

Convolutional Neural Networks

Outline



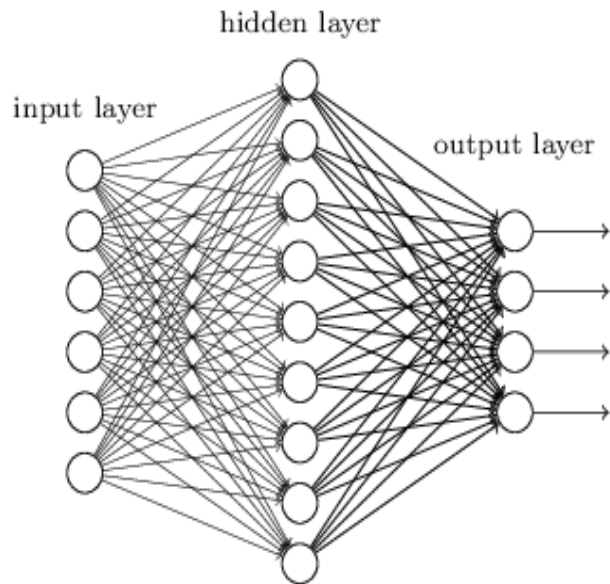
- Deep Neural Network
- Image to Vector
- Convolutional Neural Networks
- CNN Architectures
- Data Augmentation
- Transfer Learning
- Summary

Deep Neural Networks

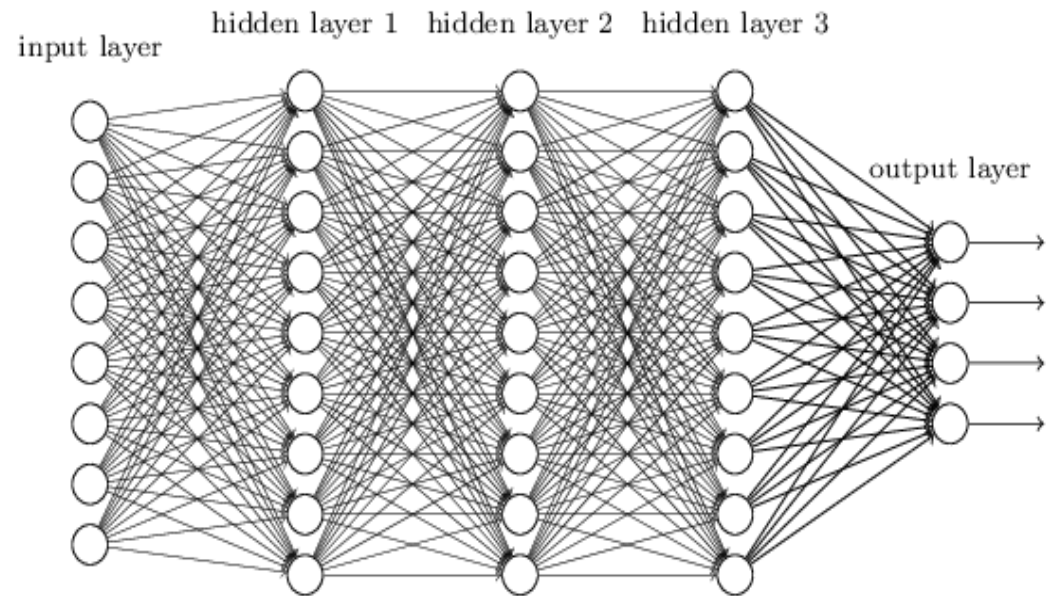
- Deep-learning networks are distinguished from the more commonplace single-hidden-layer neural networks by their **depth**.
- A deep neural network is a neural network with a certain level of complexity, a neural network with more than two layers.
- In deep-learning networks, each layer of nodes trains on a distinct set of features based on the previous layer's output.
- The further you advance into the neural net, the more complex the features your nodes can recognize, since they aggregate and recombine features from the previous layer.

Deep Neural Networks

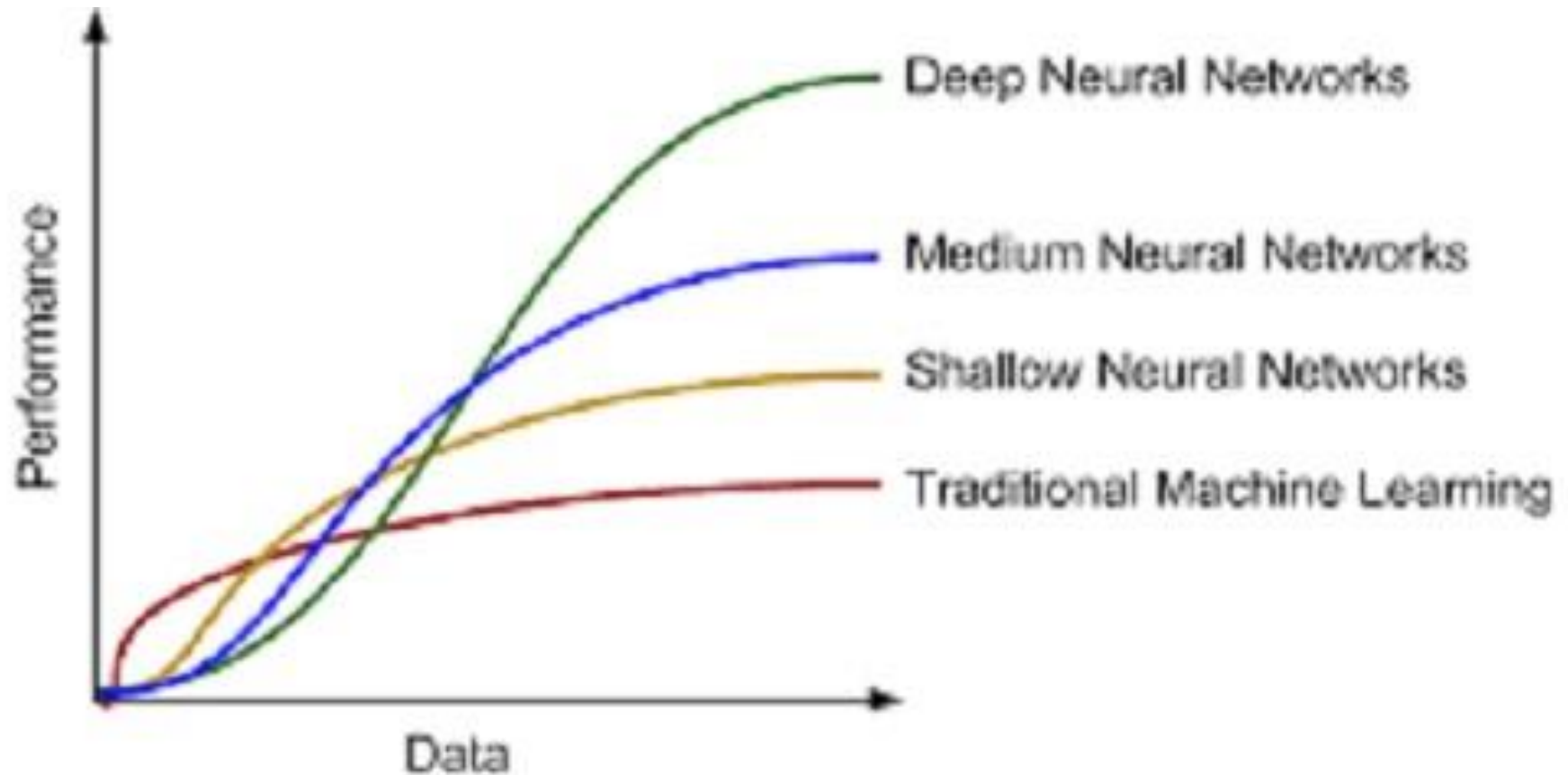
"Non-deep" feedforward
neural network



Deep neural network

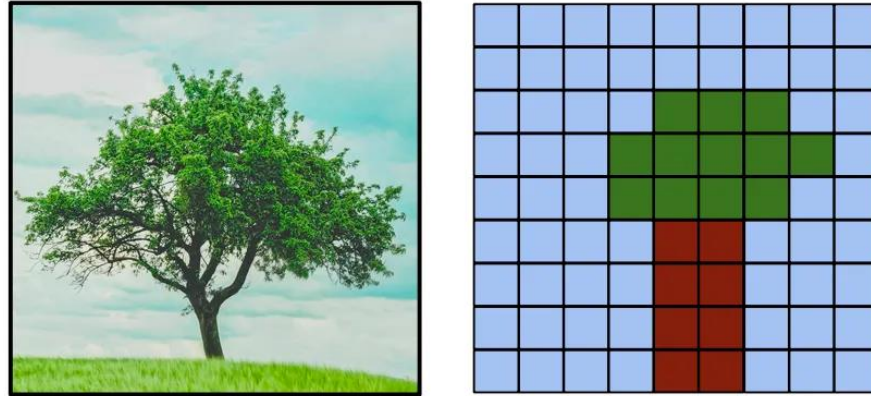


Performance of Network Size

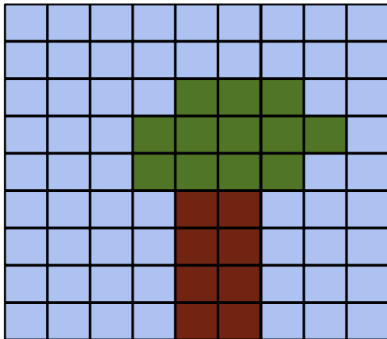


Deep Neural Networks for Images

□ Image to vector



Tree image. On the left is the original digital image, on the right is the simplified pixelated tree image

