A whirlwind review/overview of deep learning (continued)

CPSC 440/550: Advanced Machine Learning

`cs.ubc.ca/~dsuth/440/23w2`

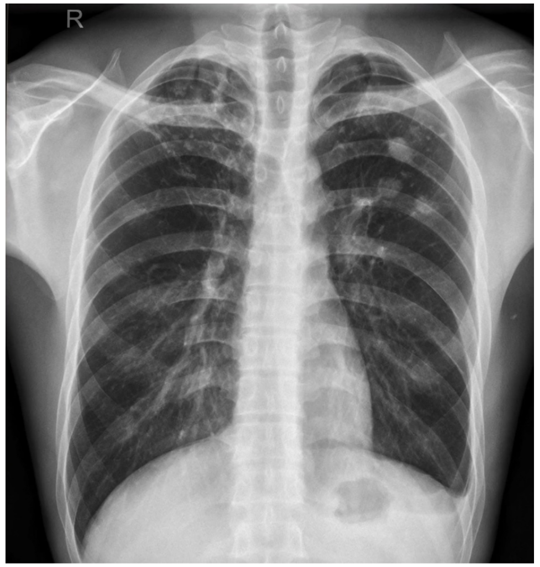
University of British Columbia, on unceded Musqueam land

2023-24 Winter Term 2 (Jan–Apr 2024)

Next Topic: CNN review

Motivation: X-ray abnormality detection

- Want to build a system that recognizes abnormalities in x-rays:



“Abnormality detected”
(binary classification)

- Applications:
  - Fast detection of tuberculosis, pneumonia, lung cancer, etc
- Deep learning has led to incredible progress on computer vision tasks
  - Much of this progress driven by convolutional neural networks (CNNs)

Convolutional Neural Network (CNN) motivation

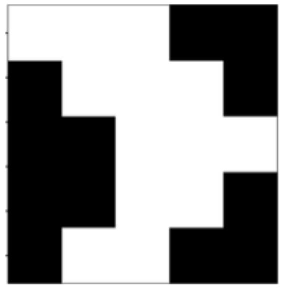
- Consider training neural networks on 500 pixel by 500 pixel images
  - So the number of inputs *d* to first layer is 250,000 (or 3x that, if colour)
- If first layer has *k*=10,000, then *W* has 2.5 billion parameters
  - We want to avoid this huge number (due to storage and overfitting)

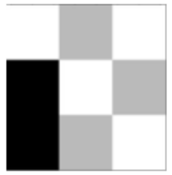
- CNNs drastically reduce the number of parameters by:
  - Having activations only depend on a small number of inputs
  - Using the same parameters on the connections of many activations
- Done using layers that look like "convolutions" in signal processing

Illustration of 2D Convolution

- 2D convolution:
  - Inputs: an "input" image *x* and a "filter" image *w*
  - Output: new image *z* whose pixels are dot products of filter and image region)

Input image **Filter image** **Output image**





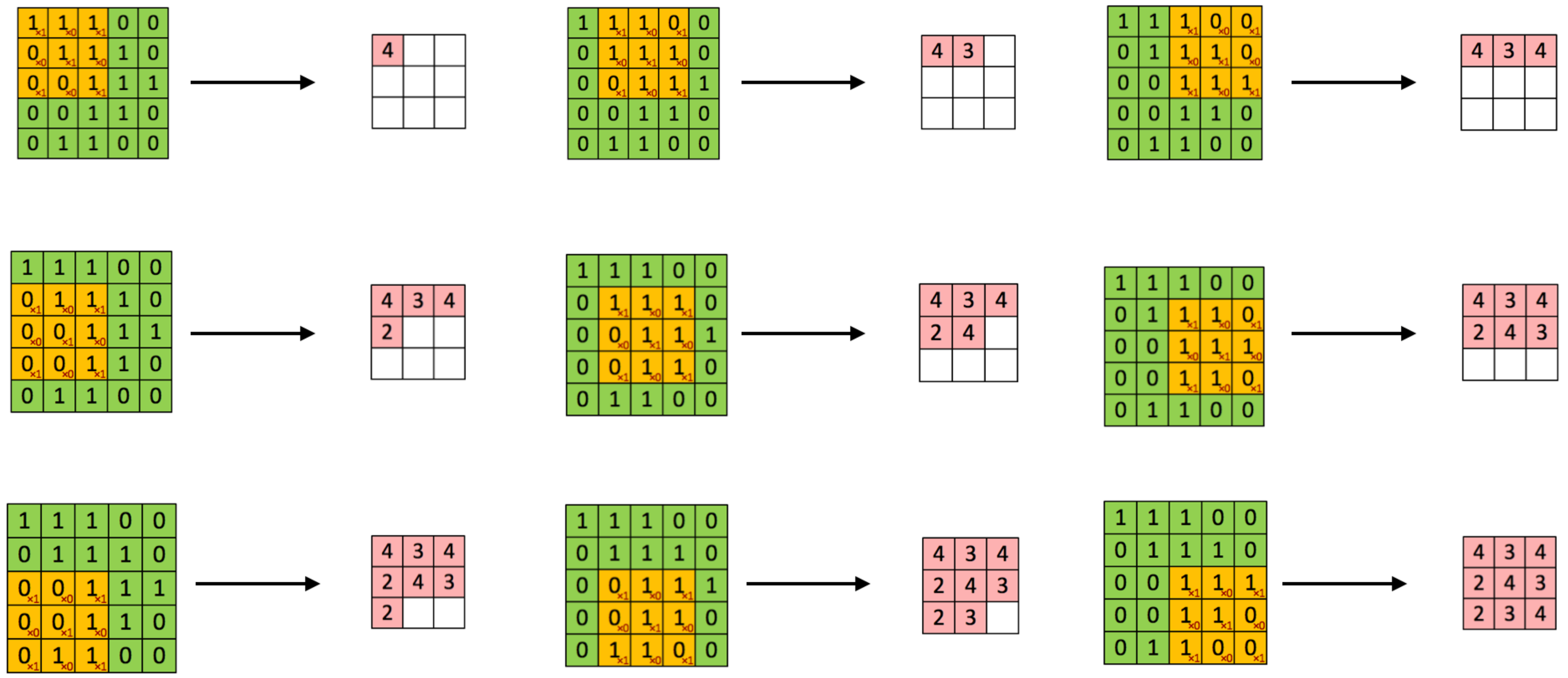
$$\underset{x}{\begin{array}{|c|c|c|c|c|}\hline 1&1&1&0&0\\\hline 0&1&1&1&0\\\hline 0&0&1&1&1\\\hline 0&0&1&1&0\\\hline 0&1&1&0&0\\\hline\end{array}} * \underset{w}{\begin{array}{|c|c|c|}\hline 1&0&1\\\hline 0&1&0\\\hline 1&0&1\\\hline\end{array}} = \underset{z}{\begin{array}{|c|c|c|}\hline 4&3&4\\\hline 2&4&3\\\hline 2&3&4\\\hline\end{array}}$$

Illustration of 2D Convolution

- 2D convolution:
  - Inputs: an "input" image *x* and a "filter" image *w*
  - Output: new image *z* whose pixels are dot products of filter and image region)

Filter image

1	0	1
0	1	0
1	0	1

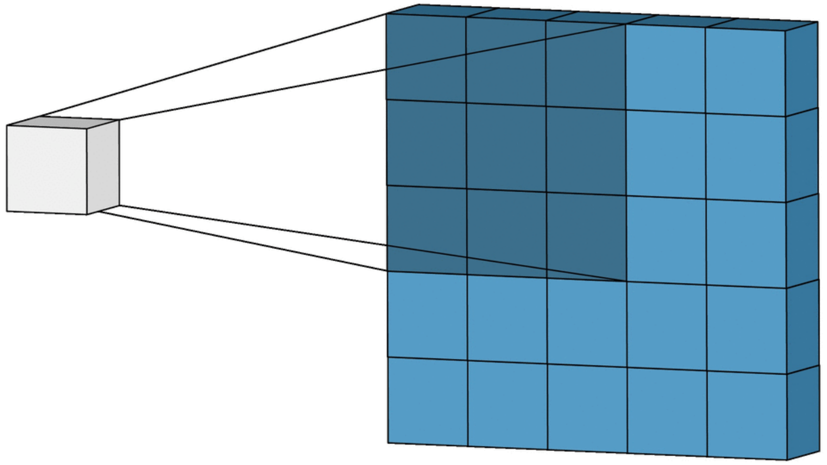


https://scientistcafe.com/ids/convolutional-neural-network.html

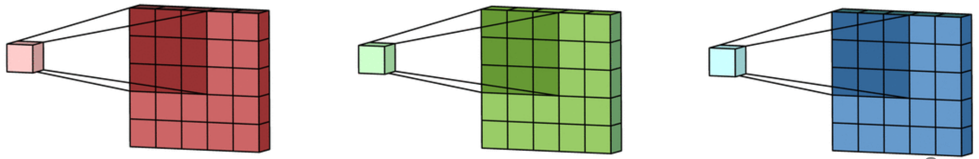
Illustration of 2D Convolution

- 2D convolution:
  - Inputs: an "input" image *x* and a "filter" image *w*
  - Output: new image *z* whose pixels are dot products of filter and image region)
- As a formula:

$$z[i_1, i_2] = \sum_{j_1=-m}^{m} \sum_{j_2=-m}^{m} w[j_1, j_2]\, x[i_1 + j_1, i_2 + j_2]$$



  - Final image *z* can be written as usual *z=W' x*
    - *W'* will be sparse, with filter values in *W* repeated
- 3D convolution (for colour images):
  - Weighted dot product across all three dimensions



https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

bonus!

Formal Convolution Definition

- We have defined the convolution as:

$$z_i = \sum_{j=-m}^{m} w_j x_{i+j}$$

- In other classes you may see it defined as:

$$z_i = \sum_{j=-m}^{m} w_j x_{i-j}$$

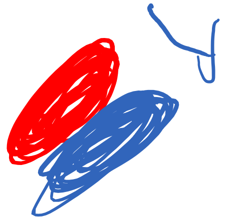
(reverses 'w')

$$z_i = \int_{-\infty}^{\infty} w_j x_{i-j} \, dj$$

(assumes signal + filter are continuous)

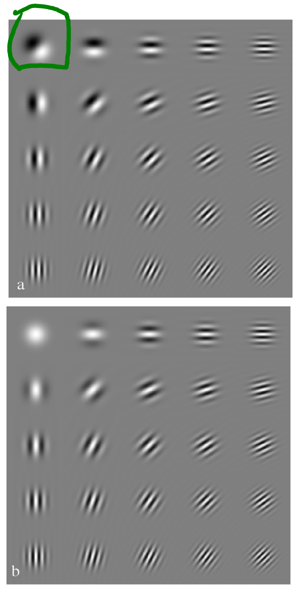
- For simplicity we're skipping the "reverse" step, and assuming *w* and *x* are sampled at discrete points (not functions)
- But keep this mind if you read about convolutions elsewhere

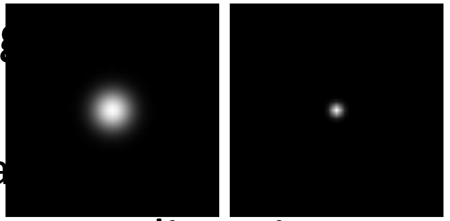
"signed" image:
- gray means 0
- white means 1
- dark means -1



Convolutions

- Pre-2012, people often designed the filters by hand
  - Filters can approximate "derivatives" or "integrals" of the image regions.
    - Derivative filters will up to 0, integral filters will add up to 1
  - Three of the most-common filters that people used:
    - Gaussian filters: integral filter, giving the average brightness in a re
      - Variance of the Gaussian controls the amount of smoothness
      - This produces a pixel feature that is less sensitive to noise than pixel's raw va
    - Gabor filters: derivative filters, measuring changes in brightness along a direction
      - We typically compute these for different orientations and "frequencies"
      - This gives a set of features that is useful in describing edges in the image
    - Laplacian of Gaussian filter: total second-derivative filter
      - Complements Gabor filters: helps describe if change is due to an edge, line, or continuous change
  - Similar filters may be used early in the eyes visual processing
  - Results of convolutions kind of like the "bag of words" making up images





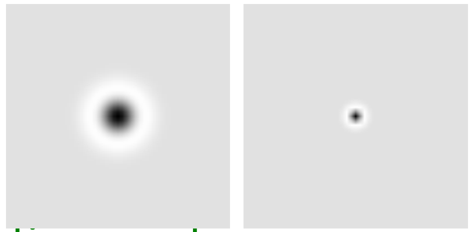
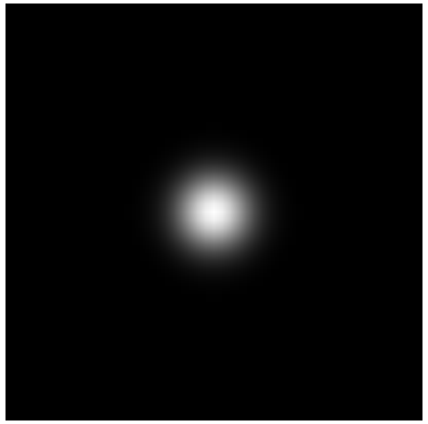


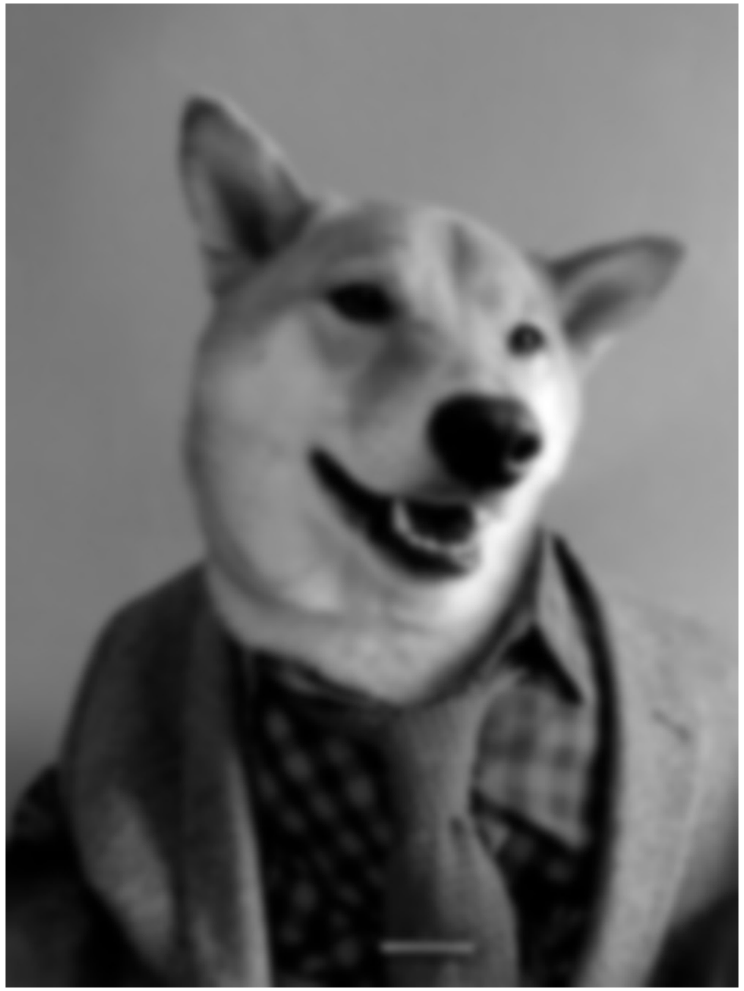
Image Convolution Examples

Gaussian Convolution:

$*$



$=$

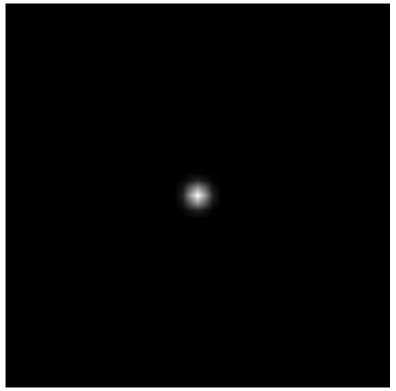


blurs image to represent average (smoothing)

Image Convolution Examples

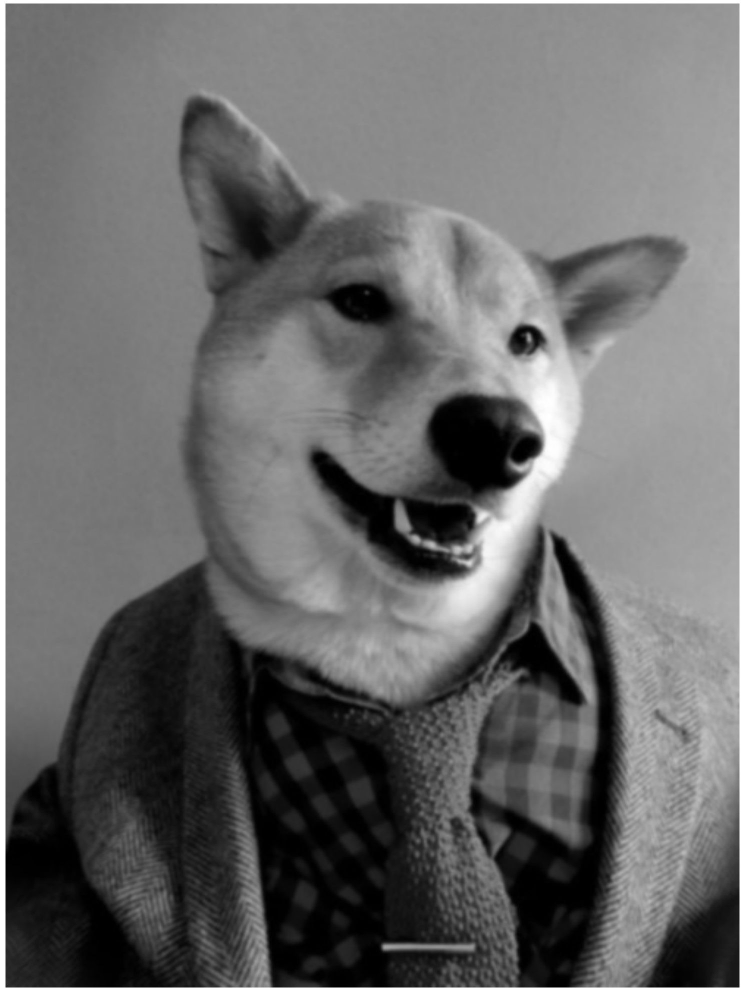
Gaussian Convolution:

$*$



(smaller variance)

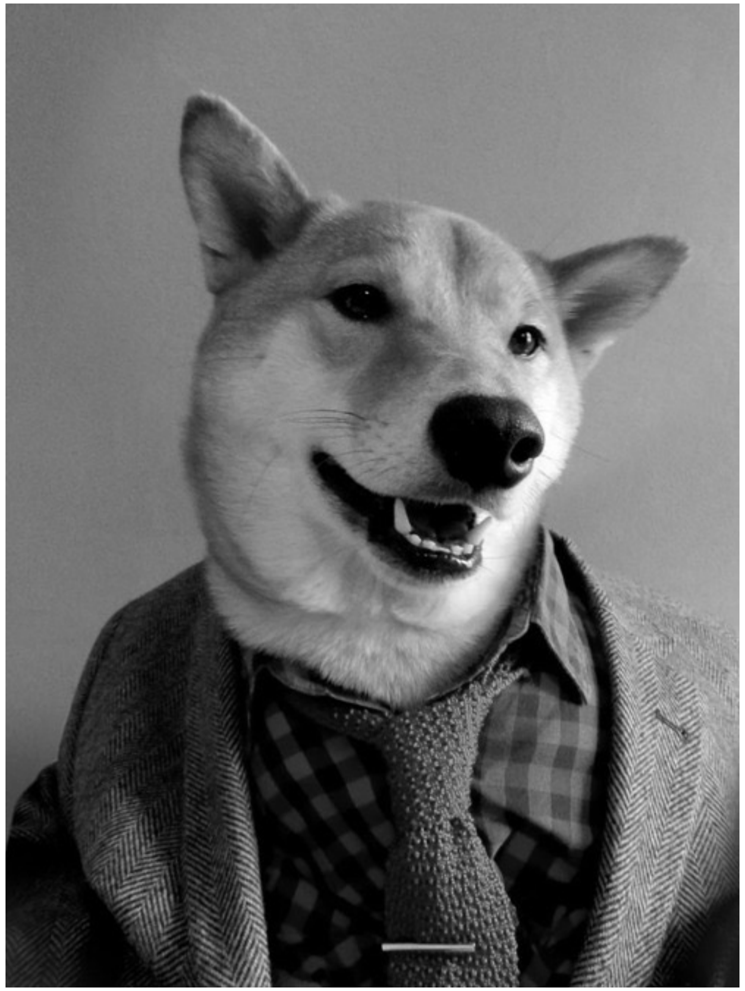
$=$



<u>blurs</u> image to represent average (smoothing)

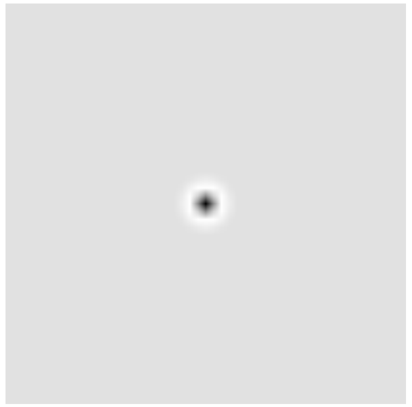
Image Convolution Examples

"signed" image (gray is 0)



$*$

Laplacian of Gaussian



"How much does it look like a black dot surrounded by white?"

