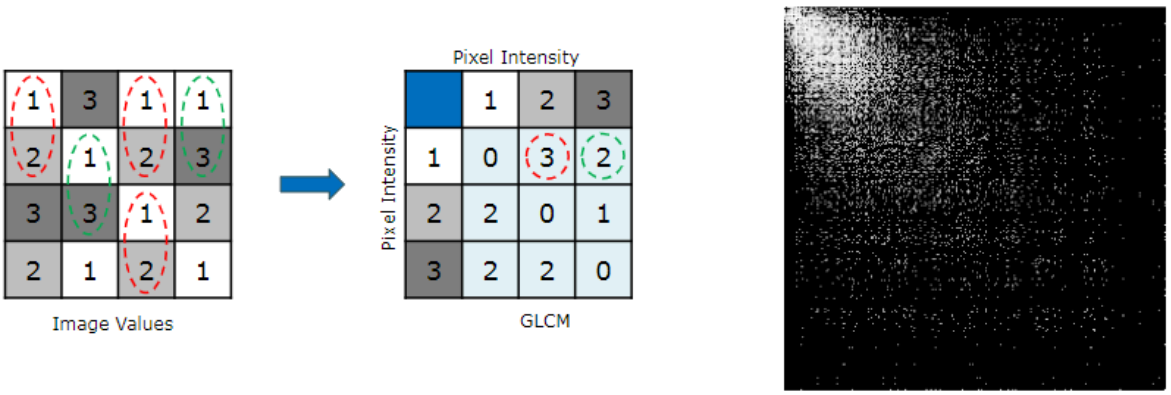
14.1.1.1.2. Multiresolution

- kreisförmige Nachbarschaft nutzen (statt Block)
- Möglichkeit verschiedene Radien zu erfassen



14.1.2. Grey Level Co-occurrence Matrices (GLCM)

- Zusätzlich zu LBP analysiert GLCM die **Häufigkeit** der Grauwert-Kombinationen
- Resultat: 256x256 Matrix (für 256 grey values)
- Aus der Matrix lassen sich weitere Merkmale berechnen (Entropie, Energie, Homogenität, Kontrast, Dissimilarität)



- Pixelwert 1 neben 2 kommt 3 mal vor
- Pixelwert 1 neben 3 kommt 2 mal vor

14.1.3. Filter Banks

galt vor den Neuronalennetzwerken als bester Ansatz

Idee: Eine ganze **Sammlung von Filtern** (die Bank) wird auf das Bild angewandt

Ziel: Komplexität von Texturen erfassen zu können

Vorgehen:

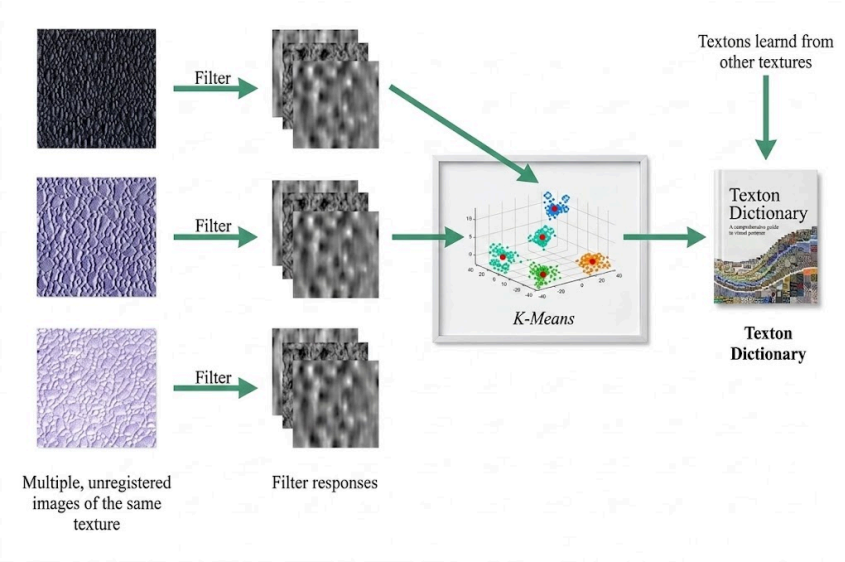
- **Convolution**: jeder Filter wird über das Bild geschoben
- **Merkmalsvektor**: Resultat alles Filter bildet den Merkmalsvektor
- **Klassifizierung**: Ein Klassifikator (SVM, Random Forest) lernt nun die Klassifizierung

14.1.3.1. Texture Characteristics - Textons

- Weiterführung von Filter Banks
- durch z.B. K-means Clustering werden **Muster-Bausteine** → **Textons** gefunden
- ergibt ein **Wörterbuch von Texturen**
  - Bild wird nicht mehr durch Filterwerte beschrieben, sondern dadurch, welche Textons an welcher Stelle vorkommen

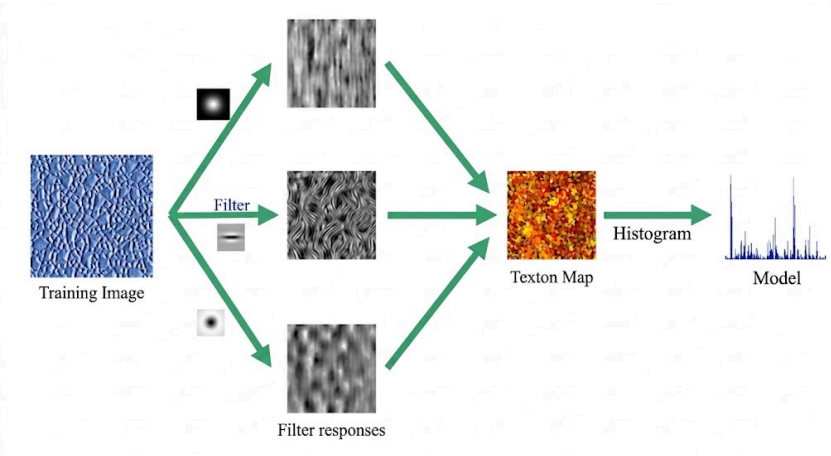
Learning State I:

- Texton Wörterbuch durch Clustering erstellen

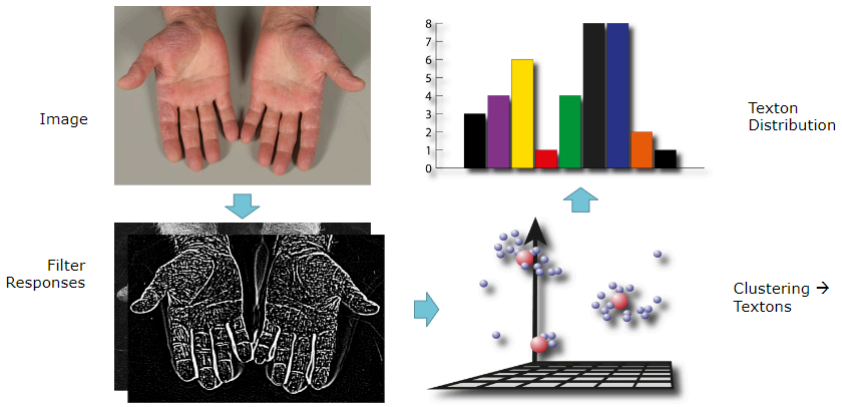


Learning State II: Modell Generierung

- Zuordnung jeder Filterantwort zum nächstgelegenen Texton
- Erstellung eines Histogramms der Textons



Beispiel:



14.2. Metrics

true-positive: korrekt positiv **false-positive**: falsch positive **true-negative**: korrekt negativ **false-negative**: falsch negativ

Accuracy: Welcher Anteil alle Pixel korrekt klassifiziert wurde

Precision: Wenn das Modell eine Klasse vorhersagt, wie oft hat es dann recht?„ Eine hohe Precision bedeutet, dass das Modell wenig „Falsch-Positive“ Fehler macht.

Recall: Wie viel von der tatsächlichen Fläche hat das Modell gefunden?„ Ein hoher Recall bedeutet, dass das Modell fast alles von dem Objekt erkannt hat (wenig „False Negatives“).

F1-Score: Kombiniert Precision und Recall, guter Indikator für Gesamtqualität

IoU (Intersection over Union / Jaccard-Index): Sie beschreibt das Verhältnis der **Schnittmenge** zur **Vereinigungsmenge** zwischen der Vorhersage und dem tatsächlichen Label (Ground Truth).

- **IoU = 1:** Perfekte Übereinstimmung.
- **IoU > 0.5:** Wird oft als gute Vorhersage gewertet.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

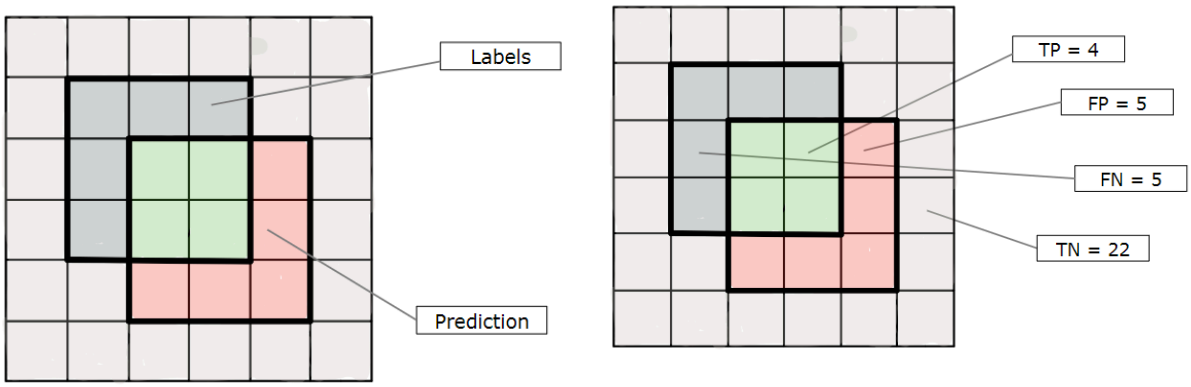
$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{F1} = \frac{2TP}{2TP + FP + FN}$$

$$\text{IoU} = \frac{TP}{TP + FP + FN}$$

14.2.1. Beispiel



- Accuracy = 0.72
- Precision = 0.44
- Recall = 0.44
- F1 Score = 0.44
- IoU = 0.28

14.3. Convolutional Neural Network for Semantic Segmentation

Klassische Ansätze funktionieren bei der semantischen Segmentierung recht gut, aber die Merkmalserstellung ist mühsam und fehleranfällig.

14.3.1. Problem mit CNN für semantic segmentation

Rezeptiven Feldes (ohne Pooling)

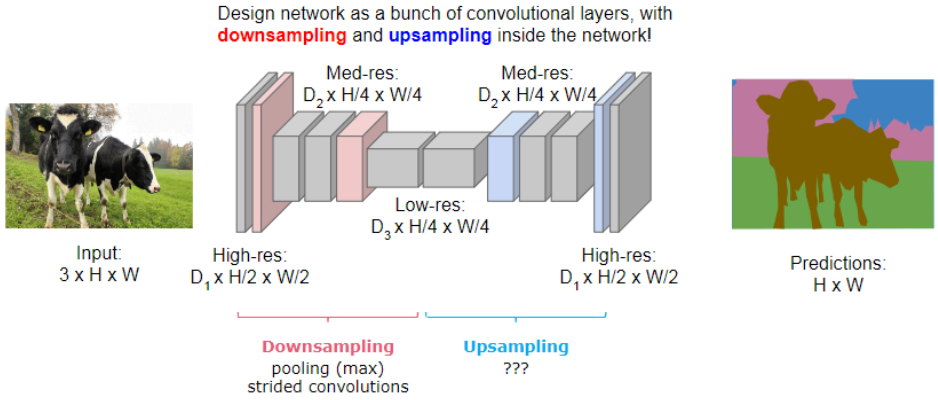
- bei der Convolution braucht man extrem viele Schichten
- das macht das Netzwerk tief und langsam

Informationsverlust durch Downsampmling

- in klassischer Bildverarbeitung: Pooling um Datenmenge zu reduzieren, damit das rezeptive Feld schnell gross wird
- bei der Segmentierung, muss das Ergebnis für jeden einzelnen Pixel berechnet werden
- durch Downsampling gehen räumliche Details verloren

Idee: pooling (max) und Upsampling

- Max Pooling: Max Pooling

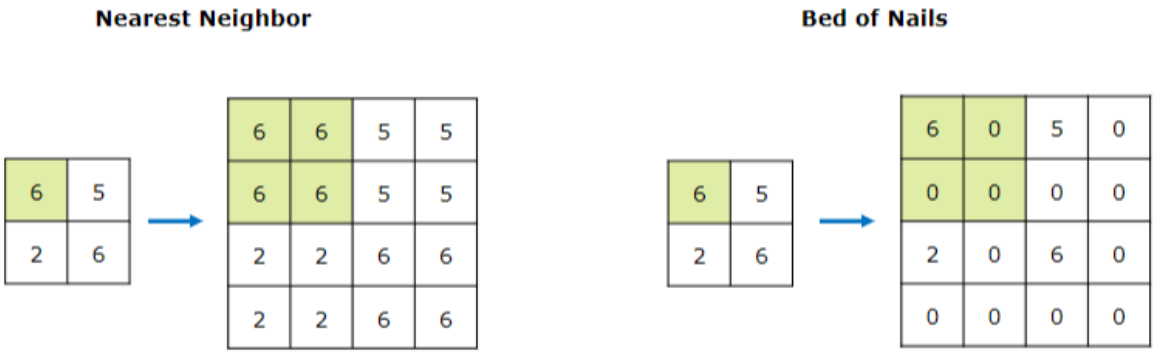


14.3.2. Upsampling

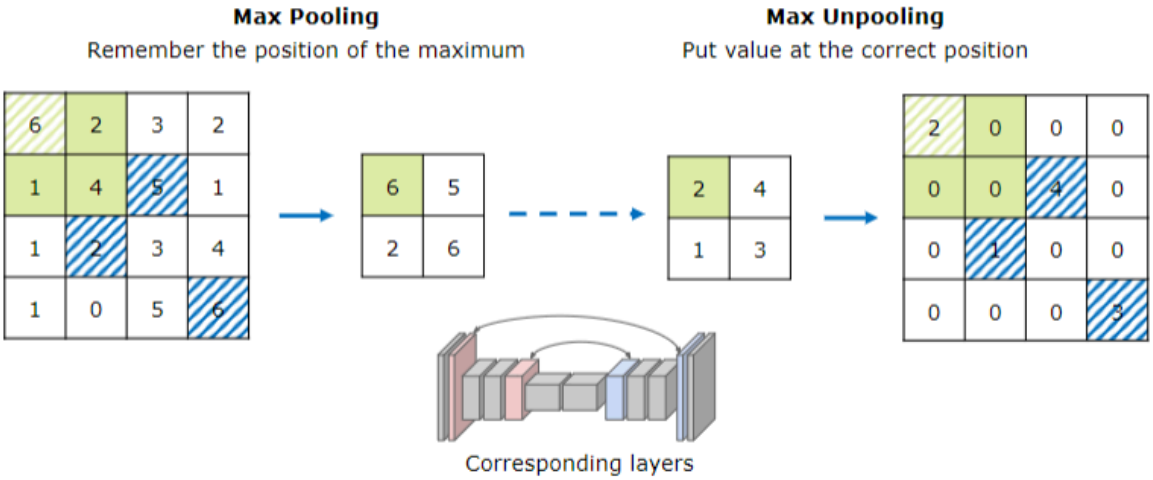
Nearest Neighbor:

- sehr einfach
- gleiches Pixel für alle Nachbarn verwenden
- es entstehen Blöcke

Bed of Nails:
- Pixel ist immer oben links, die Nachbarn werden mit 0 aufgefüllt
- Bild sieht aus wie ein Nagelbrett



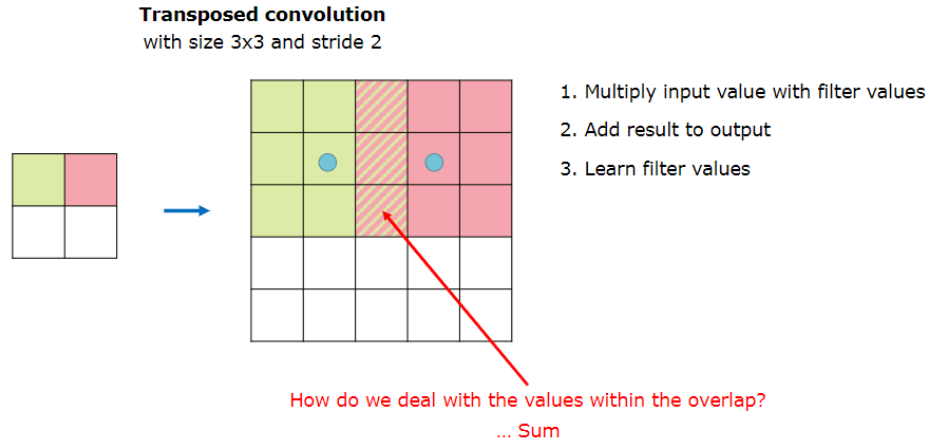
Max Unpooling:
- Intelligenter
- beim Max-Pooling merkt man sich wo (an welchem Index) der maximale Wert lag → Pooling Indices / switches
- dadurch wird der Wert an die richtige Position gesetzt
- Vorteil: die Positionen von Kanten und feinen Strukturen viel besser erhalten

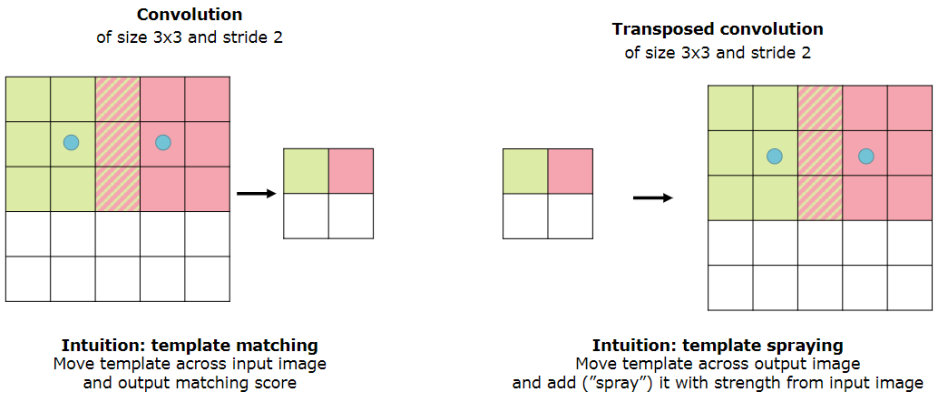


14.3.3. Strided and Transposed Convolutions
lernbare Faltungen

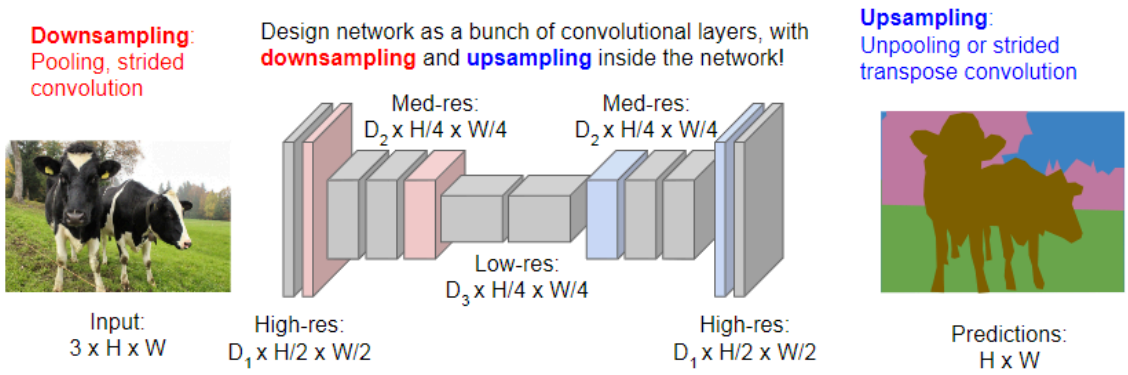
14.3.3.1. Strided Convolution (Runterrechnen)
- Filter springt mehr als nur ein Pixel
- [[CVAI - 05.2 - Convolutional Neural Networks for Image Classification==Downsampling - Strided convolutions]]

14.3.3.2. Transposed Convolution (Hochrechnen)
Vorgehen:
- Man nimmt einen einzelnen Pixelwert aus der kleinen Eingabe-Map.
- Dieser Wert wird mit einem lernbaren Filter (Kernel) multipliziert.
- Das Ergebnis ist ein ganzer Block (z. B. 3x3) im grösseren Ausgabebild.
- Dort, wo sich die ausgegebenen Blöcke überlappen, werden die Werte einfach addiert.





14.4. Fully Convolutional Networks (FCN)



Problem: Der Teil mit niedriger Auflösung enthält nützliche allgemeine Informationen, allerdings mit geringer räumlicher Auflösung. → U-Net Architektur

14.4.1. U-Net Architektur

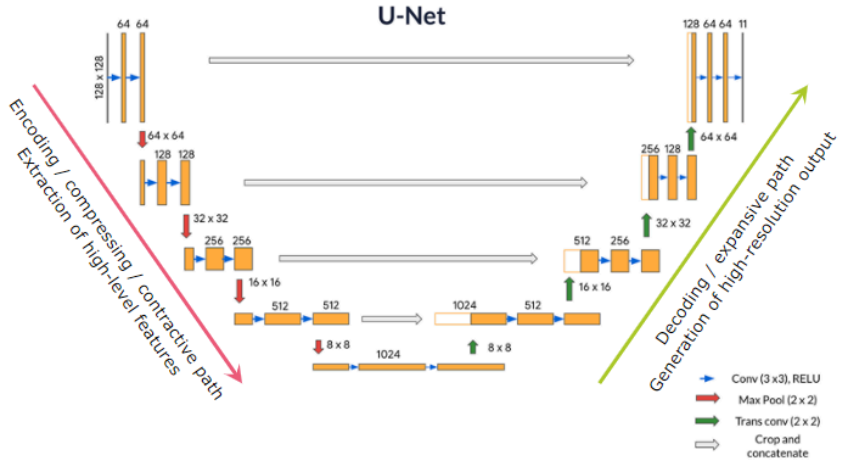
DEFINITION: Das U-Net kombiniert einen **Encoder** (Kontext-Gewinnung) mit einem **symmetrischen Decoder** (Lokalisierung) und verbindet beide durch **Skip Connections**, um verlorene räumliche Details für die finale Maske zurückzuholen.

Struktur:
- linke Seite: Encoding
- rechte Seite: Decoding

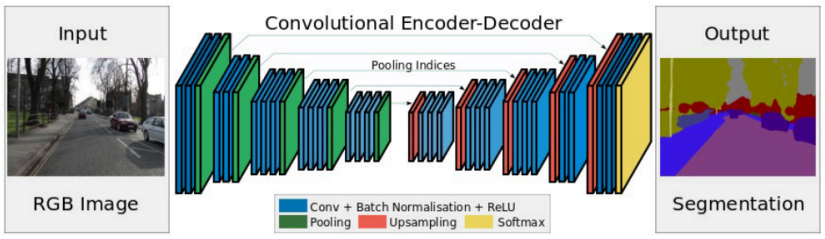
Shortcut connections:

- Concatenation: Merkmale von der linken Seite werden direkt auf die rechte Seite kopiert

Vorteile:
- Präzision: auch feine Strukturen werden exakt segmentiert
- Effizienz: mit relativ wenigen Trainingsbildern, sehr gute Ergebniss



14.4.2. SegNet



- effiziente Architektur mit geringem Speicherbedarf
- nutz Max-Unpooling und strided convolutions

Problem:
- viele Details gehen verloren
- grosse receptive fields (200x200 pixels und mehr)

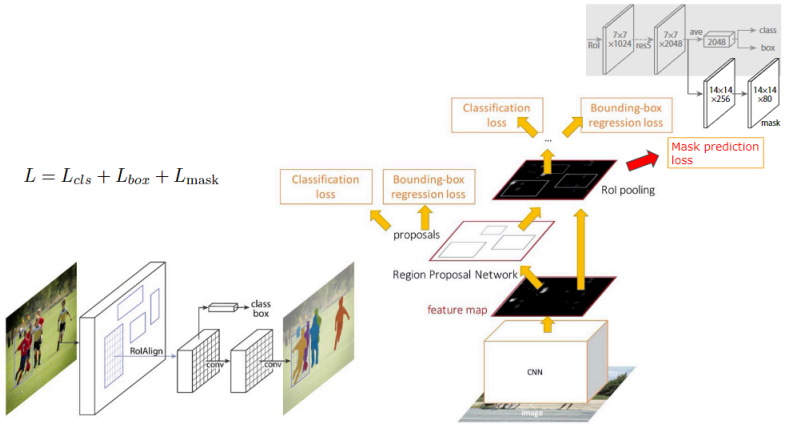
Idee: Training on Batch of Patches

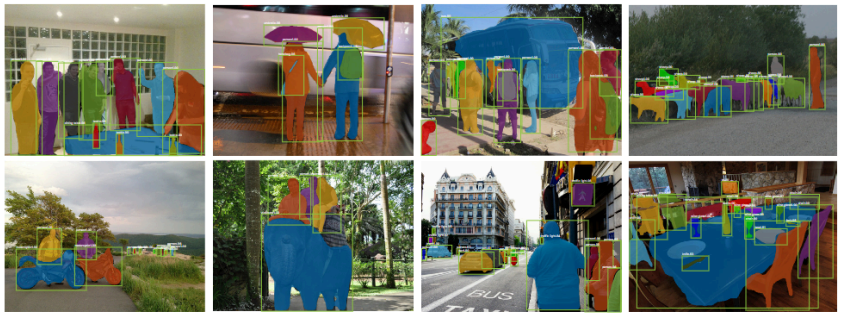
14.4.3. Training on Batch of Patches
- Bild wird in viele kleine Ausschnitte (**Patches**) aufgeteilt
- viele Patches (auch von unterschiedlichen Bildern) werden zu **Batches** zusammengefasst
- das macht das Training stabiler

14.5. Instance Segmentation - Mask R-CNN
Semantic vs. Instance Segmentation
- Semantic Segmentation: Objekte einer Klasse werden als Einheit betrachtet
- Instance Segmentation: jedes einzelne Objekt wird als individuelle Instanz erkannt

Funktion:
- basiert auf Faster R-CNN

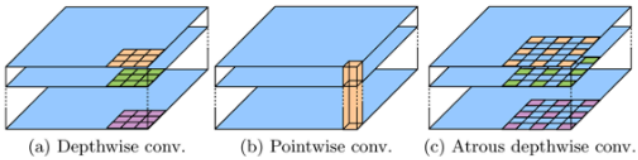
Vorgehen:
- Region Proposal: sucht nach Objekten im Bild (Bounding Box)
- Klassifizierung der Region
- mask-branch: für jedes Pixel innerhalb der Box wird entschieden ob es zum Objekt gehört oder nicht

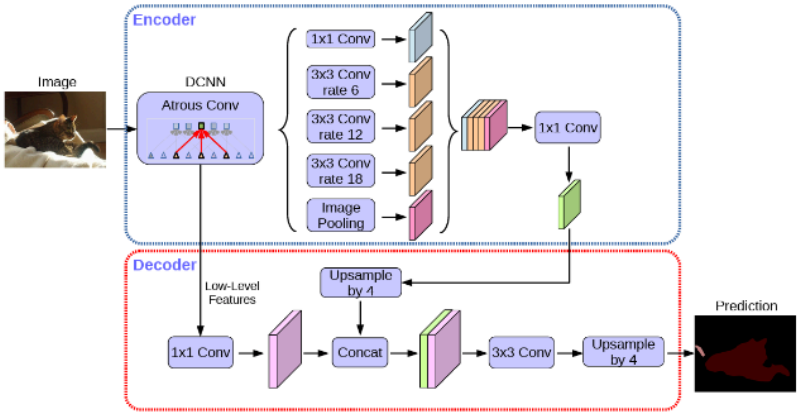




14.6. DeepLab V3+

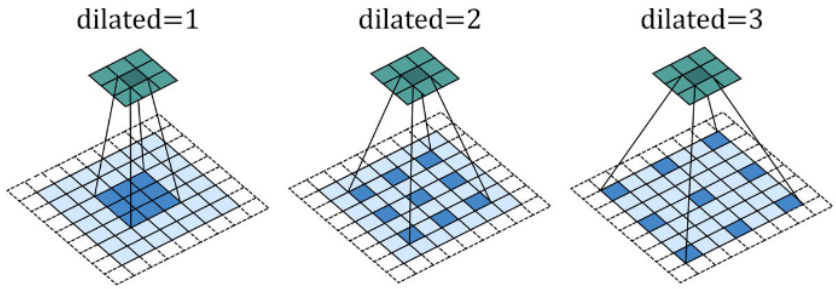
- Weiterentwicklung von FCN
- Merkmale
  - **Encoder-Decoder Architektur**
  - **Atrous (dilated) convolution**
  - **ASPP** (Atrous Spatial Pyramid Pooling)
    - mehrere Atrous Faltungen werden mit unterschiedlichen Raten ombiniert, um Kontextinformationen auf verschiedenen Skalen zu erfassen.
  - **End-to-end trainierbar**





