Convolutional Neural Networks

Most of the materials are taken from here

Nastaran Okati

Outline

- ▷ Architecture
- ▷ ConvNet Layers
- ▷ ConvNet Architectures
- ▷ Transfer Learning
- ▷ Visualization

1.
Architecture

Architecture Overview

- Convolutional Neural Networks are very similar to ordinary Neural Networks
- So what changes? ConvNet architectures make the assumption that the inputs are images, which allows us to encode certain properties into the architecture.
- These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

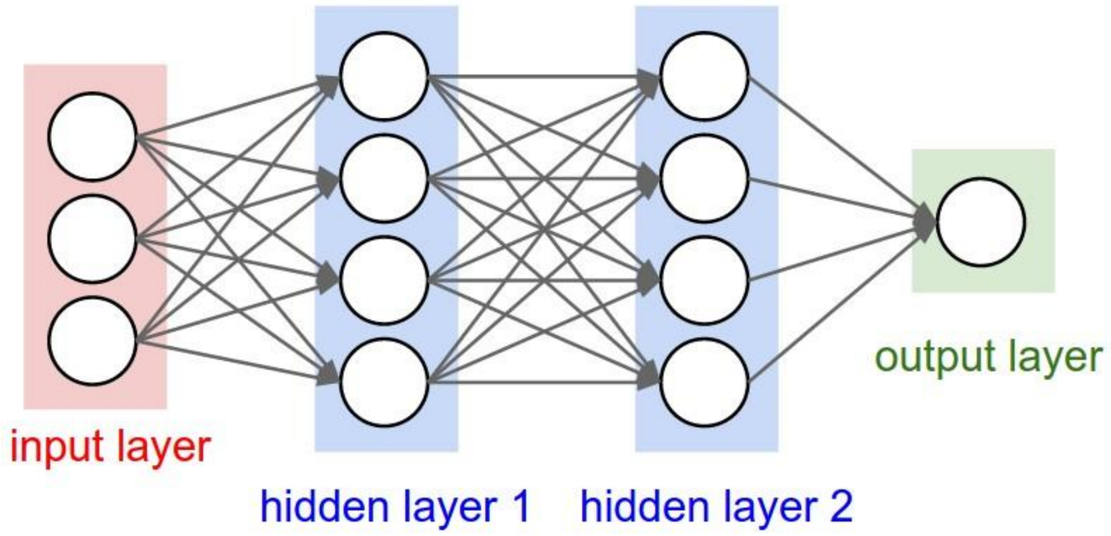
Problem with regular NNs

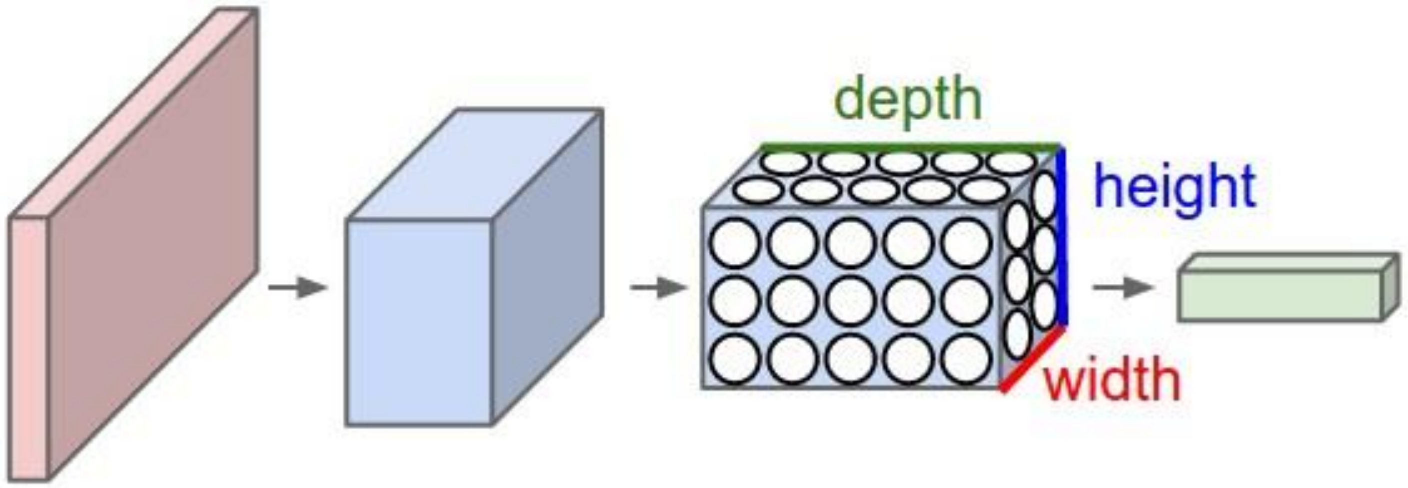
- Regular Neural Nets don’t scale well to full images. Why?
- In CIFAR-10, images are only of size 32x32x3, so a single fully-connected neuron in a first hidden layer of a regular NN would have 32*32*3 = 3072 weights.
- This fully-connected structure does not scale to larger images.
- Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

3D Volumes of Neurons

- ConvNets take advantage of the fact that the input consists of images
- The layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.
- The neurons in a layer will only be connected to a small region of the layer before it
- It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.

CNN vs NN


input layer
hidden layer 1
hidden layer 2
output layer


depth
height
width

2.