$=$

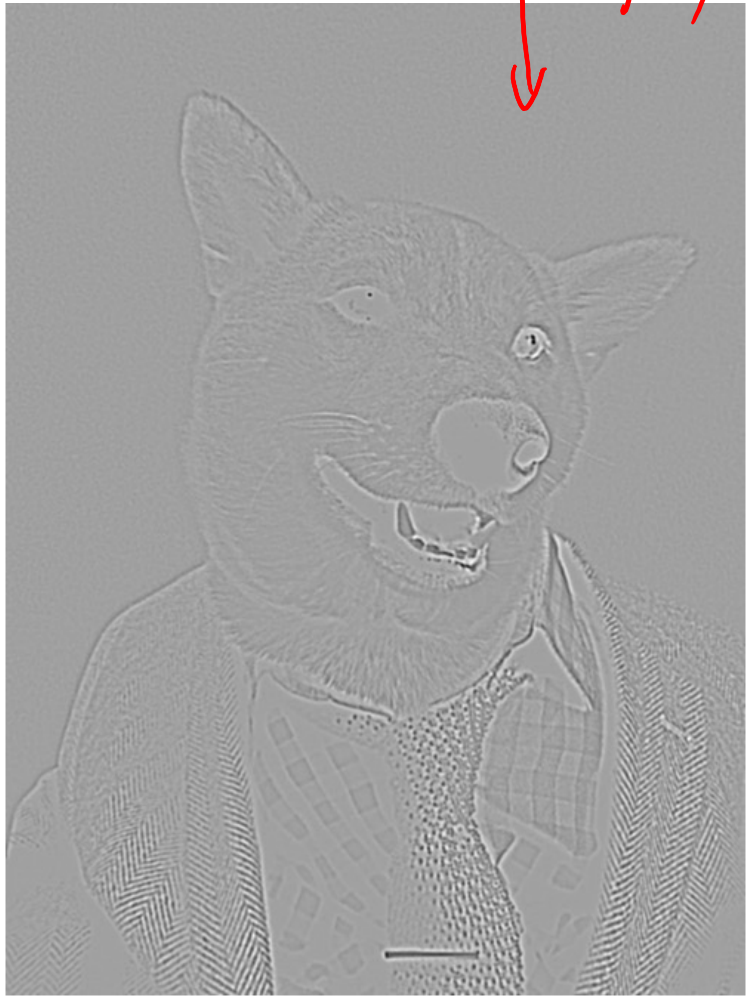
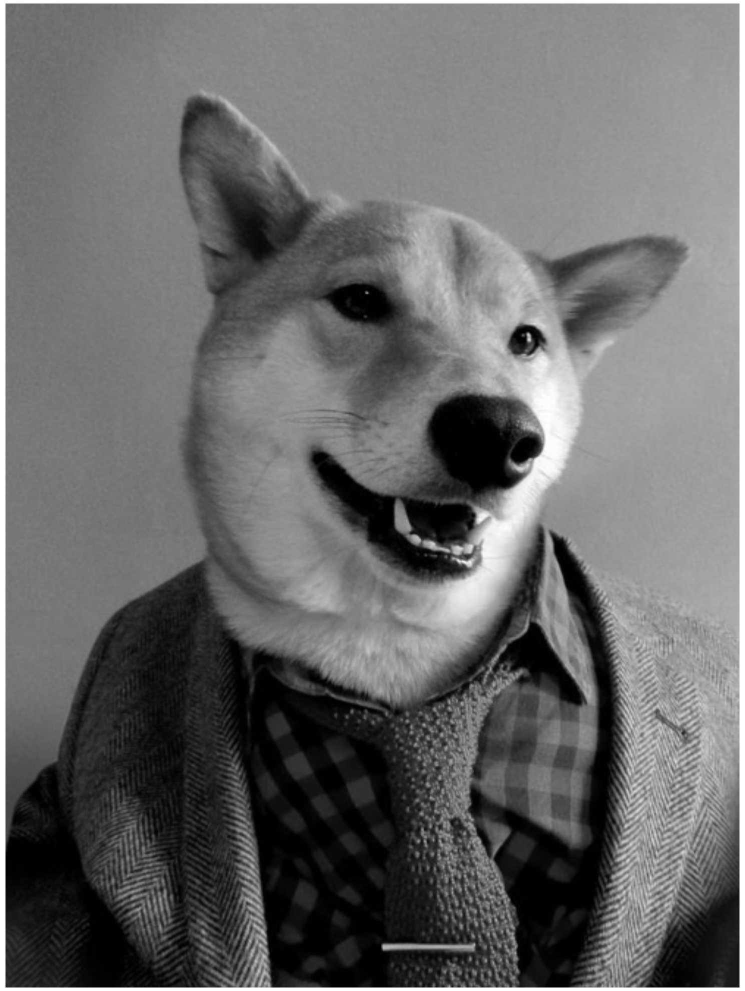
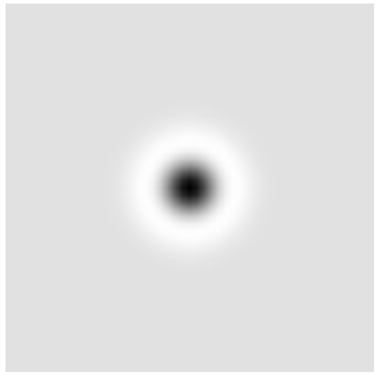


Image Convolution Examples



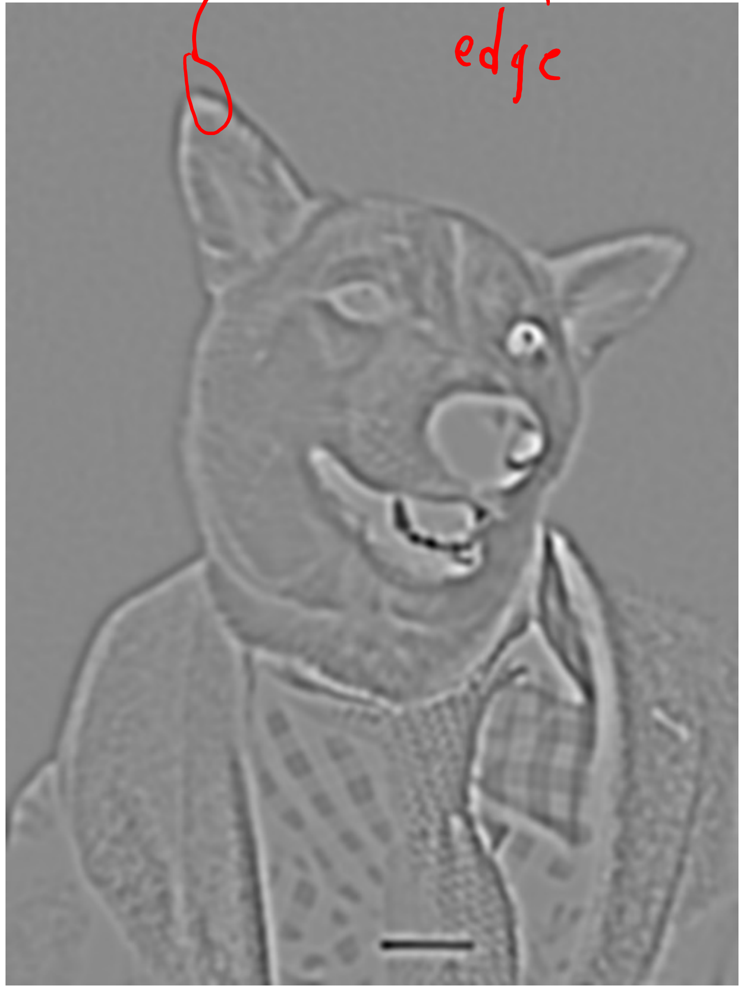
Laplacian of Gaussian

$*$



$=$

(larger variance)



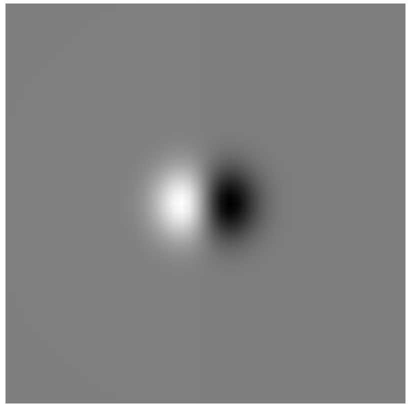
Black/white as sides of edge

Similar preprocessing may be done in basal ganglia and LGN.

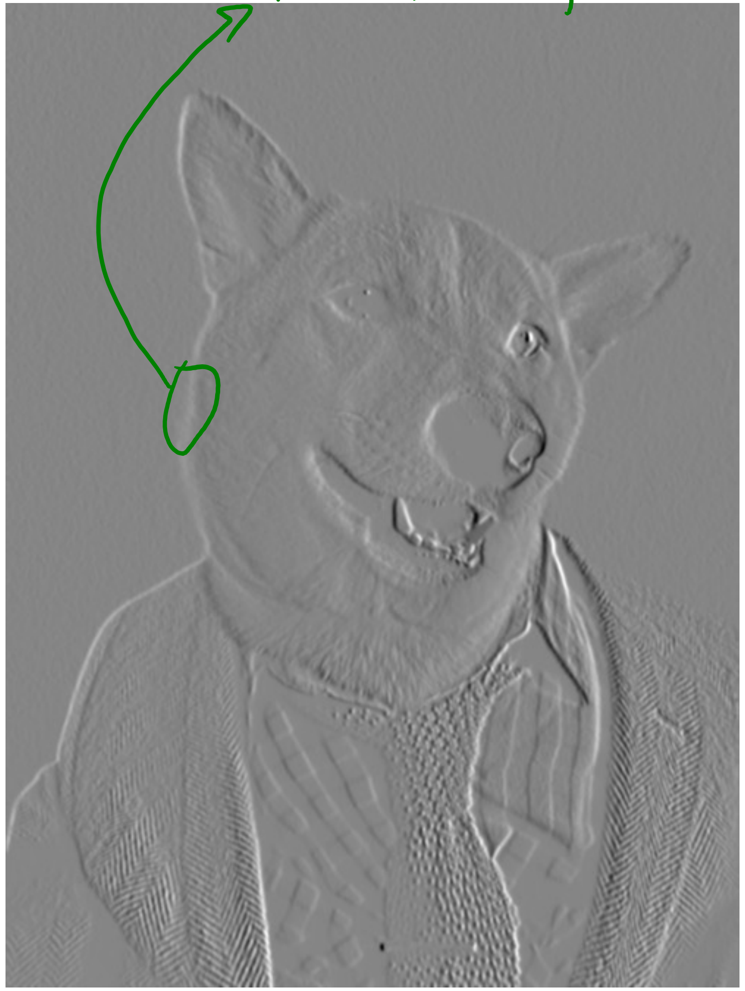
Image Convolution Examples

Gabor Filter
(Gaussian multiplied by sine or cosine)

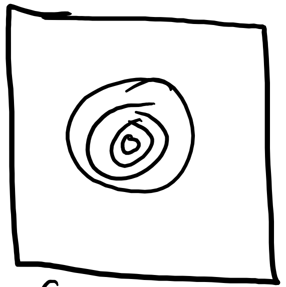
$*$



$=$

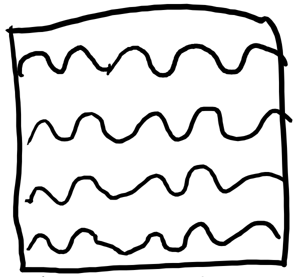


$\parallel$



Gaussian

$.*$

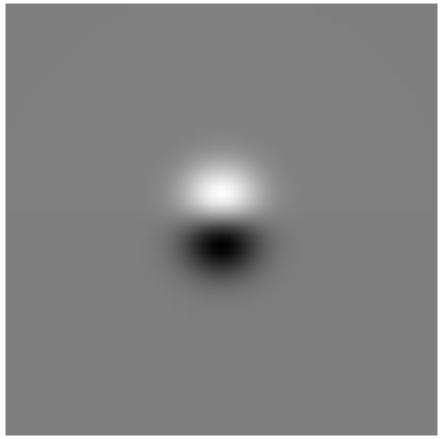


Parallel Sine functions

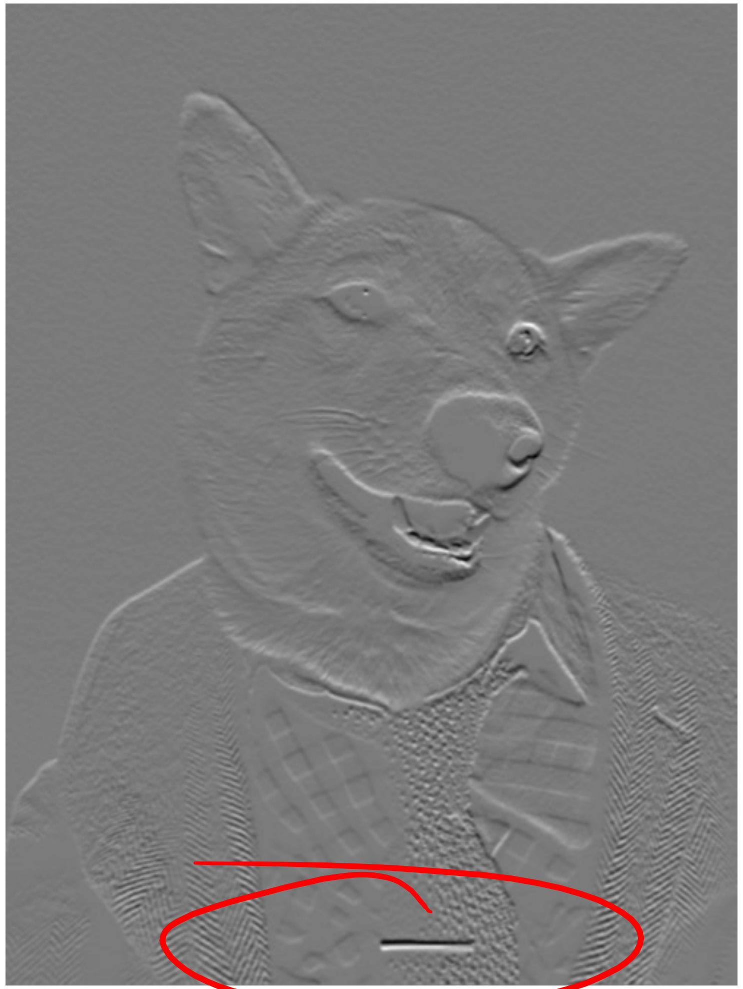
Image Convolution Examples

Gabor Filter
(Gaussian multiplied by
sine or cosine)

$*$



$=$



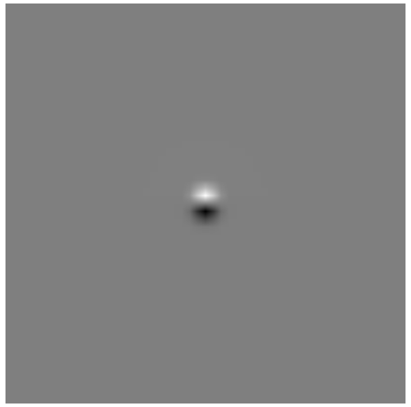
Different orientations of
the sine/cosine let us
detect changes with different
orientations.

$\rightarrow$ 2d derivatives have a direction.

Image Convolution Examples

Gabor Filter
(Gaussian multiplied by sine or cosine)

$*$



(smaller variance)

$=$

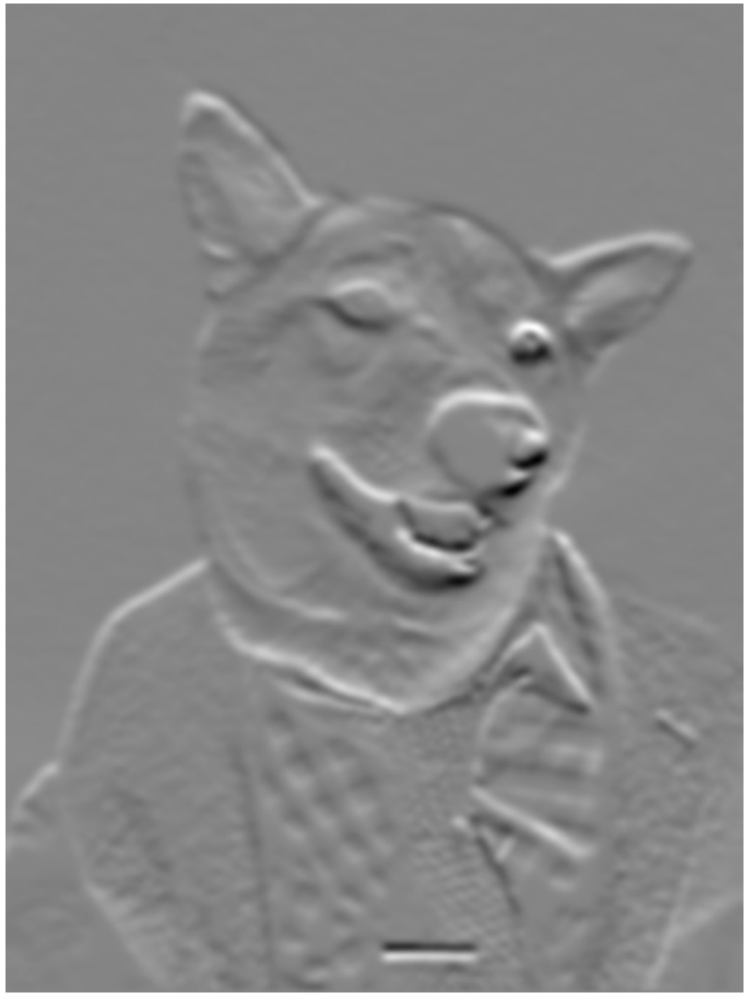
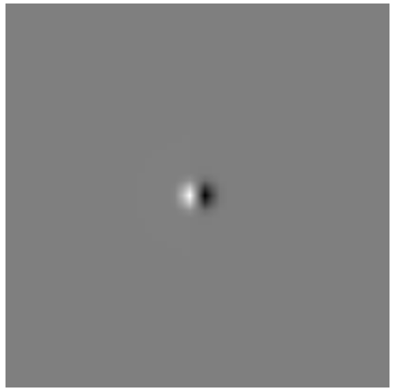


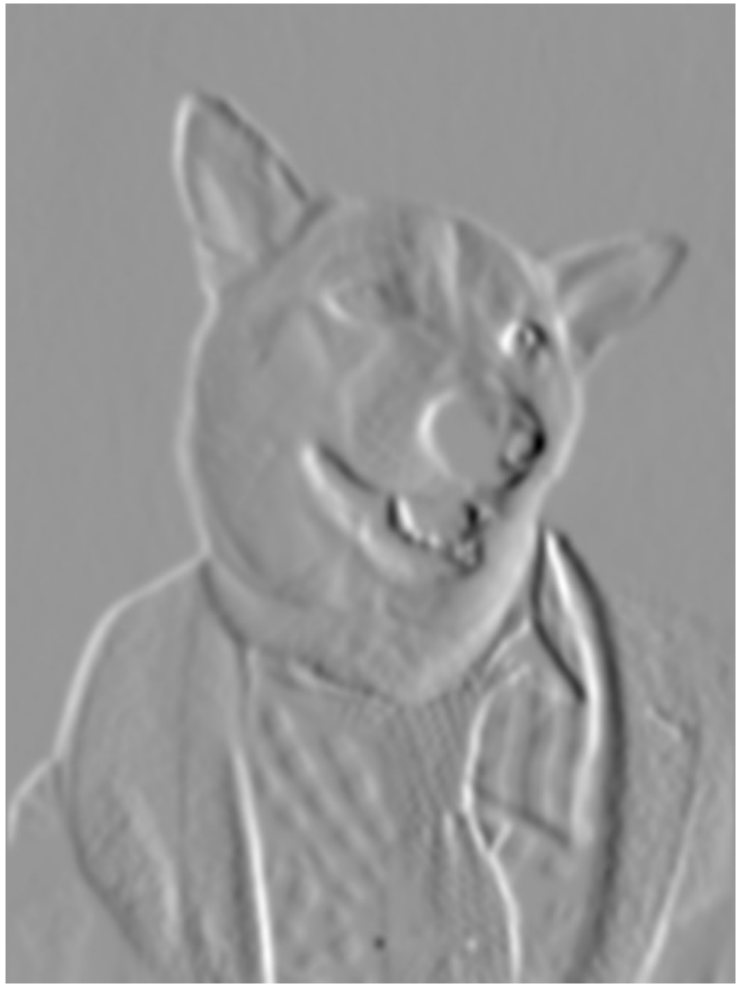
Image Convolution Examples

Gabor Filter
(Gaussian multiplied by sine or cosine)

$*$



$=$



(smaller variance)

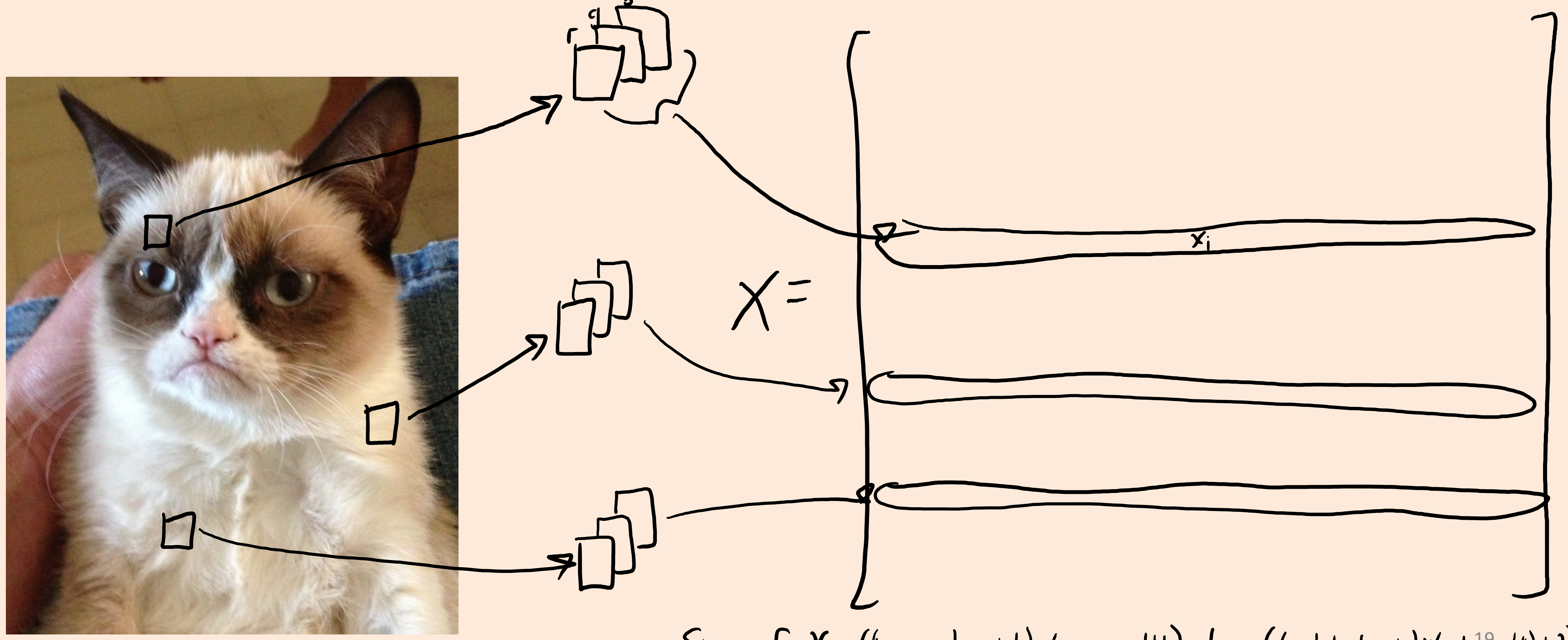
Vertical orientation

- Can obtain other orientations by rotating.
- May be similar to effect of V1 "simple cells."

bonus!