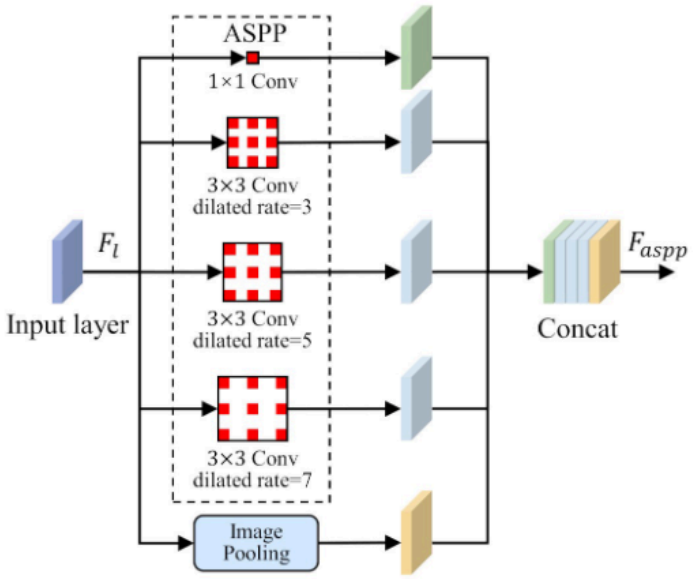
14.6.1. Atrous (dilated) convolution

Vergrösserung des rezeptiven Feldes eines Faltungsfilters durch Überspringen einiger Pixel



14.6.2. Atrous Spatial Pyramid Pooling (ASPP)

- Mehrere dilatierte Faltungsoperatoren mit unterschiedlichen Dilatationsraten
  - Kontextinformationen auf verschiedenen Skalen
- Zusammenführung der Ergebnisse zur Gewinnung reichhaltigerer multiskaliger Merkmalsdarstellungen



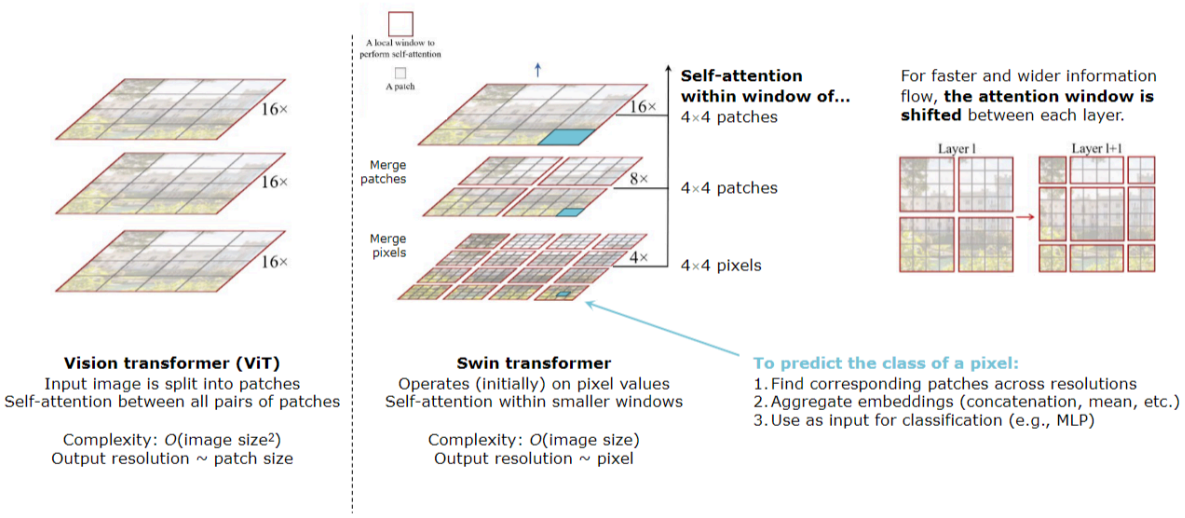
14.7. Vision Transformer (ViT)

Probleme bei der semantic segmentation mit ViT:

- Berechnung der Self-Attention ist extrem aufwendig
- ViT arbeite daher meistens auf **Patches** statt Pixel
- Ausgabe von ViT sind Embeddings
  - haben keine Entsprechung zu einzelnen Pixel
  - nicht passend für Segmentierung

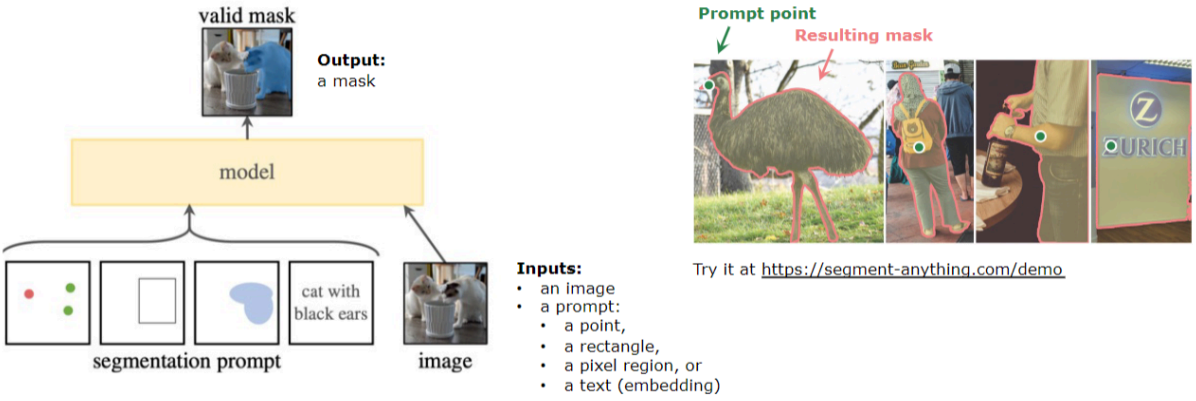
Lösung: **Swin Transformer**

- Hierarchisches Design
- Shifted Windows: self Attention wird innerhalb kleiner Fenster berechnet
- Hierarchie: führt Pixel und Patches schrittweise zusammen

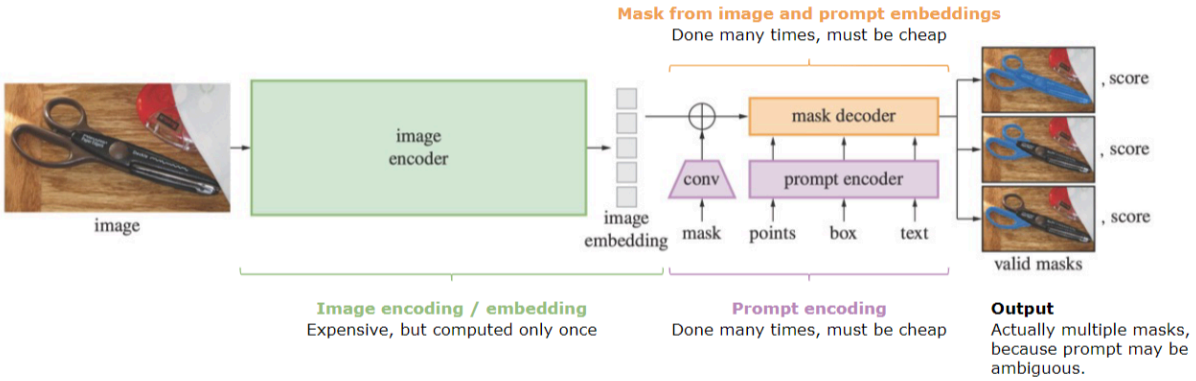


14.8. Segment Anything (SAM)

- aktuellstes Modell von Meta (2023)
- prompt-basiert
- keine echte semantische Segmentierung im klassischen Sinne
  - returniert nur eine mask für ein Objekt
- um jedes Pixel zu klassifizieren, muss das model oft das Bild durchlaufen



Idee: Aufwendige encoding nur einmal durchführen, embeddings daraus für alle prompts verwenden



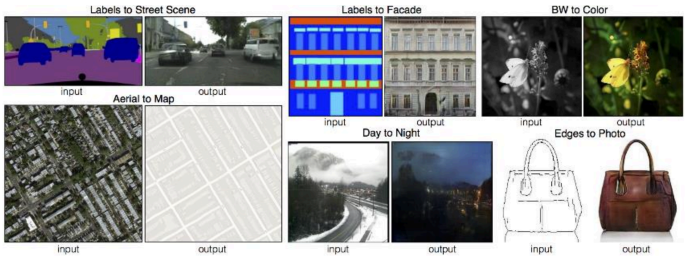
15. Image Generation

15.1. Anwendungen

Generierung eines Bildes anhand eines Prompts

Objektentfernung / Bildreparatur (Inpainting):
- Störende Elemente aus einem Bild löschen
- KI füllt entstandenes „Loch“ nahtlos

Bild-zu-Bild-Übersetzung (Image-to-Image Translation):
- Skizzen zu Fotos
- Kolorierung: Schwarz-Weiss-Fotos automatisch und realistisch einfärben.
- Kontextwechsel:
  - Tageszeit eines Bildes ändern (Tag zu Nacht)
  - Satellitenbilder in klassische Strassenkarten umwandeln
- Layouts zu Fotos



15.1.1. Representation Learning

Grundkonzept: Generative KI lernt ausschliesslich die Repräsentation **gesunder/normaler** Gehirnanatomie

Anomalieerkennung: Ein krankhaftes Bild wird durch das Modell „normalisiert“ (restauriert).

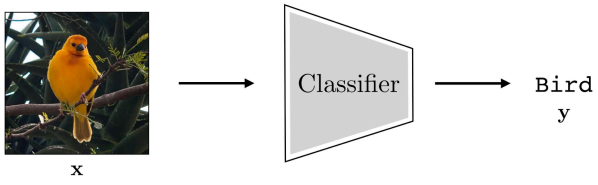
Detektion: Die Differenz zwischen dem ursprünglichen krankhaften Input und der gesunden Restauration macht die Krankheit in einer **Anomaly Map** sichtbar.

15.2. Discriminative Models

DEFINITION: Unterscheidung und Kategorisierung von Daten.

Lernen: Lernen der Wahrscheinlichkeitsverteilung $p(y|x)$ (Mapping von Input zu Label).

Many-to-one: viele verschiedene Bilder führen zu einer Klassifizierung, z. B. „Vogel“.



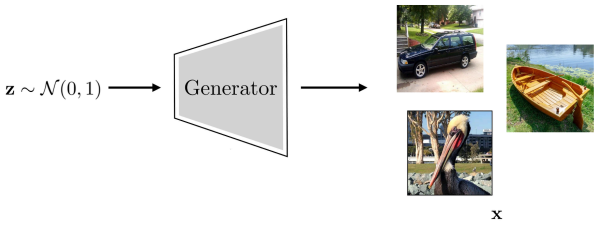
Ziel: Unterscheidung und Kategorisierung von Daten.

15.3. Generative Models

DEFINITION: Verstehen, wie Daten verteilt sind, um neue, ähnliche Instanzen zu erzeugen.

Lernen: Lernen der zugrunde liegenden Datenverteilung $p(x)$ ohne Labels.

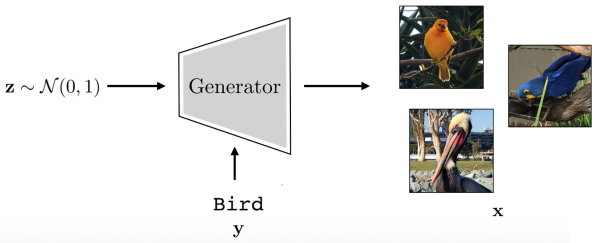
One-to-many: aus einem latenten Rauschen z entstehen vielfältige neue Daten wie Autos oder Boote.



15.4. Conditional Generative Models

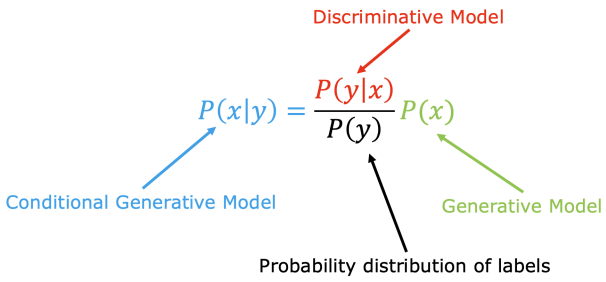
DEFINITION: Ein Modell, welches Daten mit einem zusätzlichen Input y generiert.

- Lernen mit zusätzlicher Information y der bedingten Verteilung $p(x|y)$
- Kontrolle über die Generierung (z. B. über Label y Vogel)



15.5. Bayes' Rule

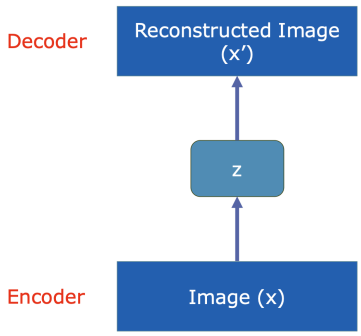
Anhand von Satz von Bayes: Theoretisch möglich Conditional Generative Model aus Discriminative Model & Generative Model zusammenzusetzen



In der Praxis aber nicht wirklich verwendet.

15.6. Auto Encoders

DEFINITION: Neuronales Netzwerk, das die Eingabedaten in einer komprimierten Form darstellt z und anschliessend möglichst genau rekonstruiert x'.



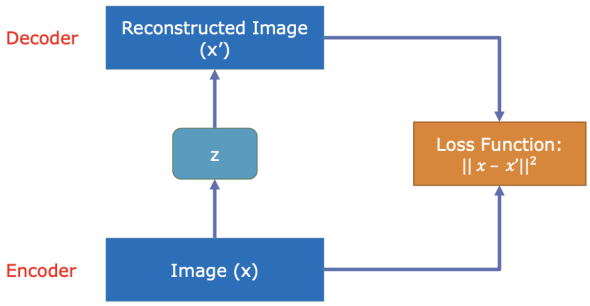
Encoder:
- komprimiert Eingabedaten (x)
- Früher: lineare Schichten
- Heute: CNNs mit ReLU

z:
- Bild in komprimierter Form
- Vektor

Decoder:
- erstellt aus der komprimierten Darstellung z das ursprüngliche Bild x'
- früher: lineare Schichten
- heute: Upscaling CNNs

Bottleneck:
- Da $z < x$, erzwingt das Modell eine **Dimensionsreduktion** und filtert Rauschen/unwichtige Details heraus.

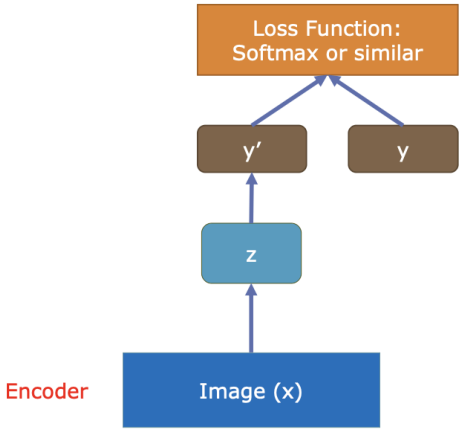
15.6.1. Training



- Modell so trainieren, dass das Bild von z **reproduziert** werden kann.
- **Unsupervised Learning**
- Training mit **Loss Function** (Mean Squared Error)
  - Loss $= \|x - x'\|^2$
- **Backpropagation**: Vergleich zwischen Original und Rekonstruktion als Fehlersignal.
- **Vorteil**: Es braucht keine Labels (Modell lernt aus der Struktur der Daten).

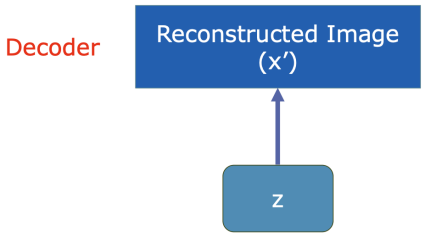
15.6.2. Representation Learning

- Wenn das Modell gut trainiert ist, ist z eine gute Repräsentation des Bildes x.
- Nach dem Training wird der Decoder verworfen.
- Encoder kann als Basis für andere Aufgaben (z. B. Klassifikation) verwendet werden.
  - So braucht man weniger gelabelte Daten.



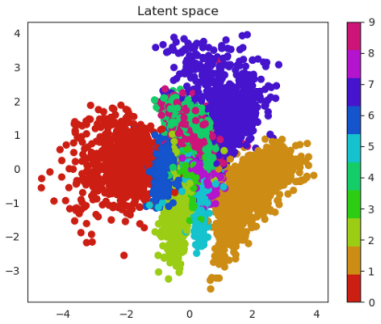
15.6.3. Generating new samples - Problem

Idee: Mit z und dem Decoder Bilder generieren (Stichproben aus dem latenten Raum z direkt in den Decoder einspeisen).



Problem mit autoencoders: latente Raum z hat ist oft unregelmässig und hat Lücken (Generierung von neuen, sinnvollen Bilder ist dadurch schwierig), man weiss nicht was herauskommt

- ähnliche Bilder können auf unterschiedlichen Codes (Vektoren) gemappt werden
- ähnlicher Code kann zu unterschiedlichen Bildern führen



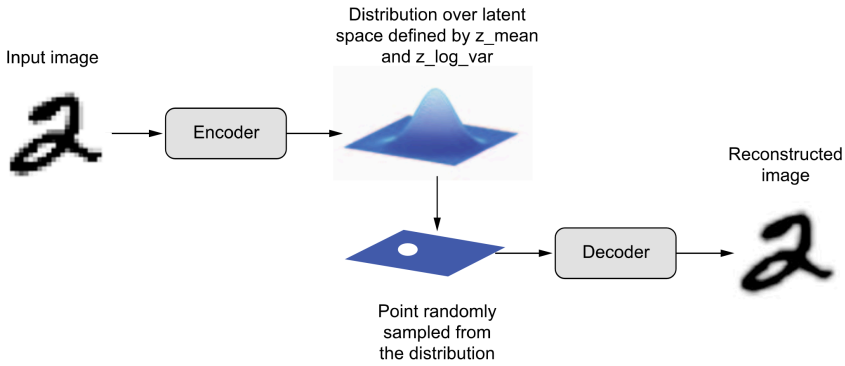
- **Lücken & Sprünge (Gaps):** interpolieren zwischen zwei Punkten nicht möglich.
- (ein Punkt zwischen „7“ und „1“ ergibt kein sinnvolles Bild).

→ Nicht gut geeignet für Generierung
Lösung VAE

Auto Encoders sind aber für anomaly defectors gut geeignet.

15.7. Variational Autoencoders - VAE

DEFINITION: Der Encoder lernt keinen statischen Vektor, sondern beschreibt den latenten Space durch eine **Wahrscheinlichkeitsverteilung** (Mittelwert μ und Varianz σ^2).

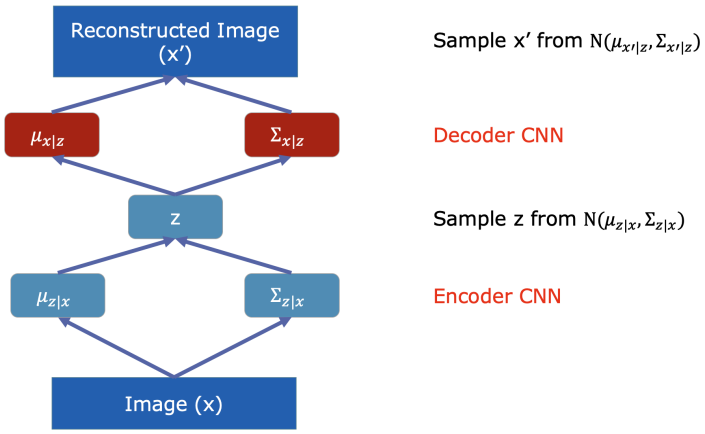


- **Sampling:** zufälliger Punkt z wird an den Decoder übergeben
- latente Raum z wird glatter und strukturierter

→ Besser für Image Generation geeignet

Autoencoder vs. Variational autoencoder: Strukturierung des latenten Raums (z)
- AE: Eingabebild wird auf einzelne Punkte (Vektor) abgebildet
  - Problem: Raum kann Lücken haben
- VAE: Eingabebild wird auf eine Wahrscheinlichkeitsverteilungen (Gauss-Verteilung mit Mittelwert μ und Varianz σ)
  - latente Raum wird glatt und strukturiert

15.7.1. Probability Distributions (Wahrscheinlichkeitsverteilung)



- **Encoder-Output:** Schätzt Mittelwert ($\mu_{z|x}$) und Varianz ($\Sigma_{z|x}$), um den latenten Punkt z aus einer Normalverteilung $\mathcal{N}$ zu sampeln.
- **Decoder-Output:** Erzeugt nicht direkt Pixel, sondern ebenfalls Parameter ($\mu_{x|z}$, $\Sigma_{x|z}$) einer Verteilung, aus der das finale Bild x' generiert wird.

15.7.2. Training

15.7.2.1. Integral Problem

Ziel: Maximierung der **Likelihood-Funktion** $p_{\theta(x)}$, um die optimalen Modellparameter zu finden.

Das Integral-Problem: Die direkte Berechnung von:

$$p_{\theta(x)} = \int p_{\theta(z)} \; p_{\theta(x \mid z)} \; dz$$

ist nicht lösbar da der latente Raum z zu hochdimensional ist.

15.7.2.2. Bayes Problem

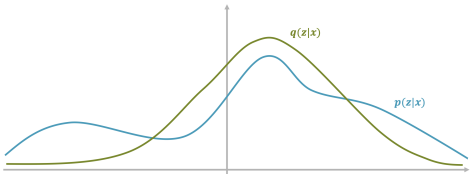
Bayes-Ansatz: Die Umformulierung mittels Bayes-Rule führt zu $p_{\theta(z|x)}$.

$$p_{\theta(x)} = \frac{p_{\theta(x \mid z)} \; p_{\theta(z)}}{p_{\theta(z \mid x)}}$$

- **Zähler:** Berechenbar durch Decoder ($p_{\theta(x|z)}$) und Gaussian Prior ($p_{\theta(z)}$).
- **Nenner:** $p_{\theta(z|x)}$ ist mit dem **Decoder allein nicht berechenbar** und mathematisch extrem komplex .

15.7.2.3. Lösung Approximation

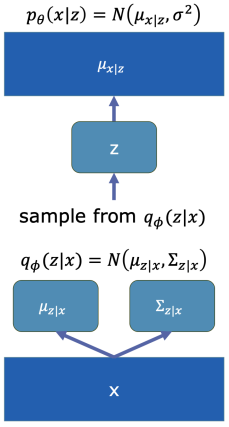
Lösung: Die komplizierte Verteilung $p(z|x)$ wird durch eine einfache, handhabbare Gauss-Verteilung $\boldsymbol{q(z|x)}$ angenähert.



$$p_{\theta(x)} = \frac{p_{\theta(x \mid z)} p_{\theta(z)}}{p_{\theta(z \mid x)}} \approx \frac{p_{\theta(x \mid z)} p_{\theta(z)}}{q_{\varphi(z \mid x)}}$$

15.7.2.4. ELBO Objective & Training

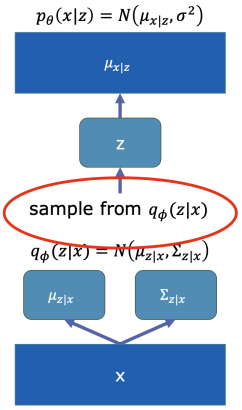
- **ELBO (Evidence Lower Bound):** Da die direkte Berechnung der Datenwahrscheinlichkeit unmöglich ist, maximiert man die ELBO als Ersatz-Zielwert.



$$\underbrace{E_{z \sim q_{\varphi(z|x)}}\left[\log p_{\theta(x|z)}\right]}_{\text{Reconstruction Loss}} - \underbrace{D_{KL}\left(q_{\varphi(z|x)}, p(z)\right)}_{\text{Prior Loss}}$$

- **Reconstruction Loss:** Optimiert den Decoder, damit der Output x' dem Input x so nah wie möglich kommt (entspricht der Minimierung des **L2-Abstands**).
- **Prior Loss (KL-Divergenz):** Reguliert den Encoder, damit die latente Verteilung einer Standard-Normalverteilung ($\mu = 0, \sigma = 1$) folgt.
- **Spannungsfeld:** Der Reconstruction Loss strebt nach Präzision (Varianz gegen 0), während der Prior Loss den Raum glättet und kompakt hält.

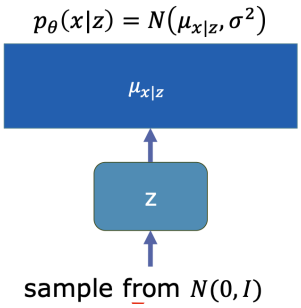
15.7.2.5. Reparametrization Trick



- **Problem:** Man kann nicht direkt durch eine zufällige Sampling-Operation (Stochastik) backpropagieren.
- **Lösung:** Auslagerung des Zufalls durch ein separates Rauschen ε.
- **Formel:** $z = \mu + \sigma \cdot \varepsilon$ macht das Netzwerk differenzierbar, da μ and σ nun wieder normale, lernbare Parameter sind.

15.7.2.6. Sample zur Generierung

- **Generative Phase:** Da der Prior Loss den latenten Raum während des Trainings normalisiert hat, wird der Encoder zur Bilderzeugung nicht mehr benötigt.



- **Prozess:** Man zieht einen Punkt z einfach direkt aus einer Standard-Normalverteilung $N(0, I)$ und lässt ihn durch den Decoder laufen, um ein neues, realistisches Bild zu erhalten.

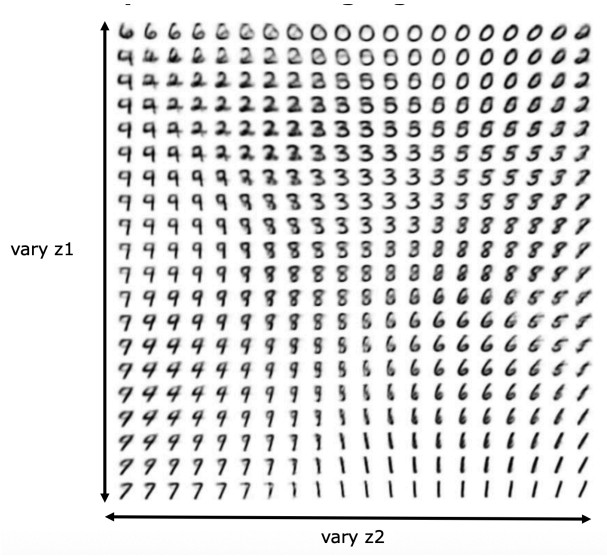
15.7.3. Latent Space

DEFINITION: Keine „toten Zonen“; jeder Punkt im latenten Raum führt zu einem sinnvollen Output (im Gegensatz zum Standard-AE).

Semantische Struktur: Ähnliche visuelle Konzepte gruppieren sich automatisch nah beieinander
- erlaubt gezielte Manipulation des Outputs

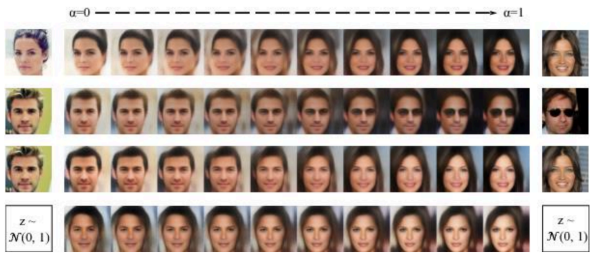
15.7.3.1. Interpolation

- Ermöglicht glatte Übergänge zwischen zwei Datenpunkten
  - z.B. von 7 zu 1
- jeder Zwischenschritt im Raum ergibt ein realistisches Bild.



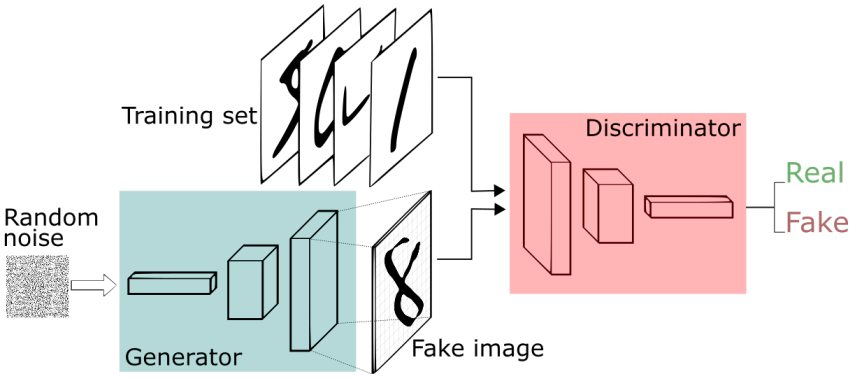
15.7.3.2. Disentangling

- Dimensionen von z werden unabhängig
- einzelne Merkmale (z. B. Rotation, Dicke, Brille) lassen sich isoliert steuern



15.8. Generative Adversial Networks -GAN

DEFINITION: Ein GAN besteht aus zwei neuronalen Netzen, die in einem wettbewerbsorientierten Spiel (**Adversarial Game**) gegeneinander antreten.



Generator (CNN):

- **Input:** Ein Vektor aus zufälligem Rauschen (**Random Noise** z).
- **Ziel:** Erzeugt künstliche Bilder, die so realistisch sind, dass sie den Diskriminator täuschen.

