# Regularization and Dropout in Neural Networks / Convolution Neural Networks

**Dr. Mahmoud N Mahmoud**
*mnmahmoud@ncat.edu*

North Carolina A & T State University
Regularization

October 14, 2020

# Talk Overview


1. Regularization in Neural Networks
2. Introduction to Convolution Neural Networks
3. CNN Architecture Example
4. Transfer Learning
5. CNN Architectures

# Outline

1 Regularization in Neural Networks

2 Introduction to Convolution Neural Networks

3 CNN Architecture Example

4 Transfer Learning

5 CNN Architectures

# REGULARIZING NEURAL NETWORKS

**We have several means by which to help "regularize" neural networks – that is, to prevent overfitting**

- Regularization penalty in cost function
- Dropout
- Early stopping
- Stochastic / Mini-batch Gradient descent (to some degree)

# PENALIZED COST FUNCTION

- One option is to explicitly add a penalty to the loss function for having high weights.
- This is a similar approach to Ridge Regression

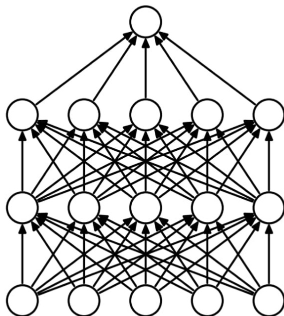
$$J = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{m} W_i^2$$

- Can have an analogous expression for Categorical Cross Entropy

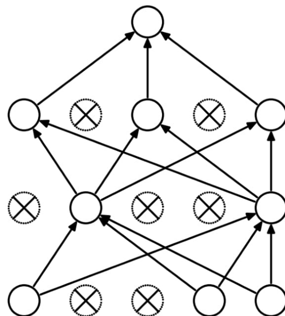
# DROPOUT

- Dropout is a mechanism where at each training iteration (batch) we randomly remove a subset of neurons
- This prevents the neural network from relying too much on individual pathways, making it more "robust"
- At test time we "rescale" the weight of the neuron to reflect the percentage of the time it was active

# DROPOUT—VISUALIZATION



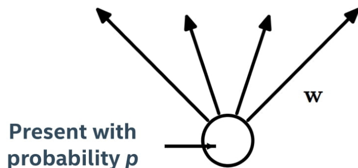
(a) Standard Neural Net

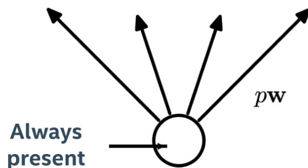


(b) After applying dropout

# DROPOUT—VISUALIZATION

**If the neuron was present with probability *p*, at test time we scale the outbound weights by a factor of *p*.**




(a) At training time (b) At test time

# EARLY STOPPING

- Another, more heuristical approach to regularization is early stopping.
- This refers to choosing some rules after which to stop training.
- Example:
  - Check the validation log-loss every 10 epochs.
  - If it is higher than it was last time, stop and use the previous model (i.e. from 10 epochs previous)

# OPTIMIZERS

- We have considered approaches to gradient descent which vary the number of data points involved in a step.
- However, they have all used the standard update formula:

$$W := W - \alpha \cdot \nabla J$$

- There are several variants to updating the weights which give better performance in practice.
- These successive “tweaks” each attempt to improve on the previous idea.
- The resulting (often complicated) methods are referred to as “optimizers”.

# MOMENTUM

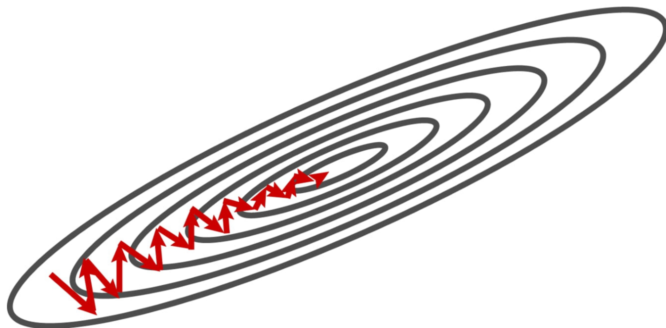
- Idea, only change direction by a little bit each time.
- Keeps a "running average" of the step directions, smoothing out the variation of the individual points.

$$v_{\mathrm{t}} := \eta \cdot v_{t-1} - (1-\eta) \cdot \nabla J$$

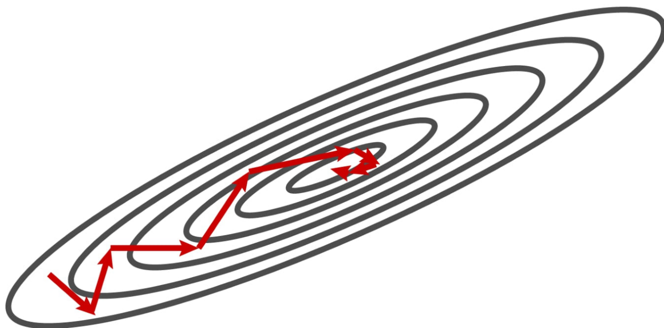
$$\mathrm{W} := W - \alpha \cdot v_t$$

- Here, $\eta$ is referred to as the "momentum". It is generally given a value <1

## GRADIENT DESCENT VS MOMENTUM



# GRADIENT DESCENT VS MOMENTUM



# ADAGRAD

- Idea: scale the update for each weight separately.
- Update frequently-updated weights less
- Keep running sum of previous updates
- Divide new updates by factor of previous sum

$$W := W - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla J$$

$$G_t := \sum_{i=1}^{t} \nabla J_i^2$$

# RMSPROP

- Quite similar to AdaGrad.
- Rather than using the sum of previous gradients, decay older gradient more than more recent one
- More adaptive to recent updates

$$W := W - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla J$$

$$G_t := \beta \cdot G_t - (1 - \beta) \cdot \nabla J^2$$

# ADAM

**Idea: use both first-order and second-order change information and decay both over time.**

$$\mathrm{m_t} = \beta_1 \mathrm{m_{t-1}} + (1 - \beta_1)\nabla J \qquad \mathrm{v_t} = \beta_2 \mathrm{v_{t-1}} + (1 - \beta_2)\nabla J^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_1^t}$$

$$W := W - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \odot \hat{m}_t$$

# WHICH ONE SHOULD I USE?!

- RMSProp and Adam seem to be quite popular now.
- Difficult to predict in advance which will be best for a particular problem.
- Still an active area of inquiry.

# Outline

1 Regularization in Neural Networks

2 Introduction to Convolution Neural Networks

3 CNN Architecture Example

4 Transfer Learning

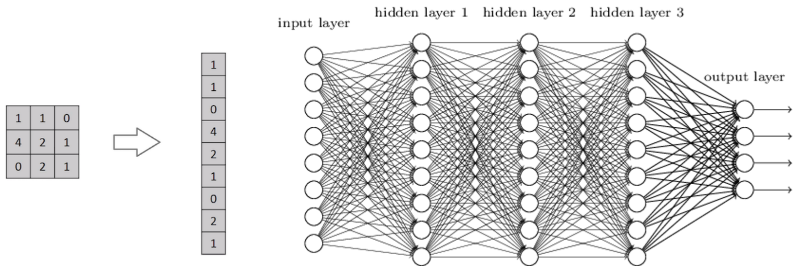
5 CNN Architectures

# MOTIVATION—IMAGE DATA

- So far, the structure of our neural network treats all inputs interchangeably.
- No relationships between the individual inputs
- Just an ordered set of variables
- We want to incorporate domain knowledge into the architecture of a Neural Network.

# Convolutional Neural Networks (ConvNets)

- It is a special structure to deal with image inputs.
- Why not just flatten the image and feed it to a Multi-Level Perceptron for classification purposes?