Unsupervised Learning of Filters for Image Patches

- Consider building an unsupervised model of image patches:

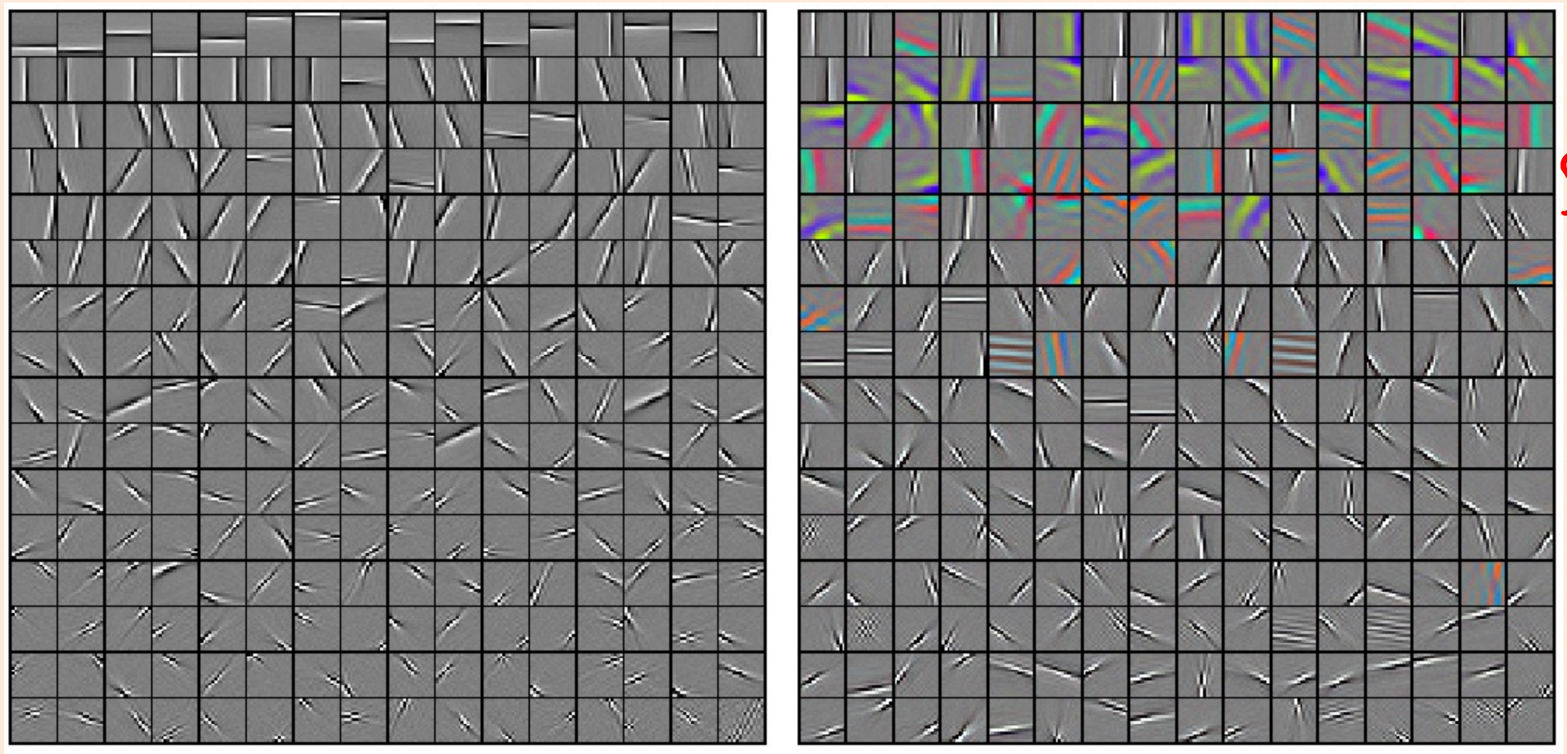


Size of X: (image height)×(image width) by ((patch height)×(patch width)×3)

bonus!

Unsupervised Learning of Filters for Image Patches

- Some methods to do this generate Gaussian/LoG/Gabor filters:
  - These filters are motivated from both neuroscience and ML experiments.

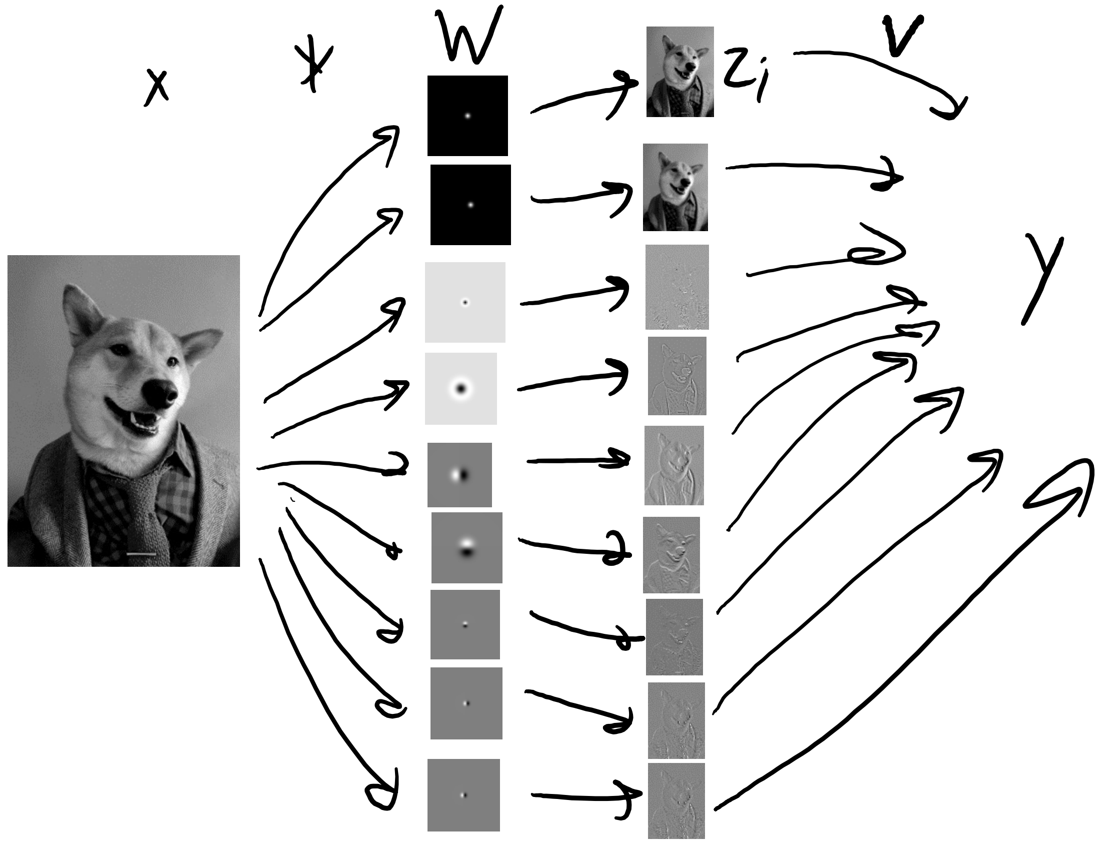


(c) With whitening - gray. (d) With whitening - RGB.

http://lear.inrialpes.fr/people/mairal/resources/pdf/review_sparse_arxiv.pdf

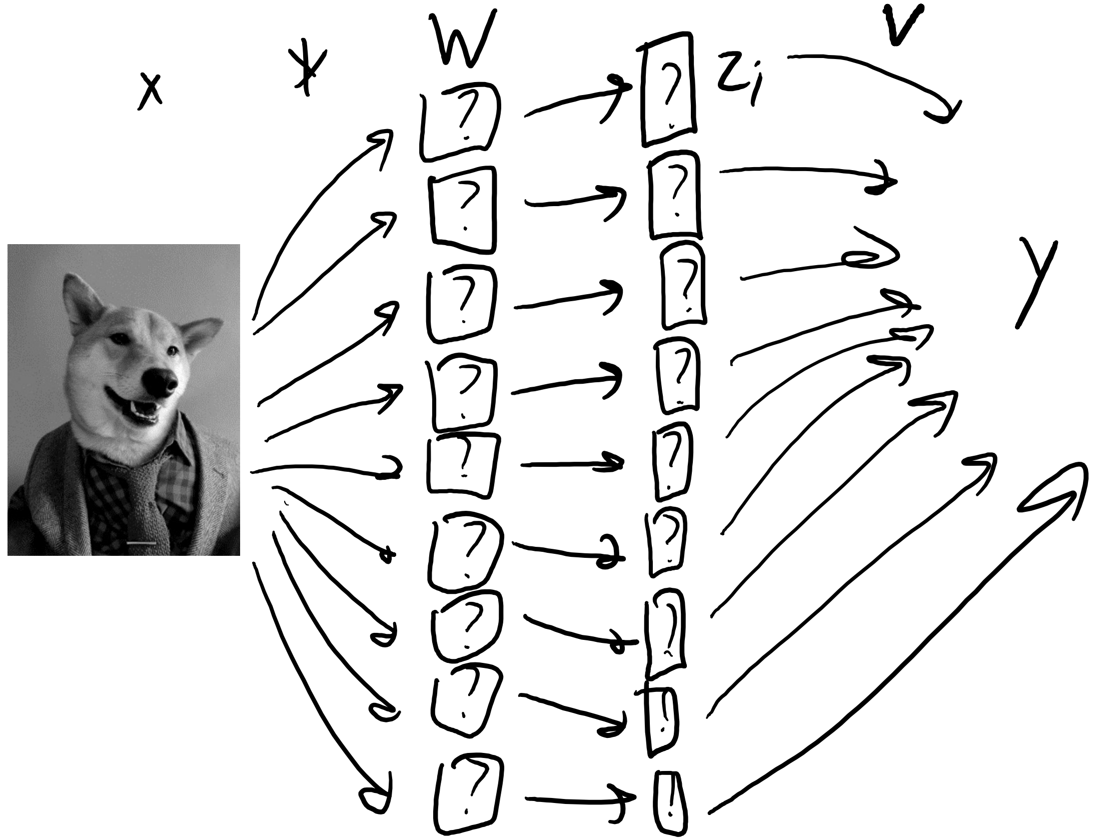
Motivation for Convolutional Neural Networks

- Classic vision methods use fixed convolutions as features:
  - Usually have different types/variances/orientations
  - Can do subsampling or take maxes across locations/orientations/scales



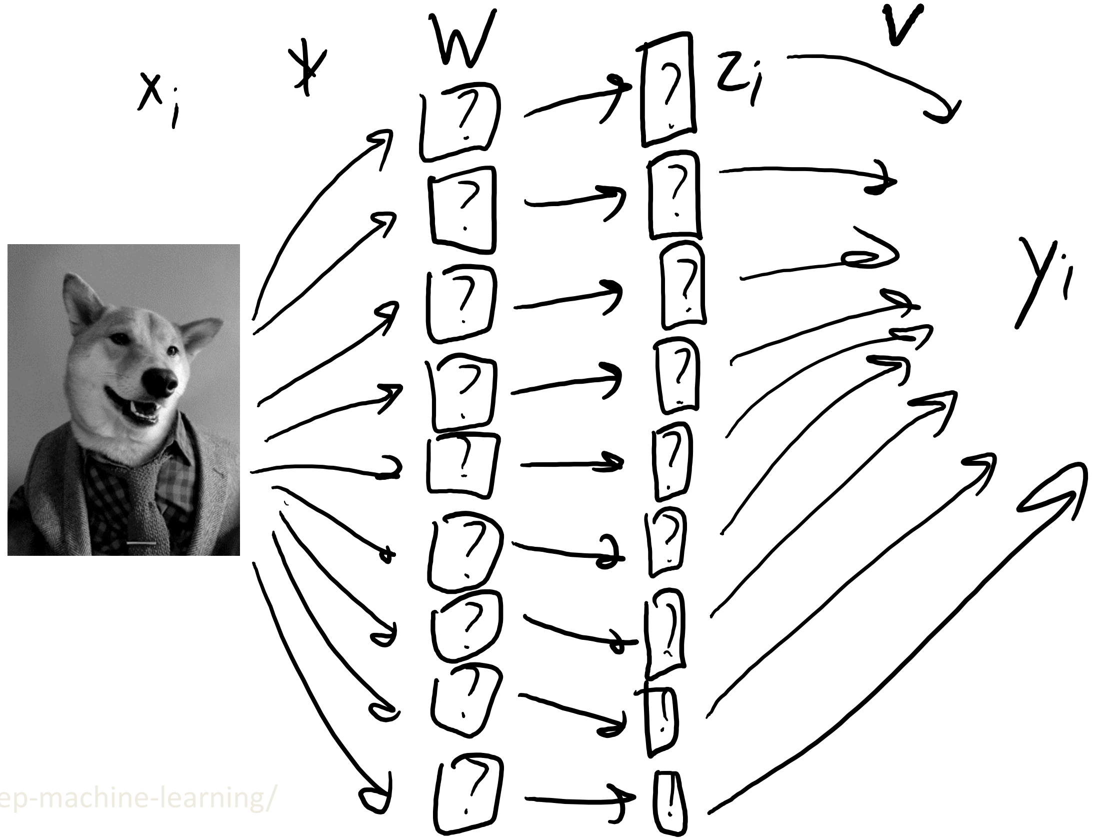
Motivation for Convolutional Neural Networks

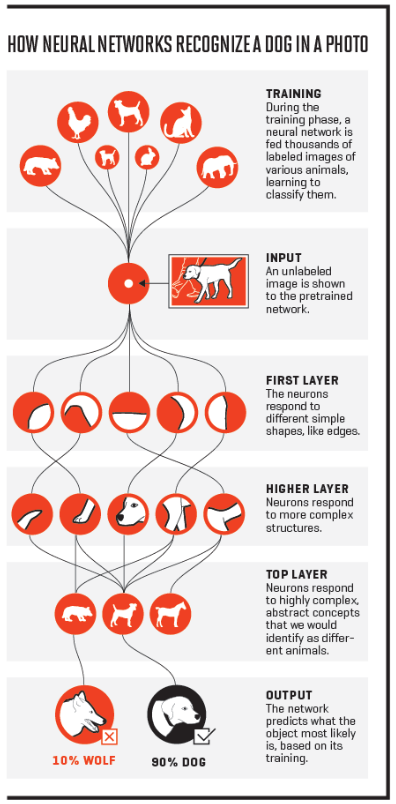
- Convolutional neural networks learn the convolutions:
  - Learning *W* and *v* automatically chooses types/variances/orientations
  - Don't pick from fixed convolutions, but learn the elements of the filters



Motivation for Convolutional Neural Networks

- Convolutional neural networks learn the convolutions:
  - Learning *W* and *v* automatically chooses types/variances/orientations
  - Can do multiple layers of convolution to get deep hierarchical features




HOW NEURAL NETWORKS RECOGNIZE A DOG IN A PHOTO

TRAINING
During the training phase, a neural network is fed thousands of labeled images of various animals, learning to classify them.

INPUT
An unlabeled image is shown to the pretrained network.

FIRST LAYER
The neurons respond to different simple shapes, like edges.

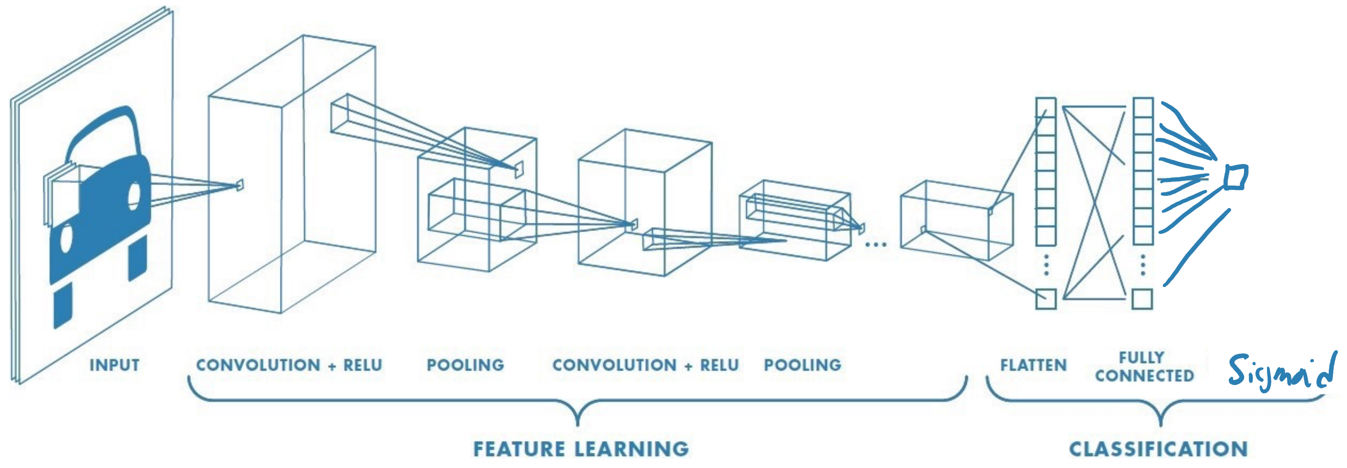
HIGHER LAYER
Neurons respond to more complex structures.

TOP LAYER
Neurons respond to highly complex, abstract concepts that we would identify as different animals.

OUTPUT
The network predicts what the object most likely is, based on its training.

