
Deep Learning and Applications

Today's Outline

- Visualizing CNNs
- Transfer Learning
- Neural Net Training Tricks
 - Data Augmentation
 - Weight Initialization/Batch Normalization/Dropout

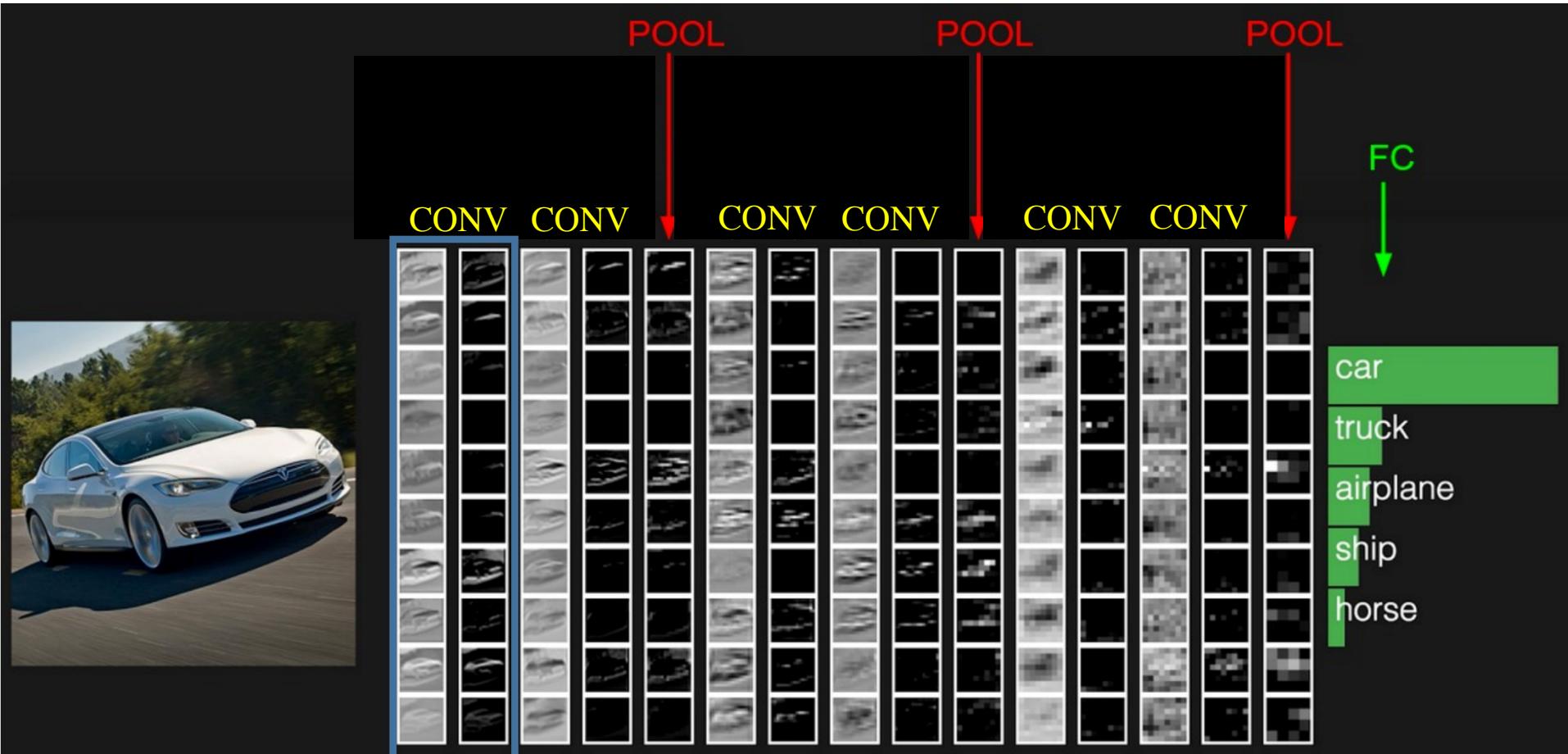
Quick Review: Convolutional Neural Networks

Recap of CNN Architecture

- Typically a CONV is followed by a POOL
- Closer to the output, use FC layers
- In CONV, smaller filters are preferred (say $3 * 3 * z$)
- Input image should ideally be divisible by 2 many times



Example: A CNN Architecture



- A different sequence of layers
- Number of filters (layer **depth**) is 10
- Activation tensors (flattened along **depth**) are shown

¹Figure: <http://cs231n.github.io/convolutional-networks/>

Example: CONV Layer Parameter Count

- Input tensor of size $90 * 90 * 10$
 - Say we have 5 filters, each is $3 * 3 * 10$
 - Stride is 1 and zero padding is 1
 - Then output tensor will be $90 * 90 * 5$
 - We can calculate manually for other strides and padding values
-
- Number of parameters is $5 * (3 * 3 * 10 + 1) = 455$
 - Contrast with Fully connected net:
 - Number of inputs is 81000
 - Number of hidden layer neurons is 40500
 - Hence, the number of parameters is $> 3,280,500,000$

CNN and Backpropagation

- Backpropagation through a CONV layer
 - Constitutes a set of matrix-matrix products and whatever is the behavior for the nonlinearity
- Backpropagation through a POOL layer
 - Essentially like ReLU where one can keep track of the index of the maximum
- (You will not have to do this by hand in real-life)

Questions?

Visualizing CNNs

Combating Non-Interpretability

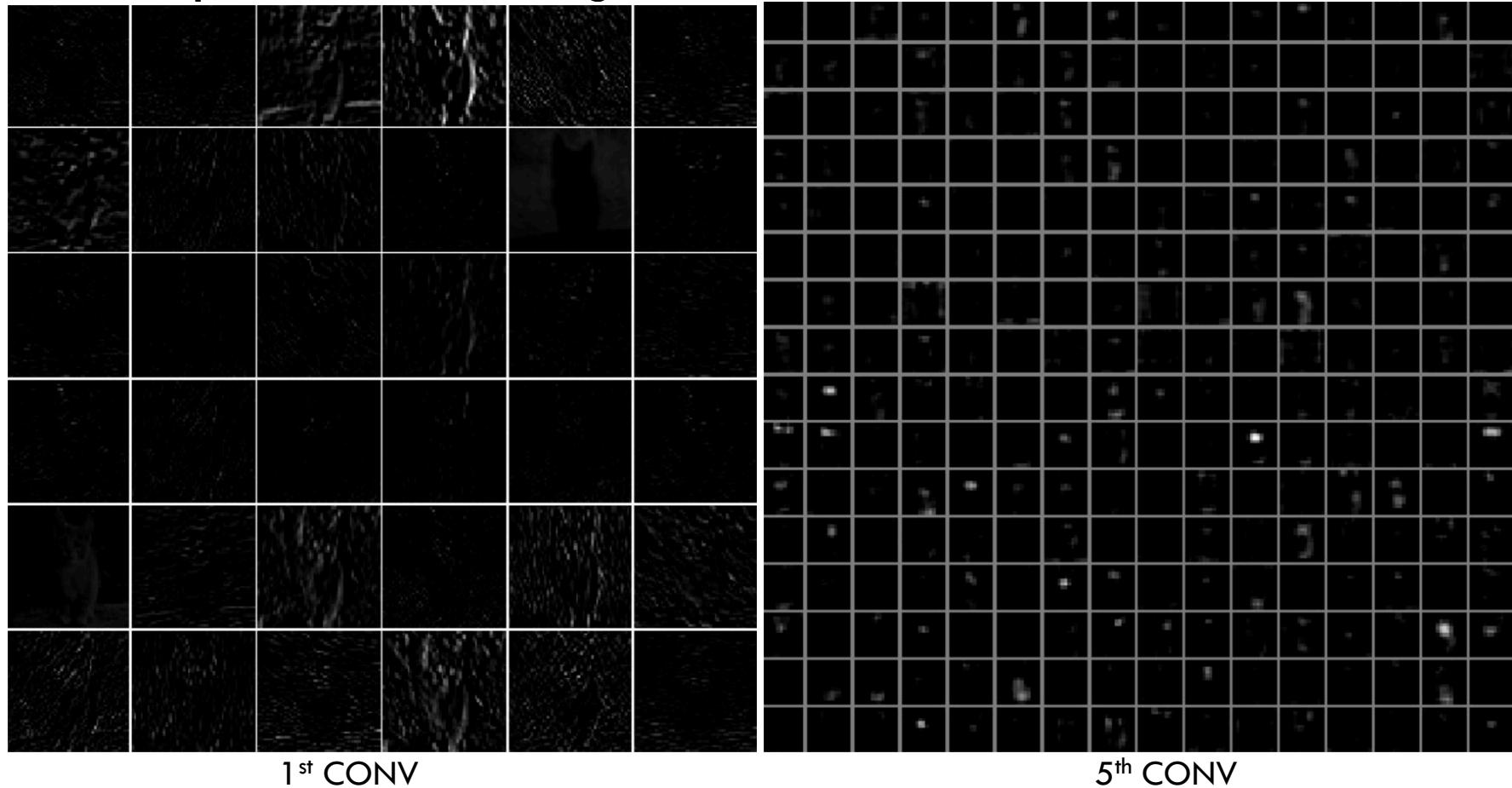
- Common criticism: learned features are not interpretable
- We will look at a few attempts
 - Look at activations
 - Look at weights
 - Look at images in an embedded space
 - Look at impact of occlusion
 - Look at images that activate neurons highly