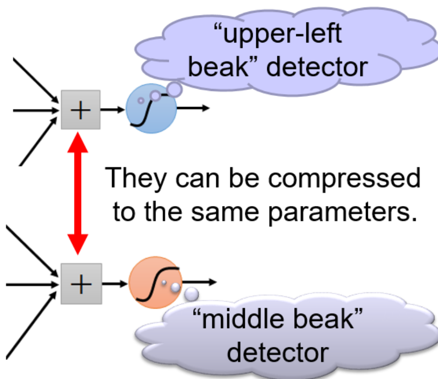

Do we really need all the edges?

# MOTIVATION—IMAGE DATA

- Fully connected would require a vast number of parameters
- MNIST images are small (32 x 32 pixels) and in grayscale
- Color images are more typically at least (200 x 200) pixels x 3 color channels (RGB) = 120,000 values.
- A single fully connected layer would require (200x200x3)2 = 14,400,000,000 weights!
- Variance (in terms of bias-variance) would be too high
- So we introduce “bias” by structuring the network to look for certain kinds of patterns

# Convolutional Neural Networks (ConvNets)

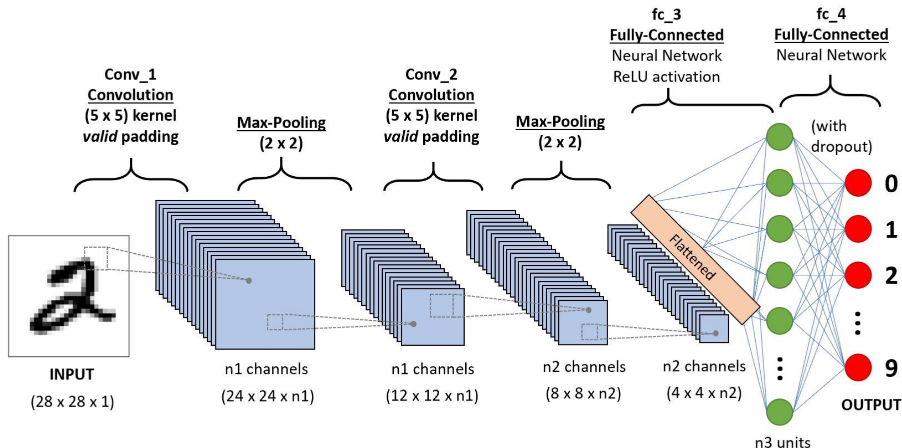
What about Spatial and Temporal dependencies between pixels. (Edges, patterns, curves ...)




In MLP, Large network is needed for all possible positioning of the peak

# Convolutional Neural Networks (ConvNets)

In primitive image processing methods filters are hand-engineered, with enough training, ConvNets have the ability to learn filters/characteristics.



# MOTIVATION

- Features need to be "built up"
- Edges -> shapes -> relations between shapes
- Textures
- Cat = two eyes in certain relation to one another + cat fur texture.
- Eyes = dark circle (pupil) inside another circle.
- Circle = particular combination of edge detectors.
- Fur = edges in certain pattern.

# KERNELS

- A *kernel* is a grid of weights "overlaid" on image, centered on one pixel
- Each weight multiplied with pixel underneath it
- Output over the centered pixel is $\sum_{p=1}^{P} W_p \cdot pixel_p$
- Used for traditional image processing techniques:
  - Blur
  - Sharpen
  - Edge detection
  - Emboss

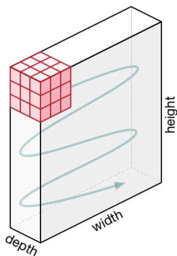
# Convolution Layer The Kernel (1/2)

It is a small matrix of learnable weights which have a purpose to detect certain feature in the image (e.g. Horizontal edes).

- **Stride:** The number of shifts the filter is moving for each convolution step.
- **Padding:** Pads the input volume with zeros around the border.
  - Can preserve (as much as possible) the size of the input to prevent losing information.

# Convolution Layer The Kernel (2/2)

For RGB images kerenls can be 3d matrices.



# KERNELS AS FEATURE DETECTORS

## Can think of kernels as a "local feature detectors"

**Vertical Line Detector**

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

**Horizontal Line Detector**

| | | |
|---|---|---|
| -1 | -1 | -1 |
| 1 | 1 | 1 |
| -1 | -1 | -1 |

**Corner Detector**

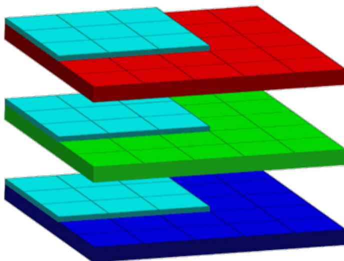
| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 1 | 1 |
| -1 | 1 | 1 |

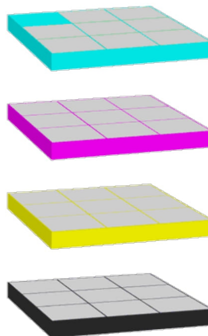
# CONVOLUTIONAL NEURAL NETS

## Primary Ideas behind Convolutional Neural Networks:

- Let the Neural Network learn which kernels are most useful
- Use same set of kernels across entire image (translation invariance)
- Reduces number of parameters and "variance" (from bias-variance point of view)

# CONVOLUTIONS



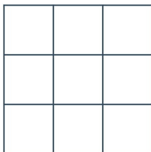


# CONVOLUTION SETTINGS—GRID SIZE

## Grid Size (Height and Width):

- The number of pixels a kernel "sees" at once
- Typically use odd numbers so that there is a "center" pixel
- Kernel does not need to be square

**Height: 3, Width: 3**



**Height: 1, Width: 3**

**Height: 3, Width: 1**

# CONVOLUTION SETTINGS—PADDING

## Padding

- Using Kernels directly, there will be an “edge effect”
- Pixels near the edge will not be used as “center pixels” since there are not enough surrounding pixels
- Padding adds extra pixels around the frame
- So every pixel of the original image will be a center pixel as the kernel moves across the image
- Added pixels are typically of value zero (zero-padding)

# WITHOUT PADDING

**Input**

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 0 | 3 | 1 |
| 1 | 0 | 0 | 2 | 2 |
| 2 | 1 | 2 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 2 | 1 | 1 | 1 |

**Kernel**

| | | |
|---|---|---|
| -1 | 1 | 2 |
| 1 | 1 | 0 |
| -1 | -2 | 0 |

**Output**

| | | |
|---|---|---|
| -2 | | |
| | | |
| | | |

# WITH PADDING

Input

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 0 | 3 | 1 | 0 |
| 0 | 1 | 0 | 0 | 2 | 2 | 0 |
| 0 | 2 | 1 | 2 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Kernel

| | | |
|---|---|---|
| -1 | 1 | 2 |
| 1 | 1 | 0 |
| -1 | -2 | 0 |

Output

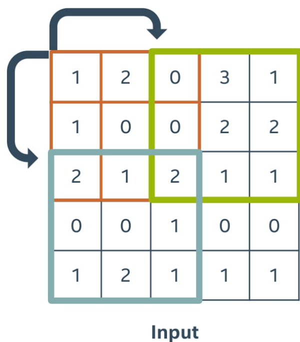
| | | | | |
|---|---|---|---|---|
| -1 | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# CONVOLUTION SETTINGS

## Stride

- The ”step size” as the kernel moves across the image
- Can be different for vertical and horizontal steps (but usually is the same value)
- When stride is greater than 1, it scales down the output dimension

# STRIDE 2 EXAMPLE—NO PADDING




| -1 | 1 | 2 |
|---|---|---|
| 1 | 1 | 0 |
| -1 | -2 | 0 |

**Kernel**

| -2 | 3 |
|---|---|
| 0 | |

**Output**

# STRIDE 2 EXAMPLE—WITH PADDING

| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 3 | 1 | 0 |
| 0 | 1 | 0 | 0 | 2 | 2 | 0 |
| 0 | 2 | 1 | 2 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Input**

| -1 | 1 | 2 |
|---|---|---|
| 1 | 1 | 0 |
| -1 | -2 | 0 |

**Kernel**

| -1 | 2 | |
|---|---|---|
| 3 | | |
| | | |

**Output**

# CONVOLUTIONAL SETTINGS—DEPTH

- In images, we often have multiple numbers associated with each pixel location.
- These numbers are referred to as “channels”
  - RGB image—3 channels
  - CMYK—4 channels
- The number of channels is referred to as the “depth”
- So the kernel itself will have a “depth” the same size as the number of input channels
- Example: a 5x5 kernel on an RGB image
  - There will be 5x5x3 = 75 weights

# CONVOLUTIONAL SETTINGS—DEPTH

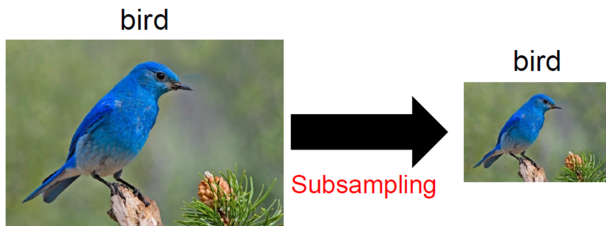
- The output from the layer will also have a depth
- The networks typically train many different kernels
- Each kernel outputs a single number at each pixel location
- So if there are 10 kernels in a layer, the output of that layer will have depth 10.