Diskriminator (CNN):

- **Input:** Erhält abwechselnd echte Bilder (z. B. echte Schlafzimmer) und generierte Fake-Bilder.
- **Ziel:** Klassifiziert korrekt, ob ein Bild **„Echt“ oder „Fake“** ist.

Training

- **Gemeinsames Training:** Beide Netze werden gleichzeitig trainiert und verbessern sich gegenseitig.
- **Lerneffekt:** Wenn der Diskriminator Fälschungen erkennt, lernt der Generator, bessere Bilder zu erzeugen. Wird der Diskriminator getäuscht, lernt er, genauer hinzusehen.

15.8.1. Training

MinMax-Spiel

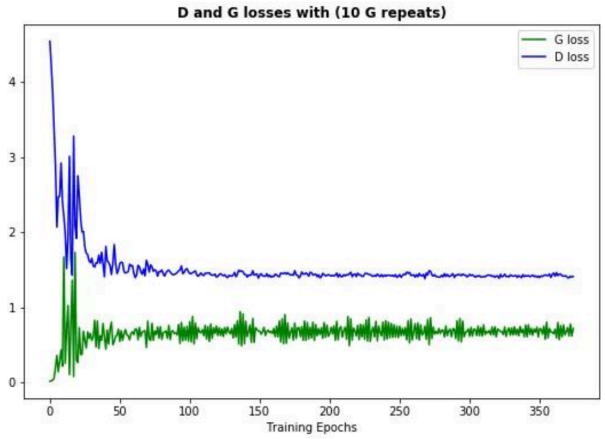
- **Diskriminator (D):** Versucht, die Funktion zu **maximieren**. Will echte Bilder (x) als „1“ (real) & generierte Bilder ($G(z)$) als „0“ (fake) klassifizieren.
- **Generator (G):** Versucht, die Funktion zu **minimieren**. Will, dass Fake-Bilder als real erkannt werden.

$$\min_G \max_D V(D,G) = E_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + E_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- $E_{x\sim p_{\text{data}}(x)}[\log D(x)]$: Erwartungswert der korrekten Erkennung echter Daten durch den Diskriminator.
- $E_{z\sim p_{z(z)}}[\log(1 - D(G(z)))]$: Erwartungswert dafür, dass der Diskriminator die vom Generator erzeugten Fake-Bilder entlarvt.

15.8.2. Probleme

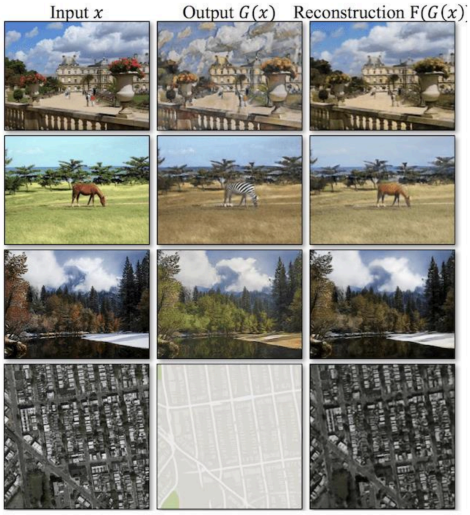
- **Instabilität:** Kein einfaches Minimierungsproblem, sondern ein dynamisches, nicht-kooperatives Spiel (MinMax).
- **Mode Collapse:** Generator erzeugt nur eine sehr begrenzte Variation an Outputs (repetitive, ähnliche Bilder).
- **Vanishing Gradients:** Diskriminator wird zu schnell perfekt; Generator erhält keine nützlichen Gradienten mehr zum Lernen.



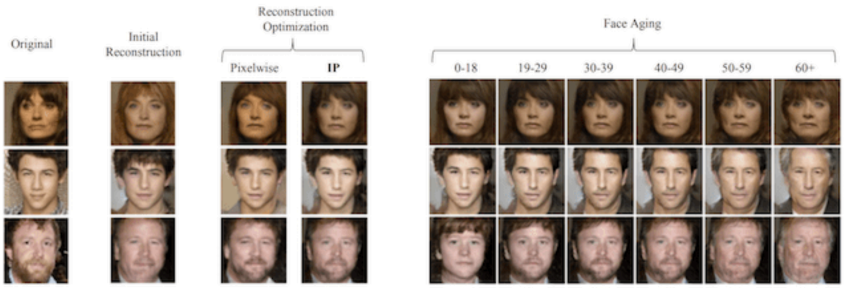
- **Nicht-Konvergenz:** Modelle oszillieren (schwingen) und finden kein stabiles Gleichgewicht.
- **Hyperparameter-Sensitivität:** Extrem empfindlich gegenüber kleinsten Einstellungen (Lernrate etc.).
- **Lösungen:** Einsatz von **Wasserstein Loss**, Regularisierung und spezialisierten Architekturen (z. B. StyleGAN).

15.8.3. Verwendung

Image-to-Image Translation

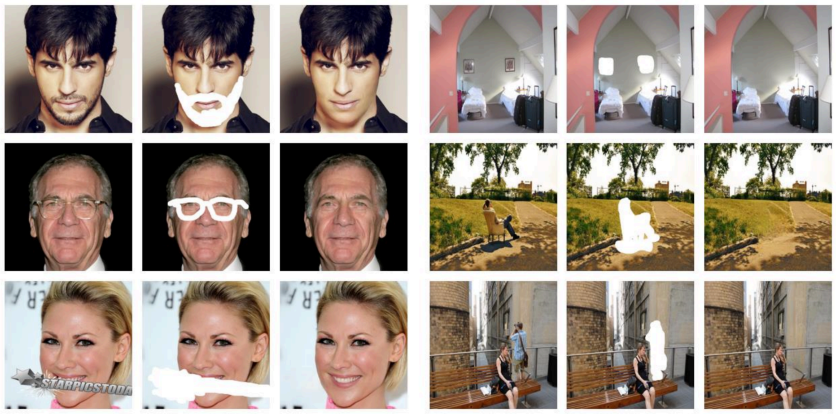


Face Aging



Photoshop

- Face Editing
- Object Removal



(a) Face Editing (b) Object Removal

15.9. DDPMs (Denoising Diffusion Probalistic Models)

DEFINITION: Generation von purem **Noise** zu einem neuen **Bild**.

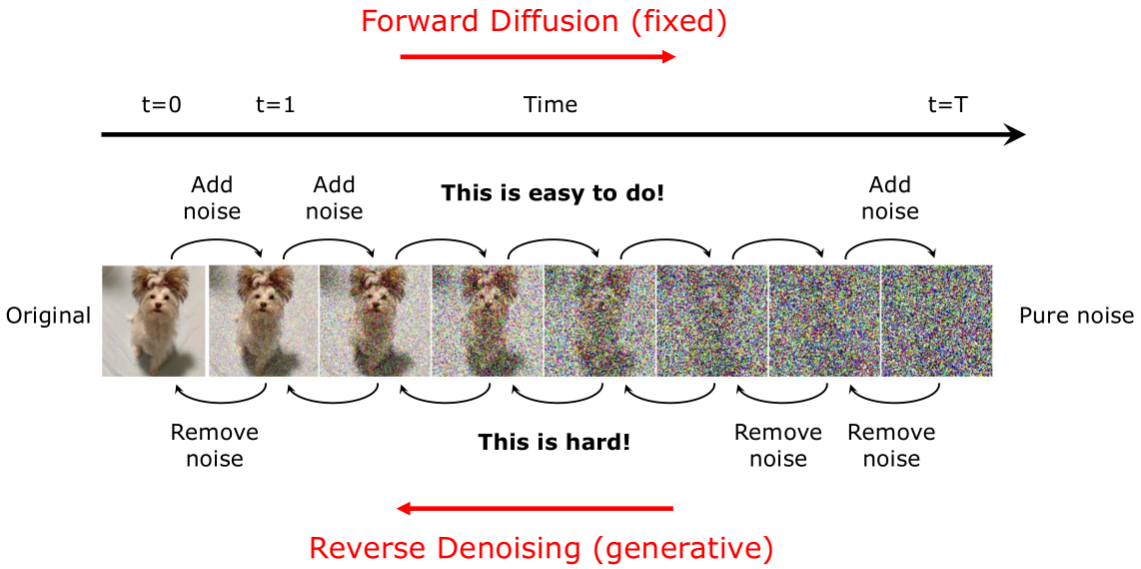
Zwei-Phasen-Konzept:

- **Forward Diffusion** (Fixer Weg)
- **Reverse Denoising** (Generativer Weg)

1. Forward Diffusion (Fixer Weg)

- Schrittweise Rauschen zum Bild hinzufügen ($t = 0 \rightarrow T$)
- Einfach zu berechnen

2. Reverse Denoising (Generativer Weg)
- Rauschen schrittweise entfernen ($T \rightarrow t = 0$).
- Schwer zu berechnen
- Training eines neuronalen Netzes, um das Rauschen schrittweise zu entfernen.



15.9.1. Forward Diffusion

Schrittweises Hinzufügen von Gauss-Rauschen (**Normalverteilung**).

Einzelner Schritt ($q\left(x_t|x_{\{t-1\}}\right)$):
- Bild x_t basiert auf dem Bild davor ($x_{\{t-1\}}$).

$$q\left(x_t \mid x_{\{t-1\}}\right) = \mathcal{N}\left(x_t; \underbrace{\sqrt{1-\beta_t}x_{\{t-1\}}}_{\text{Mean}}, \underbrace{\beta_t \mathrm{I}}_{\text{Variance}}\right)$$

- β_t: Noise Schedule
- **Mittelwert / Mean:** Vorheriger Schritt wird leicht herunterskaliert
- **Varianz:** neues Rauschen wird addiert

Joint Distribution (Gesamtprozess):
- Berechnung Pfad vom Originalbild (x_0) zum Rauschen (x_T).
- Gesamtwahrscheinlichkeit: Produkt aller Einzelschritte:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1})$$

Endzustand (x_T):
- **Pure Noise**, spezielle Normalverteilung (Mittelwert 0, Varianz 1).

Beispiel:

$$q\left(x_{\{1:3\}}|x_0\right) = q(x_1|x_0) * q(x_2|x_1) * q(x_3|x_2)$$

15.9.2. Diffusion Kernel

DEFINITION: Rauschen **nicht Schritt für Schritt** hinzufügen. >Sondern **jeder beliebige Zeitpunkt t kann berechnet** werden. >**Schritte können übersprungen werden.**

$|\{\alpha\}_t$: Produkt aller bisherigen Schritte ($1 - \beta_s$):

$$\overline{a}_t = \prod_{s=1}^{t}(1-\beta_s)$$

→ Diffusionskernel (Gauss-Kernel) definieren mit $|\{\alpha\}_t$

$$q(x_t|x_0) = N\left(x_t; \sqrt{\overline{a}_t}x_0, (1-\overline{a}_t I)I\right)$$

Problem: Direktes Sampling aus Verteilung nicht differenzierbar
- schwer zum Trainieren

→ Reparametrization Trick

15.9.3. Reparametrization Trick

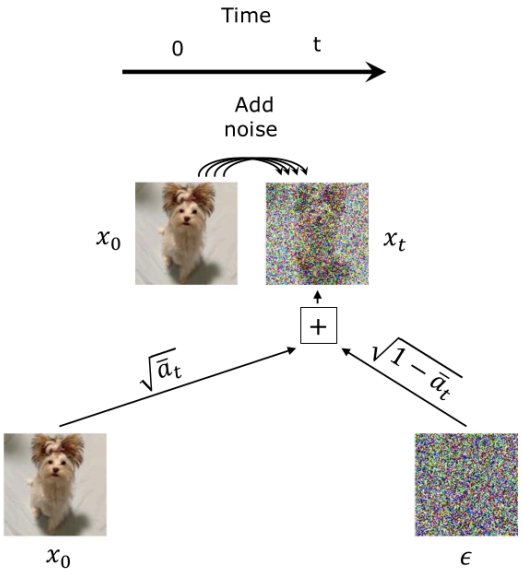
Umwandlung des Samplings in eine berechenbare Formel

$$x_t = \underbrace{\sqrt{\overline{\alpha}_t}x_0}_{\text{Mean}} + \underbrace{\sqrt{1-\overline{\alpha}_t}\varepsilon}_{\text{Std. dev.}} \quad \text{where} \quad \varepsilon \sim \mathcal{N}(0,\mathrm{I})$$

- $\sqrt{\overline{\alpha}_t}x_0$**:** Anteil des Originalbildes (Mean)
- ε**:** Das eigentliche Rauschen (Random Noise aus $N(0,I)$).
- $\sqrt{1-\overline{\alpha}_t}$**:** Gewichtung des Rauschens

15.9.4. $\overline{\alpha}_t$

DEFINITION: **Mischverhältnis ($\overline{\alpha}_t$):** Steuert den Bild-Restanteil beim **Direktsprung** von x_0 zu x_t.



$$x_t = \sqrt{\overline{a}_t}x_0 + \sqrt{(1-\overline{a}_t)}\epsilon \text{ where } \epsilon \sim \mathcal{N}(0,\mathrm{I})$$

- $\overline{\alpha}_t$ **gross:** Viel Originalbild (x_0), wenig Rauschen.
- $\overline{\alpha}_t$ **klein:** Wenig Originalbild, viel Rauschen (ε).

15.9.5. Noise Schedule

DEFINITION: **Noise Schedule (β_t):** Steuert die Rausch-Intensität pro **Einzelschritt**.

- in der Praxis oft linear

$$\overline{a}_t = \prod_{s=1}^{T}(1-\beta_s) \to 0 \text{ and } q(x_t, x_0) \approx N(x_t; 0, I)$$

15.9.6. Ancestral Sampling

In der Praxis ist nur die anfängliche Verteilung bekannt.

Um $x_t \sim q(x_0)$ zu samplen, wird folgendermassen vorgegangen:
1. $x_0 \sim q(x_0)$
2. $x_t \sim q(x_t \mid x_0)$

Das wird **Ancestral Sampling** genannt.

15.9.7. Generatives Denoising

Generation basiert ebenfalls auf Ancestral Sampling

1. Sampling $x_t \sim N(x_t; 0, I)$
2. Iteratively Sampling $x_{t-1} \sim q(x_{t-1} \mid x_t)$

Für x_t wird angenommen, dass es sich um die Normalverteilung handelt:

$$x_t = N(x_t, 0, I)$$

Die Denoising Distribution ist definiert als:

$$p_{\theta(x_{t-1} \mid x_t)} = N(x_{t-1}; \mu_\theta(x_t, x), \sigma_t^2 I)$$

Hier ist $\mu_\theta(x_t, t)$ ein **trainierbares Netzwerk** (meistens ein U-Net)

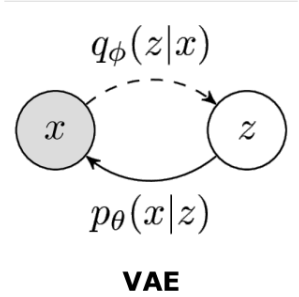
Der Denoising-Prozess kann als Produkt der Distributions beschrieben werden:

$$p_\theta\left(x_{x_0.T}\right) = p(x_T) \prod_t^T = 1 p_\theta(x_{t-1} \mid x_1)$$

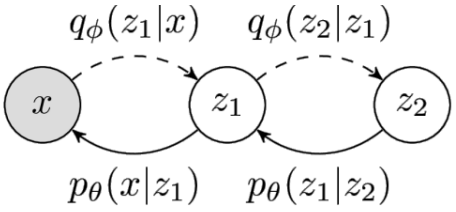
15.9.8. Training

DDPMs sind ähnlich wie hierarchische VAEs

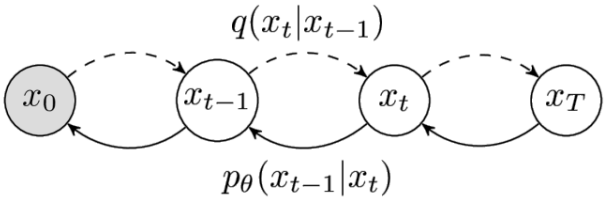
Zum Vergleich:



VAE



Hierarchical VAE



DDPM

DDPMs können auch mit einem ELBO Objective trainiert werden.

Formel:

$$\mathbb{E}_k \left[D_{\mathrm{KL}}(q(x_T \mid x_0) \parallel p(x_T)) + \sum_{t>1} D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t)) - \log p_\theta(x_0 \mid x_1) \right]$$

Erklärungen zur Formel:

- $D_{\mathrm{KL}}(q(x_T|\ x_0) \parallel p(x_T)) = L_T$
  - Konstante
- $D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t)) = L_{t-1}$
  - beschreibt die Abweichung zwischen dem Denoising model und einer Verteilung des weniger verauschten Bildes, unter Andbetracht der Tasache das wir wissen wie das verauschtere Bild aussieht und wie der denoised Output aussehen sinnvollen
- $\log p_\theta(x_0 \mid x_1) = L_0$
  - Reconstruction Term wie bei VAEs

Da es sich bei L_{t-1} bei beiden Operanden um Gaussverteilungen handelt, kann die Divergenz als L2-Distanz der Durchschnitte berechnet werden:

$$L_{t-1} = D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t)) = E_q\left[\frac{1}{2\sigma^2} \parallel \overline{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t) \parallel^2\right] + C$$

Hier kann man nun $\overline{\mu}_t(x_t, x_0)$ und $\mu_\theta(x_t, t)$ substituieren

$$\overline{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{1-\beta_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\overline{a_t}}}\varepsilon\right)$$

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\overline{a_t}}}\varepsilon_\theta(x_t, t)\right)$$

Das kann nun wieder parametrisiert und auch simplifiziert werden:

$$L_{\text{simple}} = E_{x_0 \sim q(x_0), \varepsilon \sim N(0,I), t \sim U(1,T)}\left[\left[|\varepsilon - \varepsilon_\theta\left(\sqrt{\overline{a}_t}x_0 + \sqrt{(1-\overline{a_t})}\varepsilon, t\right) \parallel^2\right]\right.$$

$$\left(\sqrt{\overline{a}_t}x_0 + \sqrt{(1-\overline{a_t})}\varepsilon, t\right) = x_t$$

15.9.8.0.1. Pseudocode

Training:

repeat:

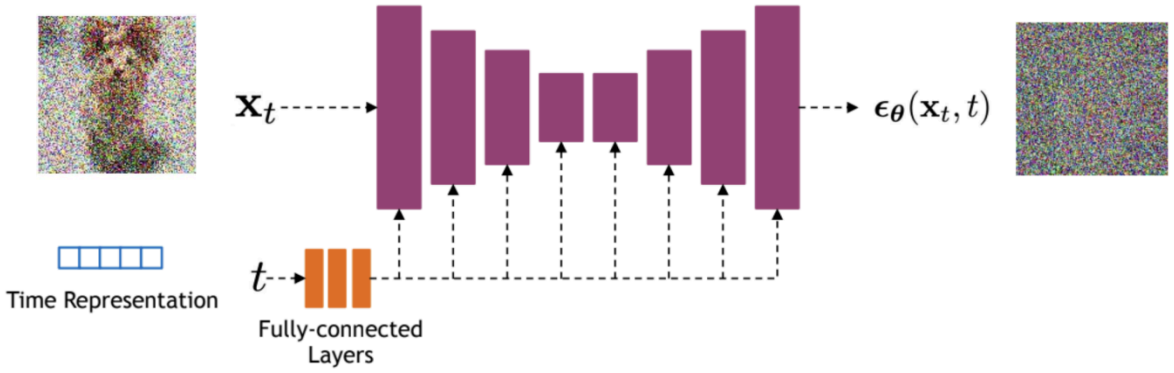
$x_0 \sim q(x_0)$
$t \sim \text{Uniform}(\{1, ..., T\})$
$\varepsilon \sim N(0, I)$
Take gradient descent step on $\|\varepsilon - \varepsilon_\theta\left(\sqrt{\overline{a}_t}x_0 + \sqrt{(1-\overline{a_t})}\varepsilon, t\right) \parallel^2$
until converged

Sampling:

$x_t \sim N(0, I)$
for $t = T, ..., 1$ do
$z \sim N(0, I)$ if $t > 1$, else $z = 0$
$x_{t-1} = \frac{1}{\sqrt{a_t}}\left(x_t - \frac{1-a_t}{\sqrt{1-a_t}}\varepsilon_\theta(x_t, t)\right) + \sigma_t z$
end for
return x_0

15.9.9. Model Architecture

Meist eine **U-Net** Architektur (mit ResNet-Blöcken und Self-Attention).



- **Input:** Verrauschtes Bild x_t + **Zeitinformation** t.
- **Output:** Das Modell schätzt **nur das Rauschen** (ε_θ), nicht das fertige Bild.

Zeit-Embedding: Modell muss wissen, welcher Schritt gerade bearbeitet wird (via Sinus-Kurven oder Fourier-Features).

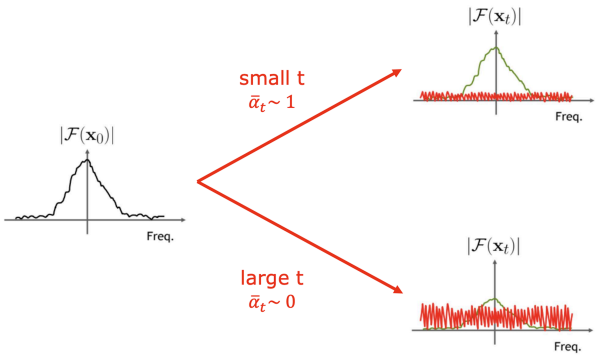
WICHTIG: Das vorhergesagte Rauschen wird vom aktuellen Bild abgezogen, um zum nächsten (saubereren) Schritt zu gelangen.

15.9.10. Content Detail Trade Off

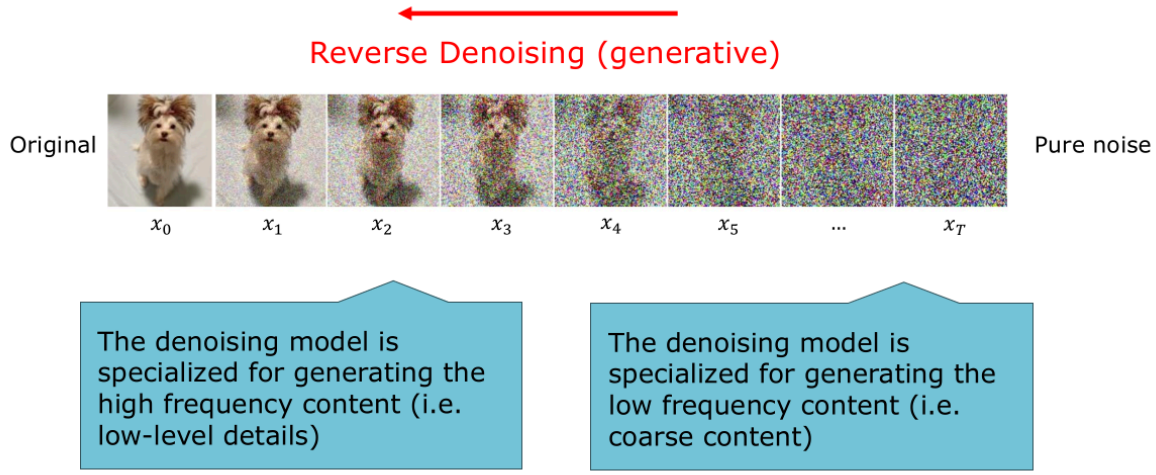
Je weiter die Zeit voranschreitet, desto weniger Content vom Bild und desto mehr Noise ist zu sehen.

WICHTIG: Noise überdeckt zuerst feine Details, erst danach grobe Strukturen.

Forward Process (Zerstörung):
- **Hohe Frequenzen: Feine Details** werden **zuerst** zerstört (kleines t).
- **Niedrige Frequenzen: Grobe Strukturen** bleiben **am längsten sichtbar**.



Reverse Process (Generierung):
- **Zuerst:** Rekonstruktion der **groben Inhalte** (niedrige Frequenzen).
- **Zuletzt:** Verfeinerung durch **Details** (hohe Frequenzen).



→ Modell spezialisiert sich je nach Zeitschritt auf grobe Formen oder feine Texturen.

15.9.11. Performanz - Distillation

Problem: Sampling ist **super langsam**

- viele iterative Schritte nötig, z. B. 30-50+.

Lösung (Distillation): Reduzierung der benötigten Schritte zur Bildgenerierung.

DEFINITION: Wissenstransfer von einem grossen auf ein effizientes Modell.

Teacher Model:

- Auf Originaldaten trainiert.
- Hohe Qualität, aber rechenintensiv (viele Schritte).

Student Model:

- Trainiert auf den **Outputs des Teachers**.
- Lernt, mehrere Denoising-Schritte zu **überspringen**.

15.9.12. Latent diffusion models

DEFINITION: Diffusion erfolgt im **Latent Space** statt im Pixel-Raum.

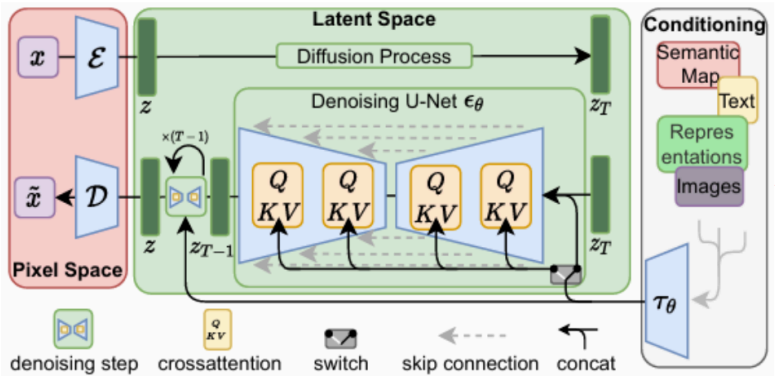
Diffusion wird im Latent Space angewendet:

- Perceptual Autoencoder, um das Image zu komprimieren.
- Output des Perceptual Autoencoders befindet sich im Latent Space.

diffusion in latent space
ɔtual autoencoder for image compression
ion model operating in the resulting latent space

Diffusion in latent space!

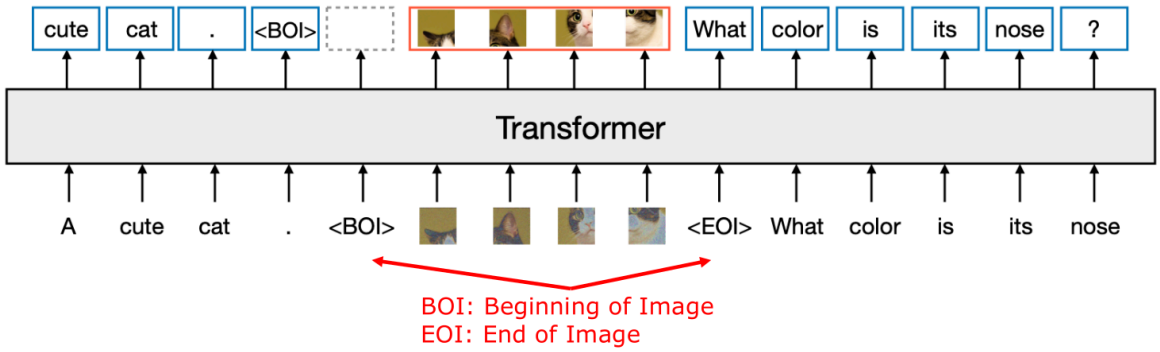
$$L_{latent} = E_{z_0 \sim q(z_0), \epsilon \sim \mathcal{N}(0,I), t \sim \mathcal{U}(1,T)} \left[\left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} z_0 + \sqrt{(1-\bar{\alpha}_t)} \epsilon, t \right) \right\|^2 \right]$$



Conditioning via attention.

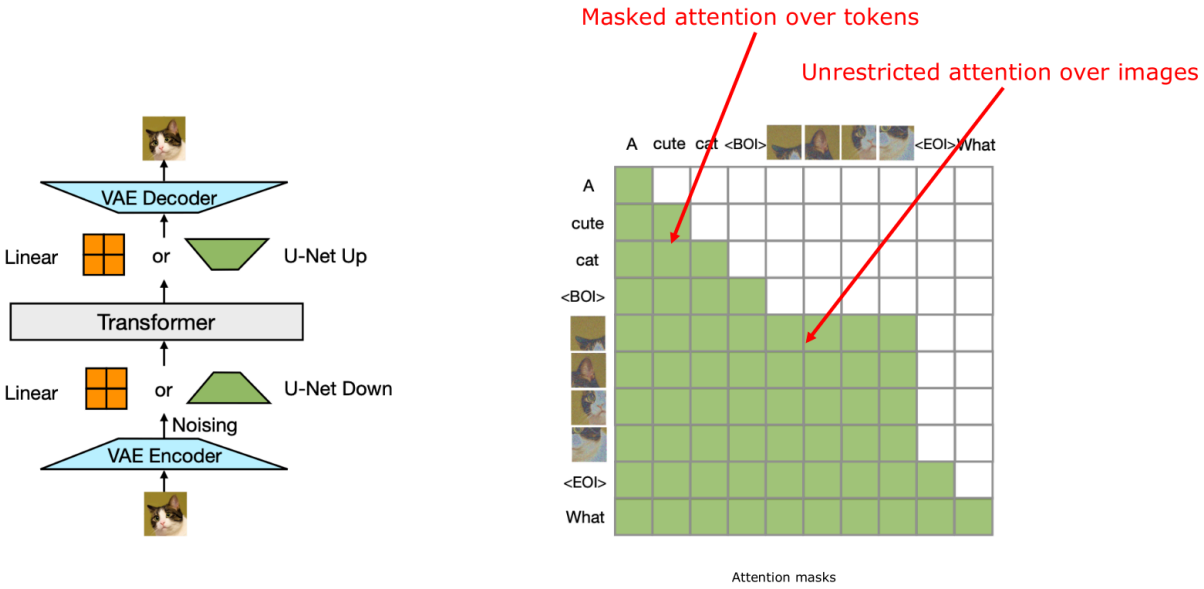
15.10. Transfusion

DEFINITION: Ein einziges Modell (Transformer) generiert gleichzeitig **Text und Bild**



Zu beachten:

- Attention bei Text nur auf nächstes Token.
- Attention bei Bildern auf ganzes Bild.