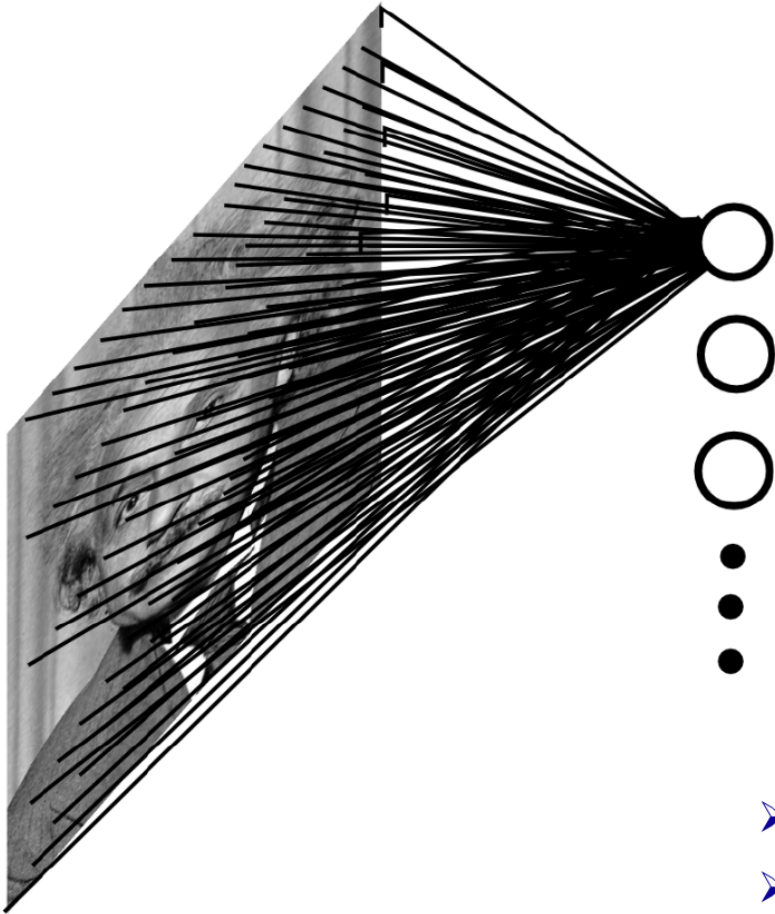


A digital image is a **2D grid of pixels**.

A neural network expects a **vector of numbers** as input.

Deep Neural Networks for images

- Example: 200x200 image, 40K hidden units, **~2B parameters!**



- Spatial correlation is local
- Too many parameters, will require a lot of training data!

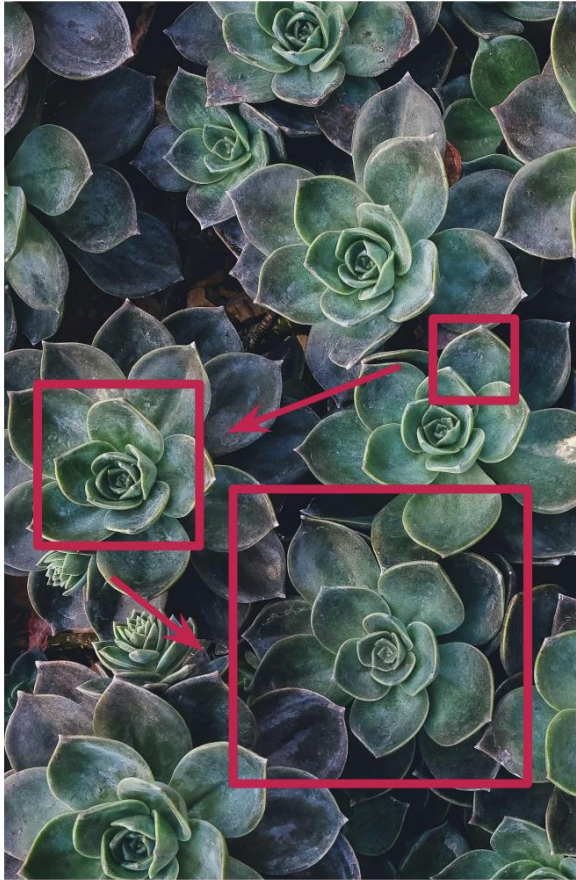
Locality and translation invariance



Locality: nearby pixels are more strongly correlated

Translation invariance: meaningful patterns can occur anywhere in the image

Topological Structure



Weight sharing: use the same network parameters to detect local patterns at many locations in the image

Hierarchy: local low-level features are composed into larger, more abstract features



edges and textures



object parts

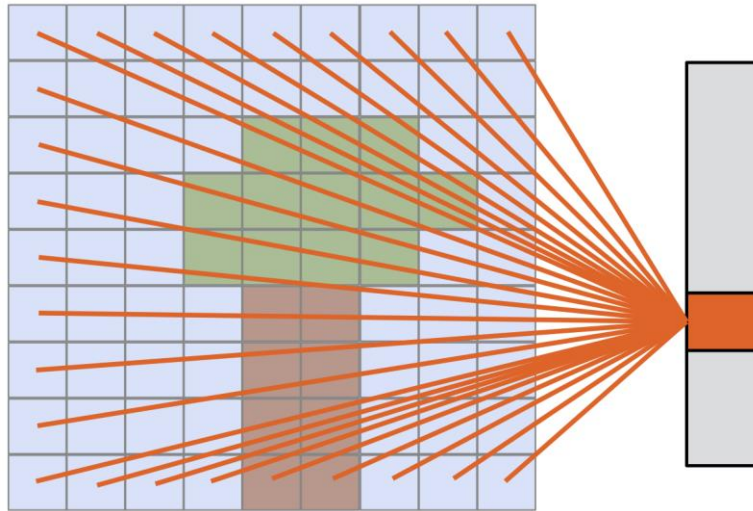


objects

Drawback of using Fully Connected Deep NN

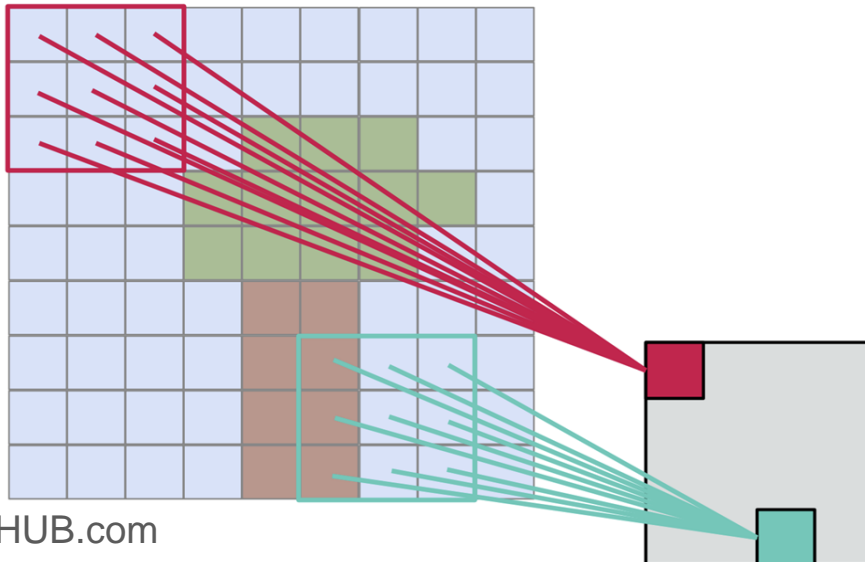
- ❑ **Lack of Spatial Information Preservation**
 - ❑ Fully connected networks treat the input data as a flat vector, ignoring the spatial relationships between pixels in an image.
- ❑ **Lack of Feature Hierarchy:**
 - ❑ Fully connected networks do not have the capability learn hierarchical features.
 - ❑ Fully connected networks require a significant amount of data to learn effective feature representations from scratch.
- ❑ **High Dimensionality**
 - ❑ Images are high-dimensional data, and fully connected layers in deep networks require a massive number of parameters.
 - ❑ This high dimensionality can lead to overfitting, especially when dealing with limited training data.
- ❑ **Computational Intensity**
 - ❑ The sheer number of parameters in a fully connected network makes it computationally intensive to train and deploy. Training large fully connected networks may require significant computational resources and time.
- ❑ **Lack of Translation Invariance**
 - ❑ Fully connected networks are not inherently designed to capture translation-invariant features.

From fully connected to locally connected



fully-connected unit

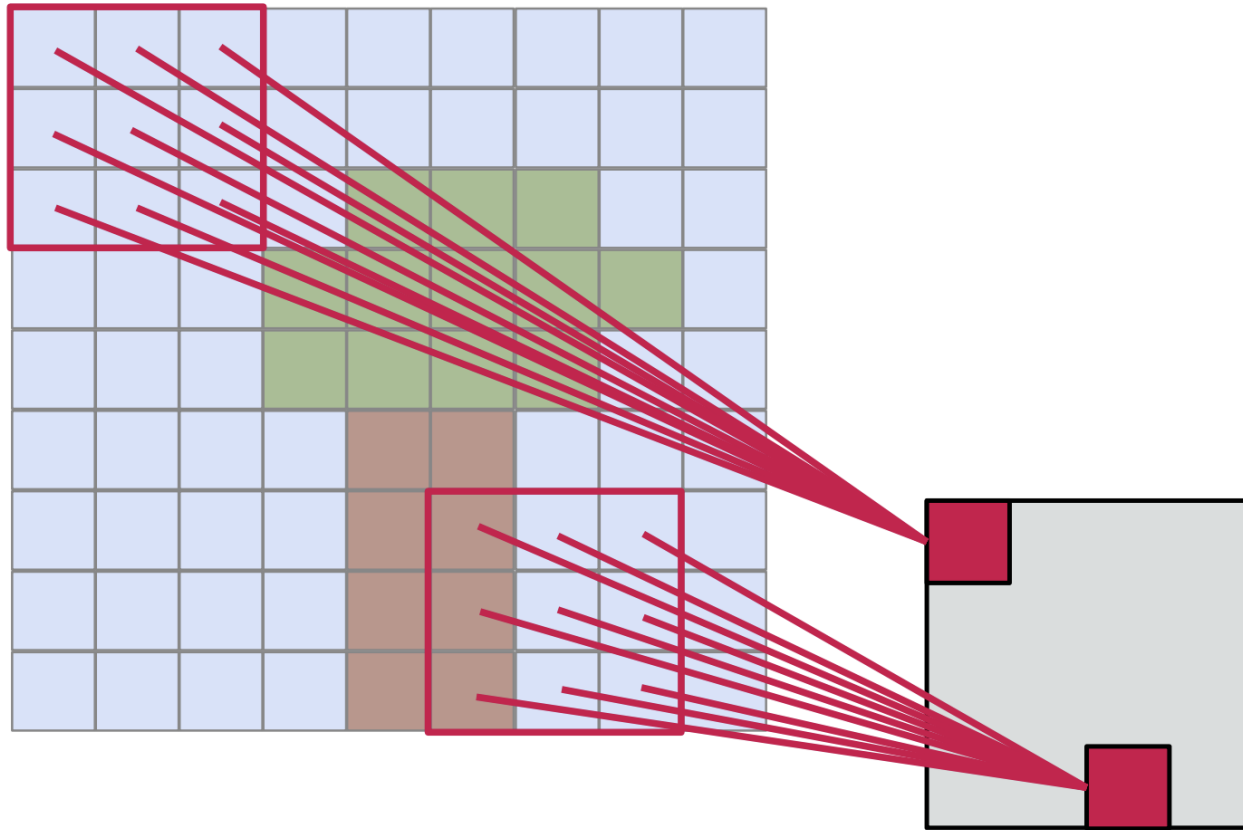
$$y = \sum_{i \in \text{image}} \mathbf{w}_i \mathbf{x}_i + b$$