# POOLING

- Idea: Reduce the image size by mapping a patch of pixels to a single value.
- Shrinks the dimensions of the image.
- Does not have parameters, though there are different types of pooling operations.

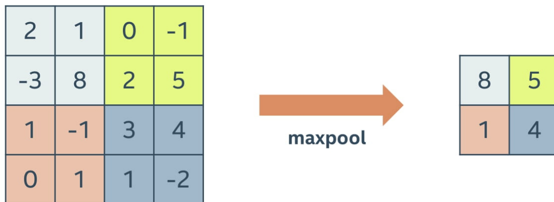
# Pooling Layer

- Pooling layer is responsible for reducing the spatial size of the Convolved Feature.
- This is to decrease the computational power required to process the data.
- Subsampling pixels will not change the object.



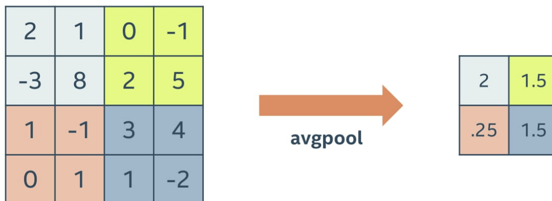
# POOLING: MAX-POOL

- For each distinct patch, represent it by the maximum
- 2x2 maxpool shown below

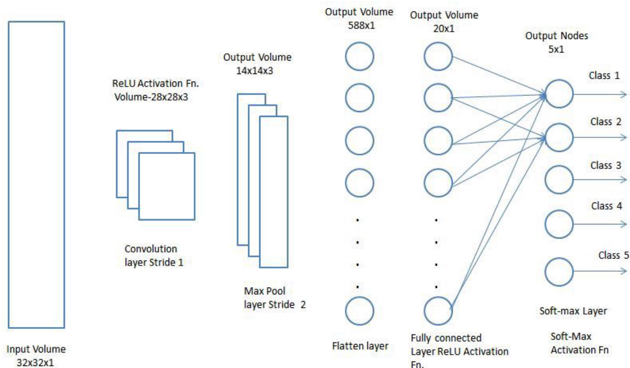


# POOLING: AVERAGE-POOL

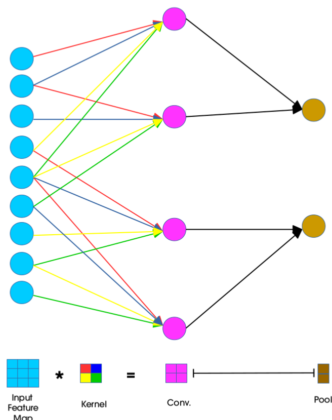
- For each distinct patch, represent it by the average
- 2x2 avgpool shown below



# Flatten - Fully Connected Layer



# CNN vs MLP




In CNN, very few weights to learn compared to MLP without loss of generality

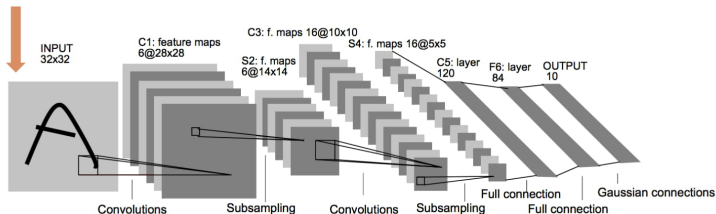
# Outline


1. Regularization in Neural Networks
2. Introduction to Convolution Neural Networks
3. CNN Architecture Example
4. Transfer Learning
5. CNN Architectures

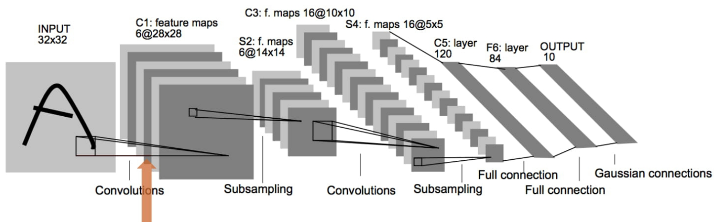
# LeNet-5

- Created by Yann LeCun in the 1990s
- Used on the MNIST data set
- Novel Idea: Use convolutions to efficiently learn features on data set

# LeNet—Structure Diagram

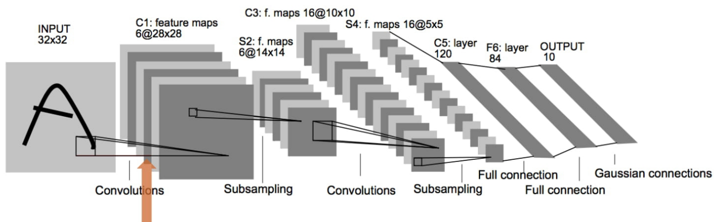


# LeNet—Structure Diagram

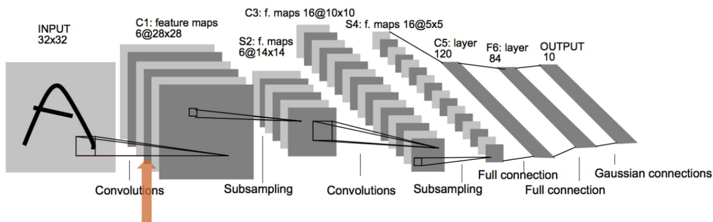



Next, we have a convolutional layer.

# LeNet—Structure Diagram

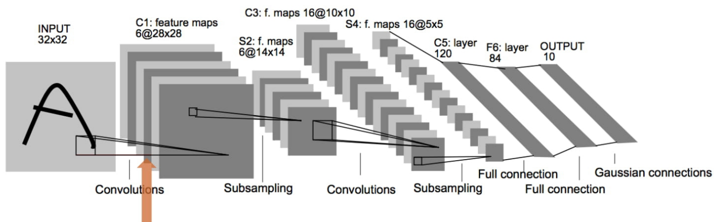


# LeNet—Structure Diagram



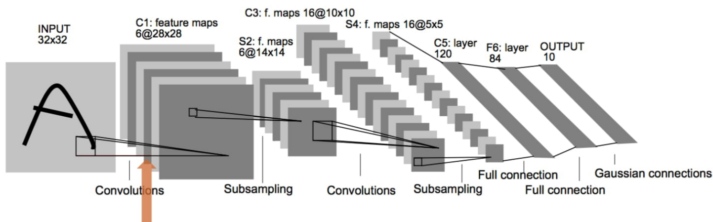

This means the resulting "filter" has dimension 28x28. (Why?)