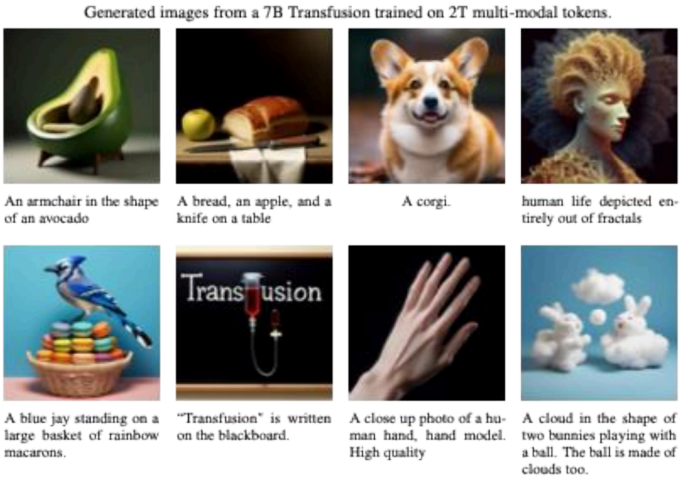
Text:

- Strings werden mithilfe von fixem Vokabular tokenisiert.
- Embedding-Matrizen konvertieren Tokens zu hochdimensionalen Vektoren.
- Language Models berechnen die Probability des nächsten Tokens.

Bilder:

- Bild wird in **8x8 Patches** unterteilt.
- Patches werden via **VA** in **Latent Representations** komprimiert.
- Training erfolgt mittels **DDPM** (Diffusion) auf diesen Patches

Beispiele von Tranfusion:



15.10.1. Weiterführende Themen

- CLIP
- DALLE-2
- Stable Diffusion
- Diffusion LLMs for Text

15.11. Texture Synthesis

DEFINITION: Aus kleinen Texturbild ein beliebig grosses Bild generieren.

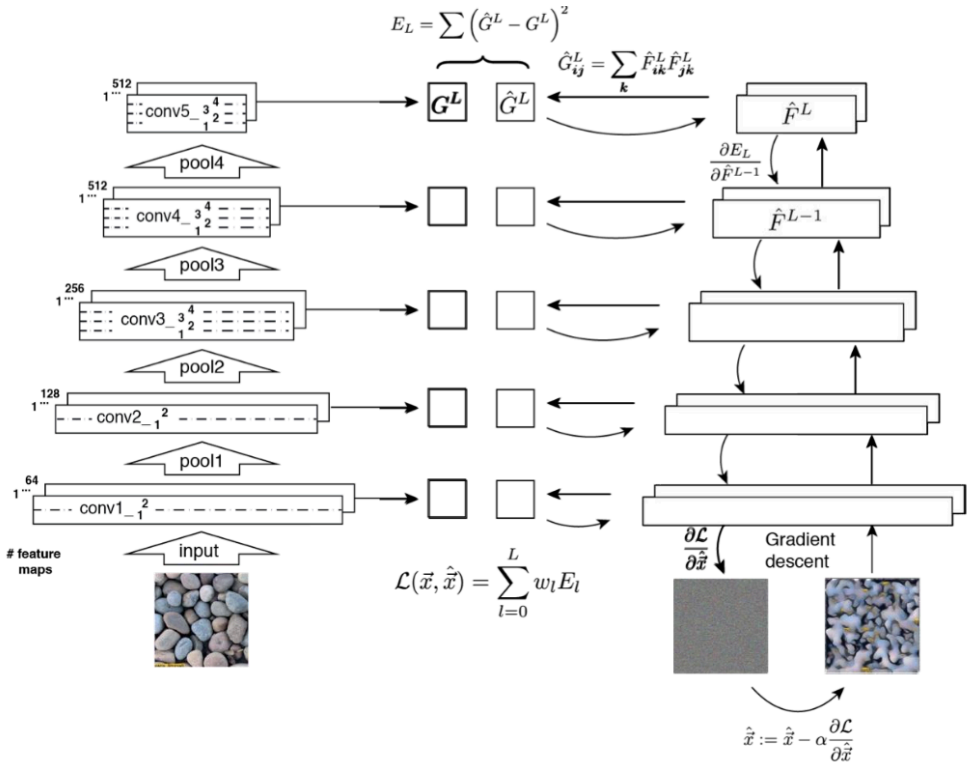
Zwei Hauptansätze:
1. **Klassisch:** Pixel/Patches replizieren oder Texturmodell finden.
2. **Neural:** Convolutional Networks (CNN) als Modell nutzen.

15.11.1. Neural Texture Synthesis
- Bild wird in einem Pretrained Network verarbeitet.
- Die Activations von jedem Layer sind die Feature Maps $F^l \in R^{N \times M}$ der Textur.
- Die Gram Matrix G^l_{ij} berechnet die Correlations zwischen den Feature Maps auf demselben Layer.

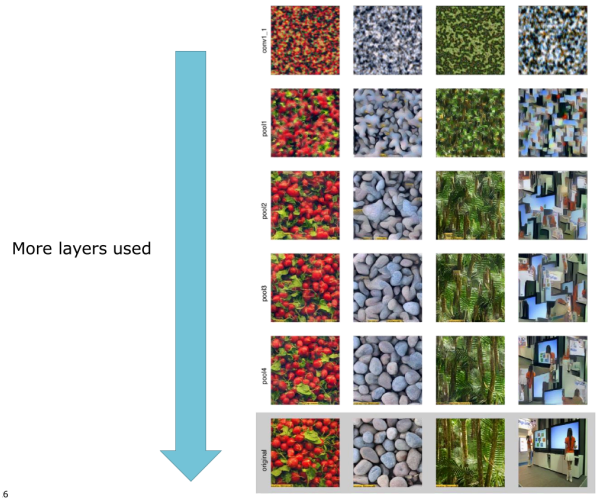
$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

Gram Matrix:
- Dient als **Texture-Model**.
- Beschreibt die **Korrelation** zwischen verschiedenen Feature-Maps.
- Erfasst den „Stil“ oder die „Textur“, ohne die exakte räumliche Anordnung (den Inhalt) zu speichern.

Anwendung:
1. Starte mit einem Noise Image.
2. Gradient Descent, um ein Bild zu finden, welches die Gram Matrix bestimmter Layers vom Originalbild matcht.
3. Minimiere die Mean Squared Distance zwischen den Matrizen vom originalen und generierten Bild.

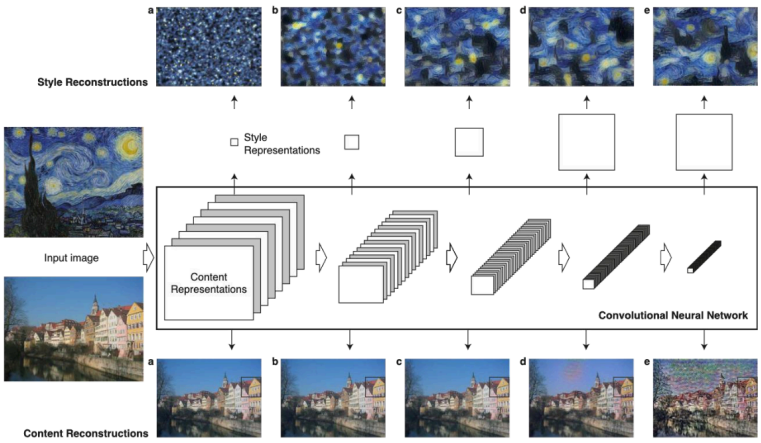


Layer-Einfluss:
- **Niedrige Layer:** Farben & lokale Punkte.
- **Tiefe Layer:** Komplexe Strukturen & Muster.



15.12. Neural Style Transfer

DEFINITION: Kombination des **Inhalts** (*Content*) eines Bildes mit dem **Stil** (*Style*) eines anderen Bildes.



15.12.1. Content Representation

- Jedes Input-Bild generiert Filter Responses $F^l \in R^{N \times M}$ auf jedem Layer.
- Gradient Descent, bis ein Bild mit den gleichen Responses gefunden wird.
- Loss wird berechnet als (P_l ist die Response vom generierten Bild):

$$L_{\text{content}(\vec{p},\vec{x},l)} = \frac{1}{2} \sum_{i,j} \left(F_{ij}^l - P_{ij}^l \right)^2$$

- Spätere Layer definieren den Bildinhalt

15.12.2. Style Representation

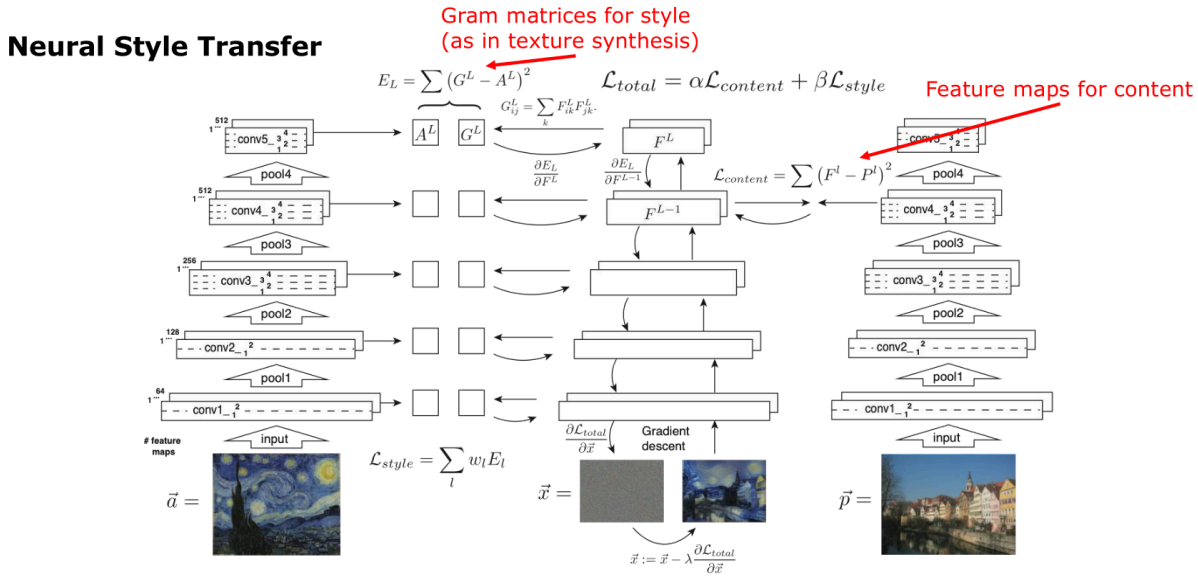
- Aus den Filter Responses $F^l \in R^{N \times M}$ werden die Gram-Matrizen berechnet.
- A^l ist die Matrix vom Originalbild, G^l die Matrix vom generierten Bild.

- Loss wird berechnet:
  - Für ein Layer:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

  - Für mehrere Layers:

$$L_{\text{style}(\vec{a},\vec{x})} = \sum_{l=0}^{L} w_l E_l$$



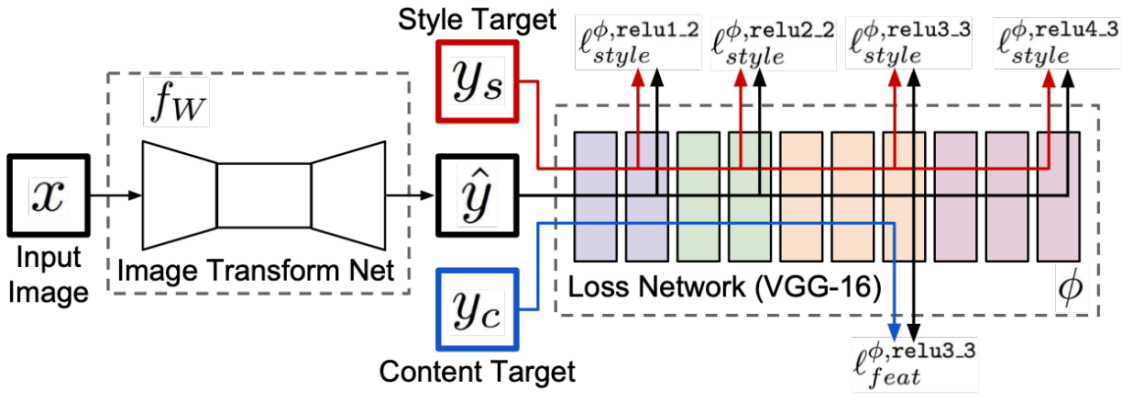
WICHTIG: Basierend auf dem Noise Image kann das Resultat immer ein wenig anders aussehen.

15.12.2.1. Fast Style Transfer

Problem: Style Transfer ist langsam

- viele Durchläufe durch das CNN

Lösung: Neuronales Netz trainieren



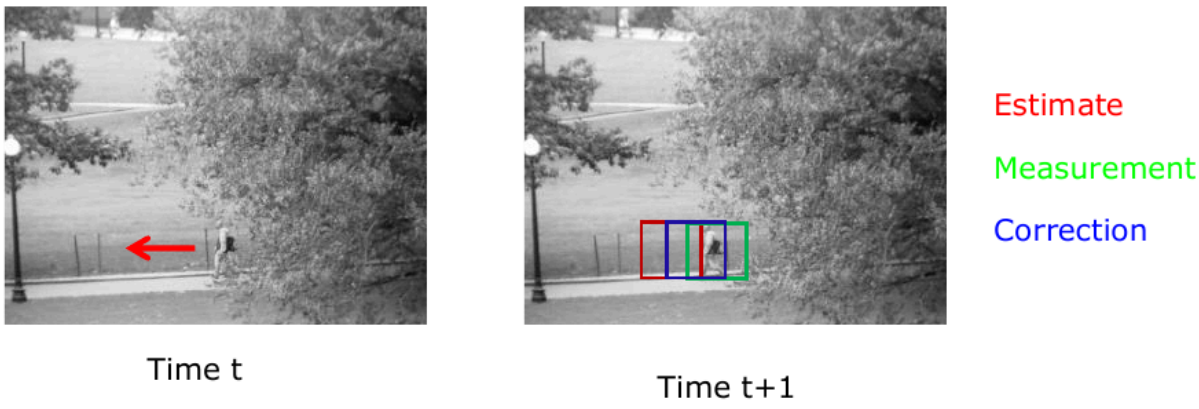
16. Tracking

DEFINITION: Tracking ist die fortlaufende Schätzung des aktuellen und zukünftigen Zustands eines Objekts durch die Kombination von Vorhersagen aus Modellen und verrauschten Messungen.

- **Bestimmt:**
  - Wo befindet sich ein Objekt später?
  - Wo befand sich ein Objekt über eine Zeit?
  - Wo ist ein Objekt jetzt?

16.1. Detection vs. Tracking

- **Detection**: Das Objekt wird in jedem einzelnen Frame unabhängig gesucht und identifiziert.
- **Tracking**: Nutzt die Bewegungshistorie des Objekts, um dessen zukünftige Position zu schätzen (**Bewegungsmodell**).
  - **Vorteil**:
    - Einmal gefunden = Tracking einfacher
    - Sagt nächsten Aufenthaltsort voraus
    - Nur noch lokale Region durchsuchen, keine Vollbildsuche mehr



16.2. Dynamik

- Nutzt die Bewegungsdynamik zur Schätzung der nächsten Position.
- **Benötigt wird:**
  - Der aktuelle Zustand (Position und Geschwindigkeit, als Zustandsvektor X_t).
  - Ein Systemmodell, das beschreibt, wie man vom aktuellen Zustand X_t auf den nächsten Zustand X_{t+1} kommt.

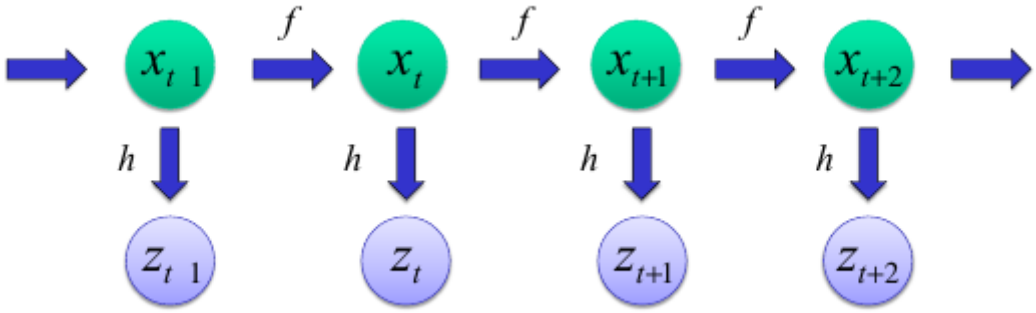
16.3. Tracking

- **Problemstellung**: Das primäre Ziel ist es, den aktuellen Zustand (x) eines Objekts zu finden. Da meist nicht alle Parameter direkt messbar sind, bleiben gewisse Aspekte verborgen.
- **Zustand (x)**: Umfasst alle Eigenschaften, die für das Modell relevant sind (beispielsweise Position und Geschwindigkeit).

16.3.1. Modelle

Das Tracking basiert grundlegend auf einem **Hidden Markov Model** mit folgenden Annahmen:

- Der neue Zustand hängt nur vom direkten Vorgängerzustand ab.
- Der Zustand selbst kann nicht direkt gemessen werden.
- Die einzelnen Messungen sind völlig unabhängig voneinander.



Für die Schätzung sind zwingend zwei Modelle notwendig:

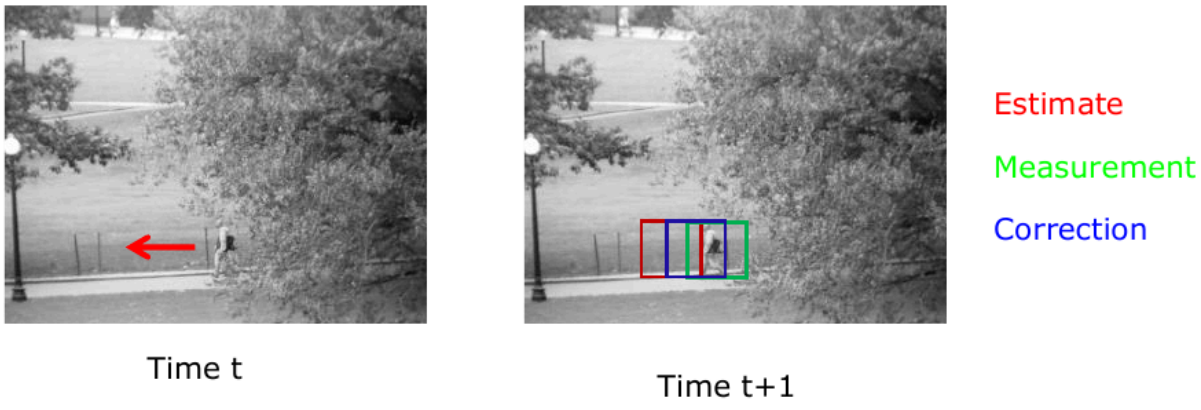
- **Systemmodell**: Bestimmt den Übergang zum nächsten Zustand mittels Wahrscheinlichkeiten.
  - **Allgemein**: $x_{t+1} = f(x_t, v_t)$ wobei v_t das Rauschen (Simulationsunsicherheit) beschreibt.
  - **Linear**: $x_k = Ax_{k-1} + \varepsilon_k$ mit normalverteiltem Systemrauschen $\varepsilon_k \sim N(0, Q_\varepsilon)$.

- **Messmodell**: Beschreibt den Zusammenhang zwischen den Messungen und den verborgenen Zustandsinformationen.
  - **Allgemein**: $z_t = h(x_t, w_t)$ mit Rauschen w_t.
  - **Linear**: $z_k = Cx_k + \delta_k$ mit normalverteiltem Messrauschen $\delta_k \sim N(0, Q_\delta)$.

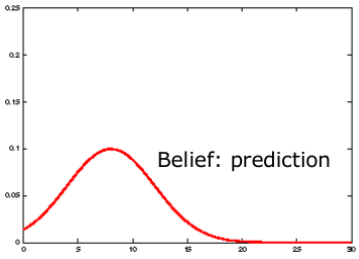
16.3.2. Ablauf und Wahrscheinlichkeiten

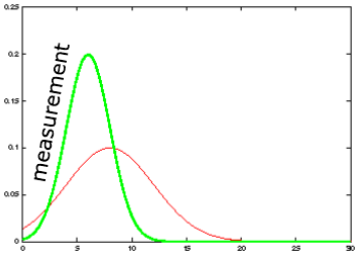
Die Grundidee der meisten Tracking-Algorithmen folgt diesem Ablauf:

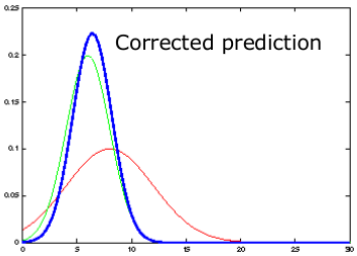
1. **Estimate**: Zustand für den nächsten Zeitschritt schätzen (meist ungenau, da wir nicht alles berücksichtigen können).
2. **Measurement**: Messung vornehmen.
3. **Correction**: Schätzung anhand der Messung korrigieren.



Aufgrund der Unsicherheiten in den Schätzungen und Messungen arbeitet man beim Tracking nicht mit absoluten Werten, sondern mit Wahrscheinlichkeitsverteilungen.







16.3.3. Beispiel

16.3.3.1. 2D-Kinematikmodell

Für ein Objekt, das sich mit konstanter Geschwindigkeit in 2D bewegt.

- **Zustandsvektor**: Besteht aus Position und Geschwindigkeit zum Zeitpunkt t.

$$X_t = \begin{pmatrix} p_{x,t} \\ p_{y,t} \\ v_{x,t} \\ v_{y,t} \end{pmatrix}$$

- **Physikalische Grundlagen**:
  - **Position**: $P_t = P_{t-1} + \Delta t V_{t-1}$
  - **Geschwindigkeit**: $V_t = V_{t-1}$
- **Systemmodell**: Berechnet den neuen Zustand X_t aus dem vorherigen Zustand X_{t-1} mittels Matrixmultiplikation mit A.

$$X_t = A \cdot X_{t-1} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_{x,t-1} \\ p_{y,t-1} \\ v_{x,t-1} \\ v_{y,t-1} \end{pmatrix}$$

- **Messmodell**: Berechnet die Messung Z_t aus dem aktuellen Zustand X_t mittels Matrix C. Da nur die 2D-Position gemessen wird, extrahiert die Matrix die Positionen und ignoriert die Geschwindigkeiten.

$$Z_t = C \cdot X_t = \begin{pmatrix} p_{x,t} \\ p_{y,t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} p_{x,t} \\ p_{y,t} \\ v_{x,t} \\ v_{y,t} \end{pmatrix}$$

- **Rauschen**:
  - Q_ε: Systemrauschen (Unsicherheit des Bewegungsmodells).
  - Q_δ: Messrauschen (Unsicherheit der Sensoren).

16.3.3.2. Praxisbeispiele

- **Rakete**:
  - **Zustand**: Position (im Bild) und Geschwindigkeit.
  - **Systemmodell**: Physikbasierte Bewegung (Kinematik und Schwerkraft).
  - **Messmodell**: Aktuelle Position, jedoch nicht die Geschwindigkeit.
- **Fussballspieler**:
  - **Zustand**: Position auf dem Feld und Geschwindigkeit.
  - **Systemmodell**: Kinematisches Modell oder Verhaltensmodellierung (z.B. mittels Deep Learning).
  - **Messmodell**: Position des Spielers aus verschiedenen Kameraperspektiven.
- **Quadrocopter**:
  - **Zustand**: 3D-Position, Flugrichtung (Heading) und Geschwindigkeit.
  - **Systemmodell**: Kinematisches Modell basierend auf Beschleunigung und Steuerungseingaben.

- **Messmodell**: Datenfusion aus Gyroskop, Beschleunigungssensor, Magnetometer, Drucksensor, Ultraschall (Höhe) und Kamera (Odometrie).

16.4. Wahrscheinlichkeiten und Rauschen

Systemmodelle und Messungen sind in der Praxis nie exakt (z.B. Pixel-Ungenauigkeiten oder abweichendes Bewegungsverhalten).

- Ungenauigkeiten werden als **Rauschen** (Noise) modelliert, weshalb mit Wahrscheinlichkeitsverteilungen gerechnet wird.
- **A priori**: Wahrscheinlichkeit des Zustands **vor** der Messung.
- **A posteriori**: Wahrscheinlichkeit des Zustands **nach** der Messung.

16.5. Zustandsberechnungen

Aus den bisherigen Bausteinen (Zustand, System- und Messmodell, Rauschen) ergeben sich zwei Kernfragen für Tracking-Algorithmen:

1. Wie verbinden wir das Systemmodell mit den Messungen?
2. Wie erhalten wir die beste Schätzung für den nächsten Zustand?

16.6. Kalman Filter

DEFINITION: Ein etabliertes Verfahren zur Zustandsschätzung. Es vereinigt Vorhersage und Schätzung zur Optimierung.

- **Voraussetzungen**:
  - Das Systemmodell ist linear.
  - Das Messmodell ist linear.
  - Das Rauschen ist Gausssch (normalverteilt).
- **Ergebnis**: Der Zustand wird durch seinen Mittelwert und eine Gauss-Verteilung berechnet.

16.6.1. Ablauf (Predict & Correct)

Der Filter iteriert kontinuierlich durch zwei Schritte ($k \leftarrow k+1$):

1. **Predict (Vorhersage)**: Schätzung des nächsten Zustands und der Unsicherheitsmatrix (P) basierend auf dem physikalischen Modell.

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1}$$

$$P_{k|k-1} = AP_{k-1|k-1}A^H + Q_\varepsilon$$

2. **Correct (Korrektur)**: Vergleich der Vorhersage mit den tatsächlichen Messungen (z_k) zur Korrektur.

$$K_k = P_{k|k-1}C^H\left(CP_{k|k-1}C^H + Q_\delta\right)^{-1}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\left(z_k - C\hat{x}_{k|k-1}\right)$$

$$P_{k|k} = (I - K_kC)P_{k|k-1}$$

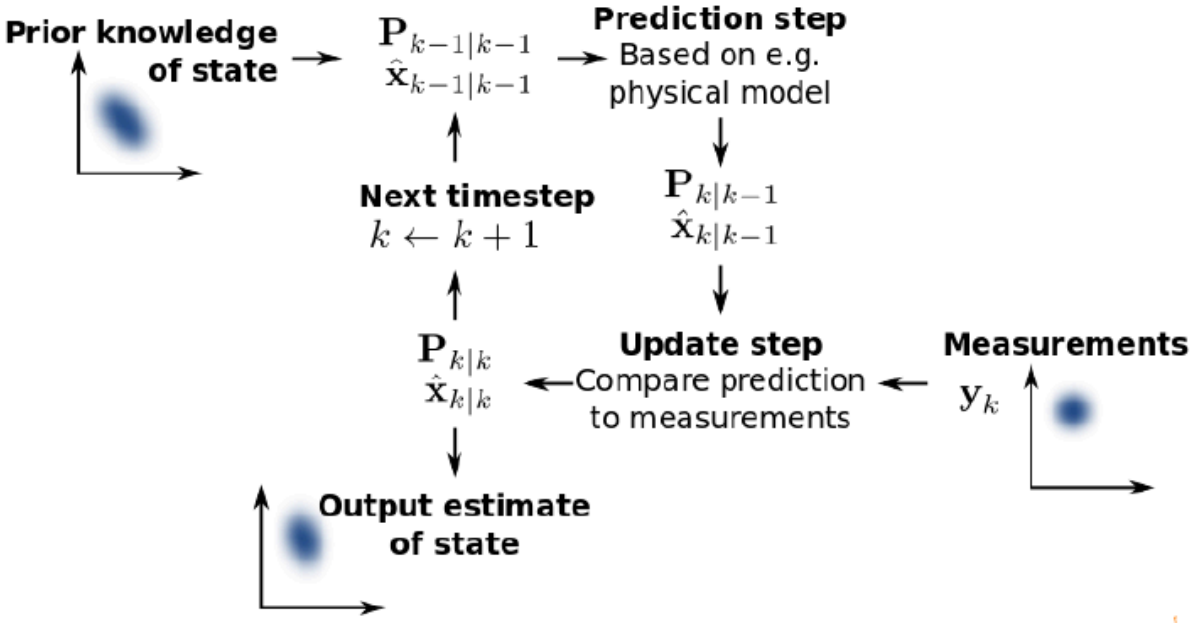


Abbildung 1: Verknuepfung von Vorhersage und Messung zur optimalen Zustandsschaetzung.

