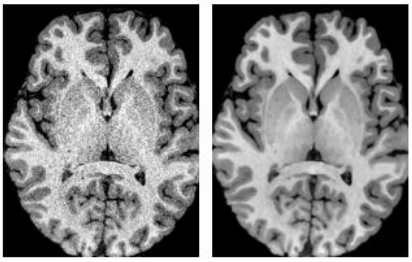
Overview

1. Introduction

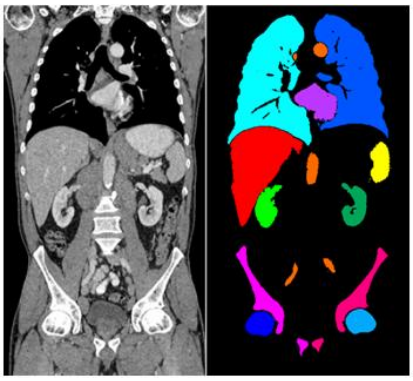
1.1 Main tasks in medical image analysis

1.1.1 Enhancement



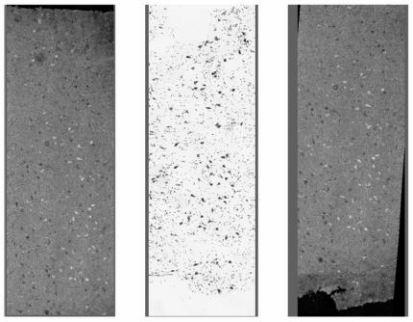
- At each pixel / voxel assign a new value or a set of values
- The new values can represent
  - Noise-free intensities
  - Higher-resolution information
  - Artifact-free intensities
  - Intensities of different modalities
- Information at each pixel
  - Intensity at and around the pixel
  - Intensity at different modalities

1.1.2 Segmentation



- At each pixel / voxel assign a label
- Labels can be
  - Organ: liver, spine, etc.
  - Part of organs: individual vertebrae, etc.
  - Lesions: tumors, pathologies, etc.
- Information at each pixel
  - Intensity at and around the pixel
  - Intensity at different modalities

1.1.3 Registration



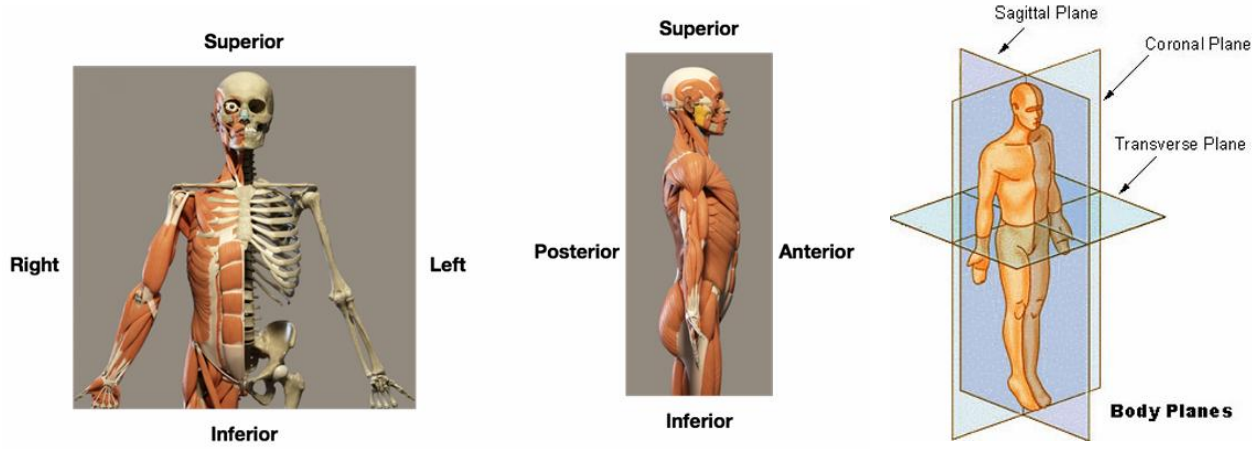
- Determine spatial transformation between two images
- Creates a correspondence between points
- The transformation can be linear or non-linear
- Often the problem might be ill-posed and regularization is used

1.1.4 Classification / regression

- Mapping from image to label
- Labels can be
  - Diagnosis: disease / healthy
  - Outcome prediction: success / complication
  - Presence of an object
- Segmentation / registration can be thought as classification / regression tasks with special structure, i.e., per pixel / voxel prediction

1.2 Image formats

1.2.1 Anatomical axes



- **Axial plane**: Divides the body into **superior (top)** and **inferior (bottom)** sections.
- **Coronal plane**: Divides the body into **anterior (front)** and **posterior (back)** sections.
- **Sagittal plane**: Divides the body into **left** and **right** sections.

1.2.2 Basic image properties

Image size	**Pixel / voxel size**	**Field-of-view (FOV)**
- Number of pixels / voxels - 2 or 3 integers - 256 * 256 * 128	- The size a pixel / voxel occupies in the real world - 2 or 3 real numbers - 0.6 * 0.6 * 3.5 mm^3	- The size an entire image occupies in the real world - 2 or 3 real numbers - **Product of image size and pixel / voxel size** - 153.6 * 153.6 * 448 mm^3

2. Enhancement and pre-processing

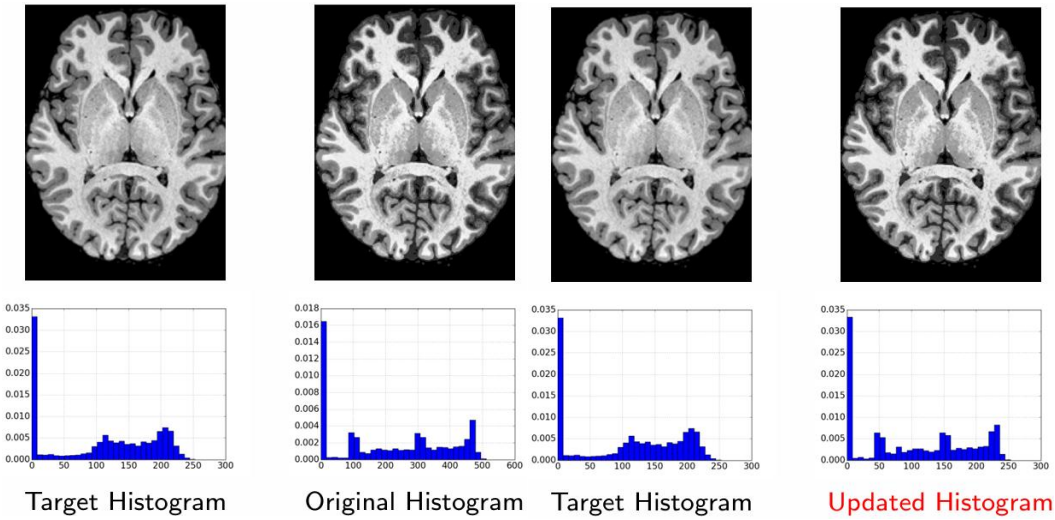
2.1 Intensity normalization

2.1.1 Problem source

- Intensity variations between images
- Differences in scanners, protocols, acquisition software, reconstruction method, etc.
- Important for MRI due to lack of absolute intensities, problematic for machine learning methods, i.e., the domain shift
- Less of a problem for modalities with absolute intensities

2.1.2 Min-max (linear)

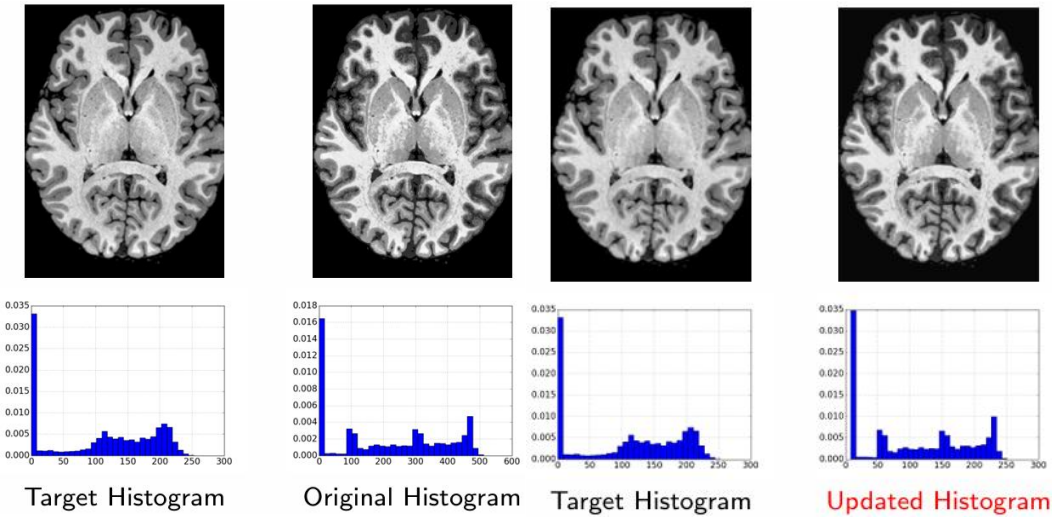
- Matching the range
- Min-max normalization: $f(j) = \frac{j - j_{min}}{j_{max} - j_{min}}(i_{max} - i_{min}) + i_{min}$, where i and j are the intensities of the two images
- Max and min values are sensitive to outliers
- Usually intensities corresponding to ϵ% and $1 - \epsilon$% of the CDF are used instead, e.g., $\epsilon = 2$
- Commonly used in machine learning based methods as a preprocessing step
- Can solve global intensity shifts and multiplicative factors
- **Cannot** solve contrast difference



2.1.3 Mean and standard deviation (linear)

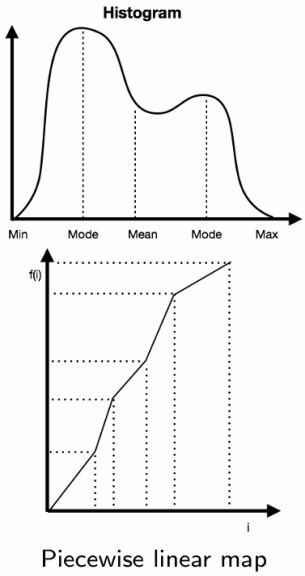
- Matching second order statistics: mean and standard deviation normalization
- $f(j) = (j - \mu_j)\frac{\sigma_i}{\sigma_j} + \mu_i$
  - $\mu_i = \frac{1}{N}\sum_{x=1}^{N} i(x)$
  - $\sigma_i^2 = \frac{1}{N-1}\sum_{x=1}^{N}(i(x) - \mu_i)^2$

- Less sensitive to outliers but still robust statistics may be used for better behavior, e.g., median, robust variance
- Often used in machine learning based methods
- Can solve global intensity shifts and multiplicative factors
- **Cannot** solve contrast difference



2.1.4 Histogram normalization with landmarks (Nyul's method - non-linear)

- Matching landmarks of image histograms
- Piece-wise linear map: non-linear mapping
- Different landmarks are possible: min and max, mean or median, quartiles, deciles, modes, etc.
- Less sensitive to outliers
- Widely used
- Population-wide use:
  - Estimate landmark positions from population data as averages
  - Map the new image's landmarks to the average
- The **contrast is changed** due to the non-linearity of the mapping.



Note: histogram matching **does not know about image content**; all pixels are considered **independent**; no spatial information / correlation between neighboring pixels; matching gradients has been proposed; difference in object size can be a problem; perfect histogram match would not be preferred in that case; use mode matching if object size is different.

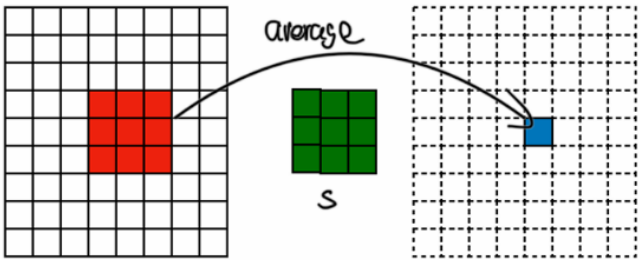
2.2 Noise suppression

2.2.1 Problem source

- Random noise and imaging artefacts due to
  - Imperfections in the acquisition device
  - Fast acquisition
  - Implants
- Doing the imaging slower gives better resolution but also has more patient movement ergo noise (more radiation possible)
- **Goal: given the noisy image *J*, we want to remove noise to get *I***
- Type of noises:
  - Additive noise
    - $J(x) = I(x) + \epsilon(x)$
    - $J(x)$ is observed, $I(x)$ is desired, and $\epsilon(x)$ is the noise, which is often considered to be independent and identically distributed (iid)
    - *Example*: Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma)$
  - Multiplicative noise
    - $J(x) = I(x)\epsilon(x)$
    - *Example*: Speckle in ultrasound and optical coherence tomography
  - More complicated noise
    - $J(x) = \sqrt{(I_r(x) + \epsilon)^2 + (I_i(x) + \eta)^2}$
    - *Example*: Noise in MR magnitude image $\epsilon, \eta \sim \mathcal{N}(0, \sigma)$ the Rician noise

2.2.2 Convolutional / linear

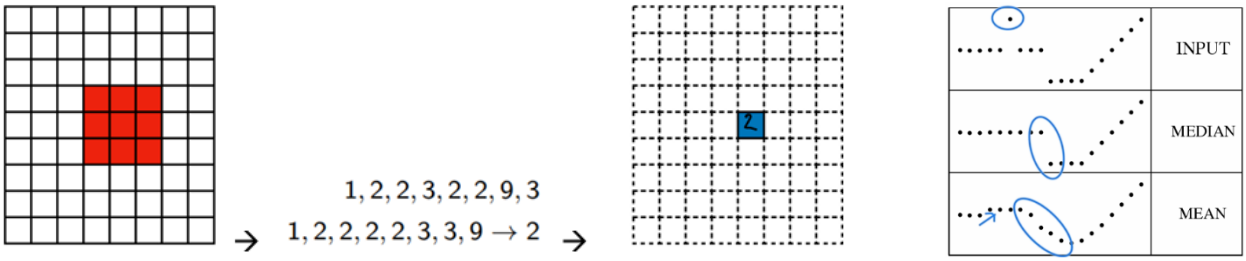
- Convolution
  - $\tilde{I} = J \cdot S$, where S is structural element (kernel) and $\tilde{I}$ is the denoised image
  - Continuous: $\tilde{I}(x, y, z) = \iiint J(x - r, y - p, z - q) S(r, p, q)\, drdpdq$
  - Discrete: $\tilde{I}(x, y, z) = \sum_r \sum_p \sum_q J(i - r, j - p, k - q) S(r, p, q)$
- Advantages
  - Efficient to apply
  - Easy to implement
- Disadvantages
  - Blurs edges
  - Not context aware
  - Sensitive to outliers



Note: Image stays the same size. It is **not possible to remove noise and not blur edges** at the same time with a **linear filter**, using non-linear filtering instead.

2.2.3 Median filtering

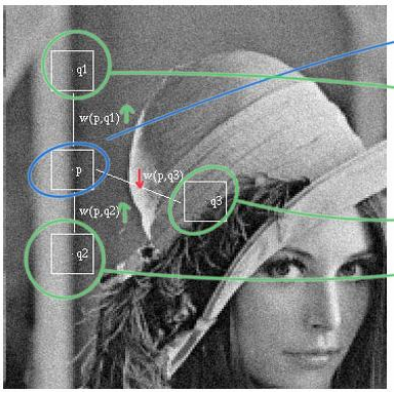
- For each location, rank order all neighboring intensities, assign the median value as the result
- Sliding window similar to convolution



- **Non-linear filter**
- Other rank order filters, such as min and max, are defined similarly
  - *Note*: Two rank filters are not commutative, i.e., min(median(x)) ≠ median(min(x))
- Advantages
  - Robustness to outliers
  - Preserves discontinuities, such as edges
  - Easy and efficient implementation (can separate rows and column filter)
  - Does not introduce new intensities
    - Keep the intensities integer
    - Applying median filtering to labels, in a post-processing step, will not introduce new labels and remove islands, e.g., segmentation
- Disadvantages
  - Patchy result
  - Image content can change, adding and removing structure

2.2.4 Non-local means

- Best approach if you do not use training
- Extension of bilateral filtering
- Most noise suppression methods can be put in the form $I(x) = \sum_{y\in\Omega} w(x,y)J(y)$ with $0 \le w(x,y) \le 1$ and $\sum_{y\in\Omega} w(x,y) = 1$ for all x
  - Weights $w(x,y)$ changes accordingly
- *Idea*: Smoothing point x use areas with similar image content, **if noise is random then averaging similar areas should get rid of it**, which uses redundancy in the images, i.e., local structures repeat.
- Non-local means uses the weight $w(x,y)$ to access similarity between image information at x and y
  - $w(x,y) = \frac{1}{Z(x)} \exp\left(-\frac{\|J(\mathcal{N}(x)) - J(\mathcal{N}(y))\|_2^2}{h^2}\right)$
    - $Z(x)$ is the normalization constant
    - h is the degree of filtering
    - $N(x)$ is the neighborhood around x
  - $I(x) = \sum_{y\in\Omega} w(x,y)J(y)$
    - h depends on the noise level (increasing h leads to **more smoothing**, can be **bad**)



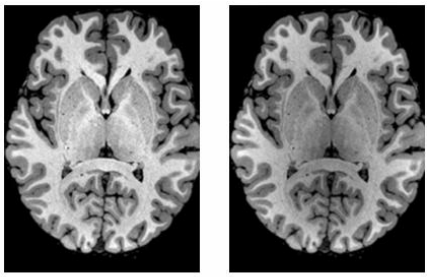
        - Size of $\mathcal{N}(x)$ is the patch size
            - $\mathcal{N}(x) = \Omega$ leads to mean filtering
            - $\mathcal{N}(x) = \{x\}$ leads to Yaroslavsky's filter, a special case of bilateral filtering
        - Size of $W(x)$ is the search size
- Advantages
    - Higher noise suppression
    - Better preservation of edges
    - Currently the **state-of-art** filtering technique that is **not based on machine learning**
- Disadvantages
    - May create artifacts

2.2.5 Optimization-based (energy-based)

- Suppose we have the additive model for the noisy image $J(x) = I(x) + \epsilon(x)$
- We have denoising as an energy minimization problem with $\hat{I} = \arg_I \min \lambda D(I,J) + R(I)$
    - $D(I,J)$ is data consistency, observing J and I is not too far away from J
    - $R(I)$ is regularization, embedding **prior information** (options on image should look like) on the **desired image**
    - λ is **weighting coefficient**
- Data consistency $D(I,J)$ is the integral over the differences between J and I
    - $D(I,J) = \int \big(I(x) - J(x)\big)^2 dx = ||I - J||_2^2$
- Regularization $R(I)$ takes the norm of the image gradient and integrates it so it wants an image with little changes, i.e., piecewise linear image
    - $R(I) = \int ||\nabla I||\, dx = \big||I|\big|_{TV}$
- Weight λ **allows noise level in the observed image**, connected to the standard deviation
    - High λ: consistency is important, **more noise**
    - Low λ: **less noise**, strong boundaries but adds changes to the picture
- Probabilistic interpretation: The energy-based formulation can be interpreted in a **probabilistic model** (MAP)
    - $\arg_I \min \lambda D(I,J) + R(I) = \arg_I \max \log P(J|I) + \log P(I)$
        - $P(J|I)$ is the likelihood, e.g., $P(J|I) = N\left(J; I, \frac{1}{2\lambda}\right)$
        - $P(I)$ is the prior, e.g., $P(I) \propto \exp(-||\nabla I||)$
- Total variation denoising model: Likelihood has Gaussian distribution
    - The standard deviation of the likelihood is linked to the weight in the energy-based formulation
- Generality: The energy-based or the corresponding probabilistic formulations are very generic and many different models can be cast in the same formulation

- $D(I,J)$ or $P(J|I)$ **encode information how the noise process work**. It does not have to additive Gaussian noise but it can be different things as we have seen before.
- $R(I)$ or $P(I)$ **encode our prior information** on what type of images are plausible. TV norm, Wavelet, Dictionary learning, etc. can be used.

2.3 Bias correction



Normal image | Image with bias

2.3.1 Problem source

- Acquisition artifact in MRI
  - Field inhomogeneity
  - Electrical characteristics of the tissue
  - Poor uniformity of coils
  - Gradient and eddy currents
- Minimal effect on visual interpretation by human, difficult to see by eyes
- Adverse effects on algorithms, especially segmentation
- Non-linear effect on intensities that vary spatially
- Remove it as a pre-processing step
- Characteristics of a bias field
  - Smoothly varying
  - Does not depend on the underlying tissue to the first order approximation
- Approximate math model (**multiplicative factor**): $j = bi + n$
  - i is the real image
  - b is the bias field
  - n is the noise
  - j is the observed image

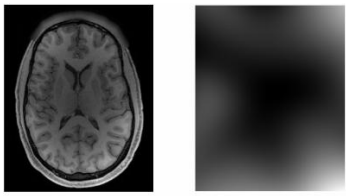


Image | Bias field

2.3.2 Basic methods to correct bias field

- At the hardware side with extra measurements (make a second empty image per image to find it and subtract it then)
  - Good: Ideally a perfectly solution
  - Bad: Additional measurement makes it hard for retrospective data analysis and usually not integrated
  - Post-acquisition methods are very useful
- Manually placed landmarks: Voxels expected to have similar intensity
  - Good: Human in the loop
  - Bad: Need for human interaction
- Joint segmentation: Same structure should have the same intensity in all voxels. Works if you know what you expect in the image
  - Good: Geared to get the best segmentation accuracy
  - Bad: Need for prior information and too specific
- N3 algorithm: Generic (takes away the bad points from above), most popular

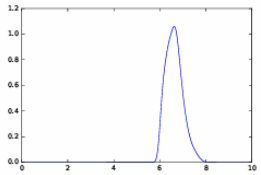
2.3.3 N3 algorithm

- *Goal*: N3 estimates $b(x)$ at every point in the image, and biased removed image can be immediately computed by $i(x) = j(x) - b(x)$
- *Key point*: N3 analyzes the **distribution** of j, b, and i to understand the e**ffect of the bias term**, i.e., how b affects the distribution of i and j
- *Assumptions*:
  - **Independence between pixels** when considering distribution of j, b, and i
  - **Bias field** and the **image intensities** are **independent**
- *Principle*:
  - Start from the main model $\tilde{j}(x) = \tilde{b}(x)\tilde{\imath}(x) + \tilde{n}(x)$
  - Focus on the noise-free case, i.e., $n(x) = 0$, taking the logarithms of both sides to make effect of the bias field additive $j(x) = b(x) + i(x)$
  - Let a and b be two independent random variables with pdfs denoted by $f_a(a)$ and $f_b(b)$; let's define $c = a + b$, we will compute the distribution of c, i.e., $f_c(c)$ using cdf of c, i.e., $F_c(c')$
    - $P(c \leq c') = F_c(c') = \int_{-\infty}^{c'} f_c(c)\, dc = \int_{-\infty}^{\infty} \int_{\infty}^{c'-b} f_a(a) f_b(b)\, dadb$
  - Compute the PDF by taking the derivative $dF_c(c')/dc'$ using Leibnitz' rule twice
    - $f_c(c') = \frac{d}{dc'} F_c(c') = \frac{d}{dc'} \int_{-\infty}^{\infty} \int_{\infty}^{c'-b} f_a(a) f_b(b)\, dadb =$ $\int_{-\infty}^{\infty} \frac{d}{dc'} \int_{\infty}^{c'-b} f_a(a) f_b(b)\, dadb = \int_{-\infty}^{\infty} f_a(c'-b) f_b(b)\, db$
  - Denote the probability distribution of j, b, and i with $J(j)$, $B(b)$, and $I(i)$, the probability distribution J is the convolution
    - $J(j) = \int I(j-b)B(b)db = I \cdot B$
    - What can we say about B?
      - $\tilde{j}(x) = \tilde{\imath}(x)\tilde{b}(x)$
      - $\tilde{b}$ is around 1 so b is around 0
      - B can be approximated with a **Gaussian** with **small standard deviation**
      - J is a smoothed version of I
      - The bias field is a **smooth field** – b is smooth
      - The estimation must determine $I(i)$ and b given only j
        - Part 1: field estimation – estimation of b given $I(i)$
        - Part 2: $I(i)$ estimation – estimation of $I(i)$
  - *Part 1*: field estimation
    - Given an intensity value j, the probability distributions $I(i)$ and $B(b)$, predict the i that may correspond j on average. Then get the bias field as the difference, applying this at every pixel / voxel

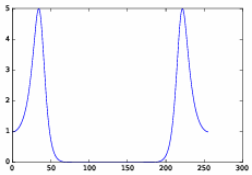
$$
\begin{aligned}
\mathbb{E}[i|j] &= \int_{-\infty}^{\infty} ip(i|j)di \\
\text{Bias field estimate: } b_e(j) &= \mathbb{E}[b|j] = j - \mathbb{E}[i|j] \\
\mathbb{E}[i|j] &= \int_{-\infty}^{\infty} i\frac{p(i,j)}{p(j)}di \\
&= \frac{1}{J(j)}\int_{-\infty}^{\infty} ip(i,j)di, \;\; b = j - i \\
&= \frac{1}{J(j)}\int_{-\infty}^{\infty} ip(i)p(b)di \\
&= \frac{\int_{-\infty}^{\infty} iB(j-i)I(i)di}{\int_{-\infty}^{\infty} B(j-i)I(i)di}
\end{aligned}
$$

- We know all the terms in this last equation and i is one dimensional, so we can solve it **numerically with ease**

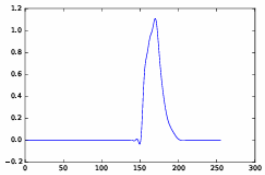
- *Part 2*: Distribution of original image $I(i)$ estimation
  - *Key observation*: $J(j)$ is a smoothed version of $I(i)$, and $B(b)$ is roughly Gaussian with a small standard deviation
  - *Approximation*: Approximate $I(i)$ by sharpening $J(j)$ assuming $B(b)$ is a Gaussian with small standard deviation
    - $J = I \cdot B \Leftrightarrow \mathcal{F}\{J\} = \mathcal{F}\{I\}\mathcal{F}\{B\}$
    - Deblurring filter: $\hat{F} = \frac{\mathcal{F}\{B\}^*}{|\mathcal{F}\{B\}|^2 + Z^2}$, $\mathcal{F}\{I\} \approx \hat{F}\mathcal{F}\{J\}$
    - With Z being a small number and $\mathcal{F}\{B\}^*$ denotes the complex conjugate
    - *Note*: this is inverse filtering
  - Estimate of the unbiased image's distribution: $I(i) \approx \mathcal{F}^{-1}\left(\hat{F}\mathcal{F}\{J\}\right)$



$J(j)$



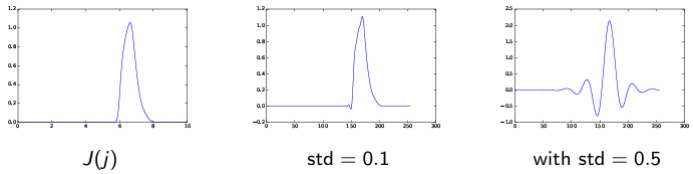
$|\hat{F}|$

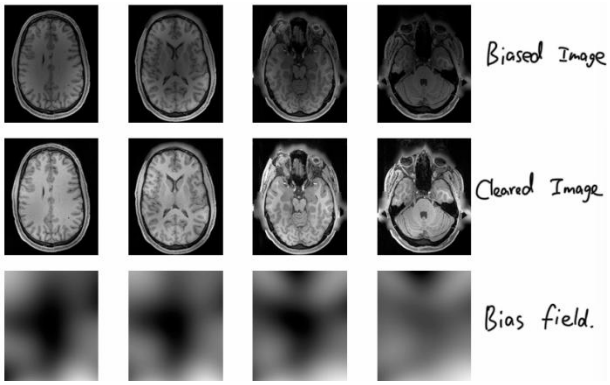


$I \approx \mathcal{F}^{-1}(\hat{F}\mathcal{F}\{J\})$

  - Field estimate: $b_e(j) = j - \frac{\int_{-\infty}^{\infty} iB(j-i)I(i)di}{\int_{-\infty}^{\infty} B(j-i)I(i)di}$
  - Using both we can estimate field at any point x in the image
  - *Need for smoothing* (there are points that lead to noise)
    - $\tilde{j}(x) = \tilde{b}(x)\tilde{\imath}(x) + \tilde{n}(x)$ – we ignored the noise
    - $\hat{F}$ is approximate and so is resulting I
    - But we did not yet use the fact that $b(x)$ should be a smooth field – there is a spatial correlation between the bias field at neighboring pixels
    - Solution: smooth approximation based on anchor points using Splines or Radial Basis Function Approximation, i.e., solve bias field for some points, interpolate for the rest

- *Need for iterations*
  - We showed that $J = I \cdot B$ and used $I \approx \mathcal{F}^{-1}\left(\hat{F}\mathcal{F}\{J\}\right)$
  - In theory we do not know B only assume it is a Gaussian
  - If we use too wide of a B the inverse filtering may fail





- Remember they have to be distributions (>0), deconvolution is hard
- Solution, iterate many times with small Gaussians (small standard deviation)
- Not a bad approximation: large Gaussians are convolutions of smaller ones

2.4 Variations in images and pixel-size

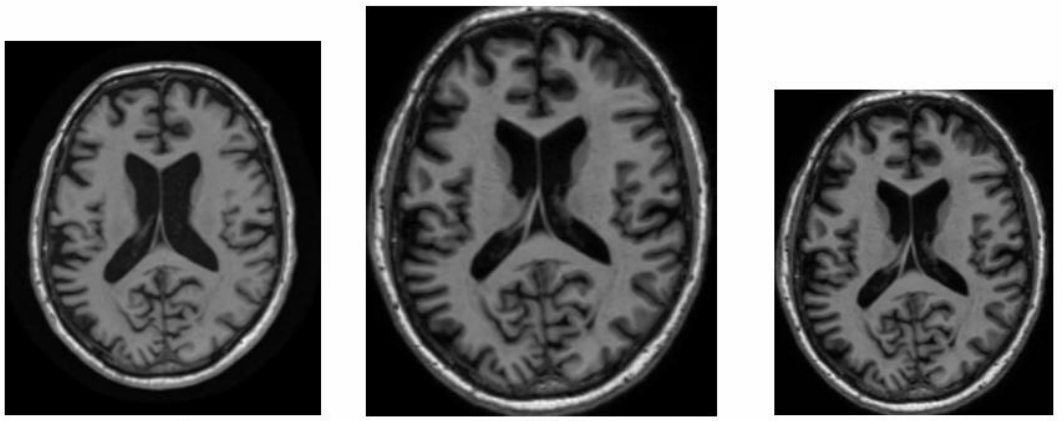
2.4.1 Problem source

- Images may come with different pixel and / or image size, they may have different field-of-view (FOV)
- May need to compare such "heterogeneous" images
- Why does it happen? Differences
  - In acquisitions
  - In FOV
  - In pixel size
- In which applications do it matter?
  - **Comparing measurements**
    - Measurements taken in an image have a correspondence in real life
    - Real life measurements rely on pixel size
    - Comparisons need to take this into account, e.g., longitudinal analysis, population statistics, normative distributions
  - **Registration**
    - Aligning images requires building point correspondence;
    - Building correspondence makes sense if two points show similar area
  - **Machine learning applications**
    - Parameters of a model are learned from a set of images
    - Applying the same model to an image with different image / pixel size or FOV requires attention

- Efficiently it is important for all applications
- Building algorithms that work for heterogeneous images is one of the most challenging tasks

2.4.2 How to tackle it?

- Resample one image to **match the pixel-size** of the other. If volumetric images in 3D, then **match the voxel-size**
- Crop or pad if necessary to make the image sizes the same

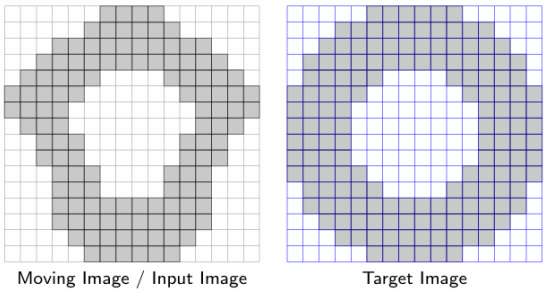


- If we are matching the image in the second column to the one in the first one, the resize ratios in x- and y-directions are $1.0 \div 1.25 = 0.8$ and $1.0 \div 1.25 = 0.8$
- If we are matching the image in the first column to the one in the second one, the resize ratios in x- and y-directions are $1.25 \div 1.0 = 1.25$ and $1.25 \div 1.0 = 1.25$

Note: **Always match the pixel / voxel-size**. Do not match the image size, it may lead to wrong normalization.

3. Spatial Transformations and image registration

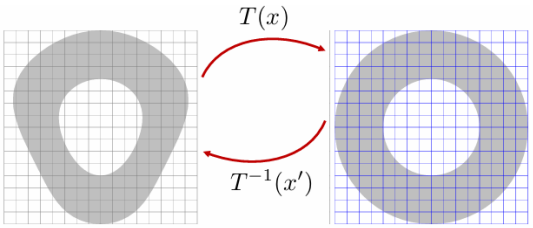
3.1 Sampling model



3.1.1 Setup

- **Moving image** is the one that will be **transformed**
- **Target image** is to target that we **want to attain with the transformation**, both images have their own pixel coordinates and corresponding Cartesian grids
- Assume there exists a spatial transformation, i.e., a coordinate transformation, that matches the moving to the target image
- In reality, images are **discretized** - we only have values at pixel locations

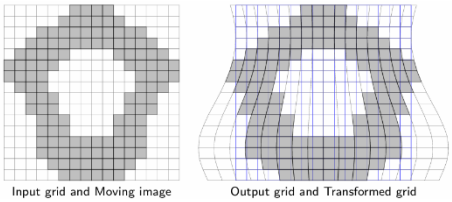
3.1.2 Transformation



- Assume $x' = T(x)$ is the transformation model with $x, x' \in \mathbb{R}^2$ in this case, where x and x' represents points in the moving and target image coordinate systems, respectively (same for 3D transformation)
- Define $x = T^{-1}(x')$ as the inverse of the transformation model
- Number of possible models: $T(x) = Ax$ (linear) or $T(x) = x + u(x)$ (non-linear)

Question: Given the images, respective pixel-grids and transformations, how do we generate the "output" transformed moving image? Forward transformation and backward transformation.