An Example CNN Visualization Tool

- Online tool by Adam Harley
 - <http://scs.ryerson.ca/~aharley/vis/conv/flat.html>

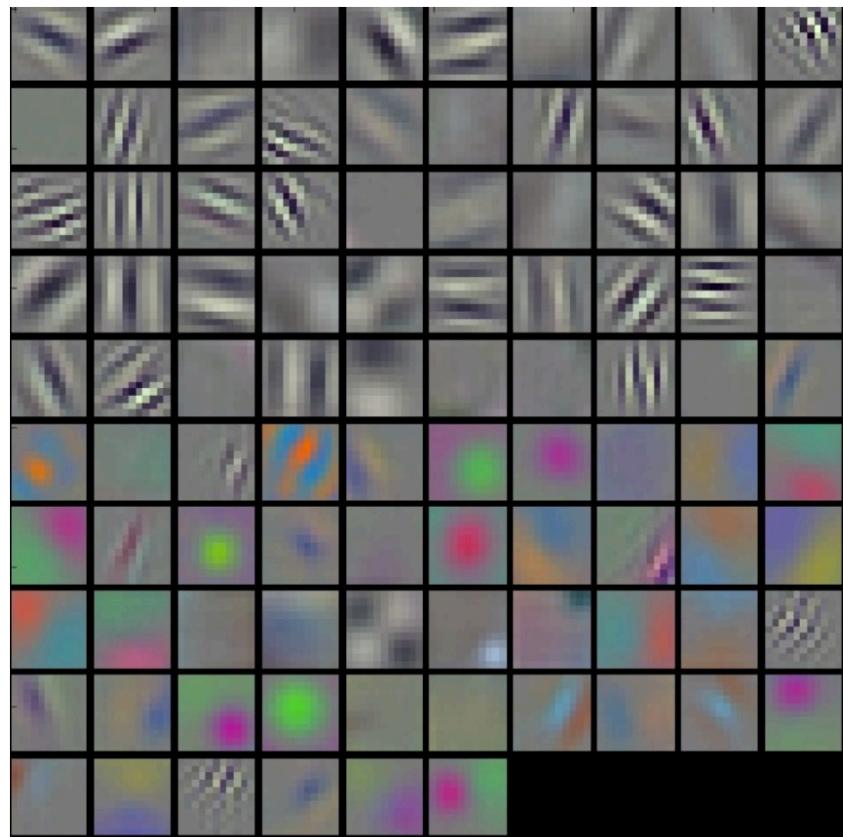
Visualize: Activations

- Useful to debug ‘dead’ filters (e.g., when using ReLU)
 - Input is a cat image

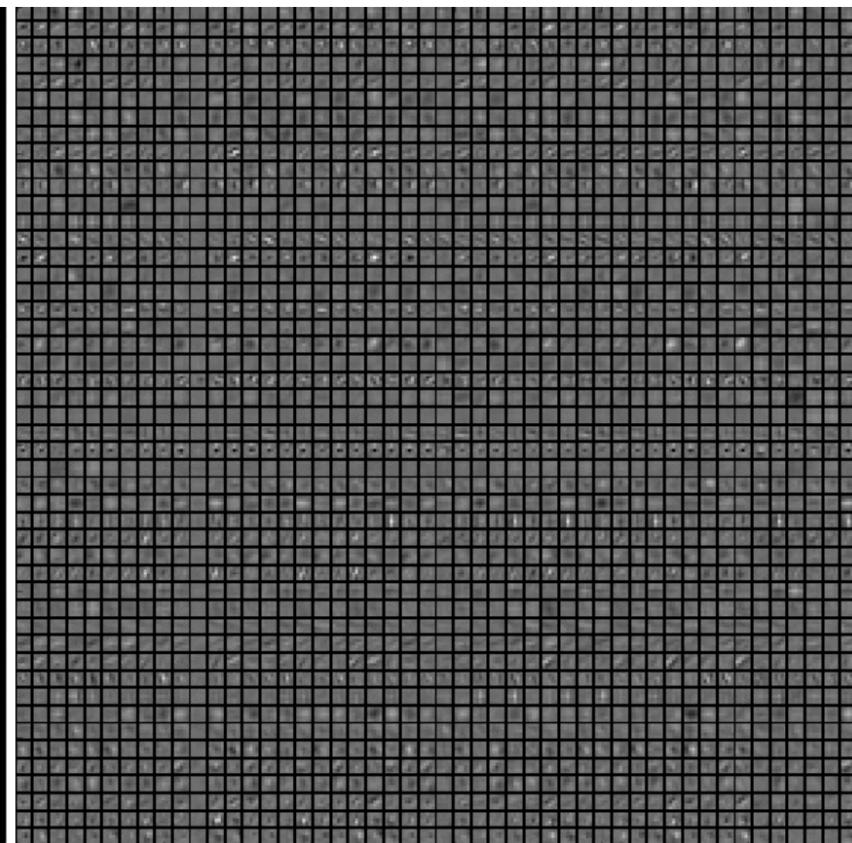


Visualize: Weights

- Useful to debug if training needs to be run more (if patterns are noisy)



1st CONV



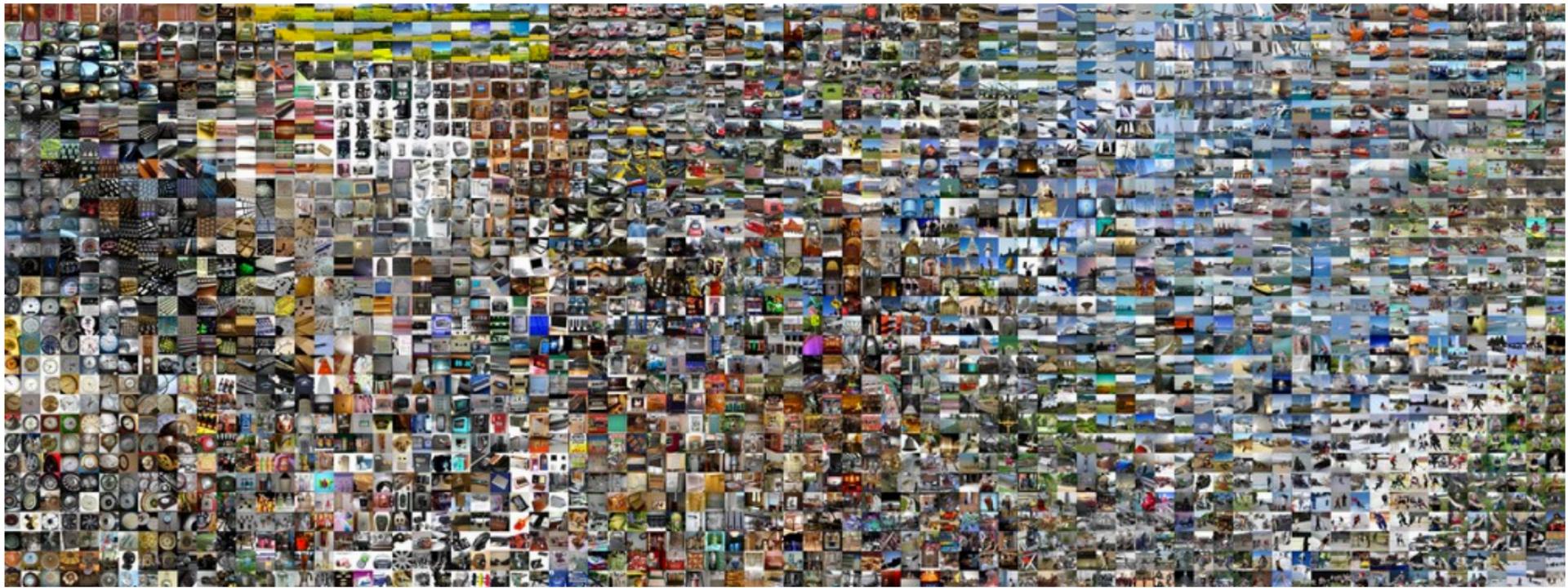
2nd CONV

Visualize: Low-Dimensional Embeddings

- CNN
 - Input: Image
 - Output: Scores
- The input to the layer that computes scores:
 - $s = W \max(0, h) + b = Wa + b$
- Activation a can be considered as a representation of the input image
- Embed a 's into a 2D space
 - Such that distance properties are preserved