16.6.2. Initialisierung und Stellschrauben (Beispiel)

Für das 2D-Kinematikmodell sieht die konkrete Vorbereitung für den Kalman Filter folgendermassen aus:

- **State Initialization**: Startzustand ($\hat{x}_0$) und anfängliche Unsicherheit (P_0).

$$\hat{x}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$P_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- **Rausch-Matrizen**:

$$Q_\varepsilon = P_0$$

$$Q_\delta = \begin{pmatrix} 2400 & 0 \\ 0 & 2400 \end{pmatrix}$$

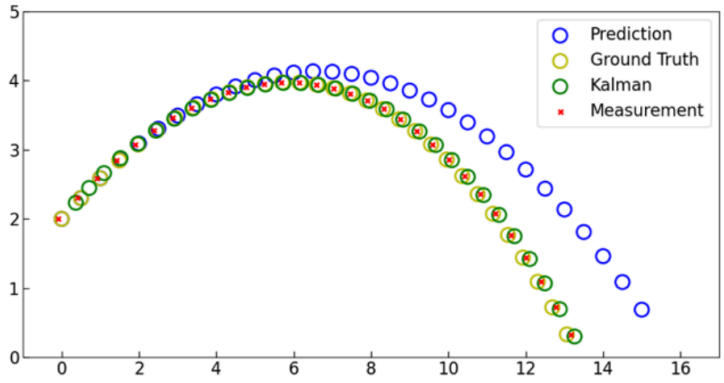
- **Systemrauschen (Q_ε) als Stellschraube**:
  - Einbau zufälliger Unsicherheiten für nicht exakt vorhersagbare Variablen.
  - Verhindert unkontrolliertes Anwachsen der Vorhersage-Unsicherheit.

16.6.3. Beispiel: Volleyball

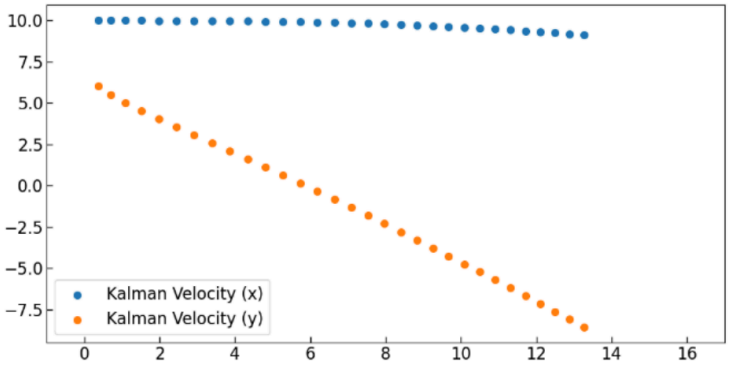
Ziel: Tracking eines Volleyballs beim Aufschlag zur Bestimmung des exakten Landepunkts.

- **Fehlerquellen:**
  - **System Model Error**: Einfaches Vorhersagemodell ohne Luftwiderstand.
  - **Measurement Uncertainty**: Unklare Ballposition im Bild (Blur, Bildeffekte).
  - **Measurement Model Error**: Ungenaue Abbildung von Zustand auf Messung (Linsenverzerrungen).
- **Kalman Filter Resultate:**
  - **Positionen**:
    - Vorhersage ohne Luftwiderstand (blaue Kreise) berechnet zu weite Flugbahn.
    - Tatsächliche Messungen (rote Kreuze) sind fehlerbehaftet.
    - Ground Truth (gelbe Kreise) zeigt die reale Position.

– Kalman Filter (grüne Kreise) pendelt sich optimal zwischen Vorhersage und Messung ein.



- **Geschwindigkeiten (x und y)**:
  - Horizontale Geschwindigkeit in x-Richtung (blaue Punkte) verlangsamt sich.
  - Diese Verlangsamung fehlt im zugrundeliegenden Systemmodell.
  - Korrektur-Effekt entsteht ausschliesslich durch den Einbezug der Messdaten.

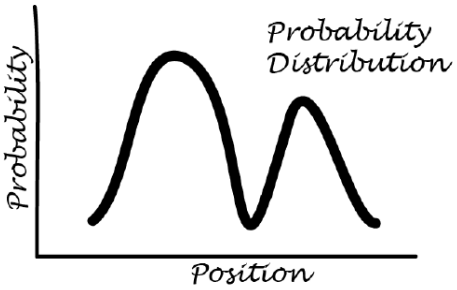


16.7. Particle Filtering

DEFINITION: Ein iteratives Verfahren (Sequential Monte Carlo Method), das Wahrscheinlichkeitsverteilungen durch eine Menge von gewichteten Samples, sogenannten Partikeln, approximiert.

16.7.1. Nachteile Kalman Filter

- Schätzung ist zwingend eine Gauss-Funktion
- Keine multiplen Hypothesen möglich
- Setzt lineare Modelle voraus



16.7.2. Idee (Sequential Monte Carlo Method)

- **Modellierung**: Zustandsverteilungen durch Samples (Partikel)
- **Propagation**: Bewegt und verteilt Partikel

- **Messung**: Gewichtet Partikel
- **Resampling**: Zieht neue Samples anhand der Gewichtung

16.7.3. Modelle (Beliebige Funktionen)

- **Systemmodell**: $p(x_t \mid x_{t-1})$
- **Messmodell**: $p(x_t \mid z_{1:t})$

16.7.4. Berechnung (Bedingte Wahrscheinlichkeiten)

- **Prediction** (Vorhersage des Zustands vor der neuen Messung):

$$p(x_t \mid z_{1:t-1}) = \int p(x_t \mid x_{t-1}) p(x_{t-1} \mid z_{1:t-1}) dx_{t-1}$$

- **Update** (Korrektur der Vorhersage durch die aktuelle Messung):

$$p(x_t \mid z_{1:t}) = \frac{p(z_t \mid x_t) p(x_t \mid z_{1:t-1})}{\int p(z_t \mid x_t) p(x_t \mid z_{1:t-1}) dx_t}$$

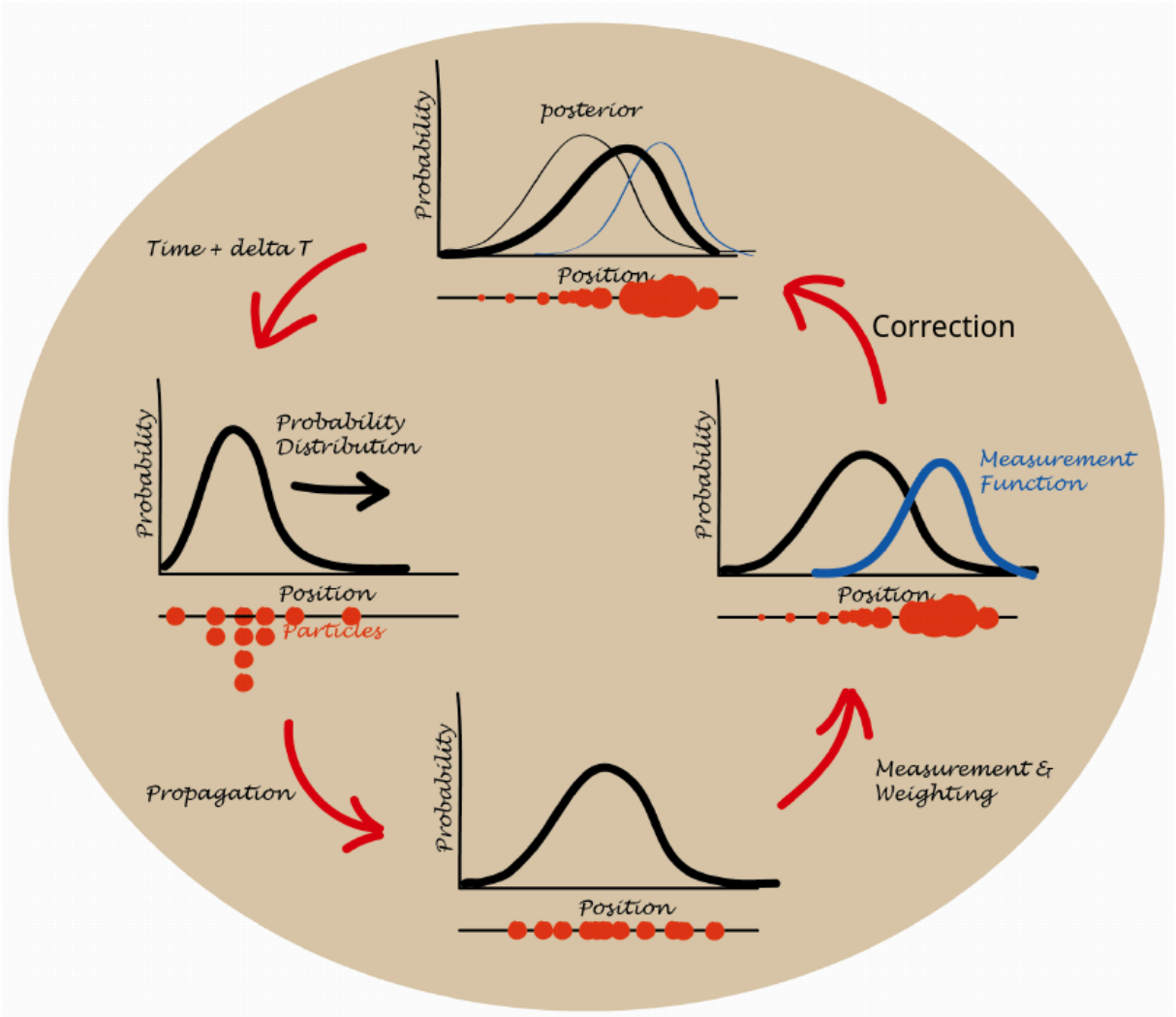
- **Zugehörige Modelle**:
  - $p(x_t \mid x_{t-1})$: Systemmodell
  - $p(z_t \mid x_t)$: Messmodell

16.8. Sampling

DEFINITION: Beschreibung von Wahrscheinlichkeitsverteilungen durch diskrete Samples (Zustand und Gewichtung), da Integrale oft nicht analytisch lösbar sind.

16.8.1. Zyklus des Particle Filterings

- **Hintergrund und Grundlagen**:
  - Integrale der bedingten Wahrscheinlichkeiten sind im Normalfall nicht exakt lösbar
  - Der Kalman Filter bietet nur für den linearen beziehungsweise Gaussschen Fall eine exakte Lösung
  - Verteilungen werden stattdessen durch eine Menge von Samples (Partikeln) approximiert
  - Jedes Sample besteht aus einem spezifischen Zustand und einer zugehörigen Gewichtung
- **Der iterative Ablauf**:
  - **Initiale Verteilung**:
    - Die Start-Wahrscheinlichkeit wird durch eine Wolke von gleichwertigen Partikeln repräsentiert
  - **Propagation (Time + delta T)**:
    - Der Zustand der Partikel wird in die Zukunft projiziert
    - Die Partikel verändern ihre Position gemäss der Dynamik des Systemmodells
  - **Measurement & Weighting**:
    - Die Messfunktion bringt die realen Sensordaten ein
    - Es wird berechnet, wie wahrscheinlich ein Partikel-Zustand die tatsächliche Messung verursacht hat
    - Partikel erhalten ein neues Gewicht (im Diagramm an der Grösse der roten Punkte erkennbar)
  - **Correction (Posterior)**:
    - Die Wahrscheinlichkeitsverteilung wird durch die neuen Gewichte korrigiert
    - Es entsteht die aktualisierte A-posteriori-Verteilung
  - **Repeat (Resampling)**:
    - Aus der korrigierten Verteilung werden neue Partikel gezogen
    - Stark gewichtete Partikel werden vervielfältigt, schwach gewichtete verschwinden
    - Dieser neue Satz an Partikeln bildet die Ausgangslage für den nächsten Berechnungszyklus



16.8.2. Praxisbeispiele Particle Filtering

- **Konkrete Messung (Measurement)**:
  - Berechnet die Wahrscheinlichkeit, dass ein Zustand x die Messung z verursacht
  - **Typisches Verfahren**: Vergleich von Histogrammen (geschätzte Position versus Modell)
- **Verhalten der Partikel (z. B. Crowd- oder Objekt-Tracking)**:
  - **Startphase**: Samples sind anfangs überall im Bild verteilt (sehr hohe Unsicherheit)
  - **Auswertung**: Gewichtung erfolgt über Pattern-Matching
  - **Filterung**: Samples mit niedriger Konfidenz (low confidence) werden weggelassen
  - **Verlauf**: Das Tracking wird über die Zeit immer besser und präziser, Partikel konzentrieren sich auf das Ziel

17. 3D Reconstruction I

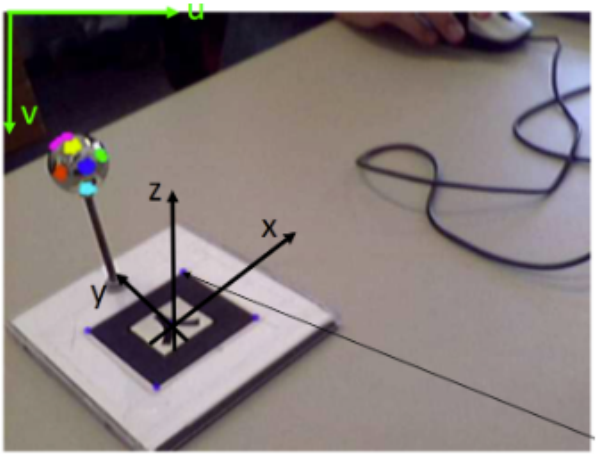
TL;DR: Gleichung: Beschreibt, wie ein Punkt von 3D in 2D projiziert wird.
$p = Cc \cdot P$

p: Image plane point → **Projektion** (X- und Y-Koordinaten) exakte Pixel in 2D (Sensor, Bildschirm) **C**: Calibration matrix → **Kalibrierung** (Matrix) Eigenschaften der Kamera (Position, Blickwinkel, Brennweite der Linse) **P**: Point in 3D space → **Rekonstruktion** (X- Y-, Z-Koordinaten) Punk im 3-Dimensionalem Raum

Probleme:
- C und P bekannt, p lösen: Projektion
- P und p bekannt, C lösen: Kalibrierung
- C und p bekannt, P lösen: Rekonstruktion

17.1. Extrinsische und Intrinsische Kameraparameter

- Aus einem 3D-Punkt (aus der realen Welt) den 2D-Punkt (Pixel auf einem Bild) berechnen.
- 3D-Koordinaten werden auf einer Marker Plane definiert.
- 2D-Bildkoordinaten: $p = (u, v)^T$
- 3D-Weltkoordinaten: $P = (x, y, z)^T$
- Beispiel
  - $P = (40.0, 40.0, 0.0)^T$
  - $p = (196, 284)^T$



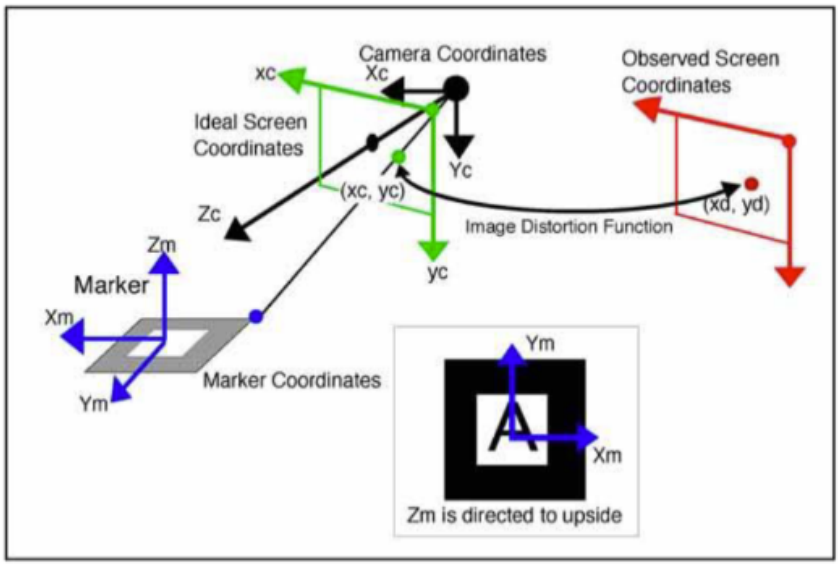
17.1.1. Extrinsische Transformation

DEFINITION: Berechnung, wo sich das 3D-Objekt relativ zur Kamera befindet.
$P' = R \cdot P + T$

$$P' = R \cdot P + T$$

- P: Punkt in der echten Welt, basierend auf einem fixen Koordinatensystem, Vektor
- R: Rotations Matrix (Drehung der Kamera)
- T: Translations Vektor (Position der Kamera im Raum)
- P': 3D Punkt aus der Sicht der Kamera, Vektor
- P: Punkt in Weltkoordinaten, basierend auf einem fixen Koordinatensystem (Vektor).
- R: Rotationsmatrix (Ausrichtung der Kamera).
- T: Translationsvektor (Position der Kamera im Raum).
- P': 3D-Punkt aus der Sicht der Kamera (Vektor).

Koordinaten werden transformiert:
- Rotation: 3x3 Rotationsmatrix R
- Translation: Vektor T



$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_x \\ r_{31} & r_{32} & r_{33} & t_x r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = (\boldsymbol{R} \quad \boldsymbol{T}) \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Extrinsische Matrix E (3x4)

$$\boldsymbol{E} = (\boldsymbol{R} \quad \boldsymbol{T})$$

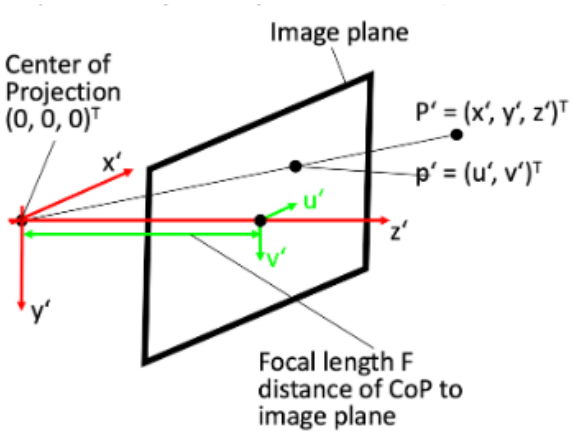
- Rotation und Translation in einer Matrix zusammengefasst
- Homogene Koordinaten
  - $(x, y, z, 1)$
  - Computer kann Drehungen und Verschiebungen in einem einzigen Schritt berechnen

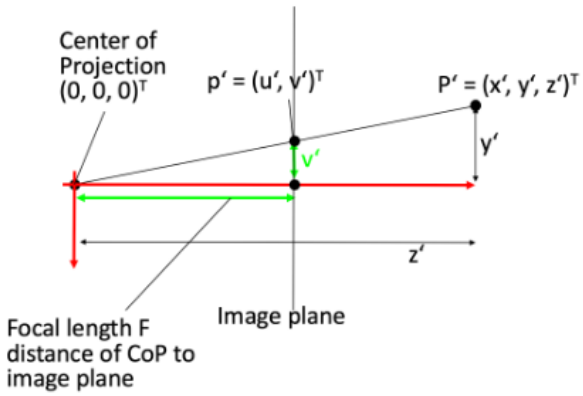
17.1.2. Intrinsische Transformation

DEFINITION: Berechnung der 2D-Punkte (u, v) auf dem Kamerasensor aus den 3D-Punkten (nach der extrinsischen Transformation).

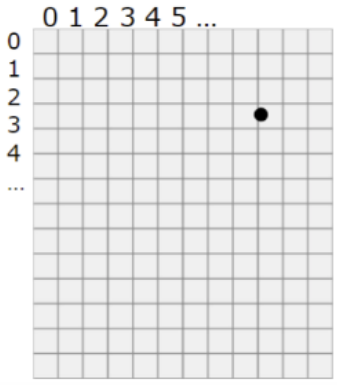
$$u' = \frac{F}{z'} \cdot x'$$

$$v' = \frac{F}{z'} \cdot y'$$





- u' und v' sind noch in 3D Koordinaten
- Die Skalierungsfaktoren (bezogen auf die physikalische Sensorgröße auf dem Chip) d_u und d_v werden in diskrete Pixelpositionen u'' und v'' umgewandelt.



$$u'' = d_u \cdot \frac{F}{z'} \cdot x'$$

$$v'' = d_v \cdot \frac{F}{z'} \cdot y'$$

Beispiel:

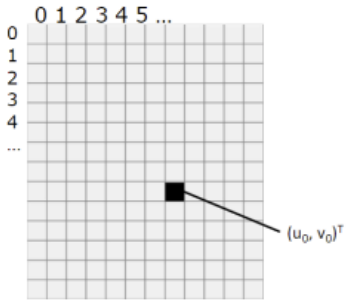
$$(u', v')^T = (5336.534, 243.983)^T$$

$$(u'', v'')^T = (357, 146)^T$$

- Die Verschiebung um den Hauptpunkt (u_0, v_0) (Schnittpunkt der optischen Achse mit der Bildebene) transformiert den Ursprung in die Bildecke (meist oben links).

$$u = d_u \cdot \frac{F}{z'} \cdot x' + u_0$$

$$v = d_v \cdot \frac{F}{z'} \cdot y' + v_0$$



Anschliessend wird die Gleichung mit z' multipliziert.

$$u \cdot z' = d_u \cdot F \cdot x' + u_0 \cdot z'$$

$$v \cdot z' = d_v \cdot F \cdot y' + v_0 \cdot z'$$

Als Matrixdarstellung:

$$z' \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} F \cdot d_u & 0 & u_0 \\ 0 & F \cdot d_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

- Damit kann man bereits weiterrechnen.
- Um z' zu eliminieren, werden homogene Koordinaten verwendet.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \begin{pmatrix} F \cdot d_u & s & u_0 \\ 0 & F \cdot d_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

$$= \boldsymbol{K} \cdot \boldsymbol{A} \cdot \boldsymbol{P'}$$

17.1.3. Kalibrierungsmatrix C

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} F \cdot d_u & s & u_0 \\ 0 & F \cdot d_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{T} \\ \boldsymbol{0}^\top & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\boldsymbol{p}_{\text{hom}} = \boldsymbol{K} \cdot \boldsymbol{A} \cdot \boldsymbol{E} \cdot \boldsymbol{P}$$
$$\boldsymbol{p}_{\text{hom}} = \boldsymbol{C} \cdot \boldsymbol{P}$$

17.1.3.1. Von homogenen Koordinaten zu 2D-Pixelkoordinaten

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$u = \frac{a}{c}$

$v = \frac{b}{c}$

17.2. Projektion p

DEFINITION: Ein 3D-Objekt auf einer 2D-Ebene abbilden.

- Wenn E (extrinsische Matrix) und K (intrinsische Matrix) bekannt sind, können 3D-Punkte in 2D-Bildpunkte projiziert werden.
- Beispiele: Augmented Reality

17.3. Kalibrierung C

DEFINITION: E und K aus bekannten 3D-Objektpunkten und den zugehörigen 2D-Bildpunkten berechnen.

- E (extrinsische Matrix) und K (intrinsische Matrix) bestimmen.
- mithilfe eines Kalibrierkörpers (**Marker**) mit bekannter Geometrie.

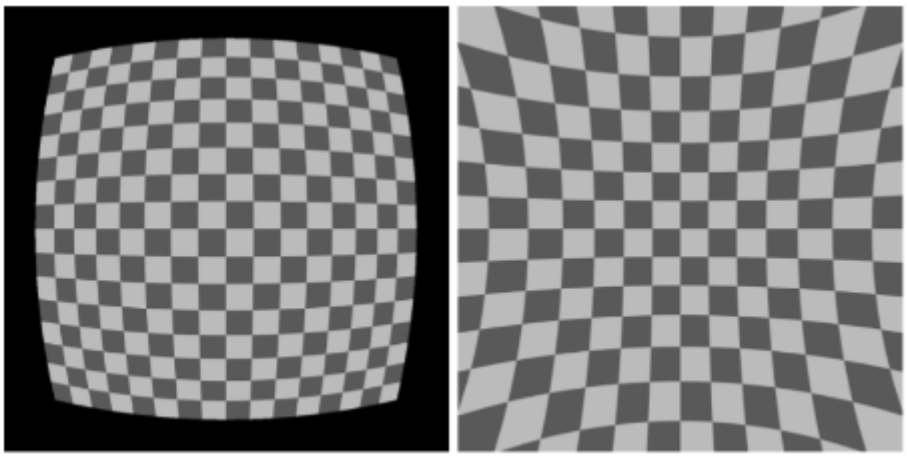
17.3.1. Direct Solution

Details muss man nicht kennen

Direct linear transform Algorithmus:

- **Ziel**: C finden (E und K)
- **Bekannt**: 3D-Weltpunkte P und deren 2D Pixel-Koordinaten p auf dem Foto
- Die Matrix C hat 11 Unbekannte (Freiheitsgrade). Da jedes Punktepaar genau 2 Gleichungen liefert (für die x- und y-Achse im Bild), brauchen wir mindestens 6 Punkte (rechnerisch 5,5), um alle Unbekannten lösen zu können.
- Man nimmt diese 6 Punktepaare, stapelt ihre Gleichungen übereinander und baut daraus eine große 12x12-Matrix (A). Durch Umstellen der Formel ($Ac \cdot (c) = 0$) lässt sich die gesuchte Kamera-Matrix C nun direkt mathematisch ausrechnen.
- **Fazit**: mindestens 6 bekannte 3D-zu-2D Punktepaare um daraus seine exakte Position und Optik (Kamera-Matrix) zu berechnen.

17.3.2. Lens Distortion



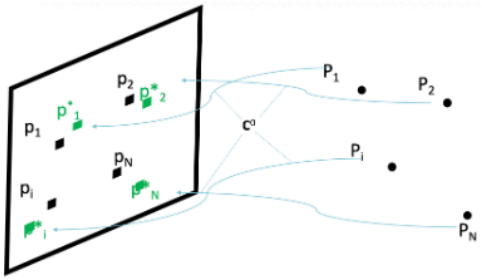
Tonnenförmig (Barrel): Typisch für Kameras. Kissenförmig (Pincushion): z. B. bei Mikroskopen.

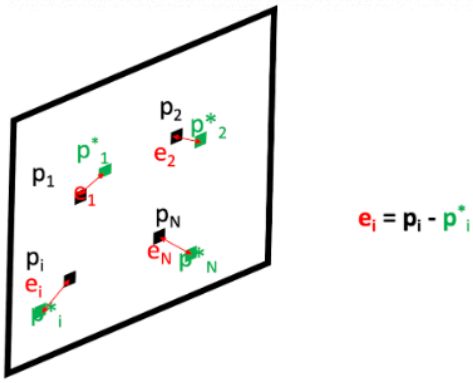
$$x_{\text{new}} = c_x + \frac{\hat{x}}{1 + k_1 \cdot r^2 + k_2 \cdot r^4 + ...}$$

$$y_{\text{new}} = c_y + \frac{\hat{y}}{1 + k_1 \cdot r^2 + k_2 \cdot r^4 + ...}$$

17.3.3. Reprojektionsfehler: Optimierung der Kameramatrix

- **Ausgangslage:** Die erste berechnete Matrix (C^0) ist nie zu 100 % exakt.
- **Der Test:** Echte 3D-Punkte (P_i) werden mit C^0 rechnerisch zurück auf das 2D-Bild projiziert (Ergebnis: p_i^*).
- **Der Reprojektionsfehler (e_i):** Die Pixel-Distanz zwischen den **echten** Bildpunkten (p_i) und den **berechneten** Bildpunkten (p_i^*).
- **Die Lösung (Nicht-lineare Minimierung):**
  - Deutlich **mehr als 6 Punkte** nutzen.
  - **Iteratives Verfahren** (z. B. Levenberg-Marquardt) anwenden.
  - Die Werte der Matrix C in kleinen Schritten anpassen.
  - **Ziel:** Die Summe aller Fehler im Bild (e_i^2) auf das absolute Minimum reduzieren.





$$e_i = p_i - p^*_i$$

$$\mathcal{L} = \sum_i \underbrace{\left\| \underbrace{\underbrace{(u,v)^\top}_{\mathbf{p}_i}}_{2\times 1} - \underbrace{\underbrace{\text{proj}(C\bar{\mathbf{P}}_i)}_{2\times 1}}_{\text{Corresponding 3D point } \mathbf{p}_i^* \text{ Projected by last estimated matrix}} \right\|^2}_{e_i} \qquad C = \arg\min_C \mathcal{L}$$

Image point

Reprojection error, quadratic

17.3.4. Probleme und Anwendungen

- Es ist noch unklar, wie man die tatsächlichen extrinsischen und intrinsischen Parameter aus **C** gewinnt.
- Andere direkte Ansätze, z. B. Tsai, können diese Trennung vornehmen.
- Ein moderner Standardansatz (State-of-the-Art): Zhangs Methode
- **Typisches Szenario in der Praxis:**
  - Kamera für ein System/eine Anwendung kalibrieren.
  - Physikalische Eigenschaften bleiben unverändert (intrinsische Parameter).
  - Meistens bleibt auch die Brennweite gleich.
  - Ziel: Kamera vorkalibrieren (intrinsische Werte berechnen).
  - Diese bleiben konstant.
  - Nur die extrinsischen Parameter müssen in Echtzeit geschätzt werden.
- Kalibrierung ist meist ein Offline-Vorbearbeitungsschritt.
- Dient zur Berechnung statischer intrinsischer Parameter.