$$y = \sum_{i \in 3 \times 3} \mathbf{w}_i \mathbf{x}_i + b$$

locally-connected units
3X3 receptive field

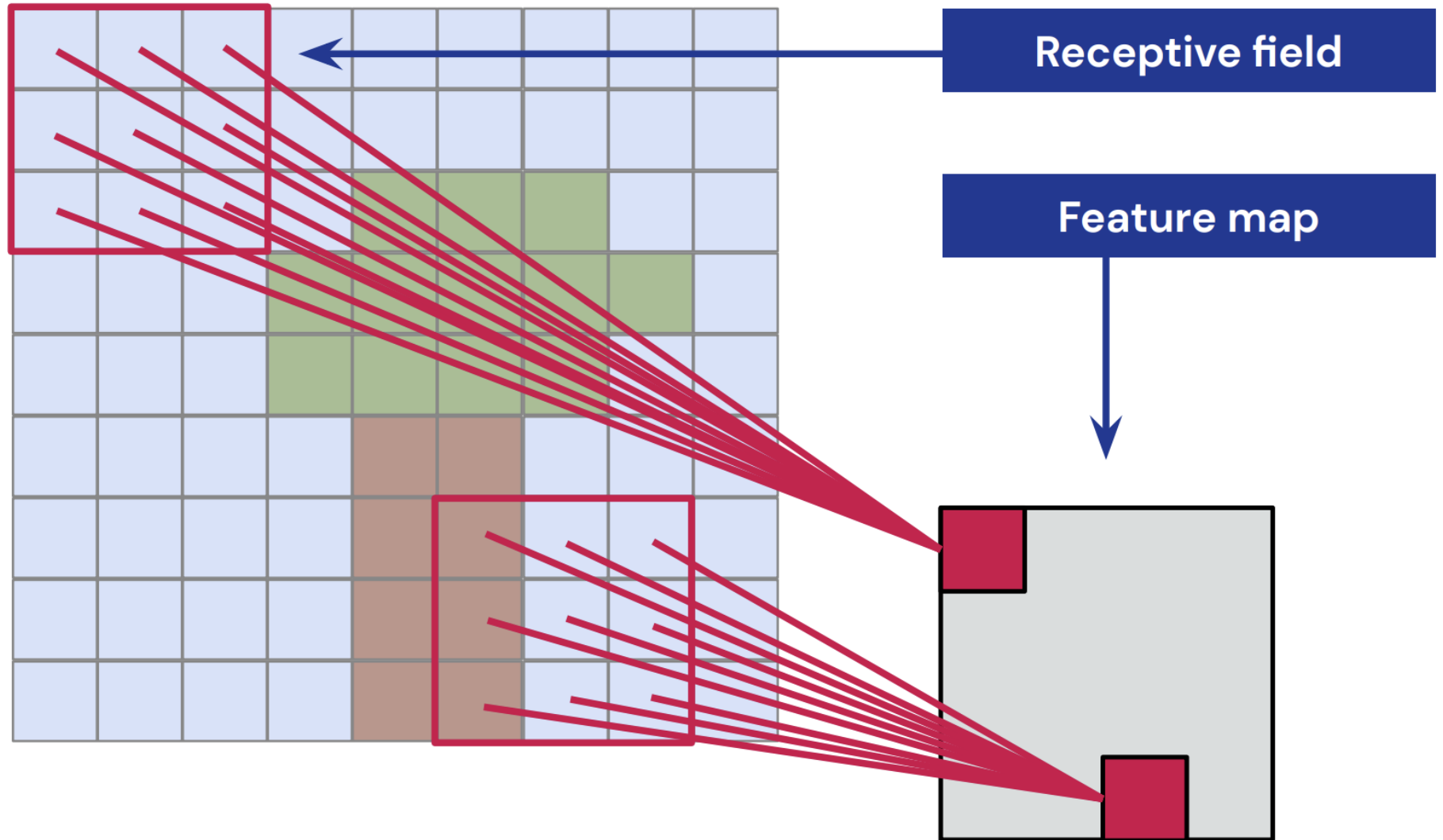
From locally connected to convolutional



$$y = \mathbf{w} * \mathbf{x} + b$$

convolutional units
3×3 receptive field

From locally connected to convolutional



Implementation: the convolution operation

1	0	1
0	1	0
1	0	1

Weight
Filter

Terminology!: Also
referred to as a
"kernel"

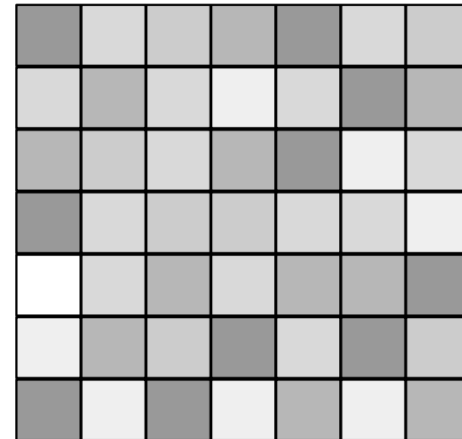
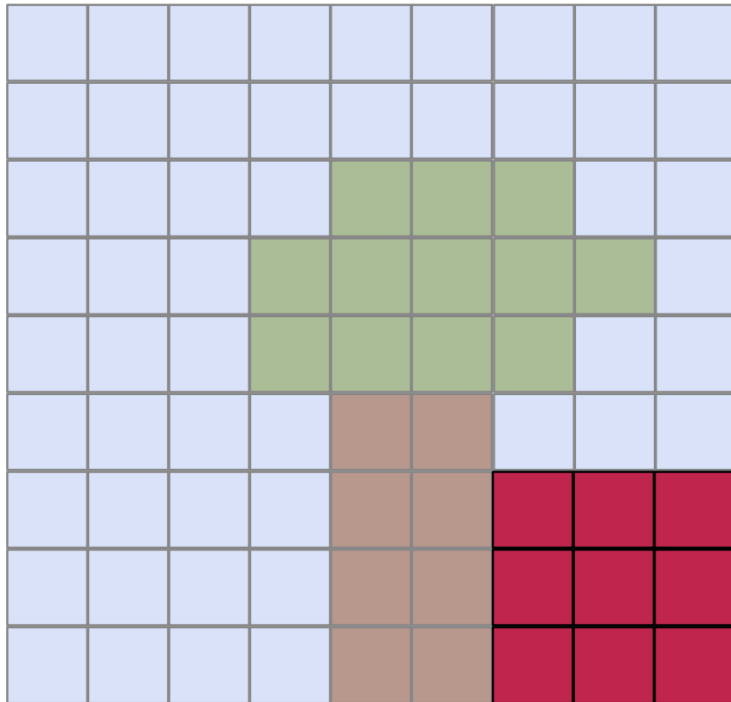
1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

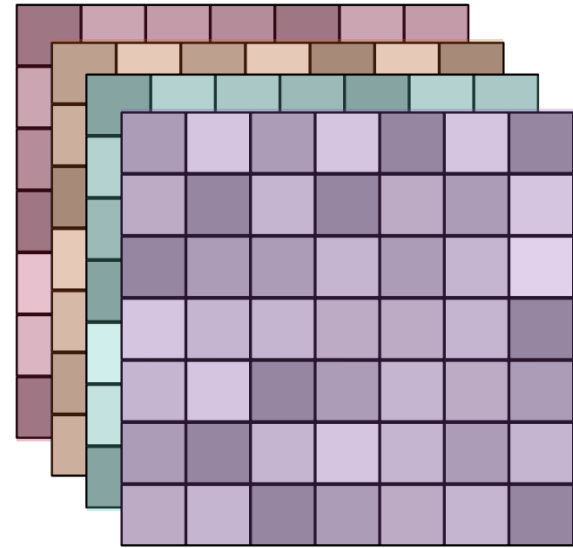
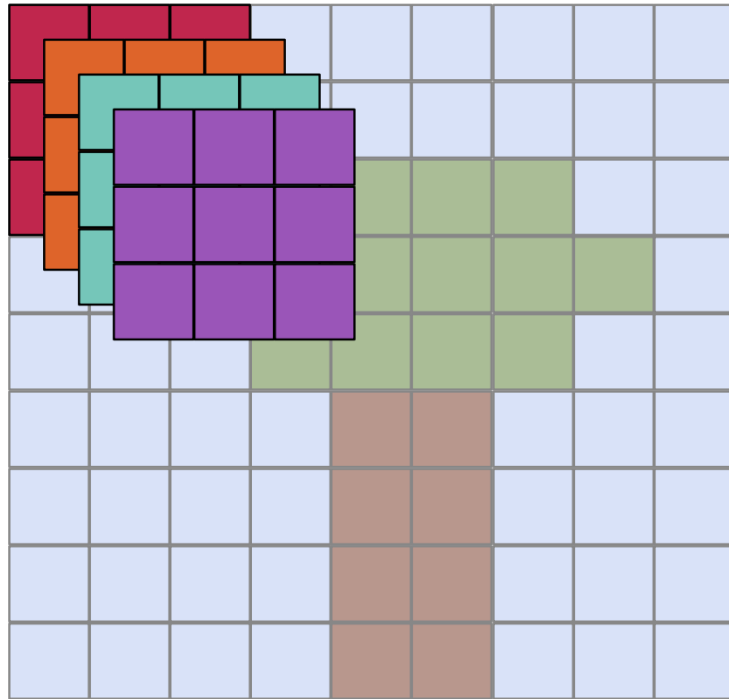
Convolved
Feature