ConvNet Layers

Layers

- We use three main types of layers to build ConvNet architectures
  - Convolutional Layer
  - Pooling Layer
  - Fully-Connected Layer (as seen in regular Neural Networks)

Example Architecture

- ▷ INPUT [32x32x3] will hold the raw pixel values of the image
- ▷ CONV layer will compute the output of neurons that are connected to local regions in the input. This may result in volume such as [32x32x12] if we decided to use 12 filters
- ▷ ReLU layer will apply an elementwise activation function. This leaves the size of the volume unchanged ([32x32x12]).
- ▷ POOL layer will perform a downsampling operation along the spatial dimensions resulting in volume such as [16x16x12].
- ▷ FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10]

Layer Parameters

- Note that some layers contain parameters and other don’t.
- The CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons).
- the RELU/POOL layers will implement a fixed function
- The parameters in the CONV/FC layers will be trained with gradient descent

Convolutional Layer

- ▷ The CONV layer's parameters consist of a set of learnable filters.
- ▷ Every filter is small spatially but extends through the full depth of the input volume.
- ▷ During the forward pass, we slide each filter across the width and height of the input volume
- ▷ As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position.

Convolutional Layer

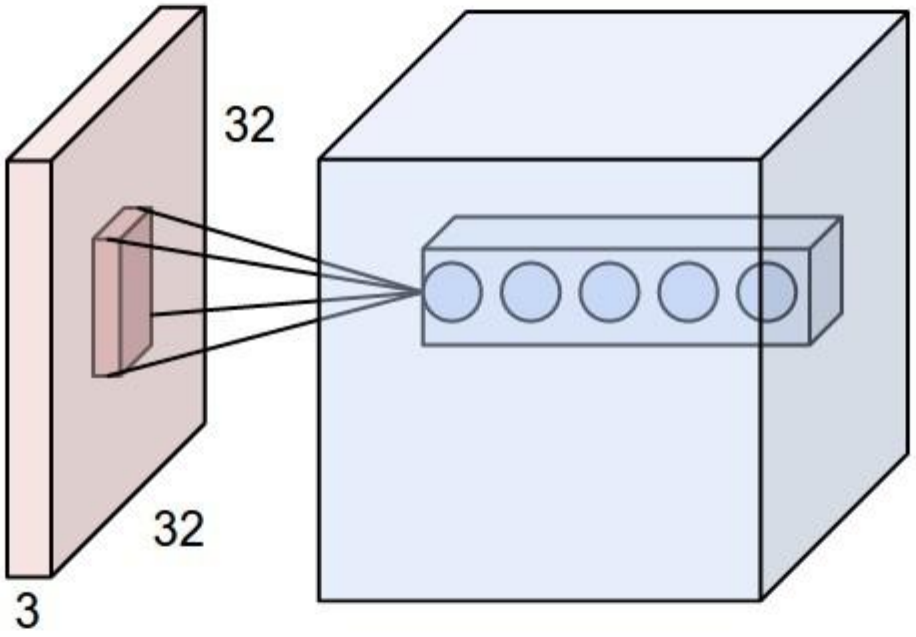
- When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume
- We will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size).
- The connections are local in space (along width and height), but always full along the entire depth of the input volume.

Example Conv Layer

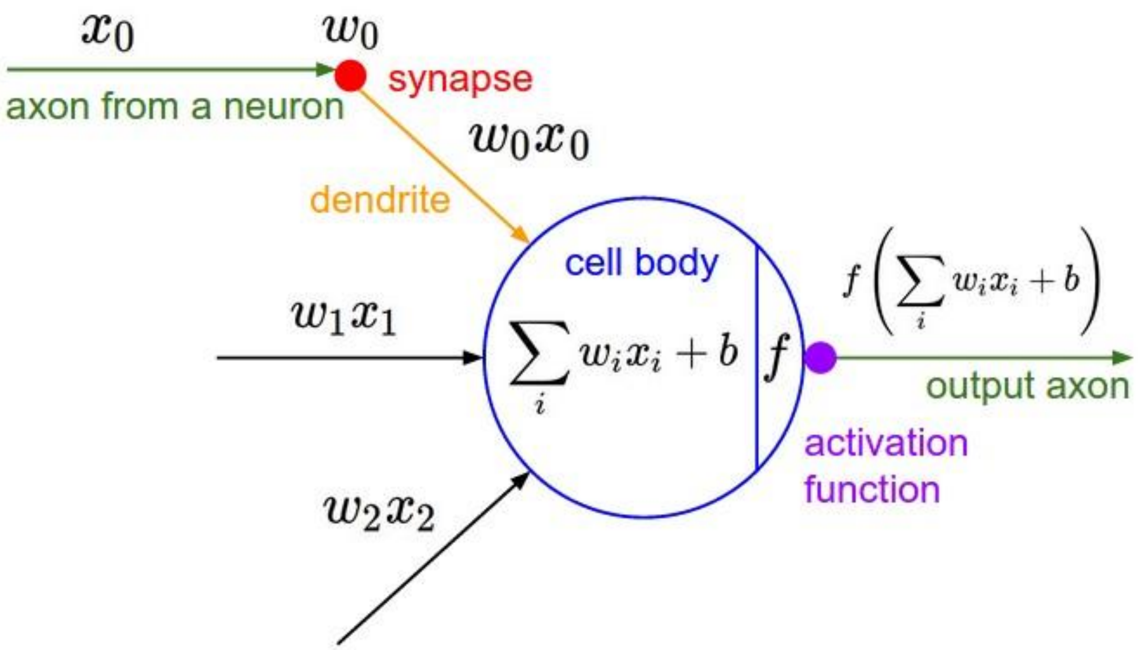
- Suppose that the input volume has size [32x32x3].