Visualize: Low-Dimensional Embeddings

- In Alexnet, the output of layer before FC layer is 4096 dim
- The t-SNE embedding is shown below:



- Similarities are class-based and semantic rather than color and pixel based
 - Implies: images close to each other are similar for the CNN¹⁵

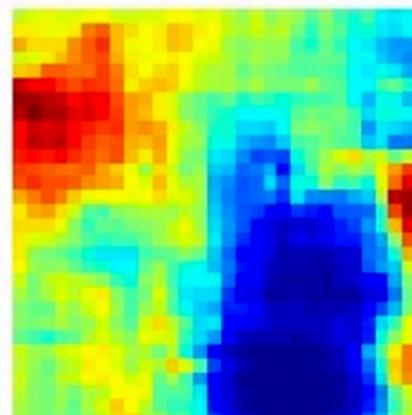
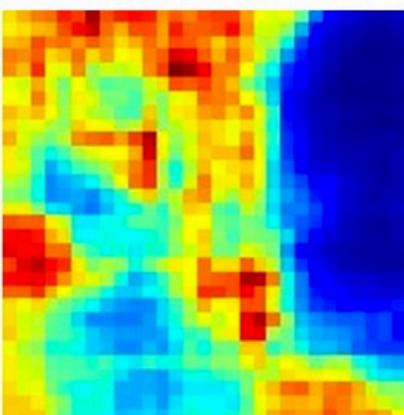
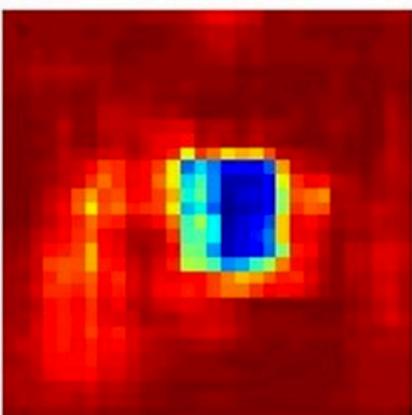
¹⁵Figure: <http://cs231n.github.io/understanding-cnn/>

Visualize: By Occlusion

- To figure out which part of the image is leading to a certain classification
- Plot the probability of class of interest as a function of occlusion

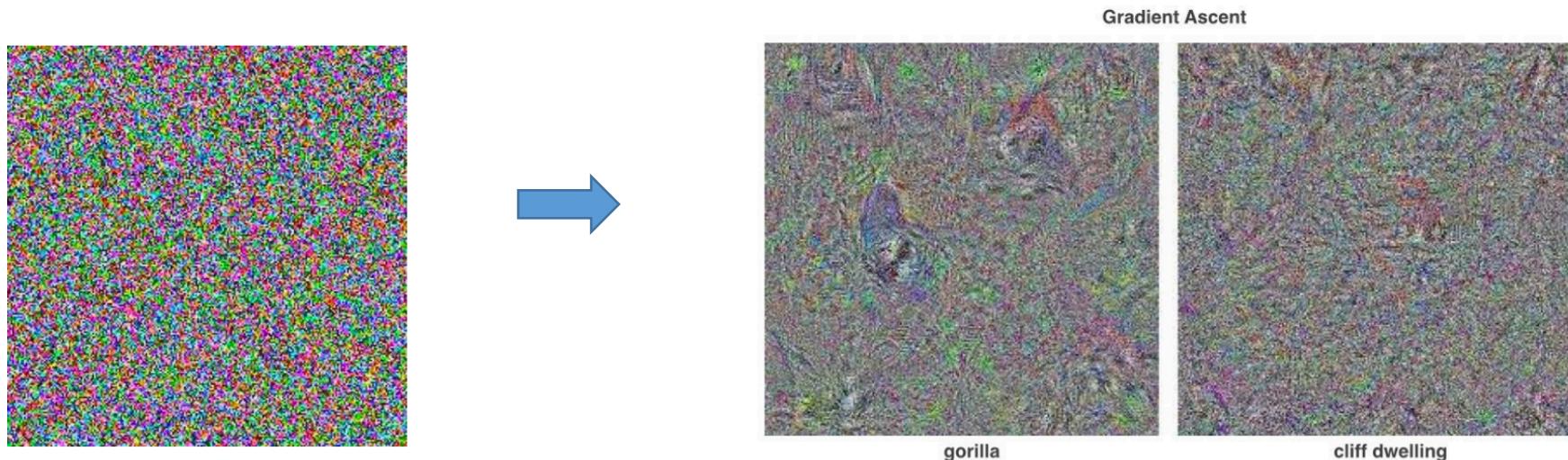
Visualize: By Occlusion

- Occlusion in grey is slid over the images and plot probability of correct class



Visualize: Synthesize Images

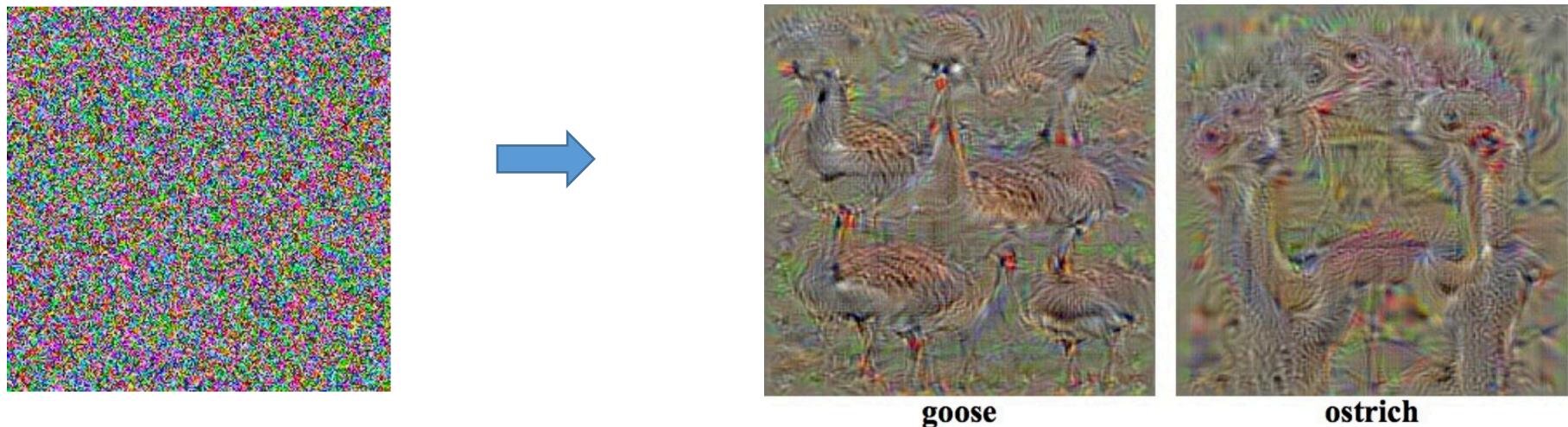
- Find images that activate a neuron the most