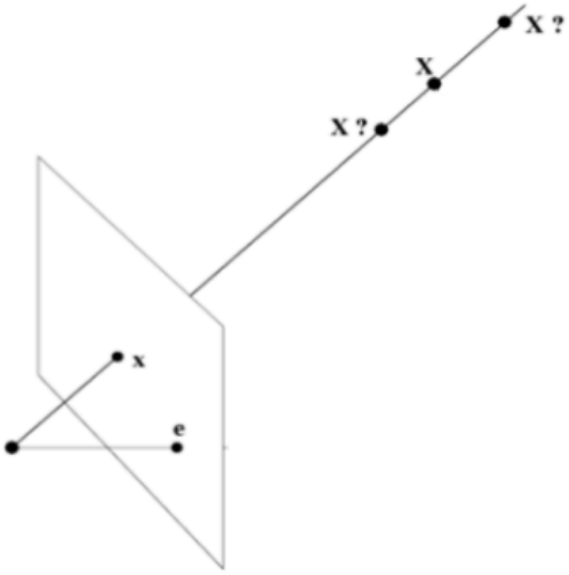
17.3.5. Open CV Function

- OpenCV 2.4: Camera Calibration and 3D Reconstruction
- OpenCV 3.4: Camera Calibration Tutorials (Python)

18. 3D Rekonstruktion II

WICHTIG: **Problem:** 3D-Informationsverlust bei der Bildaufnahme

- Die exakte Distanz (Z-Achse) geht verloren.
- Ein Bildpunkt definiert lediglich einen **Sichtstrahl**.
  - Pixel muss auf dem Strahl liegen

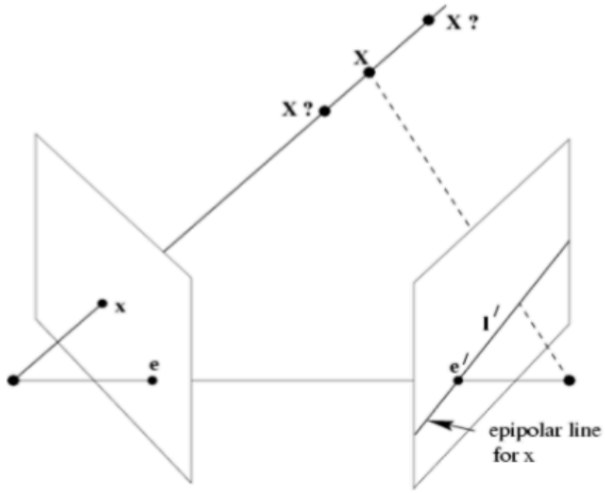


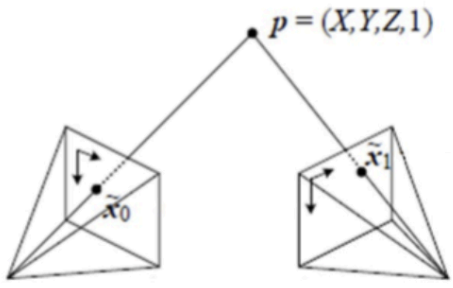
HINWEIS: **Lösung: shape from stereo**

18.1. Shape from Stereo

DEFINITION: 3D Rekonstruktion anhand 2 Bilder von verschiedenen Winkeln.

Prozess: Punkte vergleichen + Verschiebung (**Disparität**) messen. **Ergebnis:** Tiefe via **Triangulation** (3D-Rekonstruktion).





- Schnittpunkt zweier Sichtstrahlen bestimmt die Position im Raum.

- Rückgewinnung der Tiefe durch Abgleich unterschiedlicher Perspektiven.
- Wo sich Strahlen aus Kamera 1 und Kamera 2 treffen, liegt der 3D-Punkt p.
  - Durch Epipolare Geometrie muss nicht ganzes Bild abgesucht werden, sondern nur eine Linie

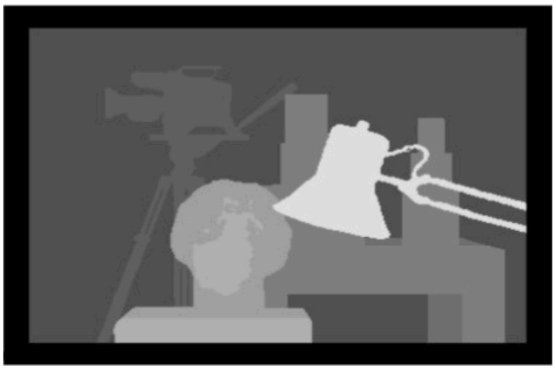
18.2. Disparität

DEFINITION: Verschiebung eines Punktes in zwei Bilder

$$D_x = x - x'$$

18.2.1. Graustufen

Verschiebung durch Grauwerte abgebildet:



- hell: grösser (256, ganz nah)
- dunkler: kleiner (0 Disparität, ganz weit hinten)
- rechts: durch einen Schätzalgorithmus
- links: vermutlich mit einem Laserscanner aufgenommen

18.2.2. Farben

Andere Darstellung der Disparität (ist ein Video, durch ein Schätzalgorithmus erstellt)

- grün: weit weg
- rot: nah
- Vorteil gegenüber Graustufen: man hat mehr Stufen als nur von 0 bis 256

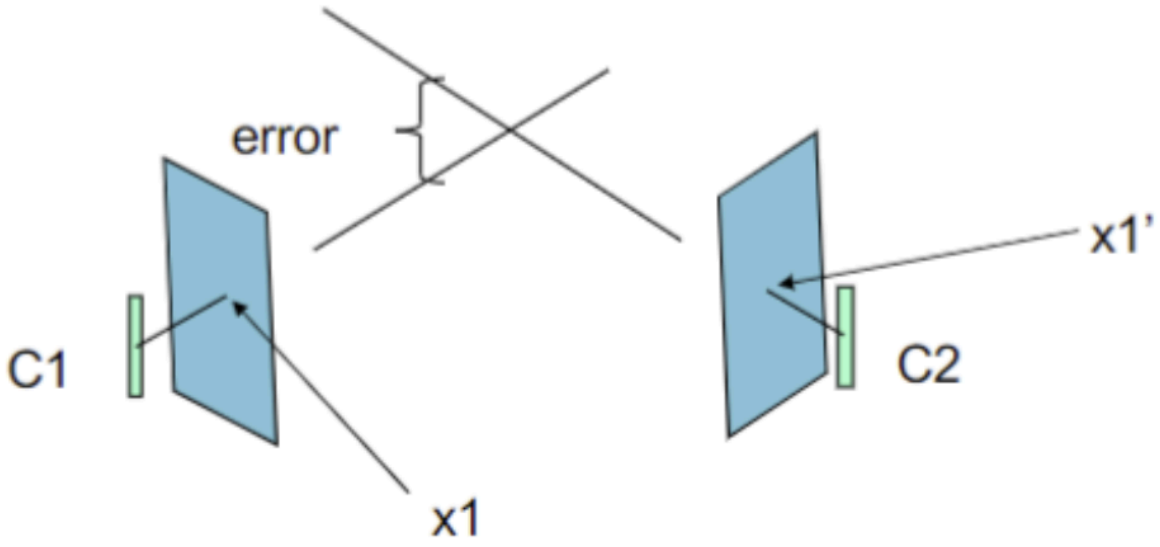
18.3. Triangulation

Problem: Ein exakter Schnittpunkt zweier Strahlen ist in der Praxis oft nicht berechenbar.

- wegen Rauschen/Ungenauigkeit

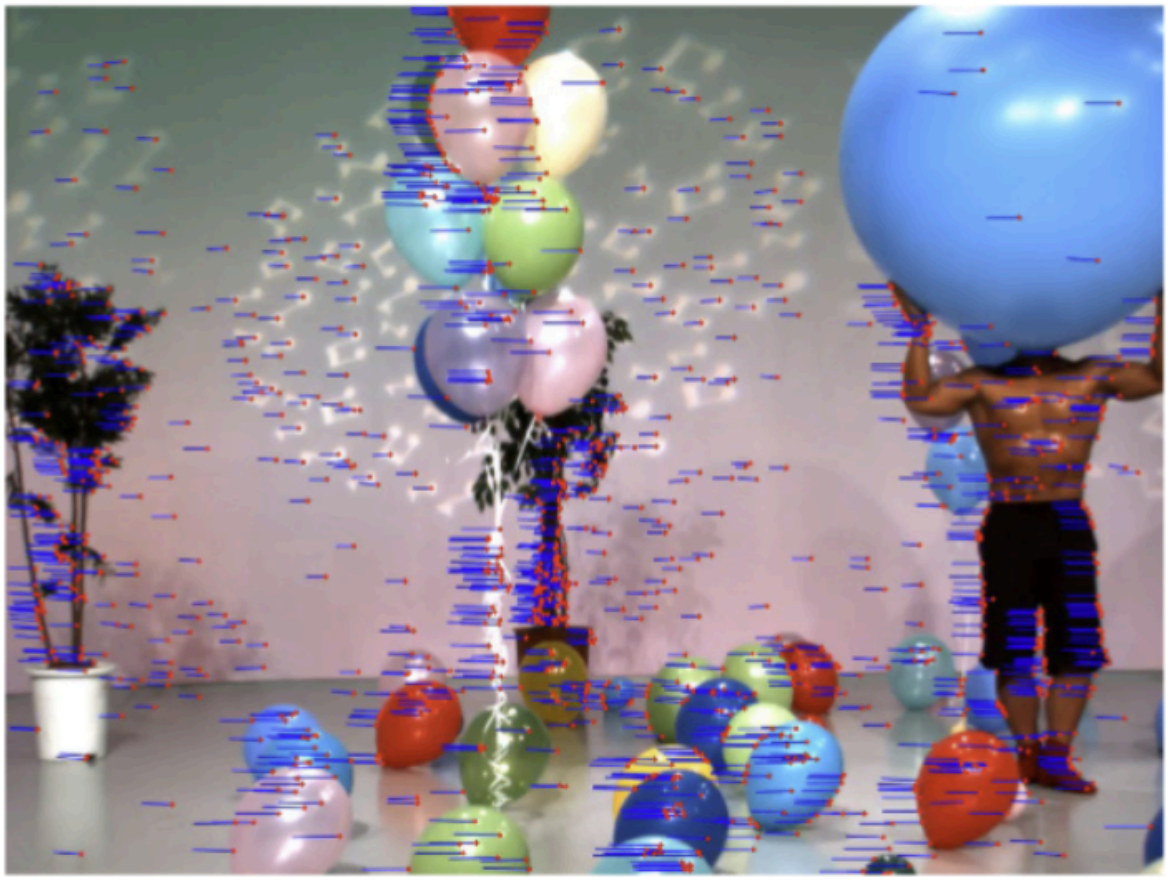
DEFINITION: Berechnung des Punktes mit dem geringsten Abstand zu beiden Geraden.

- Berechnung des 3D-Punktes aus Punktkorrespondenzen in den Bildern.
- als Matrix dargestellt



18.3.1. Sparse Disparity Estimation

- Feature matching
- kann zu unvollständigen Informationen führen



Überall wo das Feature Matching Fehler hat, wird man auch in der Tiefenschätzung Fehler haben.

- Rekonstruktion einer weissen Wand wird nicht gut funktionieren
- etwas mit klaren Ebenen / Strukturen wird besser funktionieren

18.3.2. Dense Disparity Estimation

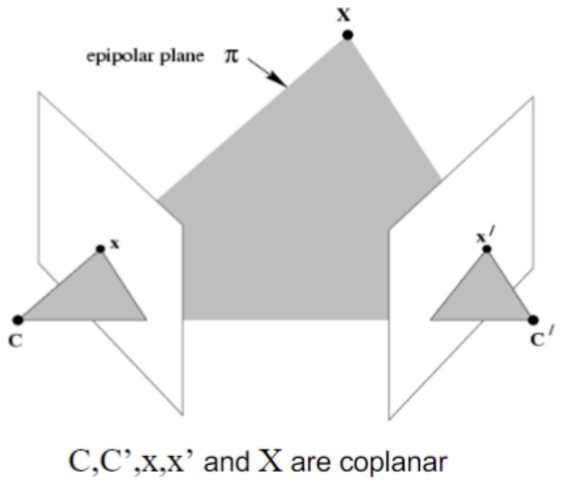
- wenn man auch andere Bereich haben möchte als nur Features
- Algorithmus berechnet disparity für jedes Pixel

18.4. Epipolare Geometrie

Definitionen

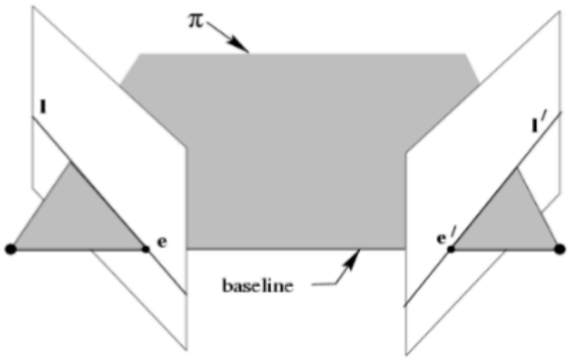
- **Epipol (e, e'):** Schnittpunkt der Baseline mit der Bildebene bzw. Projektion des anderen Kamera-Zentrums.
- **Baseline:** Verbindungslinie zwischen den beiden Kamera-Zentren.

- **Epipolarebene (π):** Ebene, welche die Baseline enthält.
- **Epipolarlinie (l, l'):** Schnittlinie der Epipolarebene mit der Bildebene; tritt immer paarweise auf.



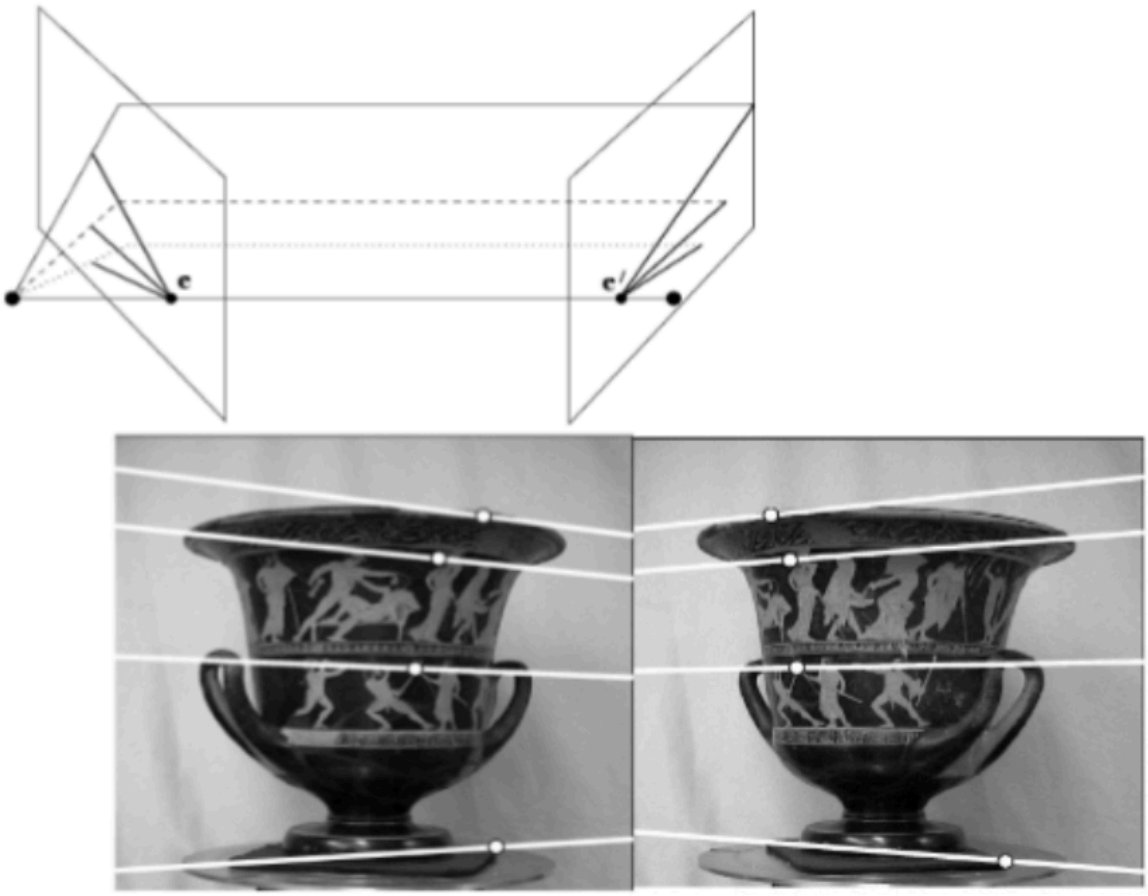
C,C',x,x' and X are coplanar

Suchbereich für korrespondierende Punkte wird von 2D (Bild) auf 1D (Linie) reduziert.



All points on π project on l and l'

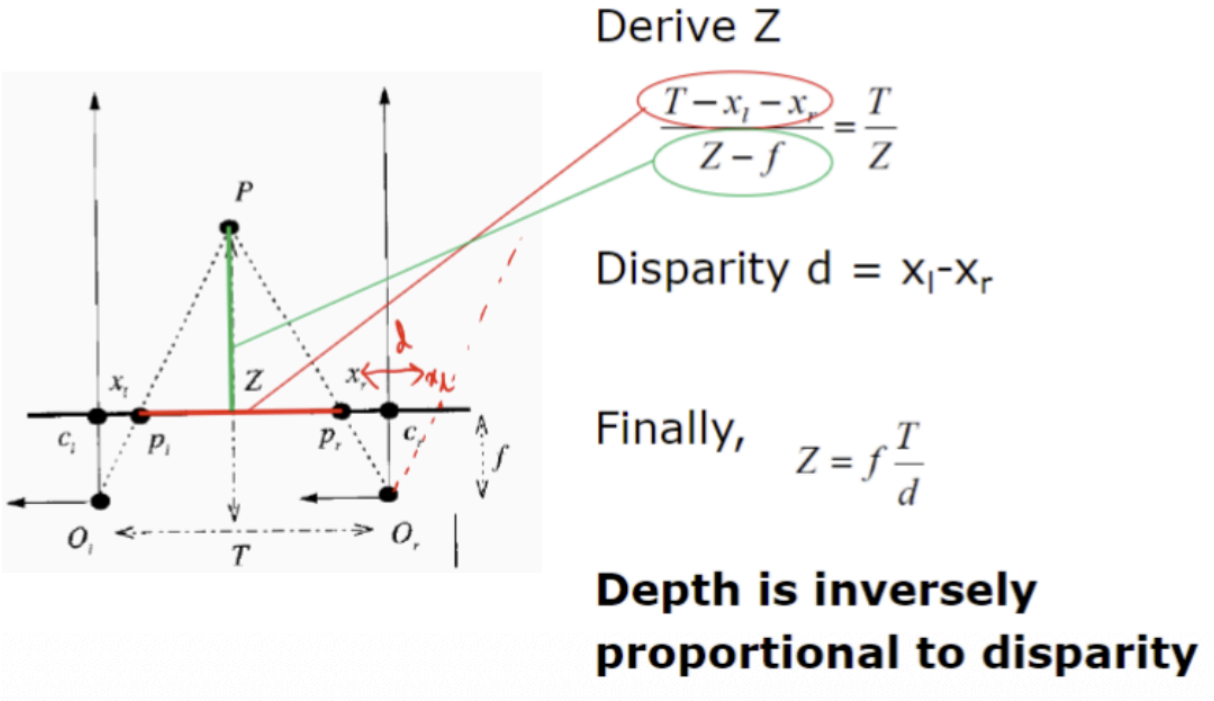
Alle epipolare planes gehen durch zwei Punkten und alle gehen durch die Baseline (geht auf wie ein Fächer)



- Punkte sind ausserhalb des Bildes, aber auf der Ebene
- die Bildebene ist mathematisch unendlich

18.4.1. Parallele Kameras

- **Geometrie:** Epipole liegen im Unendlichen.
- **Resultat:** Epipolarlinien sind **parallele horizontale Scanlinien**.
- **Matching:** Nur noch Suche entlang der identischen Bildzeile nötig (keine Geradengleichung).



18.4.2. Berechnung

- Kalibrierung des setups, z. B. mit einem Kalibriermuster, Schätzung der intrinsischen und extrinsischen Parameter
- Bildkorrektur, d. h. Projektion der Bilder auf eine gemeinsame Ebene parallel zur Basislinie
- Nun sind die Epipolarlinien parallel
- Suche nach correspondences entlang der image scan line
- Die geschätzten Disparitäten sind umgekehrt proportional zur Tiefe

Berechnung der Tiefe Z mittels Strahlensatz:

$$Z = f \cdot \frac{T}{d}$$

- $\boldsymbol{f}$**:** Brennweite
- $\boldsymbol{T}$**:** Baseline (Kameraabstand)
- $\boldsymbol{d}$**:** Disparität $(x - x') \rightarrow$ Versatz im Bild.

18.5. Rectification

DEFINITION: Projektion von Bildern nicht-paralleler Kameras auf eine gemeinsame, zur Grundlinie parallele Ebene.

- **Resultat:** Epipolarlinien werden exakt waagerecht.
- **Vorteil:** Feature Matching extrem vereinfacht (Suche nur noch auf gleicher Höhe/y-Achse).

18.6. OpenCV Functions

- https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
- https://docs.opencv.org/3.4/d9/db7/tutorial_py_table_of_contents_calib3d.html

18.7. Weitere Algorithmen

- Graph Cuts
- Belief Propagation
- Dynamic Programing
- Markov Random Fields

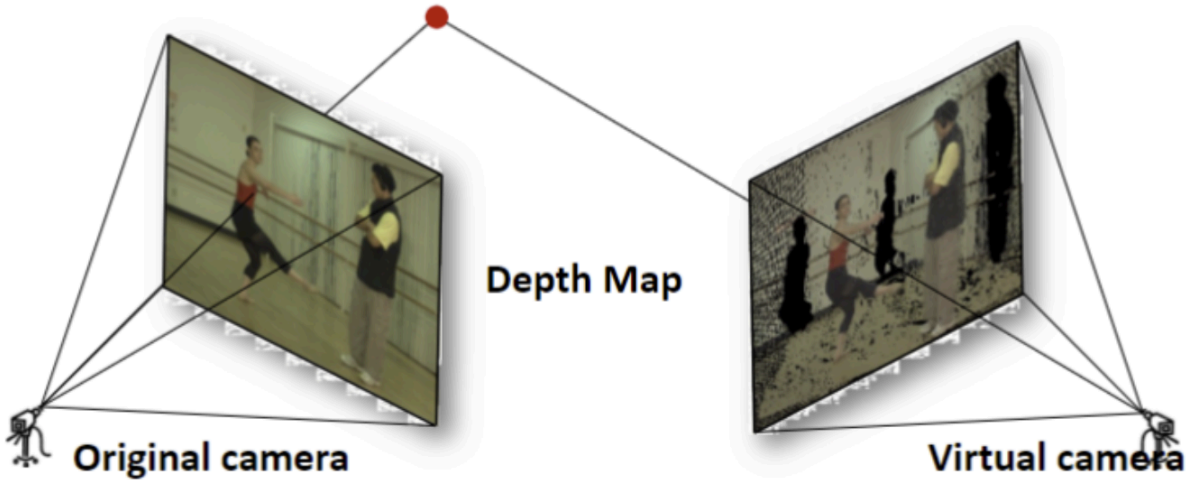
- Edge-aware diffusion
- ...
- **DEEP LEARNING**

Depth Anything: state of the art

18.8. Depth Image Based Rendering (DIBR)

DEFINITION: Erzeugung einer **virtuellen Kameraperspektive** aus einer realen Aufnahme.

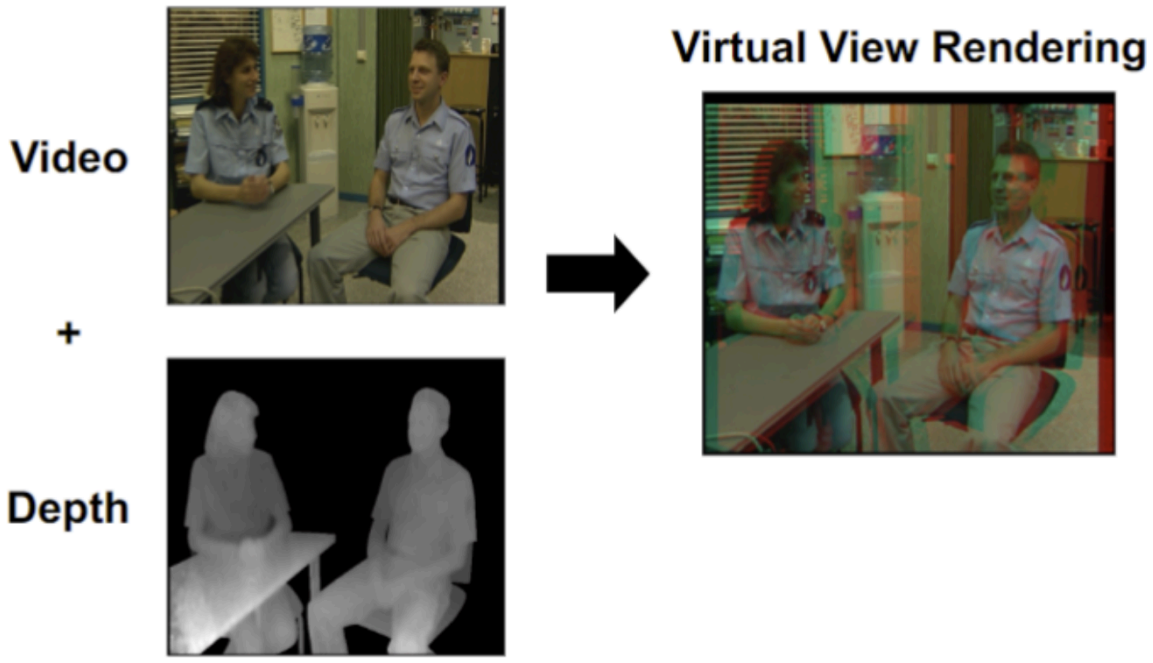
- **Voraussetzung:** Ein Originalbild + die dazugehörige **Tiefenkarte (Depth Map)**.
- **Funktionsweise:** Pixel werden anhand der Tiefendaten in den 3D-Raum projiziert und in die neue Ansicht (definiert durch Rotation R und Translation t) umgerechnet.
- **Das Problem (Disocclusions):**
  - Es entstehen schwarze Löcher
  - Diese entstehen, weil die virtuelle Kamera Bereiche „sieht„, die im Originalbild verdeckt waren. Da dort keine Farbinformationen vorliegen, bleibt das Bild an diesen Stellen leer.



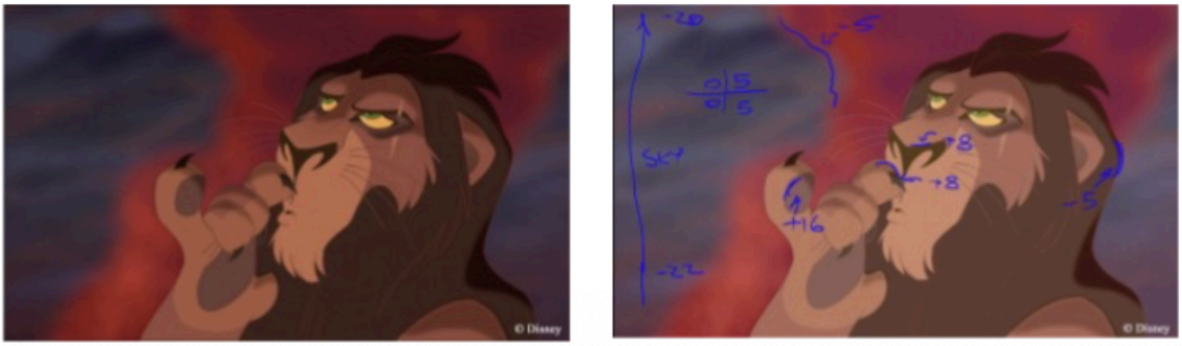
18.8.1. 2D to 3D

Das Ziel ist die nachträgliche Erzeugung räumlicher Tiefe aus flachem Quellmaterial.

- **Verfahren**: Ein 2D-Video wird mit einer **Depth Map** verrechnet, um eine neue virtuelle Ansicht (**Virtual View Rendering**) zu generieren.
- **Problem**: Durch die Perspektivverschiebung entstehen **Disocclusions** (Löcher im Bild), die mittels **Inpainting** (pixelweises Auffüllen) repariert werden müssen.