- If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter).

- Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.



32
32
3



x_0
w_0
synapse
axon from a neuron
w_0x_0
dendrite
cell body
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
w_1x_1
output axon
activation function
w_2x_2

Spatial arrangements

- Three hyperparameters control the size of the output volume: the depth, stride and zero-padding.

- The depth of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use.

- Stride corresponds to the number of pixels we jump over when sliding the window. This will produce smaller output volumes spatially.

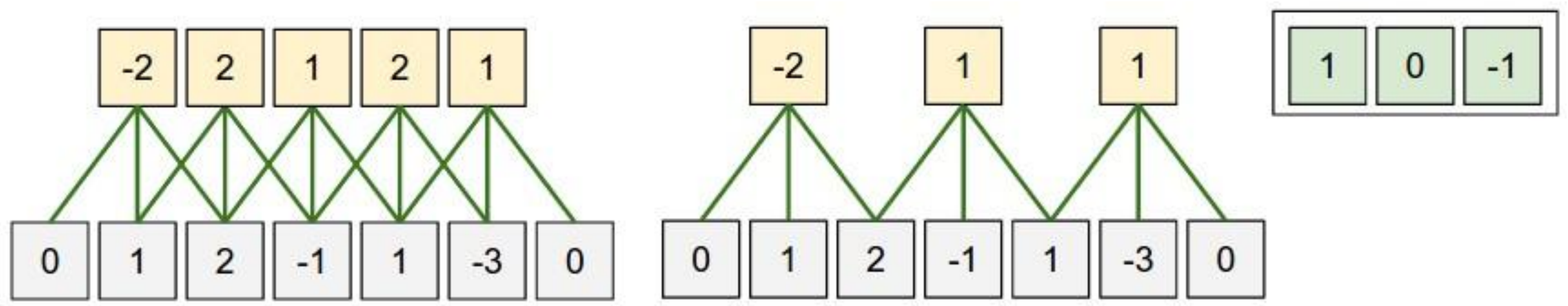
- We can pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter.

Computing the size of the output

With input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border, the correct formula for calculating how many neurons “fit” is given by $(W-F+2P)/S+1$

Effect of stride and padding

Note that the input dimension is 5 and the output dimension is equal: also 5. This worked out so because our receptive fields were 3 and we used zero padding of 1



Constraints on stride

- If the neurons don’t fit neatly and symmetrically then the setting of hyperparameters is considered to be invalid

- A ConvNet library could throw an exception or zero pad the rest to make it fit, or crop the input to make it fit

Parameter sharing

- Suppose we have images of size 227x227x3, F=11, S=4, P=0, K=96
  - What is the output volume size?
  - How many neurons are there in the Conv layer?
  - How many weights are there in the Conv layer?
- Huge number of weights!
- Solution: If one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2).
- We are going to constrain the neurons in each depth slice to use the same weights and bias.
  - How many weights in this new setting?

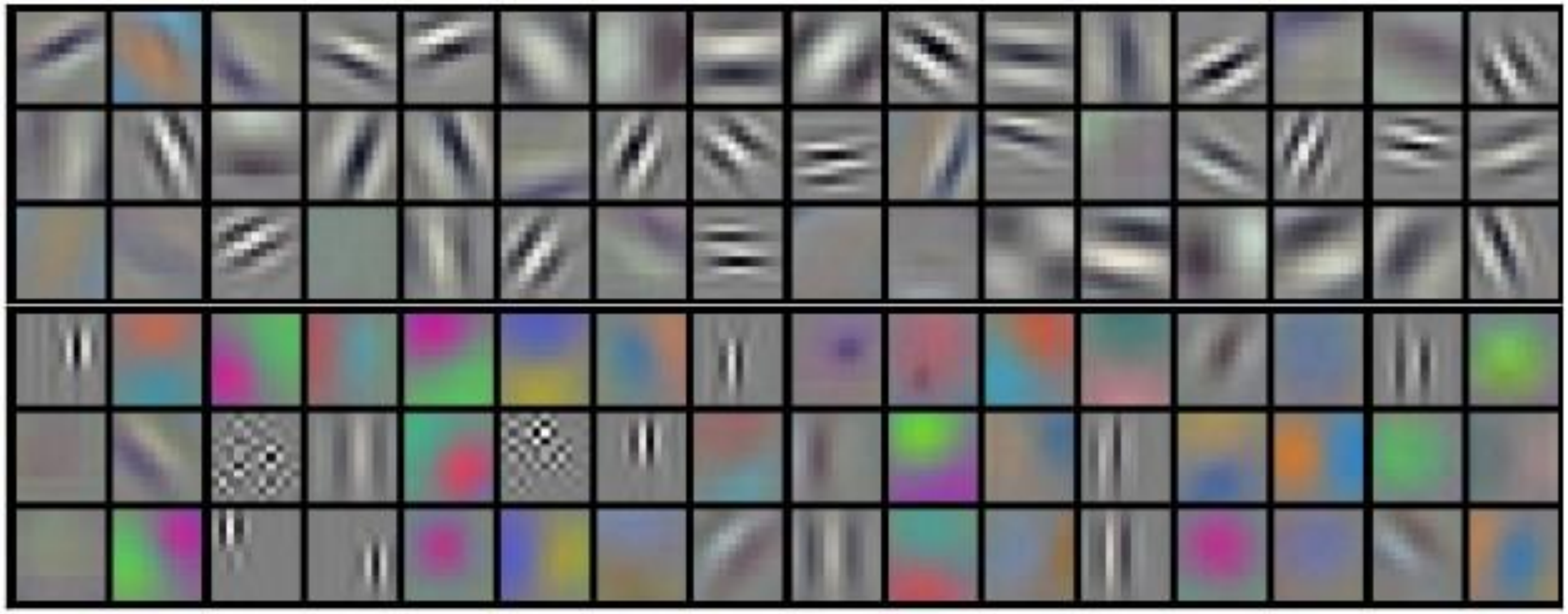
Parameter sharing

▷ During backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

▷ Notice that if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a convolution of the neuron's weights with the input volume

▷ This is why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input.