- Seed with ‘natural’ image priors

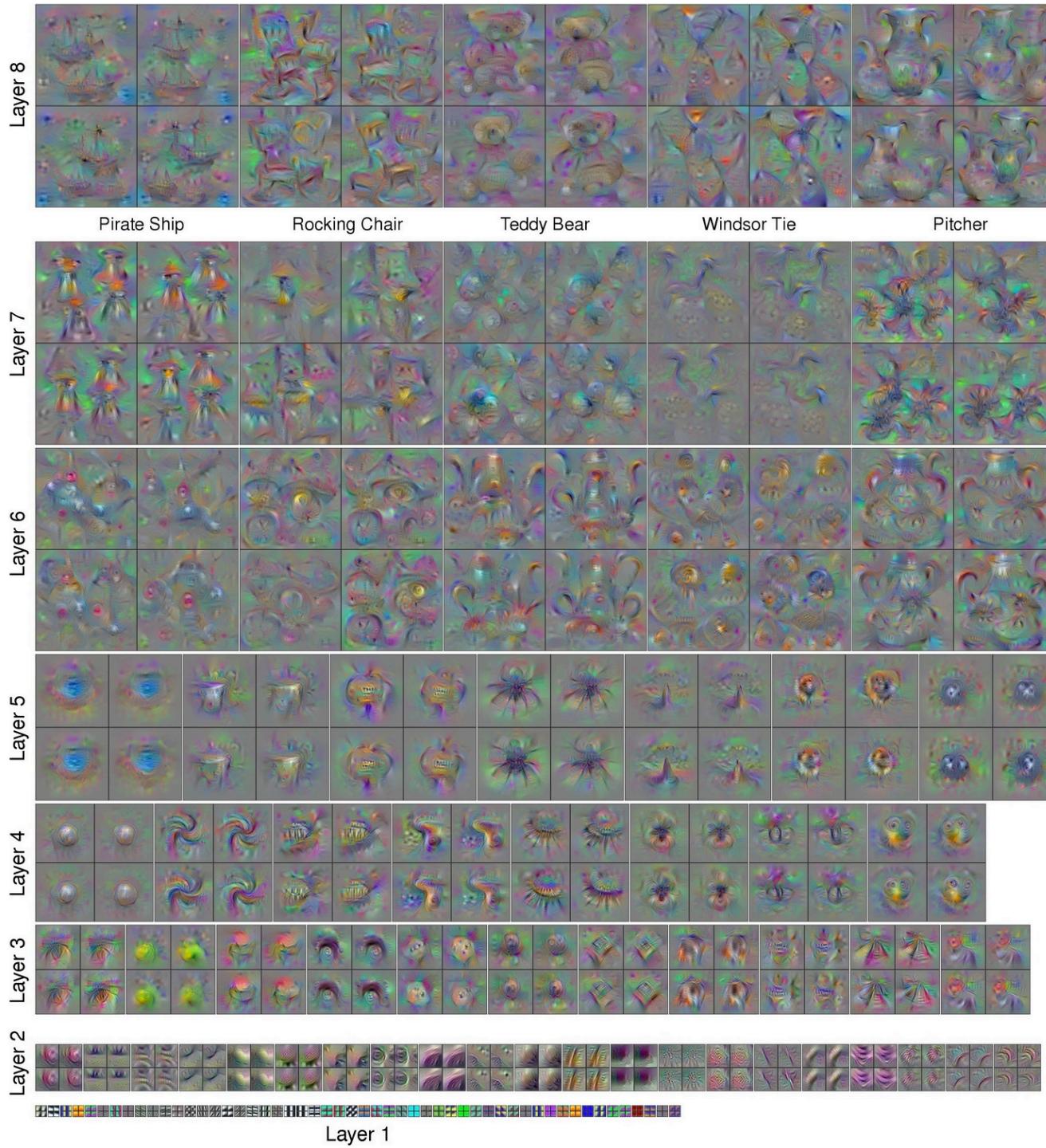
Visualize: Synthesize Images

- Find images that activate a neuron the most



- Seed with ‘natural’ image priors

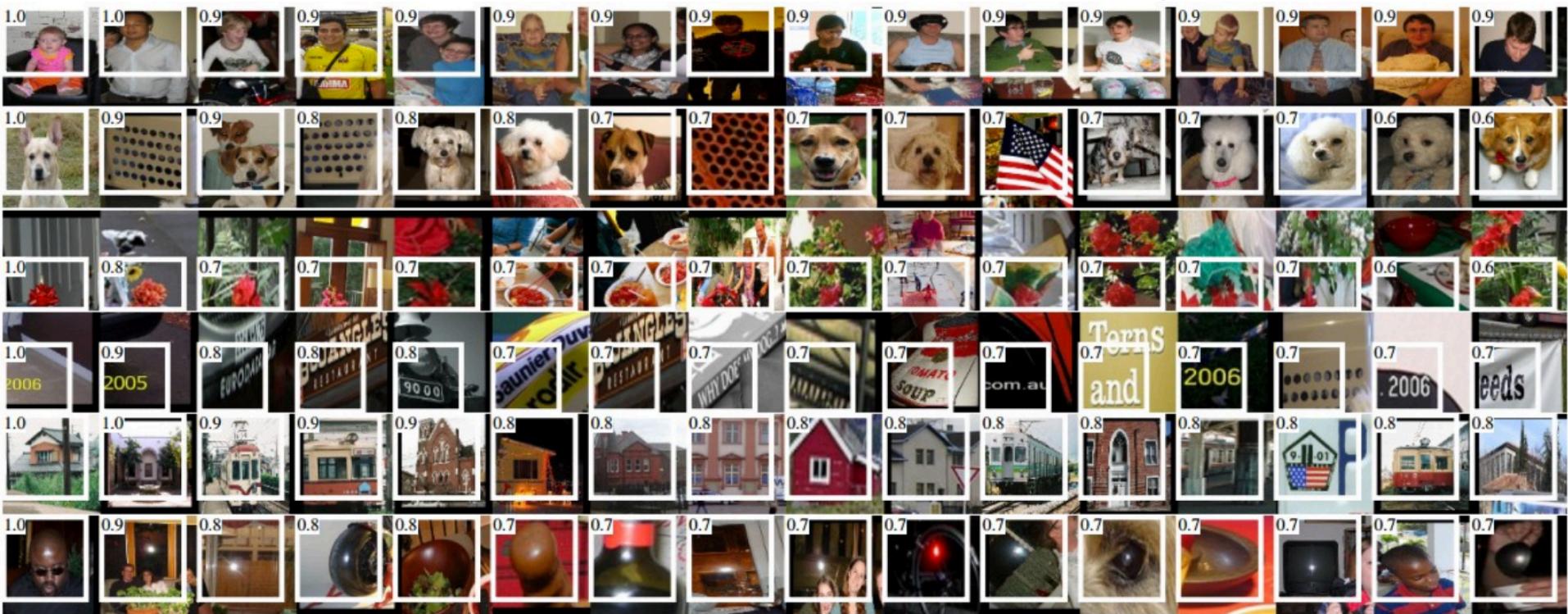
Visualize: Synthesize images



¹Figure: <http://yosinski.com/deep>

Visualize: Images that Activate a Neuron

- Track which images maximally activate a neuron
 - Understand what the neuron is tracking



5th POOL

Activation values and receptive fields of some neurons in Alexnet
(May not be a good idea...)

Questions?

Today's Outline

- Visualizing CNNs
- Transfer Learning
- Neural Net Training Tricks
 - Data Augmentation
 - Weight Initialization/Batch Normalization/Dropout

Transfer Learning

Transfer Learning

- Very few people train a deep feedforward net or a CNN from scratch
- **Myth:** “We need a lot of data to use Deep Neural Networks”
- We will see two approaches if we have small data
 - Feature extraction
 - Fine-tuning
- Both these are loosely termed as **Transfer learning**

Transfer by Feature Extraction (I)

- Get a pretrained CNN
 - Example: VGG or AlexNet that was trained on Imagenet
- Remove the last FC (that outputs 1000 dim score)
- Pass new training data to get **embeddings**