Implementation: the convolution operation



The **kernel** slides across the image and produces an output value at each position

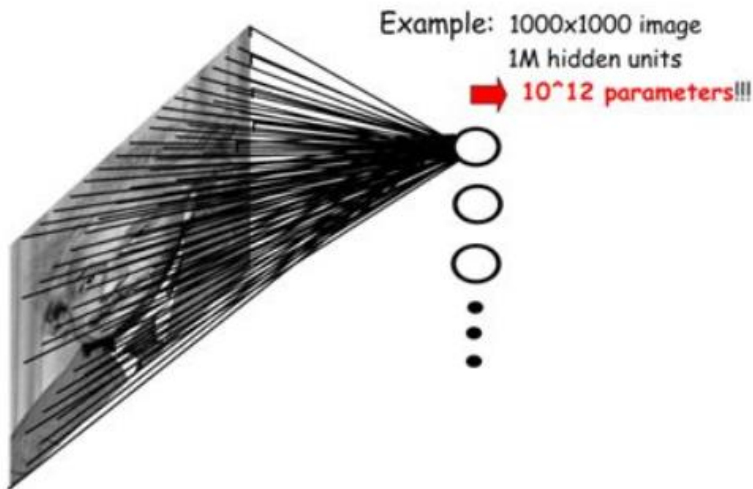
Implementation: the convolution operation



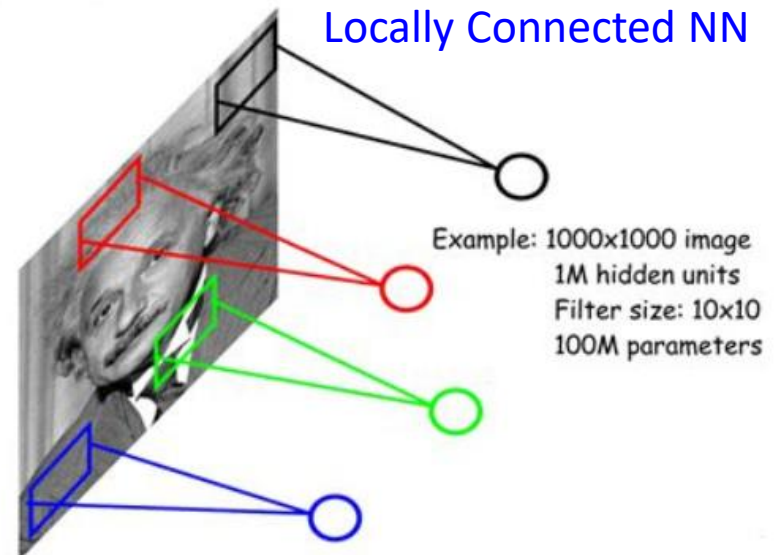
We convolve multiple kernels and obtain multiple feature maps or **channels**

From locally connected to convolutional

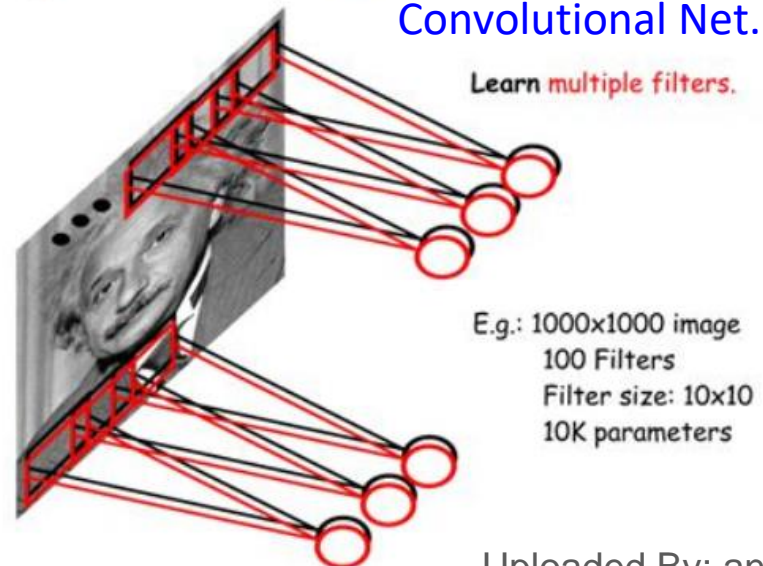
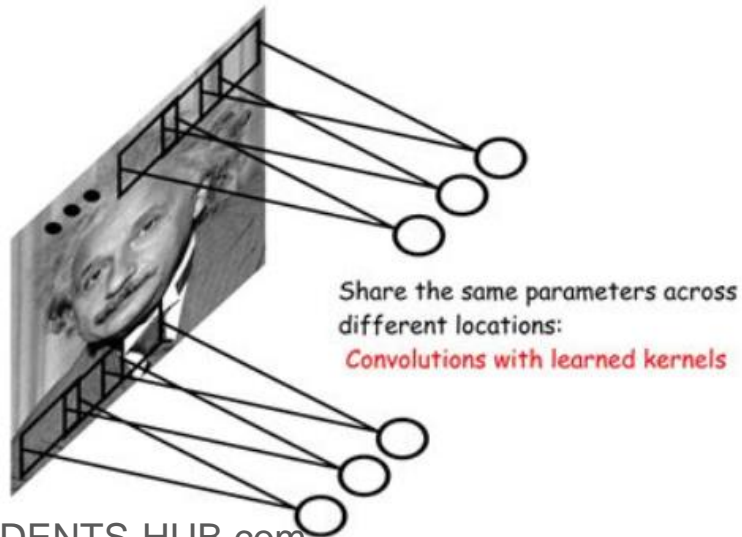
Fully Connected NN



Locally Connected NN

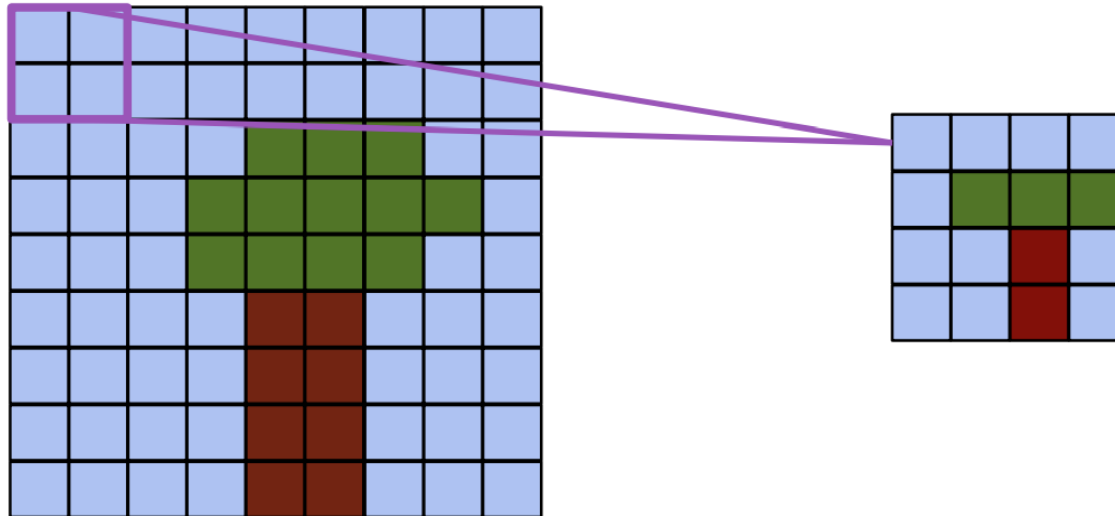


Convolutional Net.



Pooling

- Convolutional layers are typically followed by pooling layers (e.g., max pooling) that down sample feature maps, retaining the most important information while reducing spatial dimensions.
- Pooling helps the network focus on the most salient features and improves translation invariance.



Pooling: compute mean or max over small windows to reduce resolution

From locally connected to convolutional

□ Hierarchical Feature Learning

- CNNs consist of multiple layers, with each layer capturing increasingly abstract and complex features.
- The lower layers detect simple features like edges and textures, while higher layers learn to recognize more complex patterns and object parts.
- This hierarchical feature learning makes CNNs highly effective at representing structured data.

□ Sparse Connectivity

- In a convolutional layer, each neuron (unit) is connected to only a small local region of the input data, as determined by the receptive field size.
- This sparse connectivity reduces the number of computations required and promotes the extraction of localized features.

□ Weight Sharing Across Channels

- In multi-channel data (e.g., color images with RGB channels), convolutional kernels are applied independently to each channel but share weights across channels.
- This allows the network to learn cross-channel relationships and detect features that span multiple channels.

From locally connected to convolutional

□ **Local Receptive Fields**

- Convolutional operations use small, local receptive fields (kernels) to scan the input data.
- This local focus allows the network to capture patterns and features in a localized and translation-invariant manner.
- In the context of images, this means detecting small, local features like edges, corners, and textures.

□ **Parameter Sharing**

- Convolutional layers use parameter sharing, which means the same set of learnable weights (kernel) is applied across the entire input image or feature map.
- This parameter sharing greatly reduces the number of parameters in the network, making it more computationally efficient and reducing the risk of overfitting.

