Intro

Lots of applications and uses of Computer Imaging. For MNIST digit recognition, simple thresholding / defining explicit rules isn't enough for great results at distinguishing digits. We want to be able to learn the rules from data!

- Images are just arrays of pixels (possible 3 channels, or 1 - grayscale). Input image $x = [x_1, x_2, ..., x_n]^T$; xs are the columns of the image.
- $\Theta^T x = \Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \cdots + \Theta_n x_n$... this has many parameters - 785 if we have 28 pixels in our image! (28^2 + 1 for bias).
  - We can reduce number of input features in other ways - by capturing different features than raw pixel values. e.g: num white pixels, digit width, height, slant, thickness, length. Using all of them we can make a linear model: x = [length, thickness... height]^T; $\Theta^T x = \Theta_0 + \Theta_1$length..
- Logistic Regression: $h_\Theta(x) = 1 / (1 + e^{-\Theta^T x})$
- CNNs: apply convolutions using a filter / kernel with weights to the input producing a feature map that's fed into the layers (with a bias term) and an activation function (for non linearity). $Z = ReLU(w^T x + w_0)$
- 3 rows of [-1 0 1] is used for computing x gradient / edge detection, 3 columns of [-1 0 1]^T is used for computing y gradient / edge detection.
  - Convolutional filters are translation equivariant! Key for images
  - Equivariant - changes the output same way it changes input.

Padding and Strides: For an Input Size (M) we have the following terms:
- Padding (P) - how many extra values are put around edges in the conv. Useful if we don't want to decrease output size despite doing a conv.
- Stride (S) - how much the kernel is moved across after each application
- Kernel Size (K). The below formula assumes square sized kernels.
- *Output Size* = [(M + 2P - K) / S] + 1

Pooling: reduces the size of feature maps without doing learning. Allows for some local translation invariance. We define kernels (typically with stride = kernel width so there's no overlapping). e.g: *MaxPooling, AveragePooling*
AlexNet: 224x224x3 input, then maxpools & convs, until we get 7x7x512, then 2 fc layers to make 1000 outputs (as it's image classifier for 1000 categories)
Building Novel Detection Systems (Skin Cancer):
- Collect Data to try self supervised learn features OR consult experts to get ideas of features to extract.
- Start from a model with initialised weights (a classifier) and train that - *Transfer Learning* by finetuning. Training just the classification head!

Segmentation

Semantic Segmentation:
- Segmented regions are also assigned a semantic meaning.
- Different to segmentation based on 'pure' clustering of the image into coherent regions. For imaging, we care about Semantic Segmentation.
- Useful for 1) *quantitative analyses*, e.g. measuring the volume of the ventricular cavity. 2) determining the precise *location* and *extent* of an organ or tissue type e.g: a tumour, for treatment such as radiation therapy. 3) creating 3D models used for simulation, e.g. generating a model of an abdominal aortic aneurysm for simulating stress/strain distributions.

Challenges of Segmentation: noise, partial volume effects, intensity inhom., anisotropic res, imaging artifacts, limited contrast, morphological variability
- *Partial Volume Effects* - b/c of coarse sampling the result im. shows partial volume effects at the boundaries: both colours (black and white) contribute to the intensity val of boundary pixels b/c large influence area of each pixel.
- *Intensity Inhomogeneities*: diff parts of im have diff contrast (sun)
- *Anisotropic Resolution*: most 3D images have this (MRI, CT). MRI acquisition of short axis slices of the left ventricle may have intraslice resolution of 1.3mm and an inter-slice resolution of 8mm
- *Imaging Artifacts*: e.g: from metal teeth, disrupts the CT scan
- *Limited Contrast*: diff tissues have similar phys properties and thus similar intensities. Intensity based algos can fail or leak into adjacent tissue.
- *Morphological Variability*: hearts/livers vary in shape person to person!

Ground Truth: The reference the method can be compared against. There's not really a ground truth segmentation (except physical phantom/simulation)
Gold Standard: The best truth we can have - often manual segmentation by experts. But reqs training, tedious, takes time, intra/inter observer variation.
Measuring Performance: We overlay the segmentation map (SM) & gold standard (GS). We then easily see regions of agreement and disagreement.
SM & GS = TP, SM & ~GS = FP, ~SM & GS = FN, SM & ~GS = TN.
We can then construct the *Confusion Matrix*: [[TP FP], [FN, TN]]. FP = type 1 error, FN = type 2 error. From here we can compute *performance metrics*:
1. *Accuracy*: (TP + TN) / (P+N) 2. *Precision* (+v predictiv val/PPV): TP + (TP + FP)
3. *Recall* (Sensitivity / Hit Rate / TPR): TP / P = TP / (TP + FN)
4. *Specificity* (True Negative Rate): TN / N = TN / (TN + FP)
5. *F1 Score*: HMean of Prec&Rec: 2(PPV*TPR)/(PPV+TPR) = 2TP/(2TP+FP+FN)
6. *Jaccard Index*: |A∩B| / |A∪B|
7. *Dice Similarity Coefficient (DSC)*: 2|A∩B| / (|A|+|B|). Dice can be worked out from Jaccard, & DC & F1 are equiv. (|A| = TP+FN, |B| = TP + FP
8. *Volume Similarity*: 1 - ||A| - |B|| / (|A| + |B|) = 1 - |FN-FP| / (2TP+FP+FN)
9. *Surface Distance Measures*
   a. *Hausdorff Distance*: max(h(A,B), h(B,A)); $h(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\|$
   Intuitively - the largest difference between the two surface contours
   b. *Average Surface Distance* = (d(A,B)+d(B,A))/2; $d(A,B) = \frac{1}{|A|}\sum_{a \in A} \min_{b \in B} \|a - b\|$
- Both can be computed efficiently by converting the segmentation images into *distance maps* where each pixel's value is its distance to the boundary.
  - Swap the contours of the two images and walk along the contour edge, for Hausdorff take the maximum value on the edge, for ASD just mean.
  - Distance maps have the nice side effect of letting you easily generate the contour formed if the boundary changes efficiently

Pitfalls in Segmentation Evaluation:
1. *Effect of Structure Size*: For the Dice Coefficient, a 1 pixel difference has a much bigger impact on a small than big image. This is problematic if our metric aggregates over many structures of different sizes. Dice is quite invariant to different structure shapes; but very sensitive to pixel offsets
2. *Effect of Annotation Noise*: The gold standard might have a label error - if there's a pixel marked somewhere far away, massively impacts Hausdorff
3. *Effect of "empty" labelmaps*: Might have nothing labelled - then TP, FN = 0, and we get NaN (0/0) for a lot of our metrics! Needs explicit handling.
4. *Effect of resolution*: Affects calculations like DSC, Hausdorff...

Segmentation Methods:
1. *Simple Thresholding*: Select a thresh to split intensity range into 2 classes. Can use multiple thresholds for multi class, or upper/lower for one class that's within the interval and the other is outside of it. *Adv*: Simple, fast. *Disadv*: regions must be homogenous & distinct; finding consistent thresh across images is hard; leakage, isolated pixels & 'rough' boundaries likely.2
2. *Region Growing*: Start from (user selected) seed point(s), and grow a region according to an intensity threshold. *Adv*: relatively fast, yields connected region (from a seed point). *Disadv*: regions must be homogeneous, leakages and 'rough' boundaries likely, needs (user) input for SPs.
3. *Atlas Based Segmentation*: Generate template of an example segm & store those (in "Atlas") used for future segm. Has geom info abt points, curves or surfaces, or label info about voxels (anatomical regions or function). Usually constructed from examples: single subjects, populations of subjects, e.g. averaging to produce probabilistic atlases
   a. We segment via *label propagation*: check atlas (alr. has expert segms) & align them to test images via *registration*. If you can establish spatial corresp. b/w the ims you can prop info onto the test images. To aggreg info: label fusion from multi atlas (from multiple examples) propagation. i.e: each atlas "votes" for what it thinks the segmentation is, label fusion just picks the overall label via majority voting or other metrics.

Adv: Robust, acc. Makes plausible segm. Automatic. Disadv: expensive. Doesn't deal well with abnormalities. Doesn't work well for tumour segmentation.

4. *Segmentation via Dense Classification*: We can used trained CNNs (as we've seen) to perform dense (pixel by pixel) classification on the image. *Classical Approach*: Patchify im, do convs (e.g: dim 5 kerns); w/FC layer & 2 neurons at end. We patchify into 17x17(x3); 4 5x5 kern convs results in 1x1 outputs. *Problem*: Overlapping patches; the FC layer at the end is *very inefficient* as we must combine all patch results! *Better Approach*: LeNet.