# LeNet—Structure Diagram




**They use a depth of 6. This means there are 6 different kernels that are learned.**

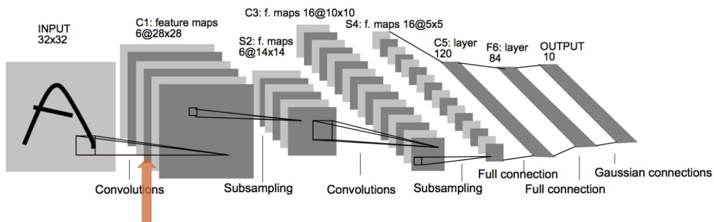
# LeNet—Structure Diagram



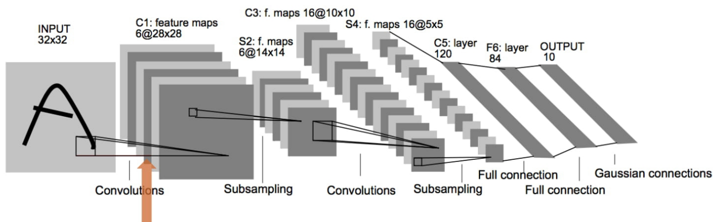

They use a depth of 6. This means there are 6 different kernels that are learned.

So the output of this layer is 6x28x28.

# LeNet—Structure Diagram



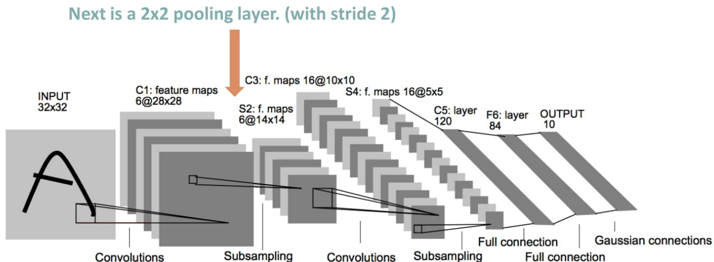
# LeNet—Structure Diagram



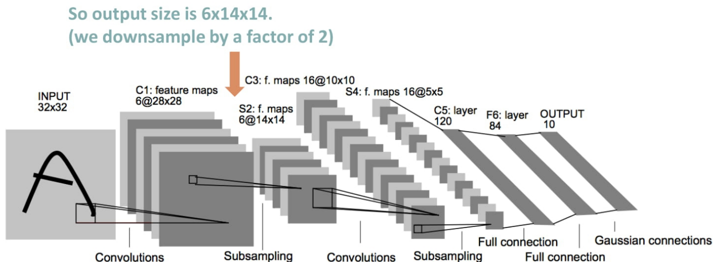

What is the total number of weights in this layer?

Answer: Each kernel has 5x5=25 weights (plus a bias term, so actually 26 weights). So total weights = 6x26 = 156.

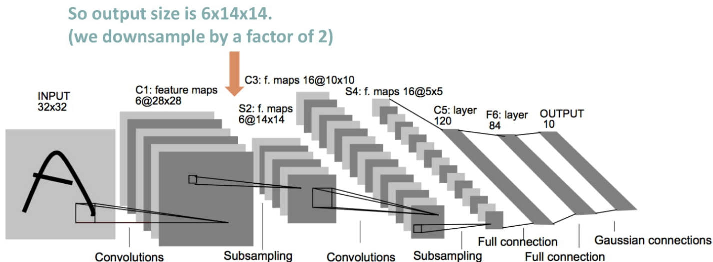
# LeNet—Structure Diagram



# LeNet—Structure Diagram

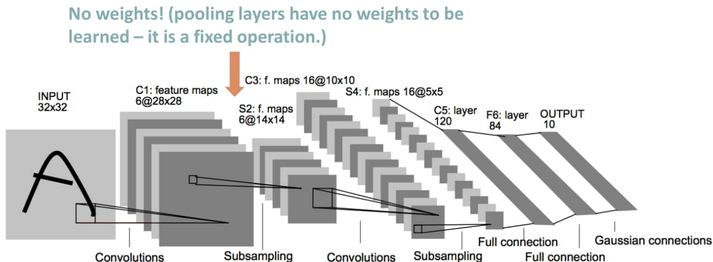


# LeNet—Structure Diagram

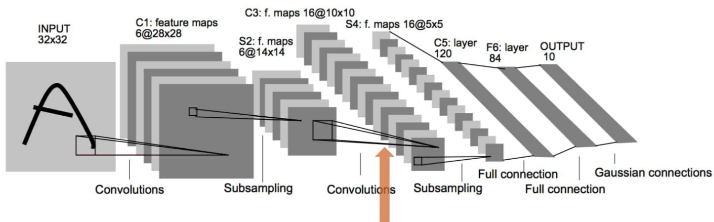



**Note: The original paper actually does a more complicated pooling then max or avg. pooling, but this is considered obsolete now.**

# LeNet—Structure Diagram

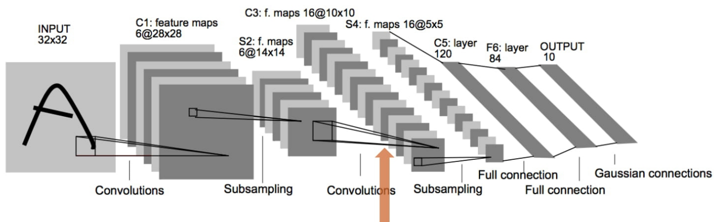


# LeNet—Structure Diagram



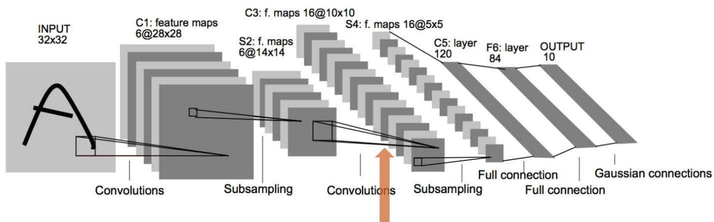

Another 5x5 convolutional layer with stride 1. This time the depth is 16.

# LeNet—Structure Diagram



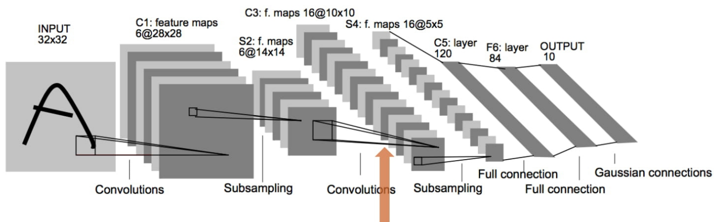

**Output size: 16 x 10 x 10 How many weights? (tricky!)**

# LeNet—Structure Diagram



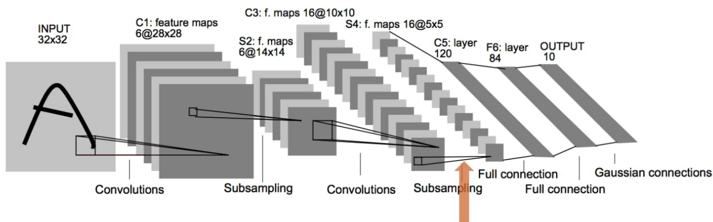

**The kernels "take in" the full depth of the previous layer. So each 5x5 kernel now "looks at" 6x5x5 pixels.**
**Each kernel has 6x5x5 = 150 weights + bias term = 151.**

# LeNet—Structure Diagram



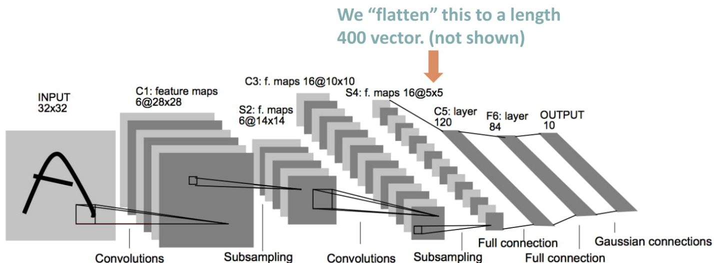

So, total weights for this layer = 16*151 = 2416.