□ **Translation Invariance**

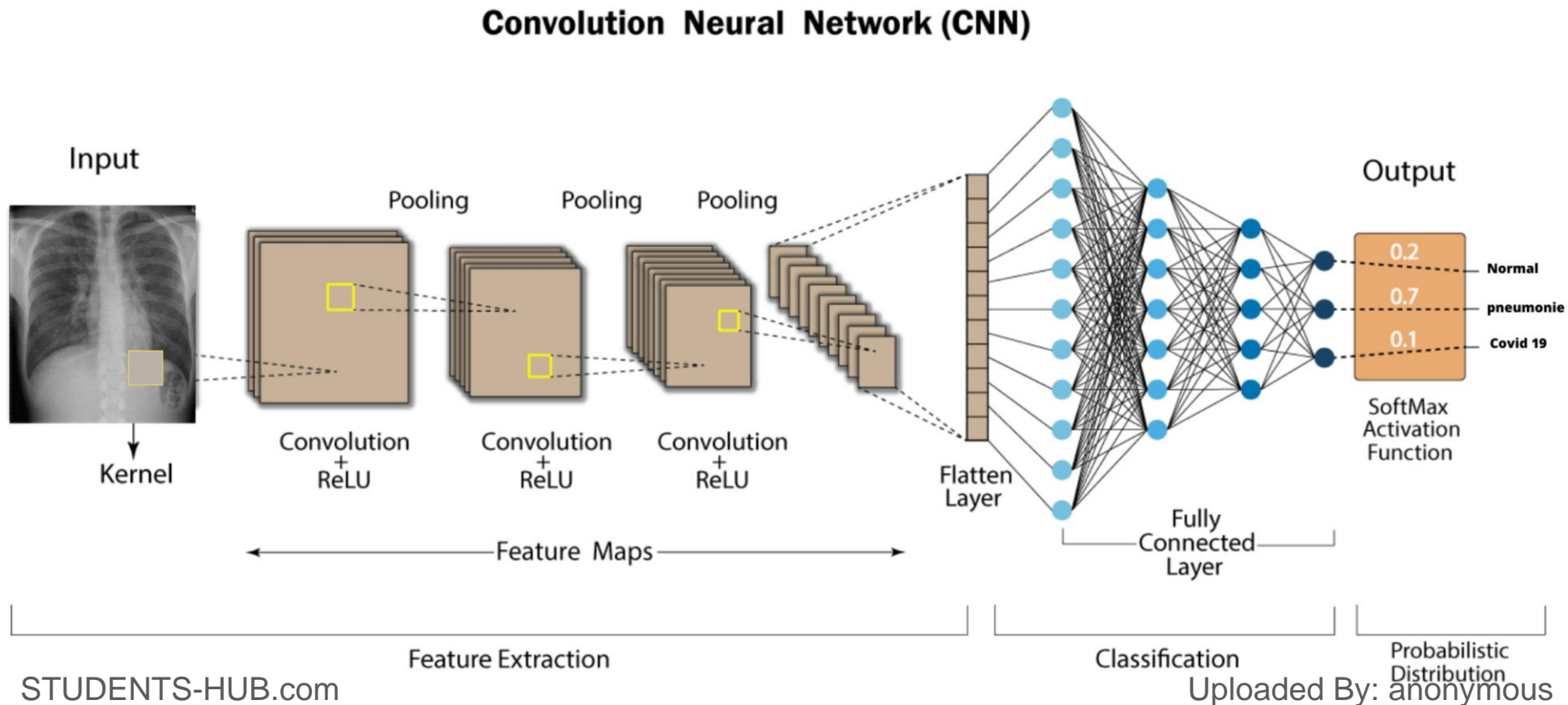
- The use of shared weights and local receptive fields enables CNNs to learn features that are invariant to translation.
- In other words, the network can recognize the same feature regardless of its position in the input data. This property is essential for tasks like object recognition in images.

Convolutional Neural Networks (CNN)

- ❑ CNN is a type of deep neural network that is particularly well-suited for image classification and recognition tasks.
- ❑ CNNs are able to learn complex relationships between the pixels in an image, which is essential for accurately classifying complex images.
- ❑ CNNs work by using a series of convolutional layers and pooling layers.
 - ❑ Convolutional layers learn to extract features from the input image, such as edges, corners, and shapes.
 - ❑ Pooling layers reduce the size of the output of the convolutional layers, and they also help to make the network more robust to noise and variations in the input image.
- ❑ Once the CNN has learned to extract features from the input image, it uses a fully connected layer to classify the image.
 - ❑ The fully connected layer takes the output of the pooling layers and combines it into a single output vector.
 - ❑ The output vector is then used to classify the image into one of a set of predefined categories.

CNN: Architecture Overview

- Three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer**







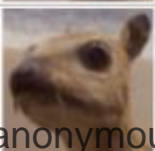


Convolutional Layer

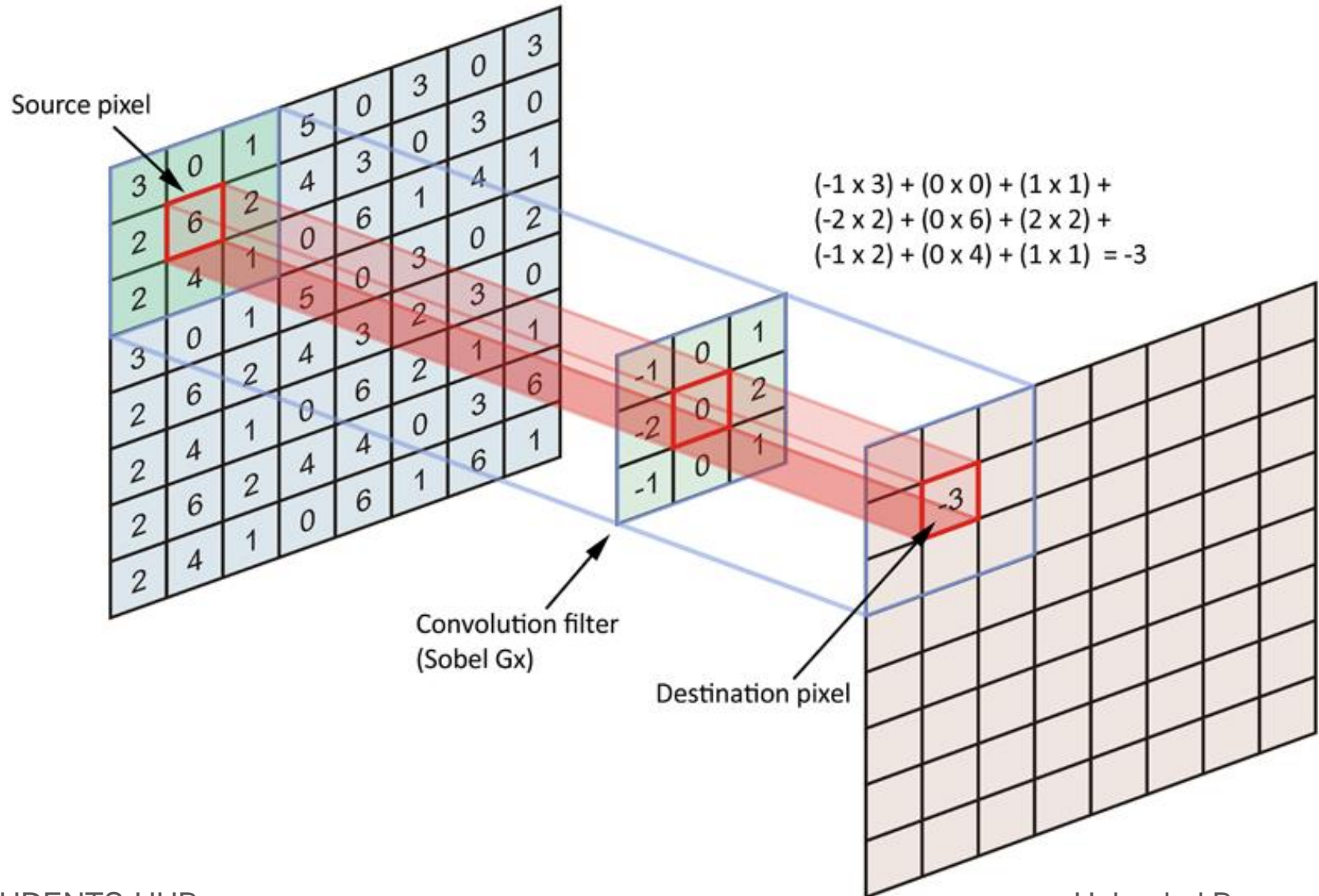
- The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.
- The CONV layer's parameters consist of a set of learnable filters.
 - ▣ Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
 - ▣ For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels).
- 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.
- Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.

Convolutional Layer

- Filters acts as feature detectors from the original input image.
- Different values of the filter will produce different Feature Maps for the same input image.
- The initialization can be random (typically mean zero), or can be based on pre-trained model weights
 - ▣ uniform distribution $[-1/\text{fan-in}, 1/\text{fan-in}]$
 - fan-in: the number of inputs to a hidden unit

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolution operation



Convolution operation

1	0	1
0	1	0
1	0	1

Weight
Filter

Terminology!: Also referred to as a “kernel”

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

Convolution Operation



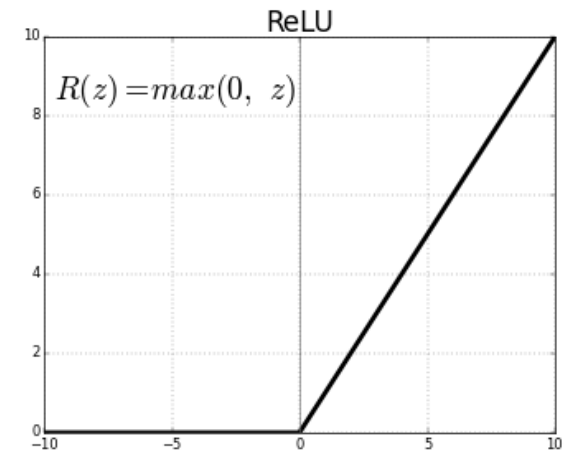
Input

Rectified Linear Unit (ReLU)

- ReLU introduces non-linearity to the model.
 - ▣ The ability to capture non-linear relationships is crucial for the expressiveness of neural networks.
 - ▣ Without non-linear activation functions like ReLU, the entire network would behave like a linear function, limiting its capacity to learn complex patterns and representations.
- By applying a ReLU activation function after a convolution layer, the network can learn to focus on the most important features in the image and ignore the less important features.
- Other non linear functions such as **tanh** or **sigmoid** can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

Rectified Linear Unit

$$f(x) = x^+ = \max(0, x),$$



Input Feature Map



ReLU



Rectified Feature Map



Key Properties of Rectified Linear Unit

□ Computational Efficiency

- ReLU is computationally efficient to compute compared to some other activation functions like sigmoid or tanh.
- The ReLU operation involves a simple thresholding, and it avoids the computational cost associated with exponentials (as in sigmoid and tanh).