Example learned filters



Summary of Conv Layer

- Accepts a volume of size $W1 \times H1 \times D1$

- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

- Produces a volume of size $W2 \times H2 \times D2$ where:
  - $W2=(W1-F+2P)/S+1$
  - $H2=(H1-F+2P)/S+1$
  - $D2=K$

- With parameter sharing, it introduces $F \cdot F \cdot D1$ weights per filter, for a total of $(F \cdot F \cdot D1) \cdot K$ weights and K biases.

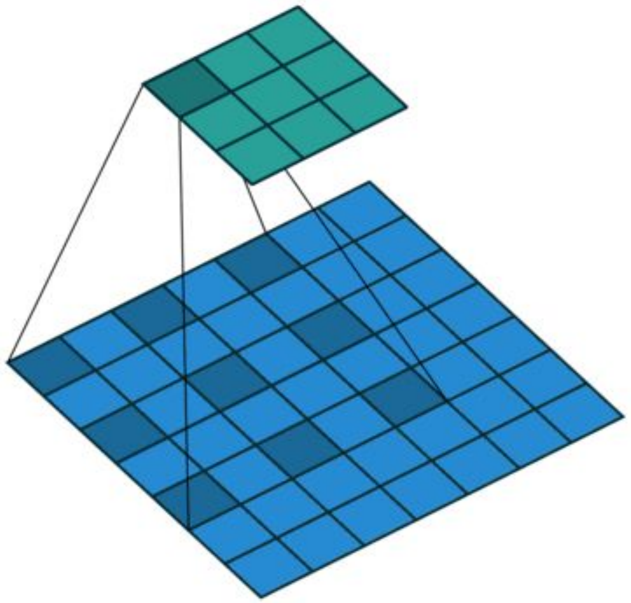
Backpropagation in Conv Layer

The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). This is easy to derive in the 1-dimensional case with a toy example

Dilated convolutions

- A recent development is to introduce one more hyperparameter to the CONV layer called the dilation

- Filters that have spaces between each cell, called dilation



Pooling Layer

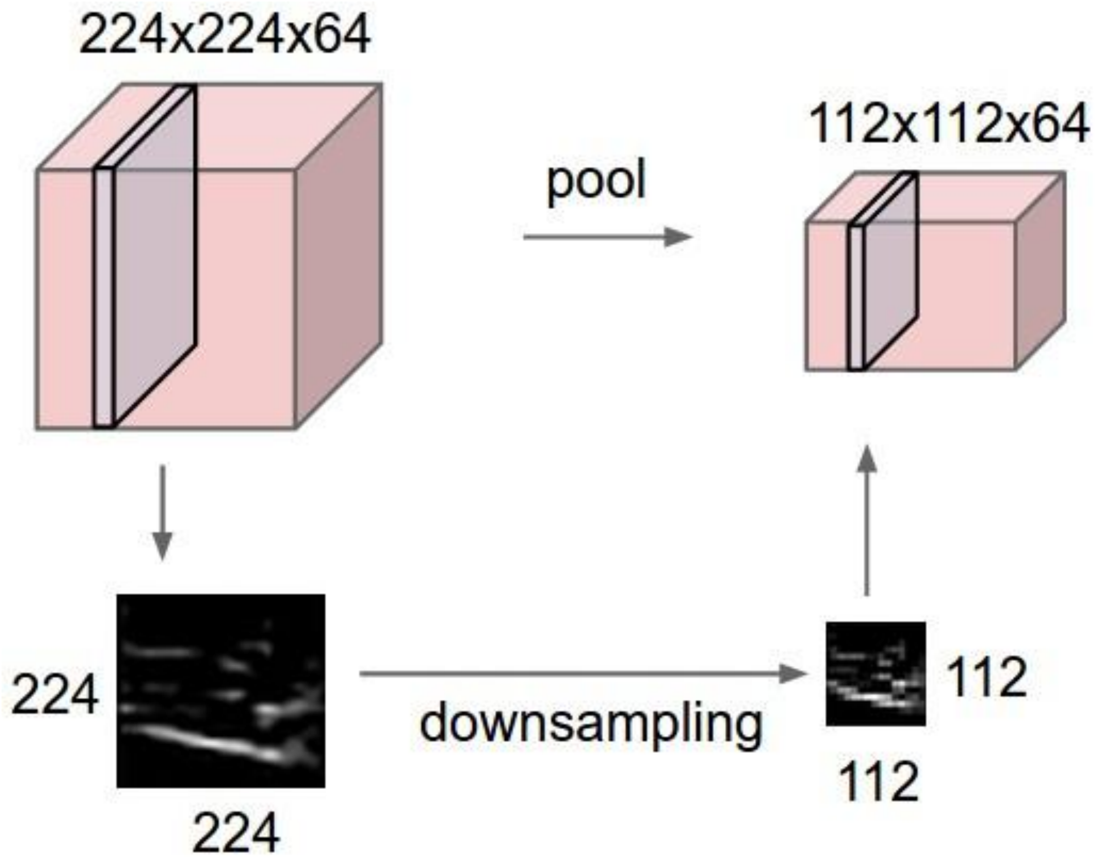
- This layer reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting

- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.