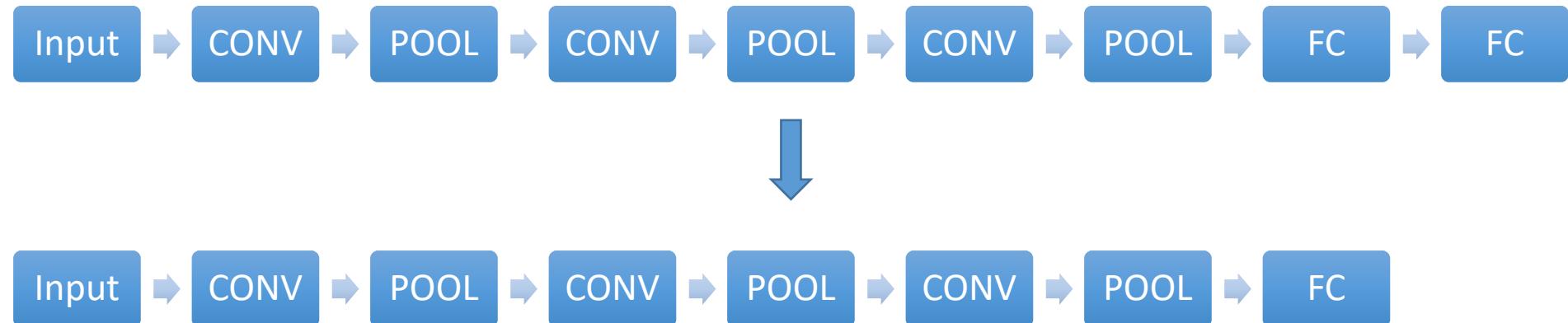
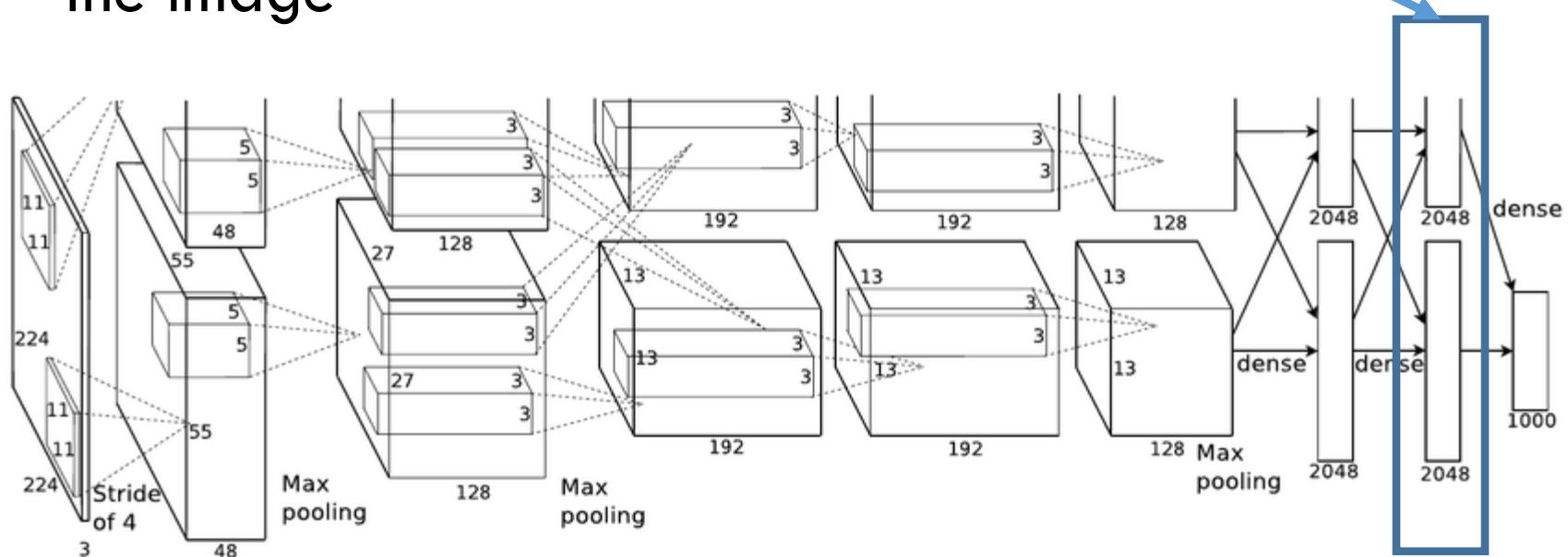


Image Embeddings

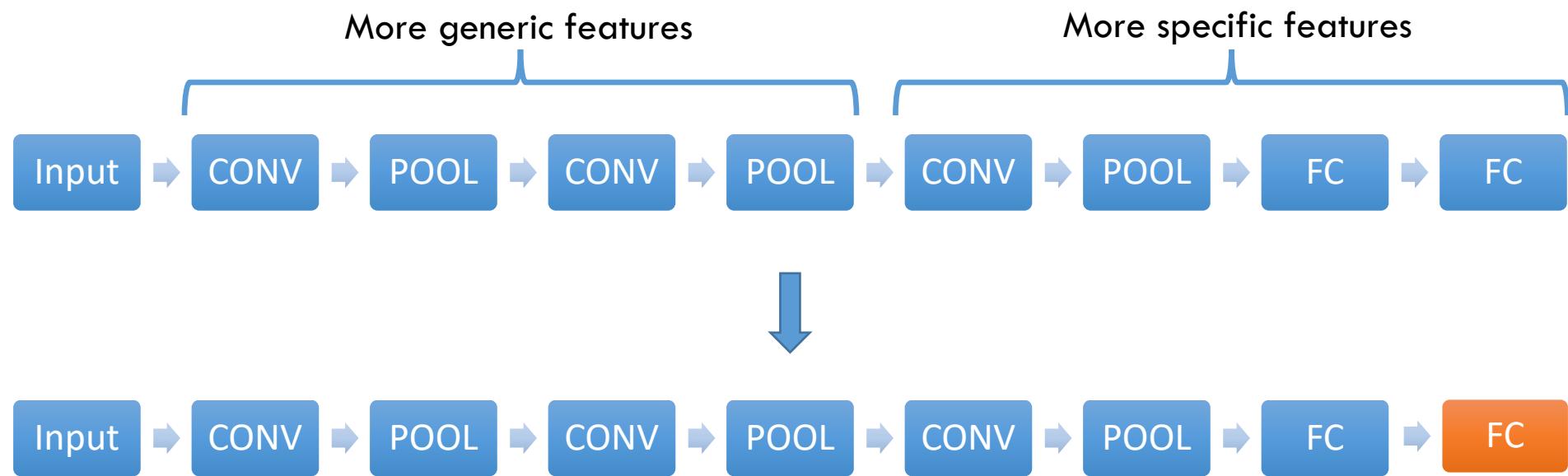
- We can think of the penultimate hidden layer activations (a 4096 dim vector) as an **embedding** of the image



- This is the **activation vector** or the **representation** or the **CNN code** of the image

Transfer by Feature Extraction (II)

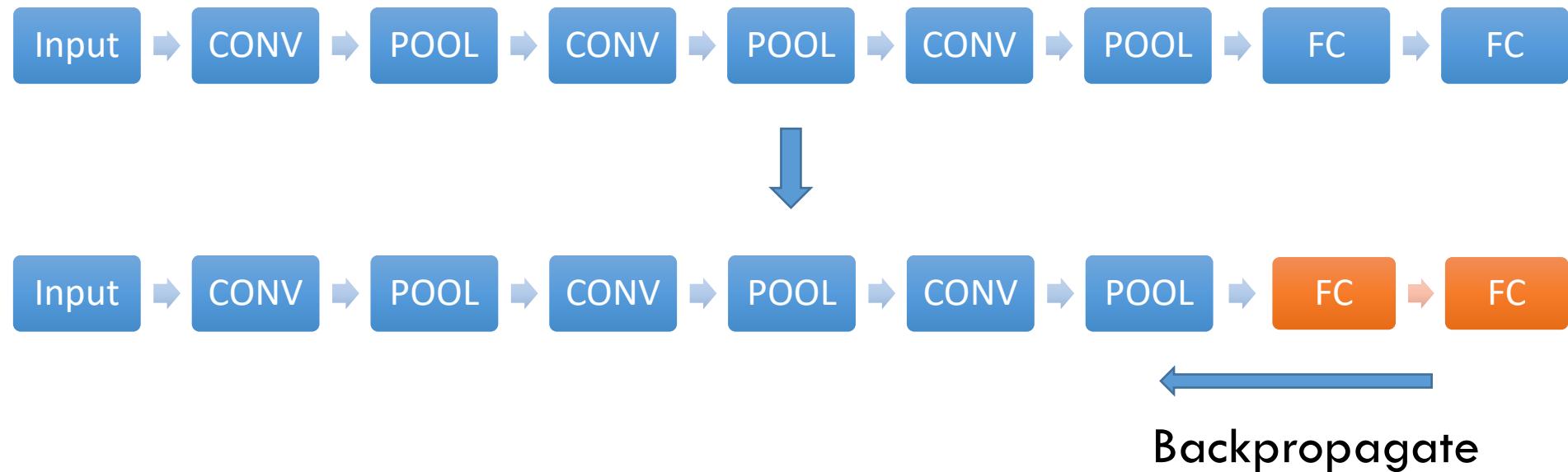
- Input these to a linear or non-linear classifier!