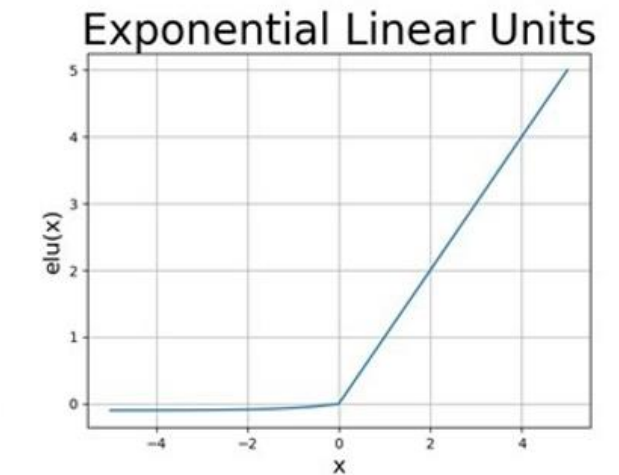
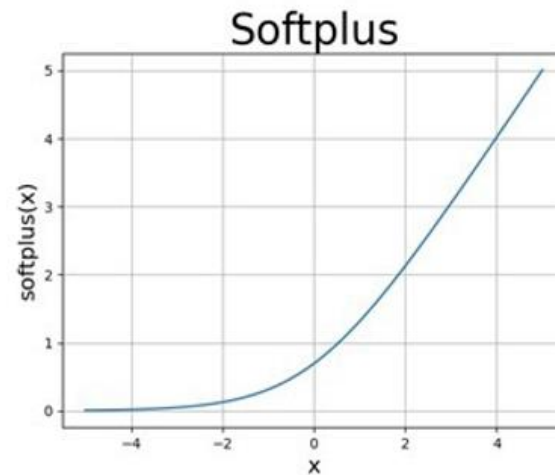
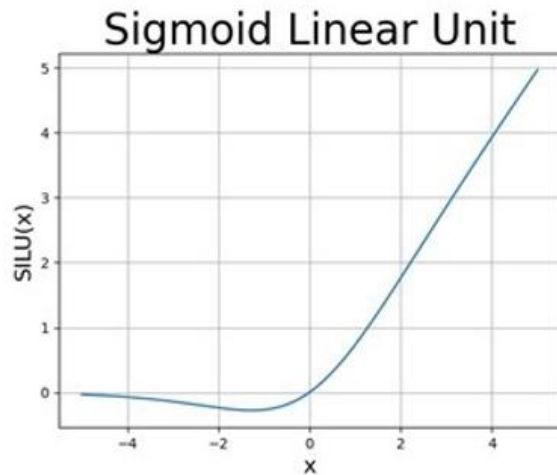
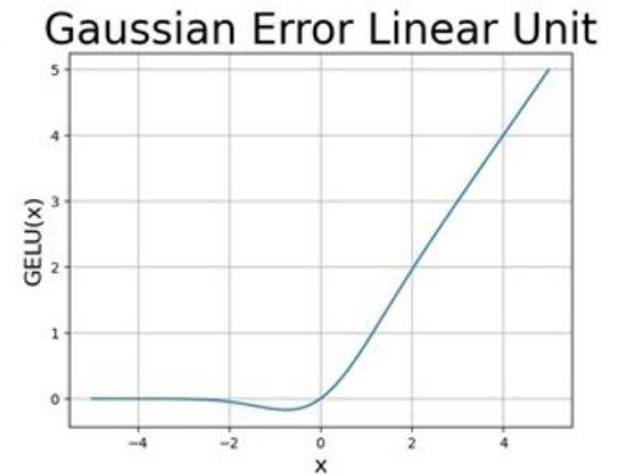
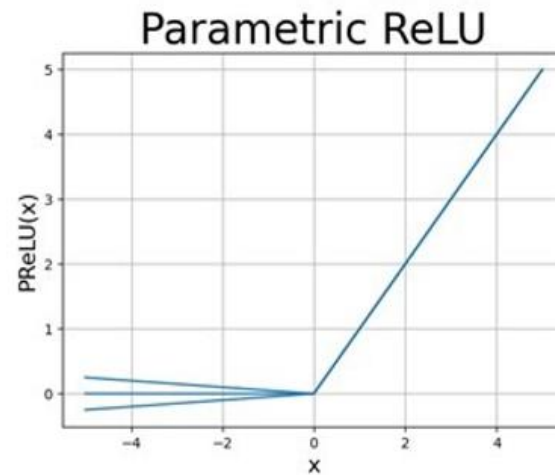
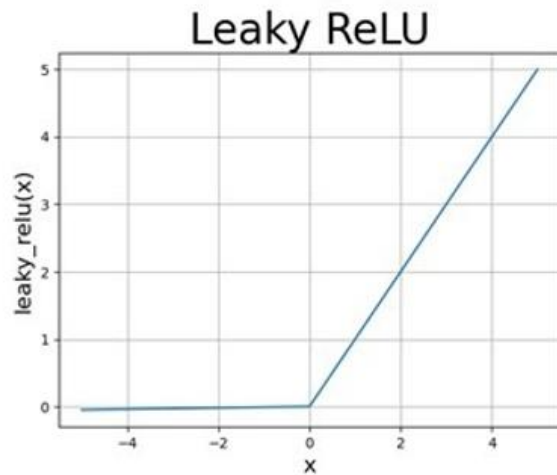
□ Mitigating the Vanishing Gradient Problem

- ReLU helps mitigate the vanishing gradient problem, which can occur during backpropagation in deep networks.
- The vanishing gradient problem arises when gradients become extremely small as they are propagated back through many layers, making it challenging to update the weights effectively. ReLU's derivative is 1 for positive inputs, allowing gradients to flow more easily.

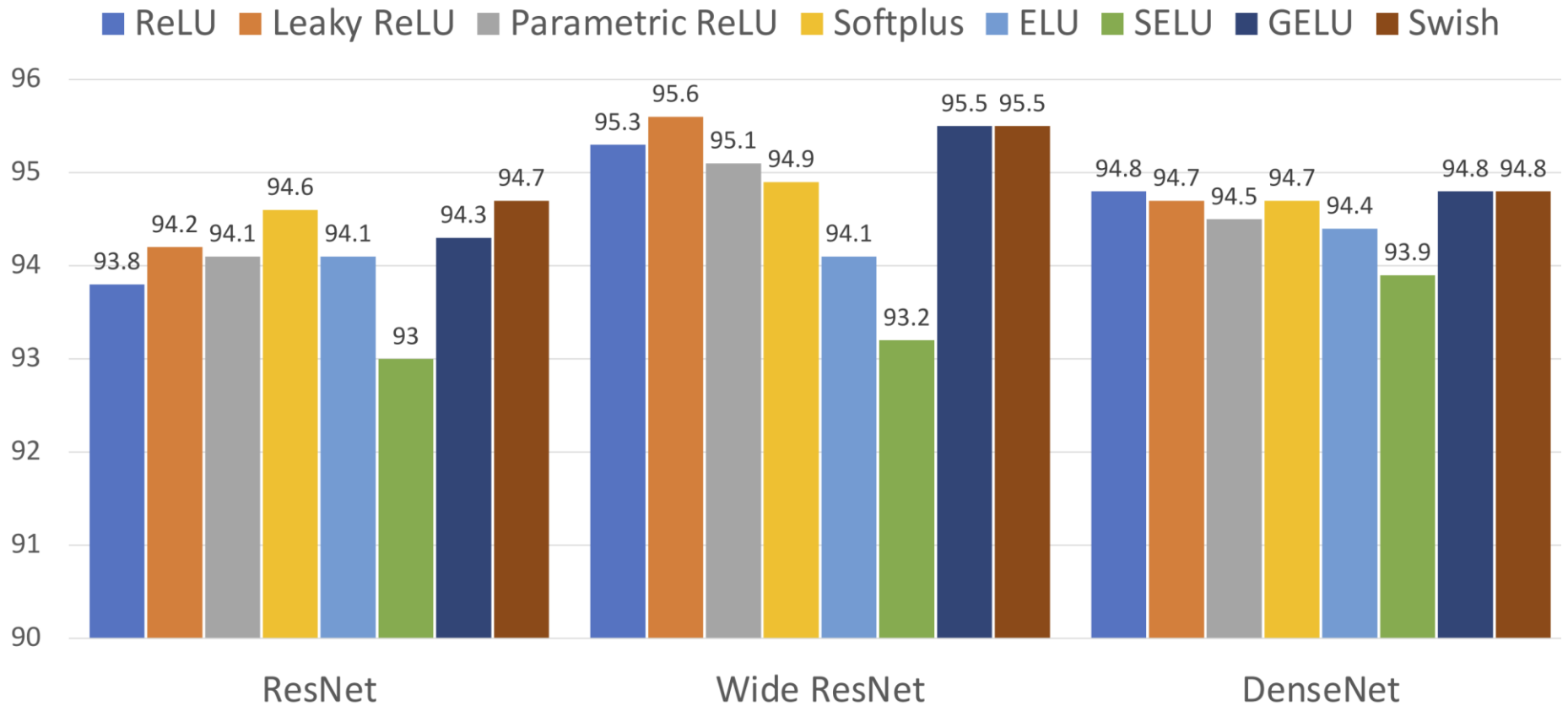
□ Sparse Activation

- ReLU activation leads to sparsity in the network. Since ReLU sets all negative values to zero, it can result in sparse activation patterns, where only a subset of neurons is activated.
- This can be beneficial for memory efficiency and computational speed.

Rectified Linear Unit Variants



Accuracy on CIFAR10



Don't think too hard. Just use ReLU

- Try out Leaky ReLU / ELU / SELU / GELU if you need to squeeze that last 0.1%
- Don't use sigmoid or tanh
- Some (very) recent architectures use GeLU instead of ReLU, but the gains are minimal

Feature Map Parameters

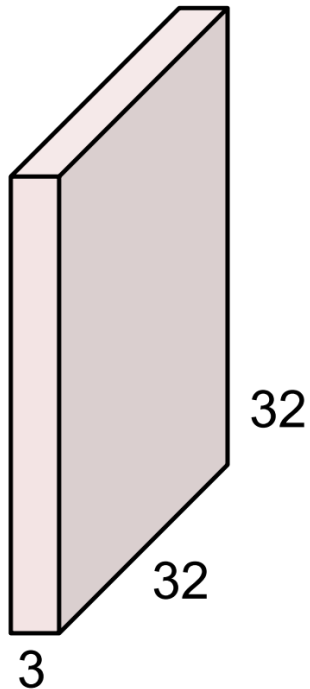
- The size of the Feature Map (Convolved Feature) is controlled by three parameters that we need to decide before the convolution step is performed:
 - ▣ **Depth:** Depth corresponds to the number of filters we use for the convolution operation.
 - ▣ **Stride:** Stride is the number of pixels by which we slide our filter matrix over the input matrix.
 - ▣ **Zero-padding:** Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix.