3.1.3 Forward transformation ($x \rightarrow x'$)

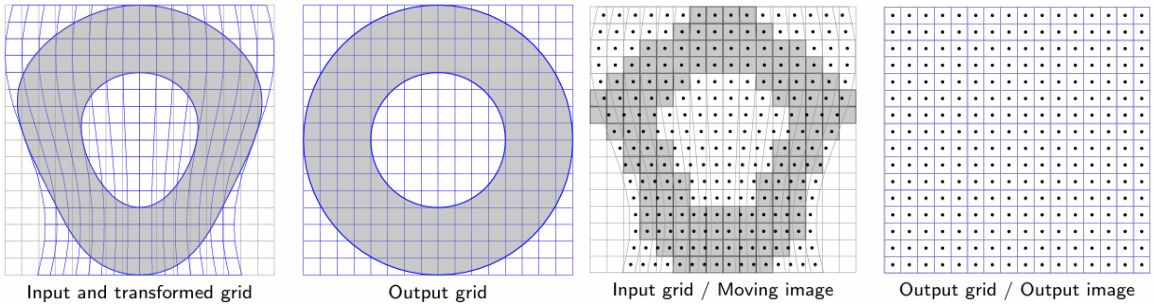


- $T(x) = \left\{\frac{x_1}{[0.15\sin(2.25x_2)+0.85]}, x_2\right\} = \{x_1', x_2'\} = x'$
- We **transform the <u>input</u> grid**, where the blue contour is the object's contour in the target image and the gray area is the transformation of the moving image.
- In the discrete case each pixel from the moving image will be transformed $x' = T(x)$ and the pixel intensity will be copied to the output image
- Problems
  - **Transformed pixel** coordinates x' **does not necessarily fall on pixel** exactly
  - **Two pixels** can be mapped to the **same output pixel**
  - **Same output pixel** can be mapped to **two output pixels**
  - **Some output pixels** may **not** even **get a value** assigned
- Solution (but **complicated**)
  - Intensities of the output can be computed using partial areas
  - Overlap between each transformed pixel and output pixels are computed
  - This overlap is used in a complex intensity assignment step

3.1.4 Backward transformation ($x' \rightarrow x$)

- $T^{-1}(x') = \{x_1'[0.15\sin(2.5x_2') + 0.85], x_2'\} = \{x_1, x_2\} = x$
- We **transform the <u>output</u> grid**, where the blue contour is the object's contour in the target image and the gray area is the transformation of the moving image
- In the backward mapping, for each output pixel, the corresponding location in the input / moving image is determined, which is easier to deal with
- Problems
  - Transformed locations of the output pixels do not generally fall exactly on input pixels
- Solutions
  - **<u>Interpolation</u>**: pixel intensity (all channels if necessary) are interpolated using the nearest input image pixels

<u>Note</u>: **<u>Backward</u>** transformation is **<u>always easy to use</u>** for transforming images **through interpolation**.



3.2 Interpolation models

3.2.1 Nearest neighbor interpolation

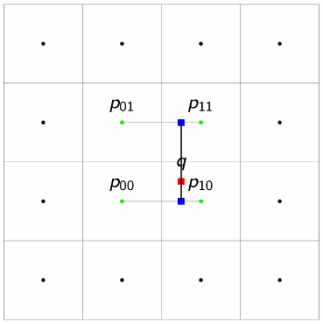
<u>Idea</u>: Assign the intensity value of the **closest pixel** with known intensity - assigning the value of the nearest neighbor

- Nearest neighbor interpolation is **very fast**
- It **does not introduce new intensity** values in the image
- May introduce **heavy artifacts in gray level** images (boundary can be very bad)
- Nearest neighbor interpolation yields a **piece-wise <u>constant</u> function**. This is **<u>not continuous nor differentiable</u>**
- While it is not used for gray level or multispectral images, it is used for **transforming labels** since it has the nice property of not introducing any new values

3.2.2 Bilinear interpolation

<u>Idea</u>: Uses **closest 4 pixels' intensity values** (extension of linear interpolation in 2D or trilinear interpolation)

- Linear interpolation takes the distance; the **closer the dot is to one point, the more weight this pixel value becomes**
- Bilinear interpolation is **fast**



- It will introduce **new intensity values** in the image **due to interpolation** (good for continuous figures, no segmentation)
- Does not introduce as big artifacts as nearest neighbor interpolation
- Bilinear interpolation leads to **piece-wise <u>linear</u> functions**, which is **continuous** but **<u>not differentiable</u>**
- It may smooth and blur the image, leading to **decrease in spatial resolution**
- Trilinear interpolation in 3D repeats the same procedure for an additional dimension and uses 8 points
- Formulation with 4 unknown parameters
  - $I(x_1, x_2) = \frac{x_2^1 - x_2}{x_2^1 - x_2^0} I(x_1, x_2^0) + \frac{x_2 - x_2^0}{x_2^1 - x_2^0} I(x_1, x_2^1)$
  - $I(x_1, x_2) = a + bx_1 + cx_2 + dx_1x_2$
  - Corresponding to the four unknowns we have the intensity values at the four corners leading to a linear system of equation
  - $$\begin{bmatrix} 1 & x_1^0 & x_2^0 & x_1^0 x_2^0 \\ 1 & x_1^0 & x_2^1 & x_1^0 x_2^1 \\ 1 & x_1^1 & x_2^0 & x_1^1 x_2^0 \\ 1 & x_1^1 & x_2^1 & x_1^1 x_2^1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} I(x_1^0, x_2^0) \\ I(x_1^0, x_2^1) \\ I(x_1^1, x_2^0) \\ I(x_1^1, x_2^1) \end{bmatrix}$$
    - We can solve the system of equations and use a, b, c, d coefficients to interpolate at any x in between $p_{00}, p_{01}, p_{10}, p_{11}$.
  - Trilinear interpolation can be defined similarly with 8 unknowns
    - $I(x_1, x_2, x_3) = a + bx_1 + cx_2 + dx_3 + ex_1x_2 + fx_1x_3 + gx_2x_3 + hx_1x_2x_3$

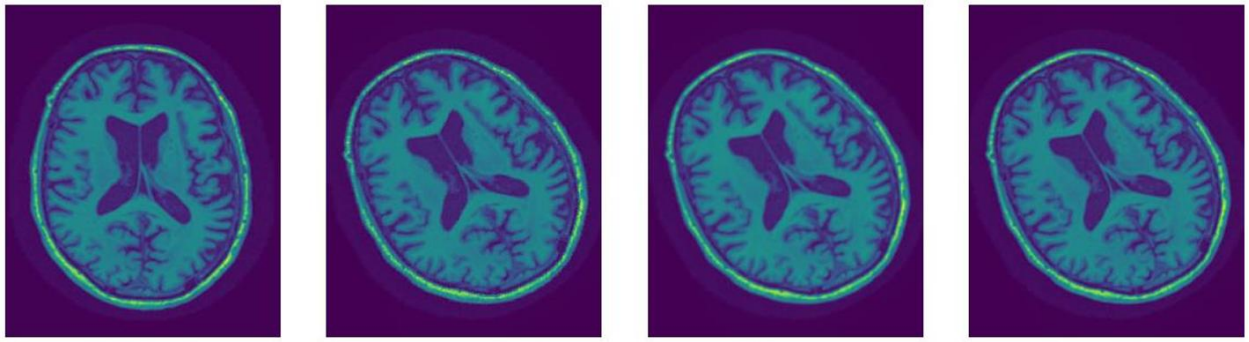
3.2.3 Bicubic interpolation

<u>*Idea*</u>: Beyond bilinear interpolation, we want an even **smoother function**, not just a piece-wise function but one that is **differentiable**

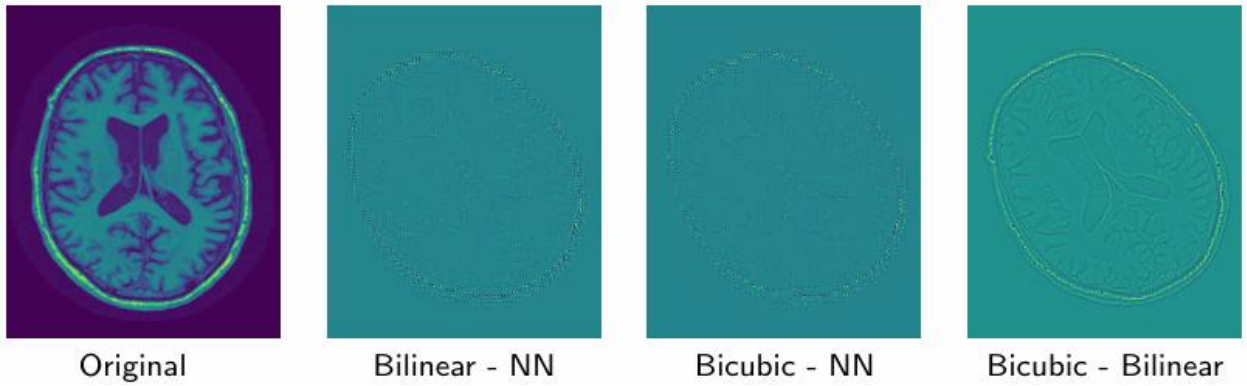
- Uses closest 4 pixels' information, but it requires **not only the intensity values but also the derivatives at those points**
- Therefore, it is common to use **16 closest points** in the interpolation
- Extension of cubic interpolation in 2D, tricubic in 3D
- Similar idea as bilinear interpolation: cubic interpolation in one dimension followed by the other
- Similar to the bilinear interpolation, a linear system of equations can be constructed and solved (16 unknowns). Information to determine these unknowns are taken from the 4 closest neighbors as
  - Intensities at $p_{00}, p_{01}, p_{10}, p_{11}$ (4 values)
  - ∂x_1 and ∂x_2 at $p_{00}, p_{01}, p_{10}, p_{11}$ (8 values)
  - $\partial x_1 x_2$ mixed at $p_{00}, p_{01}, p_{10}, p_{11}$ (4 values)
  - <u>*Remark*</u>: In order to compute the derivatives further closest pixels must be used. Therefore, bicubic interpolation uses more pixels than the closest 4

- *Alternative way*: use a convolution kernel (takes the two points and gives a distance-dependent measure of similarity) the "Catmull-Rom" kernel
- Bicubic interpolation **is better at retaining gray values** than bilinear and nearest neighbor. It has the **highest quality interpolation** amongst them
- Computation requirement is the highest
- Bicubic interpolation leads to smooth functions that are **both continuous and differentiable**
- Tricubic interpolation in 3D repeats the same procedure for an additional dimension and larger number of points
- Bicubic interpolation on label maps: **NOT good** - bicubic interpolation produces extra values. These additional labels arise due to interpolation. They do not exist and should be avoided whenever resampling label maps.

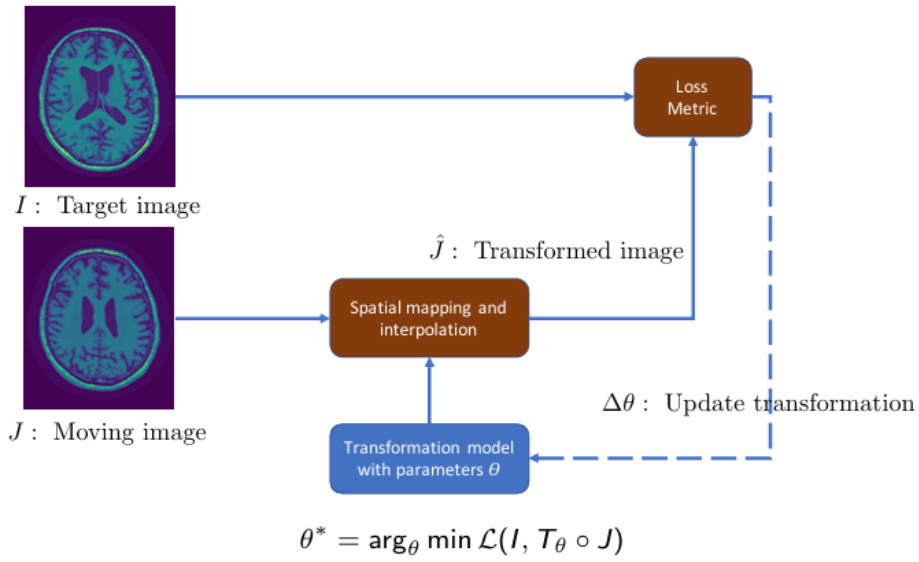
3.2.4 Interpolation method comparison



- Nearest neighbor yields bad quality
- Bilinear is a good trade-off
- During a registration, the interpolation needs to be computed many times during the optimization, where **bilinear's computational advantage is important for this**, one may use bilinear during the optimization but **bicubic at the end**
- For **transforming label maps**, it is better to use **nearest neighbor interpolation**



3.3 Transformation models



- How do we have that $T(x)$
- Transformation function
- What are the parameters?
- **This part is about the blue box**

3.3.1 Linear transformations

3.3.1.1 Linear mapping as transformations

- The general form of the transformation is a matrix-vector product: $x' = T(x) = Ax + t$, where the transformation is defined by the matrix A and the translation (determine the direction and length to move) vector t
- The dimension of this matrix and the translation vector depend on the dimension of the input and output spaces
- Transformations between 2 two-dimensional spaces $x \in \mathbb{R}^2, x' \in \mathbb{R}^2 \rightarrow A \in \mathbb{R}^{2\times2}, t \in \mathbb{R}^2$
- Transformations between 2 three-dimensional spaces $x \in \mathbb{R}^3, x' \in \mathbb{R}^3 \rightarrow A \in \mathbb{R}^{3\times3}, t \in \mathbb{R}^3$
    - These are the most commonly used cases
    - One can also consider transformation of a 3D space to 2D and a 2D space to 3D

Remark 1: This is actually **not linear due to the translation** t, i.e., $T(\alpha x_1 + \beta x_2) \neq \alpha T(x_1) + \beta T(x_2)$, it is linear in the homogeneous coordinates

Remark 2: They are global and called linear transformation models because the **Jacobian of the transformation is the same** for all x

3.3.1.2 Parameterization in the most generic case

- In two-dimensions $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, t = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$, 4 + 2 = 6 free parameters
- In three-dimensions $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$, 9 + 3 = 12 free parameters
- Linear transformations have small number of parameters compared to non-linear transformations

- The same parameters apply to the entire image
- Therefore, the **number of parameters** is **independent** of the **number of pixels / voxels**, $T(x) = Ax + t, \forall x$
- If the matrix A is invertible then the inverse transformation is given as $T^{-1}(x') = A^{-1}x' + t', t' = -A^{-1}t, \forall x'$

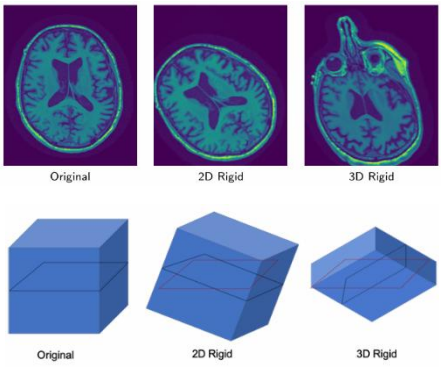
3.3.1.3 Homogenous coordinates

- One can use the homogeneous coordinates, which is $\tilde{T}(x) = \tilde{A}\tilde{x}, \tilde{A} = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix}, \tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$, where 0 is a matrix of appropriate size
- For example, in 2D, transformation matrix with the homogeneous coordinates can be written as $\tilde{A} = \begin{bmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_{22} & t_2 \\ 0 & 0 & 1 \end{bmatrix}$

3.3.1.4 Rigid transformations

- Only **translation** and **rotation**
- One of the most commonly used transformation models in medical image analysis
- In 2D, it is parameterized with one angle θ and one translation vector t,
  - $x' = T(x) = Rx + t$
  - $R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$,

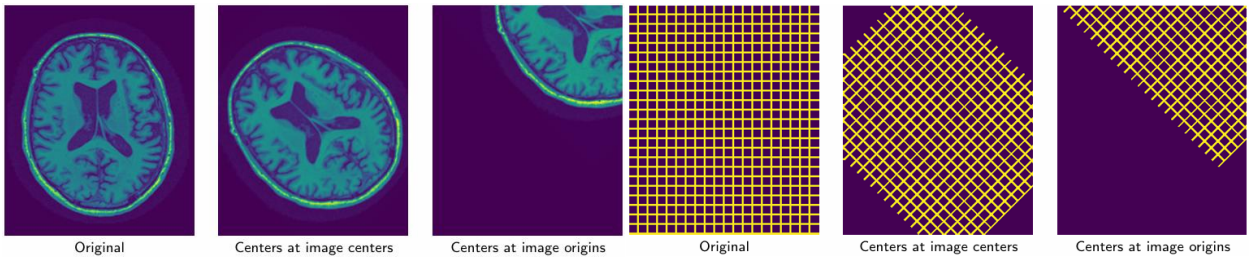
  the transformation rotates the xy-plane about the origin counterclockwise by θ, 3 parameters in total
- In 3D, it can be defined through three rotations and a translation. It is parameterized with three angles $\theta_{xy}, \theta_{xz}, \theta_{yz}$ and a translation vector t, i.e., $R = R(\theta_{xy})R(\theta_{xz})R(\theta_{yz})$, each angle defines rotation about another axis:
  - θ_{xy}: rotation about the z-axis (rotation of the xy plane)
  - θ_{xz}: rotation about the y-axis (rotation of the xz plane)
  - θ_{yz}: rotation about the x-axis (rotation of the yz plane)
    - $R(\theta_{xy}) = \begin{bmatrix} \cos(\theta_{xy}) & -\sin(\theta_{xy}) & 0 \\ \sin(\theta_{xy}) & \cos(\theta_{xy}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$
    - $R(\theta_{xz}) = \begin{bmatrix} \cos(\theta_{xz}) & 0 & \sin(\theta_{xz}) \\ 0 & 1 & 0 \\ -\sin(\theta_{xz}) & 0 & \cos(\theta_{xz}) \end{bmatrix}$
    - $R(\theta_{yz}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_{yz}) & -\sin(\theta_{yz}) \\ 0 & \sin(\theta_{yz}) & \cos(\theta_{yz}) \end{bmatrix}$



Note: **order of the rotations matters**

- The transformation can push the object out of the FOV
- 3D rigid transformation can change the visible anatomy in the same slice. Essentially a different cross-section occupies the same slice after transformation

- Interpolation is performed using bilinear and trilinear methods
- *Center of rotation*: A tricky piece of information that needs to be defined is the centers of coordinate systems in both domains



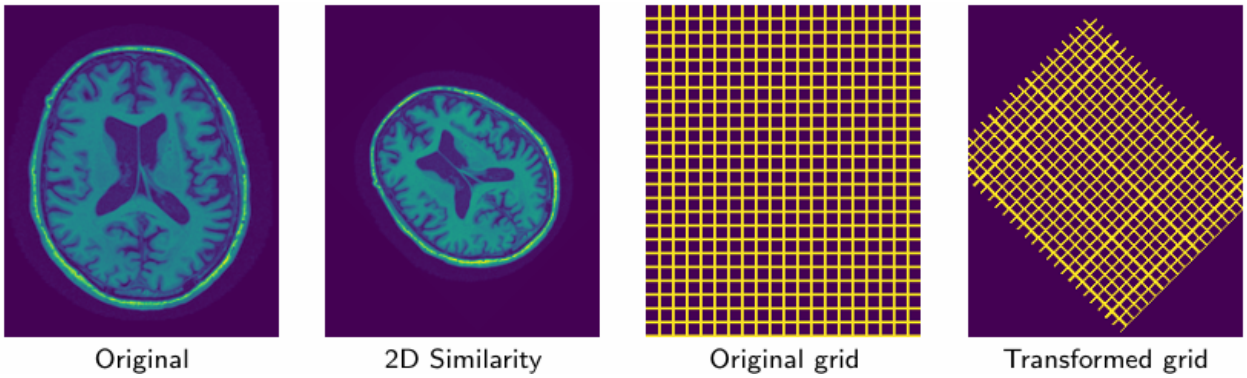
Note: Center of transformation applies to all transformations and is especially important for linear transformations where the same matrix applies to the entire image.

3.3.1.5 Similarity transformations

- Shape is preserved, **rotation** and **translation** and **scaling**
- It is also used quite extensively in medical image analysis
- It is parameterized with a random matrix, a translation vector t, and a scaling factor
  - $x' = T(x) = RSx + t$
  - $S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$ or $S = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix}$

  Four parameters in 2D and seven parameters in 3D (adding one parameter to rigid body motion)

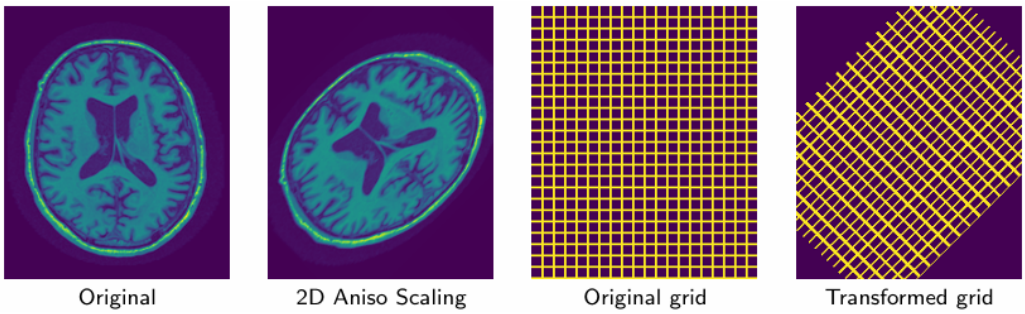
Note: $x' = SRx$ is the **same** as $x' = RSx$



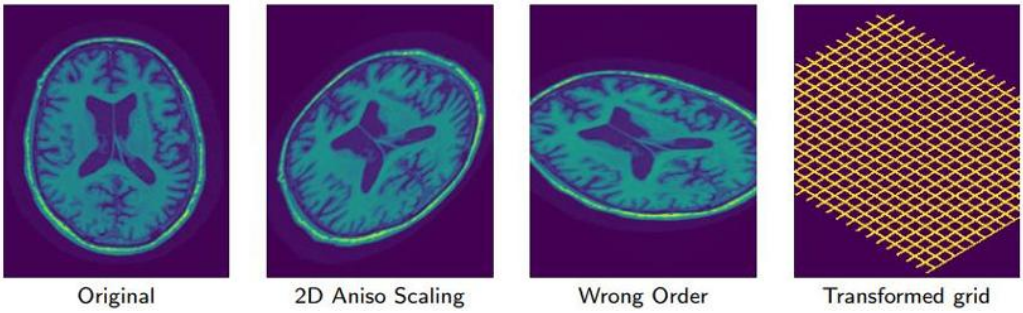
3.3.1.6 Affine transformations

- One can also think about scaling with different factors in different dimensions, angles and shapes don't need to be preserved
- In this case, the transformation is defined through a rotation matrix, translation and a **scaling matrix**
  - $x' = T(x) = RSx + t$
  - $S = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix}$ or $S = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix}$

- This anisotropic form is no longer a similarity transformation, i.e., shapes are not preserved, order matters, meaning that $x' = SRx$ is not the same as $x' = RSx$
    - Number of parameters increases
        - 2D: 1 (rotation) + 2 (translation) + 2 (scaling) = 5 parameters
        - 3D: 3 (rotation) + 3 (translation) + 3 (scaling) = 9 parameters



- Applying the transformation with $T(x) = SRx + t$ yields shearing



- Affine transformation extends the mapping with anisotropic scaling with shearing, the transformation is defined through the equation with the additional **shearing matrix** W
    - $x' = T(x) = WRSx + t$
    - $W = \begin{bmatrix} 1 & w \\ 0 & 1 \end{bmatrix}$ or $W = \begin{bmatrix} 1 & w_1 & w_2 \\ 0 & 1 & w_3 \\ 0 & 0 & 1 \end{bmatrix}$
    - Number of parameters increases
        - 2D: 1 (rotation) + 2 (translation) + 2 (scaling) + 1 (shearing) = 6 parameters
        - 3D: 3 (rotation) + 3 (translation) + 3 (scaling) + 3 (shearing) = 12 parameters
- A **full affine transf**ormation would also have **reflection**, which can be achieved with **negative scaling factor**
- Affine transformation with shearing is **rarely used** in medical image analysis as shearing model is not used often

3.3.2 Non-linear transformations

The general form of the transformation is $x = T^{-1}(x') = x' + u(x')$, where the transformation is defined by the function $u(x')$, note that

- This function can be **non-linear**

- We directly model the **inverse transformation from output (target)** domain **to the input (moving)** domain
- This is due to the difficulty in establishing the inverse transformation for a given arbitrary forward transformation
- Remember that it is more beneficial from the sampling point of view to work with the inverse transformation
- The question is "How do we parameterize $u(x')$?"
- In the naïve parameterization, one defines a displacement vector for each pixel / voxel independent from each other
    - Every pixel could move in a different amount
        - Number of parameters in 2D: (number of pixels) * 2, two components for each displacement vector
        - Number of parameters in 3D: (number of pixels) * 3, three components for each displacement vector
    - This transformation model can generate **very noisy transformations**. Such transformations may not be realistic and even change the topology of the underlying image
- During the optimization, we would like to only search through smooth transformations, which can be achieved by the parameterization
- We need models that can generate **smooth displacement fields** for any set of parameters, effectively reducing the number of free parameters
- There are two strategies
    - Interpolation based
    - Physical model based

3.3.2.1 Kernel-based interpolation models

Idea: Define displacement vectors for sparse set of "control" points and interpolate in between with a smooth model.

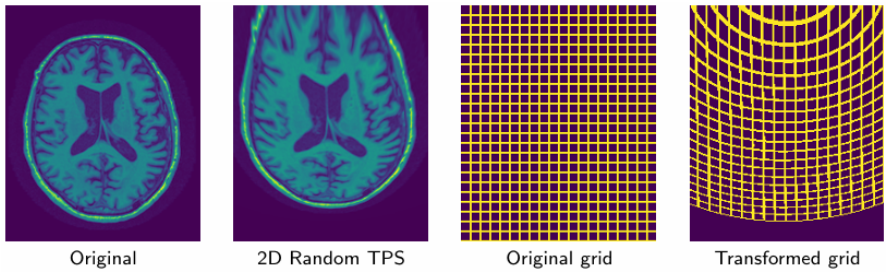
- Radial basis function
    - $u(x') = \sum_{n=1}^{N} \phi(|x' - x'_n|)d_n$
    - $\phi: \mathbb{R}^+ \rightarrow \mathbb{R}$
    - $d_n \in \mathbb{R}^2$ or $d_n \in \mathbb{R}^3$

  where x_n are the control points, ϕ are the basis functions and d_n are the non-linear coefficients, defining the transformation
- Two commonly used forms
    - Thin Plate Splines (TPS)
    - Free Form Deformations (FFD) with B-spline

Thin plate splines (TPS)

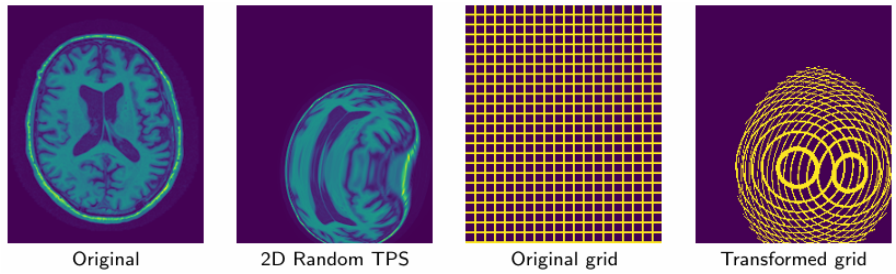
- Uses the radial basis function models, where the basis functions are defined as $\phi(|x' - x'_n|) = \phi(r) = r^2 \log r$

- Sometimes defined together with an affine transformation as follows, where A and t define a global linear transformation and TPS defines local refinement, $x = Ax' + t + \sum_{n=1}^{N} \phi(|x' - x'_n|)d_n$
- **Control points** do not have to be uniformly spaced, they can be randomly dispersed
- In landmark-based registration, where TPS is extensively used, control points are given by the user, points which are easily recognizable in both images (**<u>global support</u>**)
- **Mostly used for landmark-based registration** not intensity-based registration
- The parameters of the non-linear transformation model are the coefficients $\{d_n\}_{n=1}^{N}$
    - In 2D: (number of control points) * 2
    - In 3D: (number of control points) * 3
    - If affine transformation is included, then there are the parameters of the linear transformation $Ax' + t$
- TPS is a **non-linear deformable** transformation model with **very few number of parameters**
- <u>*Example*</u>: TPS with 50 control points (100 parameters)



Smooth transformation that can also apply non-linear deformations to the object in the image

- <u>*Example*</u>: TPS with 25 control points (50 parameters)

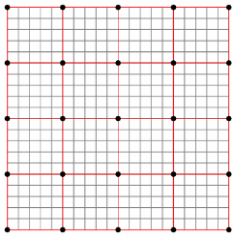


- Transformations are bounded to be **smooth due to the interpolating kernel ϕ**
- Effects of each control point and the parameters d_n defined on it are global
- **Increasing the control points** can lead to **more complicated transformations, more flexible**
- Mostly used for **landmark-based registrations**, not intensity-based registration
- In landmark-based registration, correspondence between a number of points are given, and model parameters are found to satisfy those correspondences

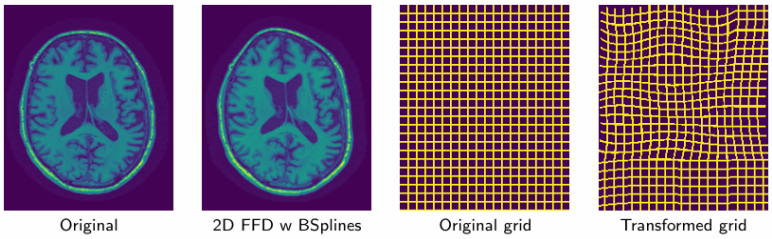
Free form deformation with B-splines

- Starts with the same radial basis function, the kernel is different

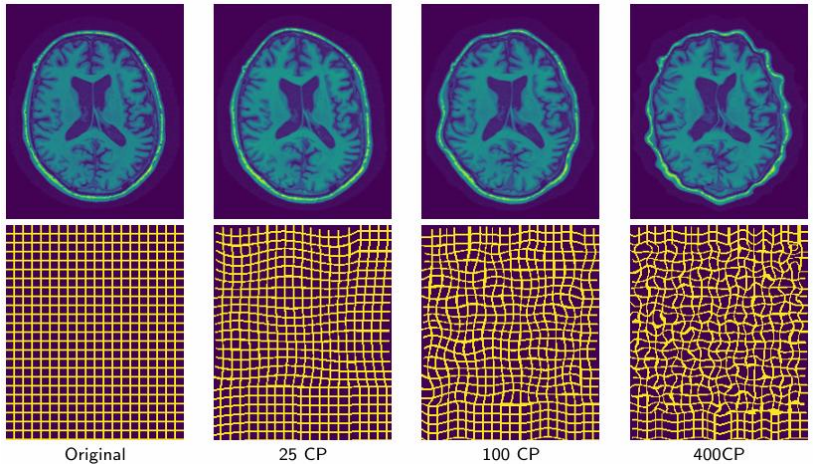
- Three kernels in three dimensions. There are $N_x \times N_y \times Nz$ control points placed in a uniform grid with spacing δ
  - $u(x') = \sum_{l=0}^{3} \sum_{m=0}^{3} \sum_{n=0}^{3} B_l(\mu_1)\, B_m(\mu_2)\, B_n(\mu_3)\, d_{i+l,j+m,k+n}$
  - $i = \left\lfloor \frac{x'_1}{\delta} \right\rfloor - 1, j = \left\lfloor \frac{x'_2}{\delta} \right\rfloor - 1, k = \left\lfloor \frac{x'_3}{\delta} \right\rfloor - 1$
  - $\mu_1 = \frac{x'_1}{\delta} - \left\lfloor \frac{x'_1}{\delta} \right\rfloor, \mu_2 = \frac{x'_2}{\delta} - \left\lfloor \frac{x'_2}{\delta} \right\rfloor, \mu_3 = \frac{x'_3}{\delta} - \left\lfloor \frac{x'_3}{\delta} \right\rfloor$
  - $B_0(s) = \frac{(1-s)^3}{6}, B_1(s) = \frac{(3s^3-6s^2+4)}{6}, B_2(s) = \frac{(-3s^3+3s^2+3s+1)}{6}, B_3(s) = \frac{s^3}{6}$
- Every pixel is influenced by $3 \times 3 \times 3$ control points, **needs many control points**, where we lay another grid onto the pixels, here we have a red control point every 5 pixels (gray)



- For every pixel, find the closest control point (its basis), usually the left top one (arbitrary), compute distance pixel to basis, plug those values into the interpolation stuff
- Choose the kernels so that the **derivatives are continuous** (just a definition, can have other conditions)
- Leads to smooth transformations (**only local changes**) that can also apply non-linear deformations to the objects in the image, the following is an example with 25 control points



- Transformations are always smooth even when the displacements at the control points are large
- Influence of increasing the number of control points, influence of a control point is a smaller place, picture becomes **less regular**, **more control points less smooth transformation**



- Gird of control points is often subsampled from the original image grid
- The parameters of the non-linear transformation model are the coefficients $\{d_n\}_{n=1}^{N}$
    - In 2D: (number of control points) * 2
    - In 3D: (number of control points) * 3
- One can again include an affine transformation similar to TPS models
- One can also consider optimizing locations of the control points (have the position also as a parameter) - increasing the number of parameters
- Transformations are bounded to be smooth due to the interpolation kernel
- Effects of each control point and the displacement field on it are local
- **Increasing the number of control points**, i.e., using a **finer grid**, can lead to **less smooth transformations**
- This provides the ability to perform multi-resolutional transformations, starting with few and increasing the number of control points
- FFD with B-splines is **mostly used for intensity-based registration** not necessarily for landmark-based

3.3.2.2 Physical models

Idea: Pixel / voxel-wise modeling - a displacement field for each grid point (no interpolation anymore)

- Assume a physical model of transformation often motivated from physical phenomenon, such as fluids and elastic body
- **Very large number of parameters** but restricted transformations based on the underlying model
- Coarse classification
    - Elastic body models
    - Fluid flow models
    - Curvature registration
    - Diffusion models
    - Flows of diffeomorphisms
- Not much detail is mathematically heavy
- Look for displacement vectors and the field of the displacements has to fulfill certain equations (the constraints, i.e., **reduces DOF**)
- Flow based models are different in their ODE that constrains it and also an integration
    - Theory of ODE says that if $v(x,t)$ is smooth then the resulting transformation is a diffeomorphism
    - Leads to the large deformation diffeomorphic metric mapping (LDDMM) registration algorithm
    - **Number of parameters are even higher**, **one for each pixel / voxel and time**
    - Used with regularization models on $v(x,t)$, reducing parameters

- A variation of the flow model is formulated with stationary vector field (SVF) (v does not depend on time), so only a velocity field per pixel, not time-dependent anymore (better for integration)

3.4 Metric / loss functions

	Same subject's images	Different subjects' images
Same modality	Intra-subject, intra-modality	Inter-subject, intra-modality
Different modality	Intra-subject, inter-modality	Inter-subject, inter-modality

- **Intra-subject is easier than inter-subject** (we expect the **same anatomy**)
  - Inter-subject registration is often **difficult** because differences in anatomies even when images show the same structures
  - Lack of exact correspondence between two images (presence of pathologies make registration even more difficult)
  - Alignment strategy would change, transformations are different
- **Intra-modality is easier than inter-modality**
  - Need to use different loss functions
  - Intensity-based loss functions that are not affected by differences in modality variations

3.5 Registration

3.5.1 Landmark-based registration

Main question: What loss metrics $L(\theta)$ do we use and how do we optimize, i.e., solve the $\arg_\theta \min$ problem?