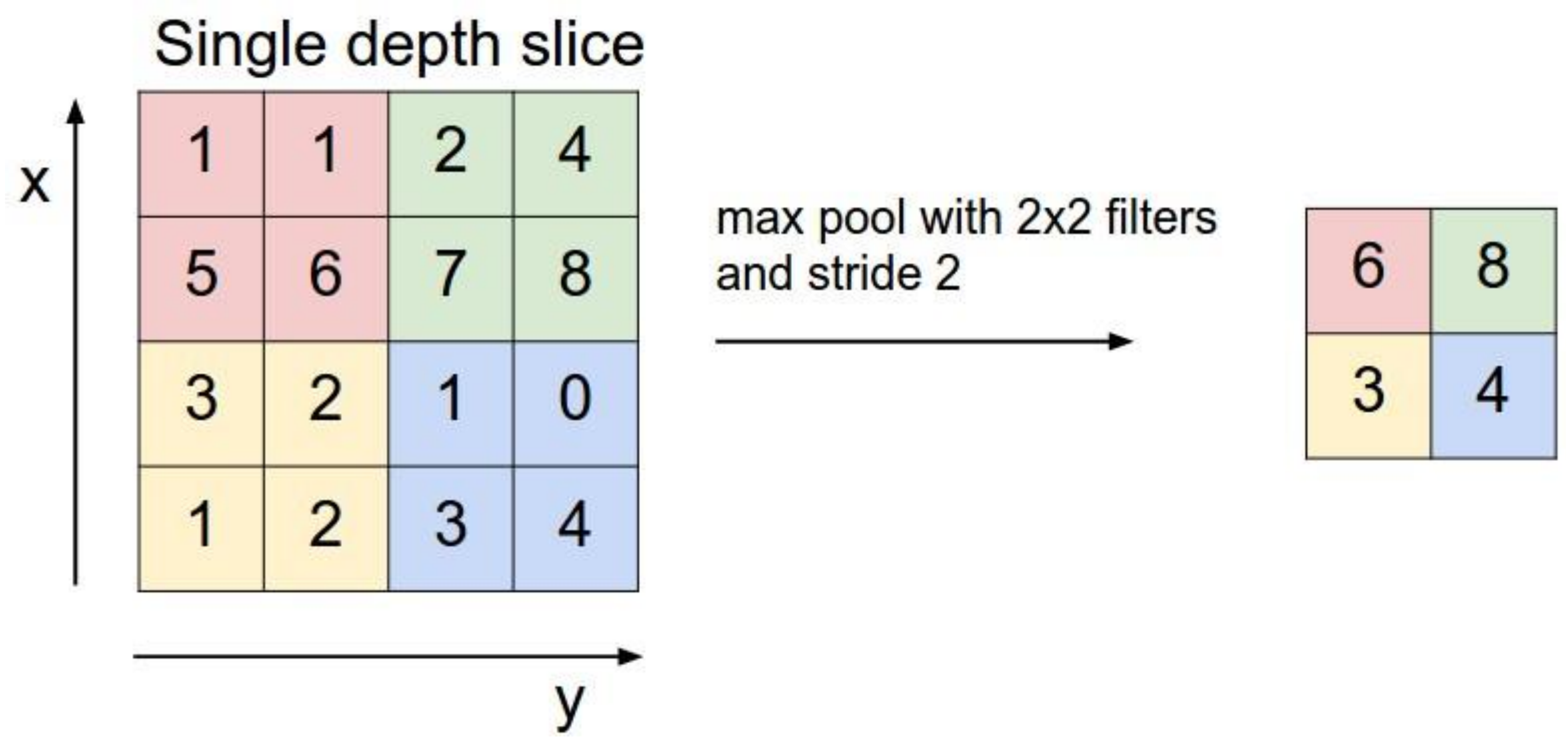
- Pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height

- What percentage of the activation do we keep after this layer?

Pooling



Max Pool


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters and stride 2
6 8
3 4

Summary of Pooling Layer

- Accepts a volume of size $W1 \times H1 \times D1$

- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,

- Produces a volume of size $W2 \times H2 \times D2$ where:
  - $W2=(W1-F)/S+1$
  - $H2=(H1-F)/S+1$
  - $D2=D1$

- Introduces zero parameters since it computes a fixed function of the input

General Pooling

- In addition to max pooling, the pooling units can also perform other functions, such as average pooling or even L2-norm pooling.

- Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.

Backpropagation in pooling layer

- The backward pass for a max(x, y) operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward pass.

- During the forward pass of a pooling layer it is common to keep track of the index of the max activation so that gradient routing is efficient during backpropagation.

Fully-connected Layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks.
- The only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters.
- The neurons in both layers still compute dot products, so their functional form is identical

3.
ConvNet Architecture

Common Architectures

- INPUT -> FC, implements a linear classifier.
- INPUT -> CONV -> ReLU -> FC
- INPUT -> [CONV -> ReLU -> POOL]*2 -> FC -> ReLU -> FC
  - A single CONV layer between every POOL layer.
- INPUT -> [CONV -> ReLU -> CONV -> ReLU -> POOL]*3 -> [FC -> ReLU]*2 -> FC
  - Two CONV layers stacked before every POOL layer. This is a good idea for deeper networks, because multiple stacked CONV layers can develop more complex features of the input before the destructive pooling operation.