Depth

- Using different filters (or kernels) for each convolutional layer in a Convolutional Neural Network (CNN) allows the network to learn a diverse set of features at different levels of abstraction.
 - ▣ First layers: The first layers of a CNN for image classification typically use edge detection filters, color filters, and texture filters to extract different features from the input images.
 - ▣ Intermediate layers: The intermediate layers of a CNN for image classification typically use more complex filters to learn more abstract features from the input images.
 - ▣ Final layers: The final layers of a CNN for image classification typically use object-specific filters to detect specific objects in the input images. These filters are typically complex and have large receptive fields.

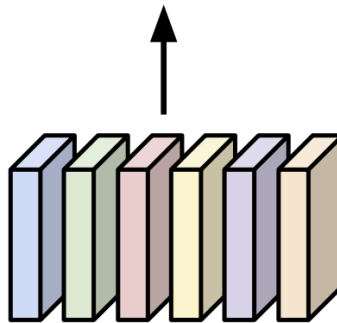
Different Activation Maps

3x32x32 image

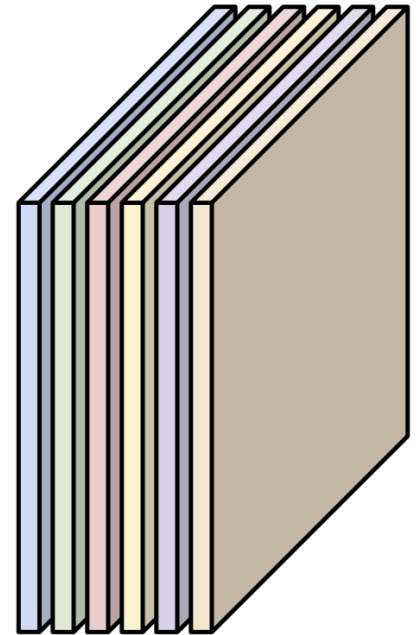


Convolution
Layer

6x3x5x5
filters



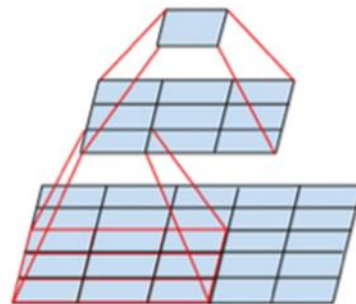
28x28 grid, at each
point a 6-dim vector



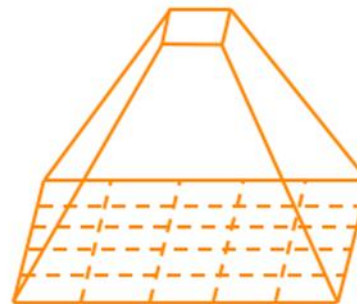
Stack activations to get a
6x28x28 output image!

Receptive Fields

- Region of the input that each element of an activation map is influenced by
- Important to manage the size of the RF:
 - ▣ A small RF can miss important information in an image
 - ▣ But too big an RF causes overfitting (intuition: since a large enough RF eventually is the same as a dense nn)



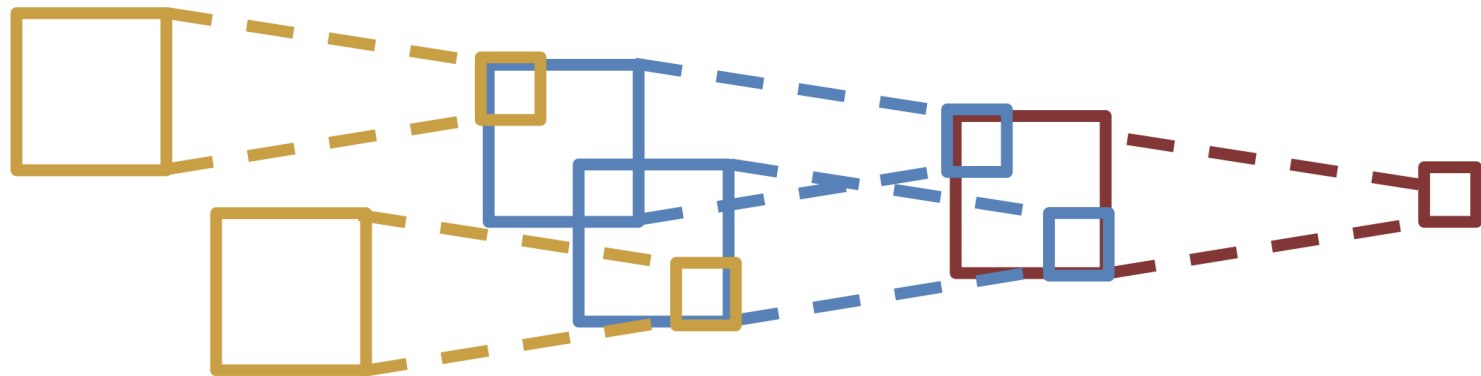
two successive
3x3 convolutions



5x5 convolution

Receptive Fields

- For convolution with kernel size K , each element in the output depends on a $K \times K$ receptive field in the input
- Each successive convolution adds $K - 1$ to the receptive field size. With L layers the receptive field size is $1 + L * (K - 1)$
- Problem: For large images we need many layers for each output to “see” the whole image
- Solution: Downsample inside the network
 - ▣ Stride the convolution

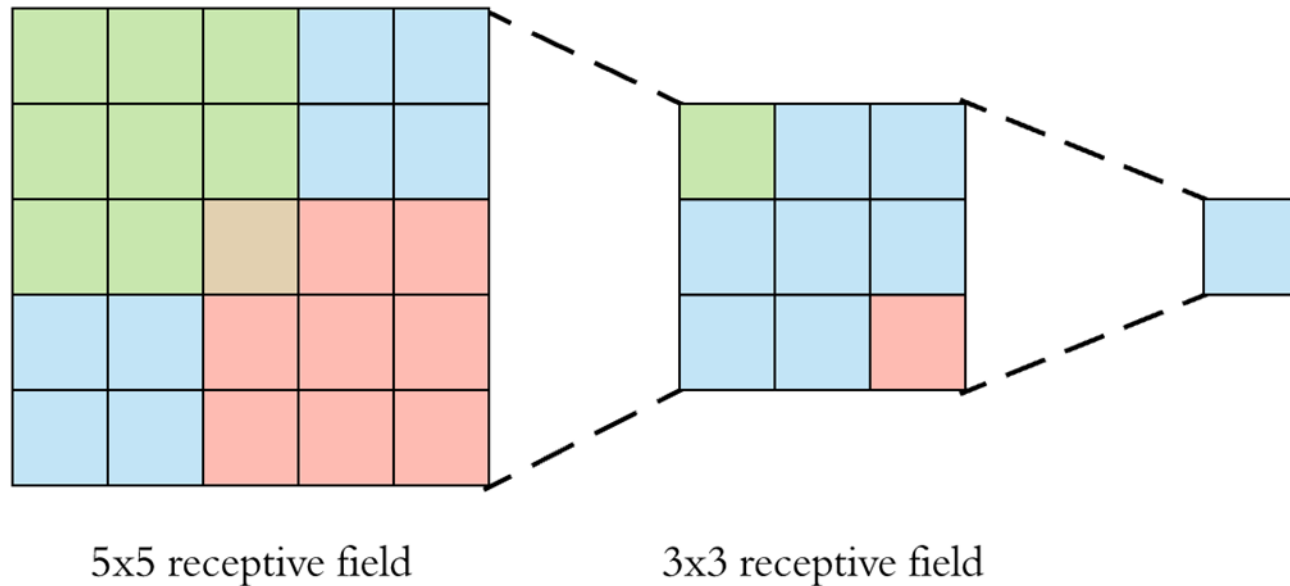


Input

Output

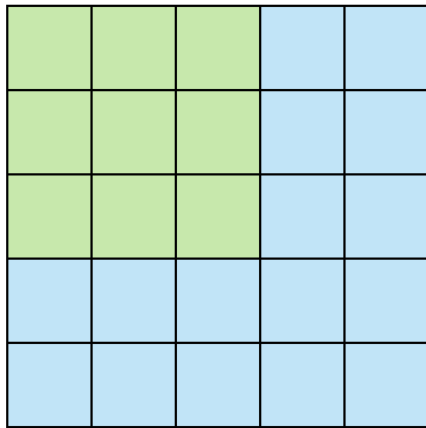
Receptive Fields

- Are 2 - 3×3 convolutions the same as one 5×5 convolution?
 - No
 - 2 3×3 convolutions has less parameters and more non-linearities

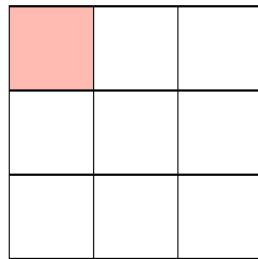


Stride

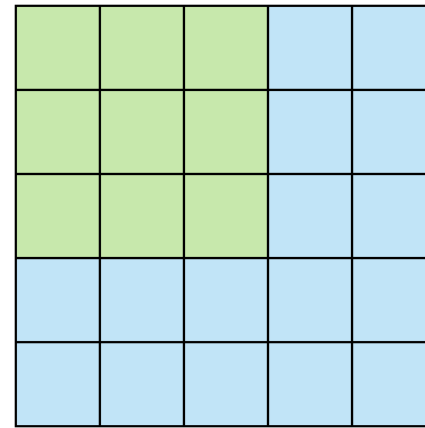
- The amount of pixels to slide the filter by (both horizontally and vertically):
 - ▣ A stride of 1 will shift the filter every pixel
 - ▣ A stride of 2 will shift the filter every 2 pixels



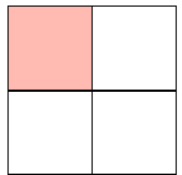
Stride 1



Feature Map



Stride 2



Feature Map

Why using Stride

□ Spatial Dimension Reduction

- ▣ By using a stride greater than 1, the convolution operation skips some positions, leading to a reduction in the spatial dimensions of the output feature map.
- ▣ This reduction can be intentional, especially in the early layers of a CNN, where capturing fine-grained spatial details might be less critical.
- ▣ This reduction can be beneficial for computational efficiency and memory usage.