3.5.1.1 Landmarks

- *General definition*: any conspicuous object which characterizes a neighborhood or district, e.g., when you see the Kremlin, you know you are in Moscow
- Landmarks in medical images
  - Prominent locations in an image
  - Distinct from their surroundings, similar to interest points from computer vision
  - Easily identifiable
  - Can be used to orient ourselves in an image
  - Landmarks will be used to **define correspondence** between images
  - Landmark definition depends on the imaged structure and type of images to align
  - Most commonly **manually placed**
- Landmarks for registration
  - Should be **identifiable across images** same landmark in both of the images
  - Corresponding landmarks will **guide registration** procedure and **help align** the images
  - Correspondence between landmarks define correspondence between different pixels / voxels

- Also works if one is MRI and the other is CT (when they are visible in both modalities, human places the landmarks)

- Landmark-based loss function
  - $L(\theta) = \sum_{n=1}^{N} ||x_n^{(1)} - T_\theta\left(x_n^{(2)}\right)||^2, x \in \mathbb{R}^d$
  - $x_n^{(1)}$ is the n-th landmark in the first image
  - $x_n^{(2)}$ is the corresponding landmark in the second image
  - $T_\theta\left(x_n^{(2)}\right)$ is the **transformed version** of $x_n^{(2)}$
  - θ is the set of transformation parameters
  - d is the dimension of the problem, e.g., 2D, 3D, 3D + time, …
  - $||\cdot||^2$ is the l_2 loss
  - Other types of losses are also possible

3.5.1.2 Linear registration

- Given the landmark-based loss function
  - $L(\theta) = \sum_{n=1}^{N} ||x_n^{(1)} - T_\theta\left(x_n^{(2)}\right)||_2^2, x \in \mathbb{R}^d$
- When aligning with linear transformation, T_θ is defined through a matrix A and a translation t
  - $L(A, t) = \sum_{n=1}^{N} ||x_n^{(1)} - \left(Ax_n^{(2)} + t\right)||_2^2, x \in \mathbb{R}^d$
- The loss is **minimized** to determine the best transformation parameters A and t
  - $\arg_{A,t} \min \sum_{n=1}^{N} ||x_n^{(1)} - \left(Ax_n^{(2)} + t\right)||_2^2$
- *Example*: Rigid transformation, same subject, same modality

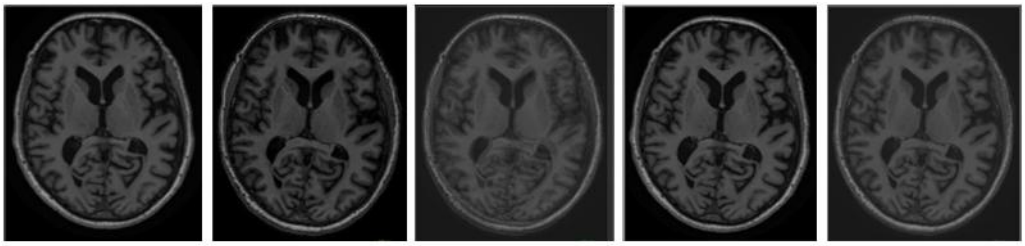


Figure: From left to right: time point #1, time point #2, overlay, time point #1 registered with rigid registration with landmarks, overlay after registration

  - Result from rigid transformation - T_θ is a rigid transformation, i.e., rotation and translation
  - Using only 4 landmarks the rigid registration correctly aligns the images
  - Notice that the overlay is sharper - meaning **good alignment**
- *Example*: Rigid transformation, different subject, same modality
  - **Rigid registration may not be enough** to account for **differences between different subjects**
  - Landmarks: trachea bifurcation, liver tip, and pelvis tips
  - Notice that the landmarks **do not match perfectly**
  - The head does not have any landmarks, only chances for good overlap

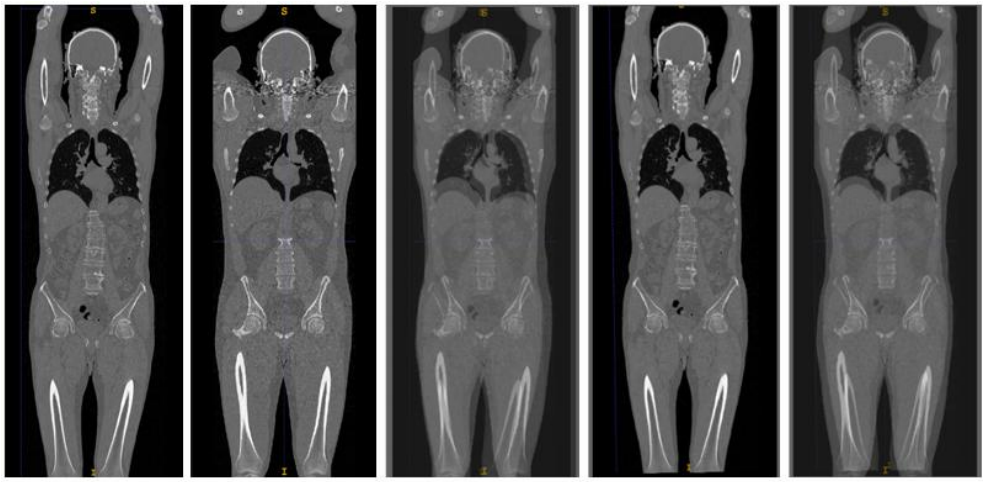


Figure: From left to right: subject #1, subject #2, overlay, subject #1 registered with rigid registration with landmarks, overlay after registration

- *Example*: Affine transformation, different subject, same modality
    - Affine registration may be a little bit better in accounting for inter-subject differences
    - Notice the landmarks match better
    - At three landmarks, alignment for the landmarks are perfect but for the other parts it is not. That is because affine transformation has 6 unknowns, and 6 equalities - given by three landmarks - given a well-determined system of equations
    - Increasing number of landmarks yields worst alignment for the landmarks but better for other areas. **More landmarks lead to an over-determined system of equations**
    - Why does the affine transformation **not align** all the landmarks perfectly?
        - With 6 parameters, it does not have the degrees of freedom to satisfy more than 3 correspondences in 2D. Note that 3 correspondences lead to 6 equations; each coordinate will give one equation

3.5.1.3 Non-linear alignment with thin plate splines

- *From TPS*: the control points do not have to be uniformly spaced, they can be randomly dispersed, TPS is extensively used, especially for landmark-based registration
- Aligning with TPS
    - Given the landmark-based loss function
        - $L(\theta) = \sum_{n=1}^{N} ||x_n^{(1)} - T_\theta\left(x_n^{(2)}\right)||_2^2, x \in \mathbb{R}^d$
    - When aligning with TPS, the cost function becomes
        - $L = \sum_{n=1}^{N} ||x_n^{(1)} - x_n^{(2)} - \sum_{k=1}^{N} \phi\left(\left|x_n^{(2)} - x_k^{(2)}\right|\right) d_k ||_2^2, x \in \mathbb{R}^d$
    - Note that the displacement vectors are defined through the same landmarks, similar to linear, the loss is minimized to determine the best transformation parameters
        - $\arg_{\{d_k\}_{k=1,\dots,N}} \min \sum_{n=1}^{N} ||x_n^{(1)} - x_n^{(2)} - \sum_{k=1}^{N} \phi\left(\left|x_n^{(2)} - x_k^{(2)}\right|\right) d_k ||_2^2$

- After determining the optimal parameters d_k^* one can use it to find the displacement vectors everywhere in the target image, not just the landmarks
  - $T_\theta\left(x^{(2)}\right) = x^{(2)} + \phi\left(\left|x_n^{(2)} - x_k^{(2)}\right|\right) d_k^*$

- *Example*: TPS transformation, same subject, same modality

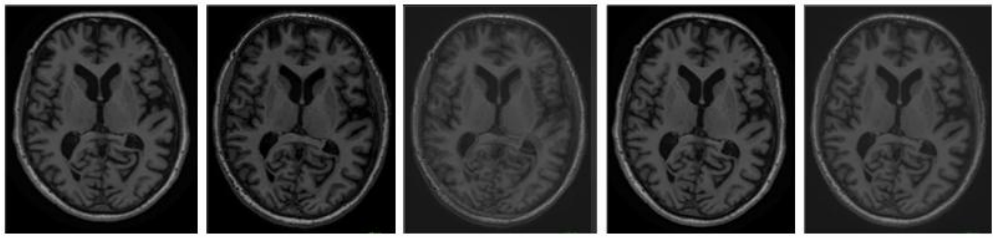


Figure: From left to right: time point #1, time point #2, overlay, time point #1 registered with TPS using 5 landmarks, overlay after registration

  - Results from TPS registration - T_θ is a TPS transformation
  - Notice that the over lay is sharper - meaning good alignment
- *Example*: TPS transformation, different subject, same modality

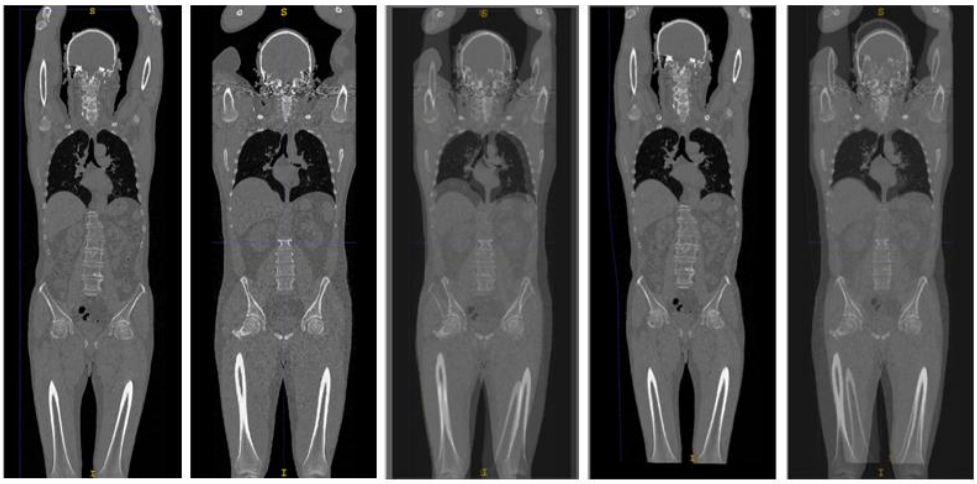


Figure: From left to right: subject #1, subject #2, overlay, subject #1 registered with TPS registration with 12 landmarks, overlay after registration

  - TPS has **larger degrees of freedom** and model complicated transformations
  - 12 different landmarks are used to get a good alignment
  - Notice that landmarks are perfectly aligned but further, the alignment in the rest of the image is also much better than linear transformation

General comparison: TPS can model deformable / non-linear transformations, which is good since people are not linear naturally

3.5.2 Intensity based registration

3.5.2.1 Intensity-based loss function

- $L = d(I, T_\theta \circ J)$
- Distance $d(\cdot,\cdot)$ is defined over the intensities of the fixed image I and the moving image J after transformation, i.e., $T_\theta \circ J$

- There are **no landmarks**, hence no input information on corresponding locations in the images
- Harder problems as there is no unique solution to the problem
- Applicable in **more general situations** as it does not require landmark placement
- Definition of $d(\cdot,\cdot)$ is very important and depends on the type of registration

3.5.2.2 Intra-modality loss metrics

One of the most commonly used metrics is the **Sum of Squared Distances (SSD)**

- When the images are the **same**, then **SSD is 0**
- There is no unique solution to this problem, one can warp images in any way to perfectly minimize the SSD to its minimum value 0
- It assumes the **intensities are comparable** across the images, hence, it **would not work for registering images with different modalities**
- Image gradients are important for SSD: only changes in transformation that are parallel to the image gradient can change, hence reduce the loss
- Improvement of the SSD
    - Using a robust loss instead of l_2, such as l_1. Optimization may become more difficult
    - Instead of using intensities, one can consider features extracted from larger patches

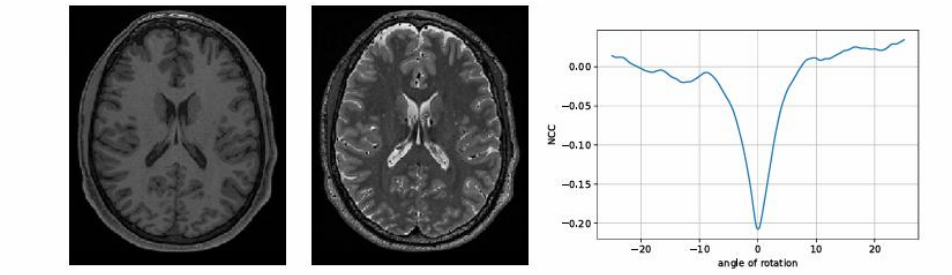
3.5.2.3 Inter-modality loss metrics

- SSD assumes intensities are comparable
- In inter-modality registration, when we aligning images from **different modalities**, we **cannot assume they are directly comparable**
- Inter-modality loss metrics are of statistical form, two of the most commonly used ones are
    - **Mutual information**
    - **Normalized cross correlation**

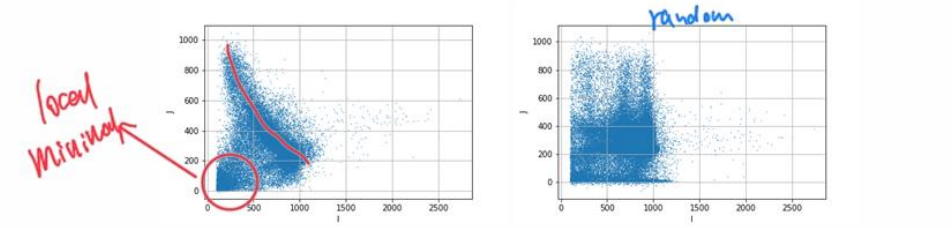
  They both assume there is a **statistical relationship between the intensities of the two images**

Normalized Cross Correlation (NCC)

- NCC is the 2D version of the Pearson correlation coefficient
- It assumes **pixel intensities as independent random variables**
- It achieves maximum score of 1 if $I(x)$ and $J(T_\theta^{-1}(x))$ are **linearly related** and the same relationship holds for all $x \in \Omega$
- NCC can be both **positive and negative**, both cases are fine, and that's why the loss is often defined in terms of NCC squared
- NCC **does not go down to -1**, this is because in reality, the relationship is never perfectly linear, moreover, background pixel may influence the total loss



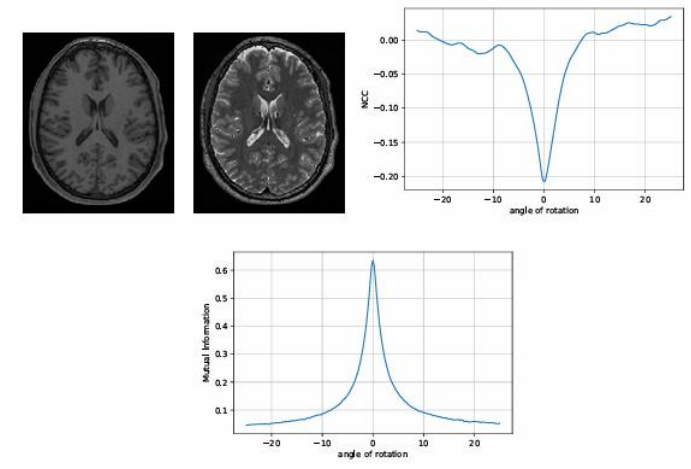
T1w image, T2w image, NCC computed at different angle of rotations. Images are aligned originally. NCC is computed only in the foreground.



Intensity scatter plots at 0 and 10 degrees of rotation

Mutual information

- Mutual information is defined as
  - $L(\theta) = d(I, T_\theta \circ J) = MI(I, J)$
  - $MI(I, J) = D_{KL}\big(p(I,J) \big| p(I)p(J)\big)$
  - Where $D_{KL}(P|Q)$ is the Kullback-Leibler divergence that measures a distance between two distributions P and Q. MI measures KL divergence between the joint distribution of intensities and product of marginal distributions of the same
  - It assumes **pixel intensities as independent random variables**
  - MI takes the value 0 if $p(I,J) = p(I)p(J)$, when the intensities in the two images are **independent** from each other – no statistical relationship
  - Images that are **not aligned** would have **no statistical relationship**, thus, **low MI**
  - **Perfectly aligned** images should have a **strong statistical relationship**, hence, MI should be **maximized** during a registration process
- *Example*:



Comparison of NCC and MI:

- **MI is more powerful**, it can capture complicated relationships. For instance, it does not have ripples for large angles in the rotation example
- MI can be **difficult to compute**, NCC is easier to compute

3.5.3 Optimization

- Optimization for image registration is a difficult problem in general
- Landmark-based cost function is easier of the two, as long as the degrees of freedom of the transformation is consistent with the given number of landmarks
- Optimization for linear transformation models is easier than non-linear transformation models
- In general, **intensity-based registration problems using non-linear transformation** models (named as deformable registration) **are ill-posed**, i.e., there are many different solutions that can give similar results
- To overcome the ill-posed nature of the problem, regularization is often used when using non-linear transformation models

3.5.3.1 Regularization

To make the optimization well-posed, **a regularization is often added** to the loss functions $L(\theta) = d(I, T_\theta \circ J) + \lambda r(T_\theta)$, $T_\theta = x + u(x)$, where $r(T_\theta)$ prefers some sort of transformations over others and λ is a weight

- **Tikhonov regularization** schemes that prefer smoothly varying $u(x)$ are often used
  - $r(T_\theta) = \int \phi(\nabla u)\, dx$
  - Where ∇ denotes gradient in space and ϕ is a convex function with minimum at zero
  - *Examples*:
    - l_2 regularization that prefers smoothly varying displacement fields
      - $r(T_\theta) = \int (\nabla u)^2\, dx$
    - l_1 regularization that prefers piece-wise constant $u(x)$
      - $r(T_\theta) = \int |\nabla u|\, dx$
- Energy terms from continuum mechanics are also commonly used
  - Linear elastic regularization uses the Cauchy strain tensor $V = V(u)$ and minimized the strain energy caused by the displacement field
  - Hyperelastic regularization allows for larger deformations using the simplest material model, the Saint Venant-Kirchhoff model with the Green-St. Venan strain tensor $E = E(u)$

3.5.3.2 Gradient descent

The complete registration loss function, together with the data term $d()$ and the regularization $r()$ is ready for optimization

- Linear transformation models – linear registration problem
  - $L(\theta) = d(I, T_\theta \circ J)$, $T_\theta = Ax + t$
- Non-linear transformation models – non-linear registration problem

    - $L(\theta) = d(I, T_\theta \circ J) + \lambda r(T_\theta), T_\theta = x + u(x)$
- Most commonly, the optimization can be done via continuous methods
    - $\theta_{t+1} = \theta_t + \alpha g(L(\theta_t))$
    - Where α is a step-size parameter, t indices optimization iterations, and $g()$ is a gradient-based function, the simplest being the gradient
    - $\theta_{t+1} = \theta_t - \alpha \frac{dL(\theta)}{d\theta}|_{\theta_t}$
- Second order optimization schemes and discrete formulations of the optimization is also possible; gradient free techniques are possible

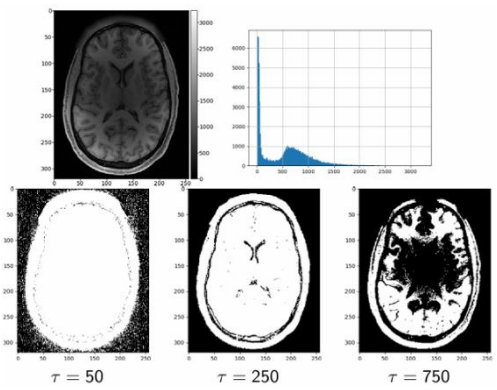
3.5.3.3 Sequential optimization

- It is common to **first align images with linear transformation** followed by deformable non-linear registration, which helps to get a better alignment
- Even with regularization terms, **registration can get stuck in local minima**. To avoid this, multiscale optimization is used. At coarse resolution, registration is easier because there are not that many details in the images. Registration starts at the coarsest scale where the problem of aligning the images is easier, results are used as initialization of the next step.

4. Segmentation

What is segmentation?

- At each pixel / voxel, assign a label
- Labels can be
  - Organ: liver, spine, etc.
  - Part of organs: individual vertebrate, etc.
  - Lesions: tumors, pathologies, etc.
- Information at each pixel
  - Intensity at and around the pixel
  - Intensity at different modalities



4.1 Basic segmentation

4.1.1 Thresholding

- $L(x) = 1$ if $I(x) > \tau$, $L(x) = 0$ if $I(x) \leq \tau$, where $I(x)$ is the intensity of the pixel / voxel
- Particularly **useful in easy foreground-background subtraction**
- Preprocessing method for region of interest determination
- Used routinely in histology and other microscopic images where stains provide the necessary contrast
- Use histogram analysis to determine the threshold
  - Too high: not capture enough information and having noise
  - Have an **intensity distribution** histogram and **make a cut when there is a low part**
  - In colored images, do the cut in all three channels
- Otsu's thresholding peaks and troughs of the histogram
  - $\min(p_1\sigma_1^2 + p_2\sigma_2^2)$
  - p_1 and p_2 are the number of pixels in fore and background
  - σ_1^2 and σ_2^2 are the intensity variances in the groups

Note: There are global thresholding (for the entire image) and local thresholding (per ROI or patch). **Image noise** can lead to **isolated foreground or background islands**. Multiple objects would require multiple thresholds, and a nice generalization is K-means algorithm.

4.1.2 K-means clustering

- Unsupervised clustering algorithm, assigning pixels to clusters so that they have the **closest distance to the means**
- Voxels / pixels are clustered according to features such as intensity, colors, temporal sequence, gradient, wavelet transform, etc.
- Automatically assigns each voxel / pixel to one of the N clusters, we want to distribute pixels to N groups using features, note that when we have **2 features**, then it is the **same as thresholding**
- Criteria:

- Homogeneity within groups
- Reducing variance over features within group
- Take into account multiple features

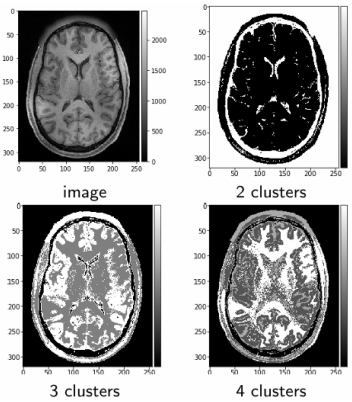
Note: K-means is **easy to implement**, it aims to find cluster assignments and mean values that explain the data as best as possible with the given number of clusters. The **choice of number of clusters has a major influence**, methods include heuristic methods based on variance, non-parametric Bayesian methods. **Initialization is important**, K-means can **get stuck in local minima**. Probabilistic formulation is possible that can be **more robust and allows extensions**, e.g., $\max(\ln p(I))$.

Algorithm 1 K-Means

1: Describe each pixel with features $f(x) \in \mathbb{R}^m$, e.g. for only intensity $m = 1$ and for RGB $m = 3$
2: Randomly choose N points as the group centers, $m_i^0 \in \mathbb{R}^m,\ i = 1, \ldots, N$
3: **while** $\|m_i^t - m_i^{t-1}\| > \epsilon$ for any i **do**
4: Assign groups: $c(x) = \arg_i \min \|f(x) - m_i\|_2$
5: Recompute means:

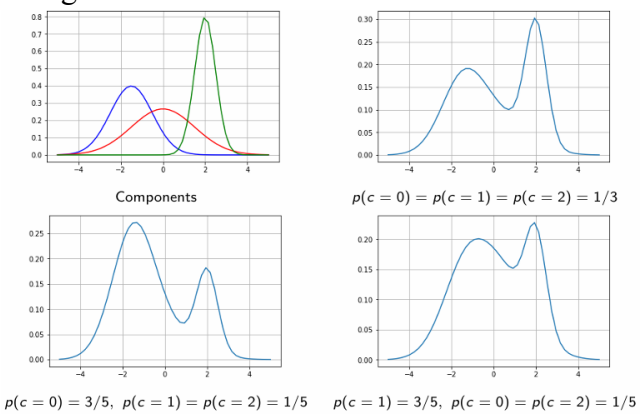
$$m_i = \frac{\sum_{x \in \Omega} \delta_{i=c(x)} f(x)}{\sum_{x \in \Omega} \delta_{i=c(x)}},\quad \delta_{i=c(x)} = \begin{cases} 0, & i \neq c(x) \\ 1, & i = c(x) \end{cases}$$

6: **end while**



4.1.3 Expectation-Maximization algorithm

- Start by assuming **all pixels are independent** and model the intensities as a mixture model, given a feature vector f, the mixture model is
  - $p(f) = \sum_{n=1}^{N} p(f|c = n)p(c = n)$
    - $p(c)$ is the **prior probability of observing (latent) label** c, also called **mixture coefficients**
    - $p(f|c)$ is the **likelihood model** defined as that point, often taken as a Gaussian (gaussian mixture model), having two class specific parameters
- Mixture model - 1D with 3 (latent) labels: even with the same labels, the probability changes the distribution

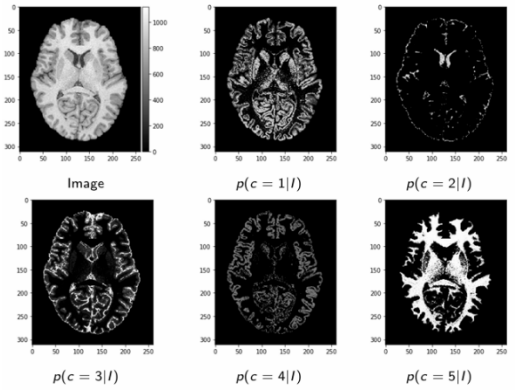


- Fit the model parameters to the image intensities in an unsupervised way, we assume
  - A number of classes N

- Intensities at different pixels are independent from each other
- The model to fit is
  - $\max_\theta \log p(I|\theta) = \max_\theta \log \sum_{x\in\Omega} p(I(x)|\theta) = \max_\theta \sum_{x\in\Omega} \log p(I(x)|\theta)$
  - $\log p(I(x)|\theta) = \log \sum_{n-1}^{N} p(I(x)|c(x)=n)p(c(x)=n)$
    - θ is the set of model parameters
    - $p(c(x)=n) = \pi_n$ are class probabilities
    - μ_n, Σ_n are likelihood parameters
- After fitting, segmentation is defined through posterior-distribution using optimal parameters
  - $p(c(x)|I(x))$
  - $c^*(x) = \arg\max p(c(x)|I(x))$
- We can optimize using gradient descent - **difficult**, Expectation-Maximization is a better alternative of gradient descent

Expectation-Maximization algorithm

- **General idea**: two alternating steps
  - E-step: Assume a set of likelihood parameters and **"soft" assign samples to labels**
  - M-step: **Assume soft assignments** and maximize likelihood parameters, e.g., find the best mean and standard deviations of the Gaussians
- Iterative algorithm with two step process
  - E-step: compute $p(c|I,\theta_{old})$ given θ_{old}
    - $p(c(x)|I(x),\theta_{old}) = \frac{p(I(x)|c(x),\theta_{old})\, p(c(x)|\theta_{old})}{p(I|\theta_{old})} = \frac{\mathcal{N}(I(x)|\mu_{c(x)}^{old},\Sigma_{c(x)}^{old})\pi_{c(x)}^{old}}{\sum_{n=1}^{N}\mathcal{N}(I(x)|\mu_n^{old},\Sigma_n^{old})\pi_n^{old}}$
  - M-step: maximize $Q(\theta|\theta_{old})$, i.e., $\max_\theta \mathbb{E}_{p(x|I,\theta_{old})}[\log p(I,c|\theta)]$
    - $\pi_n = \frac{\sum_{x\in\Omega} p(c(x)=n|I(x),\theta_{old})}{|\Omega|}$
    - $\mu_n = \frac{\sum_{x\in\Omega} I(x)p(c(x)=n|I(x),\theta_{old})}{\sum_{x\in\Omega} p(c(x)=n|I(x),\theta_{old})}$
    - $\Sigma_n = \frac{\sum_{x\in\Omega}(I(x)-\mu_n)(I(x)-\mu_n)^\top p(c(x)=n|I(x),\theta_{old})}{\sum_{x\in\Omega} p(c(x)=n|I(x),\theta_{old})}$
  - Set $\theta_{old} = \theta^*$
  - Start with a random θ_{old} iterate E and M steps



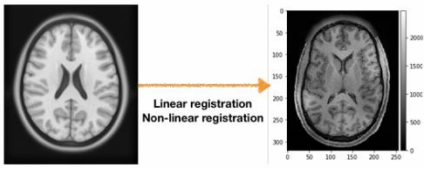
<- 5 components example

Note: EM algorithm is **robust**. **Initialization** can be important if done badly, **number of components is important**, need **prior information** regarding **how many different structures** would like to extract. Outputs already useful. Extension to include better prior is possible such as atlas.

4.2 Atlas-based segmentation

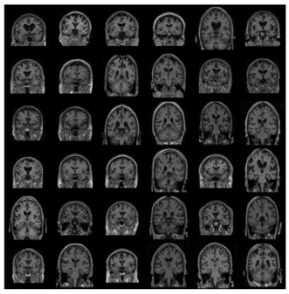
4.2.1 Formulation

- **Adding prior information** to the segmentation - no longer a simple clustering
- **Atlas is registered to the image**
- Template at a reference frame
  - Align the atlas to the image
  - Linear or non-linear registration
  - The image is mapped on a reference frame
  - At each point, we have a rough idea what structure to find



- For each location provides information about what is there
- **Deterministic** - world atlas
- **Probabilistic**, i.e., $p(c(x))$ - brain atlas
  - Start with the mixture model
    - $p(I(x)|\theta) = \sum_{n=1}^{N} p(I(x)|c(x) = n, \theta)p(c(x) = n|\theta) = \sum_{n=1}^{N} \mathcal{N}(I(x)|\mu_n, \Sigma_n)\pi_n$
    - Where $\theta = \{\mu_n, \Sigma_n, \pi_n\}_{n=1}^{N}$, parameters are determined in an unsupervised way
  - Add the atlas information
    - $p(I(x)|\theta) = \sum_{n=1}^{N} \mathcal{N}(I(x)|\mu_n, \Sigma_n)p_{\text{atlas}}(c(x) = n)$
    - Where $\theta = \{\mu_n, \Sigma_n\}_{n=1}^{N}$, atlas brings in prior information
  - Posterior distribution takes into account the atlas
    - $p\big(c(x) = n\big|I(x)\big) = \frac{\mathcal{N}\big(I(x)\big|\mu_n, \Sigma_n\big)p_{\text{atlas}}(c(x)=n)}{\sum_{n=1}^{N} \mathcal{N}\big(I(x)\big|\mu_n, \Sigma_n\big)p_{\text{atlas}}(c(x)=n)}$
    - Posterior takes into account the **atlas** and the **image intensities**
    - Information propagation from the atlas to the image
    - No need to optimize for class probabilities $p(c(x) = n) = \pi_n$
    - Still need to optimize for μ_n and Σ_n, especially for MRI
- Applicable to any anatomical structure
- Created by averaging many different images
- Advantages
  - Much better segmentation
  - **No randomness** in components, we know what we ask for
  - Used very commonly
  - Easily generalizable to other body parts with an appropriate atlas
  - Can be generalized to other modalities, may fix all parameters for CT for instance
  - An outlier class can be added to detect outliers
- Disadvantages
  - Need an atlas for each body part
  - Atlas itself is very important, need an atlas for different age groups, gender, race, etc.

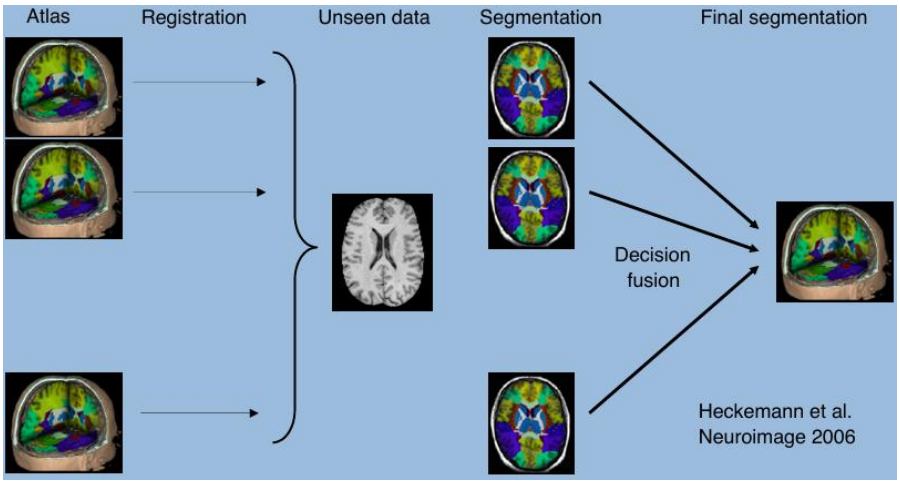
    - Accuracy of image registration is very important
- Remedies to improve robustness to the atlas and registration
    - **Multi-atlas segmentation**
    - **Patch matching**: Use patch-matching to perform local searches in a sliding window manner



4.2.2 Multi-atlas segmentation

Idea: instead of using one, use **many different atlases**, propagate information from all of them and aggregate (more registration)

- **Align all the atlas images to the test image**
- Aggregate information coming from all the atlases
- Remodeling $p_{\text{atlas}}(c(x) = n)$
- During aggregate weigh atlases based on the quality of registration
    - $w_m \propto \left|\left|I(x) - A_m\right|\right|_2$
    - $w_m \propto \left|\left|I(x) - T \circ A_m\right|\right|_2$
    - $w_m \propto \left|J_{T_{A_m \to I}}\right|$
- **Many atlases reduce importance of a single**
- **Less likely to have bad registration** with weighing scheme
- Leads to elegant probabilistic models
- Advantages
    - Solves some of the problems of atlas-based segmentation
    - **Highly accurate results**
    - State-of-the-art for a very long time
- Disadvantages
    - Often requires non-linear registration
    - Computationally **very expensive** due to many registrations



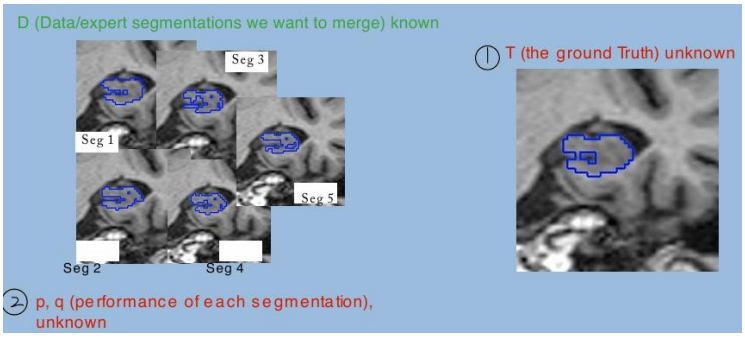
- How do we merge:
    - **Global fusion strategies**:

- **Majority voting**: Take a democratic vote which label appears the most often at this pixel (use an odd number of atlases)
    - Dice coefficient used to see how many atlases is enough (overlap of ground truth segmentation over the segmentation with N atlases increases in the beginning and then stagnates)
- **Weighted voting**: pixel-wise aggregation, based on averaging the distance transforms of the segmentations

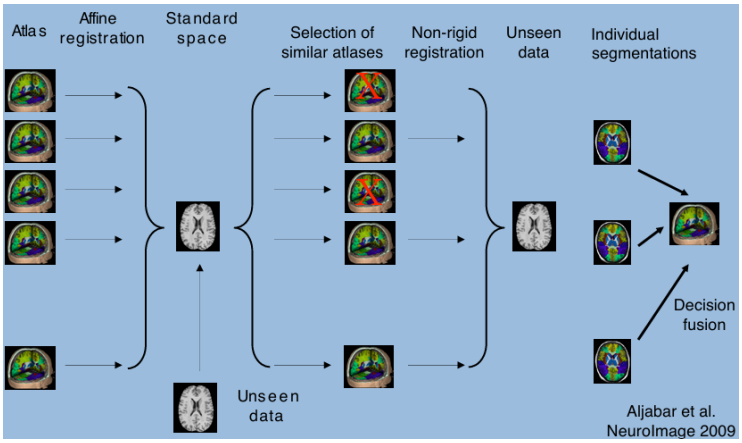
- **Local fusion strategies**:
    - Locally weighted fusion
    - Simultaneous Truth And Performance Level Estimation (STAPLE)
- **Shape based averaging**

4.2.2.1 Simultaneous truth and performance level estimation (STAPLE)

- What is the most probable underlying true segmentations when you see the segmentations?
- Analogy: a senior doctor and an apprentice and a biomedical engineer working there look at segmentations. **The more experienced, the more weight the opinion (segmentation) has**
- Can we find the performance of each segmentation (p, q) and the ground truth (T) that maximizes the likelihood of observing the segmentation (D)?