# LeNet—Structure Diagram

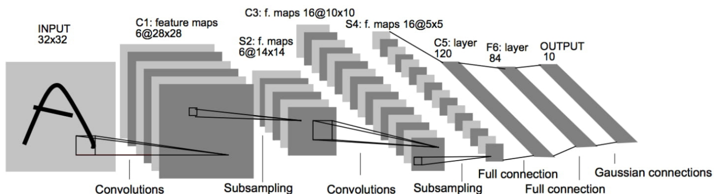



Another 2x2 pooling layer.
Output is 16 x 5 x 5.

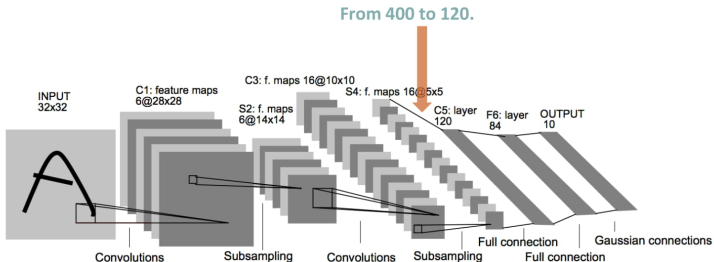
# LeNet—Structure Diagram



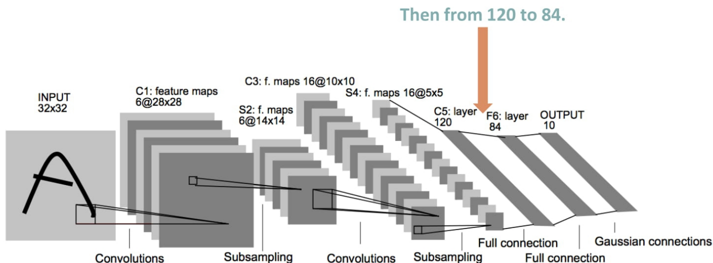
# LeNet—Structure Diagram



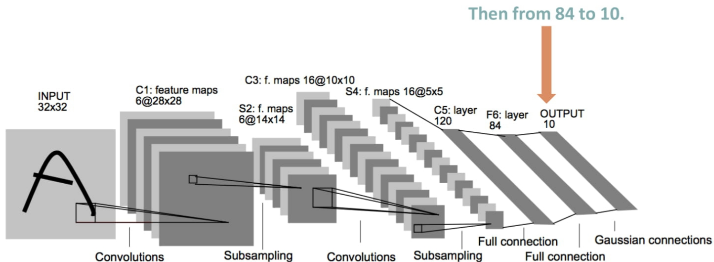
# LeNet—Structure Diagram



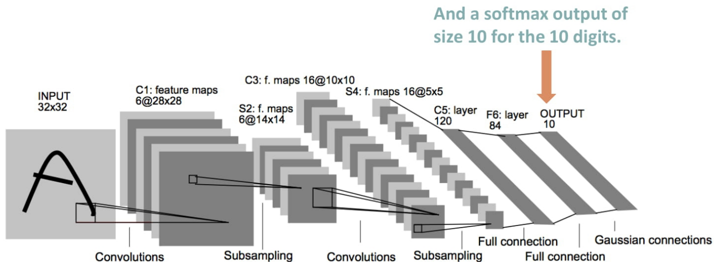
# LeNet—Structure Diagram



# LeNet—Structure Diagram



# LeNet—Structure Diagram



# LeNet-5

## How many total weights in the network?

| | | |
|---|---|---|
| Conv1: 1*6*5*5 + 6 | = | 156 |
| Conv3: 6*16*5*5 + 16 | = | 2416 |
| FC1: 400*120 + 120 | = | 48120 |
| FC2: 120*84 + 84 | = | 10164 |
| FC3: 84*10 + 10 | = | 850 |
| **Total:** | **=** | **61706** |

**Less than a single FC layer with [1200x1200] weights!**

**Note that Convolutional Layers have relatively few weights.**

# Outline

1 Regularization in Neural Networks

2 Introduction to Convolution Neural Networks

3 CNN Architecture Example

4 Transfer Learning

5 CNN Architectures

# What is Transfer Learning?

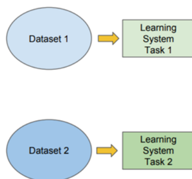
## Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is re-used on a second related task.

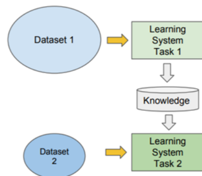
- Only works in deep learning if the model features learned from the first task are general.

**Traditional ML** vs **Transfer Learning**

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks




- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data

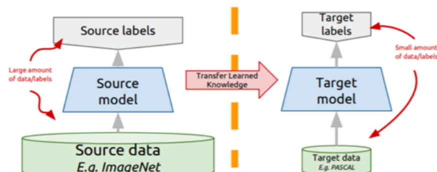


# Transfer Learning Idea

- Take a network trained on different domain for a different Source Task.
- Adapt it to for your domain and Target Task

**Variations**

- Same domain, differnet task.
- Different domain, same task.



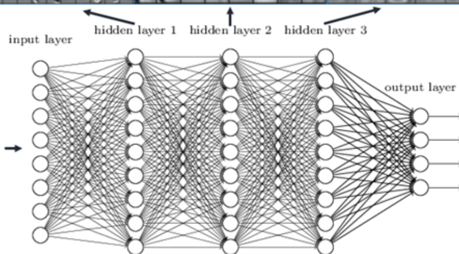
# Why Transfer Learning?

- Alternative is to build model from scratch
  - Time cosuming
  - Hard feature engineering process.
- Can Lessen the data demands.
- Transfer Learning can be used for privacy.
- Can be used to improve a model performance.

# How does it work?

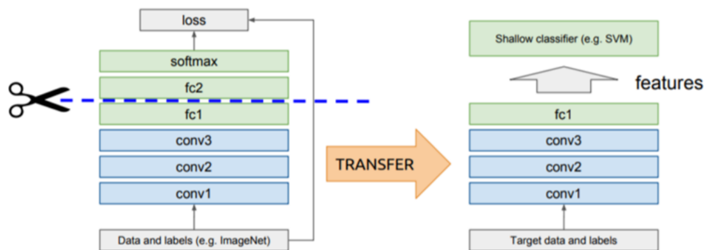
- Deep learning systems and models are layered architectures that learn different features at different layers.
- The initial layers have been seen to capture generic features, while the later ones focus more on the specific task at hand.

Deep neural networks learn hierarchical feature representations



# How does it work?

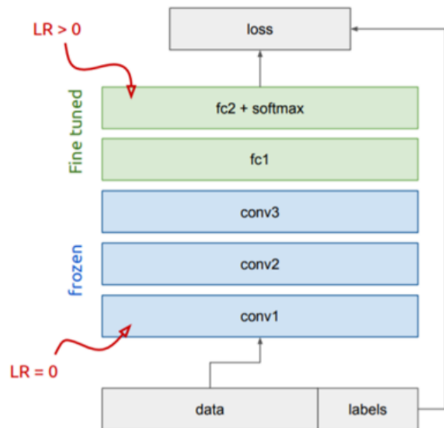
- The key idea is to leverage the pre-trained models weighted layers to extract features but not to update the weights of the models layers during training with new data for the new task.



# Freeze or Fine-tune

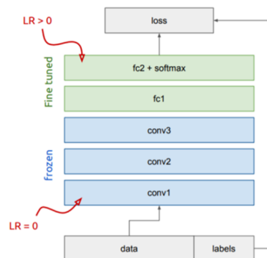
- **Freezed layers:** not updated during backprop.
- **Fine-tuned layer:** updated during backprop.

In general, we can set learning rates to be different for each layer. Our goal is retraining the head of a network to recognize classes it was not originally intended for.



# Freeze and Fine-tune

1. If you have very small dataset you may freeze all layers except softmax in the source model and replace it and train only softmax of target model.
2. If you have small dataset you may freeze large number of layers in the source model while train low number of layers.
3. If you have large dataset you may freeze low number of layers while train more number of layers.