Receptive Field (of a decision) - set of pixels that influence that decision LeNet (and any other CNN) can only take in images of the size they were designed for if we use FC layers. We can fix that by replacing FC layers with a *Fully Convolutional Layer* which have kernel size exactly equal to input size. Feeding a 32x32 into LeNet ends up with a 1x1 classification answer. Feeding a 64x64 ends in a 9x9 *classification map*, which has both classification and localization info (where high activation is = where object is!)

```
class LeNet(nn.Module):
  def __init__(self):
    super(LeNet, self).__init__()
    self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
    self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
    self.fc1 = nn.Linear(16 * 5 * 5, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)

  def forward(self, x):
    x = F.max_pool2d(F.relu(self.conv1(x)), 2)
    x = F.max_pool2d(F.relu(self.conv2(x)), 2)
    x = x.view(x.size(0), -1)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return F.log_softmax(x, dim=1)

class FCLeNet(nn.Module):
  def __init__(self):
    super(LeNet, self).__init__()
    self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
    self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
    self.conv3 = nn.Conv2d(16, 120, kernel_size=5) s
    elf.conv4 = nn.Conv2d(120, 84, kernel_size=1)
    self.conv5 = nn.Conv2d(84, 10, kernel_size=1)

  def forward(self, x):
    x = F.max_pool2d(F.relu(self.conv1(x)), 2)
    x = F.max_pool2d(F.relu(self.conv2(x)), 2)
    x = F.relu(self.conv3(x))
    x = F.relu(self.conv4(x))
    x = self.conv5(x)
    return F.log_softmax(x, dim=1)
```

Upsampling: We can do AverageUnpooling or MaxUnpooling - average unpooling has each pixel duplicated by unpool kernel size, top-left MaxUnpool has the pixel duplicated once into top left of the kernel and the rest zeroed. *Unpooling with spatial information*: you record which pixel indices were the maxpool donors. Then after some intermediate calcs, we put the intermediate results back into those specific indices. A form of *Encoder-Decoder* net.

Encoder-Decoder Networks: Have high res input, convs and downsampling, then we reupsample at the end with unpooling. *VGG-16* instead uses maxpooling to downsample and unpooling to upsample. *U-Net*: has "skip layers" that copy and concat the features from the downsampling stage to the upsampling stage before they are convolved - producing better upsamples.
Transpose Convolutions: Convolutions can be seen as a learned way of downsampling. Transpose convolutions are a learned way to upsample - i.e: for an input size of 2x2 we can have a kernel size 3x3 and end with a 4x4 result.
Dilated Convolutions: Dilation of 1 means each pixel in the receptive field of the convolution has a gap of 0 to the next either horizontally or vertically.
Atrous Spatial Pyramid Pooling: Stacks lots dilated convolutions of multiple sizes within a single layer to gain lots and lots of context at once!
Padding obviously introduces artifacts.
Multi-Scale Processing: To help the network see more context, we can add pathways which process down-sampled images of a bigger patch, and concat them to the pathway which processed the normal sized patch.
Vision Transformers: uses Multi-Head Attention for images by patchifying. Each gets a seperate embedding. Uses the patch for a linear projection before adding on the position embedding before encoding & classifying w/ MLP head. We also pass in a special token - the class token to the encoder that summarises patch info. Learnable. Enc. involves: input: A -> norm: B -> multi-head att: C -> concat A&C: D -> norm: E-> MLP: F -> concat D&F: Output
Transformers: patchify->linearly proj patches->transformer enc. -> dec. -> out
Segment Anything model produces segment masks for all detectable objects.

Self-Supervised Learning

Self-Supervised Learning: Goal - building pattern recognition into networks without supplying the labels. Why? - supervised classifiers require *lots* of data, and in domains like healthcare the labelling process is very expensive. We do however have lots of unlabelled images. We want to learn w/o labels!
- Supervised Learners from their encoders outputted image representation h. We want to learn the repre h, that summarises the main information in the image without needing any external labels. We need to create "proxy" tasks that only require the images to create the supervision signal

Contrastive Learning: Teaching the network to recog. meaningful im. pairs
- Main idea: a good repre. will put similar ims. close together in emb. space
- An artificially perturbed version of the same image, should have repre closest to the original image compared to all other images. This is the key principle of Contrastive Learning: create multiple versions of the same image and teach the network to recognise the correct pair
- *SIMCLR* - implements this. *Encoder* - produces h for downstream tasks. *MLP* - produces a smaller repre z to compute the contrastive loss.
  - *Pipeline*: Take image, augment it to produce copy. Encode both, MLP both, produce two zs - compute contrastive loss on zs - the two similar images need close z, but the further ones should have distant z. This is our training objective
  - *Augmentations* must be sophisticated enough. They need to reflect what information the model should disregard and what it should focus on. Needs to be hard enough, otherwise trivial information is learned
    - Rotate, crop, flip, colour distort, cutout (mask), blur, noise, sobel
    - If the augmentation isn't hard the model will learn colour, bg, etc
  - Keep in mind we don't want to learn b/w the two images that it's e.g: a cat but that it is THIS cat. We push other cats further from our pair
  - *NT-Xent loss* (normalised temperature contrastive loss):
    - Similarity sim(u,v) = (u^T v) / (||u|| ||v||)
    - For each positive pair (i,j) we have loss:
      $l_{i,j} = -\log \left(e^{sim(z_i, z_j)/\tau}\right) / \left(\Sigma_{k=1 \text{ to } 2N} 1[k \neq i] e^{sim(z_i, z_k)/\tau}\right)$
    - To min loss we need to max the above -> max nuner, min denom
      - Maximizing numerator: pulling the positive pairs closer
      - Minimize denominator: push the negative pairs apart
      - The final loss is computed across all positive pairs, both (i,j) and (j,i), in a minibatch, by averaging all $l_{i,j}$
    - Temperature τ controls how much to penalise hard negatives (negative pairs wrongly mapped close to each other). A low temperature penalise them more.
  - There's other losses too: triplet loss:

$$\mathcal{L}_{\text{triplet}}(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-) = \sum_{\mathbf{x} \in \mathcal{X}} \max\left(0, \|f(\mathbf{x}) - f(\mathbf{x}^+)\|_2^2 - \|f(\mathbf{x}) - f(\mathbf{x}^-)\|_2^2 + \epsilon\right)$$

  - *x* is one image, *x+* is the corresp pos and *x-* a randomly selected neg image. ε is the margin parameter controlling how far the negatives should be compared to the positive pairs (like temperature τ).

Evaluating Self-Supervised Models:
- Check perf on d-stream classification task or use (smaller) labelled dataset
- *Linear Probing*: 1) pretrain the encoder with selfsupervised learning e.g. SimCLR. 2) load the pretrained encoder, frz all the weights of the encoder. Train classif head on top of the frozen encoder with cross-entropy loss.
  - For finetuning, we *don't* freeze the weights.
  - For SimCLR to learn a good representa., very large batch sizes are crucial - if we only have 4 ims in a batch its easy to see what the pairs are. 4096.
  - SimCLR requires very large batch size to provide good representation → v high computational reqs (expensive, difficult to train). Bootstrap Your Own Latent (BYOL) or DINO are alternatives to reduce the cost:
    - *BYOL*: removes the need for neg pairs in loss fns, one just optimises the similarity on the positive pairs. So you don't need as big batches.
      - Introduces a new training paradigm to avoid learning trivial embs
      - Has a diff network arch: the input im is put into a "student" net that learns weights by gradient descent, and a "teacher" net that is the moving average of the weights of the student net.
        - Teacher updated slowly - i.e: 99% of old weight + 1% of new.
        - Other than that it's the same - we put the image and aug image to the view and teacher respectively.
      - Our loss maxes the similarity between positive pairs embs (output of student prediction and teacher projection).
      - BYOL is more robust to smaller batch size than SimCLR
    - *DINO*: similar approach to to BYOL, w/ teacher-student. Instead they use a softmax each then cross entropy-approach both for loss.
      - *Multi-Crop Training*: Instead of simple "pairs" for training, they create multiple local views (small crops of the image) and two global views (big crops). All local and global crops only pass through the student, but the teacher only sees the global crops
      - DINO uses a transformer-based encoder, showing better results on ImageNet over ResNet (see *VisionTransformers*)
      - Thanks to the ViT architecture, one can visualise the atten maps of the network for generating self-supervised seg maps w/DINO

Generative Approaches: Another way to use the image as supervision signal is to teach the network to reconstruct the original image from a masked image.
- Instead of recognising pairs of images, learn how to fill in the gaps (MAEs)
- Uses patch-wise, transformer-based encoders and decoders
  - Patchify -> encode unmasked: h -> add masked tokens -> decode all
  - The encoder is much larger than the decoder.
    - This forces the representation h to be as detailed as possible.
- Loss function is MSE predicted x and real image $\Sigma(\hat{x}_i - x_i)^2$; i is pixel index
- Mask Ratio hyperparam; mask ratio of 75% good for f-tning&linearprobing

Joint Embedding Prediction: combines the two prior approaches.
- MAEs avoid some drawbacks of CL, but training a full recon model is costly
- Not necessary to fully learn a perfect dec. model as unused after training
- The idea behind i-Jepa: combining strength of CL and MAE approaches
  - We mask most of the image, leaving one region visible to the net. We encode this producing an embedding via the *context decoder*.
  - We do the same for many parts of the image, putting it through the *target encoder*. You obtain one embedding for each patch
  - THEN we take the context to the predictor, and try to work out which embeddings passed through the target encoder are part of that context
  - Loss is MSE but in the emb space $\frac{1}{M}\sum_{i=1}^{M} D\left(\hat{\boldsymbol{s}}_y(i), \boldsymbol{s}_y(i)\right) = \frac{1}{M}\sum_{i=1}^{M}\sum_{j \in B_i} \|\hat{\boldsymbol{s}}_{y_j} - \boldsymbol{s}_{y_j}\|_2^2.$
  - Loss is avg L2 dist b/w pred. patch lvl reprs $\hat{s}_y(i)$ and target patch-lvl reprs $s_y(i)$

Applications of Self Supervised Models in Healthcare:
- Great for training on unlabelled scans - hard to access experts and have them do the labelling.
- *REMEDIS*: Developed by Google. Used for diagnostic imaging.
  - Pretraining models with SSL: improves performance overall and improves model generalisation to external deployment settings
  - REMEDIS starts with ImageNet weights (pretraining on non medical data). It's then adapted to medical domain with intermediate contrastive SSL w/o labelled medical data. Pretty much SIMCLR. Lastly it's finetuned to specific downstream ML tasks - either on ID or OOD labelled medical data using supervised learning. All increases robustness
- Retinal images disease detection using MAEs (Zhou et al., Nature 2023). Takes CT scan and fundus image of eye.