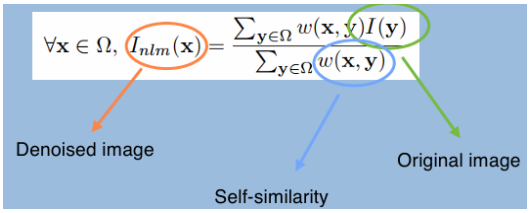


- With the **fusion and selection** strategy, we can reduce the registration time



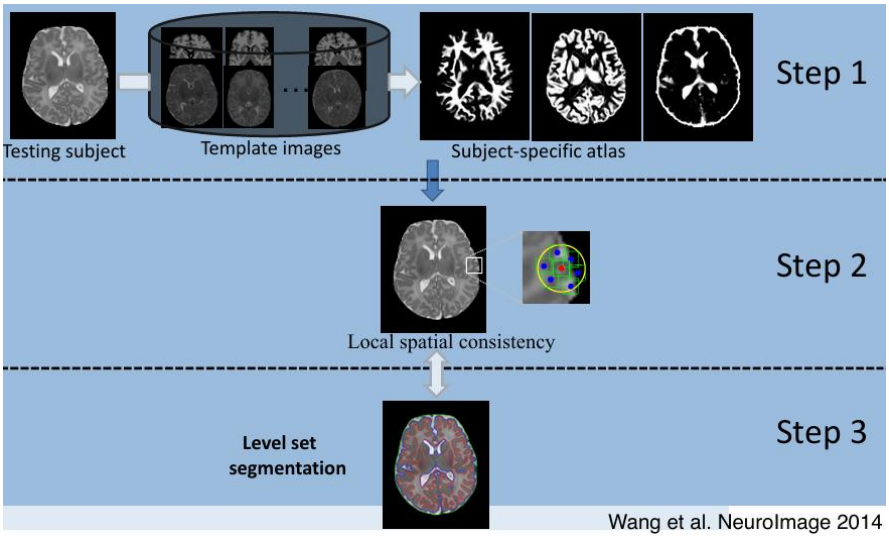
-Select the **most similar atlases** and only register those
-**Advantage**: faster and more accurate (the not similar atlases add noise)
-**Disadvantage**: we need a method to do that

4.3 Patch-based segmentation



Idea: Use **coarse registration**, i.e., rigid or affine registration, use the same concept as label fusion

- Process
  - Start with denoising the image
    - E.g., neighborhood averaging strategy called **non-local means** (NLM): assumes that every patch in a natural image has **many similar patches** in the same image
  - Apply the same idea as in denoising to label fusion: when two pixels have **similar intensities** they may have **similar labels** (similar tissues)
- Patch-based multi-organ segmentation of brain structures: hierarchical, patch-based label fusion process, it gets finer with added hierarchy



- At step 1, we create a **subject-specific atlas**, an atlas that has been constructed for that person (biased towards that), then it gets more accurate
  - How do we get that personalized atlas?
    - Having many template images with the region of interest marked and another **bigger square to compensate for misalignment**
    - **Elastic net optimization**: we get an **approximation** (not segmentation) of the person's brain
    - Use that **subject-specific atlas as a prior** for looking at patches, then use those as prior and at last, use the whole image as a prior
    - Also do a leave-one-out cross validation
    - **Multi-patch based segmentation has better dice-coefficient than majority voting**
- Patch-based labeling vs. sparse representation classification (SRC) vs. Discriminative dictionary learning for segmentation (DDLS)
  - SRC assumes **a target patch represented by a few representative patches**

- DDLS adds a **learned linear classifier**
- DDLS **decrease computational burden** via **smaller size dictionary**, better exploit discriminative power of patch library
    - The steps of DDLS
        - Extract patch p
        - Solve our problem, describe the patch with variable (get alphas)
        - Use the alphas to do segmentation
    - DDLS may have some improvements

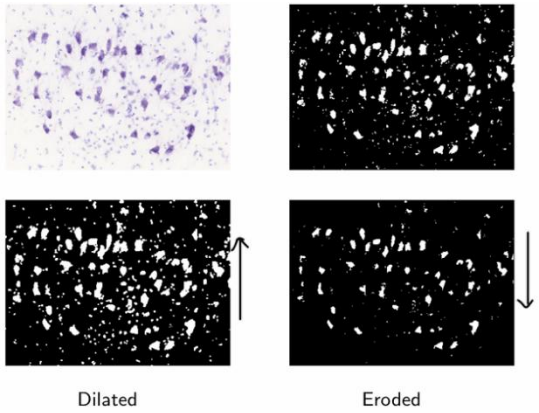
4.4 Spatial constraints in segmentation

So far, we have not considered spatial information

- All the pixels are **independent**
- Atlases include some spatial information but the final segmentation still assumed **independence**
- In real images this is a **highly unreasonable assumption**
- Neighboring pixels are more likely to belong to the same object
- Two main approaches to integrate spatial information
    - Pre / post-processing
        - Smoothing the images before hand
        - Morphological operations
    - Random field priors
        - Markov random fields
        - Conditional random fields

4.4.1 Morphological operations

- Dilation: $C = A \oplus B \triangleq \{x | B_x \cap A \neq 0\}$
- Erosion: $C = A \ominus B \triangleq \{x | B_x \subseteq A\}$
- A is the image and B is a structural element, e.g., $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$.
- Properties
    - Shift-invariant
    - Non-linear
    - Based on neighboring pixels defined through structural elements
    - Applied in a sliding window approach, e.g., structural element slides across the image
    - View binary images as sets of pixels
    - Defined in binary but extended to gray-level images
    - Can also combine the two to get useful operations
        - Opening $(A \ominus B) \oplus B$ – erosion then dilation, remove island
        - Closing $(A \oplus B) \ominus B$ – dilation then erosion, more contiguous



4.4.2 Random field priors

Idea: formulating neighborhood consistency in a probabilistic model / energy model, main idea is to punish inconsistency in labeling of neighboring voxels, use probabilistic formulation and Markovian property to define punishment

4.4.2.1 Markov random field

- Focusing on the joint distribution of labels and intensities
  - $p\big(I(x)\big) = \sum_{n=1}^{N} p(I(x)|c(x) = n)p(c(x) = n)$
  - $p\big(I(x), c(x)\big) = p(I(x)|c(x))p(c(x))$
  - $p(I, c) = \prod_{x\in\Omega} p\big(I(x)\big|c(x)\big)p\big(c(x)\big)$

  all pixels are **independent**
- In order to model the connection between pixels, we change the prior
  - $p(I, c) = p(c)\prod_{x\in\Omega} p\big(I(x)\big|c(x)\big)$

  Joint distribution changes

4.4.2.2 Markovian property

- MRF sets up the prior distribution based on the Markovian property
- Specific form depends on the neighboring structure
  - $G(x) =$ neighbors of x
  - $p(c(x)|c(/x)) = p(c(x)|c(G(x)))$
  - $p(c(x))$ only depends on its immediate neighbors if all others are given
- Energy formulation
  - $p(c(x)|c(/x)) = p(c(x)|c(G(x)))$
  - A common way to define it is through defining an energy
    - $E(c) = \sum_{x\in\Omega}\sum_{y\in G(x)} d\big(c(x), c(y)\big)$
    - $d(c(x), c(y))$ is a distance between the labels
  - Given the energy we can define a probability distribution
    - $p(c) = \frac{1}{Z}\exp\{-E(c)\}$
    - Z is the normalization constant, lower distance means higher probability, it enforces consistency
- The posterior becomes $p(c|I) = \frac{p(I|c)p(c)}{p(I)}$
- The maximization is $\arg\max_{c} p(c|I) = \arg\max_{c} p(I|c)p(c) =$ $\arg\max_{c} \exp\{-E(c)\}\prod_{x\in\Omega} p(I(x)|c(x))$
  - Assume given the label of a voxel, its intensity is independent from the intensities of other voxels
- End up with an energy minimization formulation
  - $\arg\min_{c}\{\sum_{x\in\Omega} g\big(I(x)\big|\theta_{c(x)}\big) + \sum_{x\in\Omega}\sum_{y\in G(x)} d\big(c(x), c(y)\big)\}$
  - $\sum_{x\in\Omega} g\big(I(x)\big|\theta_{c(x)}\big)$ is the **unary term** (data model), the data term in the model

- $\sum_{x\in\Omega}\sum_{y\in G(x)} d\big(c(x),c(y)\big)$ is the **pairwise term**, the consistency term that imposes neighboring pixels to have similar labels
- Note that when $c(x)$ is a continuous variable just like regression problems, the optimization is easy, but hard when it is categorical variable
- **Remedy: stochastic relaxation / Gibbs sampling**

4.4.2.3 Stochastic relaxation / Gibbs sampling principle

- Iterative method to approximate solution
- Monte-Carlo sampling method
- Sample from the posterior distribution is hard
- Much easier to sample from $p(c(x)|c(/x),I) = \frac{p(I(x)|c(x))p(c(x)|c(G(x)))}{\sum_{c(x)} p(I(x)|c(x))p(c(x)|c(G(x)))}$
- Start from a random c_0 assignment, such as $\arg\max_{c} \prod_{x\in\Omega} p\big(I(x)\big|c(x)\big)$
- Sample from the **posterior of each voxel** given the others
- Iterate over all voxels and some number of times
- Properties
  - Stochastic in nature - every time you run you get something else
  - Complicated optimization problem
- Advantage: Very simple implementation and effective
- Disadvantages:
  - Solution depends on parameters and initial conditions
  - May not converge to a pleasing result
  - Convergence may be slow

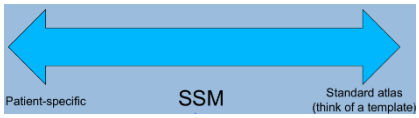
4.4.2.4 Conditional random fields

- Use the energy formulation in a discriminative way
- Consider a model that directly predicts $p(c(x)|I)$ **without the Bayesian treatment**
  - $\arg\min_{c}\{\sum_{x\in\Omega} g(c(x)|I) + \sum_{x\in\Omega}\sum_{y\in G(x)} d\big(c(x),c(y)\big)\}$
- Explicitly model g
- Can be used with machine learning algorithms outputs
- Unary term can be the predictions of a random forest
- Cleaning up the machine learning segmentation results

4.5 Active shape models for segmentation

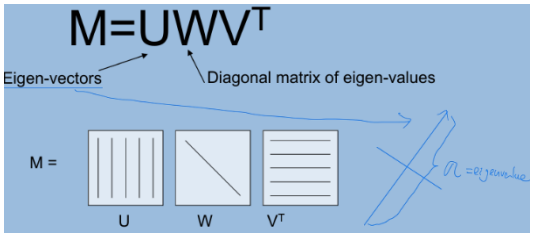
4.5.1 Statistical shape modeling setup and recap

- Prior knowledge (what we know in advance) vs. observation (what we see)
  - Patient specific -> lots of observation
  - Standard atlas -> lots of prior
  - **Statistical shape model (SSM) in between**



- Principal component analysis (PCA) to compute the SSM
  - PCA is used to reduce the dimensionality of a **linear** system
  - PCA projects the original data onto a lower dimensional space

    - PCA yields the **main direction** (and **secondary directions**), and creates an **orthogonal coordinate system** that **maximizes the variance** of the points in each direction
    - Points can be referenced along the main component (axis) using a parameter b
    - Removes some information - noise
- Single value decomposition (SVD) **diagonalizes the covariance matrix** of the original data. It directly yields the eigen-vectors and eigen-values of $Cov(M)$



    - Directly yields the eigen-vectors, most important one comes first
- PCA **assumes** that the original data follows a **Gaussian distribution** (even if in reality it does not)
- There also exist several linear and non-linear dimensionality reduction techniques such as factor analysis, multidimensional scaling (MDS), etc.
    - PCA is cheap and easy to interpret; for medical images, there are also not so many pictures around to make good non-linear things; PCA does the job

4.5.1.1 Shape representation: point clouds

- Create the point cloud that represents a shape (e.g., a femur)
- Every surface point $p_i = \{x_i, y_i, z_i\}$
- The surface is represented by concatenating the points
    - $S = \{p_1, p_2, \cdots, p_n\} \Rightarrow s_i = \{x_{i,1}, y_{i,1}, z_{i,1}, x_{i,2}, y_{i,2}, z_{i,2}, \cdots, x_{i,n}, y_{i,n}, z_{i,n}\}$
    - n is the number of points that constitute the surface
- Assume we have two different objects of the same class (i.e., two femurs from two different patients)
    - Initially, isosurfacing results in point clouds with a different number of points
    - We should establish anatomical correspondence between the points
    - Two clouds should have the same amount of points
- So far we have
    - A set of aligned shapes
    - Each shape has the same number of surface points
    - Surface points correspond anatomically and / or spatially
    - Each shape is represented by a vector of coordinates
        - We can represent the set of shapes in a matrix by concatenating the m shape vectors s_i
- Modeling of shape variability
    - Subtract mean from each instance of $M \rightarrow D = M - mean(M)$
    - Apply SVD directly on matrix D
- Model evaluation: Commonly, three measures are used to evaluate a statistical shape model

- **Compactness** is a measure of **how good the data reduction process is**, i.e., $C(\lambda_t) = \frac{\lambda_t}{\lambda_{total}}$. If a model is compact enough, it allows the generation of new shape instances **using as few parameters as possible**. It is represented by a curve showing how much of the total variance a certain number of modes of variation can capture
- **Generalization** is performed using **cross-validation reconstruction** experiments. It represents the ability of the model to generate new instances from the class.
    - The errors are measured and averaged over all training instances yielding the generalization curve of the model generalization
    - It is also dependent on the number of modes of variation (parameters used in the reconstruction process)
- **Specificity** is an indicator of how good the model is in generating instances **similar to those presented in the training set**. It is measured by generating a large number of random instances using different number of modes, for every new instance, compute the distance to the closest shape in the training set

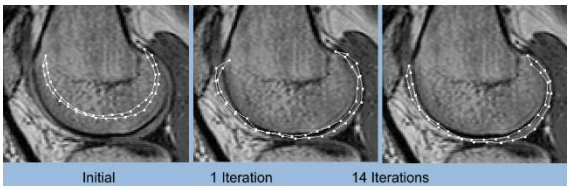
4.5.2 Active shape models (ASM)

Idea: 1) Start with SSM and image; 2) Initially place SSM in image; 3) Sample along SSM surface normal, find largest gradient, 4) Deform pose (scale, rotation, translation) and shape (PCA modes) of models to best fit to extracted image features (largest gradient)

- Initialization
    - Approximately align SSM with structure of interest
    - Many different methods, choose the one that suit
        - User interaction
        - Centroid alignment
        - Chamfer matching
        - Atlas registration
        - Global search on entire image (evolutionary algorithms)
    - The morphing is driven by **edge detection**, some methods are
        - Sobell
        - Canny - gives continuous edges
    - For the edge tracking, need normals
        - Points + connections give mesh -> so we know the neighbors of each point
        - **Cross product** of two vectors yields orthogonal vector -> this is the surface **normal**
    - Track the edge
        - Given point on SSM and normal at that point
        - We **sample** along the normal profile
        - Parameters: sampling range, sampling step size -> yields $2k + 1$ samples
        - Choose point with **largest feature values** (gradient)

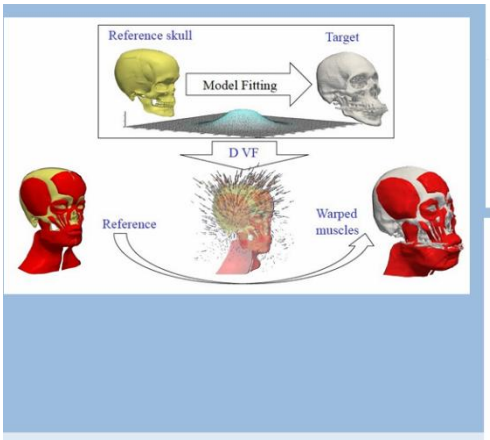
- Pose estimation: a rough alignment we do as a step for our morphing (a rigid transformation)
    - Optimize registration process according to certain cost function (e.g., mean Euclidean distance)
- Shape estimation
    - PCA-based deformation -> keep only k most important PCA modes (e.g., so that k modes account for 90% of variance)
    - Threshold b to allow for plausible shapes only, e.g., $|b_i| < 3\sigma_i$ (3 standard deviation capture more than 95% of variation in Gaussian distribution)
- Optimization schemes
    - Optimize pose and shape part separately or jointly
    - E.g., iterative method
        - Compute profile along normals
        - Adjust pose
        - Adjust shape
        - Iterate from beginning until convergence
- The complete ASM algorithm
    1. Examine a region of the image around each point $\mathbf{X}_i$ to find the best nearby match for the point $\mathbf{X}'_i$
    2. Update the parameters $(X_t, Y_t, s, \theta, \mathbf{b})$ to best fit the new found points $\mathbf{X}$
    3. Apply constraints to the parameters, $\mathbf{b}$, to ensure plausible shapes (eg limit so $|b_i| < 3\sqrt{\lambda_i}$).
    4. Repeat until convergence.

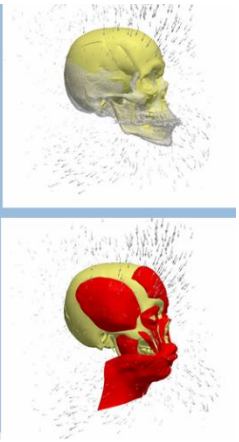
4.5.2.1 Advanced methods



- Multiresolution pyramid
- Handling noise by averaging along normal points
- Intensity profiles: methods
    - Samples along profile for the i-th image
    - Avoid intensity changes -> use gradient values
    - Normalize
    - Repeat for each data point
    - Assume Gaussian distribution
    - Quality of fit for new sample s: Mahalanobis distance
    - Pick the lowest values (higher probability)

Example: CMF surgery. Plan the surgery so that the face still looks the same after the operation; automatically segment the skull (is now a spin-off company) on the left we see an atlas of facial muscles





The model was enhanced with
- Facial muscles
- Muscle fibers (anisotropy of tissue properties)
- Predefined surgical approaches
- Data preprocessing (e.g., cropping)
- Simulation-specific (e.g., sliding contact areas)

- How to build model that include intensity variance? **Texture models**
  - Wrap each training sample to mean shape to obtain mean-free patch
  - Sample intensities from shape-normalized image to form a texture vector
  - Normalize intensity to eliminate lighting variations
  - Start from the beginning, same as SSM
  - We want to obtain a combined appearance model, which contains both shape and texture variations
  - We do a combined PCA
- Model fitting
  - Minimize difference between new image and the synthetic AAM image
  - Key step: estimate parameter update from current image sample
  - Difference vector
- The complete AAM algorithm

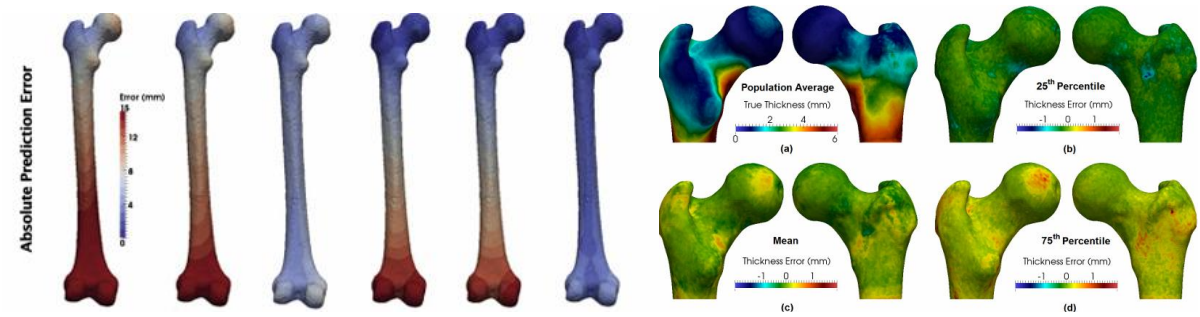
1. Project the texture sample into the texture model frame using $\mathbf{g}_s = T_{\mathbf{u}}^{-1}(\mathbf{g}_{im})$
2. evaluate the error vector, $\mathbf{r} = \mathbf{g}_s - \mathbf{g}_m$, and the current error, $E = |\mathbf{r}|^2$
3. compute the predicted displacements, $\delta\mathbf{p} = -\mathbf{R}\mathbf{r}(\mathbf{p})$
4. update the model parameters $\mathbf{p} \to \mathbf{p} + k\delta\mathbf{p}$, where initially $k = 1$,
5. calculate the new points, $\mathbf{X}'$ and model frame texture $\mathbf{g}'_m$
6. sample the image at the new points to obtain $\mathbf{g}'_{im}$
7. calculate a new error vector, $\mathbf{r}' = T_{\mathbf{u}'}^{-1}(\mathbf{g}'_{im}) - \mathbf{g}'_m$
8. if $|\mathbf{r}'|^2 < E$ then accept the new estimate, otherwise try at $k = 0.5$, $k = 0.25$ etc.
9. Repeat until convergence

- Applications of AAM
  - 3D simulation of 2D pictures
  - Do illumination changes, go through the eigenvectors (go from male to female), parameters: pose, shape, illumination

4.5.2.2 Conditional shape models

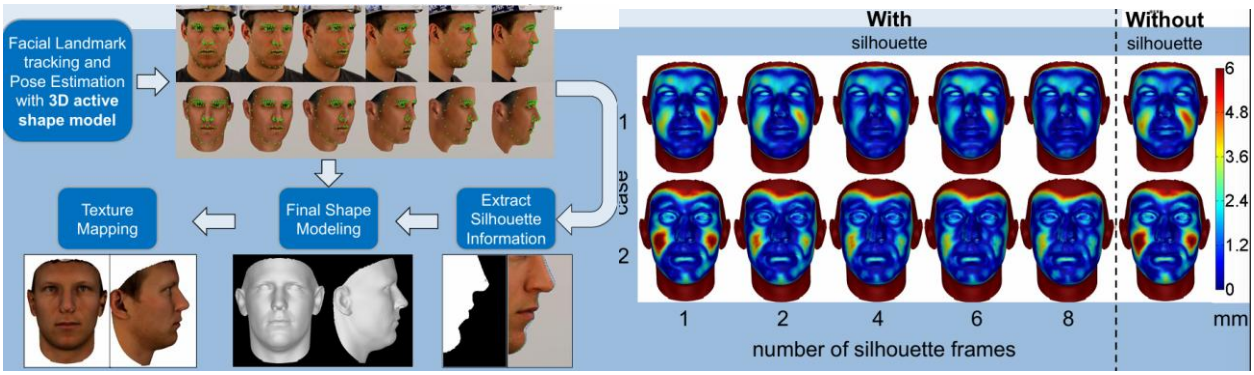
Goal: From surrogate variables to full shape surrogate variables can be length, angles, surface points, etc.

- Fast correlations of multi-organ segmentations - COSMOS
    - Organs have correlations in shape, position, etc. when a clinician corrects segmentation of one organ, the others are corrected as well
    - How: conditional statistical shape model
- Other study: combining shape models with supervised learning for bone shape and cortical thickness estimation (also later with FEM)
    - Red are the bigger errors



Example: high resolution patient-specific modeling with cortical thickness from clinical CT images.

- Cheeks are difficult to get the 3D structure of
- Adding silhouettes (edge detection) makes the results better

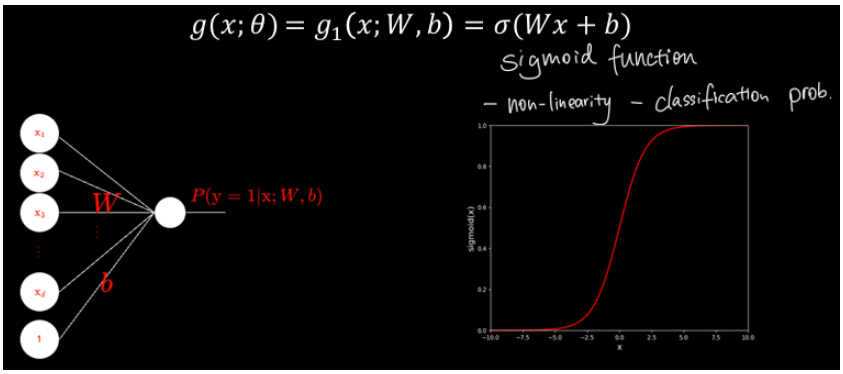


5. Convolutional neural network (CNN)

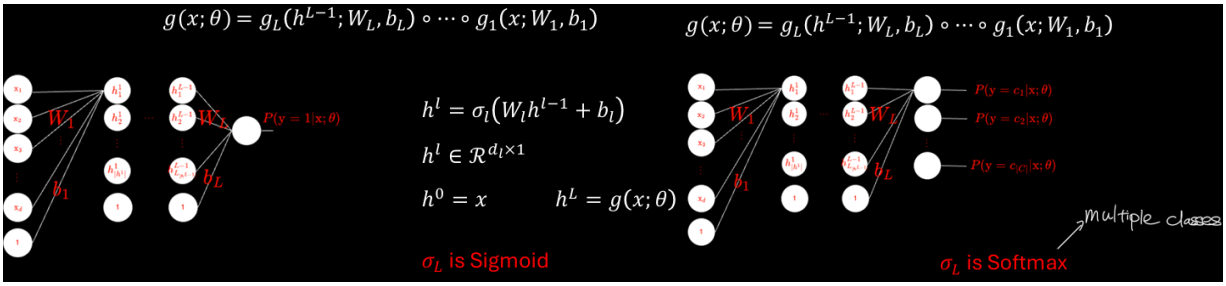
5.1 Neural networks

Idea: We have the input x (images), and the output y (some label); but we don't know the function leading to the labels. What we do have is a function g that labels training data; get f from g where θ is a set of learnable parameters, i.e., $y = f^*(x) \approx g(x;\theta)$. In training, we want to optimize our parameters $\theta^* = \arg\min_{\theta} \mathcal{L}\big(y, g(x;\theta)\big) + \mathcal{R}(\theta)$. We can also have g composed of multiple functions g_i, making the function more flexible.

- Perceptron: first "neural network" from 1958, can make binary classification



  - Each circle is a pixel, its value is the intensity, multiply everything with the weights W and the bias b, put everything into the big dot
  - Each layer has **non-linearity** so the whole thing is non-linear too
  - The output layer has softmax activation so that the probability sums up to 1
- Multi-layer perceptron



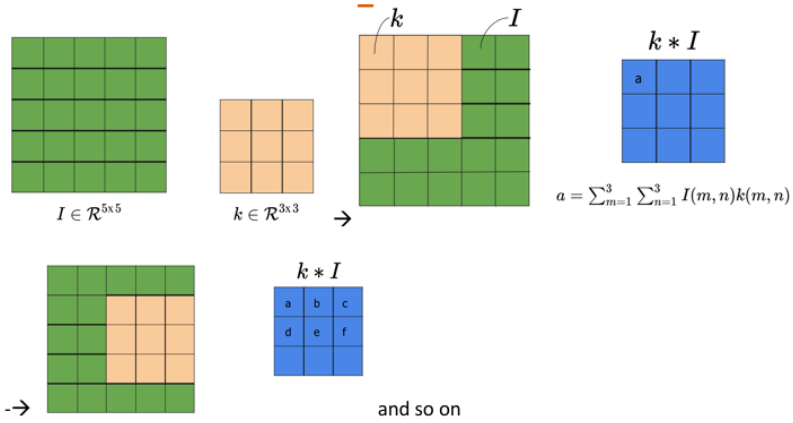
- Convolutional neural network
  - $h^l = \sigma_l(W_l * h^{l-1} + b_l)$
  - Different from the fully connected layer, which has similar equation as above, the W_l in CNN is **convolution**
- Why do we use CNN instead of FCN?
  - **FCN has too many parameters**: 64×64 image and first layer has 128 weights, then we have $64 \times 64 \times 128 = 524{,}288$ parameters
  - FCN does not exploit the hierarchy of local statistics
    - In FCN, the activation of any pixel is connected to every other pixel
    - Even though the local region would be enough to classify, but CNN does

    - **FCN has lack of translation invariance / equivariance**
- Applications
    - Image classification, say how severe an eye pathology is by looking at eye images
    - Age regression in brain images
    - Image segmentation: segment the brain tumor

5.2 Convolutional layers

5.2.1 Definition of convolution operator

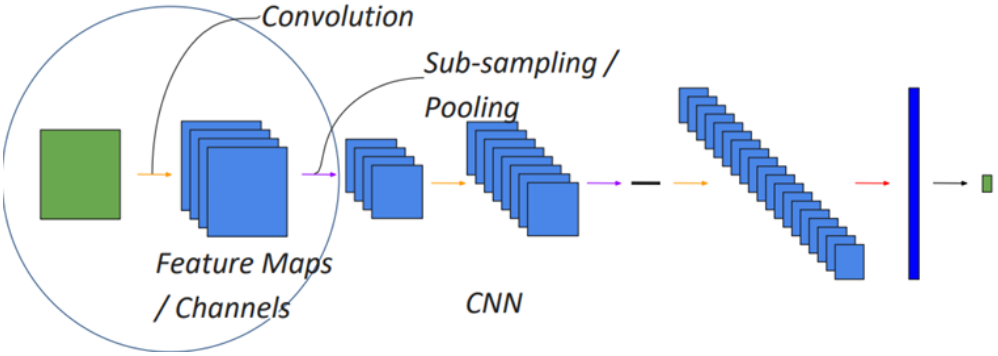
- Convolution is a mathematical operation on two functions
- 1D: consider two functions of a single variable, $f(t)$ and $g(t)$
    - Continuous: $(f * g)(t) = \int f(\tau)g(t-\tau)d\tau$
    - Discrete: $(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau)g(t-\tau)$
    - In words, convolution is the integral of the product of the two functions, after one of them is reversed and shifted
- 2D: the convolution of an image $I(x, y)$ and a convolutional kernel $k(x, y)$ is defined as $(I * k)(x, y) = \sum_m \sum_n I(m, n)k(x-m, y-n)$
- Convolution is **commutative**
    - The only reason why we need to reverse one of the functions before shifting it, not very useful in practice
- Convolution vs. cross-correlation
    - In practice, in many machine learning libraries, cross-correlation is implemented under the name of convolution
    - In the literature, convolution kernels are also referred to as convolutional weights and convolutional filters



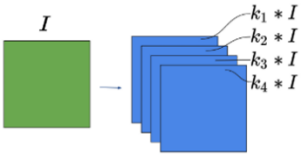
Note: The output image is **smaller** because the kernel's middle is never (in this case) at edge pixels; it does not need to be that the kernel size is equal the output size. The blue part is called feature / feature map / activation / activation map / output channel.

- Visually, cross correlation amounts to: moving the kernel successively across the image and performing a dot product at each location
- This would be the same for convolution, except that the kernel would be flipped 180 degrees before performing the raster-scan across the image and the successive dot products
- The convolution operation is a part of the convolutional layers in CNNs

- In MLPs, the layers are defined that every pixel is multiplied with the weights and biases are added
- In a convolutional layer, convolve every input layer with the kernel and sum it up



5.2.2 Convolution hyperparameters



- Number of output channels
  - This refers to the number of independent convolutions
  - On the right side, the number of output channels is 4
- Number of input channels
  - 1 for gray scale images
  - 3 for RGB images
  - N in case of multi-modal medical images
  - The number of output channels at layer L is the number of input channels for layer $L + 1$
  - The convolution kernel always has the same depth as the number of input channels for that convolutional layer
- Kernel size
  - For a convolutional layer with N_{in} input channels, $k_1 \times k_2$ kernel size and N_{out} output channels, we have N_{out} independent convolutional kernels, each of size $N_{in} \times k_1 \times k_2$
  - The weight matrix (parameters) corresponding to this convolution layer is of size $N_{in} \times k_1 \times k_2 \times N_{out}$
  - There are also N_{out} bias parameters
  - **Total amount of parameters:** $N_{in} \times k_1 \times k_2 \times N_{out} \times 2$

5.2.2.1 Padding

Goal: to deal with boundary pixels

- Boundary pixels are ignored for usual convolution, which leads to the output feature map to be smaller in size than the input image
  - *Example of calculation*
    - An image is 5×5, kernel is 3×3, output is 3×3
    - If input is $m_i \times n_i$, kernel is $k_x \times k_y$, output is $m_{i+1} \times n_{i+1}$, where $m_{i+1} = m_i - k_x + 1$ and $n_{i+1} = n_i - k_y + 1$

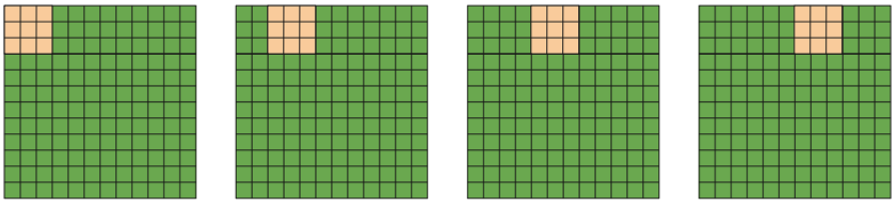
- Padding is to pad the input boundaries, such that convolution operation can be carried out on boundary pixels as well. Usually, padding is down with zeros, but one may also use symmetric padding in some cases
    - Output size = input size

5.2.2.2 Stride

- How far we shift the kernel across the image

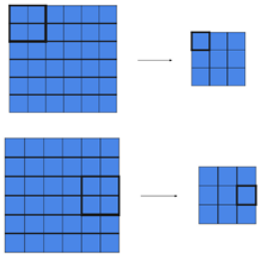
We can also skip s pixels every time (stride = s).