Don’t be a hero!

- In practice: use whatever works best on ImageNet
- Instead of rolling your own architecture for a problem, you should look at whatever architecture currently works best on ImageNet, download a pretrained model and finetune it on your data.
- You should rarely ever have to train a ConvNet from scratch or design one from scratch.

Famous Architectures!

- **LeNet:** The first successful applications of CNNs
- **AlexNet:** The first work that popularized CNN in Computer Vision
- **ZFNet**
- **GoogLeNet:** dramatically reduced the number of parameters
- **VGGNet**
- **ResNet:** currently by far state of the art CNN models and the default choice for using ConvNets in practice

Layer sizing patterns

- The input layer should be divisible by 2 many times.
  - Common numbers are 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. ImageNet), 384, and 512.

- The conv layers should be using small filters (e.g. 3x3 or at most 5x5), using a stride of S=1, and padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input.
  - What is proper P for F=3 and F=5?

- The common setting for pooling layer is to use max-pooling with 2x2 receptive fields (i.e. F=2), and with a stride of 2 (i.e. S=2).
  - It is very uncommon to see receptive field sizes for max pooling that are larger than 3. Why?

3.
Transfer Learning

Transfer Learning

- ▷ In practice, very few people train an entire CNN from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size.

- ▷ Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use transfer learning:
    - ○ ConvNet as fixed feature extractor
    - ○ Fine-tuning the ConvNet

ConvNet as fixed feature extractors

- Take a ConvNet pre-trained on ImageNet, remove the last fully-connected layer (this layer's outputs are the 1000 class scores for a different task like ImageNet), then treat the rest of the ConvNet as a fixed feature extractor for the new dataset.

- Once you extract the codes for all images, train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset. (we call the features CNN codes)

Fine-tuning the ConvNet

- The second strategy is to also fine-tune the weights of the pretrained network by continuing the backpropagation.

- It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed and only fine-tune some higher-level portion of the network.

- The earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset.

Pre-trained models

- Modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet

- It is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning.

When and how to fine-tune?

- New dataset is small and similar to original dataset.
  - train a linear classifier on the CNN codes. Why?
- New dataset is large and similar to the original dataset
  - We can fine-tune through the full network. Why?
- New dataset is small but very different from the original dataset.
  - train the SVM classifier from activations somewhere earlier in the network. Why?
- New dataset is large and very different from the original dataset.
  - We can train from scratch, however it is still better to finetune the entire network

Final words on fine-tuning

- If you wish to use a pretrained network, you may be slightly constrained in terms of the architecture you can use for your new dataset. For example, you can’t arbitrarily take out Conv layers from the pretrained network.

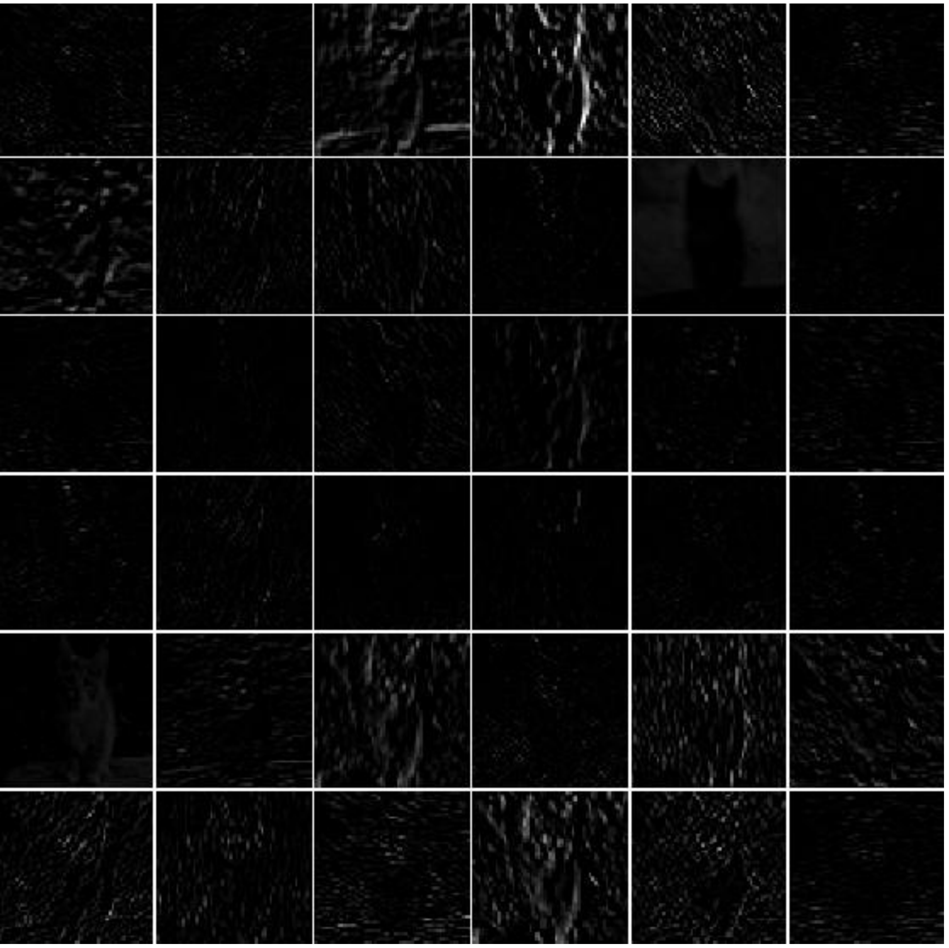
- However, some changes are straight-forward: Due to parameter sharing, you can easily run a pretrained network on images of different spatial size.