Image Registration

Image Registration: Establish spatial correspondences between images
Images: n dim arrs with scalar/vector values. f: R^d -> R^n. We will look at images with 3 dimensions, n pixels, each with 1 value. We index into them with integer indices spanning the (width, height) from bottom left. Pixel size is determined by camera. Meta info: Scale, Orientation, Position, Pixel Info
World Coordinate System: The axis of images might not align with the world's. We might have to do a transform; a translation from origin to origin, and mult by transformation matrix. We usually have offset & rot in header.
Image to world (there's a translation, rotation and scaling) matrix T_{itW} =

$$\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & ox \\ 0 & 1 & oy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} dxx & dyx & 0 \\ dxy & dyy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

World to image: $(x\ y\ 1)^T = T_{itW}^{-1} (x\ y\ 1)^T$
Im B to Im A (each im has own world): $(x_A\ y_A)^T = T^A_{itW}\ T_{BtA}\ T^B_{itW} (x_B\ y_B)^T$
We assume that the T^B_{itW} and T^A_{itW} matrices are given. But what we're trying to learn here is what T_{BtA} is!
Transformation Models: 1) Linear Models: identity; 0 DOF, rigid (rota + translation); 3/6 DOF (for 2D/ 3D), similarity (rigid+uniform scaling); 4/7 DOF, affine (rigid+nonuniform scaling+shear); 6/12 DOF. 2) Nonlinear: Deformable
Linear Transformations: 2D affine: trans (2), rot (1), scale (2), shear(1) = 6 DOF
Note that the rotation/shear matrix is about the z axis. Obvious how to modify

Translation $T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$ Rotation $R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ Scaling $S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$ Shearing $H = \begin{bmatrix} \cos\omega & \sin\omega & 0 \\ -\sin\omega & \cos\omega & 0 \\ 0 & 0 & 1 \end{bmatrix}$

To do the full shear you need to do HSH^-1 (rotate, scale, rotate back)
Affine transformations maintain the parallelness between lines. You can express any overall affine transform in the form (the 3 mats above multiplied):

$$A = H\,S\,H^{-1} R\,T = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

Free-form Deformations: Sometimes affine transforms aren't enough. We can imagine FFDs as in terms of a grid of control points over the image which the can be stretched towards and into. More points = more DOF.
Applications:
- Satellite Imaging (to reconcile / compare / combineimages of the same thing from different locations - satellite is always moving)
- Point Correspondences - Panoramic Stitching,
- Optic Flow (estimating depth using multiple views of the same object),
- Multi-Modal Image Fusion (medical imaging) b/w CT, MRI & MRA scans.
- Change Detection in medical imaging (i.e: checking for disease) by checking a healthy patient & an unhealthy, scan same person over time
  - Motion Correction (medical imaging): detecting breast tumours, take image, inject patient with contrast agent and then scan and use MC.
- Segmentation using Registration - if you have a prior segmented map and perform image registration, you can transfer those labels!
  - i.e: have a labelled atlas, register against it & then transfer (zeroshot)

Intensity-Based Registration: Estimation of transform params driven by appearance of the images. Images are registered when they appear similar
- We can overlay the images e.g: subtracting them. Try different rotations, transformations and minimize the image intensity difference
- Objective (Cost/Energy) Function: C(T) = D(I◦T, J). T is the transformation, D is dissimilarity, I◦T is the moving im, J is the fixed im
  - This is an optimization problem:T = $\text{argmin}_T C(T)$
- *Mono-modal registration:* Image intensities are related by a (simple) fn
- *Multi-modal registration:* Image intensities are related by a complex function or statistical relationship
- D: Dissimilarity Measures: Sum of sq diff (SSD), sum of abs diff (SAD):

$$D_{SSD}(I \circ T, J) = \frac{1}{N}\sum_{i=1}^{N} \left(I(T(x_i)) - J(x_i)\right)^2 \qquad D_{SAD}(I \circ T, J) = \frac{1}{N}\sum_{i=1}^{N} |I(T(x\_i)) - J(x\_i)|$$

  - Assumption: identity relationship between intens distributions
  - Application: mono-modal registration
- Correlation Coeff:
  μ_I: mean int. of im I
  μ_J: mean int. of im J

$$D_{CC}(I \circ T, J) = -\frac{\frac{1}{N}\sum_{i=1}^{N}\left(I(T(x_i)) - \mu_I\right)\left(J(x_i) - \mu_J\right)}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(I(T(x_i)) - \mu_I\right)^2}\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(J(x_i) - \mu_J\right)^2}}$$

  The top part of the formula is the cov(I,J) bottom is the 2 stdevs of ims

Intensity Based 2D Joint Identity Histograms as Dissimilarity Measures:
- Two axis on a graph, each spanning from the min intensity of one of the images, to the max of the same.
- Record the intensity of the first pixel of the first images, and the first of the second image, and then plot that as one point on our graph
- We loop over the entire thing, and ends with clusters. We see a clustering of intensities in one image that correspond to another set in the other image. We then bin them to see our counts in each bin.
  - These bins form a histogram. Identical images in the histo are a diag straight line, if one is translated then we see a blurred/smeared line.
  - Less aligned images have blurry plots but with concentrated clusters. But if you misregister by a few mm, gets blurrier. *The histogram quantifies how aligned images are!*
- We can then get a dissimilarity measure: p(i,j) = h(i,j) / N; h(i,j) = hist cnts
  - joint prob of an im pt having a value i in image I and value j in image J
  - $p(i) = \Sigma_j p(i,j)$ marginal prob of an image point having a value i in im I
  - similar for j - think of it as just computing one row or column
- You can compute SSD, SAD or CC from the joint histogram
- Shannon Entropy: $H(I) = -\Sigma_i p(i) \log p(i)$ - the amount of info in image I
- Joint Entropy: $H(I,J) = -\Sigma_i \Sigma_j p(i,j) \log p(i,j)$ - use for registr: D(I◦T, J)=H(I◦T, J)
- Mutual information MI(I,J) = H(I) + H(J) - H(I,J). Describes how well one image can be explained by another image
  - This can be rewritten in terms of marginal and joint probabilities:
  - $MI(I,J) = \Sigma_i \Sigma_j p(i,j) \log p(i,j) / (p(i)p(j))$
  - The dissimilarity measure is then defined as D_{MI}(I◦T, J)= -MI(I◦T, J)
- Normalised mutual information: NMI(I,J) =H(I) + H(J) / H(I,J) - independent of overlap in the images. D_{NMI}(I◦T, J)= -NMI(I◦T, J)
- MI still works even if we have multi-modal translation!

1) We can only compute Dissimilarity Measures in the regions of im. overlap. If overlap is small or high, we typically converge to a suboptimal solution. We can avoid this with Multi-scale Hierarchical Registration which Successively increase DOFs (like a set of gaussian image pyramids).
2) Pixel grids when computing intensities are not typically aligned... we fix this by for the moving image interpolating nearby pixels to obtain the intensity at this "pixel" (to get a more accurate intensity). Can do Nearest Neigh interpolation, or weighted sum of the nearby pixels.
3) To solve the optimization problem we need optimization strategies - we need a way to take our prior result to inform the next transformation. i.e: do the transform on moving image I, compute objective function with I and J & do Quality Evaluation but then we need an Optimization Strategy to do a Transformation Update for the next transform... Can use Gradient-descent (best), Stochastic opt, Bayesian opt, Discrete opt, Convex opt, Downhill-simplex... Multi Res is important else we'll converge to a local!

Evaluation:
- Qualitative: checkerboard the two images and see if similar. Diff Maps
  - Visual Assessment by blending the images / overlaying. Suboptimal
- Quantitative: have a ground truth to compare off the true registration
  - True error is often unknown as we don't have grd truth. We can obtain grd truth by doing artificial transforms applied to the images, or use simulations / phantoms. It is hard to define realistically deformed images however, and artificially deformed images are simplified representations of the true registration problem
  - Can compute similarity before/after regis,
  - Landmark errors: Find matching points / anatomical landmarks not used for registration. Then measure dist b/w lmarks before and after.
  - Semantic Seg: seg overlap (e.g. Dice), Surface dist (L2 / H-dorff dist)
  - Quality of deformation field 1. Consistency 2. Smoothness 3. Topology
  - What's important is eval is independent of regis features/cost fn!