For example, s = 2:



- When stride > 1, it leads to a reduction in the size of output as compared to the input
- Output size: with stride to be s_x and s_y in the two image directions, we have further reduction in output size
    - $m_{i+1} = \frac{m_i - k_x}{s_x} + 1$
    - $n_{i+1} = \frac{n_i - k_y}{s_y} + 1$
- *Example*: we want to have the input and output to have the same size, we need same padding and stride with 1

5.2.2.3 Pooling



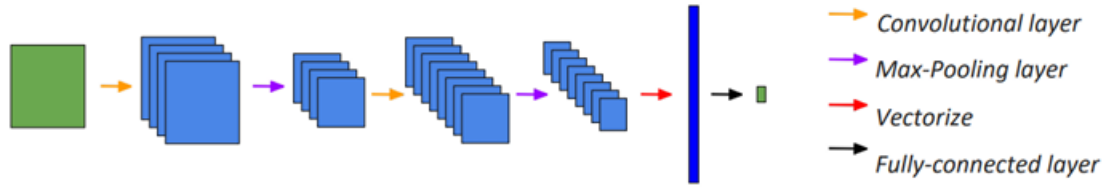
Idea: Pooling is an operation that enables "pooling" information in a neighborhood. Applied to each channel separately.

- Types of pooling
    - Linear
        - Average pooling (average value of the region)
    - Non-linear
        - Max-pooling (max value of the region), usually used
        - Min-pooling (min value of the region)
- Pooling hyperparameters
    - Size of pooling kernel
    - Stride
    - Suppose we have pooling stride = size of pooling kernel size = 2. This leads to **halving of the input** via the pooling operation. For higher kernel size and pooling stride, we get more dimension reduction

5.3 Image classification example using CNN

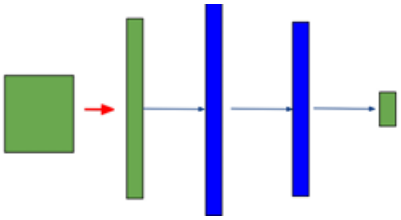
Idea: the dimensions are reduced so the fully-connected layer does not introduce too many parameters

- Input: $I \in \mathbb{R}^{28\times 28}$
- Conv. layer with 3×3 kernel, stride 1, same padding, number of channels = 4: $\mathbb{R}^{28\times 28\times 4}$
  - $N_{in} \times k_x \times k_y \times N_{out} + N_{out} = 1 \cdot 3 \cdot 3 \cdot 4 + 4 = 40$
- Max pooling layer with 2×2 kernel, stride 2: $\mathbb{R}^{14\times 14\times 4}$
  - No added parameter
- Conv. layer with 3×3 kernel, stride 1, same padding, number of channels = 3: $\mathbb{R}^{14\times 14\times 8}$
  - $N_{in} \times k_x \times k_y \times N_{out} + N_{out} = 4 \cdot 3 \cdot 3 \cdot 8 + 8 = 296$
- Max pooling layer with 2×2 kernel, stride 2: $\mathbb{R}^{7\times 7\times 8}$
  - No added parameter
- Vectorize: $\mathbb{R}^{392\times 1}$
  - No added parameter
- Fully-connected layer with 10 output units: $\mathbb{R}^{10\times 1}$
  - $392 \times 10 \times 1 = 3920$
- **Total number of parameters: $40 + 296 + 3920 = 4256$**



The dimensions are reduced so the fully-connected layer does not introduce too many parameters

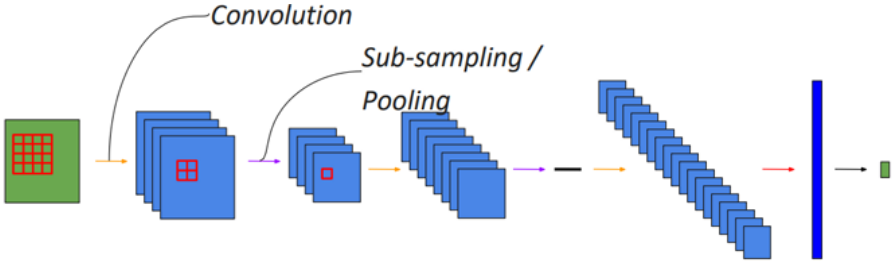
5.4 Why CNN for image analysis



Number of parameters in MLP is large

- Naïve approach: transform the image into a 1D vector
- MLPs consist of fully connected (FC) layers
- FC layers mean **large number of parameters**
- *Example*
  - An image $x \in \mathbb{R}^{M\times N}$
  - Vectorize to be $h_0 \in \mathbb{R}^{MN}$
  - The first hidden layer $h_1 \in \mathbb{R}^{d_1}$
  - Recall that $h_1 = W_1 h_0 + b_1$, $W_1 \in \mathbb{R}^{(d_1)\times(MN)}$, $b_1 \in \mathbb{R}^{d_1}$
  - If we have $M = 64$, $N = 64$, and $d_1 = 32$
  - Then we have $32 \times 64 \times 64 + 32 = 131104$ parameters in the first layer

Images are composed of a hierarchy of local statistics



- The **receptive field** is the region in the **input image** that a pixel in the **output feature map is affected by**. Note that we consider the **input image** here, not the input of a particular layer
- In the initial layers, the **CNN sees only a small portion** of the image and encodes local features, like **edges and corners**. As the network **depth increases**, the receptive field increases and more **global information is encoded**

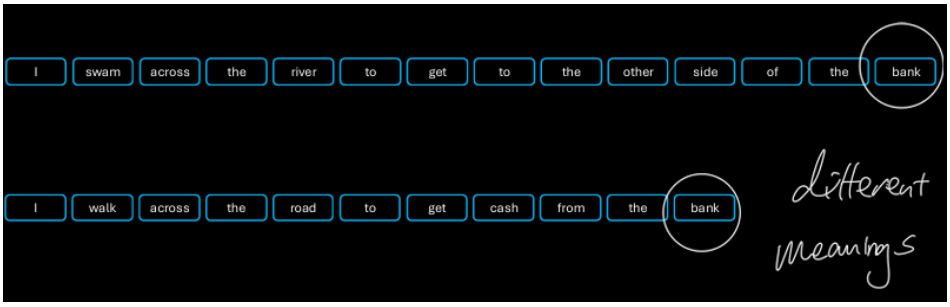
Translation equivalence and <u>approximate</u> translation invariance (IMPORTANT)

- A function f is **equivariant** with respect to a transformation T if $f\big(T(x)\big) = T(f(x))$
- A function f is **invariant** with respect to a transformation T is $f\big(T(x)\big) = f(x)$
- **Translation equivariance**
    - Required for applications like segmentation, object localization
    - In some applications, we are only interested in whether some features is present or not, but not in its exact location within the image
        - The pooling operation in image classification CNNs leads to partial translation invariance
- **Invariance to other transformations**
    - Invariance or equivariance to some other types of transformations may also be desirable, e.g., rotation equivariant
    - CNNs do not natively offer such properties
    - One way to implicitly encode equivariance is via data augmentation

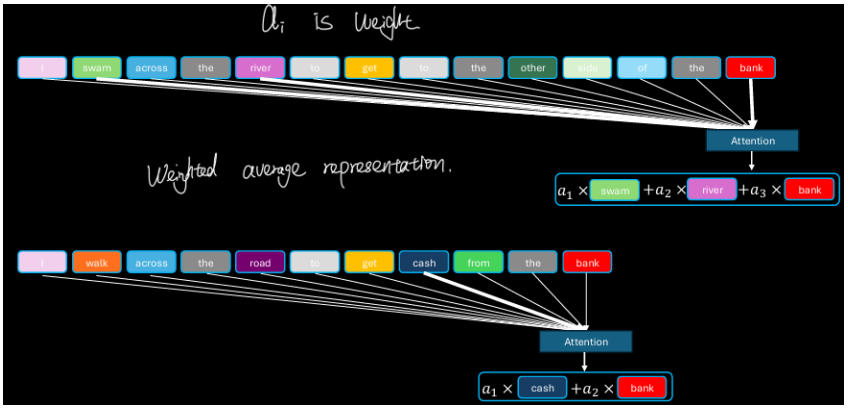
6. Transformers

Attention mechanism is the fundamental block of transformers. Now transformers are everywhere, including computer vision, natural language processing, reinforcement learning, speech recognition, translation, and graphs / science.

6.1 Motivation

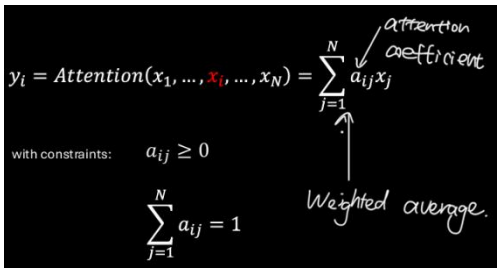


- One word (bank), two different meanings
- Context makes it possible to differentiate, some words are more important than others
  - Cash
  - Swam & river
- How would a standard network processing do it?
  - Every word has its own color, bank is initially colored the same
  - Find parameters from the words
  - Fixed weights, bank has the same representation
- Attention
  - **Some words get more weight** (attention)
  - Bank will have **two different meanings**

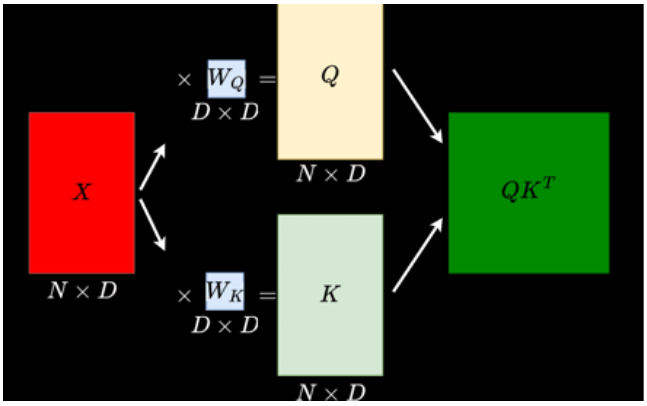


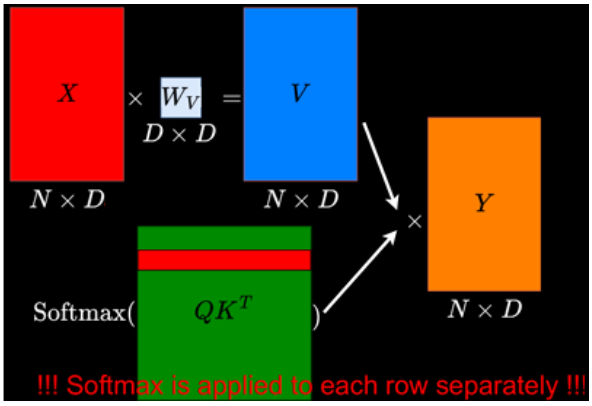
6.2 Self-attention

- Attention should be a function of all the words and gives then the input tokens' weights with whose we calculate the output tokens
- If a word gets more attention, then some others need to get less attention, i.e., sum of $a_{ij} = 1$

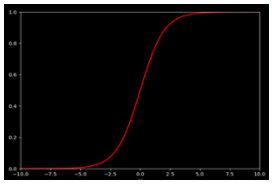


- Some terminology on information retrieval terminology
    - **Key**: movies on Netflix with their characteristic - genre, release year, leading actor …
    - **Queries**: some clues about what we want to watch, e.g., sci-fi movie after 2000 with Leonardo DiCaprio
    - **Value**:
        - This is **hard attention**, we go through all the options and see which key matches best to our query
        - In transformers, **generalize hard attention to soft attention**
        - Use **continuous variables** to **measure the degree of match** between **query** and **keys** (would get back maybe a mixture of 30% Titanic and 70% - which would not be possible in this Netflix example of course)
            - Weight the contribution of value vectors on the output using these variable
- To calculate the attention coefficient a_{ij}, we measure the **similarity** (do that with the cross product) and collect in a matrix, use the **softmax** function (to every row separately), since then all outputs are positive and they **sum up to 1**
- Any issues with $Y = \text{softmax}(XX^\top)X$?
    - **No capacity to learn from data**
    - Each feature value within a token plays an **equal role** in determining the attention coefficients, we want a more flexible model
- Add learnable parameters to help select more useful features
    - $\tilde{X} = XU, U \in \mathbb{R}^{D\times D}$
    - $Y = \text{softmax}\left(\tilde{X}\tilde{X}^\top\right)\tilde{X} = \text{softmax}(XUU^\top X^\top)XU$
    - But we **still have problem**, the input to the softmax function is **symmetric**
        - Attention from bank to swam is much stronger than the other way around
- **Solution**: add different learnable parameters to Key, Query, Value $Y = \text{softmax}(QK^\top)V$
    - $K = XW_K, W_K \in \mathbb{R}^{D\times D_K}; Q = XW_Q, W_Q \in \mathbb{R}^{D\times D_Q}; V = XW_V, W_V \in \mathbb{R}^{D\times D_V}$



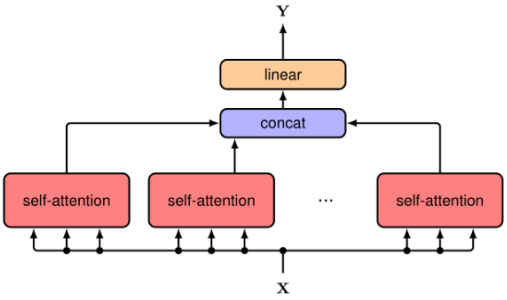


6.3 Scaled self-attention



- $Y = \text{softmax}\left(\frac{QK^\top}{\sqrt{D_K}}\right)V$
- One final refinement to the attention function
- The gradient of the softmax function become **exponentially small** for small and large inputs. When the **gradient is too small**, we have **no learning**, so we want to trick the $QK^\top$ output so it falls into the region which is not flat (blue arrow)
- We assume that Q and K follow a **Gaussian**; the summed variance of $QK^\top$ is then D_K
- **Divide** the softmax input by the **standard deviation** (root of variance)

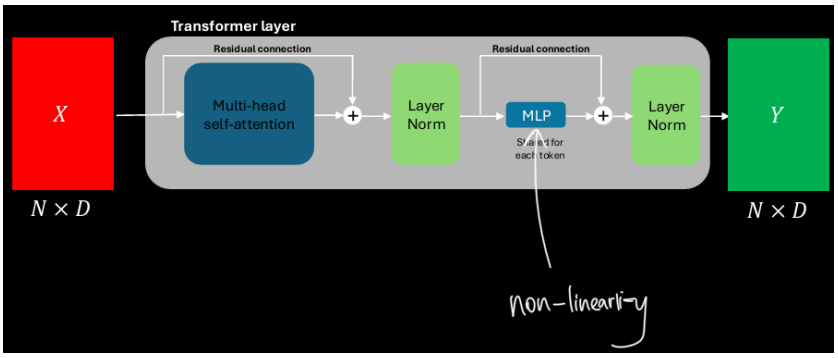
6.4 Multi-head self-attention



- Remember CNN: we have convolution between the layers
  - There might be different useful features to extract at each layer
  - So we use multiple filters
- **Attention**: there might be multiple patterns that require attention, use multiple attention heads (multi-head self-attention)
- Above we saw the scaled "single-head" self-attention
- For multi-head, we calculate the attention for every head, once we have them, we concatenate them
  - $H_h = \text{Attention}(Q_h, K_h, V_h), \forall h \in [1, H]$
  - $\text{Attention}(Q_h, K_h, V_h) = \text{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{D_{K_h}}}\right)V_h$
    - $K = XW_{K_h}; Q = XW_{Q_h}; V = XW_{V_h}$
  - $Y = \text{concat}[H_1, H_2, \cdots, H_H]W_o$
    - $\text{concat}[H_1, H_2, \cdots, H_H] \in \mathbb{R}^{N \times HD_v}; W_o \in \mathbb{R}^{HD_v \times D}$
    - W_o is a suitable matrix with weighting parameters so we can learn

6.5 Transformer layers

Goal: We want to transform x to new representation y without changing the dimensions.

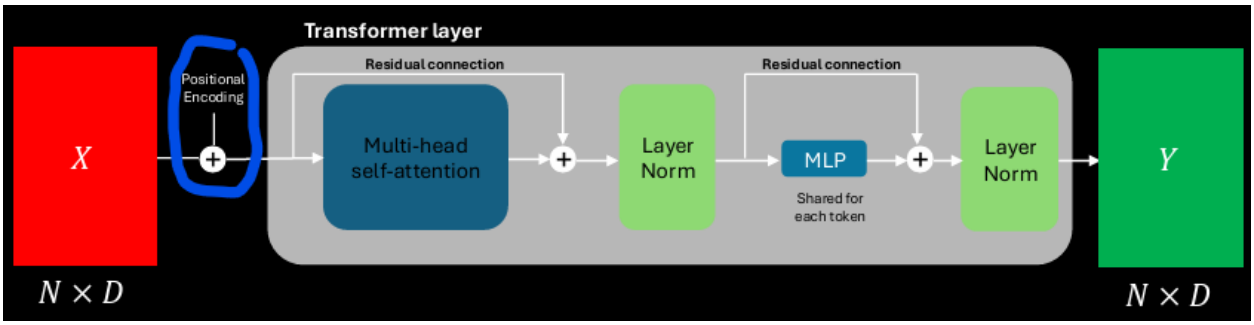


-First a multi-head self-attention layer (attention weights and non-linearity); but it is still a **linear combination**, so **not as flexible as it could be**

-Residual connection so we can pass the input data to later stages

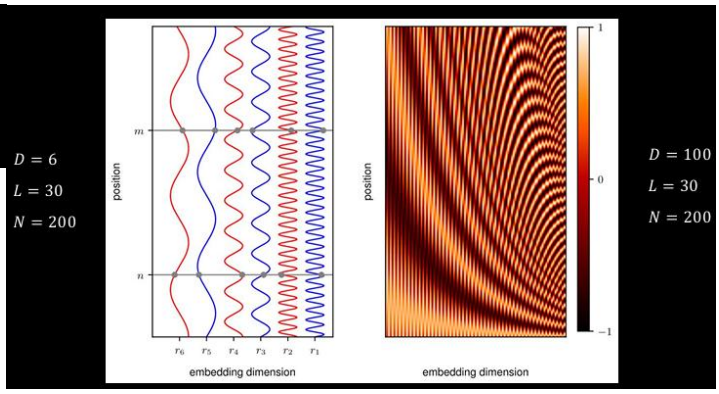
- Sum those matrices
- Use layer normalization to increase the time efficiency
- Adding a **standard multi-layer perceptron** to **overcome the transformer shortcomings** (add **non-linearity**)
- Another residual connection with summation
- Another layer normalization
- N is the number of words in the sequence and D is dimension of every word embedding
- Hyperparameters: the **amount of MLP layers is the most crucial**

6.6 Positional encoding



- Transformers are **equivariant with respect to input permutations** $f(P(X)) = P(f(X))$, remember CNNs are translation equivariant, i.e., $f(T(x)) = T(f(x))$
  - This is because of the weight matrix W
  - "I bought an apple watch" vs. "Watch an apple I bought" same for transformers
  - We need to have **token order** (positional information)
- We add the positional information with r at every beginning
  - $\tilde{x}_i = x_i + r_i, r_i, x_i \in \mathbb{R}^{D\times 1}$
  - An ideal positional encoding should
    - Provide a **unique representation for each position**
    - Be bounded
    - Generalize to longer sequences, if too short, then cannot adopt to longer ones
    - Encode the **relative position of tokens**
- An example of positional encoding

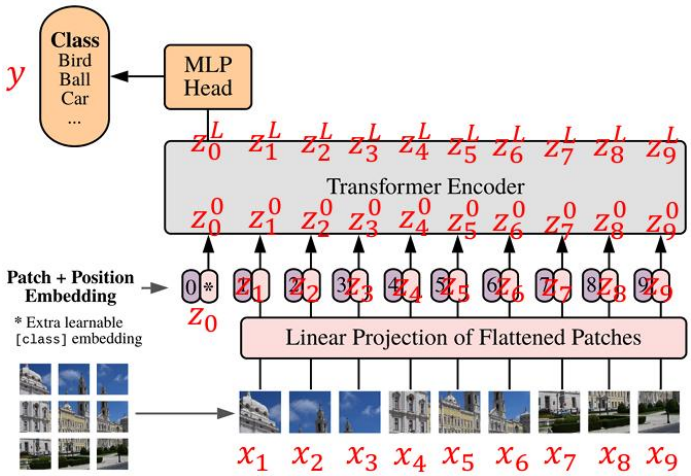
$$r_{ni} = \begin{cases} \sin\left(\frac{n}{L^{i/D}}\right), if\ i\ is\ even \\ \cos\left(\frac{n}{L^{(i-1)/D}}\right), if\ i\ is\ odd \end{cases}$$



<u>Note</u>: Positional encoding is added at the very beginning of the transformer layer. Positional encoding can also be learned, the example is a deterministic one.

6.7 Transformers for images

Idea: In speech transformers, the words are tokens. Analogically, in pictures, we first need to create tokens, divide the image into patches, then flatten the patches.



- $x_i \in \mathbb{R}^{P\times P\times C}, \forall i \in [1, N], N = HW/P^2$
- The additional embedding z_0 is a class embedding which is learnable
- Then add the positional embedding $z_i^0 = z_i + E_{pos}^i, E_{pos}^i \in \mathbb{R}^{1\times D}$
- $z_i = \text{flatten}(x_i)E, E \in \mathbb{R}^{(P^2C)\times D}, z_i \in \mathbb{R}^{1\times D}$
- Then concatenate them to get $X_0 = [z_0^0, z_1^0, \cdots, z_9^0]$, put that into the transformer encoder
- The output $X_L = \text{TransformerEncoder}([z_0^L, z_1^L, \cdots, z_9^L])$ where L is the amount of layers
- Connect to the output y with a MLP head

CNNs vs Transformers for images

- In CNNs, **locality** and **two-dimensional neighborhood** are strong inductive **biases** for vision tasks
- In vision transformers (ViT)
  - Only **MLP** layers are **local** while **self-attention** is **global**
  - **Patching** is the only **2-dimensional inductive bias**; it has to learn **geometrical properties** from scratch
  - Generally, requires **more training data** than a comparable CNN

7. Pixel-wise predictions with deep neural networks

7.1 Machine learning concepts

7.1.1 Learning

$y \approx f(x;\theta)$, parameterized mapping that goes from features to labels. It an approximation. Non-parametric mappings are also possible.

- Determine the "best" parameters using examples, i.e., **training set**, $\mathcal{D}_{tr} = \{(x_n, y_n)\}_{n=1}^{N}$
- y_i's are called **ground truth labels**
- Minimize a **loss** between predictions and ground truth labels, i.e., $L(y, f(x;\theta))$
- Training set loss is $\mathcal{L}(\mathcal{D}_{tr};\theta) = \frac{1}{N}\sum_{n=1}^{N} L\big(y_n, f(x_n;\theta)\big)$
- The **best parameters** are the ones that minimize it on $\mathcal{D}_{tr}$, i.e., $\theta^* = \arg\min_{\theta} \mathcal{L}(\mathcal{D}_{tr};\theta) = \arg\min_{\theta} \frac{1}{N}\big\{L\big(y_1, f(x_1;\theta)\big) + \cdots + L\big(y_N, f(x_N;\theta)\big)\big\}$, generalize to $\mathcal{D}_{val}$

Features	**Labels**
- x - **observed** at inference - real, categorical - D-dimensional - classification: image intensities - segmentation: image intensities at / around a pixel	- y - **not observed** at inference - real, categorical - d-dimensional - classification: class of the image - segmentation: label at a pixel

7.1.2 Hyperparameter and the validation set

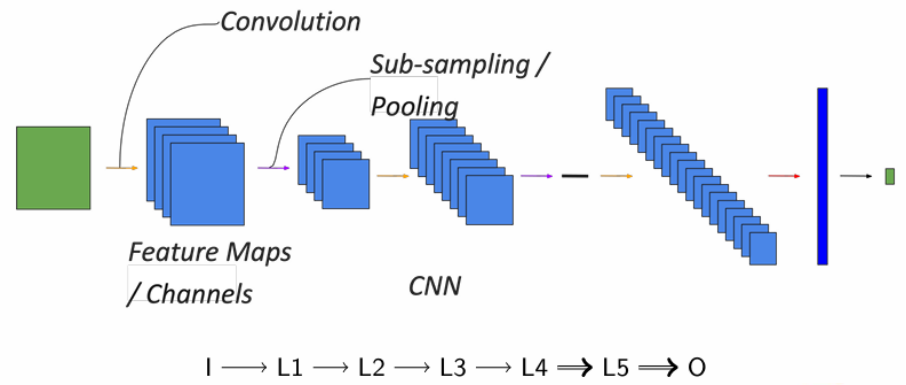
- Most algorithms have hyperparameters, i.e., ϕ, $y \approx f(x;\theta) \Rightarrow y \approx f(x;\theta,\phi)$ and $\mathcal{L}(\mathcal{D};\theta) \Rightarrow \mathcal{L}(\mathcal{D};\theta,\phi)$
- Deep neural networks
  - Network architecture
  - Stopping point in the optimization
  - Learning rate in the optimization
- **Validation** set $\mathcal{D}_{val}$ helps determine the hyperparameters
  - $\phi^* = \arg\min_{\phi} \mathcal{L}(\mathcal{D}_{val};\theta^*,\phi)$
  - Where $\theta^* = \arg\min_{\theta} \mathcal{L}(\mathcal{D}_{tr};\theta)$

7.1.3 Prediction / inference

- For any new sample x, we simply evaluate the mapping with the optimal parameters, remember that in general $y \neq f(x;\theta^*)$
- Prediction error is computed on a hold-out **test set**, either with loss function $\frac{1}{T}\sum_{t=1}^{T} L\big(y_t, f(x_t;\theta^*)\big)$ or some other functions (may not be differentiable)

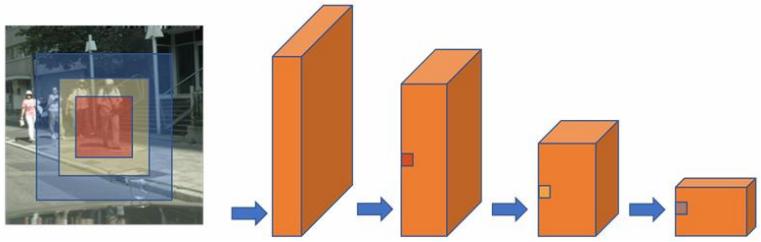
7.1.4 CNN and transformers recap

CNN



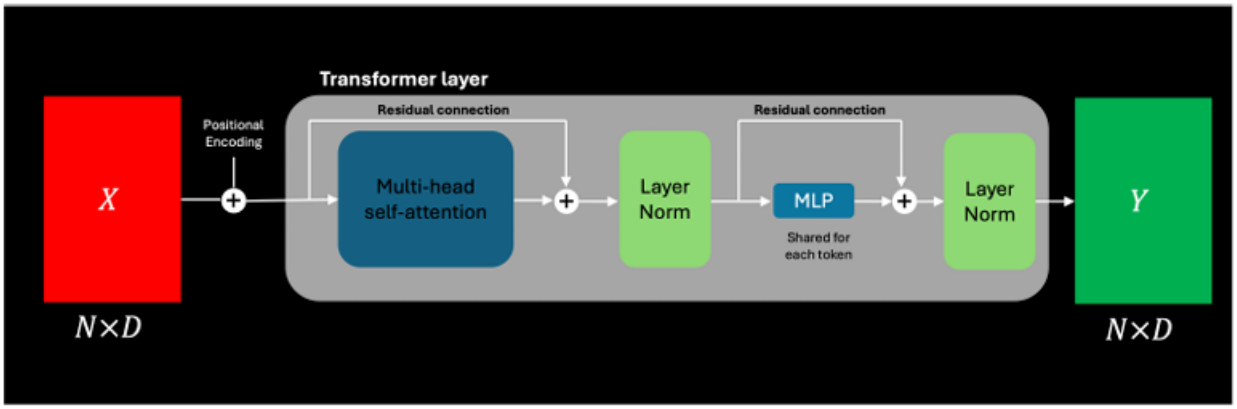
- I is the input image, $L\#$ are intermediate layers, and $O \in [0,1]$ is the output
- $\rightarrow$ is convolution, non-linear activation and pooling
- $\Rightarrow$ is fully connected layer

Receptive fields



- Size of the input pattern changes with respect to the receptive field
- The receptive field changes due to the layer and neuron
- The size of the input pattern changes as well
- Neurons at **deeper layers** have a **large contextual information** but perhaps **less local details**

Transformers



$$Y = \text{Softmax}\left(\frac{QK^T}{\sqrt{D_K}}\right)V$$

7.2 Segmentation

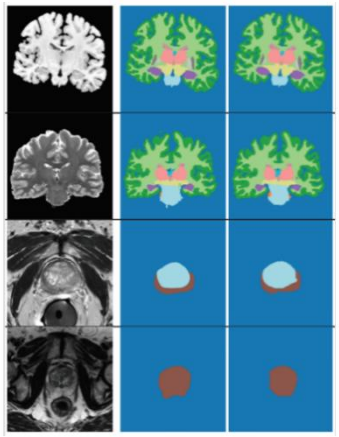
Task: Assigning labels to each pixel, indicating the structure the pixel belongs to

7.2.1 Data set

- Training set: input images and the **corresponding segmentation mask**, applied manually
- Need a **large number of images**, also need to have a **validation** set and a **test** set
  - Depends on how to segment, but the more the better

    - Depends on variability of structures / background / intensity variation
- Variations and challenges in reality
    - Ambiguity in labeling
    - Missing data in multimodal cases - a very common problem
        - May need to impute missing data during training or testing, hard on images
    - Images may be coming from different "domains", e.g., centers, scanners, sequences
        - Can change the contrast between different tissue a lot
        - Changes can happen during training, which is okay
        - Can also change during inference, which is **not good**

7.2.2 Cost function



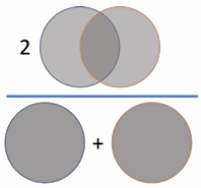
We want a method that can **compare** the true labels and the prediction, i.e., **cost function**, mean square error as an example

7.2.2.1 (Binary) cross entropy

- **Binary classification loss** for a single sample is $BCE\big(y, f(x;\theta)\big) = -1(y=1)\log f(x;\theta) - 1(y=0)\log\big(1-f(x;\theta)\big)$, assuming $f(x;\theta)$ is the probability of being of class $y=1$
    - $L\big(y, f(x;\theta)\big) = \sum_{r=1}^{D} BCE\big(y(r), f(x;\theta)(r)\big)$
    - $y(r)$ and $f(x;\theta)(r)$ are the ground truth labels and predictions at pixel r
- **Multi-class case** $CE\big(y, f(x;\theta)\big) = -\sum_k p(y=k)\log f_k(x;\theta)$
    - $L\big(y, f(x;\theta)\big) = \sum_{r=1}^{D} CE\big(y(r), f(x;\theta)(r)\big)$

Challenge with (binary) cross entropy: When there are small structures, either compared to background or other classes (in the multi-class case), the **losses associated with small structures are dominated by losses associated with larger structures**

7.2.2.2 Sorenson-Dice coefficient



Idea: Weight the error compared to the appearance of the values

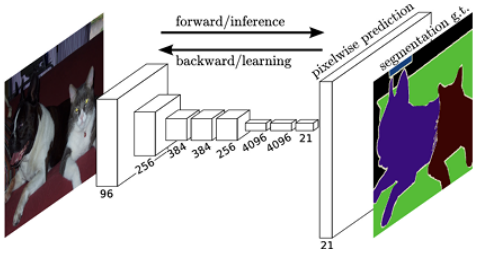
- $DSC = \frac{2|A \cap B|}{|A|+|B|}$
- Used for evaluating quality of segmentation predictions
    - $DSC = 1$ when the sets perfectly overlap
    - $DSC = 0$ when there is no overlap
- *Challenge with DSC*: It is **NOT differentiable**
    - When using **probability**, we can make it **differentiable**
    - Binary segmentation $L\big(y, f(x;\theta)\big) = \frac{2\sum_{r=1}^{D} y(r) f(x;\theta)(r)}{\sum_{r=1}^{D} y(r)^2 + \sum_{r=1}^{D} f(x;\theta)(r)^2}$
    - Multi-class segmentation $L\big(y, f(x;\theta)\big) = \sum_{k=1} \frac{2\sum_{r=1}^{D} p(y(r)=k) f_k(x;\theta)(r)}{\sum_{r=1}^{D} p(y(r)=k)^2 + \sum_{r=1}^{D} f_k(x;\theta)(r)^2}$

- There is **no contribution from wrongly classified** background pixels
- This is direct conversion from the dice coefficient, **not a loss yet**

7.2.3 Architectures

- Assume we have an image of size $D_1 \times D_2$
- Assume we have a binary segmentation problem
- Input dimension $D_1 \times D_2$
- Output dimension $D_1 \times D_2 \times k$
- Output indicates the probability a pixel belongs to each class

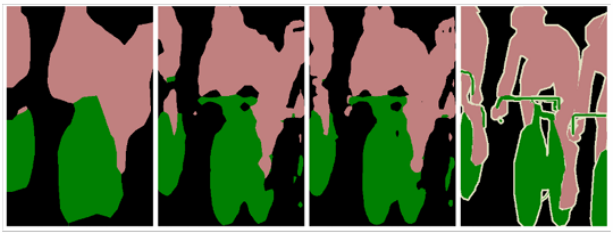
7.2.3.1 Convolutional neural network

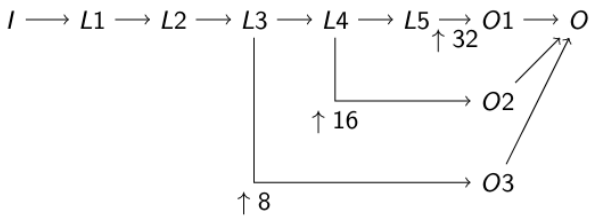


- For image classification is like
- $I \to L1 \to L2 \to L3 \to L4 \Rightarrow L5 \Rightarrow O$
- I is the input image, $L\#$ are intermediate layers, $O \in [0,1]$ is the output
- $\to$ is convolution, non-linear activation and pooling
- $\Rightarrow$ is fully connected layer
- Note that we would like an output of image size, channel dimension reduces from $L1$ to $L4$
  - $I \to L1 \to L2 \to L3 \to L4 \to L5 \to^{\uparrow s} O$
  - We upsample within the network and convert an intermediate layer $L5$ to required images size
  - The upsampling factor s depends on the size of the channel at $L5$
- $I \to 32 \to 128 \to 256 \to^{\uparrow s} 128 \to 32 \to O$
  - $\to$: 1-padding with 3×3 convolutions, ReLU activations and pooling
  - $\to$: 1-padding with 3×3 convolutions, ReLU activations
  - $\to$: 1-padding with 3×3 convolutions, sigmoid activations (binary classification)
    - Sigmoid activation ensures the output is between 0 and 1, i.e., $O(r) \in [0,1]$, for all r
  - For multi-class segmentation, the O has multiple channels in this case
    - $I \to L1 \to L2 \to L3 \to L4 \to^{\uparrow s} L5 \to O$
    - Each channel is an image indicating probabilities of each pixel belonging to one class
    - Use softmax activation $f_k(x;\theta)(r) = O(r) = \frac{\exp\{a_k(r)\}}{\sum_j \exp\{a_j(r)\}}$
      - $a_j(r)$ is the activation in channel j as pixel r
- *Challenge*: There has been so many pooling operations that details are lost, how to use both contextual information and retain high-resolution information

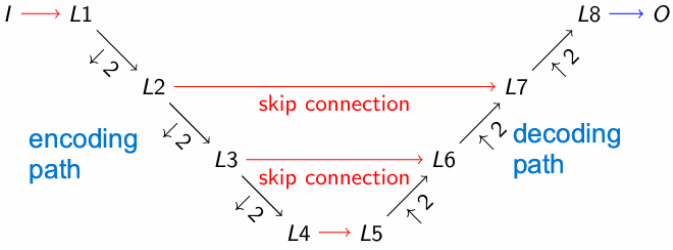
7.3.2.2 Fully connected layers with hierarchical links

Idea: Combination from different scales to retain high-resolution details in segmentation maps, improved details through aggregating information from different layers





7.3.2.3 UNet



- Skip connections **retain high-resolution information**
- They consist of convolutions and results get concatenated at the target
- Number of levels and layer compositions here are **arbitrary**, can change
- $\rightarrow_{\downarrow 2}$: convolution (same), non-linear activation, pooling
- $\rightarrow_{\uparrow 2}$: bilinear upsampling, convolution (same), non-linear activation OR
- $\rightarrow_{\uparrow 2}$: transposed convolution, non-linear activation
- $\rightarrow$: convolution (same), non-linear activation
- $\rightarrow$: convolution (same), sigmoid or softmax activation

7.3 Restoration

Task: Removing noise from an image, image super resolution, bias field removal, etc.