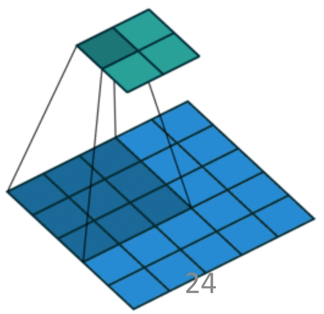
Convolutional Neural Networks

- Classic architecture of a convolutional neural network:



- Convolution layers:
  - Apply convolution with several different filters
  - Sometimes these have a "stride": skip several pixels between applying filter
- Pooling layers:
  - Aggregate regions to create smaller images (usually "max pooling")
- Fully-connected layers: usual "multiplication by W_l" in layer

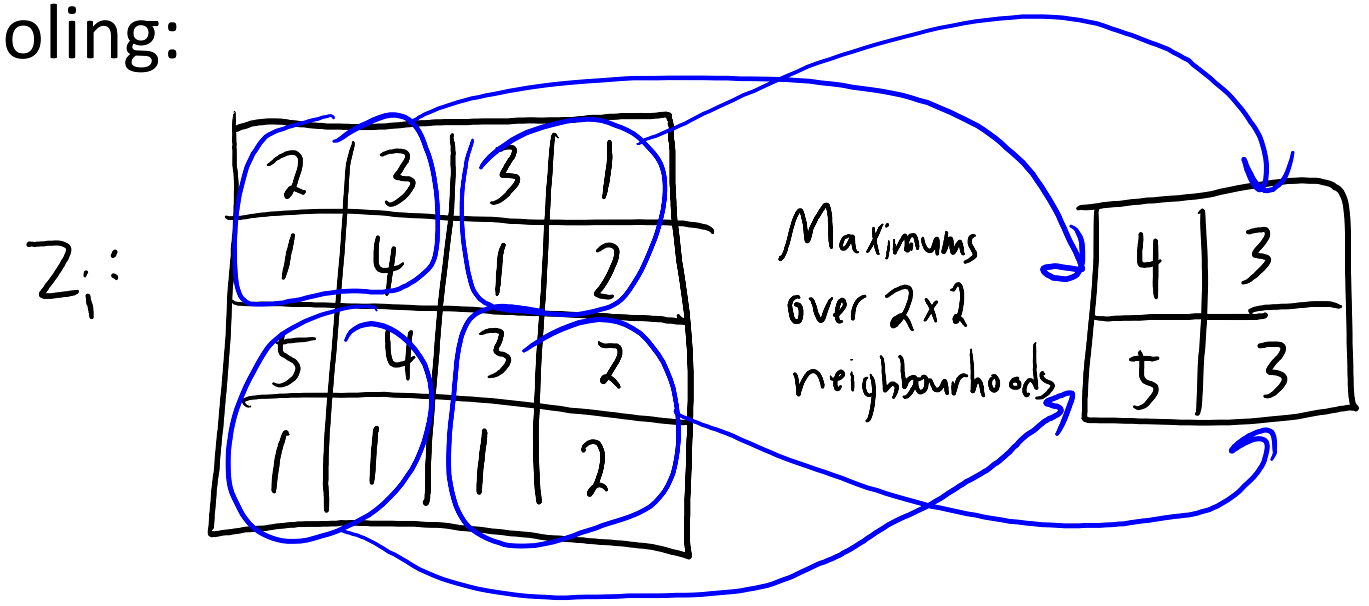
"Stride" of 2



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
https://github.com/vdumoulin/conv_arithmetic

Max Pooling Example

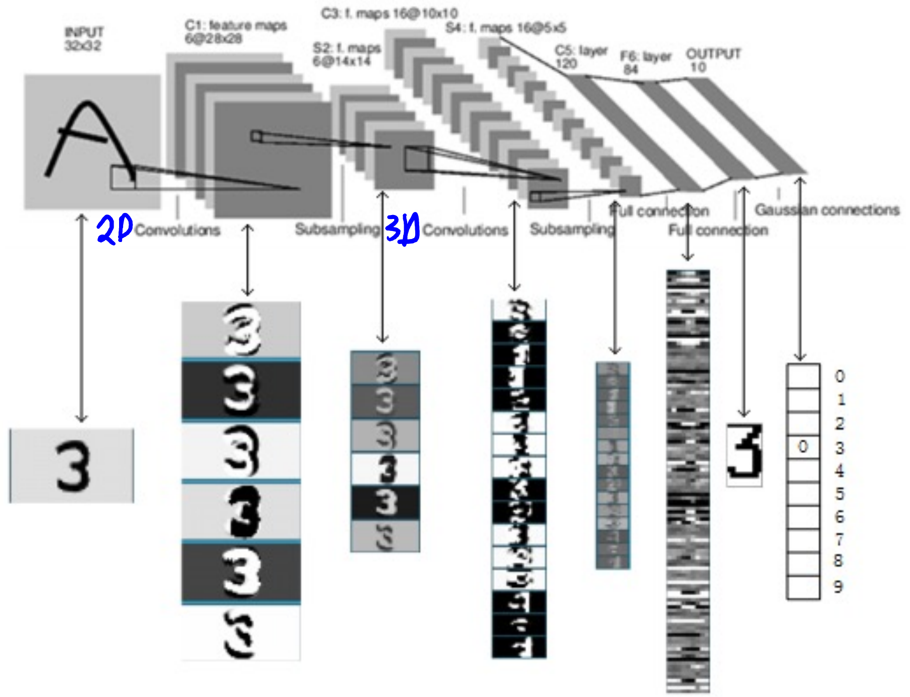
- Max pooling:



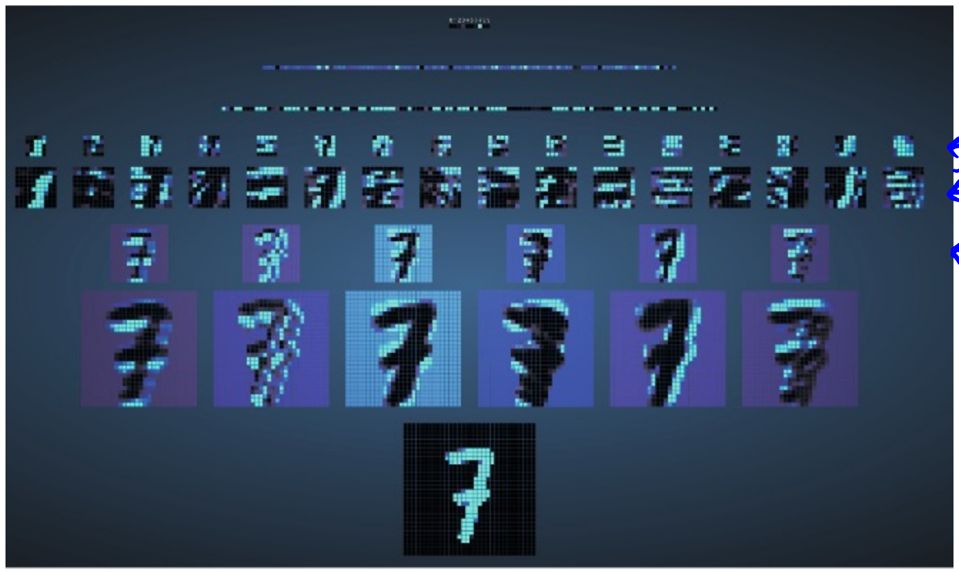
- Decreases size of hidden layer, so we need fewer parameters
  - Gives some local translation invariance:
    - The precise location of max is not important
- This is continuous and piecewise-linear but non-differentiable
  - Like ReLU, we can still optimize this type of objective with SGD

LeNet Convolutional Neural Networks

- Classic convolutional neural network (LeNet):



- Visualizing the "activations":
  - http://scs.ryerson.ca/~aharley/vis/conv
  - http://cs231n.stanford.edu



softmax

2 "fully-connected"

max pooling

3D convolutions

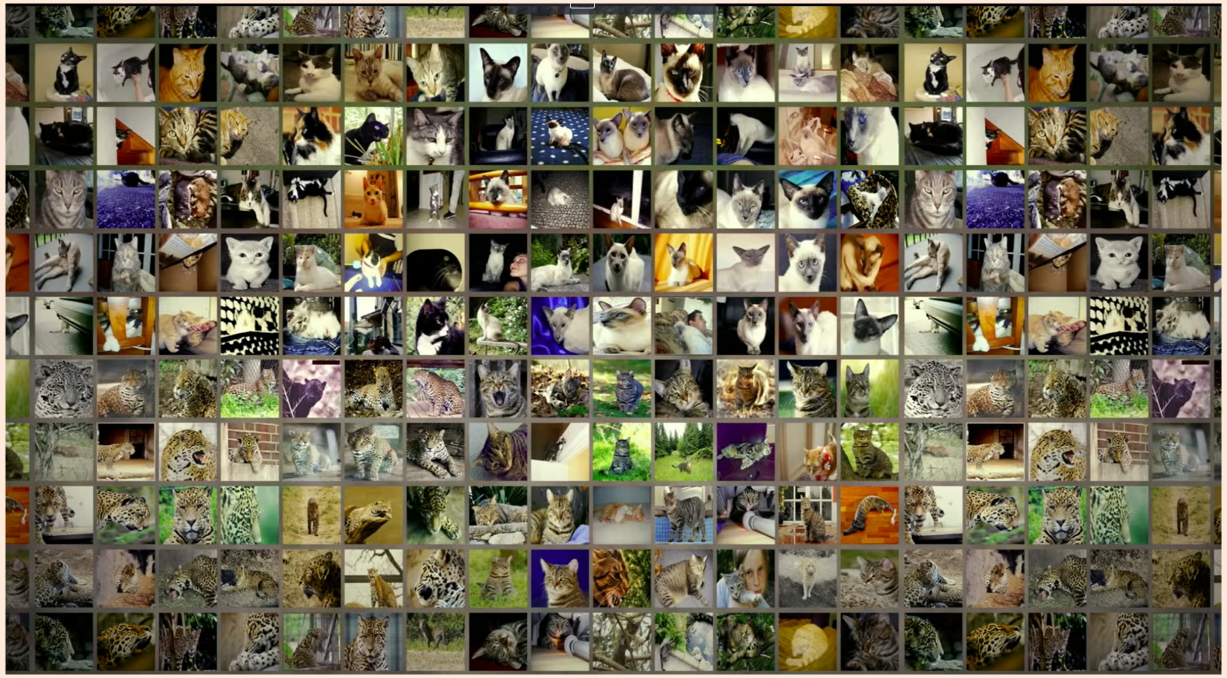
max pooling

2D convolutions

bonus!

ImageNet Competition

- ImageNet: Millions of labeled images, 1000 object classes
  - Task is to classify images into one of the 1000 class labels.
    - We will discuss multi-class classification in Part 2 of the course.
  - Everyone submits their "best" model, winners announced.



https://www.youtube.com/watch?v=40riCqvRoMs

bonus!

AlexNet Convolutional Neural Network

- Modern CNN era started with AlexNet (won 2012 competition):
  - 15.4% error vs. 26.2% for closest competitor
  - 5 convolutional layers
  - 3 fully-connected layers
  - SG with momentum
  - ReLU non-linear functions
  - Data translation/reflection/ cropping
  - L2-regularization + Dropout
  - 5-6 days on two GPUs
    - at the time – could run much faster with current hardware

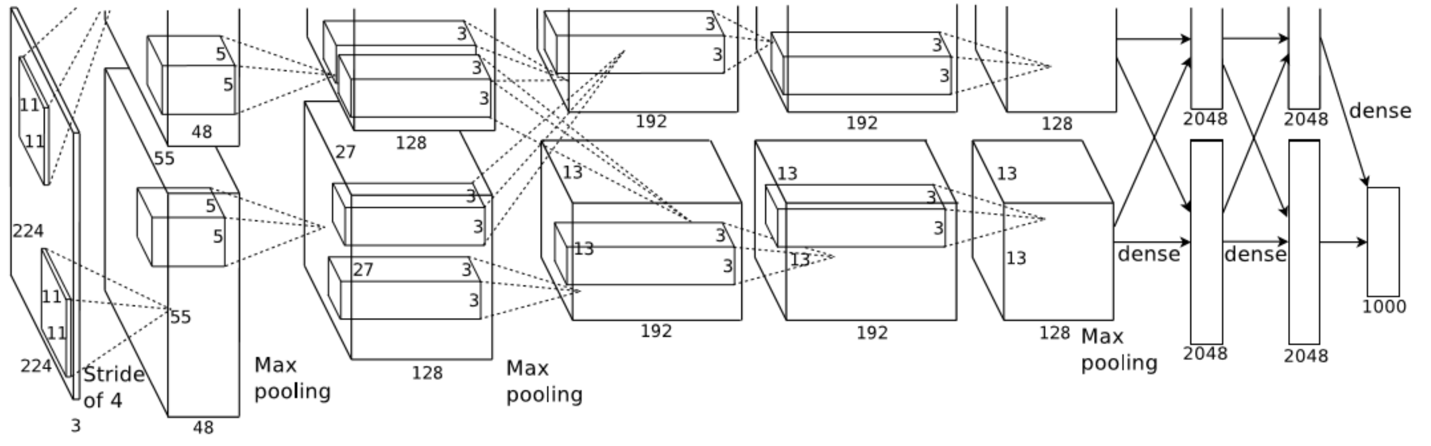
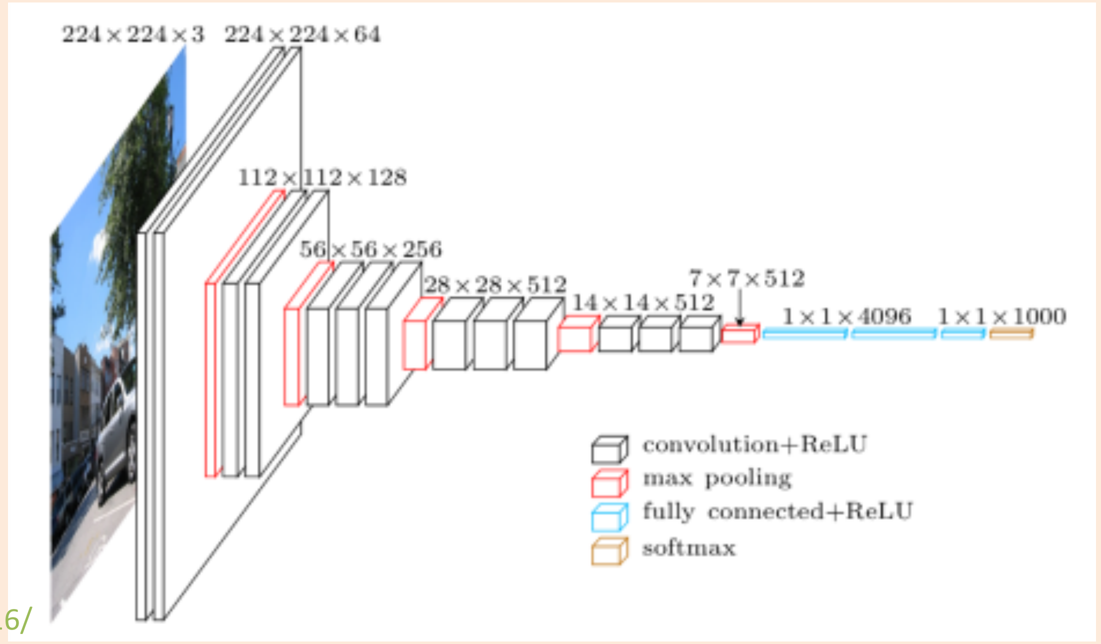


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

bonus!

ImageNet Insights

- Filters and stride got smaller over time
  - Popular VGG approach uses 3x3 convolution layers with stride of 1
    - 3x3 followed by 3x3 simulates a 5x5, and another 3x3 simulates a 7x7, and so on
    - Speeds things up and reduces number of parameters
    - Also increases number of non-linear ReLU operations

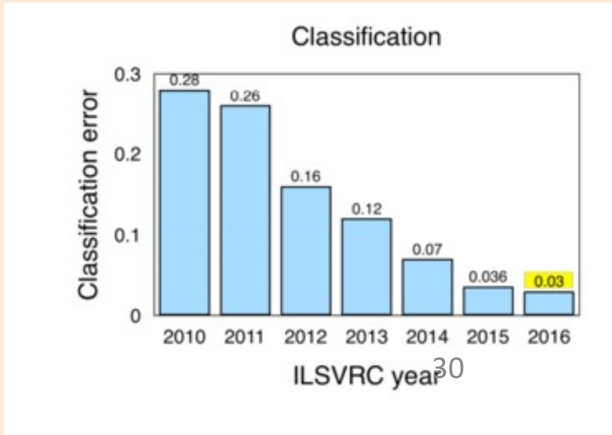


https://www.cs.toronto.edu/~frossard/post/vgg16/

bonus!

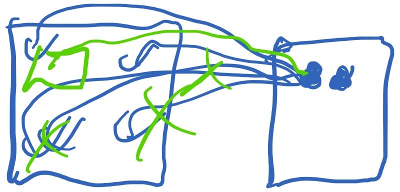
ImageNet Insights

- Filters and stride got smaller over time.
  - Popular VGG approach uses 3x3 convolution layers with stride of 1.
  - GoogLeNet used multiple filter sizes ("inception layer"), but not as popular.
- Eventual switch to "fully-convolutional" networks.
  - No fully connected layers.
- ResNets allow easier training of deep networks.
  - Won all 5 tasks in 2015, training 152 layers for 2-3 weeks on 8 GPUs.
- Ensembles help.
  - 2016 winner combined predictions of previous networks.
- Competition ended in 2017!



http://www.themtank.org/a-year-in-computer-vision

Discussion of CNNs



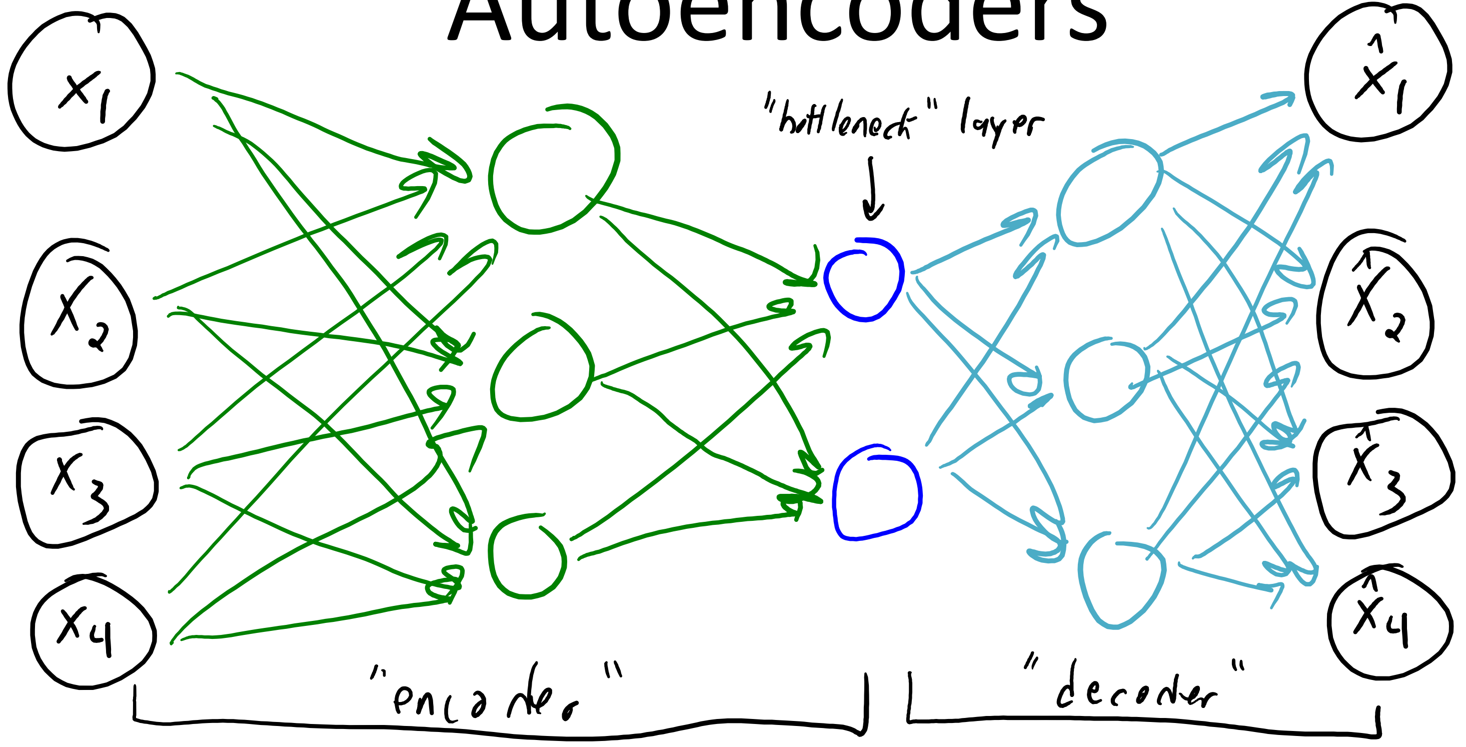
- Convolutional layers reduce the number of parameters in two different ways:
  - Each hidden unit only depends on small number of inputs from previous layer
  - We use the same filters across the image
    - So we do not learn a different weight for each "connection" like in classic neural networks

- CNNs give some amount of translation invariance/equivariance:
  - Because the filters are used across the image, they can detect a pattern anywhere in the image
    - Even in image locations where the pattern has never been seen
  - The pooling layer can also give some local invariance, against small translations of the image

- CNNs are not only for images!
  - Can use CNNs for 1D sequences like sound or language
  - Can use CNNs for 3D objects like videos or medical image volumes
  - Can use CNNs for graphs
- But you do need some notion of "neighbourhood" for convolutions to make sense.