- For example, for imagenet output 1000 dim scores
- For our data, output say 2 scores (cat vs dog)

Transfer by Fine-tuning

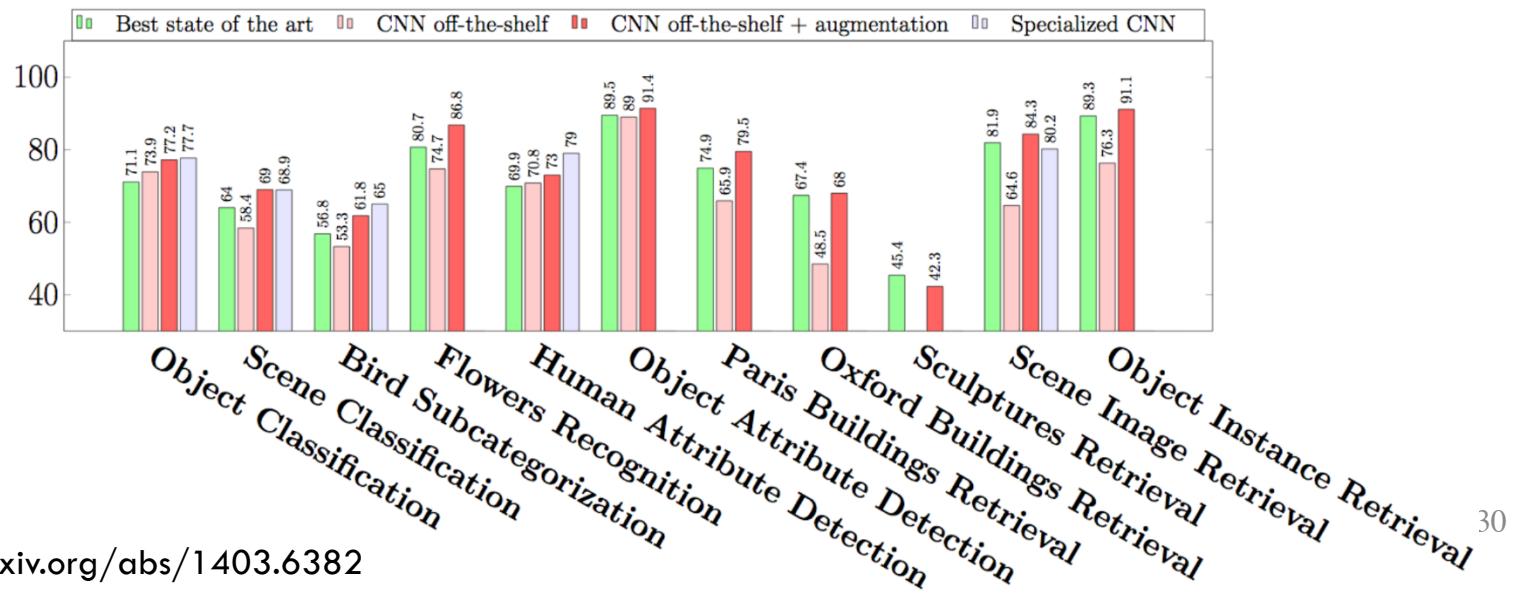
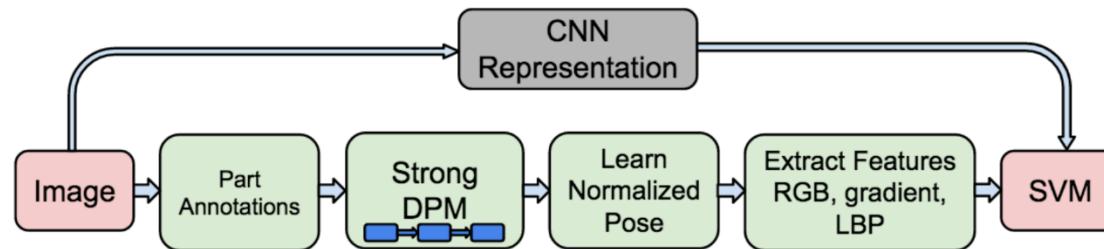
- Retrain or **finetune** additional layers of the pre-trained if we have more data



- We can even go all the way back to the first layer if there is a lot of training data available

Benefits of Transfer

- We can get a significant boost in performance compared to hand engineered classification/machine-learning pipelines



Aside: Other Vision Tasks

- Some example vision tasks are given below

Classification



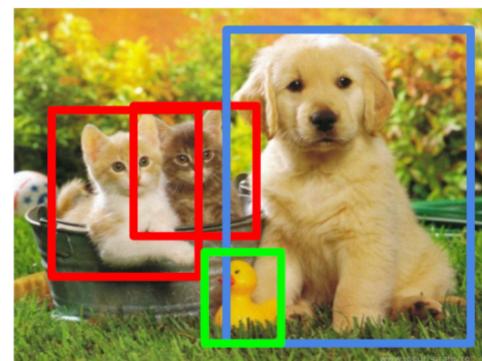
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation



CAT, DOG, DUCK

Single object

Multiple objects

Transfer Learning Choices

- When to transfer

	Similar dataset	Different dataset
Small data	Feature extract	NA
Large data	Fine-tune a bit	Fine-tune a lot

- How to transfer
 - Get pre-trained models for popular software systems

This is key for projects!

VGG Net Example

- 2nd in the 2014 ILSVRC classification task
- 3x3 conv filters with stride 1
- ReLU non-linearity
- 5 POOL layers
- 3 FC layers



Questions?

Today's Outline

- Visualizing CNNs
- Transfer Learning
- Neural Net Training Tricks
 - Data Augmentation
 - Weight Initialization/Batch Normalization/Dropout

Neural Net Training Tricks

Neural Nets in Practice

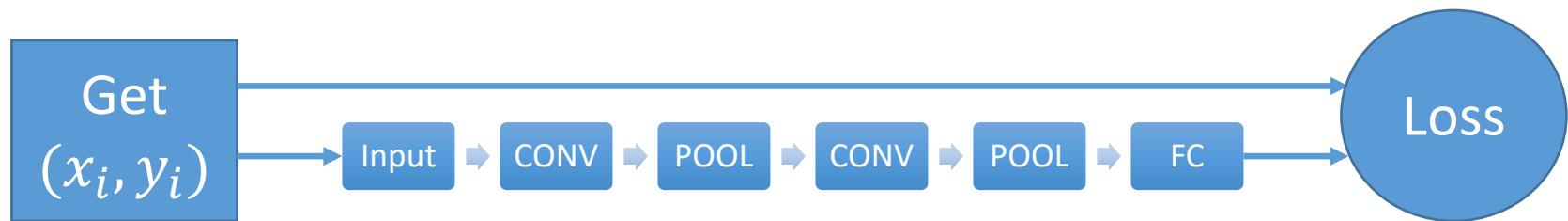
- There are a few **empirically validated** techniques that improve the performance (classification accuracy) of feedforward nets and CNNs
- We will look at some of these
 - Data: data augmentation
 - Model: initialization, batch normalization, dropout
- For our discussion, we will fix the optimization technique to be a gradient based method. We will revisit related algorithmic enhancements later.