Registration with Neural Networks:
- Once trained, computation / test time processing is fast
- *FlowNet*: Learning Optical Flow with Convolutional Networks
  - Compute optical flow between two consecutive image frames
  - Uses U-Net. Two images; concats them as a 2 channel image for input
  - *FlowNetCorrelation*: similar, but the two images go in different branches of the network. We do CNNs / operations before we concat and then compute the optical flow as in FlowNet.
  - Trained on "Flying Chairs" - they rendered moving fake chairs into the scene in different poses and thus have a ground truth to train on.
  - *FlowNet 2.0*: we feed into FlowNetC getting the Flow. We then feed Flow, Image 2 (warped with that Flow), Image 1 and Brightness Error. We concat that all and then pass it into *FlowNet* to get better Flow.
    - You repeat this multiple times. We also have *FlowNetSD* aimed at trying to compute the small displacements within images
    - We fuse the repeated Concat network result and FlowNetSD result: Flow Mag, Flow, Brightness Error to get our final flow result.
- Also done on medical images. Simpler Net to FlowNet used. Take two CT scans; do random trans to one im. Hence we know the supervision signal
- Modern approaches use unsupervised learning:
  - Spatial Transformer Networks take an image as input and use a localizer network to predict trans parameters (e.g., object pose).
  - Usually: 2 translation params or a 6D rot/translation/scaling vector.
  - These params feed into a pixel grid generator, which resamples the image via interp - an end-to-end differentiable process, making it an optimizable form of intensity-based registration.
  - The loss measures image similarity b/w the og & warped im. Despite optimizing a loss, it's unsupervised since the true trans is unknown.
  - The first stage of the network can estimate affine params directly.
  - *VoxelMorph*: General idea: take im -> global alignment -> refined alignment -> gets a correct alignment b/w two images. We use image similarity as supervision signal.
    - We have a UNet which we feed our two ims to get a displacemnt. field. We apply that to image & find loss b/w moving & fixed im. Optimize for that. At inference just forward pass through UNet for our displacement.
      - You can also generate moving image segs & feed it along w. og seg into the loss process to maximize the overlap.

Causality in Medical Imaging

Causality: the relationship between cause and effect; i.e: does chocolate consumption correlate to winning Nobel {rizes?
Reichenbach's Common Cause Principle: Either 1 of the 3 must be true:
1. Chocolate consumption causes Nobel prize win: C -> N
2. Nobel prize win causes chocolate consumption: C <- N
3. Both are caused by an unknown factor: C <- U -> N (i.e: GDP per capita...)

Simpson's Paradox: when a trend appearing in separate groups changes when combining the groups, due to a hidden factor influencing the relation.
- Consider Kidney Stone Treatment A and B. A is more successful for both small and large stones. But when combining the metrics, B actually has better overall success.
  - This happened because those with large stones were disproportionately given Treatment A, but it's a harder condition to solve.
  - We say Stone Size (S) is a confounder.
  - To resolve this, make treatment (T) independent of size: What's the recovery (R) rate if all subjects receive treatment A vs B?

Predictive Modelling: Given an im X, train model to predict label Y: P(Y|X).
Assumptions: 1) we have enough training data (X , Y) 2) Training and test data come from the same distribution.
- If X -> Y then it's a causal relationship - can predict the effect from cause
- Y -> X: anti-causal - we can predict the cause from the effect.
- Sometimes we don't have enough information

Ladder of Causation:
1. *Association*: How would seeing X change my belief in Y?
   - e.g: Modelling correlations P(Y | X). Modern ML excels at this task
2. *Intervention*: What will Y be if I do X?
   - We experiment to identify causal effects. Important for planning
3. *Counterfactuals*: What if X had not occurred? Deduce causes for obs. event

Structural Causal Models:
- A Structural Causal Model (SCM) is a triple: M = <X, U, F>
  - Two sets of variables, observed $X = \{x_1, ..., x_N\}$ & unobs. $U = \{u_1, ..., u_N\}$
  - A set of functions known as causal mechanisms: $F = \{f_1, ..., f_N\}$
  - The value of each variable is a function of its parents (direct causes):
    - $x_k = f_k(pa_k, u_k)$, for k = 1, ..., N
  - *endogenous:* caused by factors in the model. *exogenous:* outside factors
  - Example: Simple SCM: u_1 and u_2 are sampled from the world - assume $u_1, u_2 \sim N(0,1)$. We have that $x_1 = f_1(u_1)$ and $x_2 = f_2(x_1, u_2)$. Clearly x_1 and x_2 are endogenous, u_1 and u_2 are exogenous.
  - Acyclic SCMs can be represented by a Directed Acyclic Graphs (DAGs), with edges pointing from causes to effects
- SCM structures: 1) Confounder (a node that points to two nodes), 2) Collider (two nodes pointing to one node), Mediator (node x points to y and z, but y also points to z).

Observational (Association) Distribution Properties:
- SCMs w/ jointly indep exog noises are Markovian: $P(u_1, ..., u_N) = \Pi_{k=1 \text{ to } n} P(u_k)$
  - i.e: the dist of all exog noises factor out across the entire graph
- Markovian SCMs induce a unique joint observational distribution over the endogenous variables: $P_M(x_1, ..., x_N) = \Pi_{k=1 \text{ to } n} P_M(x_k \mid pa_k)$
- Each variable is independent of its non-descendants given its direct causes (causal Markov condition)

Interventional Distribution Properties:
- SCMs predict the causal effects of actions via interventions:
  - Interventions answer causal qns: e.g. what would be x_2 if we set x_1 = c?
- Interventions replace one or more of the structural assignments and are denoted with the do operator: $do(x_k = c)$
  - When we "do" something this induces a submodel and its entailed distribution is known as the interventional distribution: $P_{Mc}(X|do(c)$

Counterfactuals:
- SCMs can consider hypothetical scenarios:
  - Given that we observed (x_1, x_2) what would x_2 have been had x_1 been c?
  - e.g: if all else was equal, would I've been late had I not missed the train?
  - Counterfactual inference involves three steps:
    - 1. Abduction: Update P(U) given observed evidence, i.e. infer P(U|X)
    - 2. Action: Intervene by e.g. $do(\tilde{x_k} = c)$ and obtain the submodel M_c
    - 3. Prediction: Use <M_c, P(U|X)> to compute counterfactuals
- *Example: Computing Counterfactuals*:
  - $x_1 = f_1(u_1) = 1 + u_1$ $\quad x_2 = f_2(x_1, u_2) = 3x_1 + u_2$
  - Given we observed $\{x_1 = 2, x_2 = 4\}$, what would x_2 have been if x_1 was 5?
    - Abduction: x_1 = 2 = 1 + u1 => u1 = 1 $\quad x_2 = 4 = 3(2) + u_2 \Rightarrow u_2 = 2$
    - Action: $\tilde{x_1} = 5$ $\quad$ Prediction: $\tilde{x_2} = 3\tilde{x_1} + u_2 = 3*5-2 = 13$

Deep Structural Causal Models:
- Uses deep generative models to learn SCM mechanisms: $x_k = f_k(pa_k, u_k)$
- Tractably estimate causal effects of interventions and perform counterfactual inference, to answer "what if...?" type questions
- Abduction is challenging in complex problems, e.g. medical imaging
- Research QN: *Can we generate plausible high-fidelity image counter-factuals of real-world data, and if so, how do we evaluate them?*
- *Models*:
  a. *Normalizing Flows*: build complex probability dists via successive (invertible) transformations of simple base distributions.
     - NFs enable invertible mechanisms and deterministic abduction: $x = f_\theta(pa_x, u_x)$ $\quad u_x = f_\theta^{-1}(pa_x, x)$
     - This however doesn't scale very well to images
  b. *Variational Autoencoders*:
     - Using a VAE means abduction isn't deterministic anymore (as we're generating from a latent space)
     - We fix by doing abduction differently: $x = f_\theta(pa_x, u_x) = h(\epsilon; g_\theta(z, pa_x))$ - we factorize it into having an epsilon noise component in h which is invertible, as well as the z component from latent (non-invertible)
     - Example: we have an image x and its parent (metadata: e.g: caption)
       - Abduction: $z \sim q_\phi(z \mid x, pa_x)$ (pass it thru encoder). Then we take our parents info and pass the encoding z and parent conditioning pa_x through the decoder. We get a mean and variance for each pixel
         - Subtract output mean x from original mean μ, and divide by var to get ϵ. This is summarised here: $\epsilon = h^{-1}(\mathbf{x}; g_\theta(\mathbf{z}, \mathbf{pa}_\mathbf{x})) = \frac{\mathbf{x} - \mu(\mathbf{z}, \mathbf{pa}_\mathbf{x})}{\sigma(\mathbf{z}, \mathbf{pa}_\mathbf{x})}$
       - Action: $do(pa_x = \tilde{pa_x})$
       - Prediction: Replace the current conditioning with the new counterfactual parents: $\tilde{\mathbf{x}} \sim p_\theta(\tilde{\mathbf{x}} \mid \mathbf{z}, \widetilde{\mathbf{pa}}_\mathbf{x})$
         $\tilde{\mathbf{x}} := h(\epsilon; g_\theta(\mathbf{z}, \widetilde{\mathbf{pa}}_\mathbf{x})) = \mu(\mathbf{z}, \widetilde{\mathbf{pa}}_\mathbf{x}) + \sigma(\mathbf{z}, \widetilde{\mathbf{pa}}_\mathbf{x}) \odot \epsilon$
         - Note that the exog noise is shared between observed and counterfactual - we've inferred some latent code and changed a conditioning variable producing the new counterfactual
     - *Problem*: VAEs scale poorly to large images
     - *Solution: Hierarchical VAEs:* which are deeper and have multiple layers of latent variables. This isa menable to principled abduction allowing us to produce high fidelity image counterfactuals
       - *Diffusion Models are a type of Hierarchical VAE*: encoder is fixed rather than learned, each latent variable has the same dimensionality as and a single (denoising model is shared amongst all layers in the hierarchy. We have a diffusion loss term