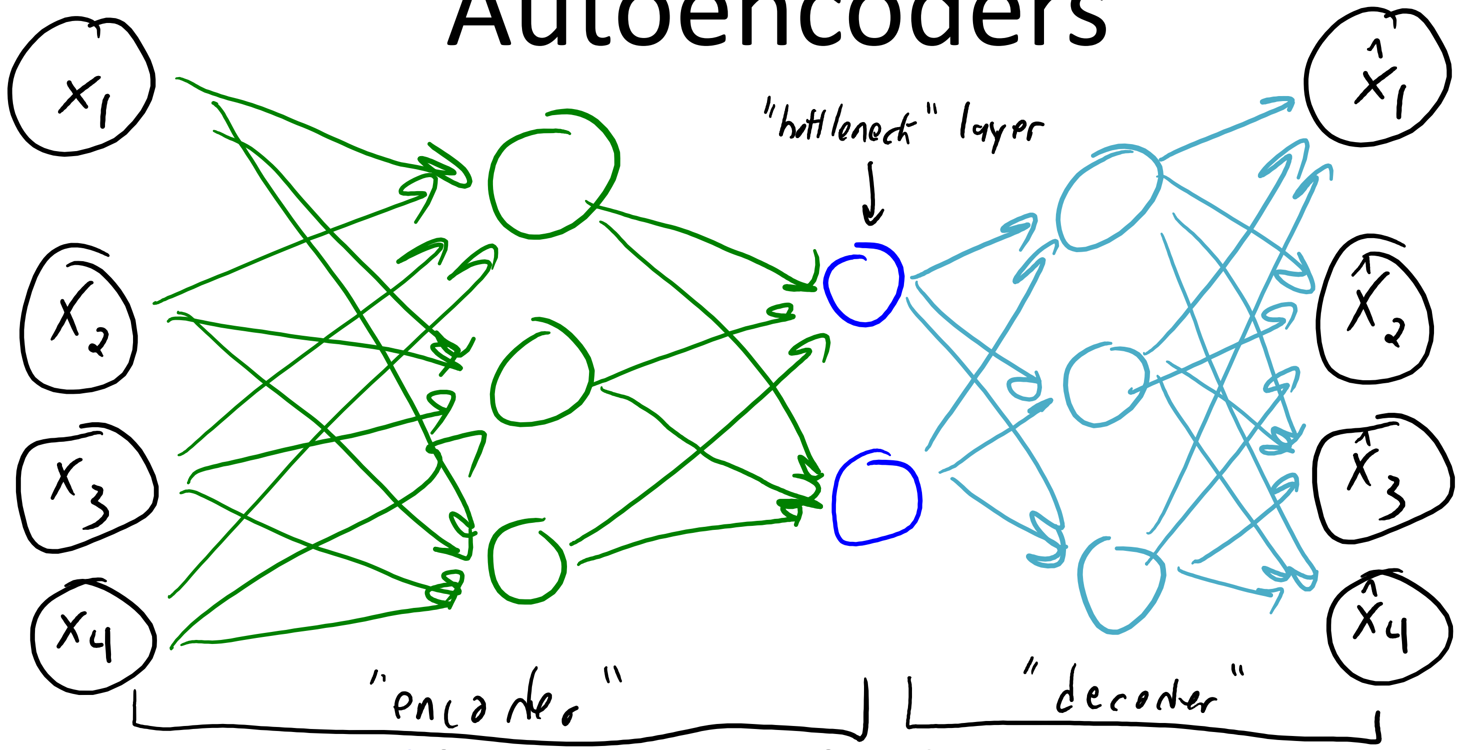
Next Topic: Autoencoders

Autoencoders



- Autoencoders try to make their output the same as the input
  - Usually have a bottleneck layer with dimension $k <$ input d
  - First layers "encode" the input into bottleneck
  - Last layers "decode" the bottleneck into a (hopefully valid) input

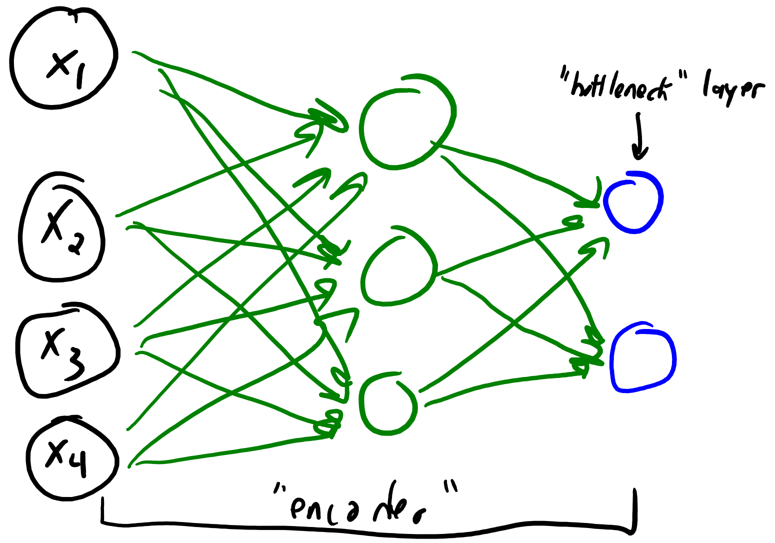
Autoencoders



- This is an unsupervised learning method.
  - There are no labels *y*
- Relationship to principal component analysis (PCA):
  - With squared error and linear network, equivalent to PCA
    - Size of bottleneck layer gives number of latent factors *k* in PCA
  - With non-linear transforms: a non-linear/deep generalization of PCA

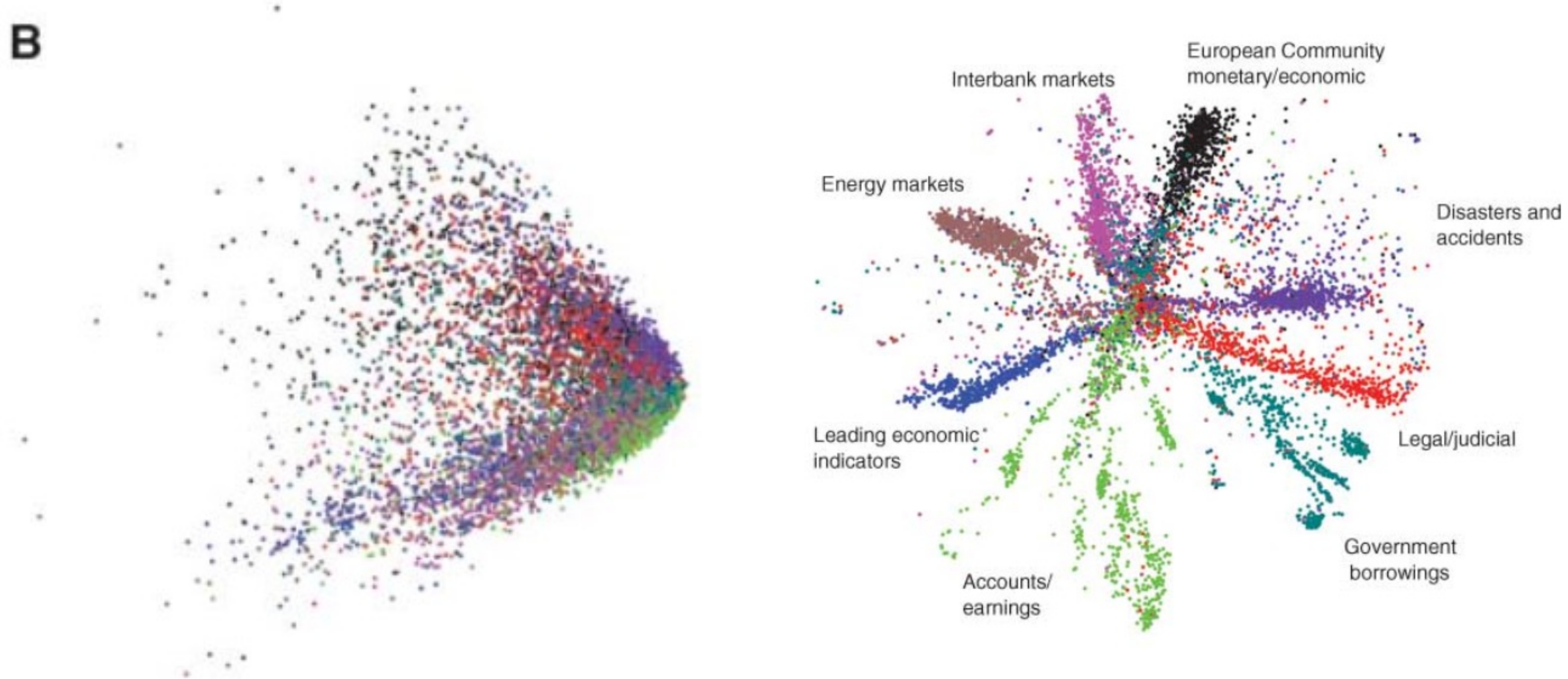
Encoder as Learning a Representation

- Consider the encoder part of the network:
  - Takes features x^i and makes low-dimensional z^i



- Ways you could use the encoder:
  - Use z^i as compressed input (reduce memory needed)
  - Set bottleneck size to 2, and plot the $z^{(i)}$ to visualize the data
  - Try to interpret what the bottleneck features $z^{(i)}$ mean
  - Use the $z^{(i)}$ as features for supervised learning
    - For the special case of PCA and regression with L2 loss, this is called "partial least squares"
  - You could add a supervised y^i to final layer of trained autoencoder + fit with SGD
    - This is called "unsupervised pre-training"
    - If you use unlabeled data to do this initialization, an example of "self-supervised" learning
      - Usually it is easier to get a lot of unlabeled data than it is to get labeled data.

PCA vs. Deep Autoencoder (Document Data)

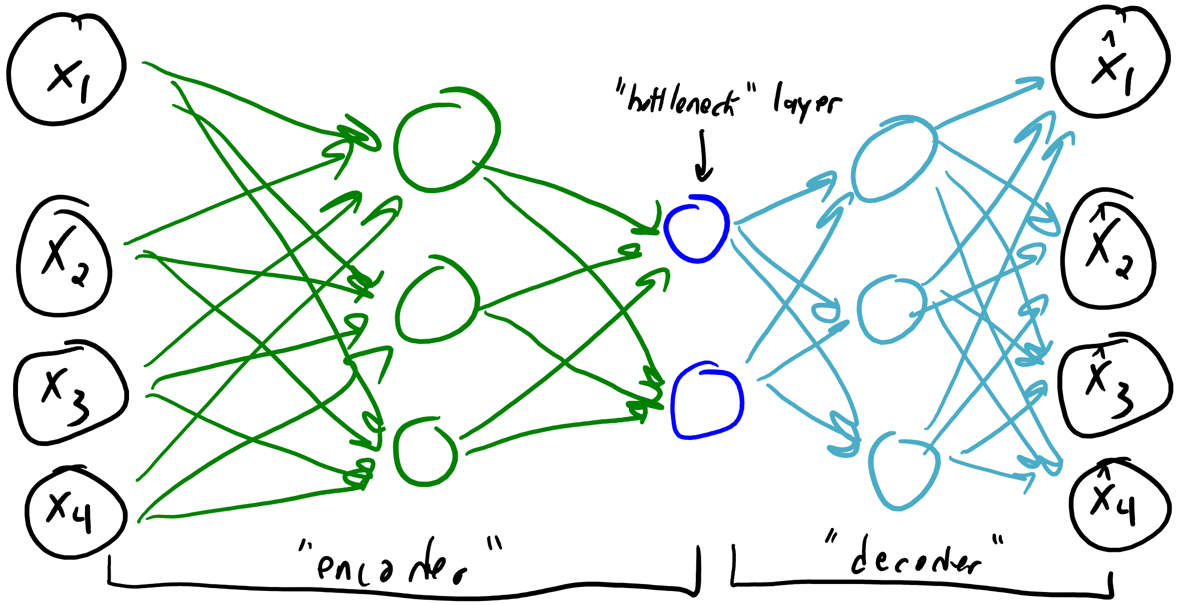


https://www.cs.toronto.edu/~hinton/science.pdf

(these days t-SNE is the usual way to make visualizations like this; see these guidelines)

Unsupervised pre-training

- One version of this: semi-supervised learning
  - 1. Train an autoencoder on all of Instagram (e.g.)
  - 2. Use its features for softmax regression on your (small) labeled dataset
- Could also "fine-tune" the whole network;
  this would be one version of unsupervised pre-training



bonus!

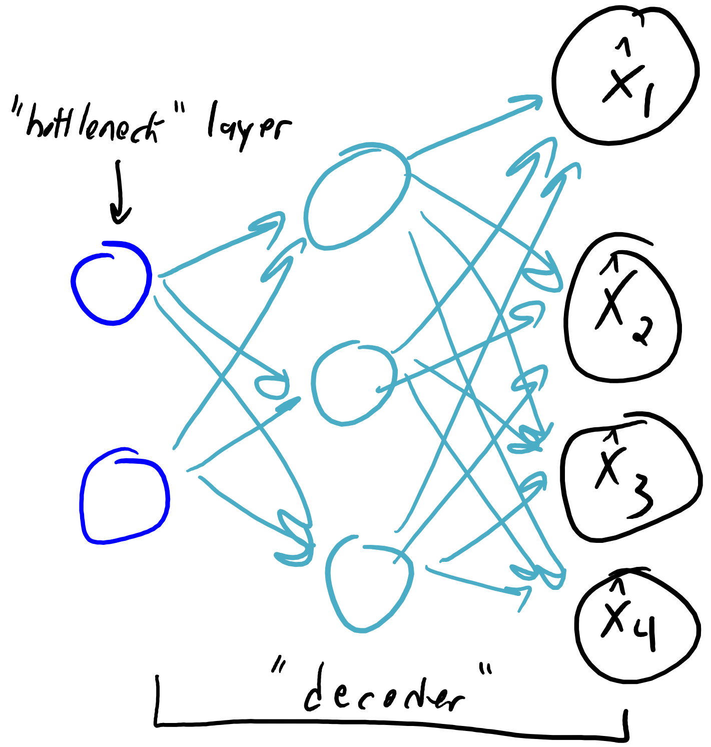
Some clarification about unsupervised pre-training

- What "unsupervised pre-training" used to mean
- Old scheme for deep networks: **stacked** denoising autoencoders
  - Train a two-layer denoising autoencoder
  - Freeze the encoder layer as first layer of your deep net
  - Train a denoising autoencoder on activations from that layer
  - Freeze its encoder as the second layer of your deep net
  - Repeat
  - Fine-tune with SGD at the end

- People don't do this anymore: we can do end-to-end SGD now

Decoder as Generative Model

- Consider the decoder part of the network:
  - Takes low-dimensional $z^{(i)}$ and makes features $\hat{x}^{(i)}$

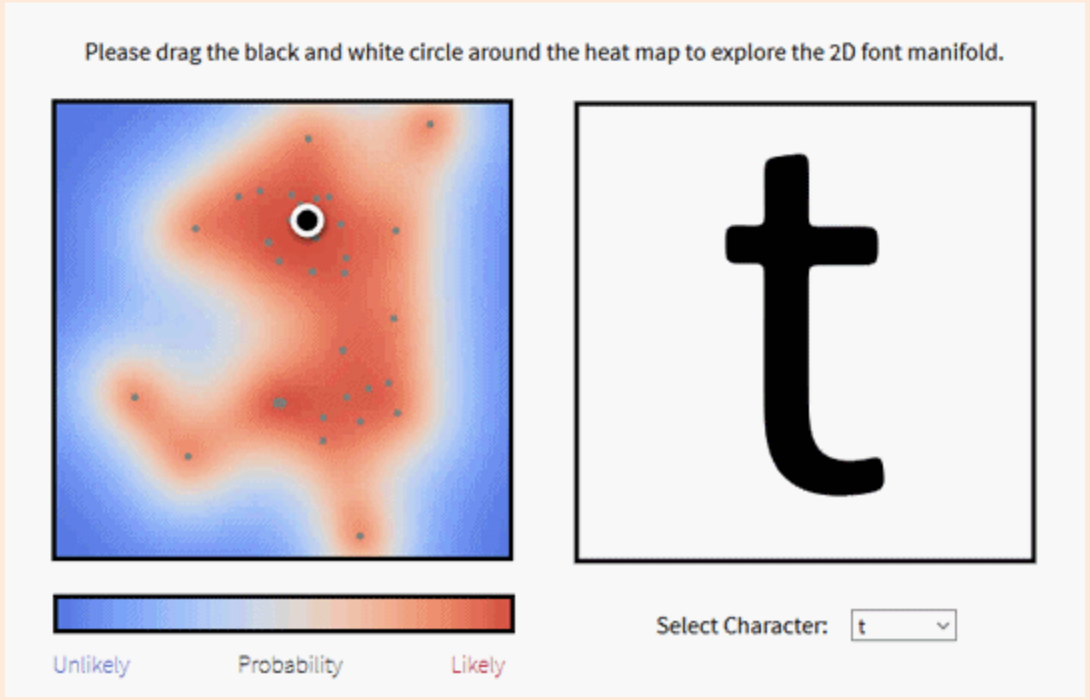


- Can be used for outlier detection:
  - Check distance to original features to detect outliers
- Can be used to generate "new data":
  - If the decoder is good, new values of *z* that "look like real *z*" should decode into $\hat{x}$ that "look like real *x*"
  - To do this "properly," need to estimate the distribution p(z)
    - This is what "Stable Diffusion" does

bonus!

Font Manifold

- Going from encoding to decoding for different fonts:


Please drag the black and white circle around the heat map to explore the 2D font manifold.
t
Unlikely
Probability
Likely
Select Character: t

- Demo here.
  - The above was generated by a Gaussian process and not an autoencoder.
  - But the decoder part of autoencoders is trying to do something like this.

bonus!

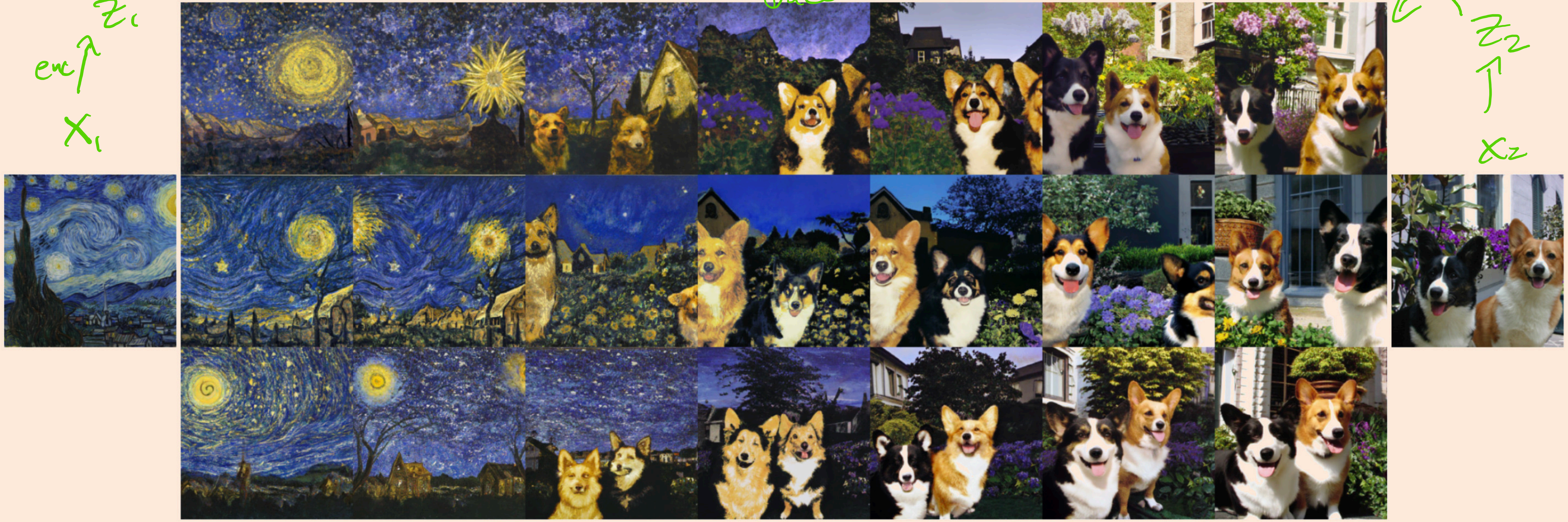
Latent Space Interpolation

$x_1 \xrightarrow{\text{enc}} z_1 \xrightarrow{\text{dec}} \hat{x}_1$

$\frac{1}{2}z_1 + \frac{1}{2}z_2$

$\downarrow \text{dec}$

$x_2 \rightarrow z_2 \xrightarrow{\text{dec}} \hat{x}_2$



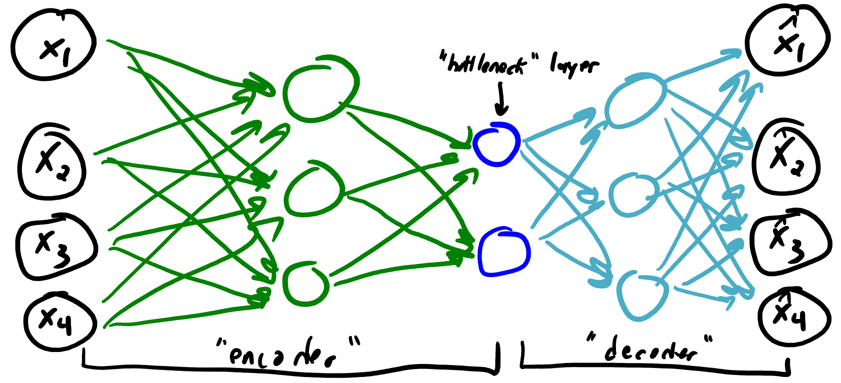
- Encode both ends; decode various points on a line between

https://arxiv.org/abs/2204.06125

Neural Networks with Multiple Outputs

- Previous neural networks we have seen only have 1 output *y*.
- In autoencoders, we have *d* outputs (one for each feature).

$$\left.\begin{aligned}\hat{x}_1 &= v_1^T h(W^3 h(W^2 h(W^1 x)))\\ \hat{x}_2 &= v_2^T h(W^3 h(W^2 h(W^1 x)))\\ &\vdots\\ \hat{x}_d &= v_d^T h(W^3 h(W^2 h(W^1 x)))\end{aligned}\right\} \hat{x} = V h(W^3 h(W^2 h(W^1 x)))$$



- For training, we add up the loss across all *j*:

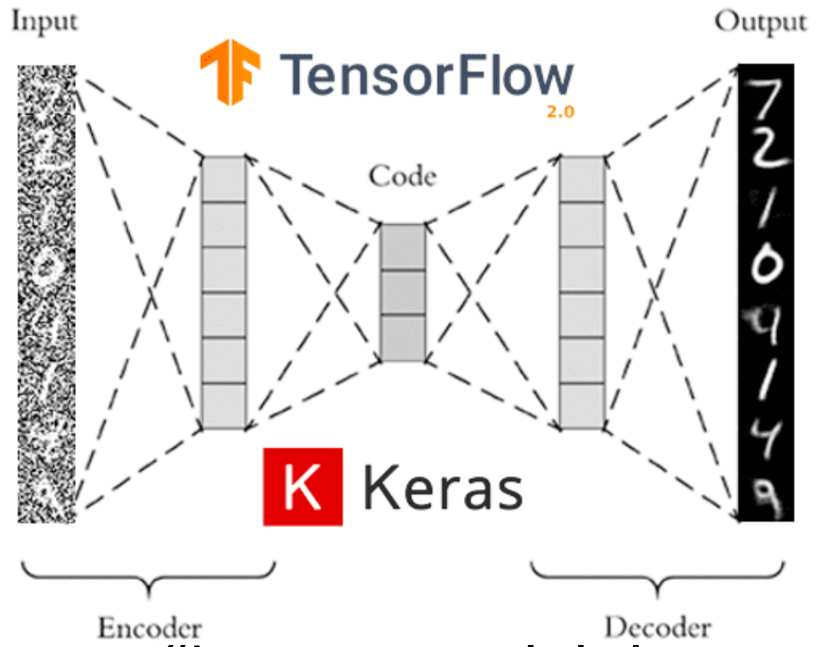
$$\hat{x}_j^i \rightarrow = v_j^T h(W^3 h(W^2 h(W^1 x^i)))$$

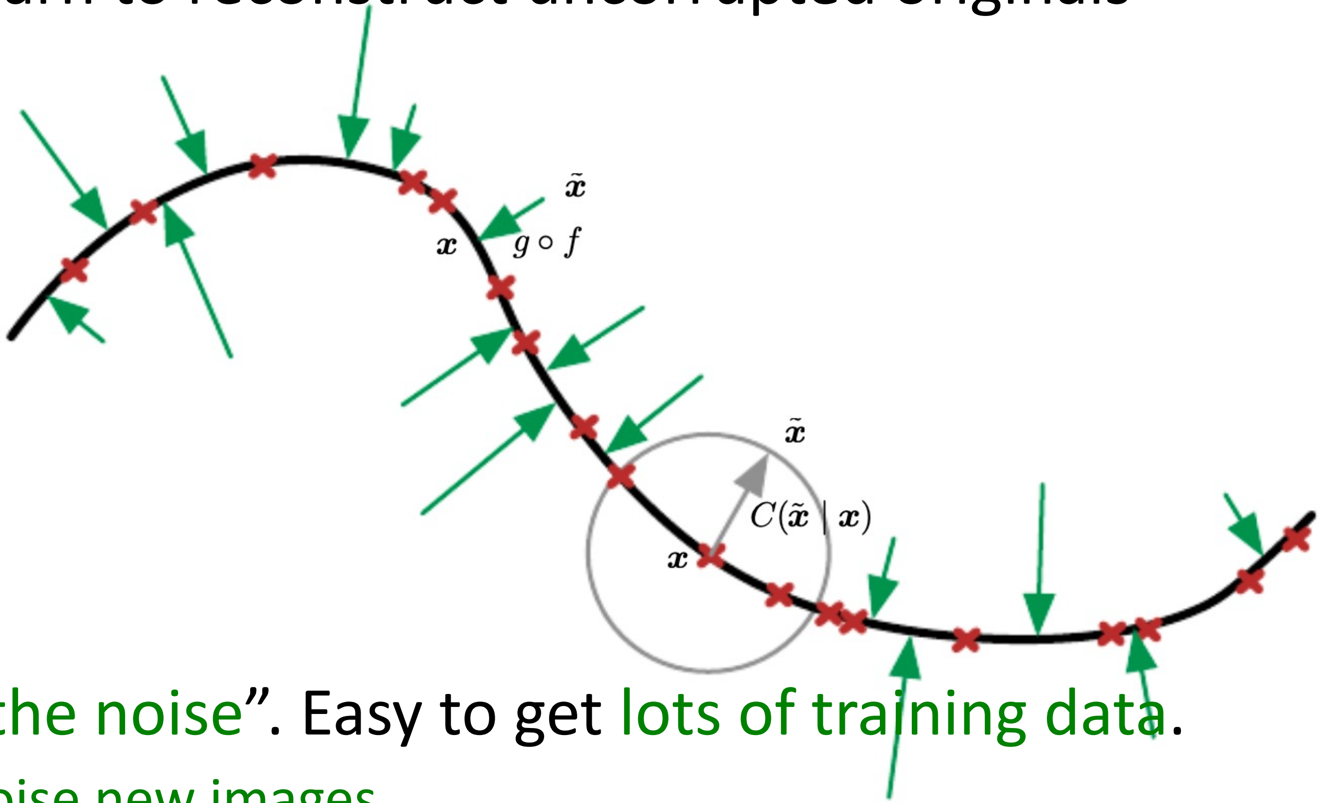
$$f(W^1, W^2, V) = \underbrace{\sum_{i=1}^{n}\sum_{j=1}^{d}(\hat{x}_j^i - x_j^i)^2}_{\text{squared error for continuous } x_j^i} \qquad f(W^1, W^2, V) = \sum_{i=1}^{n}\sum_{j=1}^{d}\underbrace{\log(1+\exp(-\hat{x}_j^i x_j^i))}_{\text{logistic loss for binary } x_j^i \in \{-1,+1\}}$$

- Fit with SGD (sampling random *i*), and usual deep learning tricks can be used
  - Even though network has multiple outputs, *f* is a scalar so autodiff works as before
  - For images, may want to use convolution layers

Denoising Autoencoders

- A common variation on autoencoders is denoising autoencoders:
  - Use "corrupted" inputs, and learn to reconstruct uncorrupted originals



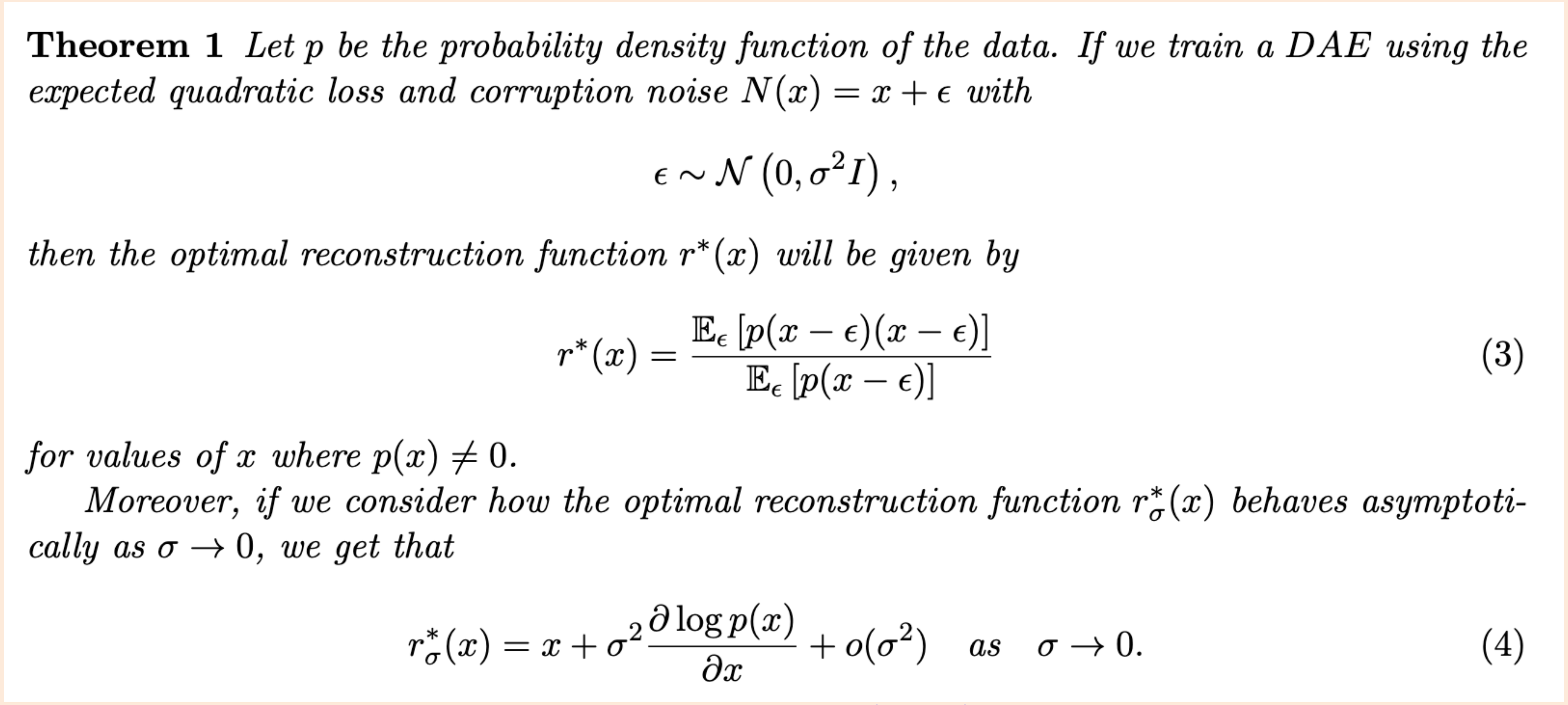


  - "Learn a model that removes the noise". Easy to get lots of training data.
    - You can apply the model to denoise new images.
    - Do not necessarily need a "bottleneck" layer.

https://www.pyimagesearch.com/2020/02/24/denoising-autoencoders-with-keras-tensorflow-and-deep-learning/
https://www.deeplearningbook.org/contents/autoencoders.html

bonus!

What Denoising Autoencoders Learn



Theorem 1 *Let p be the probability density function of the data. If we train a DAE using the expected quadratic loss and corruption noise $N(x) = x + \epsilon$ with*

$$\epsilon \sim \mathcal{N}\left(0, \sigma^2 I\right),$$

then the optimal reconstruction function $r^*(x)$ will be given by

$$r^*(x) = \frac{\mathbb{E}_\epsilon\left[p(x-\epsilon)(x-\epsilon)\right]}{\mathbb{E}_\epsilon\left[p(x-\epsilon)\right]} \tag{3}$$

for values of x where $p(x) \neq 0$.

Moreover, if we consider how the optimal reconstruction function $r^*_\sigma(x)$ behaves asymptotically as $\sigma \to 0$, we get that

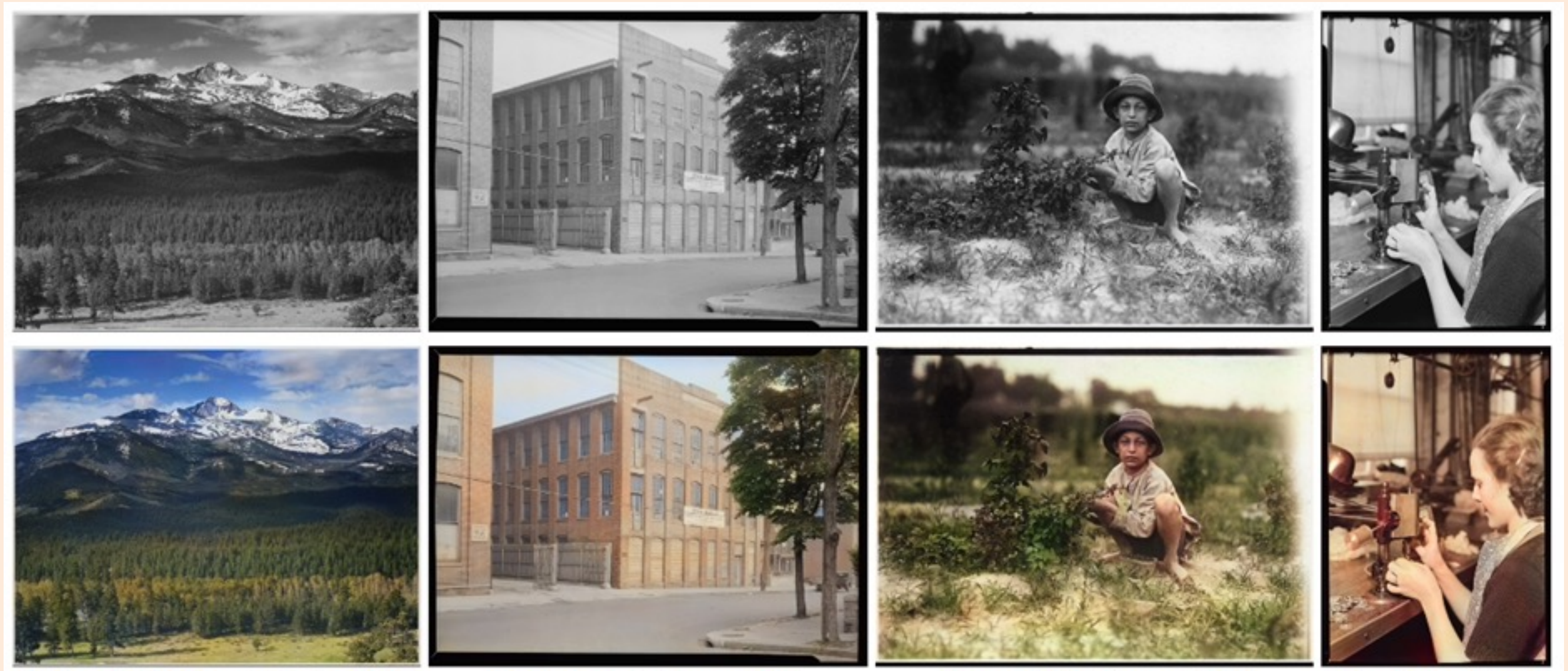
$$r^*_\sigma(x) = x + \sigma^2 \frac{\partial \log p(x)}{\partial x} + o(\sigma^2) \quad as \quad \sigma \to 0. \tag{4}$$

Alain and Bengio (2012)

- Can use to estimate "Hyvärinen score" $\frac{(r^*_\sigma(x) - x)}{\sigma^2} \approx \nabla_x \log p(x)$
- Closely related to diffusion models (later in the course!)

bonus!

Image Colourization


Colorado National Park, 1941 Textile Mill, June 1937 Berry Field, June 1909 Hamilton, 1936

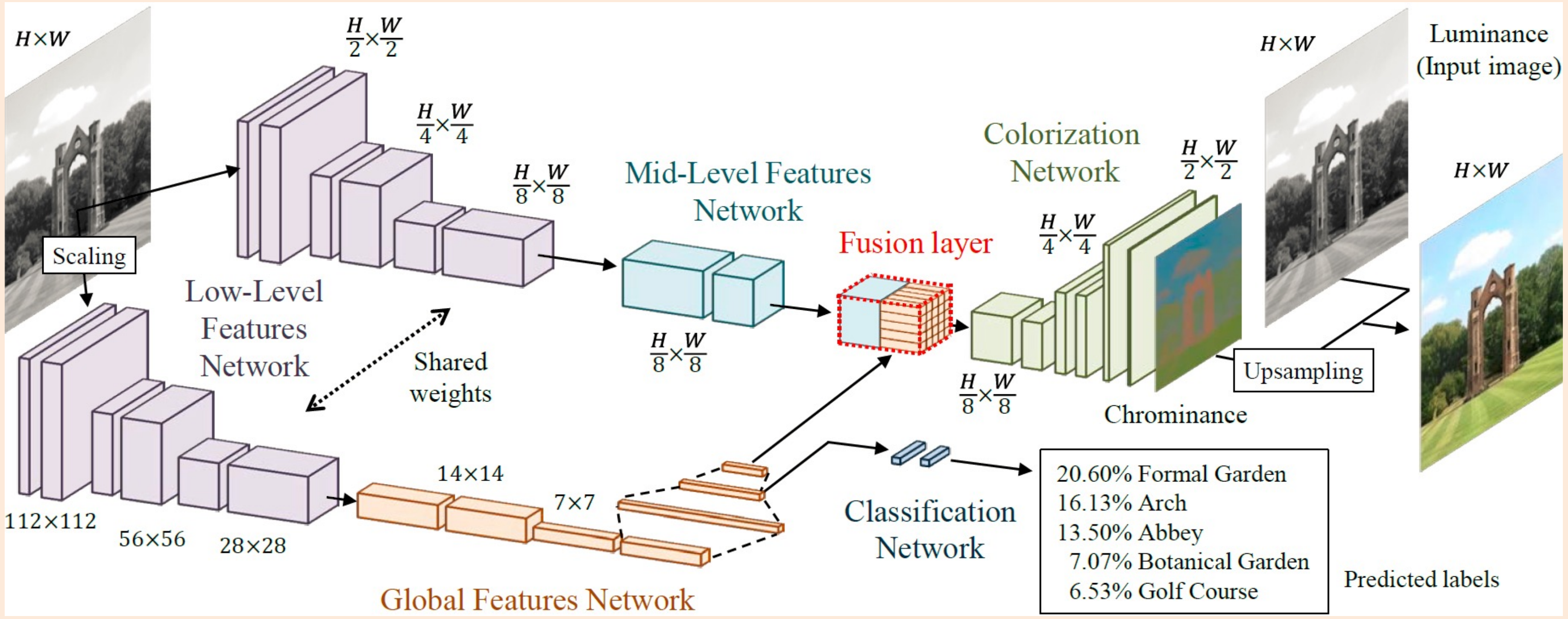
- Gallery: http://iizuka.cs.tsukuba.ac.jp/projects/colorization/extra.html
- Video: https://www.youtube.com/watch?v=ys5nMO4Q0iY

http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/

bonus!

Image Colourization

- Instead of noisy inputs, you use de-coloured inputs:



- Another application is super-resolution:
  - Learn to output a high-resolution image based on low-resolution images.

http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/

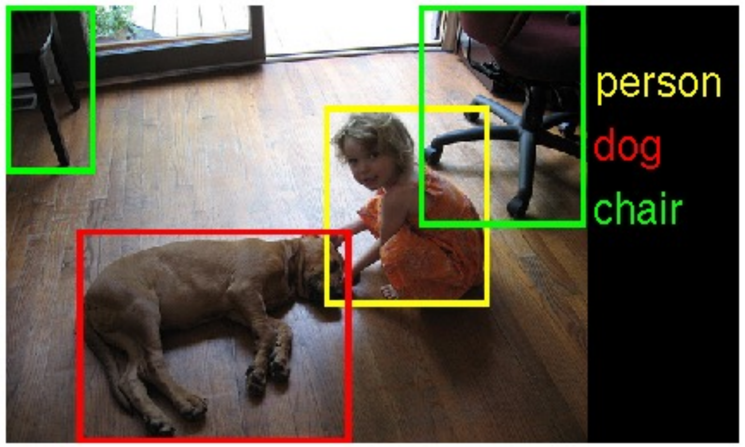
Next Topic: Multi-Label Classification

Motivation: Multi-Label Classification

- Consider multi-label classification:

$$X = \left.\underbrace{\begin{bmatrix} \quad & \quad & \quad \\ \quad & \quad & \quad \\ \quad & \quad & \quad \end{bmatrix}}_{d}\right\} n \qquad Y = \left.\underbrace{\begin{array}{ccccc} \text{cat} & \text{dog} & \text{person} & \text{chair} & \text{mouse} \\ 1 & -1 & 1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{array}}_{k}\right\} n$$

- Which of the *k* objects are in this image?
  - There may be more than one "correct" class label.



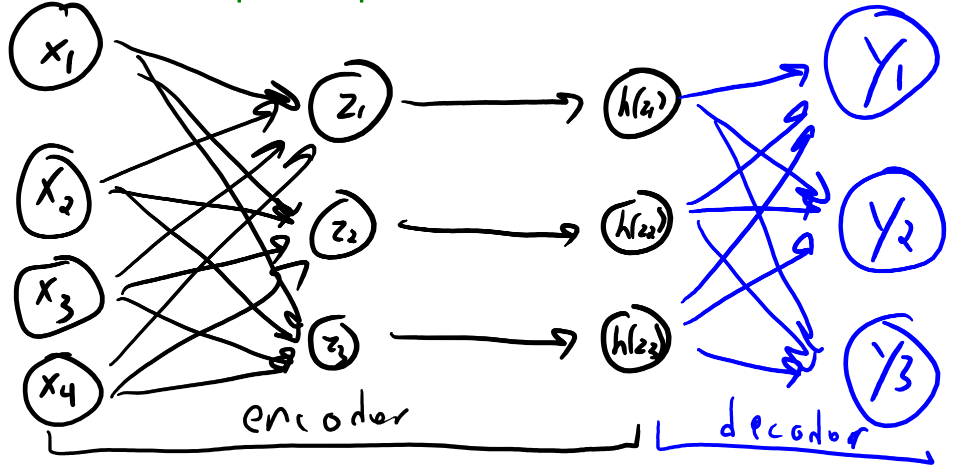
http://image-net.org/challenges/LSVRC/2013/

Independent Classifier Approach

- One way to build a multi-label classifier:
  - Train a classifier for each label
    - Train a neural network that predicts +1 if the image contains a dog, and -1 otherwise
    - Train a neural network that predicts +1 if the image contains a cat, and -1 otherwise
    - …
  - To make predictions for the *k* classes, concatenate predictions of the *k* models
- Can think of this as a "product of independent classifiers"
- Drawbacks:
  - Lots of parameters: *k**(number of parameters for base classifier).
  - Each classifier needs to "relearn from scratch"
    - Each classifier needs to learn its own Gabor filters, how corners and light works, and so on
    - A lot of visual features for "dog" might also help us predict "cat"

Encoding-Decoding for Multi-Label Classification

- Multi-label classification with an encoding-decoding approach:
  - Input is connected to a hidden layer.
  - Hidden layer is connected to multiple output units.