Data

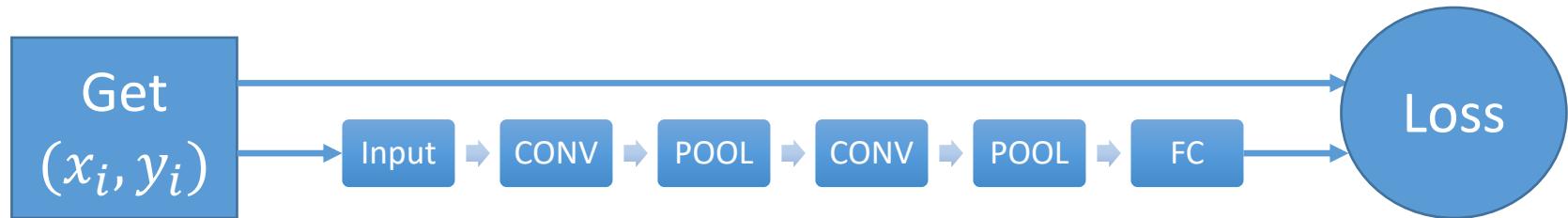
- Data:
 - How is it handled?
 - What is its quality?
- Handling:
 - Deep nets may need to read lots of data (images), so keep them in contiguous spaces of hard-disk
- Quality:
 - Collect as much clean data as possible. At the same time, unclean may also be good enough

Next: Extract the most out of existing data for CNNs

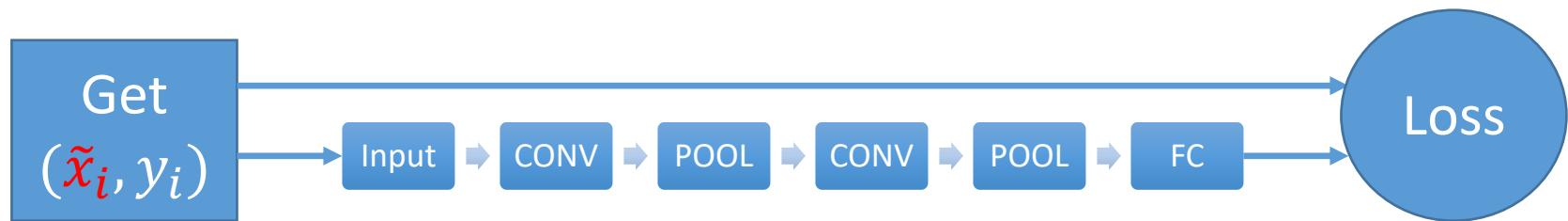
Augmenting Data (I)



Augmenting Data (I)



And



Where $\tilde{x}_i = g(x)$ is a transformation

Augmenting Data (II)

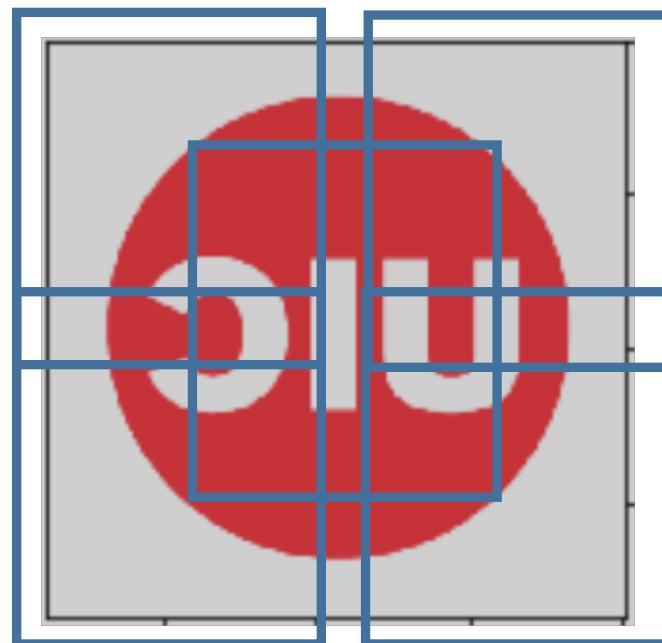
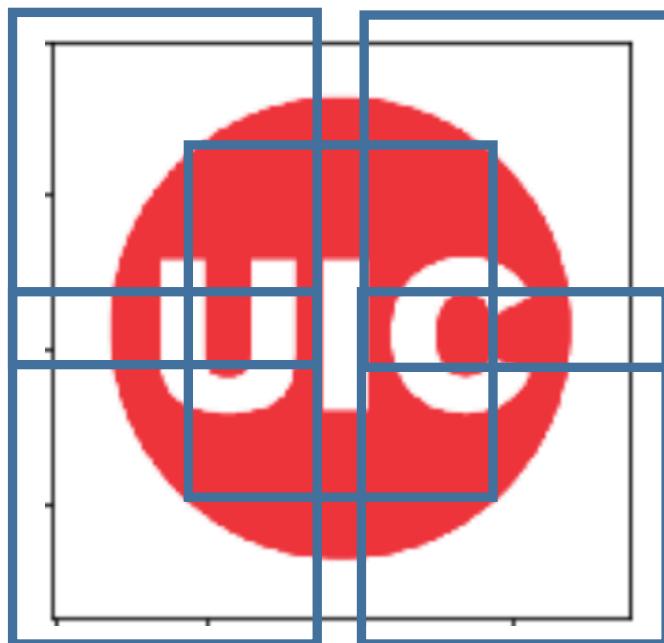
- We are changing the input without changing the label
- We then add this new example to our training set
- Widely used technique!



Random scale

Augmenting Data (III)

- At test time, **average** the predictions of a fixed set of transformations
- Example (for Resnet, the ILSVRC 2015 winner):
 - Image at 5 scales: 224,256,384,460 and 640
 - At each scale, get 10 224*224 crops



Augmenting Data (IV)

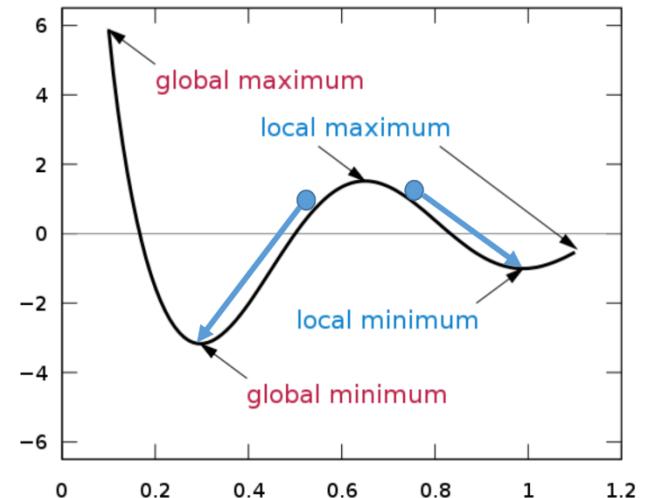
- Other ways to augment data include
 - Changing contrast and color
 - Mix translations, rotations, stretching, shearing, distortions
- This is **very useful** for small datasets
- From one point of view, this is **essentially**
 - Adding some noise during training
 - Marginalizing noise out at test

Model

- We have already seen few choices
 - Activation function or nonlinearities
 - Number of layers and number of neurons per layer
 - CNN filter choices ...
- There are other choices while training deep neural nets (including CNNs) that also **make a difference**
 - Weight initialization
 - Batch normalization
 - Dropout

Model: Weight Initialization

- Weight initialization plays a key role in training deep networks
 - Example: $W = 0$ may be bad
- Not just the issue of local optima
- But also the magnitudes of gradients in backprop
 - Activation statistics (mean and variance) influence gradients
- Heuristics available in the literature to initialize W



Model: Batch Normalization

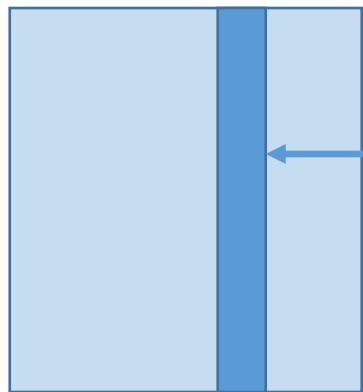
- Activations magnitudes and their statistics depend on the dataset, the network and the nonlinearity used
- Their statistics influence gradient propagation, hence also learning
- Is there a way to control them?
 - Yes, through batch normalization!