# Guiding Principles for Fine-Tuning

**While there are no "hard and fast" rules, there are some guiding principles to keep in mind.**

**1)** The more similar your data and problem are to the source data of the pre-trained network, the less fine-tuning is necessary

**E.g.** *Using a network trained on ImageNet to distinguish "dogs" from "cats" should need relatively little fine-tuning. It already distinguished different breeds of dogs and cats, so likely has all the features you will need.*

# Guiding Principles for Fine-Tuning

**2)** The more data you have about your specific problem, the more the network will benefit from longer and deeper fine-tuning

**E.g.** *If you have only 100 dogs and 100 cats in your training data, you probably want to do very little fine-tuning. If you have 10,000 dogs and 10,000 cats you may get more value from longer and deeper fine-tuning.*

# Guiding Principles for Fine-Tuning

**3)** If your data is substantially different in nature than the data the source model was trained on, Transfer Learning may be of little value

**E.g.** *A network that was trained on recognizing typed Latin alphabet characters would not be useful in distinguishing cats from dogs. But it likely would be useful as a starting point for recognizing Cyrillic Alphabet characters.*

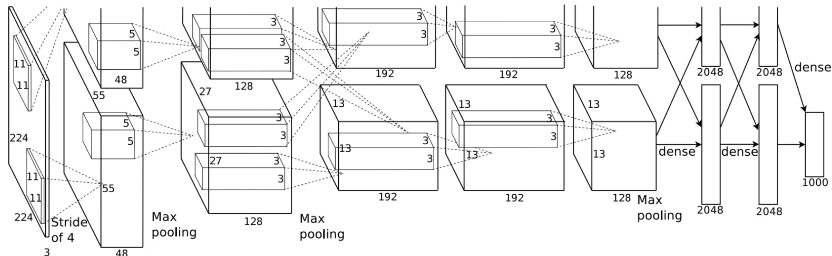
# Outline


1. Regularization in Neural Networks
2. Introduction to Convolution Neural Networks
3. CNN Architecture Example
4. Transfer Learning
5. CNN Architectures

# ALEXNET

- Created in 2012 for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- Task: predict the correct label from among 1000 classes
- Dataset: around 1.2 million images
- Considered the “flash point” for modern deep learning
- Demolished the competition
- Top 5 error rate of 15.4%
- Next best: 26.2%

# ALEXNET—MODEL DIAGRAM



# ALEXNET—DETAILS

- They performed *data augmentation* for training
- Includes cropping, horizontal flipping, and other manipulations

# ALEXNET—DETAILS

- They performed data augmentation for training
  - Cropping, horizontal flipping, and other manipulations
- Basic Template:
  - Convolutions with ReLUs
  - Sometimes add maxpool after convolutional layer
  - Fully connected layers at the end before a softmax classifier

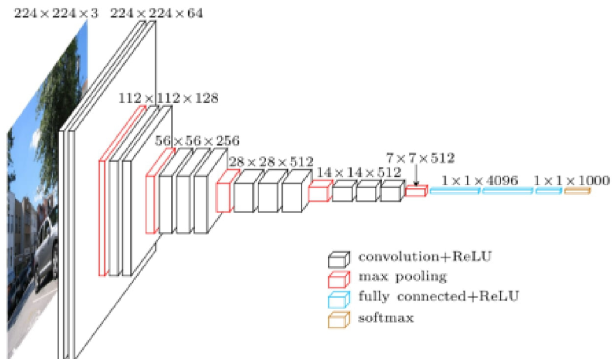
# VGG

- Simplify Network Structure
- Avoid Manual Choices of Convolution Size
- Very Deep Network with 3x3 Convolutions
- These “effectively” give rise to larger convolutions

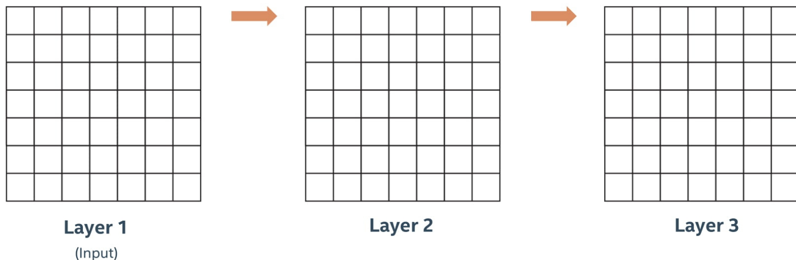
**Reference:** *Very Deep Convolutional Networks for Large-Scale Image Recognition*

Karen Simonyan and Andrew Zisserman, 2014

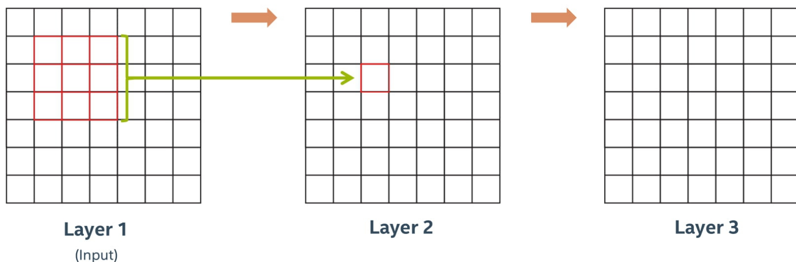
# VGG16 DIAGRAM



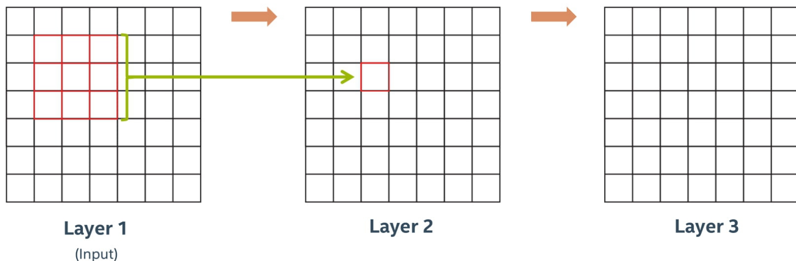
# VGG



# VGG

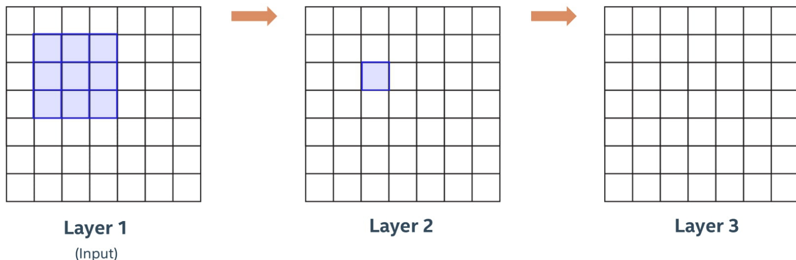


# VGG



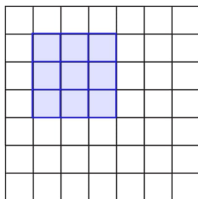
# VGG

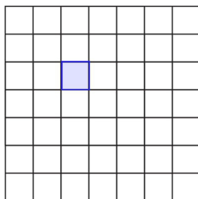
We can say that the "receptive field" of layer 2 is 3x3.
Each output has been influenced by a 3x3 patch of inputs.

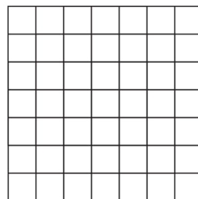


# VGG

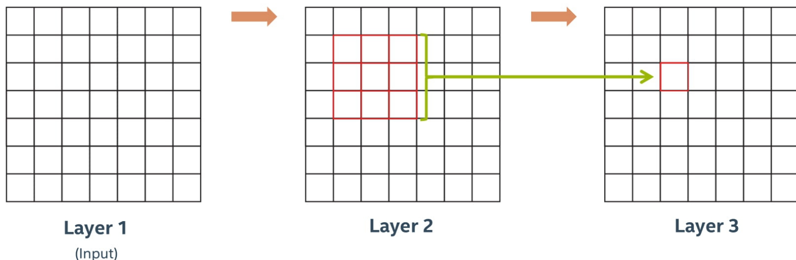
What about on layer 3?