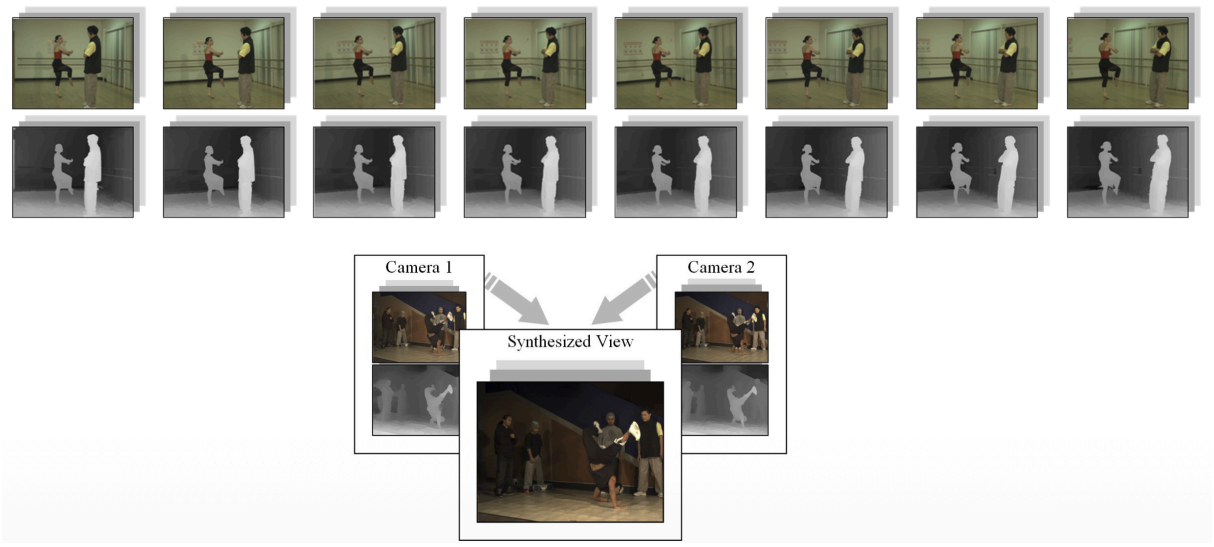
- **Manuelle Erstellung**: Da bei altem Material keine echten Tiefendaten vorliegen, werden diese von Künstlern manuell zugewiesen.



- **Technik**: Einzelnen Bildebenen wie Gesicht oder Hintergrund werden händisch relative Tiefenwerte (z. B. +16 oder −22) zugeordnet, um die Depth Map zu bauen.

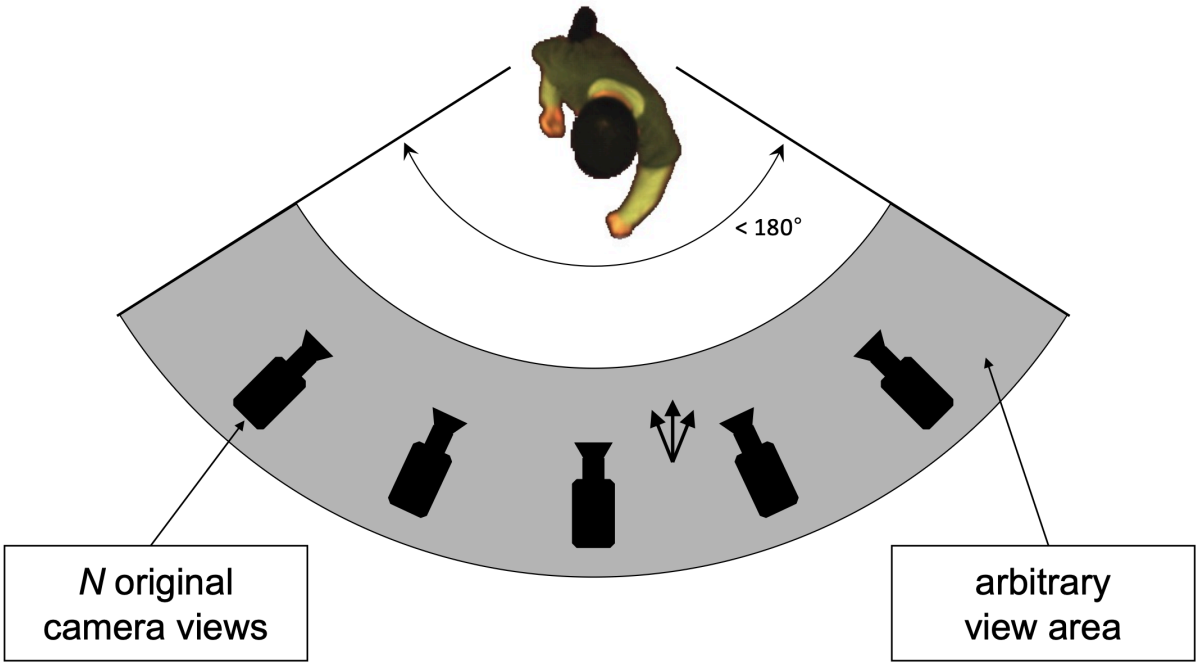
18.8.2. Multi-view Video plus Depth (MVD)

- **Aufbau:** Verwendung von mehreren (N) Videostreams, die jeweils mit eigenen Tiefeninformationen ausgestattet sind.
- **Verarbeitung:** Virtuelle Zwischenansichten werden durch Interpolation zwischen benachbarten Kamerapaaren („pair-wise switching“) berechnet.
- **Synthese:** Ein neues, künstliches Bild entsteht aus den Daten von echten Kameras plus deren Tiefenkarten.



18.8.2.1. Spatial Video Continuum

- **Das Ergebnis:** Die MVD-Technik erschafft einen zusammenhängenden, betrachtbaren Raum (arbitrary view area).
- **Kontinuierliche Bewegung:** Anstatt hart von Kamera zu Kamera zu springen, ermöglicht es stufenlose Kamerafahrten zwischen den Originalperspektiven.
- **Sichtbereich:** Der Nutzer kann sich frei innerhalb dieses durch die Kameras aufgespannten Bereichs (z. B. ein Bogen von < 180° um ein Objekt) bewegen.



18.9. Active Depth Estimation

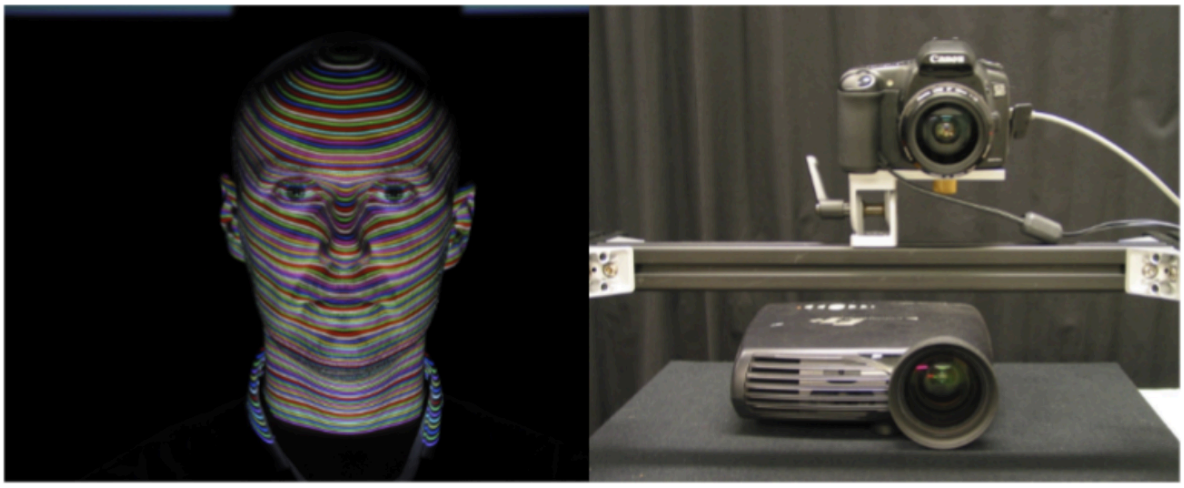
bisheriges: passive Tiefenschätzung

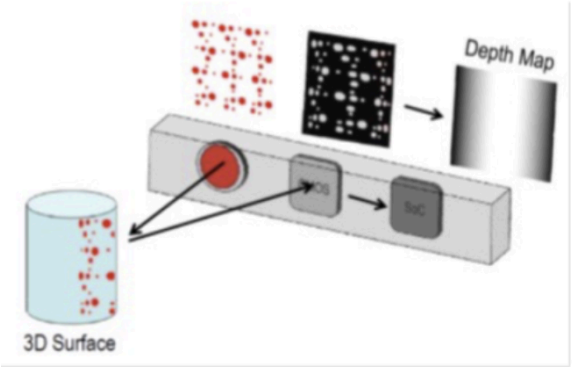
Zwei Varianten:

- **Time of flight**
- **Structured light**

18.9.1. Structured Light

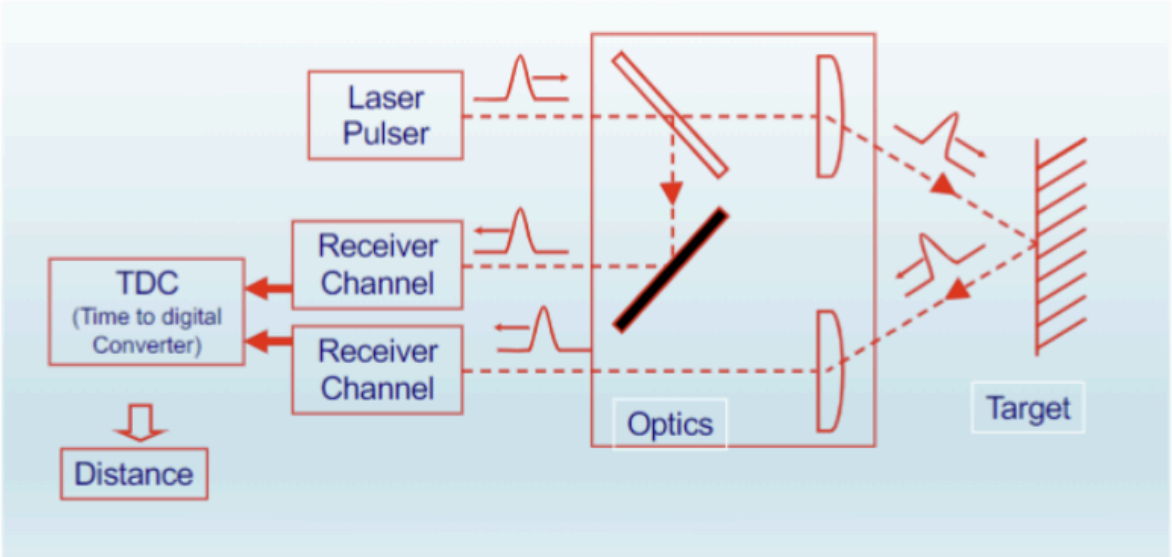
- Projiziert ein Lichtmuster in den Raum
- Tiefe wird aus der perspektivischen Verzerrung des Musters berechnet (z. B. FaceID)





18.9.2. Time of Flight

- Misst per Stoppuhr die Laufzeit eines ausgesendeten Lichtimpulses (Lichtgeschwindigkeit)
- Funktioniert wie ein Laserscanner/LiDAR.



19. 3D Rekonstruktion III

19.1. Problemstellung

WICHTIG: Weder die Kameraparameter (Kalibrierungsmatrix C) noch die 3D-Punkte im Raum (P) sind bekannt.

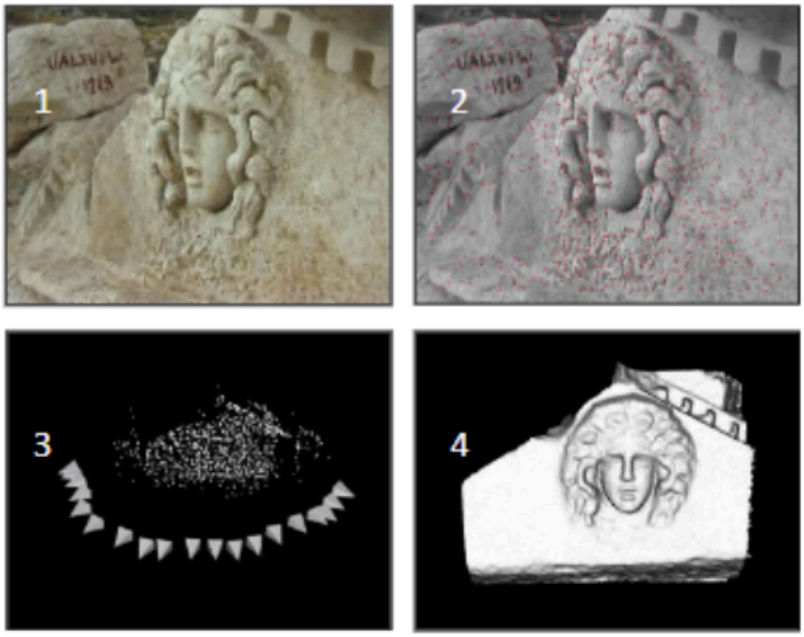
Voraussetzung zur Lösung:
- Wir können C und P nur rekonstruieren, wenn **mehrere Bilder derselben Szene aus unterschiedlichen Blickwinkeln** vorliegen (z. B. durch eine bewegte Kamera).

Lösungsansätze für dieses Problem:
- **Structure from Motion (SfM):** Berechnung im Nachhinein (Offline-Fall).
- **Simultaneous Localization and Mapping (SLAM):** Berechnung in Echtzeit.

19.2. Structure from Motion (SfM)

Der klassische SfM-Ablauf lässt sich in vier aufeinanderfolgende Hauptschritte unterteilen:

1. **Monoscopic video capturing:**
   - Aufnahme des Videos oder der Bilderserie mit einer einzelnen Kamera.
2. **Feature point extraction:**
   - **Corner point detection:** Finden markanter Punkte (wie Ecken) im Bild.
   - **Robust feature tracking:** Verfolgen dieser Punkte über mehrere Frames hinweg.
3. **Self-Calibration:**
   - Schätzung der intrinsischen und extrinsischen Kameraparameter.
   - Berechnung der 3D-Positionen der gefundenen Feature-Punkte (ergibt eine grobe, „sparse“ Point Cloud sowie den Kamerapfad).
4. **Dense disparity estimation:**
   - Berechnung von Tiefenwerten für jedes einzelne Pixel.
   - **Ziel:** Die eigentliche dichte 3D-Geometrie berechnen.
   - für z.B. selbstfahrende Autos nicht unbedingt notwendig



19.3. Stereo Constraints

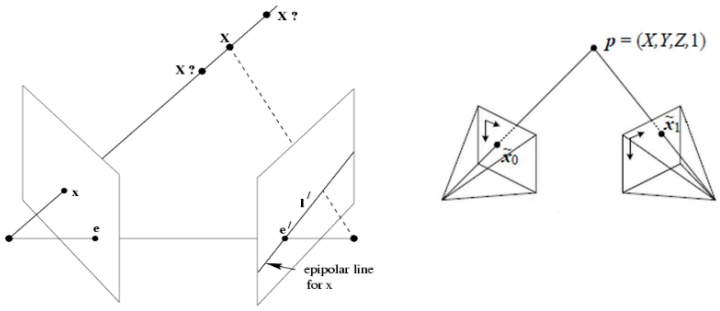
Eine zweite Ansicht ist zwingend nötig, um verlorene 3D-Informationen zurückzugewinnen (**Shape from Stereo**). Dabei müssen zwei Kernprobleme gelöst werden:

1. **Correspondence Geometry (Korrespondenz):** Gegeben Bildpunkt p in Bild 1 $\rightarrow$ Wo liegt der korrespondierende Punkt p' in Bild 2?

   (Gelöst durch: Epipolare Geometrie schränkt die Suche auf eine 1D-Linie ein).

2. **Scene Geometry (Struktur):** Gegeben korrespondierende Bildpunkte ($p_i \leftrightarrow p'_i$) und Kameras (C, C') $\rightarrow$ Wo liegt der ursprüngliche Punkt P im 3D-Raum?

   (Gelöst durch: Triangulation der Sichtstrahlen).



19.4. Fundamental & Essential Matrix

19.4.1. Fundamental Matrix (F) - Unkalibriert

DEFINITION: Die Fundamentalmatrix beschreibt die vollständige geometrische Beziehung eines Kamerapaares, wenn die Korrespondenzen der Bildpunkte gelöst sind.

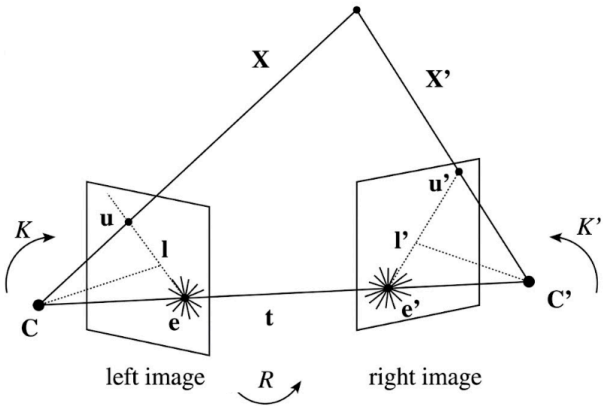
Zentrale Eigenschaft

- F kapselt alle intrinsischen (K) und extrinsischen (R, t) Kameraparameter und enthält somit alle Informationen über die Kamerageometrie.
- Sie basiert **nur** auf 2D-Bildpunkten, 3D-Koordinaten (P) entfallen komplett.
- Verknüpft direkt korrespondierende 2D-Punkte (u, u'):

$$u^T F u' = 0$$

Wichtig zu merken:
- Kalibrierung 3D Punkte zu 2D Punkte
- Fundamental matrix: man muss die 3D Punkte nicht mehr kennen, man braucht nur noch Bildpunkte
- d.h. man muss nur noch Bildpunkte tracken und mit Hilfe der fundamental matrix kann man alle Parameter berechnen die man benötigt

19.4.1.1. Herleitung



- **Voraussetzung:** Optische Achsen sind nicht parallel.

- **Komplanarität:** Die 3D-Sichtstrahlen beider Kameras und der Verschiebungsvektor (t) liegen in einer Ebene (ergibt eine Nullgleichung).
- **Rechentrick:** Das Kreuzprodukt mit t wird durch eine schiefsymmetrische Matrix $S(t)$ ersetzt, um die Gleichung zu vereinfachen.
- **Zusammenfassung:** Alle konstanten Kameraterme ergeben zusammen die Matrix F:

$$F = \left(K_1^{-1}\right)^T S(t) R^{-1} K_2^{-1}$$

$$\mathbf{X}_L^T \cdot (\mathbf{t} \times \mathbf{X}'_L) = 0$$

$$(\mathbf{K}^{-1}\mathbf{u})^T \cdot (\mathbf{t} \times \mathbf{R}^{-1}\mathbf{K}'^{-1}\mathbf{u}') = 0$$

$$\mathbf{S(t)} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

$$(\mathbf{K}^{-1}\mathbf{u})^T \left(\mathbf{S(t)}\, \mathbf{R}^{-1}\, \mathbf{K}'^{-1}\mathbf{u}'\right) = 0$$

$$\mathbf{u}^T(\mathbf{K}^{-1})^T\mathbf{S(t)}\mathbf{R}^{-1}\mathbf{K}'^{-1}\mathbf{u}' = 0$$

$$\mathbf{F} = (\mathbf{K}^{-1})^T\mathbf{S(t)}\mathbf{R}^{-1}\mathbf{K}'^{-1}$$

19.4.1.2. Schätzung

1. **Feature Matching:** Korrespondierende Punkte in beiden Bildern finden.
2. **Berechnung:** Mindestens **7 Punktpaare** (besser mehr) nutzen, um das Gleichungssystem für F initial zu lösen.
3. **Optimierung:** Iterative, nicht-lineare Minimierung des „Reprojection Errors“ zur exakten Abstimmung.

19.4.2. Essenzielle Matrix (E) - Vorkalibriert

DEFINITION: Gegenstück zur Fundamentalmatrix für den Fall, dass die Kameras bereits vorkalibriert sind, die intrinsischen Parameter K sind also bekannt.

Zentrale Eigenschaft

- Da die Kamerageometrie (K) bereits bekannt ist, kapselt E ausschliesslich die relative Bewegung (Rotation R und Translation t) zwischen den zwei Ansichten.

Eigenschaften und Zusammenhänge:

- **Normierung:** Im Gegensatz zu F arbeitet E mit „normierten" Bildkoordinaten. Diese berechnen sich aus den Bildpunkten (u) und der inversen Kalibrierungsmatrix (K^{-1}):

$$\check{u} = K^{-1}u \quad \text{und} \quad \check{u}' = K'^{-1}u'$$

- **Kerngleichung:** Daraus ergibt sich die vereinfachte Gleichung der Essenziellen Matrix:

$$\check{u}^T E \check{u}' = 0$$

- **Beziehung zu F:** Die Essenzielle Matrix lässt sich direkt aus der Fundamentalmatrix und den Kalibrierungsmatrizen ableiten:

$$E = K'^T F K$$

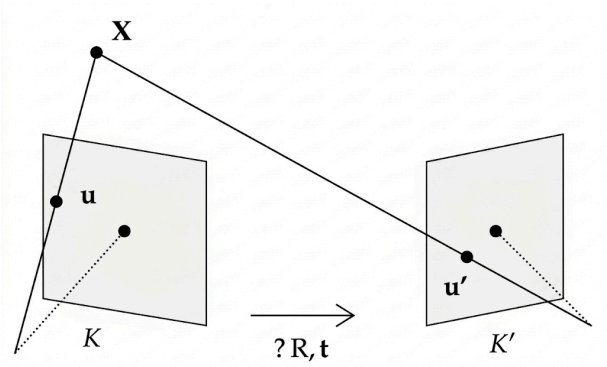
- **Nutzen:** Sobald die Matrix E berechnet wurde, lassen sich daraus die genauen Parameter für die Kamerabewegung (R und t) extrahieren (meist mittels Singularwertzerlegung / SVD).

19.4.2.1. Ego Motion Estimation (Eigenbewegung)

- **Ziel:** Schätzung der eigenen Kamerabewegung.
- **Voraussetzung:** Kameras sind bereits vorkalibriert. Es werden **nur noch** die extrinsischen Parameter (Rotation R und Translation t) gesucht.
- Begriff kommt aus der Robotik, ein autonomes System schätzt seine eigene Bewegung
- Translation und Rotation können mithilfe der ego motion estimation berechnet werden, damit kann sich das System im Raum organisieren.

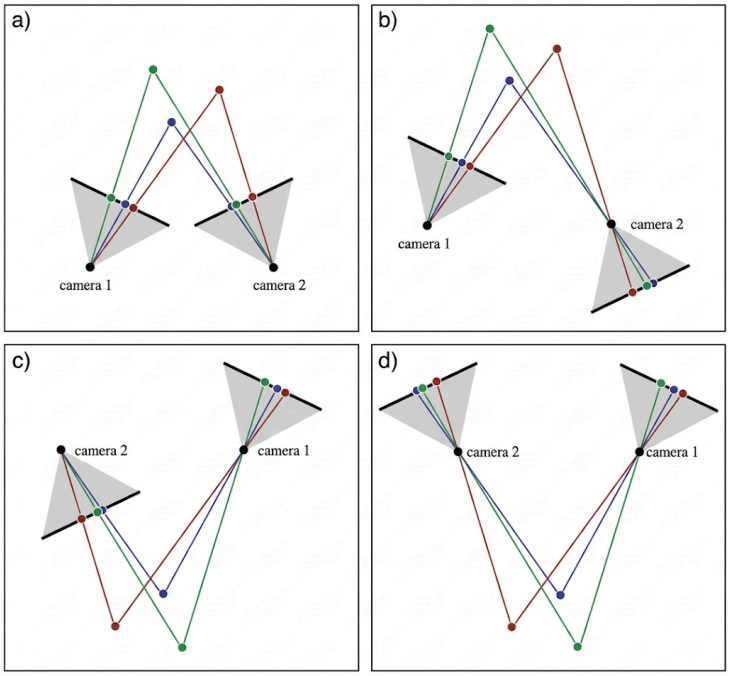
Algorithmus:

1. Bildkorrespondenzen (u_i, u_i') finden.
2. Bilddaten normieren.
3. Essenzielle Matrix (E) berechnen.
4. R und t aus E extrahieren (z. B. mittels Singularwertzerlegung / SVD).



19.4.2.2. Four-fold Ambiguity

- **Problem:** Die mathematische Extraktion von R und t aus E liefert **vier** mögliche geometrische Lösungen für die Kamerapositionen. → **System ist nicht eindeutig**
- **Lösung (Disambiguierung):** Von diesen vier Lösungen ist nur **eine einzige** in der Realität gültig. Die korrekte Konfiguration ist jene, bei der die rekonstruierten 3D-Punkte zwingend **vor** beiden Kameras liegen. → im Beispiel a (b,c,d sind hinter der Bildebene)


a)
camera 1
camera 2
b)
camera 1
camera 2
c)
camera 2
camera 1
d)
camera 2
camera 1

19.5. 2 Arten der 3D-Rekonstruktion

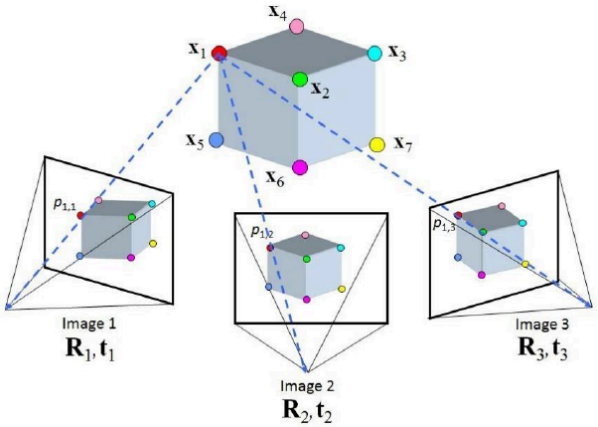
1. **Mit bekannten Markern:**
   - Kalibrierungsmatrix (C) und 3D-Punkte (P) sind bekannt.
   - **Projektionsgleichung**: $p = C \cdot P$
2. **Ohne Marker:**
   - Nur 2D-Bildkorrespondenzen vorhanden.
   - Berechnung via Fundamentalmatrix (F).
   - **Epipolargleichung**: $p_1^T \cdot F \cdot p_2 = 0$

19.6. Multi-View Structure from Motion

Für ein einzelnes Kamerapaar lässt sich SfM via F und E lösen (liefert Rotation, Translation, 3D-Punkte).

Herausforderung bei Bildsequenzen:

- Konsistenz der 3D-Szene über viele Frames sichern.
- **Lösung:** Schätzungen verknüpfen, propagieren und simultan optimieren.

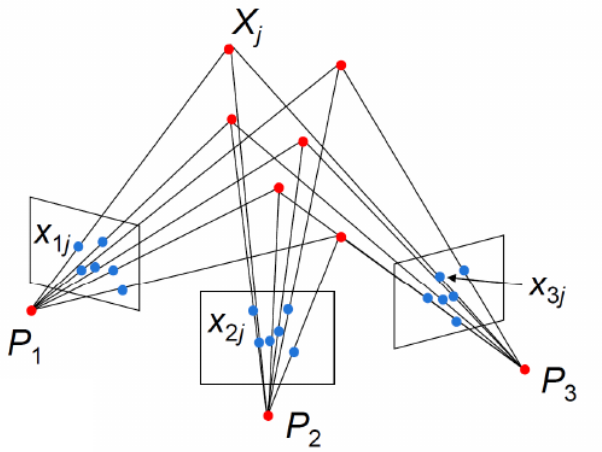


19.6.1. Mathe-Modell (m Kameras, n Punkte)

- **Gegeben:** m Bilder von n 3D-Punkten ($m \times n$ Korrespondenzen).

$$z_{ij} x_{ij} = P_i X_j \quad \text{für} \quad i = 1, ..., m \quad \text{und} \quad j = 1, ..., n$$

- **Ziel:** Aus $m \times n$ 2D-Korrespondenzen (x_{ij}) gleichzeitig m Projektionsmatrizen (P_i) und n 3D-Punkte (X_j) schätzen.



19.6.2. Inkrementelles Structure from Motion (SfM)

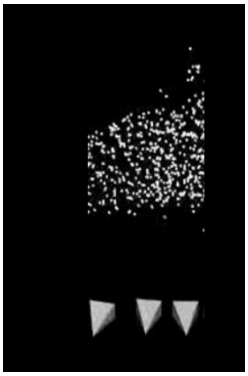
DEFINITION: Für Bildsequenzen wird SfM schrittweise (inkrementell) aufgebaut, um die 3D-Szene konsistent zu erweitern.

Ablauf in drei Phasen:

1. **Initialisierung (Kamera 1 & 2):**
   - Start mit den ersten zwei Ansichten (Berechnung via Fundamentalmatrix).
   - Kamera 1 dient als Ursprung/Referenz ($R_1 = I$, $t_1 = 0$).
   - Berechnung der Parameter R_2 und t_2 für die zweite Kamera.
   - Rekonstruktion einer ersten 3D-Teil-Punktwolke (P_{1-2}).
2. **Erweiterung (Kamera 3 hinzufügen):**
   - Die bereits bekannten 3D-Punkte (P_{1-2}) fungieren nun als „Marker“ (Kalibrierungsobjekt).
   - Darüber lassen sich die Parameter der neuen Kamera (R_3, t_3) direkt schätzen.
3. **Punktwolke aktualisieren:**
   - **Neue Punkte:** Triangulation von Punkten, die nur in Bild 2 und 3 sichtbar sind ($P_{2-3} \setminus P_{1-2}$).
   - **Verfeinerung:** Aktualisierung und Verbesserung der bereits bekannten, überlappenden Punkte ($P_{2-3} \cap P_{1-2}$).