Model: Batch Normalization

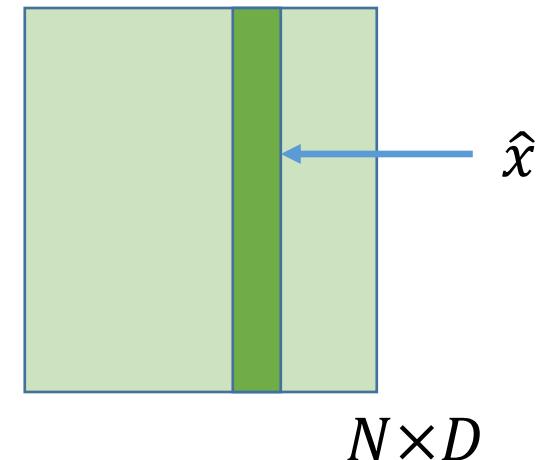
- Idea: Make each activation unit-Gaussian by subtracting the mean and then dividing by standard deviation

Batch-size = N

Number of output neurons = D



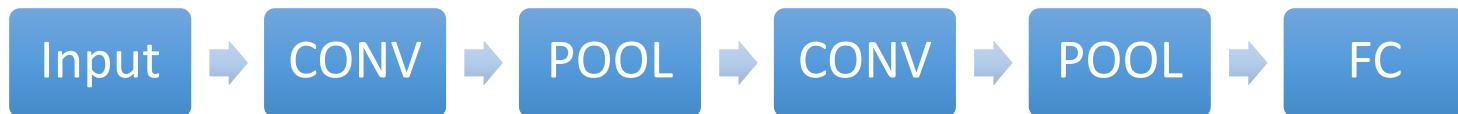
$$x \quad \hat{x} = \frac{\gamma(x - E[x])}{\sqrt{Var[x]}} + \beta$$



- Is a differentiable function: hence no issue with backpropagation
- At test time, there is no batch. Use the training data means and variances

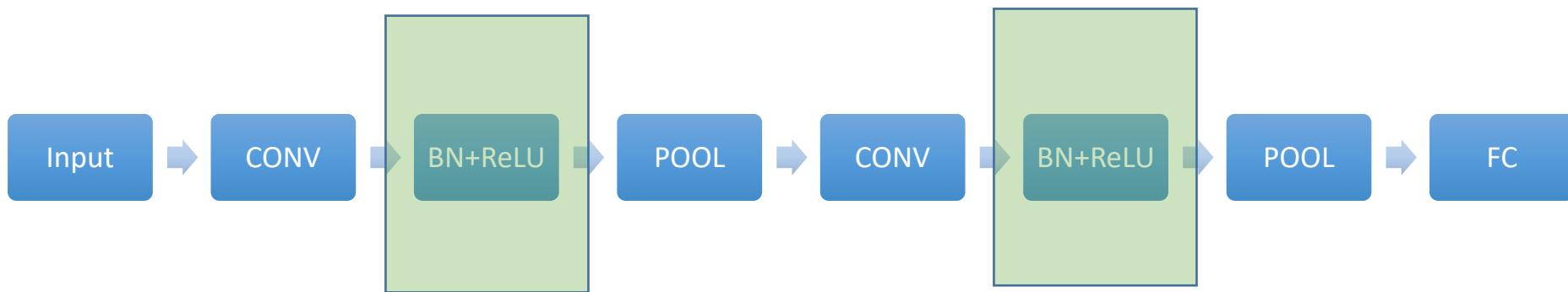
Model: Batch Normalization

- Previously,



- Now

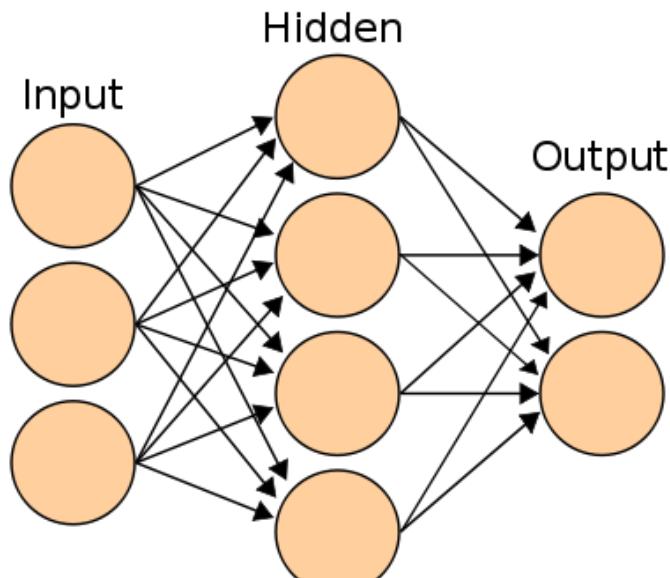
- Insert a Batch Normalization layer between CONV and nonlinearity (ReLU)



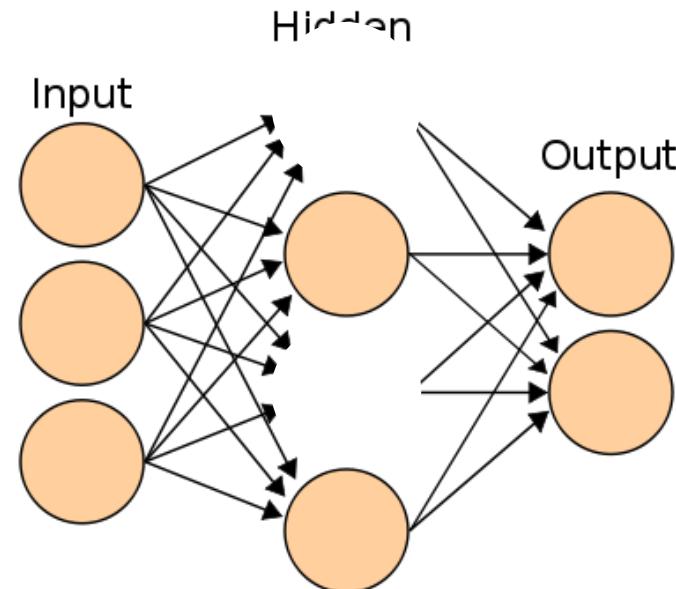
- Empirically observed: improved gradient flows, less sensitive to **initialization**.

Model: Dropout (Regularization)

- Idea: During training, every time we forward pass, we set the output of a few neurons to zero with some probability



Without dropout



One pass with dropout

Model: Dropout (Regularization)

- Intuitively, it is
 - Making us use smaller capacity of the network. Hence, can think of it as a regularization
 - Forcing all the neurons to be useful. Hence there is over-representation or redundancy
- Also think of it as
 - Subsampling a part of the network for each example
 - Thus, we get an ensemble of neural networks that share parameters

Model: Dropout (Regularization)

- Higher probability means stronger regularization
- At test time,
 - Instead of doing many forward passes
 - Perform no dropout
 - Scale all activations by the probability of dropout
- Example:
 - Say dropout with probability p
 - Originally: $f(x, W_1, b_1, W_2, b_2) = W_2 \max(0, W_1 x + b_1) + b_2$
 - With dropout: $W_2 * p * \max(0, W_1 x + b_1) + b_2$

Summary (I)

- CNN are very effective in image related applications.
 - State of the art!
- Exploit specific properties of images
 - Hierarchy of features
 - Locality
 - Spatial invariance
- Lots of **design choices** that have been empirically validated and are intuitive. Still, there is room for improvement.

Summary (II)

- We saw
 - Visualizations to understand how CNNs work
 - Transfer learning applied to CNNs (important for applications)
 - An excellent way to get a deep learning solution working
 - There is no need for large datasets to get started

Summary (III)

- Neural Nets Training Tricks
 - Revisited data: data augmentation
 - Revisited models: initialization, batch norm, dropout
- To train state of the art deep learning systems, you have to rethink:
 - (a) data, (b) models, and (c) optimization¹
 - What is the most bang per buck for your business?
- If the deep learning system is core to the business, look at engineering best practices (we saw some today)

¹We did not cover this in this lecture

Appendix

Sample Questions

- How does a 2 layer feedforward net differ from a linear classifier?
- Describe why nonlinearities are introduced in a neural network? Why is the ReLU non-linearity called a gradient gate?
- Describe the parameter sharing property of a convolutional layer
- How is backpropagation used while optimizing the parameters of a neural network?

Advice

- In spite of all these design choices, for 90% of the applications, pick an architecture that works well on an established dataset (e.g., Imagenet)
- Focus on the application and business considerations, not architectural decisions!

Practical Considerations

- Model choice: nonlinearity, number of layers, number of neurons
- Data preprocessing: batch normalization, subtracting mean of inputs
- Parameter initialization: random or zeroes?
- Learning rate: How to change?
- Batch normalization: re-normalizing activations
- Monitoring learning: plot graphs of training and validation
- Cross validation: hyper-parameter tuning is non-trivial

Partial Robustness to Input Size

- The input image size determines the tensors in intermediate stages
- Example
 - Alexnet requires 224*224*3 sized images
- What if we have a larger sized image?
 - We can ‘convert’ FC layers to equivalent CONV layers for efficiency
 - Then slide the original CNN over the larger image!
 - This leads to a ‘single’ forward pass

Partial Robustness to Input Size

- Instead of a single vector of scores, now we get a bunch of scores