- *Generative Models* are in one of two ways:
  1. Exogenous Prior: $p_\theta(Z_{\text{ILL}})$ - the latent space not touched by any of the observed variables directly
  2. Conditional Prior: $p_\theta(Z_{\text{ILL}} \mid pa_X)$ - the latent space IS affected by observed variables.
  - In any case we're stacking lots of residual blocks and have a very large latent space in order to try learn what we need
- *Latent Mediator Model*: the conditional prior (2) induces a latent mediator since Z_{ILL} is no longer exogenous
  - It still has a Markovian interpretation, so we can do causal mediation analysis:
  - CMA is the study of $p(U) = p(U_x)\left(\prod_{i=1}^{n} p(U_{pa_i})\right)\left(\prod_{i=1}^{n} p(U_{z_i})\right)$ how a treatment effect is mediated by another variable, to help explain why or how an individual may respond to certain stimulus.
  - This enables estimation of Direct (DE), Indirect (IE) and Total (TE) causal effects $DE_x(pa_x) = \mathbb{E}\left[g_y(pa_x, z_{1:L}) - g_y(pa_x, z_{1:L})\right]$
  - As an example in Morpho MNIST, you get to see that $IE_x(\widetilde{z}_{1:L}) = \mathbb{E}\left[g_y(pa_x, \widetilde{z}_{1:L}) - g_y(pa_x, z_{1:L})\right]$ $TE_x(\widetilde{pa}_x, \widetilde{z}_{1:L}) = \mathbb{E}\left[g_y(\widetilde{pa}_x, \widetilde{z}_{1:L}) - g_y(pa_x, z_{1:L})\right]$ observed digit thickness causes pixel intensity. Changing thickness changes the generated intensity but not vice versa!

Evaluating Counterfactuals:
- Soundness Theorem: the properties of composition, effectiveness, and reversibility are necessary in all causal models
- The completeness theorem states that these properties are sufficient
- We measure counterfactual soundness using these axiomatic properties:
  - *Composition*: If you intervene on a variable X by setting it to the value it would have had anyway, no other variables' values are changed.
    - Thus there exists a null-intervention
  - *Effectiveness*: Intervening on a variable to have a specific value will indeed cause the variable to take on that value.
    - For all of these axioms, we say the model is debiased if it does the 3
  - *Reversibility*: If two variables, X and Y, influence each other in a way that creates a feedback loop, they must settle on a single, consistent pair of values rather than having multiple possible solutions.
- These properties have a trade off. While composition -> reversibility, you can't really satisfy all 3 at the same time optimally. Need to pick one which is most important and maximize it unbiasedly.

Overall, it's crucial to account for the data-generating process and causality in our modelling to avoid biased predictions.
Deep SCMs can generate plausible high-fidelity image counterfactuals
Latent mediator models enable estimation of direct, indirect and total causal effects for high-dimensional variables
Sadly, measuring counterfactual effctvns needs separately trained classifiers

Inverse Problems

Inverse Problems: Given a measurement $y = Ax + n$, we want to estimate x.
- Note the noise term n. Applications: inpainting, de-blurring/noising, recon.
- The matrix A is known as the forward model

The Classical Solution for Deblurring:
- Formulating it as a Least Squares Problem: arg $\min_x D(Ax, y)$; we have that $D(Ax, y) = 1/2\,\|Ax - y\|_{\ell^2}$; the solution is $x^* = (A^T A)^{-1} A^T y$ (alr know A)
- This is an ill-posed problem though. Slightly perturbing the input image gives us a wildly different solution. There's multiple "correct" solutions...
  - To get us a more correct solution we add a *Regularization* term to the above: $\lambda R(x)$; $R(x) = 1/2\,\|x\|_{\ell^2}$; i.e: the solution is arg $\min_x D(Ax, y) + \lambda R(x)$
  - Least SQ soln with Tikhonov regularisation: $x^* = (A^T A + \lambda I)^{-1} A^T y$ - this is better conditioned and helps suppress noise.
- This gives us the general approach for images: solve the optimization prob: arg $\min_x D(Ax, y) + \lambda R(x)$. D(Ax, y) - Data Consistency term, R(x) - Regularization term, λ - Regularisation parameter.
  - Common Regularisers: 1) $R(x) = \|\nabla x\|$ - Tikhinov Regularisation (first derivative of the image should be smooth).
    2) $R(X) = \|\nabla x\|_{2,1} = \sum_{i=0}^{n}\sum_{j=0}^{n} \sqrt{(\nabla x_{ij}^T \cdot d)^2}$ - Total Variation Regularisation (don't have too much intensity fluctuations in image; d = dims)
    3) $R(x) = \|\nabla x\|_1$ - Wavelet Regularisation (∇x = gradient of image x) (don't have too much components in the wavelet domain space)
- Instead of choosing, we'd like to learn a good R from the training data:

Building a Forward/Inverse Model:
- Data: We either have lots of images and their perturbed version OR we have a forward model and we generate the perturbed versions ourselves
- We have a specific inverse depending on what our forward is

Solving the problem of building the Inverse Model:
1. Model Agnostic Approach: we can train a Neural Network that maps y to x; just optimize loss based on how close the output is to the groundtruth x.
   - Partly Model Agnostic: We improve by doing preprocessing before feeding into inverse NN model: we know our tform (e.g: downsample), so we can upsample the image before feeding into NN, improves results
2. *Decoupled Approach* (first learn then reconstruct):

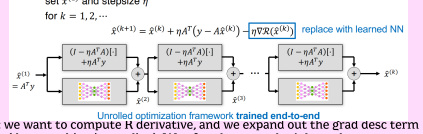
a. Deep Proximal Gradient set $z^{(1)}$ and stepsize η
we can do this with deep learning - replace the denoising step with a Neural Network
for $k = 1, 2, \cdots$
$z^{k} = \hat{x}^{(k)} + \eta A^T(y - A\hat{x}^{(k)})$ — gradient descent (DC)
$\hat{x}^{(k+1)} = \left[\arg\min_{x}\left\|x^k - x\right\|_2^2 + \eta\lambda R(x)\right]$ — denoising

b. We could also use a GAN: $R(x) = 0$ if $x \in$ on image manifold, ∞ otherwise
The goal is to find images similar to the natural images we expect
- GANs have 2 components: 1) a generator G network (takes a noise vector) and converts to an image (encoder-like). 2) we also have a discriminator D which we supply the generated image and real image. The job of the discriminator is to see whether the generation looks real or not.
- We train D to max the probability of assigning correct label. Train G to "fool" D. These correspond to the following two terms in Value Func:

$$\min_G \max_D V(G, D) = \underbrace{\mathbb{E}_{x \sim p_{data}(x)}[\log \mathcal{D}(x)]}_{\text{real}} + \underbrace{\mathbb{E}_{z \sim p_z(z)}[\log(1 - \mathcal{D}(\mathcal{G}(z)))]}_{\text{synthetic (aka "fake")}}$$

  - This discriminator is useful as a regularization function!
  - But then you can use the generator too instead of gradient descent as it produces "solutions" to the optimization problem.
- Learn generator G that outputs $x \in \mathbb{R}^d$ given $x \in \mathbb{R}^{d'}$ for $d' < d$
  - $R(x) = 0$ for $x \in$ range(G), ∞ otherwise
  - We then choose $x \in$ range(G) that best fits the data:
    - $x^* = \arg\min_{x \in \text{range}(G)} \|y - Ax\|_{\ell^2} = G(z)$

3. *Unrolled optimization*: assume that R(x) is differentiable. Then:
set $\hat{x}^{(1)}$ and stepsize η
for $k = 1, 2, \cdots$
$\hat{x}^{(k+1)} = \hat{x}^{(k)} + \eta A^T(y - A\hat{x}^{(k)}) - \eta\nabla\mathcal{R}(\hat{x}^{(k)})$ — replace with learned NN



Unrolled optimization framework **trained end-to-end**

i.e: we want to compute R derivative, and we expand out the grad desc term too (do everything in one line). We often however don't know how to differentiate R so we use an NN in practice. We do need an NN for each step - so it becomes costly unless we ensure there isn't too many iter steps