7.3.1 Data set

- We need input images and "ground truths," i.e., less noisy images
- Size of the data set - depends on the problem

Challenges

- **Missing data** in **multimodal** cases
- Images may be coming from **different "domains,"** e.g., centers, scanners, sequences
- Ground truth labels **may not be available** for all questions
  - Difficult to acquire additional ground truth images
  - Unethical to acquire additional ground truth images

7.3.2 Cost function

7.3.2.1 Basic cost functions

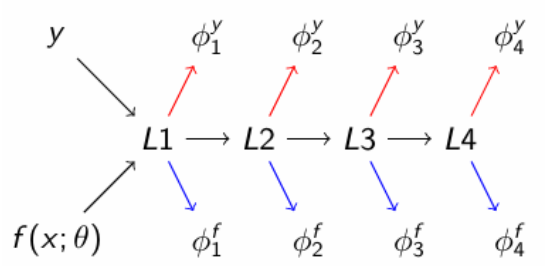
- Compare output prediction with the ground truth (pixel per pixel)
  - Mean-squared error: $MSE = \sum_r ||y(r) - f(x;\theta)(r)||_2^2$
  - Mean-absolute error: $MAE = \sum_r |y(r) - f(x;\theta)(r)|$
  - Normalized mean-squared error: $NMSE = \frac{MSE}{\sum_r ||y(r)||_2^2}$
  - Peak signal to noise ratio: $PSNR = 10 \log_{10} \frac{\max(y)^2}{MSE}$
  - Structural similarity index measure: $SSIM(a,b) = \frac{(2\mu_a\mu_b + c_1)(2\sigma_{ab} + c_2)}{(\mu_a^2 + \mu_b^2 + c_1)(\sigma_a^2 + \sigma_b^2 + c_2)}$

- Computed over many patches a and b coming from y and $f(x;\theta)$, combination of local luminance, contrast and structure comparisons

Limitations of basic cost functions

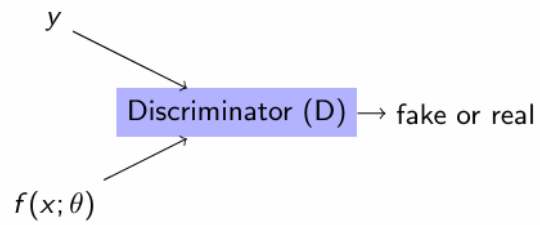
- MSE or derived loss may not capture differences between images that makes images look "perceptually" similar
- While it is unclear what is means to capture "perceptual differences," one can use neural networks to do this - advanced cost functions

7.3.2.2 Perceptual loss



-The CNN can be much deeper
-Compare the prediction and truth at every layer (also with MSE), find the perceptual loss
-Distance between "deep features" measure differences in contextual information going beyond simple pixel-wise difference
-Perceptual loss is $\sum_j MSE(\phi_j^y, \phi_j^f)$

7.3.2.3 Adversarial loss

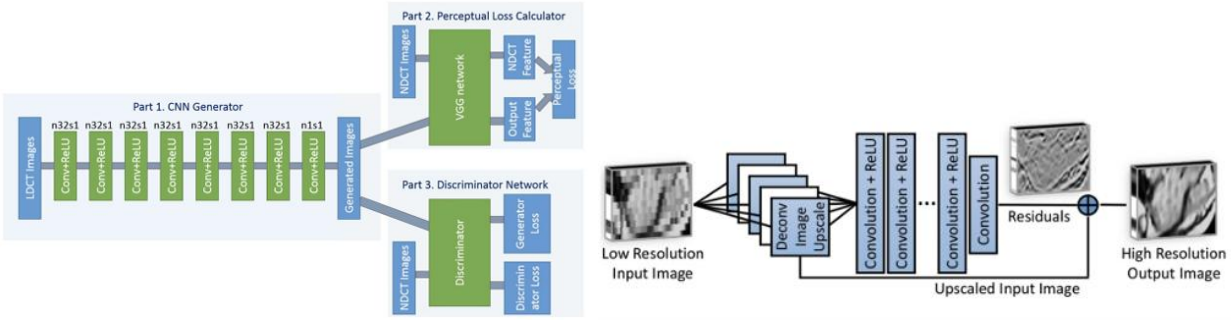


- Another way to capture perceptual differences
- Distributional distance, not between two samples but over sets of samples
- Discriminator identifies whether the input image is real or generated by the "generator" network $f(\cdot;\theta)$
- Optimized for the generator network

- $\min_{\theta} \max_{\psi} \mathbb{E}_{y\sim p(y)}[\log D(y;\psi)] + \mathbb{E}_{x\sim p(x)}[\log(1 - D(f(x;\theta);\psi))]$

7.3.3 Architectures

- Most works use Fully Convolutional Networks (FCN) or UNet structure



- Residual architectures: to make resolution better (one of the easier tasks)
  - Restorative information is added **as a residual to the original** or **naively restored image**. In this case, residuals are added to **linearly upsampled** image to restore high resolution details. Well established and used in other restoration problems

7.4 Synthesis

Task: Synthesizing one image from another one, synthesizing a **target** image from a **source** image

- Data imputation
- Reducing need for irradiation
- Reducing need for additional imaging

7.4.1 Data set

- Need input pictures and ground truth (MRI, and corresponding CT images)
- Data size: no good answer, generally, **the more the better**
  - Difficult to generalize from specific applications and projects. Some synthesis problems are possibly not solvable, e.g., generic MRI to PER synthesis

Challenges

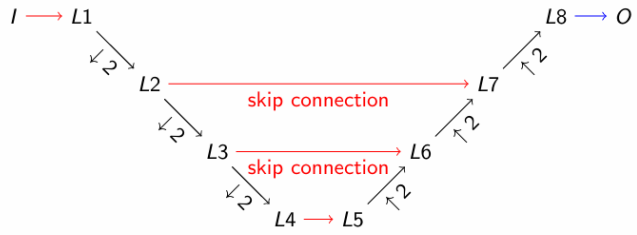
- Ambiguity in labeling
- Missing data in multimodal cases
- Images may be coming from different "domains," e.g., centers, scanners, sequences
- Labels and features **may not** be paired, **big challenge**

7.4.2 Cost function

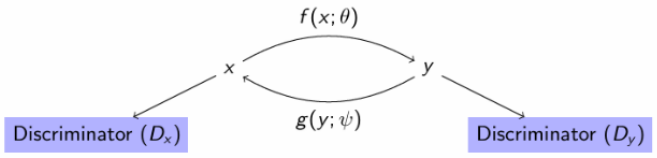
- Similar cost functions as restoration
- MAE, MSE, NMSE, PSNR, SSIM, perceptual distance, adversarial loss
- Distributional losses, e.g., adversarial loss, are particularly useful for unpaired datasets

7.4.3 Architectures

- Good old UNet is used quite often for the sample problem
  - Can split out any valid CT picture, but maybe not from the same person
- Architecture for the unpaired problem
  - Go both ways: total loss takes into account **adversarial terms and "cycle-consistency" loss**
  - **Does not fully solve the correspondence problem**



$\rightarrow_{\downarrow 2}$: convolution (same), non-linear activation, pooling
$\rightarrow_{\uparrow 2}$: bilinear upsampling, convolution (same), non-linear activation OR
$\rightarrow_{\uparrow 2}$: transposed convolution, non-linear activation
$\rightarrow$: convolution (same), non-linear activation
$\rightarrow$: convolution (same), **no activation function**
One can use this architecture with any type of loss function.



Total loss takes into account adversarial terms and "cycle-consistency" loss

$$\underbrace{L_{GAN}(g, D_x, x, y) + L_{GAN}(f, D_y, y, x)}_{Adversarial} + \underbrace{\|x - g(f(x;\theta);\psi)\|_1 + \|y - f(g(y;\psi);\theta)\|_1}_{CycleConsistency}$$

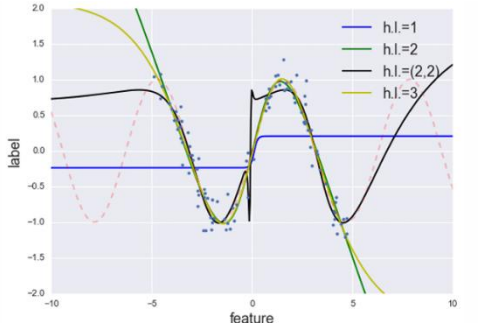
$$L_{GAN}(f, D_y, y, x) = \min_{\theta} \max_{\phi} \mathbb{E}_{y\sim p(y)}[\log D_y(y;\phi)] + \mathbb{E}_{x\sim p(x)}[\log(1 - D_y(f(x;\theta);\phi)]$$

8. Uncertainty estimation in deep learning models

Quantifying uncertainty in model predictions: predictions of machine learning models, like any other models, may be uncertain. Quantifying this uncertainty for medical imaging applications can be crucial.

8.1 Problem

- Classification
  - Some cases are very difficult to identify
  - The disease effect can be very subtle
  - Image may not provide enough information for a clear classification
  - The model should not classify it either
  - Classes predicted with high confidence are taken more into account by raters, **even when they are wrong**, can adversely affect raters
- Segmentation
  - While the images show clearly where the objects are, **boundaries** are much **less clearly defined** due to **limitations in the image resolution and contrast**
  - Prior information can be slightly **different for different people**, so the delineating the boundaries is slightly different between raters
- Detection
  - Detecting abnormalities is a crucial step
  - **Identifying clearly normal and clearly abnormal** anatomy can be tremendously **helpful for improving workflow efficiencies** in clinical practice
  - Some abnormalities are **difficult to discern** from normal anatomy
  - They **require human readers**, and identifying them by quantifying their high uncertainty is important
- Reconstruction from undersampled data
  - An ill-posed problem
  - Missing image information needs to be **complemented using prior information**, prior information can fill the image in multiple ways
  - Models **should not provide confident predictions** for information **missing from data**
- Generally in model fitting
  - When you fit models, you have some data, when applying it afterwards in reality, you may get information from data outside of the range of the training data
  - Important to quantify the variance



8.2 Mathematical treatment

8.2.1 Notation

x	features

y	labels
$f(x;\theta)$	a neural network model
θ	Parameters of the network
$\mathcal{D}$	A training data set
$\mathcal{M}$	model specification, including architecture and hyper-parameters

- Prediction happens with $y \approx f(x;\theta^*)$
- Training happens with $\theta^* = \arg\min_\theta \mathcal{L}(\mathcal{D};\theta)$
- Sources of uncertainty
  - x may not uniquely identify y
  - $\mathcal{D}$ may not uniquely identify a θ^*, and it is a random sample itself
  - $\mathcal{M}$ is probably not a unique model for the problem
  - Combination of $\mathcal{D}$, $f(\cdot;\theta^*)$ and $\mathcal{M}$ may not solve the task perfectly
    - Not necessarily a source of uncertainty but of error that may be different for each sample

8.2.2 Posterior distribution

- The posterior distribution is $p(y|x)$
- We want to find the distribution of **all labels y conditioned on x**, i.e., all labels that are likely to be associated with x
- Then we evaluate the expected value $\mathbb{E}_{y|x}[y]$ or sample from it to get predictions
- The full probabilistic model linking all the components and a factorization
  - $p(y,x,\theta,\mathcal{D},\mathcal{M}) = p(\mathcal{M})p(\mathcal{D})p(\theta|\mathcal{D},\mathcal{M})p(x)p(y|x,\theta,\mathcal{M})$

$p(\mathcal{M})$	**prior** probability for the **model specification**
$p(\mathcal{D})$	probability of **getting the training samples** $\mathcal{D}$ from the entire population
$p(\theta\|\mathcal{D},\mathcal{M})$	posterior distribution of θ, e.g., how well **each** θ can **explain** $\mathcal{D}$
$p(x)$	probability of observing x within the population
$p(y\|x,\theta,\mathcal{M})$	posterior distribution of y given x, model $\mathcal{M}$ and its parameter θ

- To get the posterior $p(y|x)$, we need to marginalize all other variables
  - $p(y|x) = \int_{\mathcal{M}} \int_{\mathcal{D}} \int_{\theta} p(y|x,\theta,\mathcal{M}) dp(\theta|\mathcal{D},\mathcal{M}) dp(\mathcal{D}) dp(\mathcal{M})$
- Assuming all models are equally likely, i.e., $p(\mathcal{M}) = const$, $\int_{\mathcal{M}} \cdot$ requires going through all possible models and averaging.
  - There are **a lot of model choices** and **infeasible to evaluating the other two integrals** for all models
- $\int_{\mathcal{D}} \cdot$ requires going through all possible training sets of size $|\mathcal{D}|$
  - Usually, we **only have one data set**
- $\int_{\theta} \cdot$ requires going through all possible θ and evaluating the model
  - Feasible, but also challenging because it is not trivial to get $p(\theta|\mathcal{D},\mathcal{M})$ as θ is often high dimensional
- **<u>The integrations are challenging, approximations are needed</u>**

8.2.3 Approximation

- We fix one $\mathcal{M}$ and one $\mathcal{D}$
- Training
  - $\theta^* = \arg\min_\theta \mathcal{L}(\mathcal{D}; \theta)$
  - $p(\theta|\mathcal{D}, \mathcal{M}) \approx \delta(\theta - \theta^*)$
- Inference
  - $y \approx f(x; \theta)$
  - $p(y|x, \theta, \mathcal{M}) \approx \delta\big(y - f(x; \theta)\big)$
- Marginalization
  - $p(y|x, \mathcal{M}, \mathcal{D}) = \int_\theta p(y|x, \theta, \mathcal{M}) dp(\theta|\mathcal{D}, \mathcal{M}) = \int_\theta \delta(\theta - \theta^*)\delta(y - f(x; \theta))d\theta = \delta\big(y - f(x; \theta^*)\big)$
- Expected prediction or any other sample
  - $f(x; \theta^*) = \mathbb{E}_{y|x,\mathcal{M},\mathcal{D}}[y]$

8.2.3.1 Sampling instead of integration

Idea: Instead of integrating all the variables, we can do **ancestral sampling** using model

- $p(y, x, \theta, \mathcal{D}, \mathcal{M}) = p(\mathcal{M})p(\mathcal{D})p(\theta|\mathcal{D}, \mathcal{M})p(x)p(y|x, \theta, \mathcal{M})$
  - $\mathcal{D}^s \sim p(\mathcal{D})$
  - $\mathcal{M}^s \sim p(\mathcal{M})$
  - $\theta^s \sim p(\theta|\mathcal{D}^s, \mathcal{M}^s)$
  - $y^s \sim p(y|x, \theta^s, \mathcal{M}^s)$
    - y^s's are samples from the distribution we are after
- To get a point estimate, you can approximate the expected value with the S samples as
  - $\hat{y} = \frac{1}{S}\sum_s y^s$
- In reality, even sampling from these distributions are **challenging** due to various reasons. We need to combine this with approximations of posterior distributions and also priors

Approximating $p(\mathcal{D})$

- Most of the time, we are given only **one training set**, we set it as $\mathcal{D}'$, two options
  - $p(\mathcal{D}) \approx \delta(\mathcal{D} - \mathcal{D}')$, i.e., approximate the distribution with a **Dirac distribution** at the given $\mathcal{D}'$
  - Approximate sampling via **sampling smaller data sets** from $\mathcal{D}'$
    - Randomly sample subset with or without replacement
    - $D^s \subset \mathcal{D}'$
    - Divide the data set into non-overlapping partitions
    - $D^s \subset \mathcal{D}'$ such that $\mathcal{D}' \cup \mathcal{D}^s = \mathcal{D}'$ and $\mathcal{D}^{s_1} \cap \mathcal{D}^{s_2} = \emptyset, \forall s_1, s_2$

Approximating $p(\mathcal{M})$

- Model specification is something we choose so we can in theory sample from it. Not trivial, prior distribution over models have been investigated in the past
- Two options

- $p(\mathcal{M}) \approx \delta(\mathcal{M} - \mathcal{M}')$, i.e., approximate the distribution with a **Dirac distribution** at a preferred model $\mathcal{M}'$
- Approximate sampling via **sampling different models**
  - Approximate sampling of different models is often done
  - By selecting a number of preferred models, e.g., a fully convolutional network and a UNet for segmentation
  - By generating new models around a preferred model, e.g., adding layers to channels of a UNet, changing architecture in the encoding path, removing skip connections
- **Not feasible to cover all possible models**. Cost of sampling multiple models is the increased time and energy needed to train all of them

Approximating $p(\theta|\mathcal{D}, \mathcal{M})$

- Use **Bayes' theorem** to better understand this and **link it to the cost functions**.
- We assume $\mathcal{M}$ is given
  - $p(\theta|\mathcal{D}, \mathcal{M}) = \frac{p(\mathcal{D}|\theta,\mathcal{M})p(\theta|\mathcal{M})}{p(\mathcal{D})}$
  - The conditioning on $\mathcal{M}$ is removed in the denominator since $\mathcal{D}$ **does not depend on the model**
- There are two parts of the numerator
  - $p(\theta|\mathcal{M})$ is the **prior distribution** for the model parameters
  - $p(\mathcal{D}|\theta, \mathcal{M})$ is the **likelihood**, i.e., how well the parameters and the model satisfy the data
- The usual training is the maximum-likelihood estimation of the parameters
  - $\arg\max_\theta \log p(\mathcal{D}|\theta, \mathcal{M})$
  - Where $\log p(\mathcal{D}|\theta, \mathcal{M})$ correspond to the loss function $\mathcal{L}(\mathcal{D}; \theta)$
- If there is a regularization over the weight parameters, e.g., weight decay, then the usual training can be seen as **the maximum-a-posterior (MAP) estimation** of the parameters
  - $\arg\max_\theta \log p(\mathcal{D}|\theta, \mathcal{M}) + \log p(\theta|\mathcal{M})$
  - Where $\log p(\theta|\mathcal{M})$ corresponds to the regularization term $\mathcal{R}(\theta)$ we have seen
- *Note*: The MAP estimation is the **same as the maximum-likelihood estimation** if there is **no regularization term**, i.e., $p(\theta|\mathcal{M}) = const$
- The difficulty with $p(\mathcal{D}|\theta, \mathcal{M})$ in deep learning models
  - $\max_\theta \log p(\mathcal{D}|\theta, \mathcal{M}) \Leftrightarrow \min_\theta \mathcal{L}(\mathcal{D}; \theta)$
    - The cost function $\mathcal{L}(\mathcal{D}; \theta)$ is highly **non-convex** with **multiple possible local minimal**
      - $p(\theta|\mathcal{D}, \mathcal{M})$ is **possibly multi-modal**
    - Different **initializations** of the minimization $\min_\theta \mathcal{L}(\mathcal{D}; \theta)$ leads to very **different "optimal" parameters** at the end of training
      - Modes of $p(\theta|\mathcal{D}, \mathcal{M})$ can be quite far away

- $p(\theta|\mathcal{M})$ can remedy these points a bit but **often does not solve it completely** without sacrificing too much of prediction performance
- Changing parameters in one layer can lead to s**ubstantial change in parameters in the other layers**
  - $p(\theta|\mathcal{D},\mathcal{M})$ has **strong statistical dependencies** between different elements of θ
  - Approximate factorization of the sort $p(\theta|\mathcal{D},\mathcal{M}) \approx \prod p(\theta_i|\mathcal{D},\mathcal{M})$ **not be very good approximations**

8.2.3.2 Approximating posterior distributions

Three main options for approximating $p(\theta|\mathcal{D},\mathcal{M})$

- $p(\theta|\mathcal{D},\mathcal{M}) \approx \delta(\theta - \theta^*)$ with θ^* being one MAP estimation
  - Assumption is that $p(\theta|\mathcal{D},\mathcal{M})$ is **unimodal** and with a **high peak**, most likely **not true**
- Approximate sampling with by solving $\theta^s = \arg\min_\theta \log p(\mathcal{D}|\theta,\mathcal{M}) + \log p(\theta|\mathcal{M})$ with a **different initialization** for every sample
  - E.g., $\theta_0^s \sim p(\theta_0|\mathcal{M})$ where a sample factorized distribution can be used as a prior
  - Assumption is that **different initialization** will **capture different modes** of the posterior distribution with multiple models
- Approximate the posterior around a MAP estimate **with a known distribution**, i.e., $p(\theta|\mathcal{D},\mathcal{M}) \approx p(\theta|\theta^*,\mathcal{M})$
  - Using Laplace approximation $p(\theta|\mathcal{D},\mathcal{M}) \approx \prod_i \mathcal{N}(\theta_i;\theta_i^*,\sigma)$
  - Where σ can be assigned after the MAP estimation
  - **Can combining one of the first two with the third**

Training with approximation for option 3

- Instead of using an approximation fixed after MAP estimation, we can train the model with the distributional approximation without any priors over θ
  - E.g., $\arg\max_\mu \log p(\mathcal{D}|\theta',\mathcal{M})$, $\theta' \sim \prod_i \mathcal{N}(\theta_i;\mu,\sigma)$
  - Where σ needs to be fixed
- We can also have priors over θ
  - E.g., $\arg\max_{\mu,\sigma} \log p(\mathcal{D}|\theta',\mathcal{M}) - D_{KL}[\prod_i \mathcal{N}(\theta_i;\mu,\sigma)\,||\prod_i \mathcal{N}(\theta_i;\mu_0,\sigma_0)]$
  - Where $\theta' \sim \mathcal{N}(\theta_i;\mu,\sigma)$
  - *Note*: D_{KL} is the Kullback-Leibler divergence and the final argument in the optimization corresponds to the Evdence Lower Bound (or variational lower bound) for $p(\mathcal{D}|\mathcal{M})$

Modeling $p(y|x,\theta,\mathcal{M})$ **instead of approximating it**

- Most neural network models are **deterministic**, **no ambiguity in generating the output prediction**
- We should see this step as modeling $p(y|x,\theta,\mathcal{M})$, the simple model would suffice since the prediction is **deterministic**

    - $p(y|x,\theta,\mathcal{M}) = \delta\big(y - f(x;\theta)\big)$
- However, this model can be improved by taking into account the fact that **multiple** y's **can correspond to the same** x in the problem
    - E.g., in super-resolution **many higher resolution images** may **correspond** to a given **lower-resolution image**, multiple segmentations can be drawn from the same image by different experts
- The model $p(y,\theta,\mathcal{D},\mathcal{M}|x) = p(\mathcal{M})p(\mathcal{D})p(\theta|\mathcal{D},\mathcal{M})p(y|x,\theta,\mathcal{M})$ is the **only dependency** of y on x. Therefore, it is the best place to integrate the fact that multiple y's can correspond to a single x
- **Beyond the simple model for** $p(y|x,\theta,\mathcal{M})$, two approaches
    - The model predicts parameters of a **distribution at the output**, not a single prediction
        - $p(y|x,\theta,\mathcal{M}) = \mathcal{N}\big(y; f(x;\theta), g(x;\theta)\big)$
        - $f(\cdot;\theta)$ is the mean, and $g(\cdot;\theta)$ is the standard deviation
        - The **distribution itself can be changed too**, does not have to be Gaussian
        - Training becomes $\max_{\theta} \log p(y|x,\theta,\mathcal{M})$
        - For the Gaussian example, it gives an L_2 norm and a term for standard deviation
            - $\max_{\theta}\left\{-\log\big(g(x;\theta)\big) - \frac{1}{2}\frac{\big(y-f(x;\theta)\big)^2}{g(x;\theta)^2}\right\}$
        - For prediction at **each pixel**, the model can be defined as $p(y|x,\theta,\mathcal{M}) = \mathcal{N}\big(y; f(x;\theta), g(x;\theta)\big)$, and integrate as $p(y|x,\theta,\mathcal{M}) = \int_z p(y|z,x)p(z|x)dz$
    - **Generating different samples given the input**, instead of modeling a probability distribution
        - To generate different samples, the model has two inputs
            - $y' = f(x,z';\theta), z' \sim p(z)$
            - Where z is a **random input**
        - To make sure that different samples of y' make sense, it is common to use a **distributional loss** in the training
            - $$\min_{\theta}\max_{\psi}\sum_{n} L(y_n, f(x_n;\theta)) + \lambda\left(\mathbb{E}_{y\sim p(y)}[\log D(y;\psi)] + \mathbb{E}_{y=f(x,z';\theta),x\sim p(x),z'\sim p(z)}[\log(1 - D(y;\psi))]\right)$$

8.2.4 Evaluation

- The **<u>most challenging component</u>** in modeling uncertainty
- How do we know the distribution generated by our model truly captures uncertainty? It is not reasonable to ask we have access to the true posterior distribution
- Ther are two main methods for evaluation
    - **Comparison with observed samples**

    - If there is a data set with multiple labels for each image, distributional distances can be computed based on the samples, e.g., Maximum Mean Discrepancy (MMD) can Generalized Energy Distance
- **Calibration error**, i.e., when I don't have observed samples
    - If we do not have multiple samples for each image or a subset. This is the most general case, we assume whether the uncertainty assigned by the model reflects error, e.g., correlation between amount of uncertainty and the error on test samples, expected calibration error (ECE)

9. Domain adaptation and learning from unlabeled examples

9.1 Domain adaptation

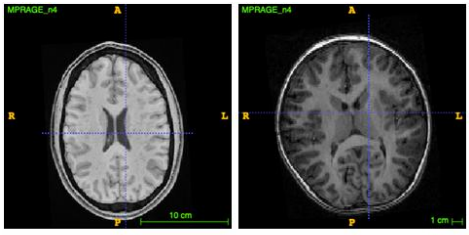
9.1.1 Problem definition

- **Problem with generalization**: models train with one training set does not perform well on another data set with different, even slightly, intensity characteristics, called **domain shifts**
- How do we tackle this problem?
- Segmentation as an example
  - Pixel-wise class assignments given image intensities
  - The state-of-the-art is clearly deep learning for a variety of anatomies, pathologies and modalities
  - Segmentation accuracies are close to inter-rater variability (when there is no domain shift)
  - Even slight differences between statistics of training and test data can lead to **substantial performance degradation**, CNNs are not robust against "domain shifts"
- Issue not specific to segmentation: also pertinent for restoration, synthesis, reconstruction, registration, etc.
- Domain shift problem definition

Type	Change	Examples of differences
Population shift	$P_D(Z)$	Ages, sexes, diets, habits, ethnicities, genetics
Annotation shift	$P_D(Y\|X)$	Annotation policy, annotator experience
Prevalence shift	$P_D(Y)$	Case-control balance, target selection
Manifestation shift	$P_D(Z\|Y)$	Anatomical manifestation of the target disease or trait
Acquisition shift	$P_D(X\|Z)$	Scanner, resolution, contrast, modality protocol

Note: Acquisition shift can be dealt with machine learning easily.

Notations



- **Domain**: a distinct subset of samples showing similar variations, e.g., center, scanner, sequence, acquisition protocol
- **Source domain**: Subset of samples with which models are **trained**
- **Target domain**: Subset of samples on which models are applied

Both can consist of multiple subsets, i.e., samples from multiple domains

- **Domain shift**
  - (x^{SD}, y^{SD}): image and label from the **source domain**
  - (x^{TD}, y^{TD}): images and label from the **target domain**

- $\mathcal{D}^{SD}$: set of images and labels from the **source domain**
- $\mathcal{D}^{TD}$: set of images and labels from the **target domain**
- Target domain $\subset$ source domain: algorithms should generalize
- Target domain $\not\subset$ source domain: **domain shift**
- Training of a model happened on set of samples different than those seen during inference

9.1.2 Naïve solution

Idea: **Separate training for each domain** with **separate training set and final model**. Inference at **each domain** with **domain-specific model**.

At source domain		At target domain	
Training data many samples $\{(x_n^{SD}, y_n^{SD})\}$	Algorithm $\min_\theta \mathcal{L}(\mathcal{D}^{SD}; \theta)$	Training data many samples $\{(x_n^{TD}, y_n^{TD})\}$	Algorithm $\min_\theta \mathcal{L}(\mathcal{D}^{TD}; \theta)$
Data at inference $(x^{SD}, \cdot)$	Prediction $y^{SD} \approx f(x^{SD}; \theta^*_{SD})$	Data at inference $(x^{TD}, \cdot)$	Prediction $y^{TD} \approx f(x^{TD}; \theta^*_{TD})$

- If enough labeled examples are present, this is the best performing model
- Requires large set of labeled samples at every domain when using the algorithm
  - Labeled samples are **expensive** to obtain because data set curation and labeling take a lot of time

9.1.3 Transfer learning

Idea: **Initial training** on the **source domain** with **many samples**. **Fine-tune** the model at the **target domain** with **few samples**.

At source domain		At target domain	
Training data many samples $\{(x_n^{SD}, y_n^{SD})\}$	Algorithm $\min_\theta \mathcal{L}(\mathcal{D}^{SD}; \theta)$	Training data very few samples $\{(x_n^{TD}, y_n^{TD})\}$	Algorithm $\min_\theta \mathcal{L}(\mathcal{D}^{TD}; \theta)$
Data at inference $(x^{SD}, \cdot)$	Prediction $y^{SD} \approx f(x^{SD}; \theta^*_{SD})$	Data at inference $(x^{TD}, \cdot)$	Prediction $y^{TD} \approx f(x^{TD}; \theta^*_{TD})$

- $\min_\theta \mathcal{L}(\mathcal{D}^{TD}; \theta)$, $\theta^{(0)} = \theta^*_{SD}$
- Model training **starts at the optimal source domain parameters**
- Requires only few labeled samples at the target domain
- At each target domain it still requires labeled samples for fine-tuning

Reducing the number of needed labeled samples for the fine-tuning step

- Use **unlabeled** samples, the semi-supervised learning setting
- **Reduce the number of parameters** to train during fine-tuning, especially useful when **unlabeled** examples are **not there**

- E.g., for a UNet, we can choose to fix most of the parameters and only change parameters between I and $L1$
- Change only the "batch-norm" parameters in the network $BN(a_l) = \gamma \frac{a_l - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} + \beta$, all the rest remain the same

9.1.4 Unsupervised domain adaptation

Idea: Train models **without** any **labels** at the **target domain**. Nothing really interesting happens at the source domain.