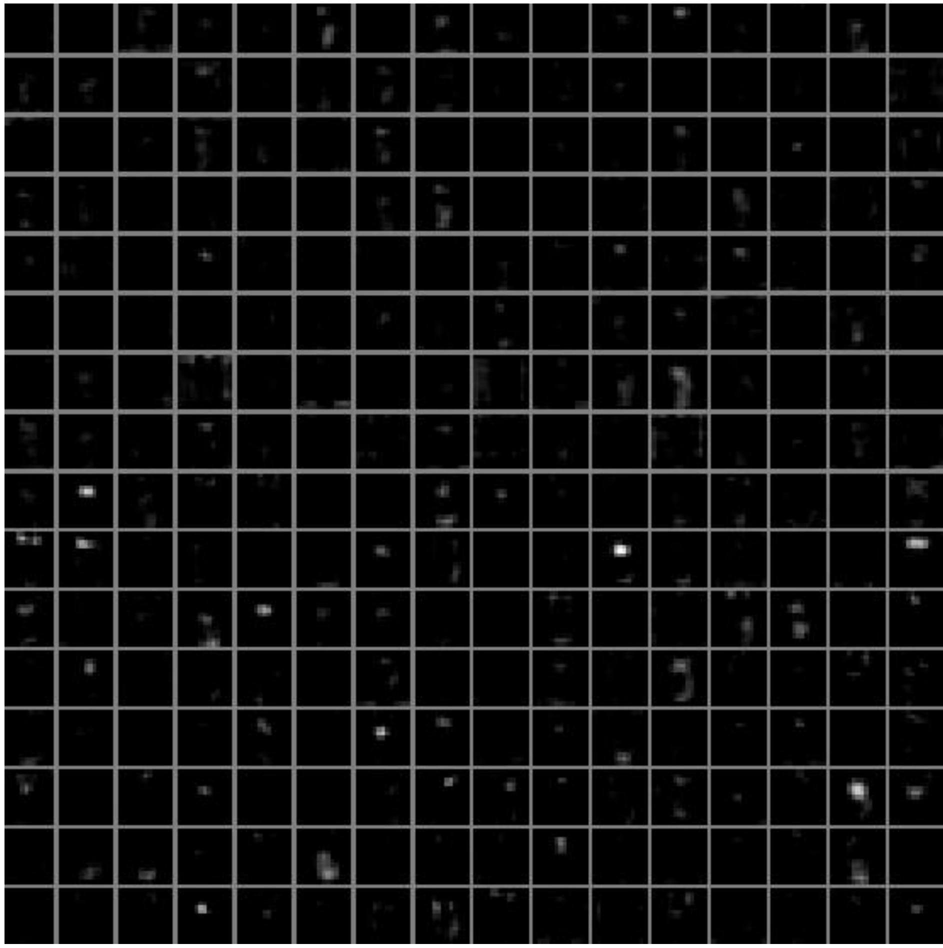
- It’s common to use a smaller learning rate for ConvNet weights that are being fine-tuned. Why?

4.