We can apply this inverse problem solving framework to many applications:
Deep Learning for Image Super-Resolution:
- Upsample low res im. to high res. Forward model: im. degradation followed by downsampling. Inverse model: interpolation based models
- *Post-Upsampling Super Resolution (Model Agnostic)*:
  - Directly upsample LR to SR. Requires many learnable upsampling layers
  - Fast, low mem required. But net has to learn entire upsampling pipeline (no prior knowledge) & only performs one specific up-sampling factor
- *Pre-Upsampling Super-Resolution*:
  - Use traditional upsampling algorithm (e.g: LERP) to obtain SR images. Then refine upsampled using a deep neural network (usually a CNN)
  - CNN only needs to correct smaller details. Can be applied to a range of upscaling factors and image sizes. But operates on SR image, thus requires more computation and memory
- *Progressive Upsampling Super-Resolution*:
  - Multi-stage: Use a cascade of CNNs to recon higher and higher res. At each stage, the images are upsampled to higher res & refined by CNNs
  - Decomposes complex task into simple tasks, reasonable efficiency. But it's sometimes difficult to train very deep models
- *Iterative Up-and-Down Sampling Super-Resolution (SoTA if using CNNs)*:
  - Alternate between upsampling and downsampling (back-projection) operations. We have up-up and down-down skip conns to later layers (for error prop), & we have mutually conn up & down-sampling stages.
  - Superior perf as it allows error feedback. Easier training of deep nets
- *Loss*: Pixel-wise loss function (either L1 or L2); L = 1/N $\sum_{i=1 \text{ to } N} |x_i - x_i\hat{}|_p$
  - *Huber-Loss:* L = 1/N $\sum_{i=1 \text{ to } N} \delta|x_i - x_i\hat{}|$, $\delta(a) = 0.5a^2$ for $|a| <= 1$ else $|a| - 0.5$ this is a mix of L1 and L2 loss. L2 is mostly used in practice.
  - *Perceptual-Loss:* Computes loss on the output θ of an interned layer l of a pre-trained network: L = 1/M $\sum_{i=1 \text{ to } M} (\theta_i^{-1}(x) - \theta_i^{-1}(x\hat{}))^2$. We take a pretrained network (e.g VGG), and we encode both our ground truth and produced upsampled image through it. We then compare the loss in the final layer activations - and this gives us a very good signal (as CNNs like ImageNet/VGG are trained to pick up important features)!
  - *Total Variation*: L = 1/N $\sum_{i=1 \text{ to } N} \sqrt{(\nabla x_i|^2 + |\nabla x_i|^2)}$; we assume the abs value of gradient of the image is low, e.g. image is piecewise constant. For 2D im:

$$\mathcal{L} = \frac{1}{n_1 n_2}\sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sqrt{|x_{i+1,j} - x_{i,j}|^2 + |x_{i,j+1} - x_{i,j}|^2}$$

  - *GAN Discriminator Loss:* We can use the discriminator output of GANs (as explained earlier) L = -log D(G(I$_{LR}$))
  - SoTA: Use GAN Loss + L2 Loss + Perceptual-Loss

Methods of Upsampling: 1) Nearest Neighbour 2) bilinear (add intermediate pixels LERP the original pixels) 3) Learnable weights via transpose convs
Parameterizing Images via Generative Networks:
- Consider a generator net Ψ, with a random fixed noise input vector z_0 and pass it through an encoder-decoder net. We get an output image.
- The network params w can be thought as a parameterization of images
- We want to fit the net to ONE single image. Given an image x, its params x are recovered by solving the following optim problem: min $\|x - \Psi(z_0; w)\|^2$. The net now encodes x, and we can rebuild the im by giving it z_0 each time
- *Deep Image Priors*: Optimizing this net is very fast - for most generative nets fitting naturally looking images is easier/faster than fitting others
  - Natural looking images fit models easily. Unnatural don't fit! We could use this to evaluate our image outputs.
  - *Application to Inpainting*: For inpainting we only reconstruct the visible pixels, implicitly inferring the pixels that are masked out by mask m: $\min_w \|m \odot (x - \Psi(z_0; w))\|^2$. We generate the weights for the image, and using that we can output the FULL image - doing our inpainting! And you only need to train one image!

Deep Learning for Image Reconstruction:
- MRI and CT scans don't produce the final image - we need to reconstruct to get something which is well human interpretable
- CT scans fire x-rays and measure how much of them went to the other side (absorption), and rotate around the subject producing a sinogram
  - High contrast, spatial res & fast to get but uses ionizing radiation
    - So we want to take minimal or even undersampled measurements and still be able to create a good reconstruction
  - *Forward Problem*: take subject & produce sinogram w/Radon transfrm
  - *Inverse Problem*: take sinogram and make the fully sampled reconstr.
- MRI: high contrast, high res & NO ionizing res, but slow acquisition. Slow is fine for static objects (brain, bones), but not moving heart, liver and fetus.
  - Options: real-time MR: fast, but 2D & relatively poor image quality. Gated MRI: fine for periodic motion, e.g respiration but requires gating (ECG or navigators) leading to long acquisition times (30-90 min).
  - *Forward Problem*: Use a fourier transform to produce K-Space
  - *Inverse Problem*: Take K-Space and produce the MR image
  - If the K-space is undersampled we violate Nyquist's theorem; the MR image is very aliased. We can solve this with Deep Learning!
- *Deep Learning Medical Scan Reconstruction Approaches:*
  a. *Image Domain*: Produce the aliased image via Fourier Transform, and then use an NN to improve quality
  b. *K-space Domain*: Use the neural network to do inpainting and then do the Fourier Transform
  c. *Model Agnostic*: Train a network that directly goes k-space to image without ever doing a Fourier Transform.
  - Training is done as normal: optimize weights w/ training set by feedback from a cost function which backpropogates errors. We use MSE cost.
    - Learning rate / grad desc as normal. We test as normal
  - Corrupted Image -> NN: Denoiser -> Transforms to Frequency Representation (forces recon image to be consistent with measured freq to be accurate for this patient - *data consistency*)
  - Applications: Quicker heart scans (as you have to hold your breath during them - difficult for ill patients)
  - If we upsample too far when our training data had a lot of different samples, we can *hallucinate* wrong results. Make sure the training data is accurate and related to what we want to apply our inverse model to!

Trustworthy Artificial Intelligence and Machine Learning

More data -> better, more robust, generalisable solutions. *Quality*: curation of data & annotations; we need it to be representative to avoid bias. *Quantity*.
Challenges of acquiring data:
- Human challenges: cost & effort of acquiring and labelling, need to provide incentives for data sharing. Legal: Consent, GDPR. Privacy: Ethics & Trust.
- Technical: Data quality, annotation and exchange formats

Secure and Privacy Preserving ML:
- Optimal privacy preservation requires implementations that are secure by default so-called "privacy by design
  - Requires minimal or no data transfer and provision of theoretical and/or technical guarantees of privacy

Federated Learning:
- Some data can't be safely shared. Can you trust the ML researchers?
- Train a ML model by sending it to each client who trains it on their own data and sends it back encrypted. We aggregate all of the model updates
- *Stochastic Gradient Descent:*
  - For a training dataset with n samples (x$_i$, y$_i$) with $1 <= i <= n$, the loss fn is $L = 1/n \sum_i L_i(x_i, y_i, \theta)$. Optimization uses grad desc: $\theta^+(j+1) = \theta^+j - \tau\nabla L(\theta)$
  - Typical grad desc uses SGD through mini batches of size m: $\nabla L(\theta) \simeq \sum_{i=1 \text{ to } m} \nabla L(x_i, y_i, \theta)$
- For Federated SGD, we have data distributed to K different clients, P_k is the set of indices of the dataset of client k, and $n_k = |P_k|$. The loss fn is then just summing over all clients and weighting their losses:
  - $L(\theta) = \sum_{k=1 \text{ to } K} (n_k / N) L_k(\theta)$. At each iteration we transmit a gradient - this could be huge (70 billion params = gradient w/ 70 billion entries)
- (Lk is just the loss of a specific client - that's what's sent to the server). With iid assumption, loss of 1 client = expected loss of whole dataset.
- For Federated Averaging: The clients themselves do parameter updates - does this for N iterations before transmitting updated grad to server. We randomly choose some clients each round to update and avg.
- If the data is not IID, our gradient descent is not very good. If unbalanced or few training samples, this happens too. Communication is expensive, and no formal security/privacy provided; prone to adversarial influence

Overview of Existing Privacy Preserving Methods:
1. *k-anonymity*: if patient records/IDs are anon, it should be impossible to link records to individuals.
   - A dataset is said to be k-anonymous if, for any person's record in a dataset, there are at least k-1 other records which are indistinguishable.
   - Thus the best a linkage attack could ever do is identify a group of k records which could belong to the person of interest.
   - Even if a dataset isn't inherently k-anonymous, it could be made so by removing entire fields of data (like names and addresses) and selectively censoring fields of people who are particularly unique.
   - Attacks can still be done by linking different anonymised tables together to narrow down info. As a result it's bad as we assume attackers have no other data sources.
2. *Homomorphic Encryption*: The entire process is encrypted: encrypted model is trained on encrypted data, the output is only decrypted as the final result is revealed. We assume we can do basic arithmetic on encrypted vals: $[x] \otimes [y] = [x+y]$; $[x] \otimes [y] = [x \cdot y]$; no interned. decryp - useful if we don't trust the server & are scared of middlemen. +,-,* are easily homomorphically encryptable, but trig and log are not; & time consuming
3. *Secure Multi-Party Computation*: individual contributions are 'sharded' and only the individual shards are shared with the participants. Only when shards are combined is the final result revealed. i.e: if we have an "Age" characteristic; and age=51, we can shard into 32 and 19 and compute on both separately (e.g compute "2; 64,38, we receive those and combine).
4. *Differential privacy*: purports the data so that information about the single individual is reduced while retaining the capability of learning. Process A is eps-diff priv if for all databases D_1 and D_2 which diffr by only 1 individual: $P[A(D_1) \in O] <= e^{eps} * P[A(D_2) \in O]$. Most used in practice. We do something random e.g: flip a coin. If heads, give our true answer, if tails (noise), flip another and if heads say one option else say the other.
   - Grants *Plausible Deniability*: we have the ability to claim we did not select a specific option. We can select our amount of noise. If noise = low less privacy protection. Else high.
   - Surveying multiple times decreases privacy.
   - An algorithm can be made approximately invariant to inclusion of a single data sample. This is achieved through the addition of noise to the output of the algorithm to perturb the dataset
     - If you don't trust anyone: add noise to the data immediately
     - If you trust the server, do the calculation and then add noise
   - *Concrete Attack Example*: *Gradient-based model inversion*:
     - If you're a MiTM then you see updates sent by client to server. You can using an algorithm reconstruct the inputs from the gradient updates and discover the original data!
       - The adversary randomly generates an image-gradient pair
       - The adversary captures the gradient update made by the victim
       - Using and loss function (often cosine similarity), the adversary minimises the difference between the captured and the generated updates by perturbing the image they control
       - The algorithm is repeated until the final iteration is reached