Ergebnis nach N Kameras:

- Das System hat nun N Rotationen (davon $1 \times$ Identität), N Translationen (davon $1 \times$ Nullvektor) und eine global konsistente Punktwolke mit insgesamt M 3D-Punkten berechnet.



19.6.3. Bundle Adjustment

- **Problem:**
  - Da bei jedem neu hinzugefügten Bild kleine Messfehler entstehen (z. B. durch ungenaues Feature-Tracking), summieren sich diese Fehler über die Zeit auf.
  - Die berechnete 3D-Welt verbiegt und verzerrt sich im Laufe der Sequenz zunehmend (Drifts).
- **Lösung:**
  - Das Bundle Adjustment korrigiert diesen kumulierten Fehler am Ende durch eine einzige globale Optimierung.

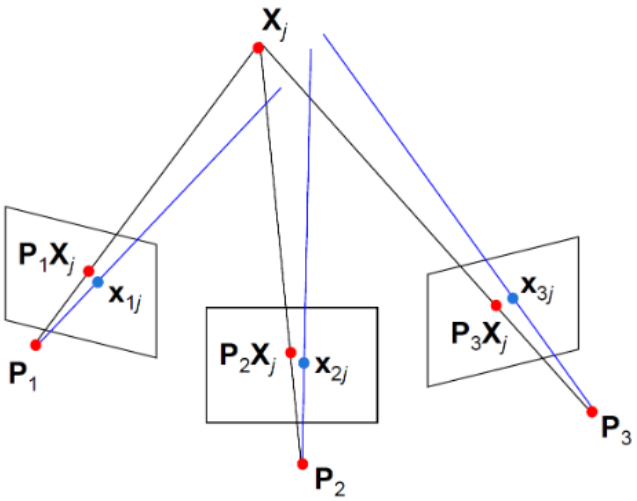
WICHTIG: Das Bundle Adjustment ist ein grosses mathematisches Verfahren, das alle bisherigen Daten (N Kamerapositionen und M 3D-Punkte) simultan anpasst, um ein in sich konsistentes Gesamtergebnis zu erhalten.

Ablauf:

- Ein Bündel von Sichtstrahlen wird mathematisch so justiert, dass systemweite Fehler minimal werden.
- **Loss-Funktion (Reprojection Error):** Ein berechneter 3D-Punkt wird virtuell auf das 2D-Bild zurückprojiziert. Ziel ist es, den Pixel-Abstand (D) zwischen dieser Projektion und dem echten Feature-Punkt für alle Punkte und Kameras zu minimieren

$$E(P, X) = \sum_{i=1}^{m} \sum_{j=1}^{n} D\left(x_{ij}, P_i X_j\right)^2$$

- **Aufwand:** Da Tausende Parameter (Punkte und Kamerapositionen) gleichzeitig optimiert werden müssen, ist das Verfahren extrem rechenintensiv



19.7. SfM vs. SLAM

In der Praxis gibt es eine klare Aufgabenteilung zwischen den beiden Verfahren, abhängig davon, ob Zeit oder Genauigkeit wichtiger ist.

Structure from Motion (SfM):

- **Stärken:** Liefert hochpräzise 3D-Rekonstruktionen von statischen Szenen.
- **Schwächen:** Extrem rechenintensiv und komplex, durch wiederholte Optimierungen und Bundle Adjustment
- **Einsatz:** Architektur, Film, Geowissenschaften.
- **Fazit: Nicht echtzeitfähig**

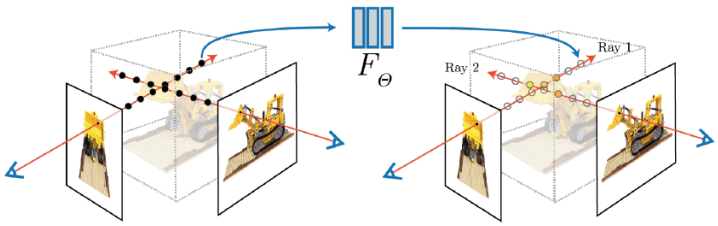
SLAM (Simultaneous Localization and Mapping):

- **Fokus:** Ein System (Auto, Drohne, AR-Brille) muss sich in einer unbekannten Umgebung **in Echtzeit** selbst lokalisieren und diese gleichzeitig kartieren.
- **Der Kompromiss:** SLAM tauscht absolute Genauigkeit gegen Echtzeit-Performance ein.
- **Funktionsweise:**
  - Nutzt Kontrolltheorie und Sensordaten-Fusion (Kameras, Lidar, Gyroskop).
  - Es wird ein Zustandsvektor X_t (Position, Orientierung, 3D-Punkte) definiert, der Frame für Frame über ein Wahrscheinlichkeitsmodell aktualisiert wird ($X_t \rightarrow X_{t+1}$).
- **Lösungsverfahren:** Meistens basierend auf Algorithmen wie dem Kalman-Filter oder Partikel-Filter.
- **Einsatz:**
  - Robotik, Drohnen, autonome Fahrzeuge, AR/VR
  - Kann markerbasiert ($p = C \cdot P$) oder markerlos ($p_1^T \cdot F \cdot p_2 = 0$) arbeiten.

19.8. Ausblick: Neural Radiance Fields (NeRF)

Könnte dies das klassische SfM in Zukunft ersetzen?

- **Konzept:** NeRF ist ein moderner Deep-Learning-Ansatz. Anstatt eine explizite 3D-Punktwolke zu berechnen, lernt ein neuronales Netzwerk die Szene als kontinuierliches, volumetrisches Strahlungsfeld.
- **Ziel (View Synthesis):** Aus einer begrenzten Anzahl an Eingabebildern können extrem fotorealistische, komplett neue Kameraperspektiven (Novel Views) flüssig gerendert werden.



20. 3D Rekonstruktion IV

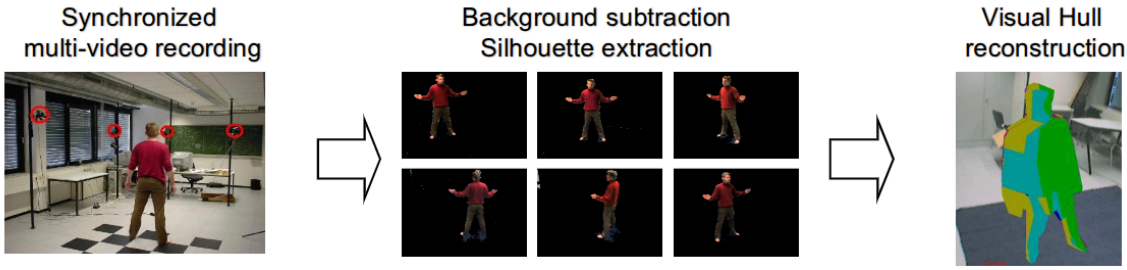
20.1. Volumetric Video

Ein Volumetric Video hat die gleiche Funktionalität wie ein Computer Graphic Object:
- freie Navigation
- kann aus jedem Winkel/Perspektive betrachtet werden
- kann in Szenen (virtuell, augmented, real) integriert werden

Unterscheidung zu Computer Graphic:
- Modellieren Aussehen von einem realen Objekt und entstehen auch aus Bildern von diesem realen Objekt

20.1.1. Shape from Silhouette (SfS) & Visual Hull

Für die Erstellung der Visual Hull sind folgende Schritte notwendig:
1. Synchronized Multi-video recording
2. Background Substraction und Silhouette extraction
3. Visual Hull reconstruction



20.1.1.1. Visual Hull Reconstruction (Voxel-Modellierung)

DEFINITION: Ein Voxel (Volumetric Pixel) ist ein Würfel mit räumlicher Ausdehnung

$$v(x,y,z) \quad \texttt{abs} \quad x \in [x_{\min}, x_{\max}], y \in [y_{\min}, y_{\max}], z \in [z_{\min}, z_{\max}]$$

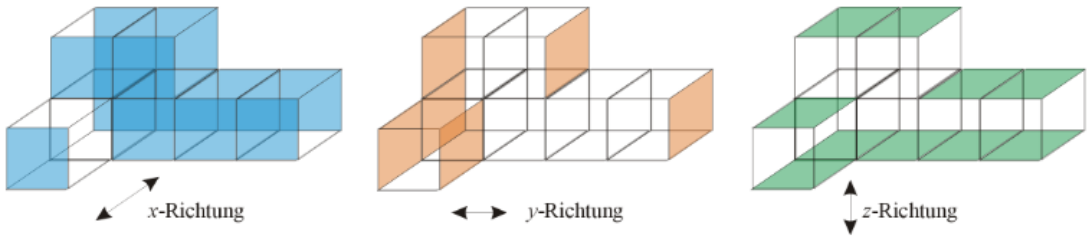
Vorgehen (Voxel Carving): Für die Rekonstruktion werden aus den verschiedenen Kameras virtuelle Strahlen durch ein Voxel-Gitter gezogen:
- Voxel, die als Hintergrund gesehen werden, werden gelöscht (Empty Space).
- Voxel, die aus allen Perspektiven innerhalb der Silhouette liegen, bleiben übrig und bilden die **Visual Hull**.

Fehler in der Silhouette führen auch zu Fehlern in der Volume Reconstruction.

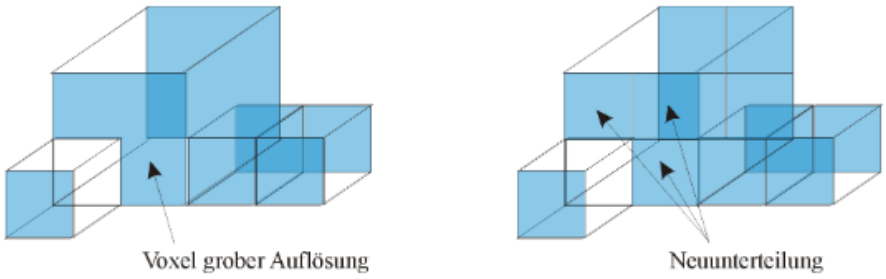
Voxel-Strukturen:
- **Uniform:** Alle Voxel sind gleich gross.
- **Octree (Hierarchisch):** Grosse Voxel füllen das Innere, an der Aussenfläche werden sie in immer kleinere Würfel unterteilt.

20.1.1.2. Wireframe Transformation

Um aus dem massiven Voxel-Volumen der Visual Hull die reine Oberfläche (ein 3D-Mesh) zu extrahieren, wird die Wireframe Transformation eingesetzt. Dabei werden lediglich die äussersten, begrenzenden Flächen der übrig gebliebenen Voxel in den 3 Richtungen detektiert.



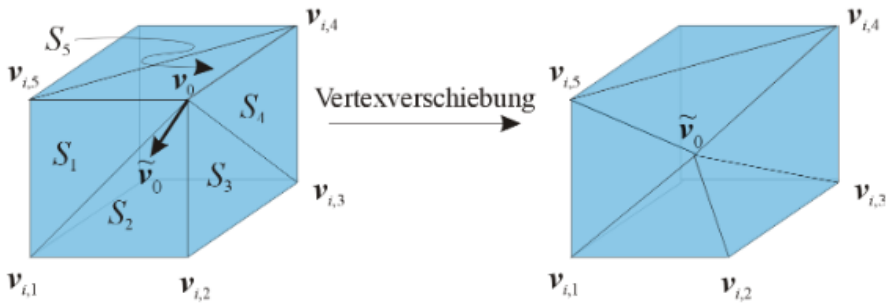
Da die Voxel unterschiedliche Auflösungen haben (wie bei der **Octree Voxel Reconstruction** beschrieben), müssen sie unterteilt werden, wenn die Aussenfläche unterschiedliche Hierarchiestufen hat.



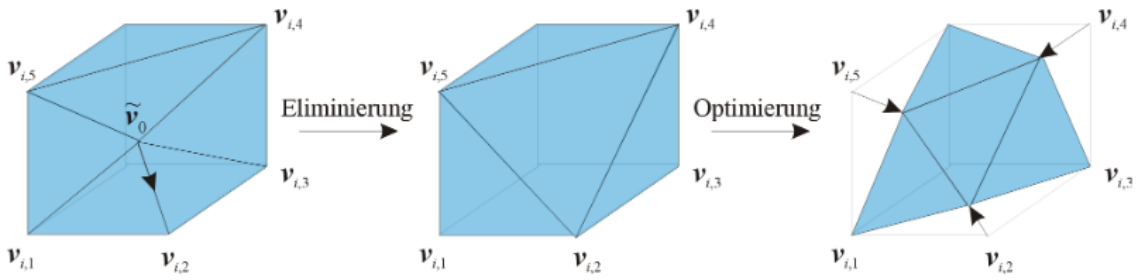
20.1.1.3. Wireframe Reduktion

Damit das Wireframe nun weniger eckig ist, wird Edge Collapsing und Surface Smoothing angewandt.

Surface Smoothing by vertex shifting: Beim Surface Smoothing werden hervorstehende Ecken abgeflacht.



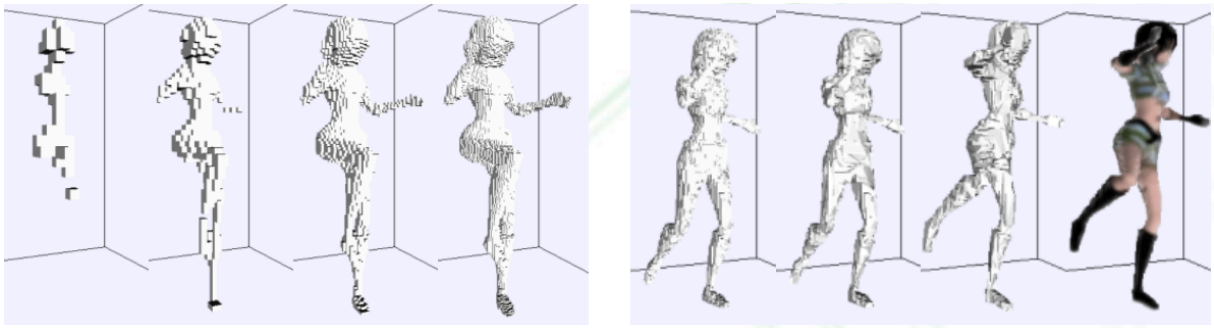
Edge Collapsing und Optimierung: Beim Edge Collapsing werden die abgeflachten Ecken komplett eliminiert und an die tatsächliche Oberfläche angeglichen.



20.1.1.4. Texturing

Beim Texturing wird dem erstellten Mesh eine Textur verliehen, um das Aussehen dem realen Objekt anzupassen.

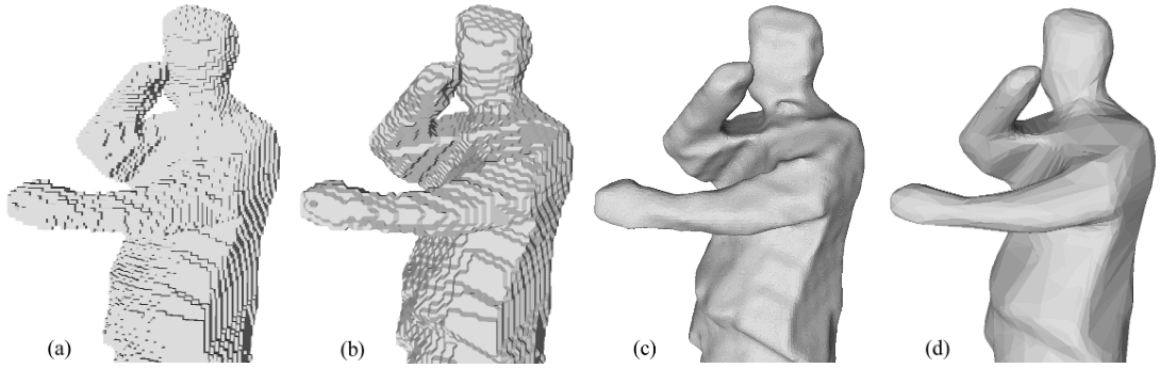
20.1.1.5. Gesamtablauf



20.1.2. Shape from Silhouette (Erweiterter Ablauf für Volumetric Video)

Diese Methode eignet sich primär für grobe Strukturen und Setups mit weniger Kameras. Feine Details und Einbuchtungen am Objekt können oft nicht exakt erfasst werden.

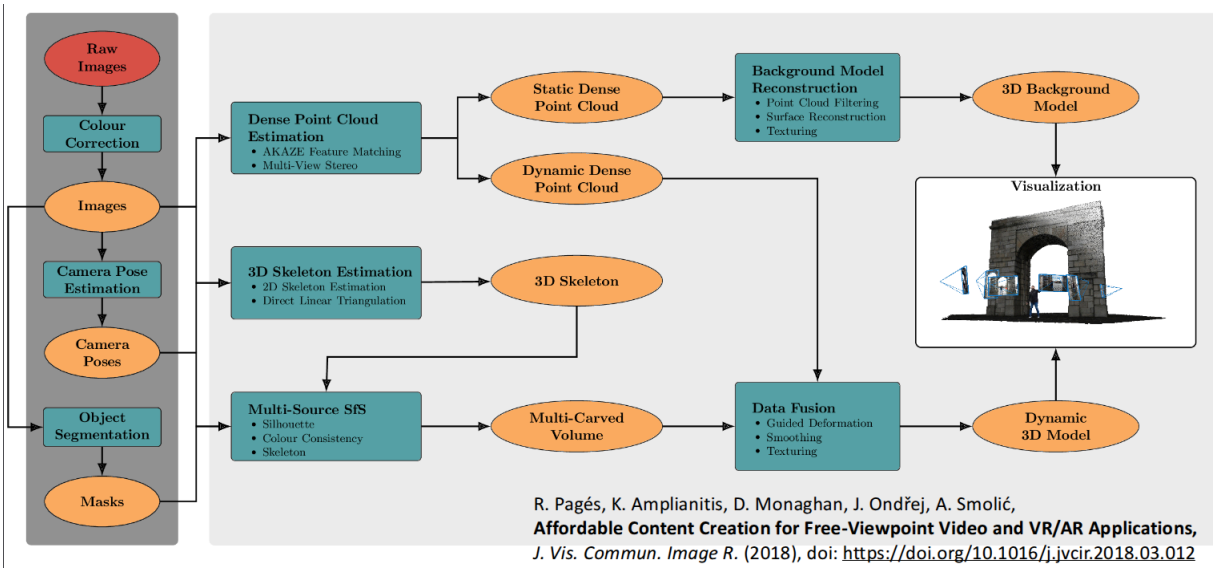
1. Hierarchical Voxel Modeling (z. B. via Octree)
2. Marching-Cubes-Approach
3. Surface Smoothing
4. Mesh Reduction



20.1.3. Hybrid Approach

1. Input von Structure from Motion (SfM) Volumetric Video
2. Input von Shape from Silhouette (SfS)
3. Input von Vorwissen über die menschliche Anatomie und Kinematik (z. B. ein 3D-Skelettmodell)

Die Kombination dieser Inputs geschieht im Schritt **Data Fusion**. Das Volumen (aus SfS) und die Punktwolke (aus SfM) werden mithilfe des 3D-Skeletts durch eine sogenannte „Guided Deformation“ zu einem finalen dynamischen Modell zusammengeführt.



R. Pagés, K. Amplianitis, D. Monaghan, J. Ondřej, A. Smolić,
Affordable Content Creation for Free-Viewpoint Video and VR/AR Applications,
J. Vis. Commun. Image R. (2018), doi: https://doi.org/10.1016/j.jvcir.2018.03.012

20.2. AI-Based 3D Shape Reconstruction

Mit AI können sehr gute Ergebnisse in der 3D Reconstruction erzeugt werden.

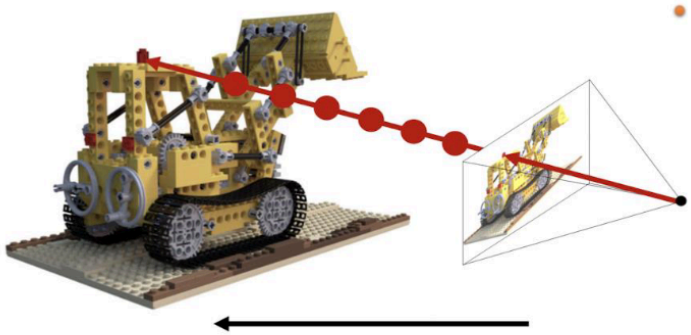
- **Single-Image Reconstruction:** Oft reicht bereits **ein einziges 2D-Bild** für die Erstellung eines vollständigen 3D-Modells.
- **Neuronale Netzwerke (Deep Learning):** Sie nutzen antrainiertes Vorwissen aus grossen Datensätzen, um verdeckte Strukturen (wie die Rückseite eines Objekts) logisch zu ergänzen.

20.2.1. 3D Rendering vs. 3D Rekonstruktion

Um diese Konzepte zu verstehen, hilft es, die Richtung der Berechnung zu betrachten:

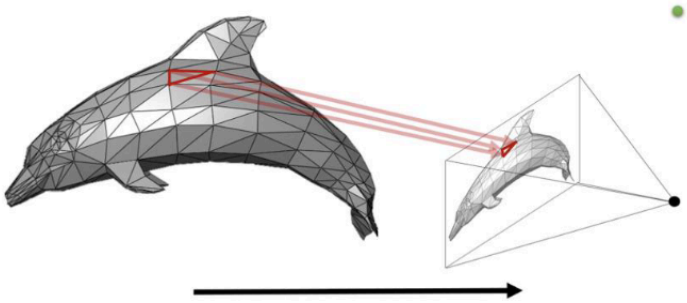
3D Rendering (Forward Mapping):

- **Ausgangslage:** Die 3D-Szene (Geometrie, Punkte, Licht) ist **bekannt**.
- **Ziel:** Ein 2D-Bild aus dieser Szene berechnen (z. B. durch Raytracing).
- **Richtung:** 3D → 2D



3D Rekonstruktion (Inverse Rendering):

- **Ausgangslage:** Nur 2D-Bilder (Fotos/Videos) der realen Welt sind **bekannt**.
- **Ziel:** Die zugrunde liegende 3D-Szene aus den Pixeln „zurückrechnen“.
- **Richtung:** 2D → 3D



20.2.2. 3D Represenations

Es gibt verschiedene Möglichkeiten zur Repräsentation von einer 3D Struktur:

- polygon mesh (vertices + texturen)
- voxel (3d arrays)
- point cloud
- Funktionen (z. B. Funktion für eine Sphere, welche als Input einen Punkt in 3D entgegen nimmt)

20.2.3. NeRF Neural Radiance Fields

Ein neuronales Netz wird verwendet, um eine gesamte 3D-Szene zu speichern. Die Szene wird durch ein vollständig verbundenes, tiefes Netzwerk repräsentiert:

- **Input:** 5D-Koordinaten, bestehend aus der räumlichen Position (x, y, z) und dem Blickwinkel (Θ, φ).
- **Output:** Die Volumendichte (σ) sowie die vom Blickwinkel abhängige Lichtstrahlung (Farbe: r, g, b) an der jeweiligen räumlichen Position.

Rendering model for ray r(t) = o + td:

$$C \approx \sum_{i=1}^{N} T_i \alpha_i c_i$$

colors

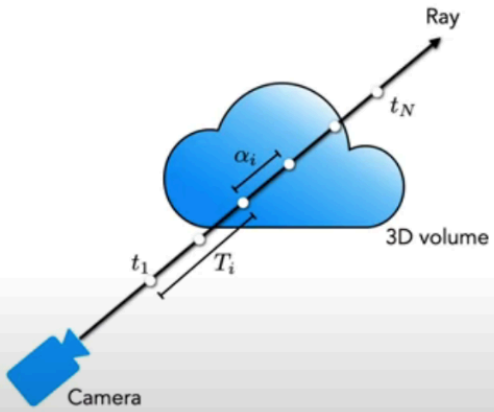
weights

How much light is blocked earlier along ray:

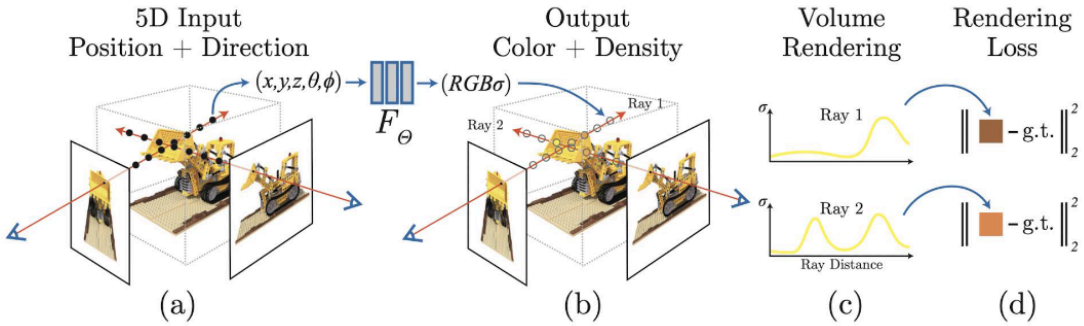
$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

How much light is contributed by ray segment *i*:

$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$



Training: Das Neuronale Netz wird trainiert mit Inverse Rendering.



WICHTIG: NeRF Rendering und NeRF Training ist sehr komplex und braucht viel Compute.

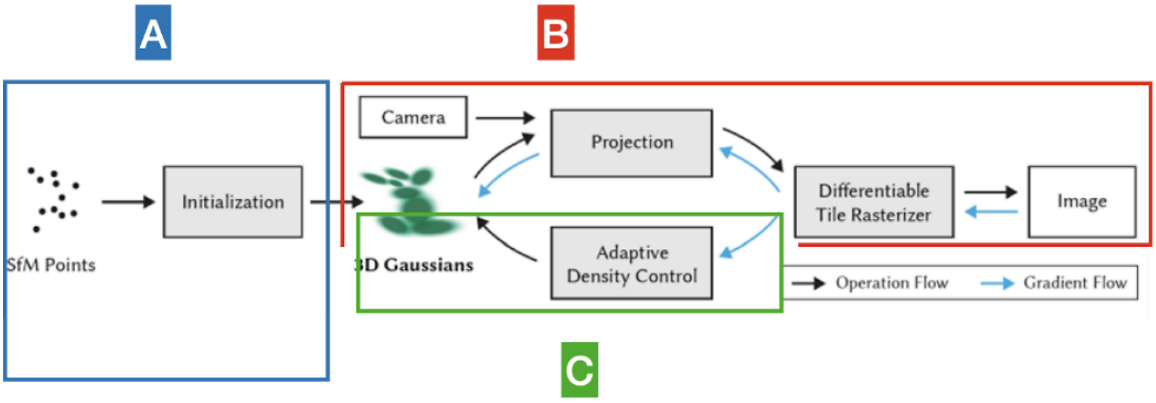
20.2.4. Gaussian Splats

Gaussian Splats sind eine Alternative zu NeRF und ermöglichen **Real-Time Rendering**.

Eigenschaften und Vorteile:

- Weniger rechenintensiv im Training und Rendering als NeRF.
- Basiert initial auf einer Punktwolke aus SfM (SfM Points).
- Nutzt eine Ansammlung von 3D-Gauss-Ellipsoiden anstelle eines neuronalen Netzwerks als Repräsentation.

Ablauf von Gaussian Splatting:



20.2.5. AI Based Content Creation Workflow

- Neue AI basierte Algorithmen ermöglichen qualitativ hochwertige 3D Artefakte von einfachen Aufnahmen.
- **Demokratisierung der Erfassung:** Statt riesiger Studios mit Dutzenden Kameras und Greenscreens genügen heute oft Smartphones und Apps (wie z. B. „Vologrames“), um volumetrische Videos aufzunehmen.
- Möglichkeit zur 3D Digitalisierung der realen Welt.
- Integration in professionelle Production Workflows ist noch nicht ganz abgeschlossen.

Ablauf von AI Based Content Creation:

1. **Capture**: Aufnahme eines Bildes oder Videos vom Objekt
2. **3D Rekonstruktion**: Verarbeitung durch ein Tool
3. **Bearbeiten**: Clean Up und Enhancing
4. **Integration**: Artefakt wird in die Ziel Applikation aufgenommen (z. B. Unity)
5. **Experience**: Publikum kann das finale Produkt nutzen

20.3. Zusammenfassung: Methoden-Auswahl

Methode	Einsatzbereich	Besonderheit
SfM	Statische Objekte	Klassisch, hohe Detailgenauigkeit möglich.
SfS (Visual Hull)	Volumetric Video (Dynamisch)	Grobe Struktur, braucht Multi-Kamera-Setup.
Hybrid	Dynamische Menschen	Nutzt anatomisches Vorwissen (Skelett).
NeRF	KI-Rekonstruktion	Fotorealistisch, aber sehr rechenintensiv.
Gaussian Splats	KI-Rekonstruktion	Echtzeit-Rendering, sehr effizient.