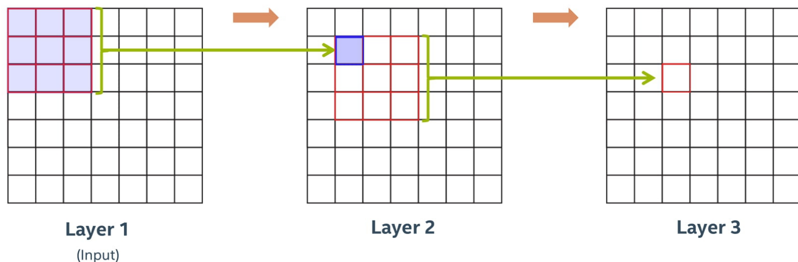




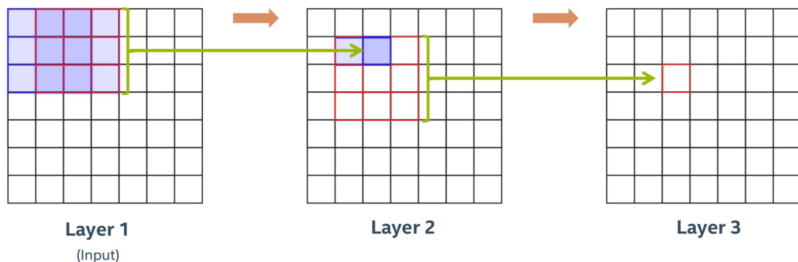
# VGG



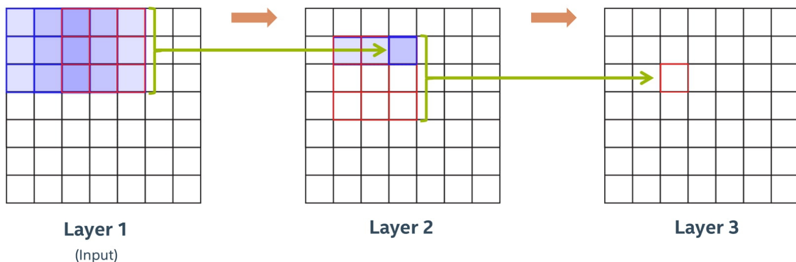
# VGG



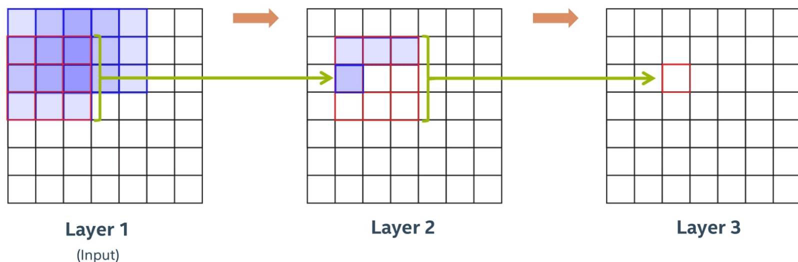
# VGG



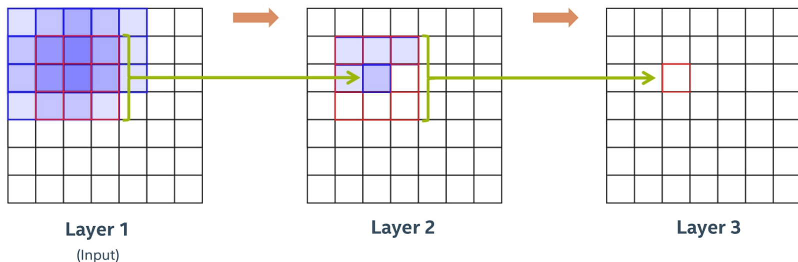
# VGG



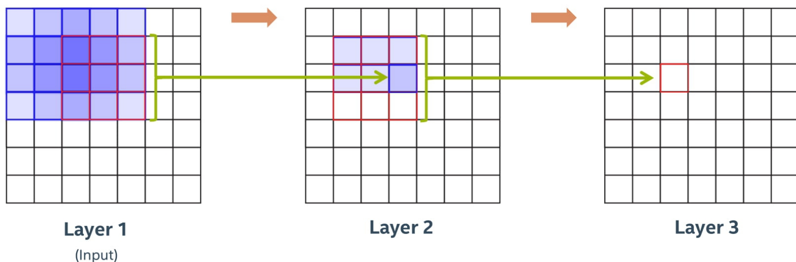
# VGG



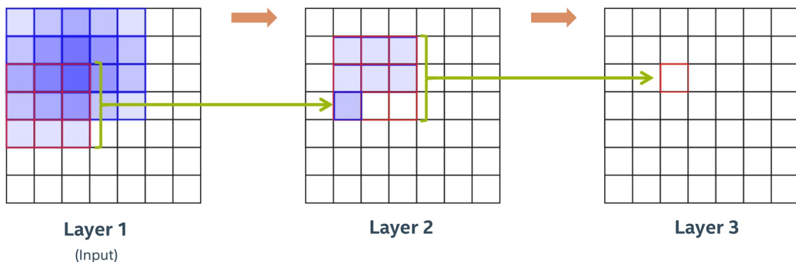
# VGG



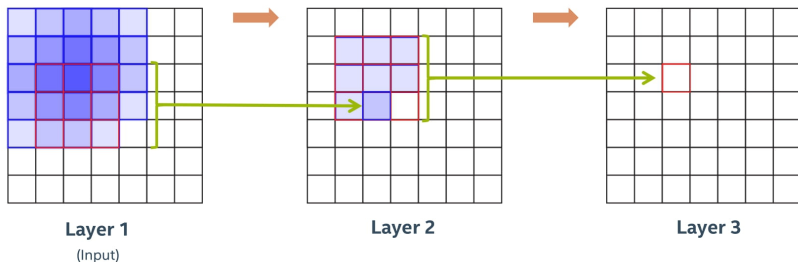
# VGG



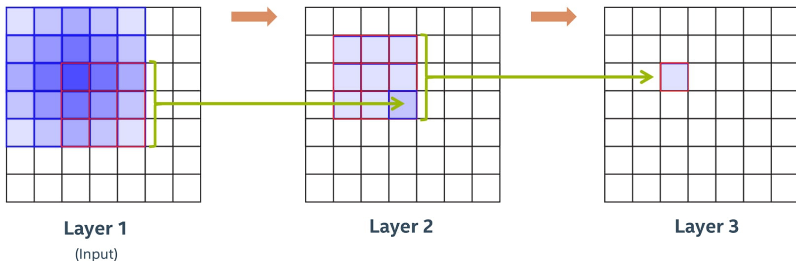
# VGG



# VGG

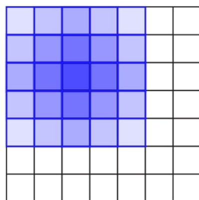


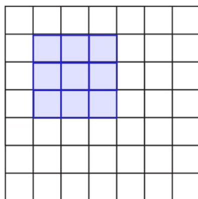
# VGG

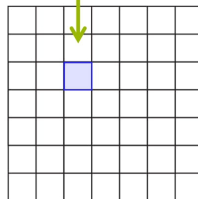


# VGG

Each square in layer 3 “sees” a 5x5 grid from layer 1.







# VGG

**Two 3x3, stride 1 convolutions in a row → one 5x5.**

**Three 3x3 convolutions → one 7x7 convolution.**

**Benefit: fewer parameters.**

One 3x3 layer

$$3 \times 3 \times C \times C = 9C^2$$

One 7x7 layer

$$7 \times 7 \times C \times C = 49C^2$$

Three 3x3 layers

$$3 \times (9C^2) = 27C^2$$

$49C^2$ → $27C^2$ → ≈45% reduction!