At target domain	
Training data many unlabeled samples	Algorithm
$\{(x_n^{TD}, \cdot)\}$ and $\{(x_n^{SD}, y_n^{SD})\}$	$\min_\theta \mathcal{L}(\mathcal{D}^{SD}; \theta) + \mathcal{L}_{adv}(\mathcal{D}^{TD}, \mathcal{D}^{SD}; \theta)$
Data at inference $(x^{TD}, \cdot)$	Prediction $y^{TD} \approx f(x^{TD}; \theta_{TD}^*)$

- Training with source domain and unlabeled target domain samples
- $\min_\theta \mathcal{L}(\mathcal{D}^{SD}; \theta) + \mathcal{L}_{adv}(\mathcal{D}^{TD}, \mathcal{D}^{SD}; \theta)$
- An adversarial loss ensures similar features are **extracted from source and target domain samples**
  - *Goal*: Ensure **similar features** are extracted from source and target domain samples
  - The network is **blind** to **differences between samples** coming from the **different domains**
  - Distributional loss between features
  - $$\underbrace{\min_\theta \sum_n L(y_n^{SD}, f(x_n^{SD}; \theta))}_{\text{Supervised loss}} + \underbrace{\max_\psi \mathbb{E}_{x \sim p_{SD}} [\log D(I(x); \psi)] - \mathbb{E}_{x \sim p_{TD}} [1 - \log D(I(x); \psi)]}_{\text{Adversarial loss}}$$
  - $I(x) = [L2(x), L3(x), L4(x)]$
  - **Supervised loss**: Task specific loss using labels on source domain samples
  - **Adversarial loss**: Distributional loss measure **distance between feature distributions of samples** coming from source and target domains
- Does not require any labeled images at target domain
- A new training at each target domain and source domain images need to transported to each target domain

9.1.5 Extreme data-augmentation

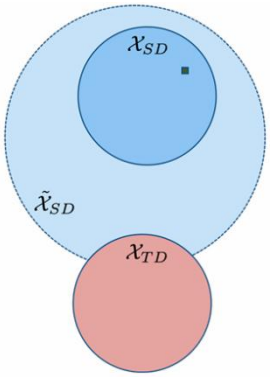
Idea: Transformations of the data samples serve as a new sample

At source domain	
Training data many samples and augmentation	Algorithm
$\{(x_n^{SD}, y_n^{SD})\} \cup \{(\hat{x}_n^{SD}, \hat{y}_n^{SD})\}$	$\min_\theta \mathcal{L}(\mathcal{D}^{SD} \cup \hat{\mathcal{D}}^{SD}; \theta)$
Data at inference $(x^{SD}, \cdot)$	Prediction $y^{SD} \approx f(x^{SD}; \theta_{SD}^*)$

At target domain	
Training data no training samples	Algorithm
$\emptyset$	None
Data at inference $(x^{TD}, \cdot)$	Prediction $y^{TD} \approx f(x^{TD}; \theta_{SD}^*)$

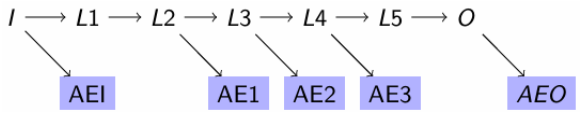
- Very effective way to tackle lack of data samples
- The hypothesis is that augmentation can mimic new domains

- **Stacked transformations**: $(\hat{x}_s, \hat{y}_s) = \tau^n_{p_n,m_n}(\tau^{n-1}_{p_{n-1},m_{n-1}}(\cdots \tau^1_{p_1,m_1}((x_s, y_s))))$
  - τ^n is the n-th transformation that is applied with probability p_n and magnitude m_n
- **Image quality transformations**: sharpness, blurriness, noise level
- **Image appearance transformations**: brightness (intensity shifts), contrast (gamma correction)
- **Spatial configuration**: rotation, scaling, deformations
- *Note*: the transformations do not affect the labels.
- Pictorially what is happening
  - X_{SD} is the set of source domain images and X_{TD} is the set of target domain images
  - Through augmentation X_{SD} is expanded to $\tilde{X}_{SD}$ hoping that the expanded set **overlaps** with X_{TD}
  - Data augmentation provides a very strong method
  - Even then, performance improvement may be **limited** to the variations covered by the augmentation strategy



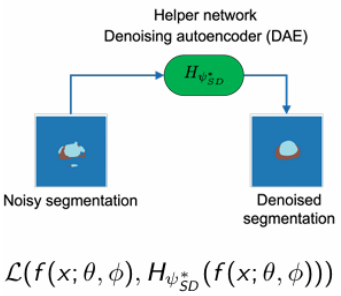
9.1.6 Unsupervised source-free domain adaptation

- Two main questions
  - What cost function will you optimize when you don't have labels? $\mathcal{L}\big(f(x^{TD}; \theta^*_{SD}, \phi)\big) = ?$
  - Given a network and a cost function, which parameters will you train / fine-tune? $f(x; \theta, \phi)$, what is ϕ?
- Cost functions to optimize
  - Entropy of the prediction $H\big(f(x;\theta)\big) = -\sum_k f_k(x;\theta)\log\big(f_k(x;\theta)\big)$
    - Assumption is that pushing confidence of the prediction higher leads to good domain generalization
    - Simple to implement, no prior information, applicable to different tasks
    - Strong assumption. Holds if the prediction is not very off and if there are many test samples that you adapt to all together
  - Auto-encoder loss of input, output, and intermediate features
    - Auto-encoders are trained with **source domain**. **Differences** between their **inputs and outputs** ae evaluated as the **cost**
    - An auto-encoder (AE) is a **map from an image to the image itself**, $x \approx D(E(x))$ where D is decoder and E is encoder, giving a good reconstruction on samples coming from the domain it was trained for



    - Generally applicable to different tasks and strong prior model
    - Input and intermediate features can be prone to domain shifts themselves

- Segmentation **prior** evaluated at the output assessing plausibility of the result
    - A denoising auto-encoder (DAE) is a map from **a noisy version of an image to itself** $x \approx D(E(\tilde{x}))$ with D is decoder and E is encoder
    - Trained on segmentation maps to learn a **prior model** on the segmentations



$\mathcal{L}(f(x;\theta,\phi), H_{\psi^*_{SD}}(f(x;\theta,\phi)))$

    - **Strong prior** leads to **accurate segmentation results**. Segmentations are **not affected by intensity changes** so the model is not affected by domain shifts of this sort
    - Specific to segmentation task
- Parameters to optimize
    - Train / fine-tune **all the parameters**
        - Model will be very **flexible**
        - Too much flexibility can lead to a result that **perfectly satisfies the cost** but the output is **no longer related to the input**
    - Train / fine-tune **a part of the parameters**
        - Model will be less flexible so that issue with too much flexibility will not be an issue
        - Initial model parameters are obtained using large curated data sets. It is a desirable **not to change** them if not needed. It was costly to train them in terms of time, data, and money
    - Train / fine-tune **batch-norm parameters**
        - Very **simple application** and very **few parameters**
        - Batch norm parameters in **deeper layers** can **affect results a lot** and it is **unclear** if they address **changes in intensity** characteristics
    - Tran / fine-tune **a shallow "adaptation" module prepended to the network**
        - **Shallow module will not be too flexible**, and we **do not modify pre-trained parameters**
        - We make the network **slightly bigger**

9.2 Learning from unlabeled examples

9.2.1 Problem

- We **do not have too many labeled examples**, but we **have access from unlabeled examples**
- A huge success of deep learning is due to large labeled data sets. The more data, the better (does not saturate), but it is still not enough for medical data because very large scale labeled data sets is **not feasible** for many of medical applications

    - Manual annotations can be very **costly to obtain**: most of the time segmentations are performed manually by experts, i.e., people who know what they are segmenting
    - The nature of the problem might **not permit generating large number of labeled examples** as a clean image takes much longer and more radiation, ground truth images may be very costly to acquire
    - In the low-dose CT example, it may be **possible to get pairs of images** from some people but **not all** (the high dose CT is only done if it isn't seen on the low-does one)
    - The **ground truth may not be possible to acquire**, simply because the technology is not there yet (super high zoomed in images)
- **Few labeled images are often possible**. To be able to work with fewer labeled images would be better
- **Unlabeled examples** are simply images **without labels**, unlabeled examples are often available in large numbers. In some problems, even unlabeled examples are not available in large numbers

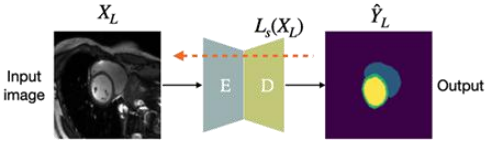
9.2.2 Noise-to-noise

Idea: Learning to **denoise without clean target data**

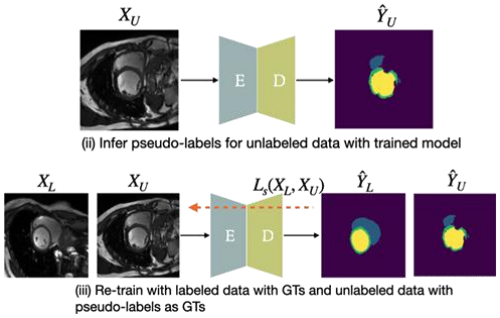
- Objective function with clean images y_n is $\arg\min_\theta \sum_n \mathcal{L}\big(y_n, f(x_n;\theta)\big)$
- Assume we do not have a clean image but a **distribution of noisy images** for the same sample x, i.e., $p(\hat{y})$
- If we minimize $\arg\min_{f(x;\theta)} \mathbb{E}_{\hat{y}}\left[\left|\left|f(x;\theta)-\hat{y}\right|\right|^2\right]$ we would get
    - $f(x;\theta) = \mathbb{E}_{\hat{y}}[\hat{y}]$
- If we now also average over the x samples, in the limit computing the expectation with respect to $p(x)$, and minimize with respect to the parameters of the network
    - $\arg\min_\theta \mathbb{E}_x\{\mathbb{E}_{\hat{y}|x}\{||\hat{y}-f(x;\theta)||^2\}\}$
- This is **minimizing the distance** between the **output of the network** and a **noisy sample** from $p(\hat{y}|x)$ and averages over $\hat{y}$ and over x. It does not use a clean image
- Simple cost function
- In some cases, it works better than supervised versions
- It is a probabilistic and statistical method, interpretable
- It is noisy, removing some edge details

9.2.3 Self-training

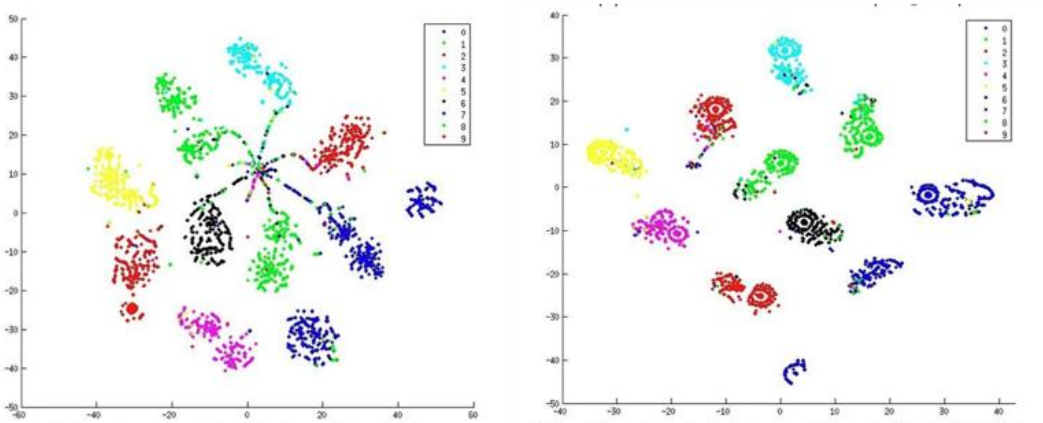
- Similar to entropy minimization
- Strategy
    - Train with labeled data with ground truths
        - $\theta_0^* = \arg\min_\theta \sum_n \mathcal{L}\big(y_n, f(x_n;\theta)\big)$
        -



- Infer pseudo-labels for unlabeled data **with trained model** in the i-th iteration
  - $\hat{y}_m = f(x_m; \theta_i^*)$
- Retrain with labeled data with ground truths and unlabeled data with pseudo-labels as ground truths
  - $\theta_{i+1}^* = \arg\min_\theta \sum_n \mathcal{L}(y_n, f(x_n; \theta)) + \lambda \sum_m \mathcal{L}(\hat{y}_m, f(x_m; \theta))$
- Iterate over steps 2 and 3



- Why does it work?
  - **Entropy** minimization
  - When we visualize features and labels, there are **clusters**; the pseudo-labels add to the density so there are **more clear boundaries after the self-training**



(a) without unlabeled data (dropNN)

(b) with unlabeled data and Pseudo-Label (+PL)

- Things that need to pay attention
  - If the **initial** model's estimates are **not bad**, **self-training works well**
  - Entropy minimization **assumes mostly correct class assignments**
  - If the initial model's estimates are **not good**, self-training **quickly diverges**
  - Additional **regularizations** on the **final estimates** can **prevent divergence**
  - Even with the regularization terms, the model can **diverge quite often**

9.2.4 Teacher-student models

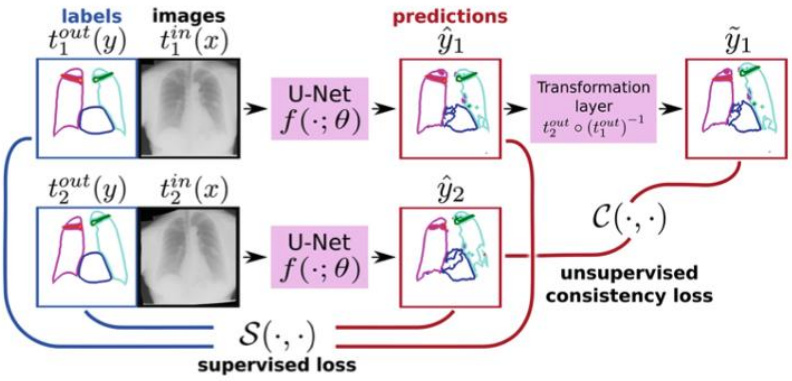
Idea: A new type of regularization - train **two interacting networks** for **generating pseudo-labels**. **One evolving slower than the other**.

- Student model: $f_s(x; \theta)$
- Teacher model: $f_t(x; \phi)$
- Cost for labeled examples: $\mathcal{L}(y; f_s(x; \theta))$
- Consistency loss: $\mathcal{L}_c(f_s(x; \theta); f_t(x; \phi))$
- **Can be applied to both labeled and unlabeled examples**

- Total loss for student network: $\sum_n \mathcal{L}\big(y_n; f_s(x_n;\theta)\big) + \lambda\big(\sum_n \mathcal{L}_c\big(f_s(x_n;\theta); f_t(x_n;\phi)\big) + \sum_m \mathcal{L}_c\big(f_s(x_m;\theta); f_t(x_m;\phi)\big)\big)$
- Teacher and student networks are trained in an alternative fashion
- In the most well-known mean teacher model, teacher network is learned as an **exponential moving average** of the student network
    - For a given teacher network, train the student network
        - $\theta^i = \arg\min_\theta$(Student loss total loss)
    - Update teacher network's parameters
        - $\phi^i = \alpha\phi^{i-1} + (1-\alpha)\theta^i$
    - Iterate between steps 1 and 2

9.2.5 Semi-supervised learning

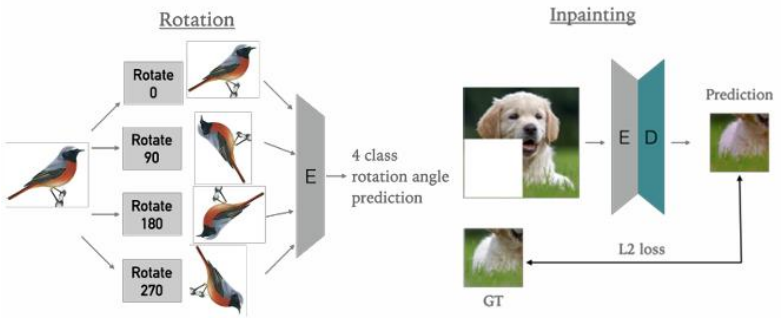
- **Semi-supervised auxiliary loss**: defined over unlabeled images **during task-specific optimization**
- Consistency under transformation
    - In pixel-wise predictions, **spatial transformation** to the input should be **matched** at the output. If ϕ is a spatial transformation, e.g., affine transformation or non-linear deformation
        - $x \Leftrightarrow y \Leftrightarrow \phi \circ x \Leftrightarrow \phi \circ y$
    - This is the underlying principle in data augmentation
- We use this s for constructing an **auxiliary loss** for **unlabeled examples** for segmentation
    - For every image, two transformations ϕ_1 and ϕ_2 can be sampled, applying them to the sample yields
        - $\phi_1 \circ x, \phi_2 \circ x, \phi_1 \circ y, \phi_2 \circ y, f(\phi_1 \circ x;\theta), f(\phi_2 \circ x;\theta)$
    - Consistency can be enforced by
        - $\mathcal{L}_{cons}(x, \phi_1, \phi_2; \theta) = \mathcal{L}\big(\phi_2 \circ \phi^{-1} \circ f(\phi_1 \circ x;\theta), f(\phi_2 \circ x;\theta)\big)$
        - Where $\mathcal{L}$ is a usual loss for pixel-wise predictions
    - Notice that the consistency loss **does not require any labels**
    - Combining with the usual loss gives the final semi-supervised cost with the second sum going over both labeled and unlabeled examples



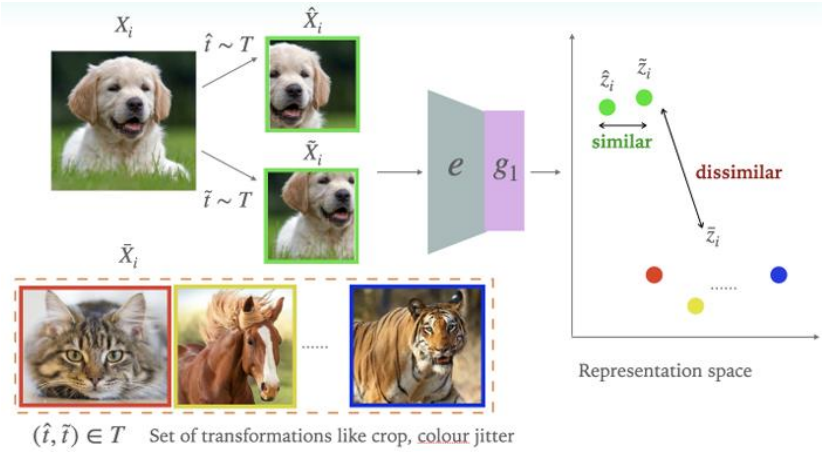
9.2.6 Self-supervised learning

Idea: Train a network with pre-text task that only requires images. The trained network will be suitable to fine-tuning with very few labeled examples to other tasks.

- **Self-supervised auxiliary loss**: defined over unlabeled images **independent of the task**
- The crucial problem is to identify a **pre-text task** that will lead to useful network parameters
- The underlying assumption is that such tasks are **good for learning "generalizable" parameters**

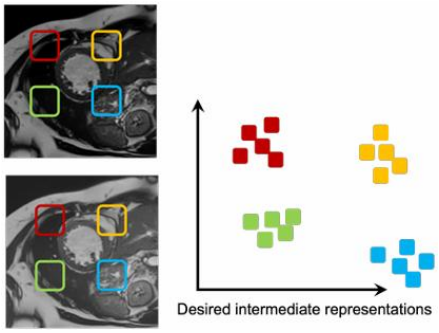


- These tasks are **hand-crafted**. One assumes that these are good, and empirically they are shown to work to some extent
  - Rubik's cube for medical image analysis, wanted to find out what the original position of the voxel was
- **Contrastive learning** - forcing compact representations with surrogate labels

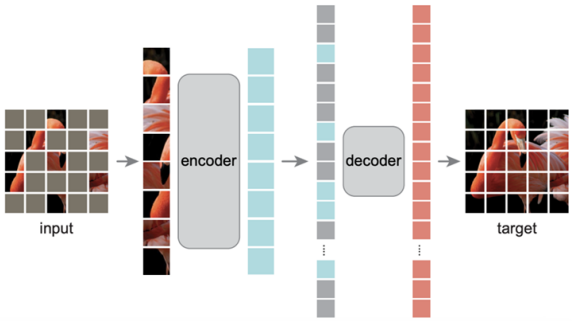


  - **Surrogate labeling**: **random transformations** of the same image are "**close**"
  - Transformation ϕ are **randomly chosen** and **not restricted to geometric** transformations. They also include **intensity changes**
  - We force them to be **close** in the representation space as well, with the cost function
    - Numerator **maximizes the cosine similarity** between samples with the **same content**
    - Denominator **minimizes the cosine similarity** between samples with **different content**

- Local contrastive loss: In addition to the global compact representations, one can force local representations, one can force local representation to be compact as well. This is particularly useful for pixel-wise predictions



- **Masked autoencoders**
  - Masked autoencoders, akin to **inpainting**, reconstruct occluded parts of images, training the model to predict missing information
  - An MAE consists of a **larger encoder** that processes the **visible parts** and a **smaller decoder** that **reconstructs the occluded parts**
  - Parts of the images are **masked randomly**; this randomness encourages the model to learn **robust, generalizable representations**
  - The trained encoder, having learned valuable features, can then serve as an effective backbone for various downstream tasks



10. Explainable AI

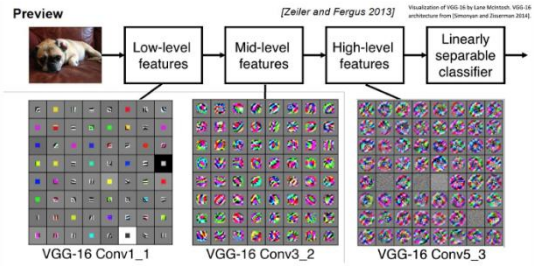
10.1 Basic concepts

- **Definition**: What is interpretability? An interpretable machine learning algorithm is described as one where the link between the features used by the machine learning and the prediction itself can be **understood by a human**
- **Another definition**: produce explanations **without** sacrificing accuracy
  - Simpler is easier to understand
  - But oversimplified is typically from an accuracy point of view, not interesting
- Interpretability in AI medical imaging
  - Interpretability **enhances data preparation** and **AI performance**
- Taxonomy
  - **Black-boxes** operating methods: **No need to have "access" to the internal of models**. Also referred to as model-agnostic
  - **White-boxes**: These are methods that **require access to the internals of the models**. Gradients-based models are one example of white box interpretability models
  - **Output**
    - Visualization, or saliency maps: Provide pixel-wise values reflecting their importance to the performance of the model being interpreted
    - Concepts: summary statement or keyword
    - Feature importance: explain models by expressing importance of features, e.g., high weights of a model

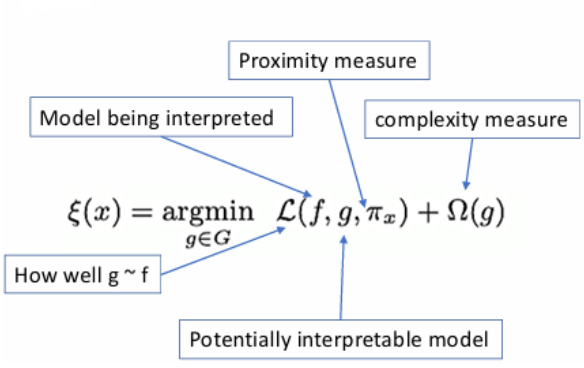
10.2 Interpretable deep learning

10.2.1 Convolutional neural networks

- Typical structure
  - Convolution layers
  - Pooling or subsampling layers
  - Fully connected layers



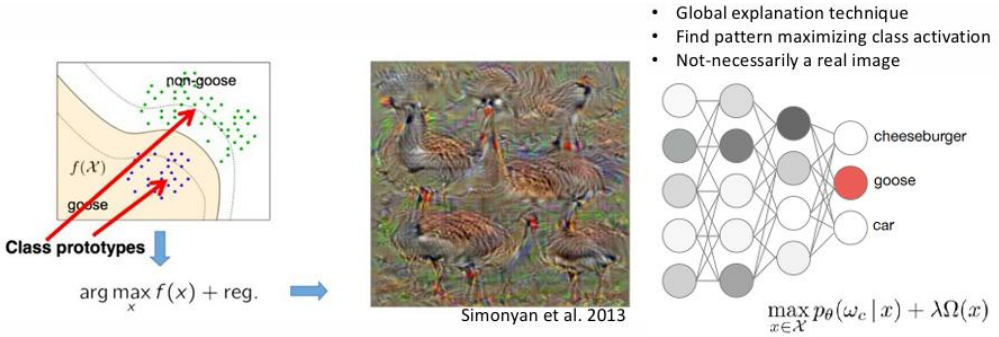
10.2.2 Local interpretable model-agnostic explanations (LIME)



Algorithm 1 Sparse Linear Explanations using LIME

Require: Classifier f, Number of samples N
Require: Instance x, and its interpretable version x'
Require: Similarity kernel π_x, Length of explanation K
  $\mathcal{Z} \leftarrow \{\}$
  for $i \in \{1, 2, 3, ..., N\}$ **do**
    $z'_i \leftarrow sample\_around(x')$
    $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
  end for
  $w \leftarrow$ K-Lasso$(\mathcal{Z}, K)$ ▷ with z'_i as features, $f(z)$ as target
  return w

- Look at the problem **locally** and make it **simpler to understand** there (perturbed images where only part of the image is visible, get a probability for the classification for those images)
- Locally weighted regression: have probabilities that are more **distant** to the region being analyzed have **less weight**
- **<u>Does not need the internals of the model</u>** (black-box)
- Find prototypes
    - Force the probability for one class to be **maximized**
    - To backpropagation and look what comes out (that creates the highest probability for that class)
    - **Drawback**: we have **no ground truth** so the picture is subject to human (subjective) evaluation



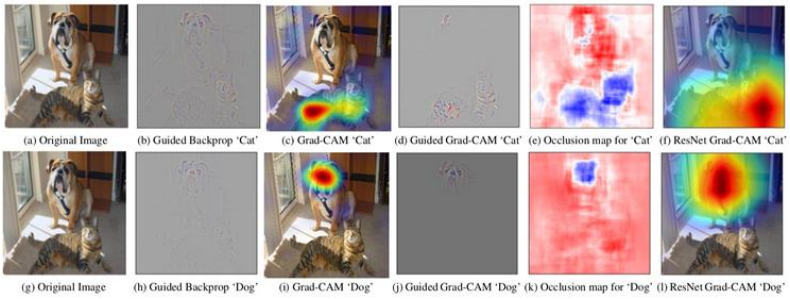
    - Too many degrees of freedom, do the same but with regularization in the image generation
        - Better to see the different classes in the prototypes

10.2.3 Gradient-based methods

<u>Idea</u>: Magnitude of gradient reflects importance (attribution) of input to output scores (the weight it gives to the feature)

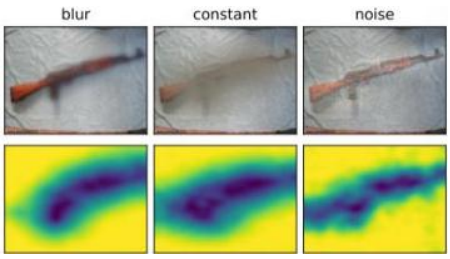
- Different variations
- Differences between backprop

10.2.3.1 Grand-CAM

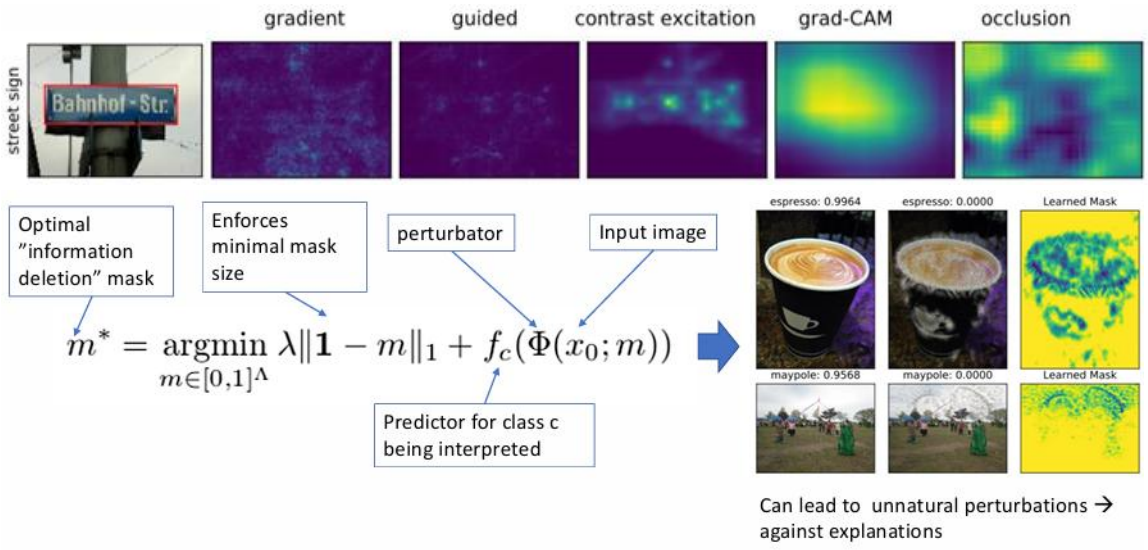


- Visual explanations from deep networks via gradient-based localization
- Applicable to any CNN-based (CAM requires convolution feature maps, global average pooling, softmax layer)
- Motivation - good visualization is **class-specific** and **detailed**
- **Negative information is removed**; sometimes, the lack of something also carries information

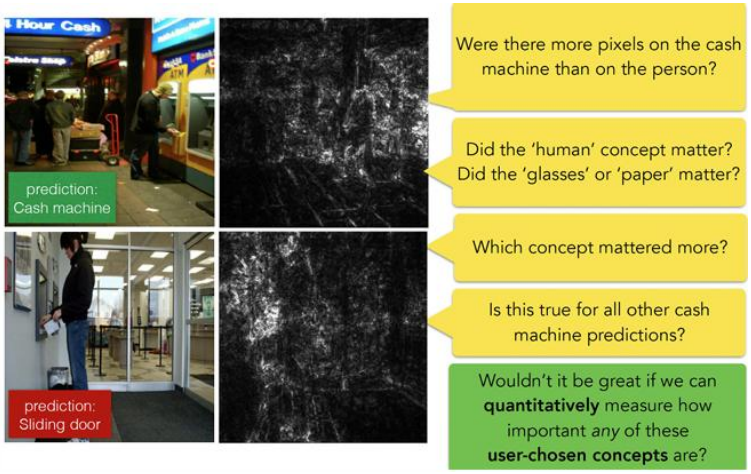
10.2.3.2 Meaningful perturbation



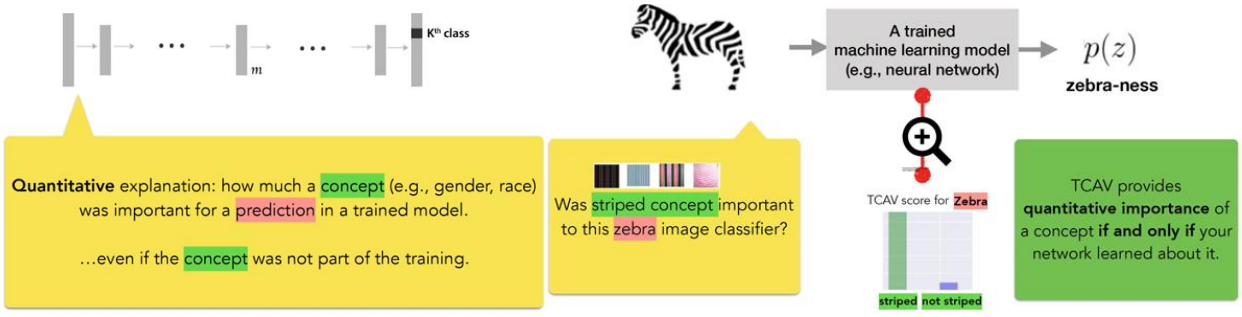
- **Motivation**: Saliency (gradient-based) approaches are not specific enough
- **Key idea**: Select the right perturbation (type of information deletion) to study the effect on the prediction function $f(x)$
- Meaning of an explanation depends on the meaning of the changes applied to the input x



Testing with concept activation vectors (TCAV)



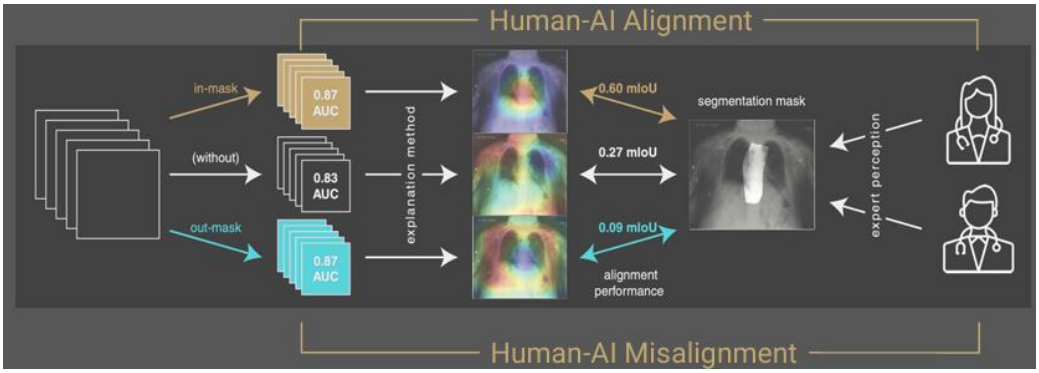
- Can we make predictions based on a concept



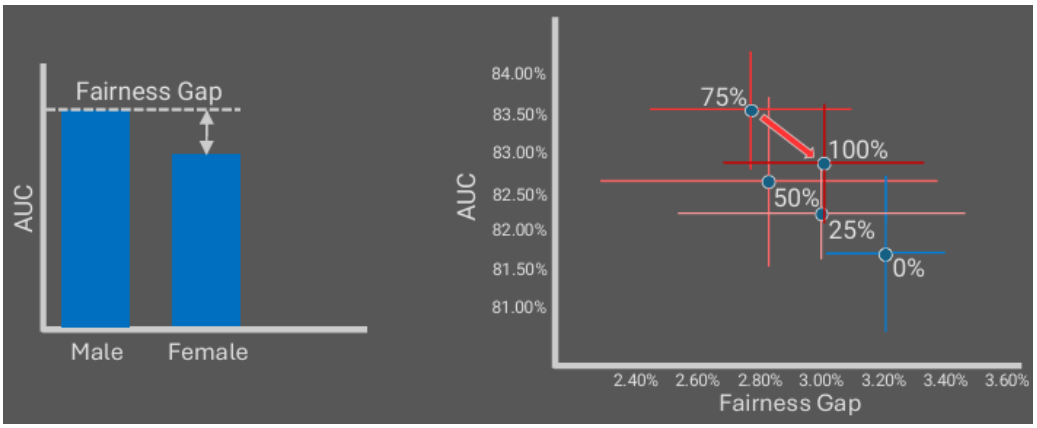
- Create concepts (e.g., stripes, legs, size, etc. for a zebra) and test if the model also used those concepts for the prediction
    - Make a library with concepts and non-concepts
    - Train a **linear classifier** to separate activations (concepts vs non-concepts)
    - CAV is the vector orthogonal to the decision boundary
    - Find the vector
    - You want the **non-concepts not to be too far off**, because then the space to fit the hyperplane is too large and the direction of the orthogonal vector is instable
    - TCAV used in two widely used image prediction models

10.3 What is AI learning?

- Right for the wrong reasons: shortcut learning as a barrier to clinical deployment



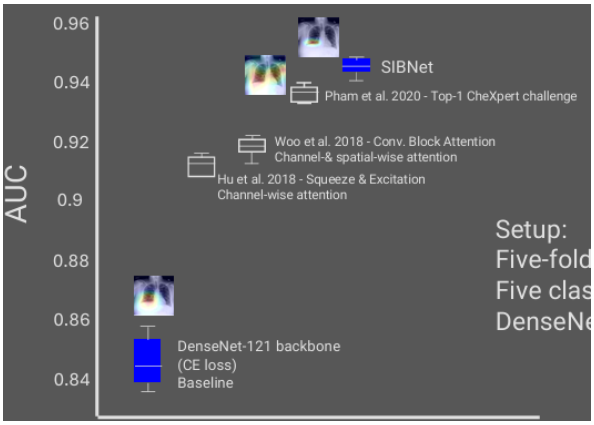
- Fair and trustworthy
    - First do no harm
    - Balance between guidance (machine and human) and pattern discovery (machine and data, inductive bias)
    - Between consolidation and cardiomegaly, there are class-specific attention maps and XAI-based inductive bias