$$\arg\min_{x' \in [0,1]^n}\left\{1 - \frac{\langle \nabla_\theta \mathcal{L}(x, y), \nabla_\theta \mathcal{L}(x', y)\rangle}{\|\nabla_\theta \mathcal{L}(x, y)\|_2 \cdot \|\nabla_\theta \mathcal{L}(x', y)\|_2} + \alpha \text{TV}(x)\right\}$$

       - where x' is recon target, x is ground truth, y label, ∇L gradient w.r.t weights, <.> inner prod. a is a hyperparam scaling the variation penalty over the image.
     - Concrete Mitigation: Differentially private stochastic gradient descent (DP-SGD):
       - Compute gradients for each individual sample (they represent independent clients)
       - Clip the calculated gradients to obtain a known sensitivity
       - Add the noise scaled by the sensitivity from step 2 4
       - Perform the gradient descent step

Interpretability and Explainability:
- We want interpretability to understand why a model made a decision. This does also help debug
- Interpretability maps/TSNE help us see why the model made a decision.
- Helps debug bias classifications (or shortcut learning) & detect bias
- Helps ensure our values are aligned and our knowledge is reflected
- Common Misunderstandings: (i) Simple models are not necessarily more interpretable. (ii) Interpretability => fairness/trust/causality/acntable /=> interpretability. (iii) Interpretability doesn't always matter => can also lead to gaming the sys
- We care about interpretability in our final model - not before/during train

Methods of trying to find interpretability:
- *Ablation Test*: How important is a data point/feature? Train without some data or features and observe/study the impact. Difficult and expensive.
- *Fit Functions*: (use first derivatives) - sensitivity analysis or saliency maps
  - Looks at the distrib, takes a sample, then we look at the gradient of that w.r.t features. (i.e: if output says class 1, perturb features & check again)
  - Not necessarily easy for humans to interpret - gradient might be jumpy
  - i.e: take first deriv and look at image - might give a clear good edge map
- *Direct visualization of filters* - Easy to implement; Limited practical value
  - First layers are easy to interpret (mostly low-level features) -
  - Higher layers more difficult to interpret (non-interpretable features)
- *Occlusions*: Mask out region in the input image and observe net output
  - If masked out region causes a significant drop in confidence, the masked-out region is important
- *Saliency Maps*: three similar approaches which do backpass differently
  - *DeconvNet*: Given a trained network and an image, choose activation at one layer (set all others to zero), and try invert network
    - We only go to a certain depth in net with fwd pass then invert. Backpass looks the same as normal
  - *Standard Backprop*: We try to compute what pixels are most significant to a neuron - we do this by backprop w.r.t input pixels rather than weights: i.e: dh / dxi. We also set in the final vector all the classes we don't care about's neuron values to 0 and then backprop to the start
  - *Guided Backprop*: Improve results by "guiding" the backprop process. Idea: Positive gradients = features the neuron is interested in Negative gradients = features the neuron is not interested in
  - Set all negative gradients in the backpropagation to zero
- *DeepDream / Inceptionism*: Attempt to understand inner workings of net
- Optimize with respect to image
  - Arbitrary image or noise as input. Instead of adjusting network parameters, tweak image towards high "X" where "X" can be - Neuron/ Activation map/Layer or Logits/Class probability. Search for images that are "interesting". Different layers enhance different features.
  - Attempts to for a specific class find the canonical representative image (similar to Deep Image Priors)
  - Find an image x such that the activation $\Phi_n(x)$ at layer n is high: $\max_x \Phi_n(x) - \lambda R(x)$
    - Algorithm – fwd prop to layer n. Don't min loss. Instead maximize L2 norm of activations of a particular NN layer
    - Backpropagate to input layer resulting im shows learned features
- *Inversion*: attempts to construct an image from a layer activation y^*:
  - $x^* = \min(\|\Phi(x) - y^*\|_{\ell^2} + \lambda R(x)$; $\Phi(x)$ is net output for x; y^* is desired activ

Robustness: Adversarial Methods
- *Adversarial Attacsks*:
  - *Perturbation*: CNNs are very easy to trick into missclassifying with a noise term (not even perceptible to human eye)
    - NNs cannot avoid these!
    - Assume a linear classifier: $\theta^T x$. Imagine an adversarial example that contains a small, nonperceivable perturbation to the input. Let's denote the perturbation as $x^* = x + n$.
    - Then, the logits of the classifier would be $\theta^T x^* = \theta^T(x+n)$
    - An adversarial example is one that which maximizes the value of $\theta^T n$ to sway the model into making a wrong prediction
    - We solve the problem: $\|x\| = \max_i |[x]|$ s.t: $\|n\|_\infty \leq \varepsilon$
      - Example perturbation: $n = \varepsilon * \text{sign}(\theta)$. size of change = εmn
      - n = size of image, m = size of weights vector
      - If image is large we can expect an adversarial example...
  - *Fast Gradient Sign Method*:
    - Perform gradient descent in order to maximize the loss (the goal of adversarial attack).
    - Consider the input image x to be a trainable param and compute the gradient with respect to the input image to create a perturbation.
      - $n = \varepsilon \cdot \text{sign}(\nabla_x L(\theta, x, y))$
    - An adversarial example can be created as: $x^* = x + \varepsilon \cdot \text{sign}(\nabla_x L(\theta, x, y))$
- Defence: Generate adversarial examples, add the generated adversarial examples to the training set, retrain model using training set. There is however almost infinite adversarial examples...

Tutorials

Tutorial 0:
Feature Learning - Handcrafting the features can be hard - we might not know what's relevant. Instead we could try to *learn* the relevant features.
- Logistic Regression has the advantage of letting us visualise our learned features (i.e: for MNIST what the model actually thinks the digits look like)
  - It's done by taking the 784 coefficients and putting back into the image
  - It hasn't actually learnt what the two digits look like, it's learnt the separation of the two (you can imagine a divider in a hyperplane)
  - A generative model understands the digits (and hence can make them)

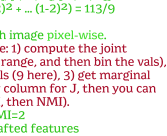
Tutorial 1: Just a normal simple ML setup like the CW. Just use AlexNet. AlexNet still performs well on out of dist data (input images shift by 2 pixels)

Tutorial 2: Segmentation: Literally just implement U-Net.
Tutorial 3: Self Supervised Learning: Just implementing some stuff from the lecture like infoNCE.
Tutorial 4: Spatial Transformers: just implementing some spatial transformer architecture as in the lectures.
Tutorial 5: Looks like T-SNE - similar to CW
Tutorial 6: This tutorial actually looks important / different enough

Exam Questions & Formulae

Receptive Field = (previous receptive field) + (product of strides before curr layer) * (curr layer kern size - 1) * (curr layer dilation)
Parameters: conv = (Kh * Kw * ChannelsIn + (bias=1))
$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$