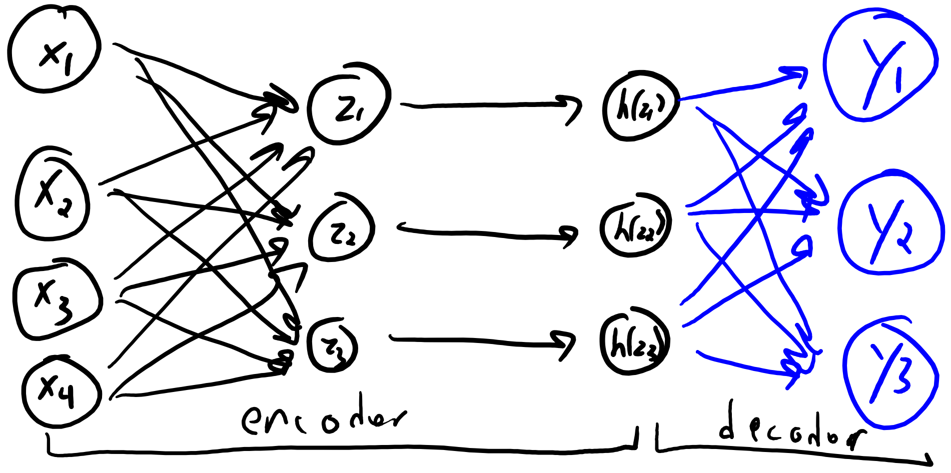
$$\hat{y}_1 = v_1^T h(Wx)$$
$$\hat{y}_2 = v_2^T h(Wx)$$
$$\hat{y}_3 = v_3^T h(Wx)$$

- Prediction: compute hidden layer, compute activations, compute output:

$$\hat{y} = V\, h(Wx)$$

- Number of parameters and cost is O($dm + mk$) for k classes and m hidden units.
  - If we trained a separate network for each class, number of parameters and cost would be O(kdm) (for ‘W’ for each class)
- Might have multiple layers, convolution layers, and so on.
- No need to have a “bottleneck” layer – ”encoder”/”decoder” is just terminology

Encoding-Decoding for Multi-Label Classification



- We usually assume that the classes are independent given last layer:

$$p(y_1, y_2, \dots, y_k \mid x_1, x_2, \dots, x_d, W, V) = p(y_1 \mid x_1, x_2, \dots, x_d, W, v_1)\, p(y_2 \mid x_1, x_2, \dots, x_d, W, v_2) \cdots p(y_k \mid x_1, x_2, \dots, x_d, W, v_k)$$

with: $p(y_1 = 1 \mid x, W, v) = \underbrace{\frac{1}{1+\exp(-v_1^T h(Wx))}}_{\theta_1}$ $\quad p(y_2 = 1 \mid x, W, v) = \underbrace{\frac{1}{1+\exp(-v_2^T h(Wx))}}_{\theta_2}$ ...

  - Conditioned on features/parameters, this is ultimately a fancy product of Bernoullis model:
    - $p(y_1, y_2, \dots, y_k \mid x, W, V) = p(y_1 \mid x, W, V)p(y_2 \mid x, W, V) \cdots p(y_k \mid x, W, V)$, where $p(y_c = 1 \mid x, W, V) = \theta_c$.
    - This makes decoding and other inference problems easy: you do inference on each y_c independently.

Encoding-Decoding for Multi-Label Classification

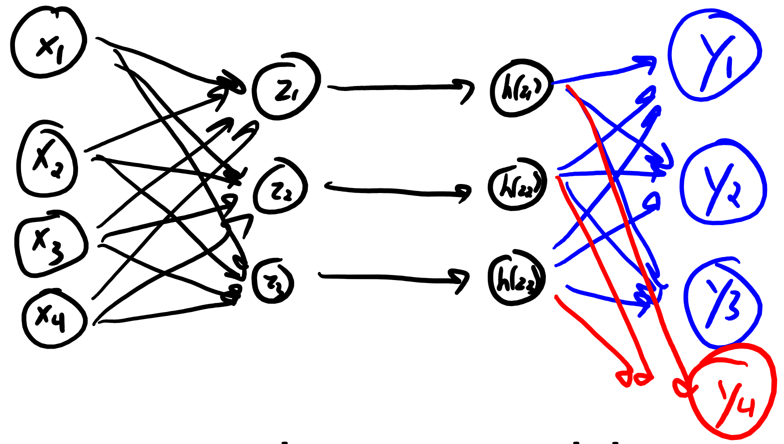
- The negative log-likelihood we optimize for MLE:

$$f(W,V) = \sum_{i=1}^{n} \sum_{c=1}^{k} \log(1 + \exp(-y_c^i v_c^T h(Wx^i)))$$

- Use backpropagation or AD to compute gradient, train by SGD.
  - You randomly sample a training example *i* and compute gradient for all labels
  - The updates of *W* lead to features that are useful across classes
  - The updates of *V* focus on getting the class labels right given the features
- Important:
  - We assumed independence of labels given the last layer
  - But the last layer can reflect dependencies
    - If "dog" and "human" are frequently together, this should be reflected in the hidden layer
      - For example, θ_{human} might be higher when the features give a high value for θ_{dog}

Pre-Training for Multi-Label Classification

- Consider a scenario where we get a new class label
  - For example, we get new images that contain horses (not seen in training)



- Instead of training from scratch, we could:
  - Add an extra set of weights v_{k+1} to the final layer for the new class
  - Train these weights with the encoding weights W fixed
    - This is a simple/convex logistic regression problem
    - If we already have "features" that are good for many classes, we may be able to learn a new class with very few training examples!

Pre-Training for Multi-Label Classification

- Using an existing network for new problems is called "pre-training"
  - Typically, we start with a network trained on a large dataset
  - We use this network to give us features to fit a smaller dataset
    - "Few-shot learning"

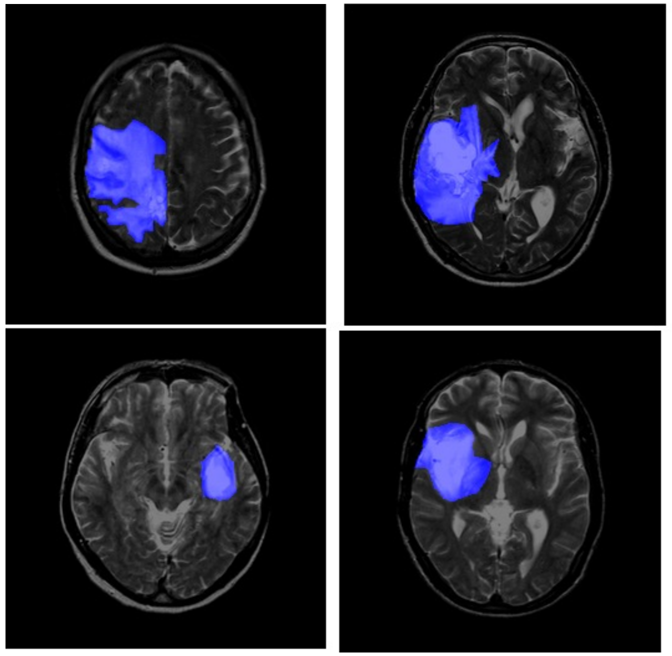
- Depending the setup, you may also update *W* and the other v_c
  - Useful if you have a lot of data on the new class
  - In this case, would typically mix in new examples with old ones

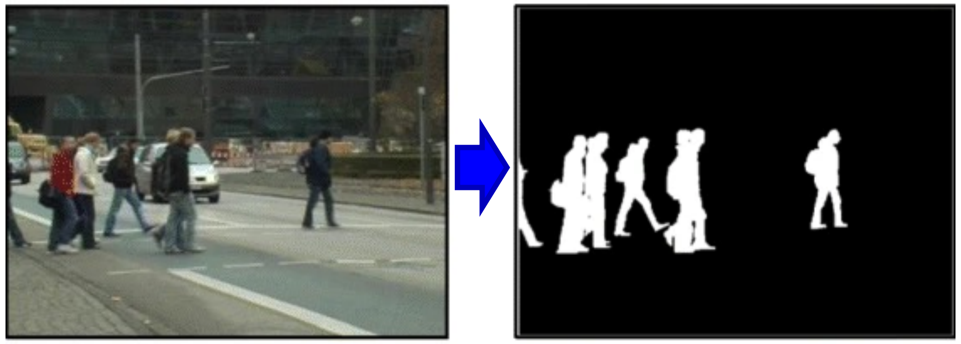
- Increasing trend in vision and language to using pre-training a lot
  - No need to learn everything about language for every language task!

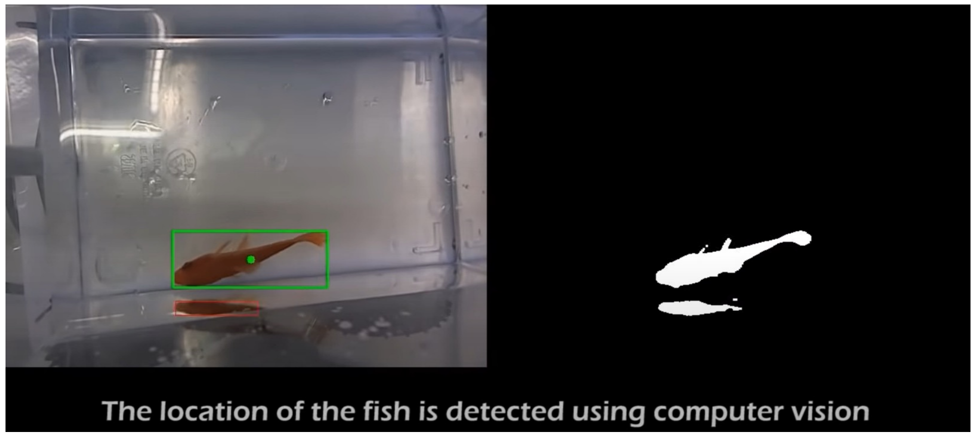
Next Topic: Fully-Connected Networks

Motivation: Pixel Classification

- Suppose we want to assign a binary label to each pixel in an image:
  - Tumour vs. non-tumour, pedestrian vs. non-pedestrian, and so on





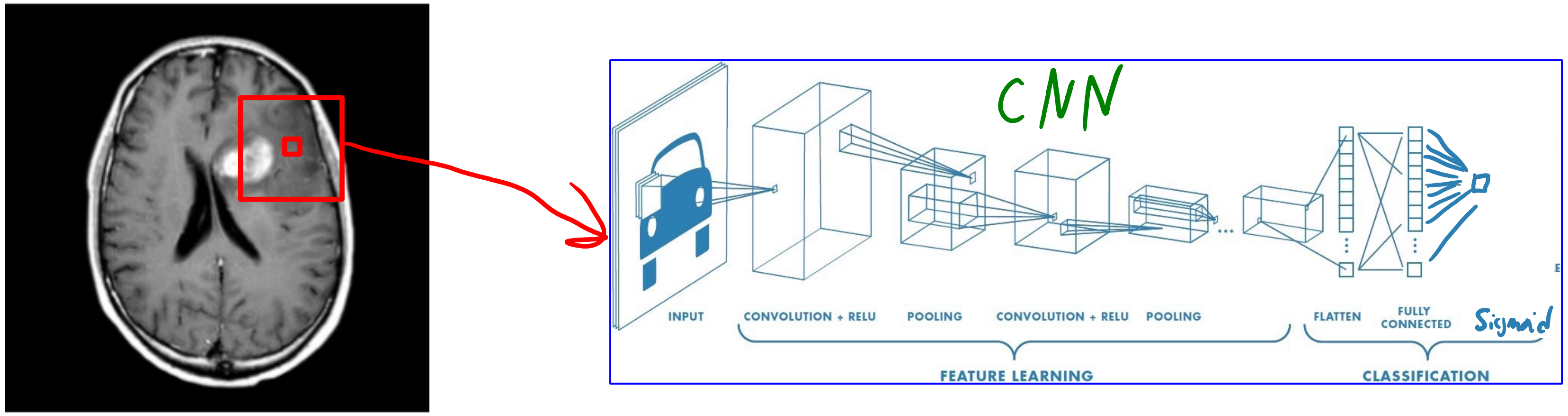


- How can we use CNNs for this problem?

https://ipsjcva.springeropen.com/articles/10.1186/s41074-019-0062-2
https://www.youtube.com/watch?v=YbNmL6hSNKw

Naïve Approach: Sliding Window Classifier

- Train a CNN that predicts pixel label given its neighbourhood

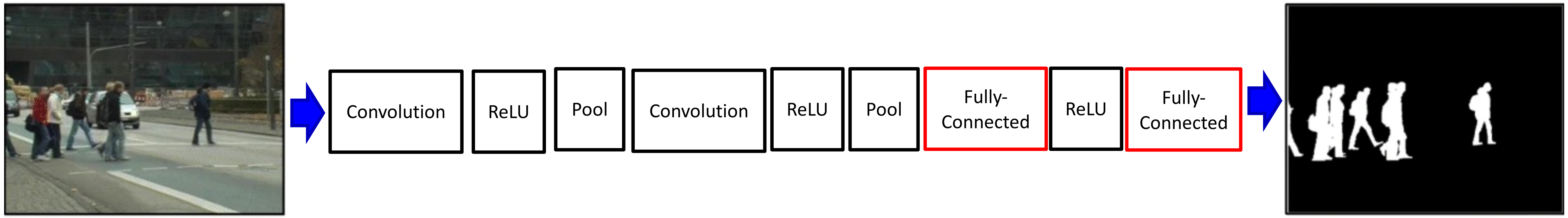


- To label all pixels, apply the CNN to each pixel
  - Advantages:
    - Turns pixel labeling into image classification
    - Can be applied to images of different sizes
  - Disadvantage: this is slow
    - (Cost of applying CNN) * (number of pixels in the image)

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Encoding-Decoding for Pixel Classification

- Similar to multi-label, could use CNN to generate an image encoding
  - With output layer making a prediction at each pixel

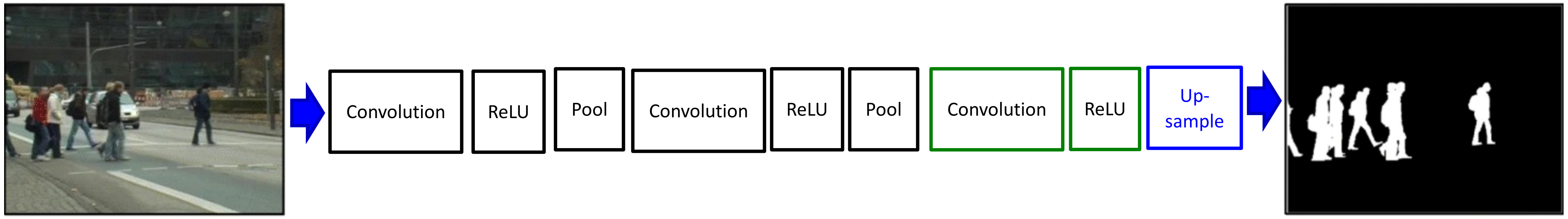


- Much-faster classification
  - Small number of “global” convolutions, instead of repeated “local” convolutions
- But, the encoding has mixed up all the space information
  - Fully-connected layers throw all that information out
  - Fully-connected layer needs to learn “how to put the image together”
    - And images must be the same size

https://ipsjcva.springeropen.com/articles/10.1186/s41074-019-0062-2

Fully-Convolutional Networks

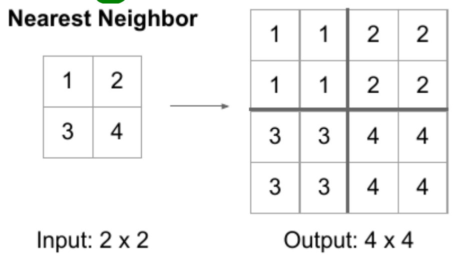
- Fully-convolutional networks (FCNs):
  - CNNs with no fully-connected layers (only convolutional and pooling)

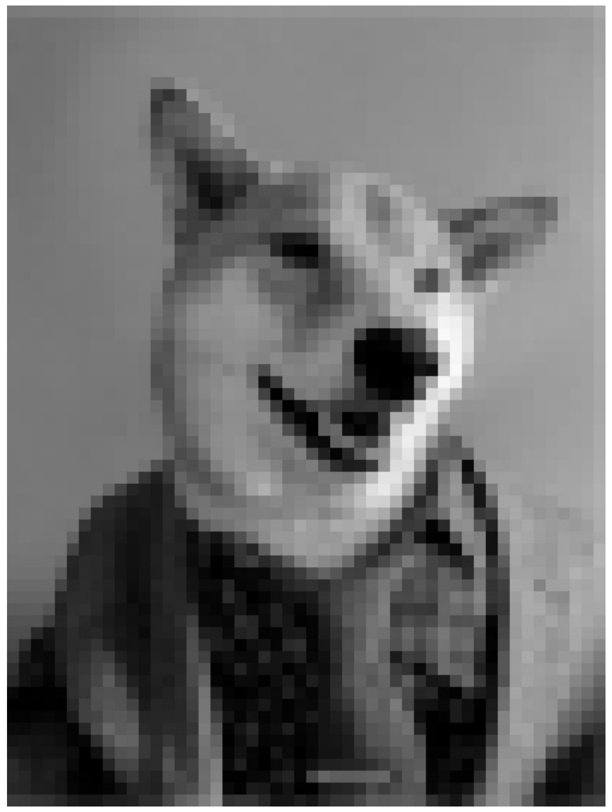


- Maintains fast classification of the encoding-decoding approach
- Same parameters used across space at all layers
  - This allows using the network on inputs of different sizes
  - Needs upsampling layer(s) to get back to image size
- FCNs quickly achieved state of the art results on many tasks

Traditional Upsampling Methods

- In upsampling, we want to go from a small image to a bigger image
  - This requires interpolation: guessing "what is in between the pixels"
- Classic upsampling operator is nearest neighbours interpolation:
  - But this creates blocky/pixelated images

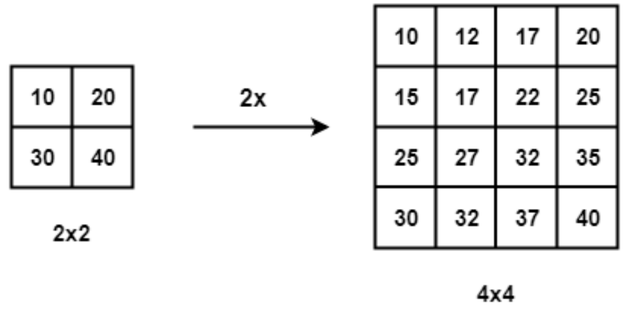




https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

Traditional Upsampling Methods

- Another classic method is bilinear interpolation:
  - Weighted combination of corners:
  - More smooth methods include “bicubic” and “splines”
- In FCNs, we learn the upsampling/interpolation operator



2x2

10	20
30	40

4x4

10	12	17	20
15	17	22	25
25	27	32	35
30	32	37	40

https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

Upsampling with Transposed Convolution

- FCN Upsampling layer is implemented with a transposed convolution
  - Sometimes called “deconvolution” in ML or “fractionally-strided convolution”
    - But not related to deconvolution in signal processing

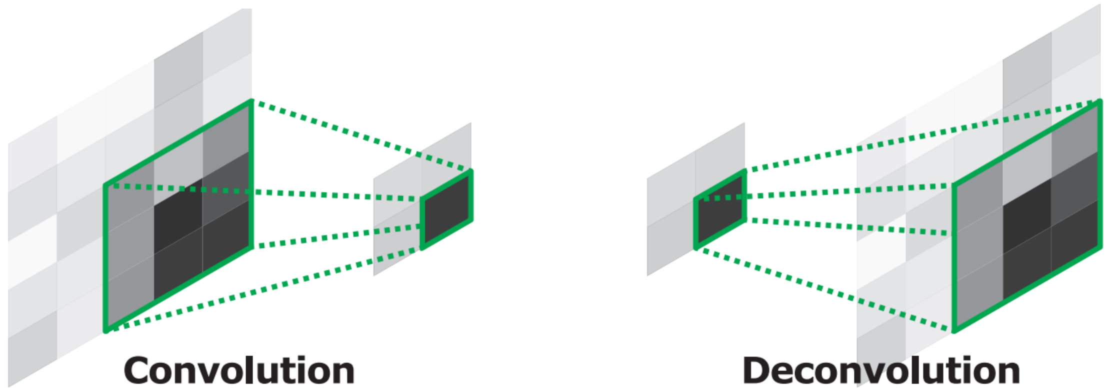
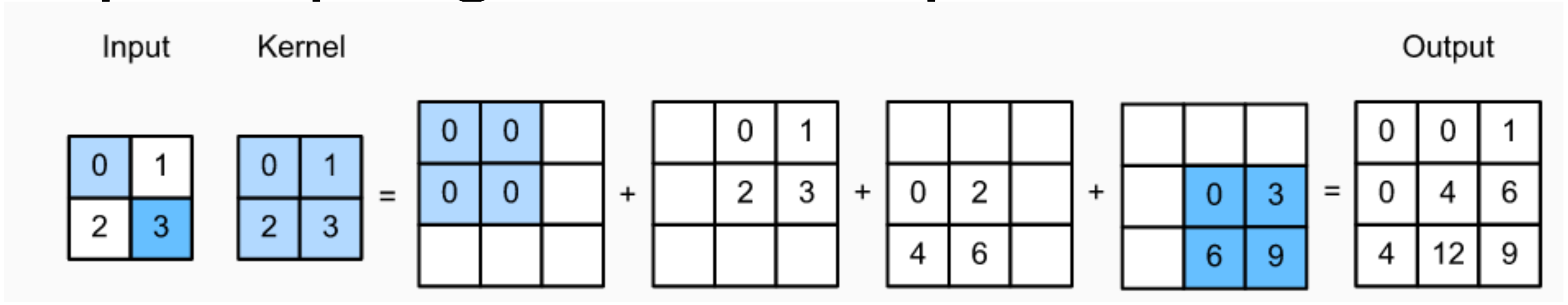
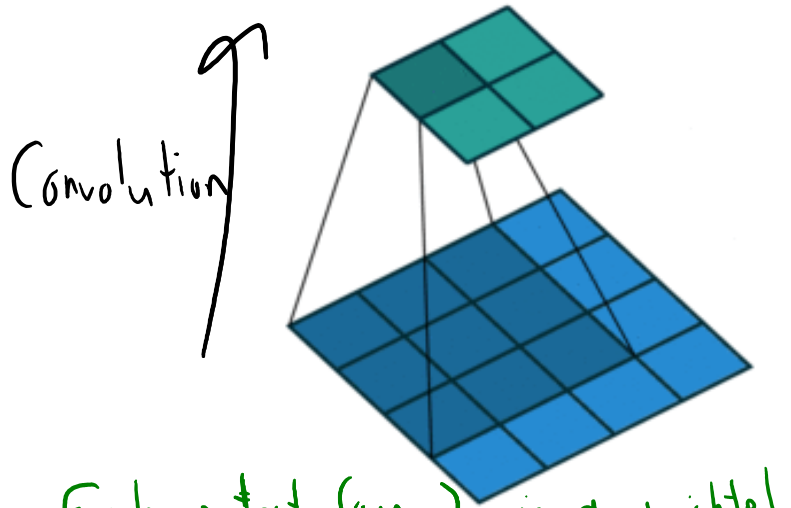