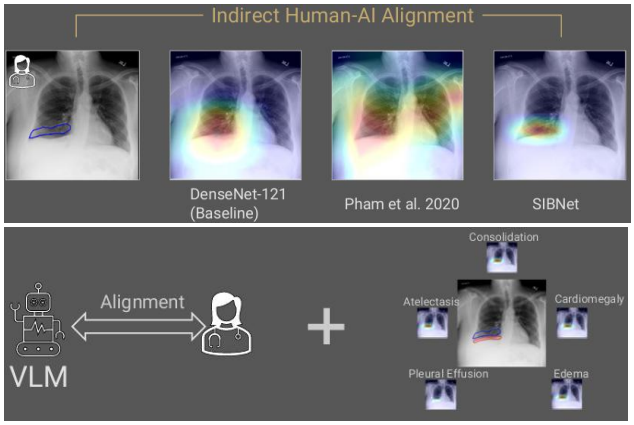


Saliency inductive bias: SIBNet

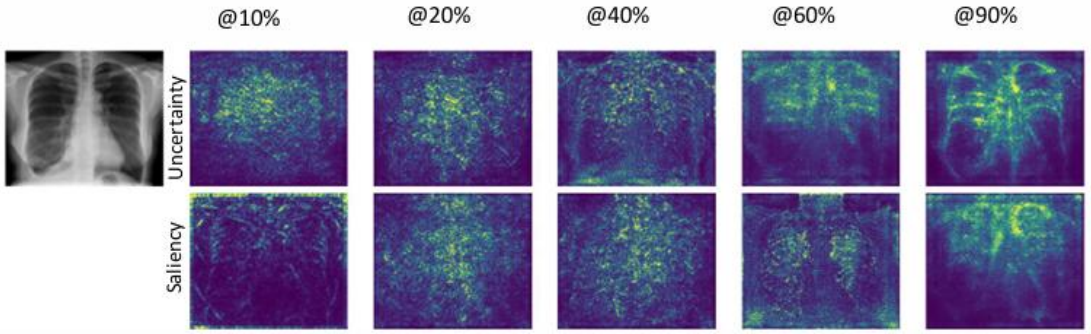
- Setup:
    - Five-fold validation
    - Five classes

- DenseNet-121 backbone

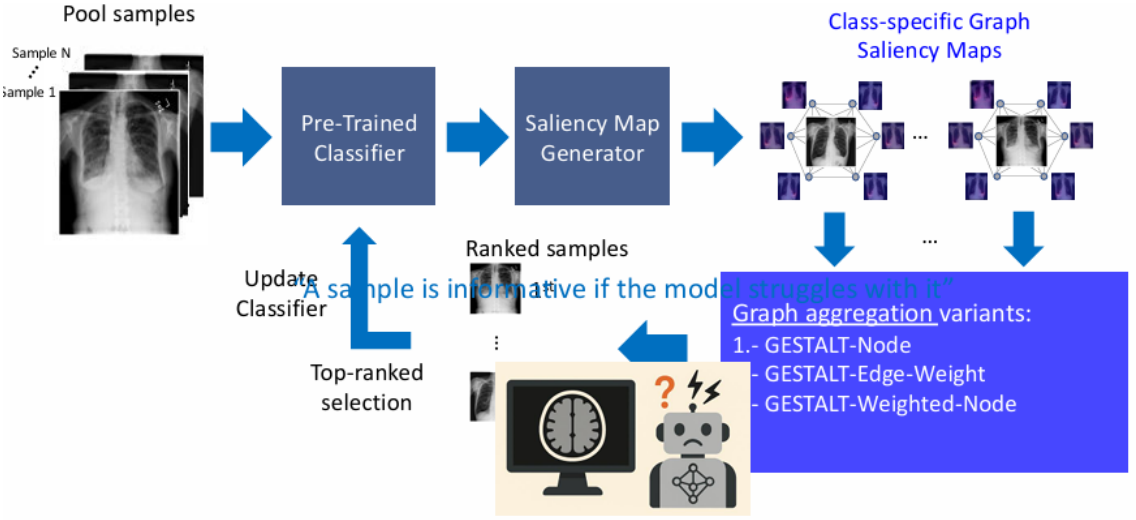


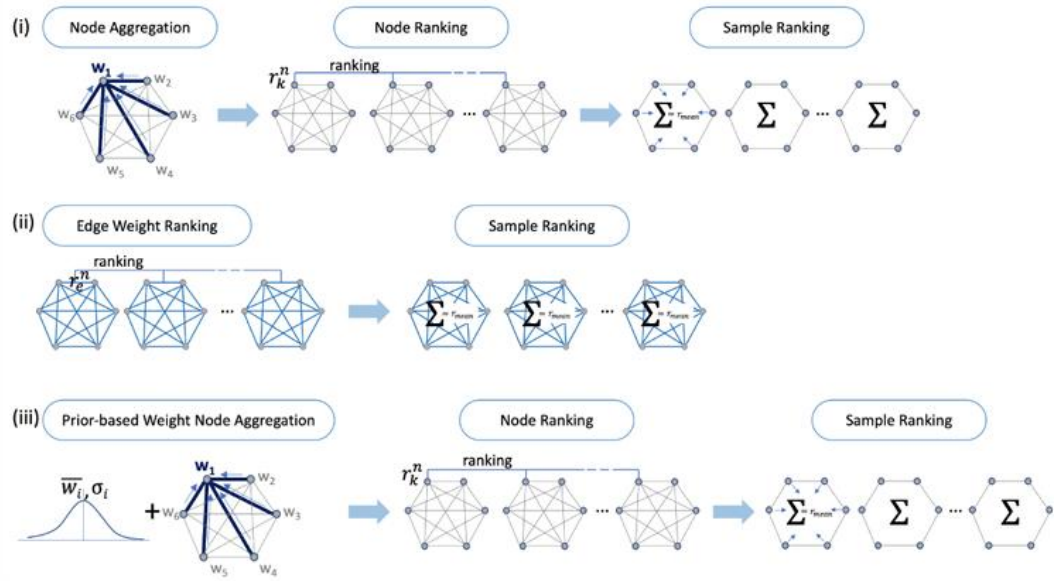


- Saliency maps evolve during training
  - Saliency maps can be seen / used as a fingerprint of model's behavior to input

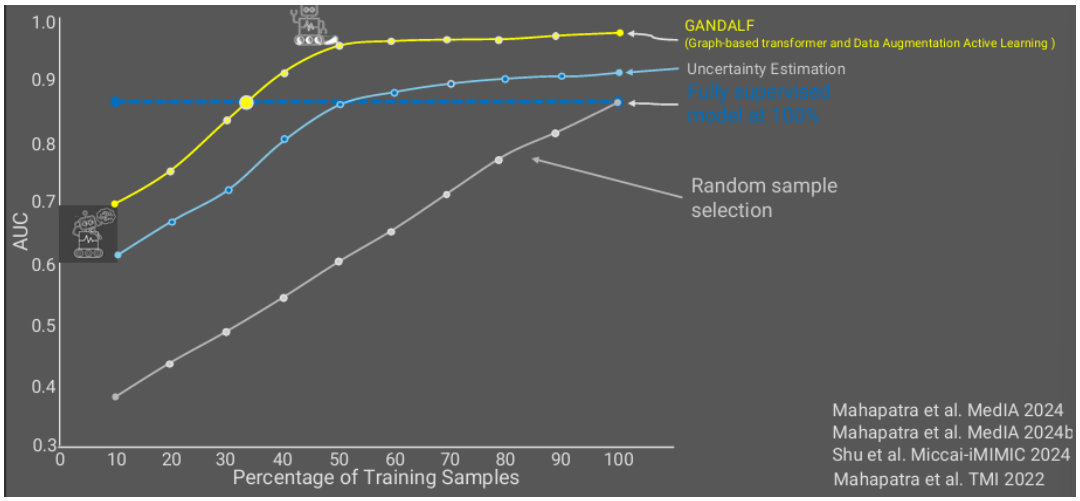


- Graph node-based interpretability guided sample selection (GESTALT)





- XAI-boosted active learning
  - When encountering a hard sample (informative), the machine can learn from it



Appendix 1: Intensity normalization summary

Problem source: Intensity variations between images; differences in scanners and acquisition devices; important for MRI due to lack of absolute intensities; less of a problem for modalities with absolute intensities.

Method	Description	Advantages & disadvantages
Min-max **(linear)**	- Matching the range - $f(j) = \frac{j - j_{min}}{j_{max} - j_{min}}(i_{max} - i_{min}) + i_{min}$ where i and j are the intensities of the two images - Intensities corresponding to ϵ% and $1 - \epsilon$% of the CDF are used	- Can solve global intensity shifts and multiplicative factors - Cannot solve contrast difference - Max and min values are sensitive to outliers
Mean and standard deviation **(linear)**	- Matching the second order statistics - $f(j) = (j - \mu_j)\frac{\sigma_i}{\sigma_j} + \mu_i$, where $\mu_i = \frac{1}{N}\sum_{x=1}^{N} i(x)$ and $\sigma_i^2 = \frac{1}{N-1}\sum_{x=1}^{N}(i(x) - \mu_i)^2$ - Less sensitive to outliers, but robust statistics, e.g., median, robust variance, may be used for better behavior	- Can solve global intensity shifts and multiplicative factors - Less sensitive to outliers - Cannot solve contrast difference
Histogram normalization with landmarks, i.e., Nyul's method (**non-linear**)	- Matching landmarks of image histograms - Piece-wise linear map (non-linear mapping) - Different landmarks can be used, e.g., min and max, mean or median, quartiles, deciles, models, etc. - Less sensitive to outliers - Histogram matching does not know about image content, all pixels are independent	- Can solve global intensity shifts and multiplicative factors - Less sensitive to outliers - Can solve contrast difference due to non-linearity of the mapping

Linear vs. Non-linear: Min-max and Z-score (mean/std) methods stretch the entire histogram evenly using a single straight-line equation. Nyul's method stretches different parts of the histogram differently (e.g., stretching the intensities between the 10th and 50th percentiles differently than the 50th and 90th percentiles). This makes the overall mathematical function non-linear.

Global vs. Local: Even though Nyul's mapping function bends and shifts (non-linear), that identical bending function is applied to the entire image at once (global).

Appendix 2: Noise suppression summary

Problem source: random noise and imaging artifacts due to imperfections in the acquisition device, fast acquisition, implants. We let J is the noisy image and I is the target image.

Method	Description	Advantages & Disadvantages
Convolutional (**linear**)	- $\tilde{I} = J \cdot S$, where S is structural element (kernel) and $\tilde{I}$ is the denoised image - Continuous: $\tilde{I}(x,y,z) = \iiint J(x-r, y-p, z-q)S(r,p,q)\,drdpdq$ - Discrete: $\tilde{I}(x,y,z) = \sum_r \sum_p \sum_q J(i-r, j-p, k-q)S(r,p,q)$	- Efficient to apply - Easy to implement - Blurs edges - Not context aware - Sensitive to outliers
Median filtering (**non-linear**)	- For each location, rank order all neighboring intensities, assign the median value as the result - Sliding window similar to convolution - Two rank filters are not commutative, i.e., min(median(x)) ≠ median(min(x))	- Robustness to outliers - Preserves discontinuities, such as edges - Easy and efficient implementations - Does not introduce new intensities (keep the intensities integer, applying median filtering to labels, in a post processing step, will not introduce new labels and remove island) - Patchy result - Image content can change, adding and removing structure
Non-local means (**non-linear**)	- Extension of bilateral filtering - Most noise suppression methods can be put in the form $(x) = \sum_{y\in\Omega} w(x,y)J(y)$ with $0 \le w(x,y) \le 1$ and $\sum_{y\in\Omega} w(x,y) = 1$ for all x - Weights $w(x,y)$ changes accordingly - Smoothing point x use areas with similar image content, if noise is random then averaging similar areas should get rid of it	- Best approach without training - Higher noise suppression - Better preservation of edges - Currently the state-of-art filtering technique that is not based on machine learning - May create artifacts
Optimization-based (energy-based, **non-linear**)	- $J(x) = I(x) + \epsilon(x)$ - $\hat{I}(x) = \arg_I \min \lambda D(I,J) + R(I)$, where $D(I,J)$ is data consistency, $R(I)$ is regularization, λ is weighting coefficient - Probabilistic interpretation: $\arg_I \min \lambda D(I,J) + R(I) = \arg_I \max \log P(J\|I) + \log P(I)$, where $P(J\|I)$ is the likelihood and $P(I)$ is the prior - $D(I,J)$ or $P(J\|I)$ encode information how the noise process work, can be different, do not have to additive Gaussian noise	

	- $R(I)$ or $P(I)$ encode our prior information on what type of images are plausible

Appendix 3: Bias correction summary

Problem source: Acquisition artifacts in MRI; difficult to see by eyes but adverse effects on algorithms; a bias field is smoothly varying, does not depend on the underlying tissue to the first approximation; math model $j = bi + n$, where j is the observed noisy image, b is the bias field, n is the noise, and i is the real image.

Basic bias correction methods

Method	Description	Advantage & disadvantage
Extra measurement at hardware side	Make a second empty image per image to find it and subtract it then. Post-acquisition methods are very useful	- Ideally a perfectly solution - Additional measurement makes it hard for retrospective data analysis and usually not integrated
Manually place landmarks	Voxels expected to have similar intensity	- Human in the loop - Need for human interaction
Joint segmentation	Same structure should have the same intensity in all voxels. Works if you know what you expect in the image	- Geared to get the best segmentation accuracy - Need for prior information and too specific

N3 algorithm

Algorithm 1 Bias correction with N3

$b = 0$, $j = \log(\tilde{j})$ observed image
$\hat{F} = \frac{\hat{B}^*}{|\hat{B}|^2 + Z^2}$ >=
while $\|\Delta b\| \leq eps$ **do**
  Estimate $J(j)$ with kernel density estimation or simple histogram
  Compute $I(i) = \mathcal{F}^{-1}(\hat{F}\mathcal{F}\{J\})$
  Compute $\Delta b_e = j - \mathbb{E}(i|j)$ at x_n anchor points.
  Get the smooth estimate $\Delta b(x) = \sum_{n=1}^{N} \Delta b_e(j(x_n))K(\|x - x_n\|)$ for all voxels
  Update $b = b + \Delta b$
  Update $j = j - \Delta b$
end while

Appendix 4: Interpolation model summary

Method	Description	Advantage & disadvantage
Nearest neighbor	- Assign the intensity value of the closest pixel with known intensity - assigning the value of the nearest neighbor - Nearest neighbor interpolation yields a piece-wise constant function, not continuous nor differentiable	- Nearest neighbor is very fast - Does not introduce new intensity values in the image - Good on label maps - May introduce heavy artifacts in gray level images (bad boundary)
Bilinear	- Uses closest four pixels' intensity values (extension of linear interpolation in 2D or trilinear interpolation) - Linear interpolation takes the distance, the closer the dot is to one point, the more weight this pixel value becomes - Leads to piece-wise linear functions, continuous but not differentiable - 4 unknown parameters (8 for trilinear) - $I(x_1, x_2) = \frac{x_2^1 - x_2}{x_2^1 - x_2^0} I(x_1, x_2^0) + \frac{x_2 - x_2^0}{x_2^1 - x_2^0} I(x_1, x_2^1)$ - $I(x_1, x_2) = a + bx_1 + cx_2 + dx_1x_2$	- Bilinear interpolation is **fast** - Introduces new intensity values (good for continuous figures, **not segmentation**) - May smooth and blur the image, leading to decrease in spatial resolution
Bicubic	- Beyond bilinear interpolation, it has a smoother function, not just piece-wise but also differentiable - Uses 16 closest points - Intensities at $p_{00}, p_{01}, p_{10}, p_{11}$ (4 values) - ∂x_1 and ∂x_2 at $p_{00}, p_{01}, p_{10}, p_{11}$ (8 values) - $\partial x_1 x_2$ mixed at $p_{00}, p_{01}, p_{10}, p_{11}$ (4 values) - Needs to compute the derivatives	- Better at retaining gray values - Best interpolation quality - Computation requirement is the highest - Bad on label maps

Note: Bilinear is a good trade-off between computation effort and interpolation quality.

Appendix 5: Linear transformation model summary

Method	Description
Linear mapping	- $x' = T(x) = Ax + t$, where the transformation is defined by the matrix A and the translation vector t - For 2D space: $x \in \mathbb{R}^2, x' \in \mathbb{R}^2 \rightarrow A \in \mathbb{R}^{2\times2}, t \in \mathbb{R}^2$, 4 + 2 = **6 free parameters** - For 3D space: $x \in \mathbb{R}^3, x' \in \mathbb{R}^3 \rightarrow A \in \mathbb{R}^{3\times3}, t \in \mathbb{R}^3$, 9 + 3 = **12 free parameters** - **Non-linear due to translation** - Same parameters apply to entire image - Number of parameters is independent of the number of pixels / voxels - Invertibility $T^{-1}(x') = A^{-1}x' + t', t' = -A^{-1}t, \forall x'$
Rigid transformation	- Only **translation** and **rotation** - For 2D space: $x' = T(x) = Rx + t$, $R = \begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}$, rotates the xy-plane about the origin counterclockwise by θ, 3 parameters - For 3D space: three angles $\theta_{xy}, \theta_{xz}, \theta_{yz}$, $R = R(\theta_{xy})R(\theta_{xz})R(\theta_{yz})$ - **Order of the rotations matters**
Similarity transformation	- Shape is preserved, **rotation**, **translation**, and **scaling** - For 2D space: $x' = T(x) = RSx + t$, $S = \begin{bmatrix}s & 0\\ 0 & s\end{bmatrix}$, 4 parameters - For 3D space: $S = \begin{bmatrix}s & 0 & 0\\ 0 & s & 0\\ 0 & 0 & s\end{bmatrix}$, 7 parameters - $x' = SRx$ is the same as $x' = RSx$ (**order does not matter**)
Affine transformation	- Scaling with different factors in different dimensions, angles and **shapes don't need to be preserved** - For 2D space: $x' = T(x) = RSx + t$, $S = \begin{bmatrix}s_1 & 0\\ 0 & s_2\end{bmatrix}$, 1 rotation + 2 translation + 2 scaling = **5 parameters** - For 3D space: $S = \begin{bmatrix}s_1 & 0 & 0\\ 0 & s_2 & 0\\ 0 & 0 & s_3\end{bmatrix}$, 3 rotation + 3 translation + 3 scaling = **9 parameters**
Affine transformation with shearing	- Extend the mapping with anisotropic scaling with **shearing** - For 2D space: $x' = T(x) = WRSx + t$, $W = \begin{bmatrix}1 & w\\ 0 & 1\end{bmatrix}$, 1 rotation + 2 translation + 2 scaling + 1 shearing = 6 parameters - For 3D space: $W = \begin{bmatrix}1 & w_1 & w_2\\ 0 & 1 & w_3\\ 0 & 0 & 1\end{bmatrix}$, 3 rotation + 3 translation + 3 scaling + 3 shearing = 12 parameters - Rarely used

Appendix 6: Non-linear transformation (kernel) summary

$x = T^{-1}(x') = x' + u(x')$, where the transformation is defined by the function $u(x')$. This function can be non-linear. Radial basis function $u(x') = \sum_{n=1}^{N} \phi(|x' - x'_n|)d_n$, $\phi: \mathbb{R}^+ \to \mathbb{R}$, $d_n \in \mathbb{R}^2$ or $d_n \in \mathbb{R}^3$, where x_n are the control points, ϕ are the basis functions and d_n are the non-linear coefficients, defining the transformation.

Method	Description
Thin plate splines (TPS)	- $\phi(\|x' - x'_n\|) = \phi(r) = r^2 \log r$ - Can also be defined with affine transformation as $x = Ax' + t + \sum_{n=1}^{N} \phi(\|x' - x'_n\|)d_n$ - Control points (given by the user) do not have to be uniformly spaced, they can be randomly dispersed - For 2D space: (number of control points) * 2 - For 3D space: (number of control points) * 3 - Increasing the control points can lead to more complicated transformations - Mostly used for landmark-based registrations, not intensity-based registration
Free form deformation with B-splines	- Same radial basis function, the kernel is different - For 3D space: there are $N_x \times N_y \times Nz$ control points placed in a uniform grid with spacing δ - For 2D space: (number of control points) * 2 - For 3D space: (number of control points) * 3 - Every pixel is influenced by 3×3×3 control points, needs many control points - For every pixel, find the closest control point (its basis), compute distance pixel to basis, plug those values into the interpolation stuff - Choose the kernels so that the derivatives are continuous - Leads to smooth transformations (only local changes) that can also apply non-linear deformations to the objects in the image - Increasing the number of control points, i.e., using a finer grid, can lead to less smooth transformations - FFD with B-splines is mostly used for intensity-based registration not necessarily for landmark-based

Appendix 7: Landmark based registration summary

Method	Description
Linear registration	- Given the landmark-based loss function: $L(\theta) = \sum_{n=1}^{N} \|\|x_n^{(1)} - T_\theta\left(x_n^{(2)}\right)\|\|_2^2, x \in \mathbb{R}^d$ - When aligning with linear transformation, T_θ is defined through a matrix A and a translation t: $L(A,t) = \sum_{n=1}^{N} \|\|x_n^{(1)} - \left(Ax_n^{(2)} + t\right)\|\|_2^2, x \in \mathbb{R}^d$ - The loss is minimized to determine the best transformation parameters A and t: $\arg_{\mathrm{A,t}} \min \sum_{n=1}^{N} \|\|x_n^{(1)} - \left(Ax_n^{(2)} + t\right)\|\|_2^2$
Non-linear alignment with thin plate splines	- Aligning with TPS - Given the landmark-based loss function: $L(\theta) = \sum_{n=1}^{N} \|\|x_n^{(1)} - T_\theta\left(x_n^{(2)}\right)\|\|_2^2, x \in \mathbb{R}^d$ - When aligning with TPS, the cost function becomes: $L = \sum_{n=1}^{N} \|\|x_n^{(1)} - x_n^{(2)} - \sum_{k=1}^{N} \phi\left(\left\|x_n^{(2)} - x_k^{(2)}\right\|\right) d_k \|\|_2^2, x \in \mathbb{R}^d$ - Note that the displacement vectors are defined through the same landmarks, similar to linear, the loss is minimized to determine the best transformation parameters: $\arg_{\{d_k\}_{k=1,\ldots,N}} \min \sum_{n=1}^{N} \|\|x_n^{(1)} - x_n^{(2)} - \sum_{k=1}^{N} \phi\left(\left\|x_n^{(2)} - x_k^{(2)}\right\|\right) d_k \|\|_2^2$ - After determining the optimal parameters d_k^* one can use it to find the displacement vectors everywhere in the target image, not just the landmarks: $T_\theta\left(x^{(2)}\right) = x^{(2)} + \phi\left(\left\|x_n^{(2)} - x_k^{(2)}\right\|\right) d_k^*$

Appendix 8: Intensity based registration summary

Intensity based loss function $L = d(I, T_\theta \circ J)$, where $d(\cdot,\cdot)$ is defined over the intensities of the fixed image I and the moving image J after transformation, i.e., $T_\theta \circ J$.

Method	Description
Sum squared distance (**intra-modality**)	- When the images are the same, the SSD is 0 - It assumes the intensities are comparable across the images, i.e., **same modality** - Image gradients are important for SSD - Can use lasso for robustness
Normalized cross correlation **(inter-modality)**	- NCC is the 2D version of the Pearson correlation coefficient - It assumes pixel **intensities** as **independent random (iid)** variables - **Maximum score is 1** if $I(x)$ and $J(T_\theta^{-1}(x))$ are linearly related - NCC can be both positive and negative, that's why the loss is usually defined as NCC squared - NCC **does not go down to -1** (the relationship never perfectly linear in reality)
Mutual information (**inter-modality**)	- $L(\theta) = d(I, T_\theta \circ J) = MI(I,J)$ - $MI(I,J) = D_{KL}\big(p(I,J) \big| p(I)p(J)\big)$ - It assumes pixel intensities as **independent random (iid)** variables - MI takes the value 0 if $p(I,J) = p(I)p(J)$, when the intensities in the two images are **independent** from each other, very different when MI is low - **Not aligned** images will have **low MI** - **Perfectly aligned** images will **maximize MI**

Note: MI is more powerful, which can capture complicated relationships, but on the same time it is hard to compute.

Appendix 9: Segmentation summary

Method	Description
Thresholding (**basic method**)	- $L(x) = 1$ if $I(x) > \tau$, $L(x) = 0$ if $I(x) \leq \tau$, where $I(x)$ is the intensity of the pixel / voxel - Use histogram analysis to determine the threshold, cut when there is a **low part** in the histogram, too high might have noise and not capture enough information - Useful in easy foreground-background subtraction
K-Means (**basic method**)	- Unsupervised clustering algorithm, assigning pixels to clusters with the **closest distance to the means** - Cluster pixels / voxels according to features such as intensity, colors, temporal, sequence, gradient, etc. - Choice of **number of clusters has major influence**, **initialization** is important - Easy to implement - Probabilistic formulation is possible, more robust - Can get stuck in local minima
Multi-atlas (**atlas-based method**)	- Atlases are **prior information** on segmentation, providing information for each location - For each atlas, we do registration, then for each segmentation based on atlas, we need to have a **fusion and merge** to the final segmentation - **Align** all the atlas image to the **test image**, **aggregate** information coming from all the atlases - Many atlases **reduce importance of a single** - Less likely to have bad registration with weighing scheme - Leads to elegant probabilistic models - Highly accurate results - State-of-the-art method - Often require non-linear registration - Computationally **very expensive** due to many registrations - Merging methods include global fusion (majority voting, weighted voting), local fusion (locally weighted fusion, STAPLE) - STAPLE reduces the number of registrations by **choosing the most similar atlases** and register these
Patch-based method	- Start with **denoising** the image, applying the idea in denoising to label fusion, **similar intensities have similar labels** - First create subject-specific atlas, then get the local spatial consistency, creating the complete segmentation - The subject-specific atlas as **prior** for looking at patches
Active shape model	- **Statistical shape model** (SSM) is the **blueprint**, **active shape model** (ASM) is the **builder**

	- SSM defines the average shape of an object and the statistical variations - Principal component analysis (PCA) and singular value decomposition (SVD) are used - PCA is used to **reduce the dimensionality** of a **linear** system, maximizing the variance of the points (can be reducing the number of landmarks) - SVD can yield the most important eigen-vector - ASM idea: **1)** Start with SSM and image; **2)** Initially place SSM in image; **3)** Sample along SSM surface normal, find largest gradient, **4)** Deform pose (scale, rotation, translation) and shape (PCA modes) of models to best fit to extracted image features (largest gradient)

Appendix 10: Convolutional neural network summary

Important properties of CNN:

- Translation equivariance
  - A function f is **equivariant** with respect to a transformation T if $f\big(T(x)\big) = T\big(f(x)\big)$
  - Required for segmentation and object localization
  - Data augmentation can encode equivariance
- *Approximate* translation invariance
  - A function f is **invariant** with respect to a transformation T is $f\big(T(x)\big) = f(x)$
  - The pooling operation in image classification CNNs lead to partial translation invariance

CNN hyperparameters

The naïve CNN have the number of parameter as $N_{in} \times k_1 \times k_2 \times N_{out} + N_{out}$, where $k_1 \times k_2$ is the kernel, N is the number of channels (gray scale $N = 1$, RGB $N = 3$)

Operation	Description
Padding	- To deal with boundary pixels, add pixels surrounding the input original image - If input is $m_i \times n_i$, kernel is $k_x \times k_y$, output is $m_{i+1} \times n_{i+1}$, where $m_{i+1} = m_i - k_x + 1$ and $n_{i+1} = n_i - k_y + 1$
Stride	- How far we shift the kernel across the image - When stride > 1, it leads to a **reduction in the size of output** as compared to the input - With stride to be s_x and s_y in the two image directions, we have the reduction size $m_{i+1} = \frac{m_i - k_x}{s_x} + 1$ and $n_{i+1} = \frac{n_i - k_y}{s_y} + 1$ - Same padding and stride with 1, the input and output have the same size
Pooling	- Pooling is an operation that enables "pooling" information in a neighborhood, applied **to each channel separately** - Linear pooling: average pooling - Non-linear pooling: max pooling, min pooling - Pooling stride = pooling kernel = 2, we will have half of the input, dimension reduced

Appendix 11: Pixel-wise predictions summary

	Segmentation	Restoration	Synthesis
Task	Assigning labels to each pixel, indicating the structure the pixel belongs to	Removing noise from an image, image super resolution, bias field removal, etc.	Synthesizing one image from another one, synthesizing a **target** image from a **source** image
Data set	- Input images and segmentation **masks** - Large number of images, **the more the better** - *Challenges*: **ambiguity in labeling**, missing data, images come from different domains	- Input images and "ground truths" - Large number of images, **the more the better** - *Challenges*: missing data, images come from different domains, **ground truth labels may not be available**	- Input images and "ground truths" - Large number of images, **the more the better** - *Challenges*: **ambiguity in labeling**, missing data, images come from different domains, **unpaired labels and features**
Cost function	- (Binary) cross entropy: losses of small structures may be affected by larger structures - Sorenson-Dice coefficient: not differentiable, need to use probability	- Basic cost functions: MSE, MAE, NMSE, PSNR, SSIM, **cannot capture perceptual differences** - Perceptual loss: compare the prediction and truth at every layer - Adversarial loss: use distributional distance over sets of samples, use discriminator to identify the truth and prediction	*Similar cost functions as restoration*
Architectures	- CNN: detail might be lost due to pooling - FCN with hierarchical links: aggregating information from different layers - UNet: having encoding and decoding paths, skip connections retain high-resolution information	- **Fully convolutional networks (FCN) and UNet** are used very often - Residual architectures: **add restorative information as residual** to the **original** or **naively restored** image	- For **paired** problem: UNet can be used, with any types of loss function, **no activation** in the last convolution layer - For **unpaired** problem: total loss takes into account **adversarial terms** and **cycle consistency loss**, still **does not fully solve the correspondence problem**

Appendix 12: Domain adaptation summary

Problem with generalization: models train with one training set **does not perform well** on another data set with **different, even slightly, intensity characteristics**, called **domain shifts**.

Method	Idea	Advantages & disadvantages
Naïve solution	**Separate training** for each domain with **separate training set** and final **model**. Inference at each domain with domain-specific model.	- This is the **best** performing model if we have **enough labeled examples** - Requires **large set of labeled samples** at **every domain**
Transfer learning	**Initial training** on the source domain with **many samples**. **Fine-tune** the model at the **target domain** with **few samples**.	- Only need a **few labeled samples** at the **target domain** - At each target domain, it **still requires labeled samples for fine-tuning**
Unsupervised domain adaptation	Train models **without any labels** at the target domain. Nothing really interesting happens at the source domain.	- Does **not require any labeled images** at **target domain** - A **new training** at **each target domain** and **source domain** images need to **transported to each target domain**
Extreme data-augmentation	**Transformations** of the data samples serve as a new sample. Transformations **do not affect** the labels.	- Very effective way to tackle lack of data samples - The hypothesis is that **augmentation can mimic new domains**

Unsupervised source-free domain adaptation	**Cost functions to optimize**
	- **Entropy of the prediction**:
	  - Simple to implement, no prior needed, applicable to different tasks
	  - Strong assumption, i.e., higher confidence of prediction leads to good domain adaptation
	- **Auto-encoder loss of input, output, and intermediate features**
	  - Auto-encoders are **trained with source domain**, differences between inputs and outputs are the **cost**
	  - Applicable to different tasks and strong prior model
	  - Input and intermediate features can be prone to domain shifts themselves
	- **Segmentation prior evaluated at the output assessing plausibility of the result**
	  - A denoising auto-encoder (DAE) is a map from a **noisy version of an image to itself**
	  - Trained on segmentation maps to learn a **prior model** on the segmentations
	  - Strong prior leads to accurate segmentation
	  - Segmentations are not affected by intensity changes
	  - Specific to segmentation task
	Parameters to optimize	

	- Train / fine-tune **all the parameters**
	  - Model will be very **flexible**
	  - Too much flexibility can lead to a result that **perfectly satisfies the cost** but the output is **no longer related to the input**
	- Train / fine-tune **a part of the parameters**
	  - Model will be less flexible so that issue with too much flexibility will not be an issue
	  - Initial model parameters are obtained using large curated data sets. It is a desirable **not to change** them if not needed. It was costly to train them in terms of time, data, and money
	- Train / fine-tune **batch-norm parameters**
	  - Very **simple application** and very **few parameters**
	  - Batch norm parameters in **deeper layers** can **affect results a lot** and it is **unclear** if they address **changes in intensity** characteristics
	- Tran / fine-tune **a shallow "adaptation" module prepended to the network**
	  - **Shallow module will not be too flexible**, and we **do not modify pre-trained parameters**
	  - We make the network **slightly bigger**

Appendix 13: Learning from unlabeled examples summary

We do not have too many labeled examples, but we have access from unlabeled examples

Method	Idea	Advantages & disadvantages
Noise-to-noise	Learning to denoise without clean target data	- Simple cost function - In some cases, it works better than supervised versions - It is a probabilistic and statistical method, interpretable - It is noisy, removing some edge details
Self-training	Train with labeled data with ground truth, **infer pseudo-labels** for unlabeled data with trained model, retrain with labeled data with ground truths and unlabeled data **with pseudo-labels as ground truths**	- If the initial model's estimates are not bad, self-training works well - Entropy minimization **assumes mostly correct class assignments** - Additional regularizations on the final estimates can prevent divergence - Even with the regularization terms, the model can diverge quite often
Teacher-student models	A new type of regularization - train **two interacting networks** for **generating pseudo-labels**. One evolving **slower** than the other. A cost for labeled examples and a consistency loss. - Can be applied to both labeled and unlabeled examples - Given a teacher network, train the student network - Update teacher network's parameters - Iterate the above two steps	
Semi-supervised learning	- **Semi-supervised auxiliary loss**: defined over unlabeled images during task-specific optimization - Consistency under transformation - Construct an auxiliary loss for unlabeled examples for segmentation	
Self-supervised learning	- Train a network with pre-text task that only requires images. The trained network will be suitable to fine-tuning with very few labeled examples to other tasks - **Self-supervised auxiliary loss**: defined over unlabeled images independent of the task - It is very important to identify a **pre-text task** that will lead to useful network parameters - Such tasks should be good for learning "generalizable" parameters - These tasks are **hand-crafted**	