1) Param count of simple logistic regression model have where raw pixel values are used as inputs? Number of channels * raw pixels + 1(bias)
2) Why do we split data into training, validation and test sets? train-test obvious; we use val to select hparams; so is biased/cant be used for testin
3) Ways to lower var? early stop, reduce features, increase training data, increase regularization penalty, data aug, dropout/batch norm layers
4) Problems with accuracy? class imbalance within the image => receive a high score by simply assigning the entire image to the dominant class
5) Design a simple enc-dec network with five conv layers and five convT layers that maps an inp of size 1×64×64 to an out of 10×64×64, with an intermediate, central rep between the enc and dec of size 32×10×10.
c1 = Conv2d(in=1, out=2, kernel=2, stride=2, pad=0) 2x32x32
c2 = Conv2d(in=2, out=4, kernel=2, stride=2, pad=0) 4x16x16
c3 = Conv2d(in=4, out=8, kernel=3, stride=1, pad=0) 8x14x14
c4 = Conv2d(in=8, out=16, kernel=3, stride=1, pad=0) 16x12x12
c5 = Conv2d(in=16, out=32, kernel=3, stride=1, pad=0) 32x10x10
ct1 = ConvTranspose2d(in=32, out=16, kernel=3, stride=1, pad=0) 16x12x12
ct2 = ConvTranspose2d(in=16, out=10, kernel=3, stride=1, pad=0) 10x14x14
ct3 = ConvTranspose2d(in=10, out=10, kernel=3, stride=1, pad=0) 10x16x16
ct4 = ConvTranspose2d(in=10, out=10, kernel=2, stride=2, pad=0) 10x32x32
ct5 = ConvTranspose2d(in=10, out=10, kernel=2, stride=2, pad=0) 10x64x64
(N.B: don't forget to write the fwd/back pass in qns like this)
6) Name a suitable (dis)similarity measure that could be considered for intensity-based registration for each of the following three pairs of images. (A) Normalised Mutual Information. Ventricles are dark in both images, but grey matter is inverted, so there is no linear relationship between intensities (B) NMI, possibly Correlation Coefficient. Darker and brighter regions seem to correspond to some extent. NMI might be the better choice. (C) Correlation Coefficient. Linear contrast change between the images, so there is a linear relationship between intensities.
7) After (almost) perfect regis b/w two images, you obtain the below joint histo. What can you say about the appearance of the two images? What would be a suitable dissimilarity measure? *(histo has v/ strongly diagonal dots but with an isolated cluster)* The histogram is obtained after registering a pre- and post-contrast scan of the same modality. The linear correlation of the majority of intensity pairs suggests mono-modality. The isolated cluster corresponds to a lesion that appears bright in the post-contrast image and dark in the pre-contrast one.
8) From 2020 exam: Design a simple spatial transformer network with two convolutional, two max pooling and two fully connected layers for estim the params of a 2D rigid transformation for inputs of size 1 x 64 x 64.
c1 = Conv2d(in=1, out=2, kernel=5, stride=1, pad=0)
c2 = Conv2d(in=2, out=4, kernel=5, stride=1, pad=0)
f1 = Linear(in=4 * 13 * 13, out=32); f2 = Linear(in=32, out=3)
Forward pass: x = f2(f1(max_pool(c2(max_pool(c1(x))))))
9) Decide whether the following medical image analysis tasks can be posed as a classification or a regression problem: a) Segmentation of brain tumours Classification b) Prediction of disease progression Regression, if disease progress continuous, classification if prog can be discretized. c) Detection of cells in microscopic images Classification: where each pixel is classified as 'cell' or 'not cell', regressed: each pixel's distance to the nearest cell centre is regressed. d) Localisation of anatomical structures: Regression, image points vote for the location of the structures by regressing offsets.
10) Is specificity a useful metric for evaluating segmentation algorithms? Why? Not useful as it depends on the extent of the 'background'
11) Cross Entropy Loss is calculated as -log p_correct_class (just note the correct class...)
12) Sketch the iterative process of intensity-based image registration. What are the three main components of this process?
image 2 -↓
image 1 → image transformation → T(I) → objective func
transformation update ↑ optimization strategy ←↓ *quality evaluation*
13) In brief, what is a spatial transformer network? A spatial transformer network allows an NN to learn how to perform spatial transformations on the input image to enhance the geometric invariance of the model. E.g in the Mnist dataset it would try to make all the images of each number look as similar as possible by scaling, shearing and rotating them.
14) Name two different loss functions that can be used to train a spatial transformer network in an intensity-based registration. Sum of squared differences Sum of absolute differences Correlation Coefficient
15) c The R-CNN algorithm was one of the first algorithms that uses CNNs for object detection. i) Briefly describe the key steps of the R-CNN algorithm for object detection. Extract region proposals (max 2000) from image using a proposal method (selective search) Selective search means we segment the image based on several factors (pixel intensities, colours) Warp each region proposal to a standardised size For each warped region compute the CNN features Classify features with a support vector machine Update bounding boxes using regression ii) What are the disadvantages of the R-CNN algorithm? Slow to train & slow inference time because the CNN features must be computed for each region one at a time. Selective search is a fixed algorithm, therefore no learning happens at that stage ad-hoc training objectives: SVM and bbox regressors can't be trained until each convnet has finished (bottleneck) iii) Explain how fast/faster R-CNN overcomes these disadvantages. Instead of feeding the region proposals to the CNN we feed the input image in and generate a convolutional feature map. We then identify regions of proposals and warp them into squares and use a RoI pooling layer to warp them into squares. This is fed into a fully connected layer which outputs a classifier and bounding box for each region of interest. Also you don't feed proposals - the conv op is done just once per image and a feature map is generated from it iv) How can an object detection method like faster R-CNN be modified to perform detection as well as segmentation of object instances? Mask R-CNN -> Add FCN that acts on region proposals and trains FCN to perform seg using annotated masks.
16) Pools have 0 params; Convs have w * h * num filters
17) You observe that this model achieves lower loss for an incorrect prediction than for a correct prediction. Why? The model predicted a higher rating for the correct answer. Cross Entropy loss is 0 for all classes except for the actual class so it only bases the loss on the higher rating of the correct answer. Initially, the network was untrained and very unsure, so predicted an almost equal probability for all classes. As it trains more, it outputs higher-probability predictions, even if they are occasionally wrong.
iv) Given your conclusions from iii), you decide to train your neural network with the accuracy as the loss instead of the cross-entropy loss. What are the potential problems with this approach? Accuracy not differentiable, if there is class imbalance the network may train itself to guess every object as the class with the highest number of entries
18) Propose a way to label new images, where each example can simultaneously belong to multiple classes. multidimensional vector that contains each type (car, sports, electric, truck, plane, ship) where 1 for every class it belongs to and 0 for those it does not.ii) To avoid extra work, you decide to retrain a new model with the same architecture (softmax output activation with cross-entropy loss). Explain why this is problematic. We now have multiple positive classes. Softmax averages the weight of all classes to 1 and therefore if there were 2 correct classes the max it could guess would be 0.5 for each. Since we are now using Sigmoid with cross entropy the network could achieve a loss of 0 by simply outputting 1 for every class Softmax encourages one of the output elements to be very high and the rest to be very low. This is appropriate when there is only one correct output label but will not let the network easily learn to output multiple labels for the same image. iii) suggest a diff final layer activ and loss for multiclass labelling: sigmoid, set output to 1 if sigmoid > 0.5, use BCE Loss
19) Calculate the SSD, SAD, CC, and NMI for the following pair of images. Write down any intermediate values that you may calculate in the process.
im_1 = [[2, 1, 2], [2, 4, 3], [5, 4, 1]], im_2 = [[1, 2, 1], [1, -1, 0], [-2, -1, 2]]
SSD = 1/N Σ (I(T(xn) - J(xn))^2 = 1/9((2-1)^2 + (1-2)^2 + ... (1-2)^2) = 113/9
SAD = 1/N Σ (I(T(xn) - J(xn)) = 25/9
CC = just plug in formula, calculate mu of each image pixel-wise.
NMI = (H(I)+H(J))/H(I,J); : H(I) = -Σip(i)log$_2$(i) (i.e: 1) compute the joint histogram (each axis is each image's intensity range, and then bin the vals), 2) make it into prob dist by dividing by num pixels (9 here), 3) get marginal probabilities for I by summing across rows, by column for J, then you can compute entropy of I and J, then compute MI, then NMI).
H(I)=0.6614, H(J)=0.6614, MI=0.0661, NMI=2
20) End to End Learning doesn't need handcrafted features
21) Which of the following can be used for regularisation in a neural network? (Check all that apply) L1 penalty on the weights Dropout Stochastic Gradient Descent with small batch sizes Data augmentation
22)A particular version of YOLO uses 11 x 11 cells for object detection across images. Furthermore, it uses four object bounding box predictions per cell and it can detect 10 classes of objects. What is the size of the output that the CNN has to predict? YOLO per bounding box predicts confidence+(x,y,w,h) = 5 values as well as 10 conditional class probs => 4x5+10=30 values per cell. 11x11x30 =3630 total. ii) You now modify this particular version of YOLO to be able to detect 1000 object classes. (4x5+1000)*121
iii) Briefly explain what steps you could take to avoid the computational difficulties that may arise from increasing the number of classes to 1000. separate obj detection from classification using a generic object detector (e.g: RCNN) and then a separate object classifier.
23) To calculate perceptual loss apply the filters as normal (no need to flip) and then compute MSE Loss b/w GT and Prediction
24) Noise eps * sign of ∆L/∆Adversarial eg = x + noise. Despite small eps, we can swing input massively (check adversarial section)
25) Normal backprop: see the activation, and where its zerod, also zero the backpass out. Deconvnet: direct to out the negative in grad. Guided Backprop: 0 both the normal backprop activation 0s and deconvnet grad 0s
26) Latent variables in a GMM represent the cluster assignments of data points. They indicate which Gaussian component (out of multiple in the mixture) each data point most likely belongs to.
27) First PC in PCA captures largest variation in dataset → encodes the most dominant anatomical variation across the dataset
28) 👎 fed learning → Clients have - diff amt data; varying quality & type of data; problems in sending large models over; bottleneck on compute capab.
29) ↑ model capacity → lower bias & incr variance → overfitting
30) output num for transformation → Rigid=3; Affine=6
31) Diag → unsupervised image reg. Cnn extracts feats; cnn outs. a trans'n model that est's. deform'n to align mov. w fixed img; Trans. applied to mov to prod warp then comp to fixed. Similarity used to refine model



1D Transpose Convolution of [3,6] with k=[2,5,1] y0=3*2=6, y1=3*5+6*2=27, y2=3*1+6*5=33, y3=6*1
Youre linearly multiplying input with kernel but w an offset for each input idx