# VGG

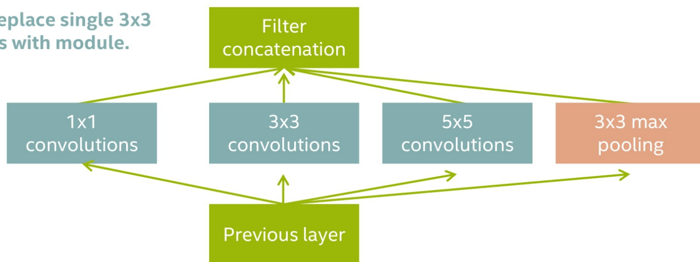
- One of the first architectures to experiment with many layers (More is better!)
- Can use multiple 3x3 convolutions to simulate larger kernels with fewer parameters
- Served as ”base model” for future works

# INCEPTION

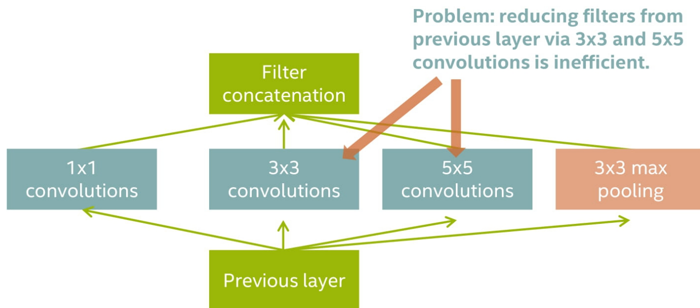
- Szegedy et al 2014
- Idea: network would want to use different receptive fields
- Want computational efficiency
- Also want to have sparse activations of groups of neurons
- Hebbian principle: “Fire together, wire together”
- Solution: Turn each layer into branches of convolutions
- Each branch handles smaller portion of workload
- Concatenate different branches at the end

# INCEPTION

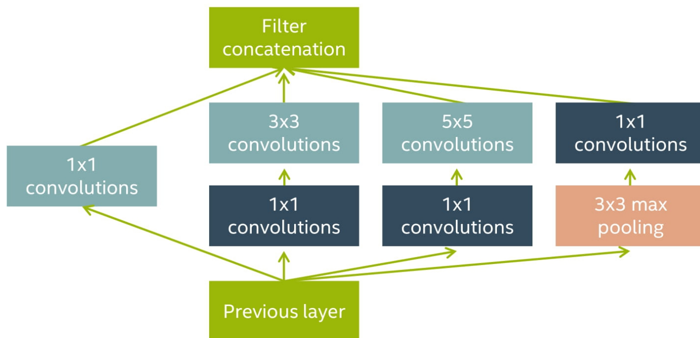
Basic idea: replace single 3x3 convolutions with module.



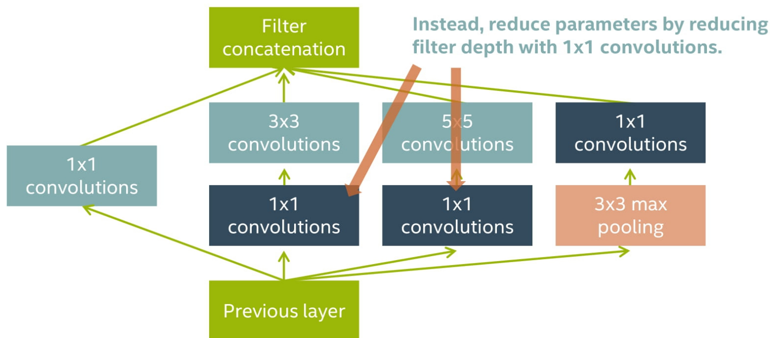
# INCEPTION



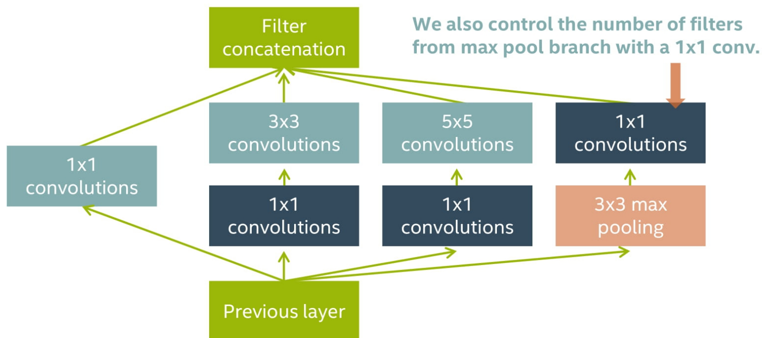
# INCEPTION



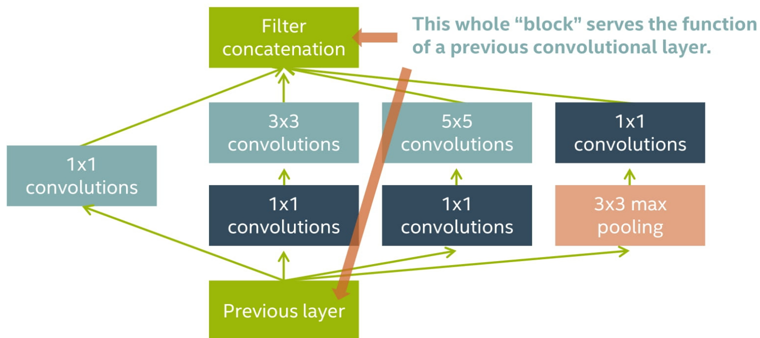
# INCEPTION



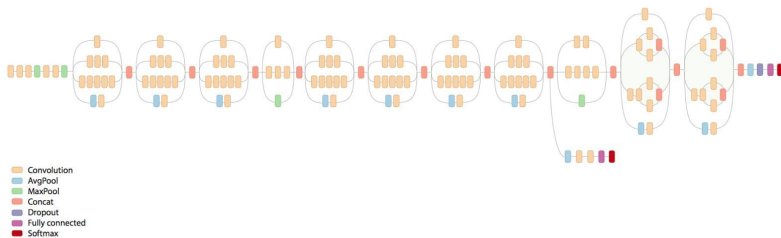
# INCEPTION



# INCEPTION

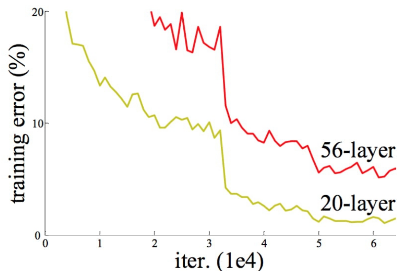


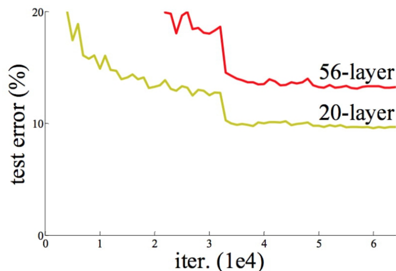
# INCEPTION V3 SCHEMATIC



# RESNET—MOTIVATION

**Issue: Deeper Networks performing worse on training data!** (as well as test data)



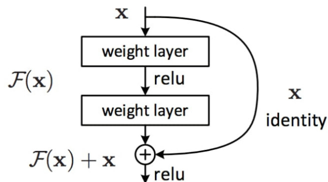



# RESNET

- Surprising because deeper networks should overfit more
- So what's happening?
- Early layers of Deep Networks are very slow to adjust
- Analogous to "Vanishing Gradient" issue
- In theory, should be able to just have an "identity" transformation that makes the deeper network behave like a shallower one

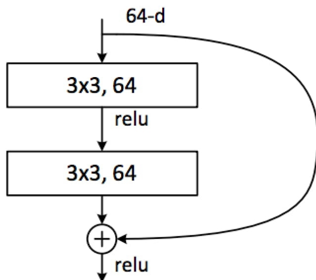
# RESNET

- Assumption: best transformation over multiple layers is close to $\mathcal{F}(x)+x$
- $x$ → input to series of layers
- $\mathcal{F}(x)$ → function represented by several layers (such as convs)
- Enforce this by adding "shortcut connections"
- Add the inputs from an earlier layer to the output of current layer



# RESNET

- Add previous layer back in to current layer!
- Similar idea to “boosting”



Thank You!

Questions