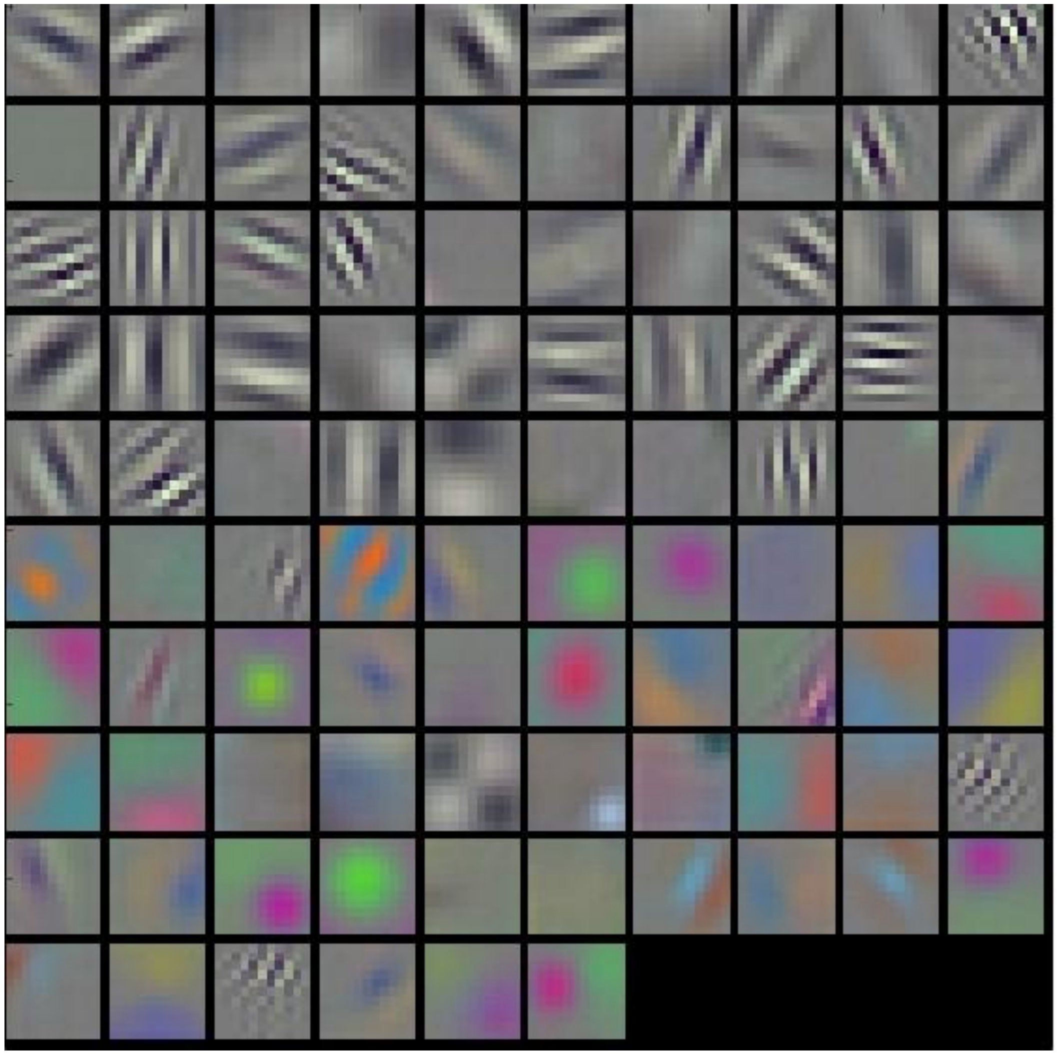
Visualization

Layer Activations

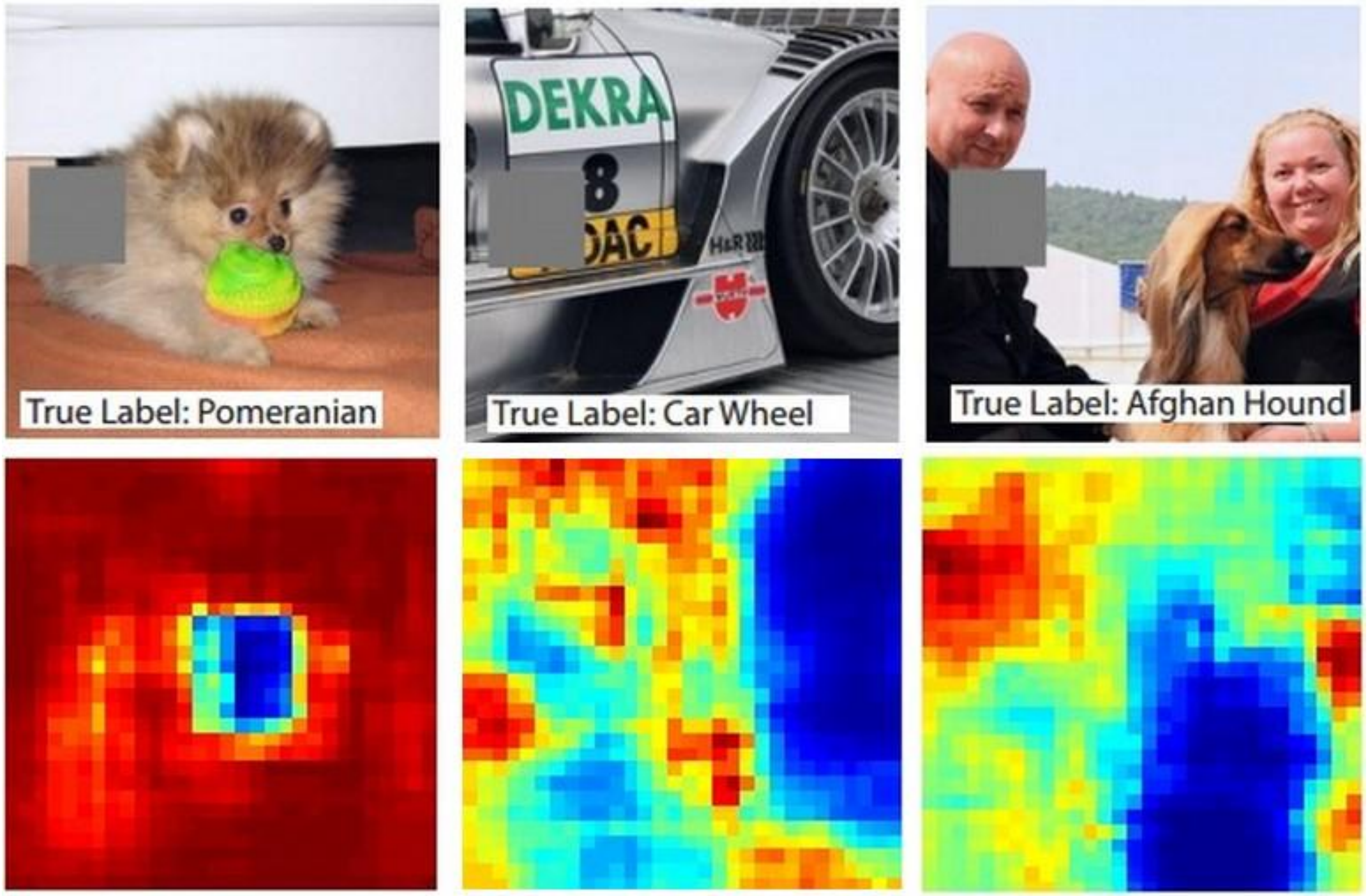




Conv Filters



Occluding


True Label: Pomeranian
True Label: Car Wheel
True Label: Afghan Hound

Thanks!

Any questions?