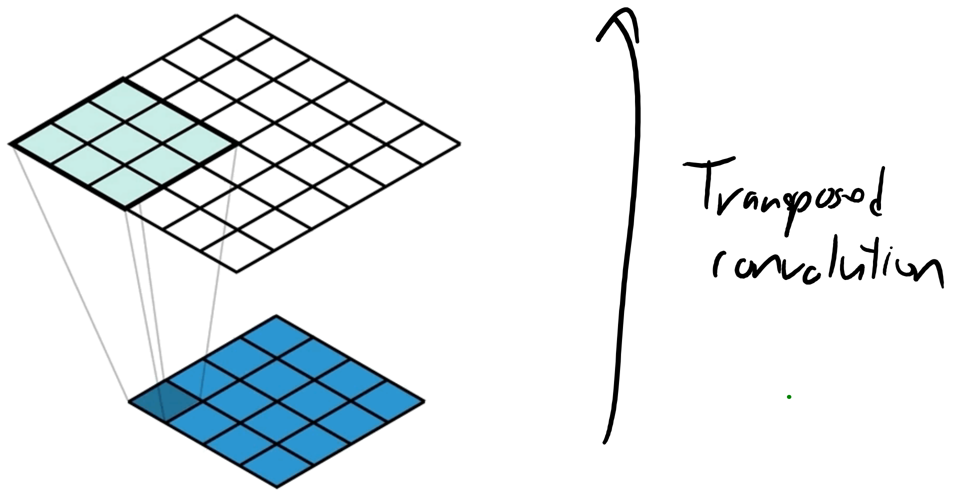
Figure 3. Illustration of deconvolution and unpooling operations.

- Convolution generates 1 pixel by taking weighted combination of several pixels
  - And we learn the weights
- Transposed convolution generates several pixels by weighting 1 pixel
  - And we learn the weights
  - This generates overlapping regions, which get added together to make final image

Upsampling with Transposed Convolution







Each output (green) is a weighted combination of one 3x3 region

Each input (blue) gives a 3x3 region. These regions overlap, so we take a weighted combination at each pixel to give final result.

- Animations here and here.

Why is it called “transposed” convolution?

- We can write the convolution operator as a matrix multiplication, z = W’ x.

$$\underbrace{\begin{bmatrix}1&2&3\\6&5&3\\1&4&1\end{bmatrix}}_{X} * \underbrace{\begin{bmatrix}1&2\\2&1\end{bmatrix}}_{W} = \underbrace{\begin{bmatrix}22&21\\22&20\end{bmatrix}}_{Z}$$

$$\overset{W}{\begin{bmatrix}1&2&0&2&1&0&0&0&0\\0&1&2&0&2&1&0&0&0\\0&0&0&1&2&0&2&1&0\\0&0&0&0&1&2&0&2&1\end{bmatrix}} \times \overset{X}{\begin{bmatrix}1\\2\\3\\6\\5\\3\\1\\4\\1\end{bmatrix}} = \overset{Z}{\begin{bmatrix}22\\21\\22\\20\end{bmatrix}}$$

- In transposed convolution, non-zero pattern of ‘W’ is transposed from convolution
  - You can implement transposed convolution as a convolution

$$\underbrace{\begin{bmatrix}1&2\\2&4\end{bmatrix}}_{X} \circledast \underbrace{\begin{bmatrix}1&2\\2&1\end{bmatrix}}_{W} = \underbrace{\begin{bmatrix}1&4&4\\4&13&10\\4&10&4\end{bmatrix}}_{Z}$$

$$\overset{W^T}{\begin{bmatrix}1&0&0&0\\2&1&0&0\\0&2&0&0\\2&0&1&0\\1&2&2&1\\0&1&0&2\\0&0&2&0\\0&0&1&2\\0&0&0&1\end{bmatrix}} \times \overset{X}{\begin{bmatrix}1\\2\\2\\4\end{bmatrix}} = \overset{Z}{\begin{bmatrix}1\\4\\4\\4\\13\\10\\4\\10\\4\end{bmatrix}}$$

- In this example the filter is the same, but does not need to be:
  - Transposed convolution is not the “reverse” of convolution (it only “reverses” the size)

https://www.machinecurve.com/index.php/2019/09/29/understanding-transposed-convolutions/

bonus!

Increasing Resolution: FCN Skip Connections

- Convolutions and pooling lose a lot of information
- Original FCN paper considered adding skip connections to help upsampling:

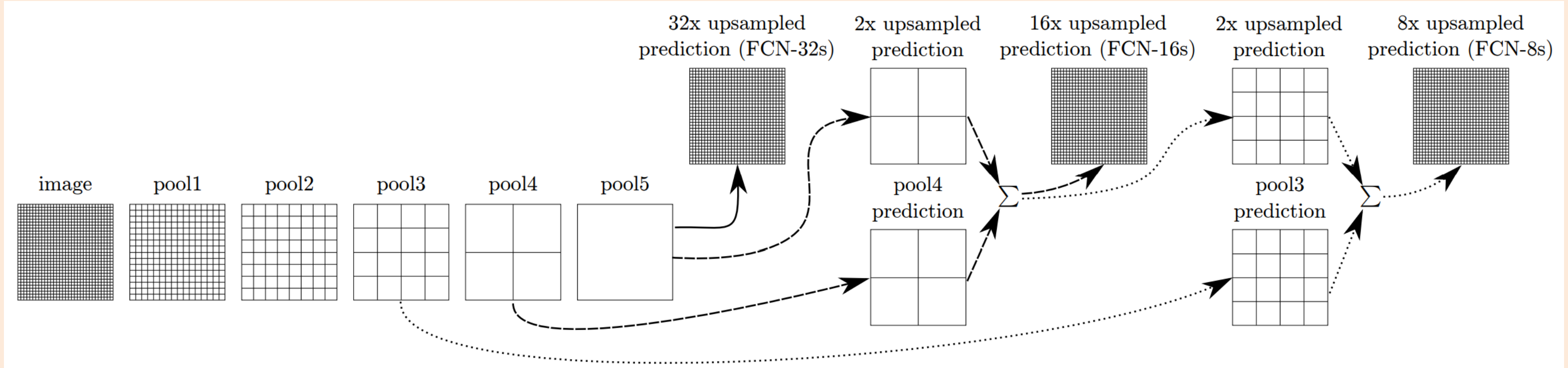
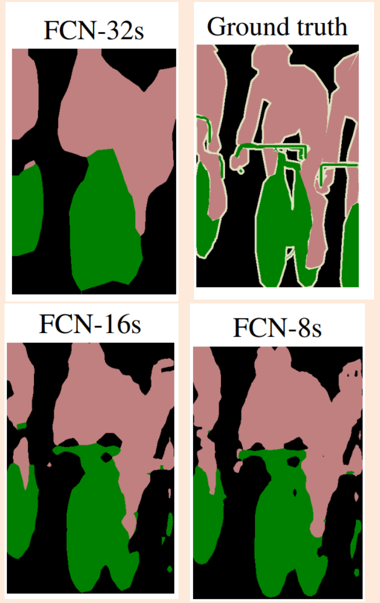


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Layers are shown as grids that reveal relative spatial coarseness. Only pooling and prediction layers are shown; intermediate convolution layers (including our converted fully connected layers) are omitted. Solid line (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Dashed line (FCN-16s): Combining predictions from both the final layer and the `pool4` layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Dotted line (FCN-8s): Additional predictions from `pool3`, at stride 8, provide further precision.



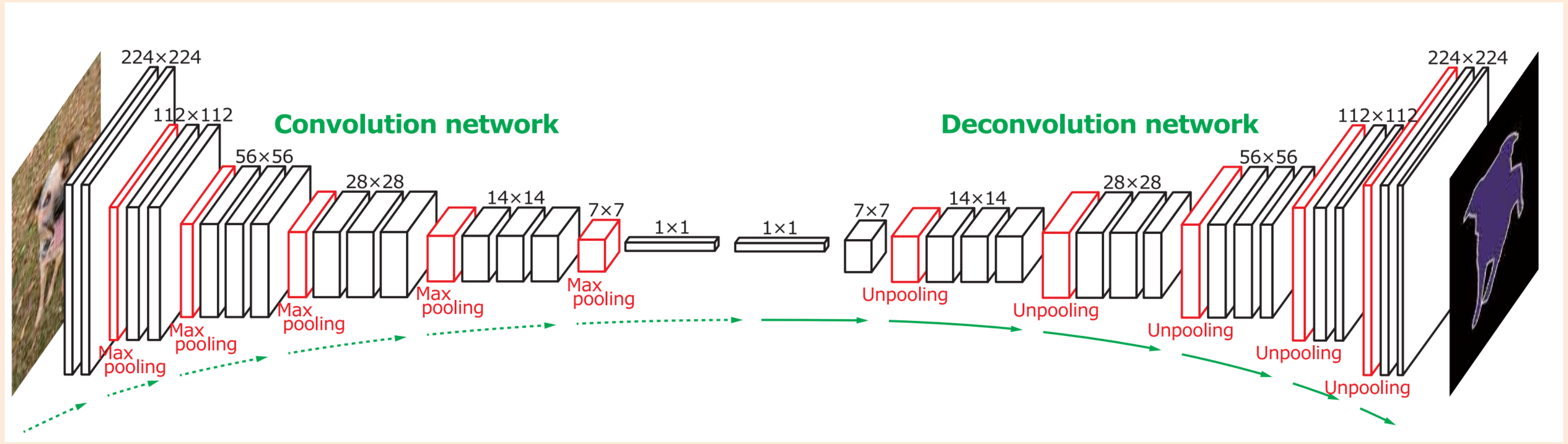
- Allows using high-resolution information from earlier layers
- They first trained the low-resolution FCN-32, then FCN-16, then FCN-8
  - "First learn to encode at a low resolution", then slowly increase resolution
  - Parameters of transposed convolutions initialized to simulate "bilinear interpolation"

https://arxiv.org/pdf/1411.4038.pdf

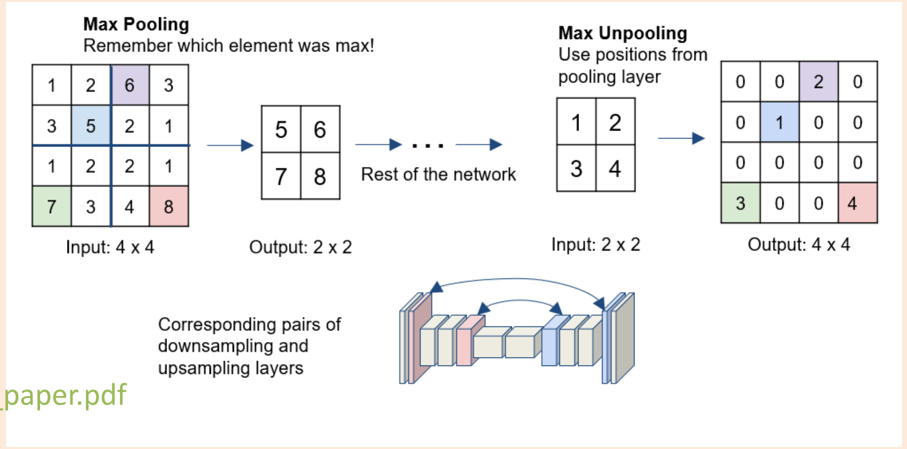
bonus!

Increasing Resolution: Deconvolution Networks

- Alternate resolution-increasing method is deconvolution networks:



- Includes transposed convolution layers and unpooling layers
  - Store the max pooling argmax values
  - Restores "where" activation happened
    - Still loses the "non-argmax" information

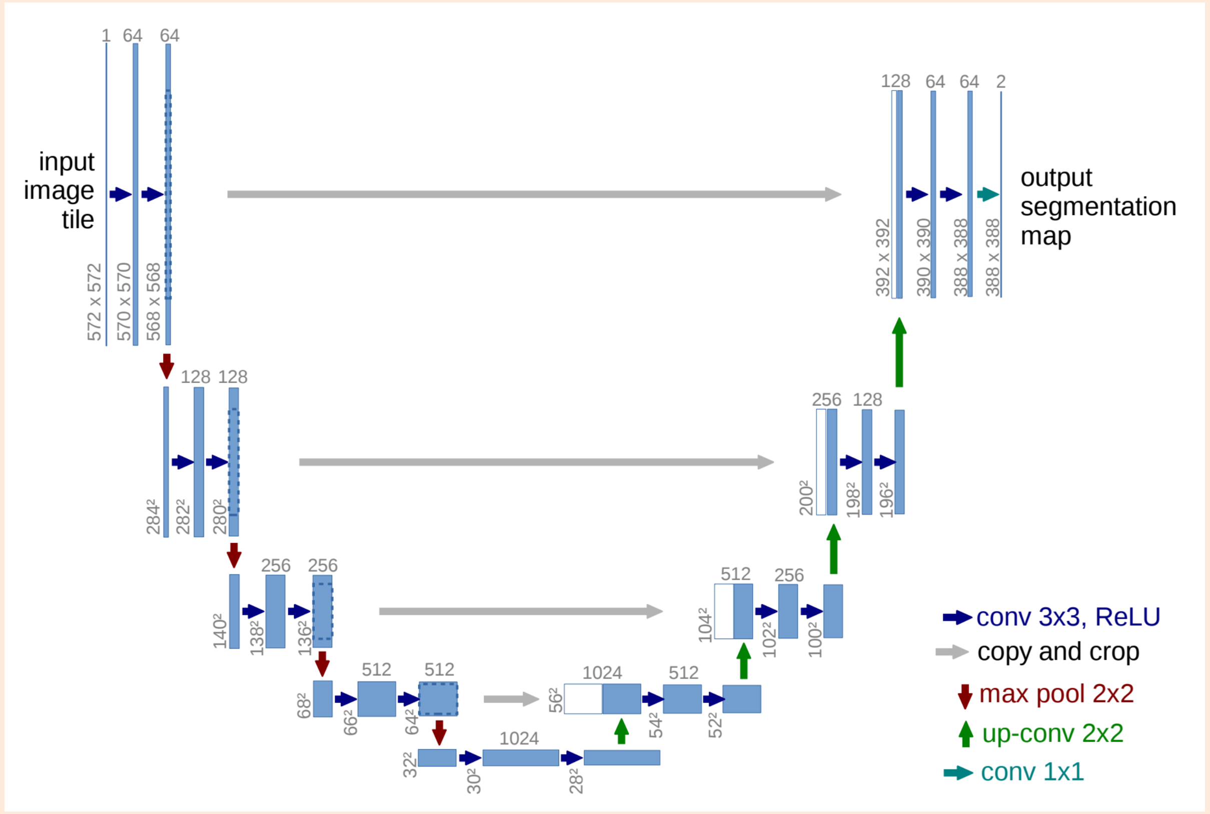


https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Noh_Learning_Deconvolution_Network_ICCV_2015_paper.pdf
https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

bonus!

Increasing Resolution: U-Nets

- Another popular variant is u-nets:



- Decoding has connections to same-resolution encoding step

https://arxiv.org/pdf/1505.04597.pdf

Source of Labels

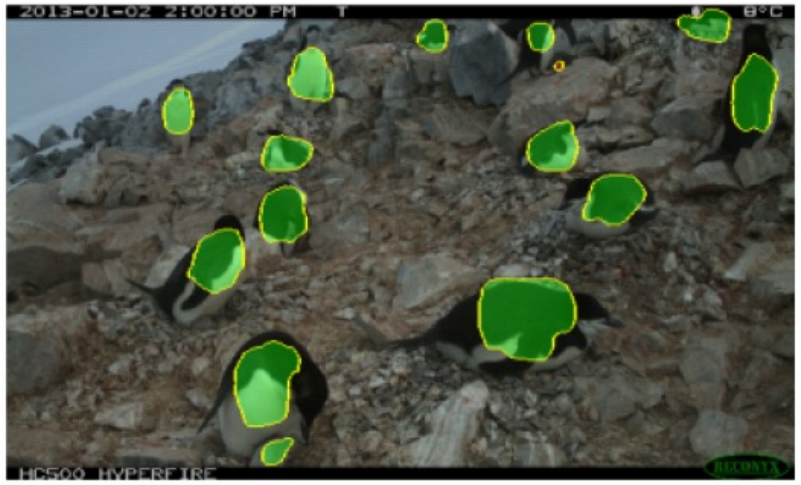
- Labeling every pixel takes a long time.
- Where we get the labels from?
  - Domain expert (medical images)
  - Grad students or paid labelers (ImageNet)
  - Simulated environments
    - High number of lower-quality examples
    - Often a net gain with fine-tuning on real images
    - Can get data at night, in fog, or dangerous situations

Video game

Google street view

Source of Labels

- Recent works recognize you do not need to label every pixel
  - You can evaluate loss/gradient on a subset of labeled pixels
  - Could have labeler click on a few pixels inside objects, and a few outside
    - Many variations are possible, that let you label a lot of images in a short time
- Penguin counting based "single pixel" labels in training data:
  - And some tricks to separate objects and remove false positives:



https://arxiv.org/pdf/1807.09856.pdf

End of Part 1 (“Binary Variables”): Key Concepts

- We discussed binary density estimation
  - Model the proportion of times a binary event happens
- We discussed the Bernoulli parameterization
- We discussed various inference tasks, given the parameter:
  - Compute probabilities, find decoding, generate samples
- We discuss different learning strategies, given data:
  - Maximum likelihood estimation (MLE), maximum a posteriori (MAP)
  - Beta distribution as a prior gives a beta distribution as posterior (“$\propto$”)
- We discussed modeling binary variables conditioned on features:
  - Tabular parameterization is flexible but has too many parameters
  - Logistic regression is limited but has a linear number of parameters

End of Part 1 (“Binary Variables”): Key Concepts

- We discussed multivariate binary density estimation
  - Refined inference tasks when we have more than one random variable:
    - Joint probability, marginal probability, and conditional probability
  - Product of Bernoullis assumes variables are independent
    - Fast inference/learning but a strong assumption
- We discussed generative classifiers:
  - Build a model of the joint probability of features and labels
    - Compared to usual discriminative classifiers that model labels given features
  - Naïve Bayes assumes features are independent given label
- We discussed neural networks:
  - Model that learns the features and classifier simultaneously
  - Alternate between linear and non-linear transformations (universal approximator)
  - Training is a non-convex problem, but SGD often works better than expected:
    - For large-enough networks we often find global, and SGD seems to have implicit regularization

End of Part 1 (“Binary Variables”): Key Concepts

- We discussed deep learning with multiple hidden layers
  - Biological motivations and efficient representation of some functions
  - Vanishing gradient problem and modern solutions:
    - ReLU, skip connections, ResNets
- We discussed automatic differentiation to generate gradient code
  - Code that generates gradient code for you (using chain rule)
- We discussed convolutional neural networks (CNNs):
  - Include convolution layers that measure image features
  - Include max pooling layers that highlight top features across space
  - Reduces number of parameters and gives some spatial invariance

End of Part 1 (“Binary Variables”): Key Concepts

- We discussed autoencoders:
  - Networks where the output is the input
  - Encodes input into a bottleneck layer, then decodes back to input
  - Non-linear dimensionality reduction
  - Denoising autoencoders learn to enhance images
- We discussed multi-label classification:
  - Where each training examples can have 0-k correct labels
  - We discussed an encoding approach where the classes shares hidden layers
    - Reduces parameters and captures dependencies between labels
  - We discussed pre-training to learn new tasks with fewer labeled examples
- We discussed pixel labeling:
  - Fully-convolutional networks maintain spatial information at all layers
    - Requires upsampling to original image size
    - Can label images of different sizes