□ Increased Receptive Field

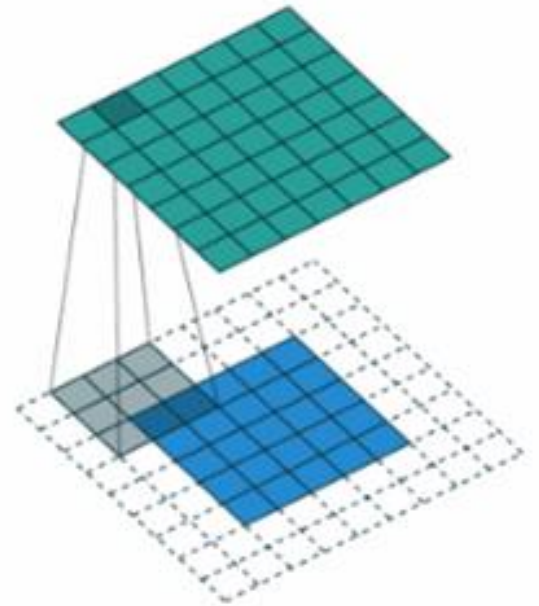
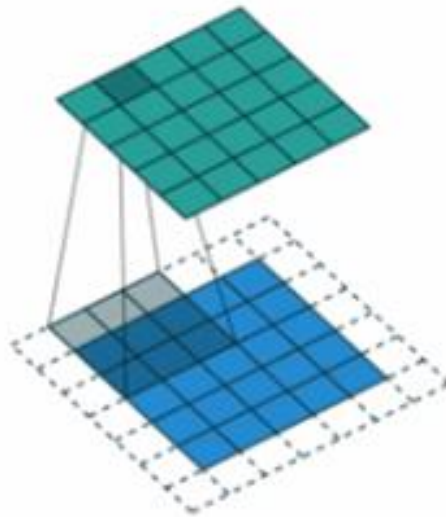
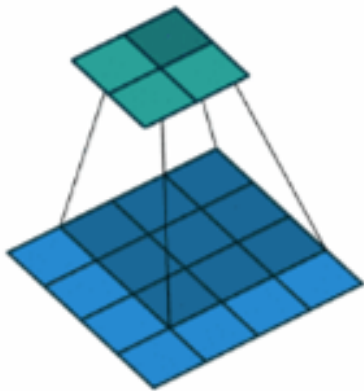
- ▣ A larger stride allows the convolutional filter to cover a larger region of the input in each step.
- ▣ This increased receptive field can help the network capture more global features and patterns.

□ Reduce overfitting.

- ▣ Using a larger stride can help to reduce overfitting by making the network more robust to noise and variations in the input images.

Padding

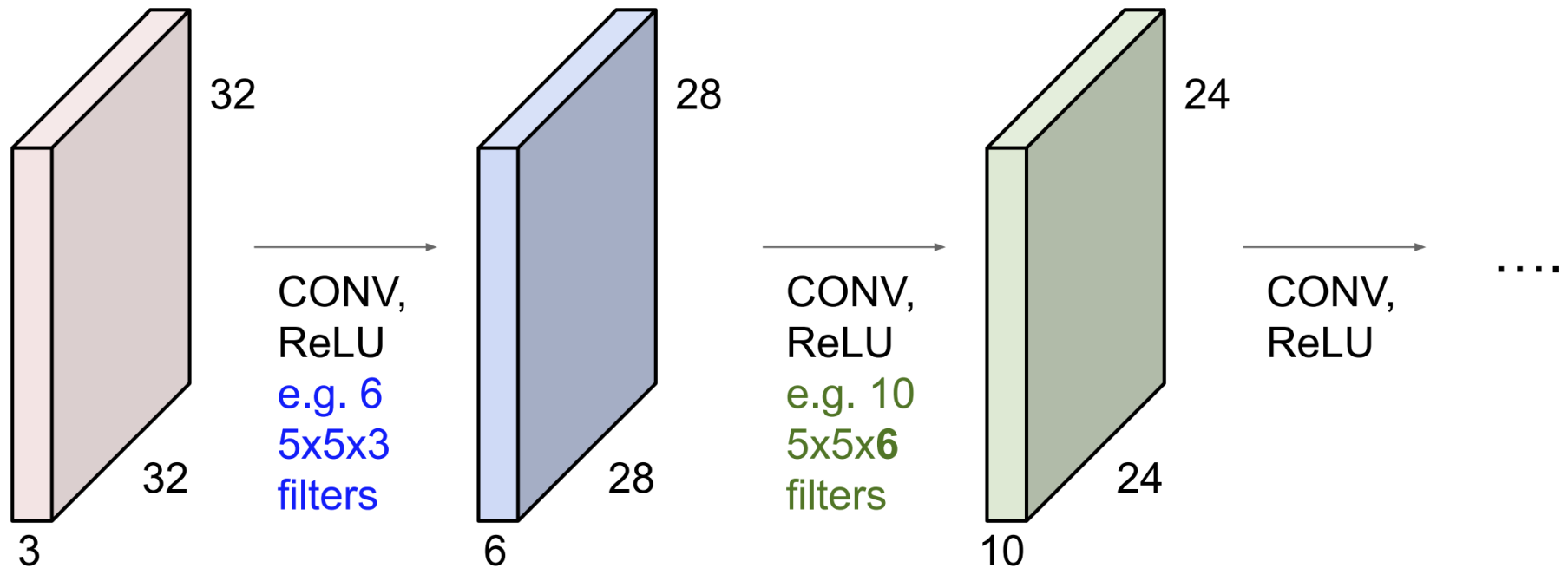
- Convolving an image with a filter results in a block with a smaller height and width — what if we want the height and width as before?



A closer look at spatial dimensions

Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 \rightarrow 28 \rightarrow 24 ...). Shrinking too fast is not good, doesn't work well.



A closer look at spatial dimensions

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

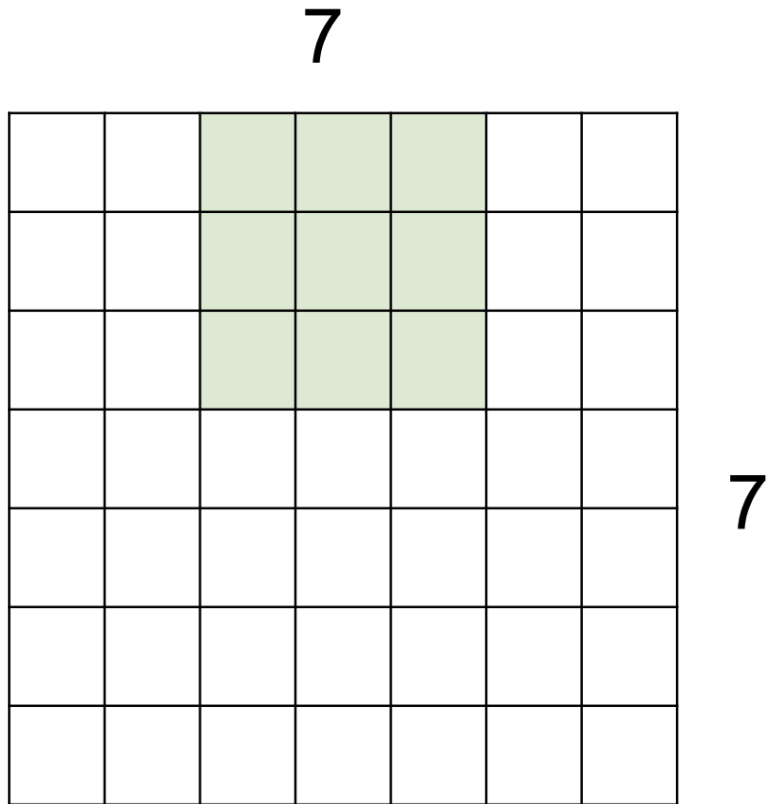
in general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

A closer look at spatial dimensions



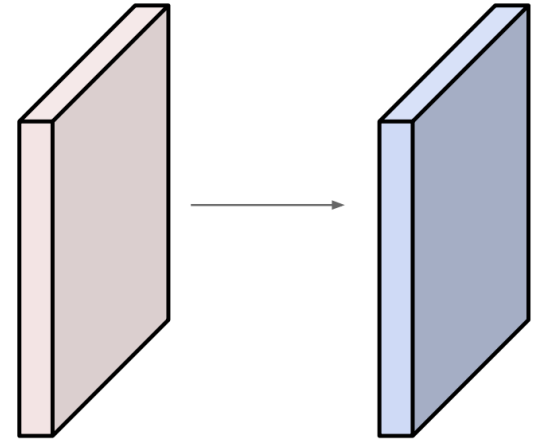
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

=> 3x3 output!

A closer look at spatial dimensions

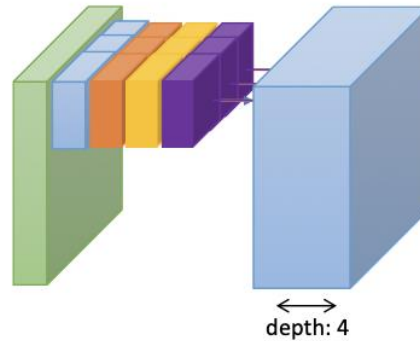
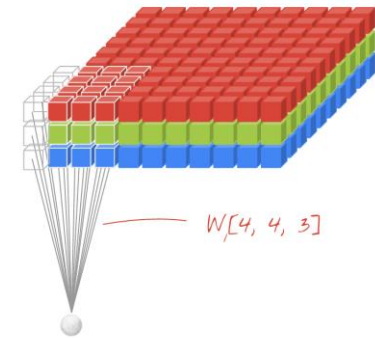
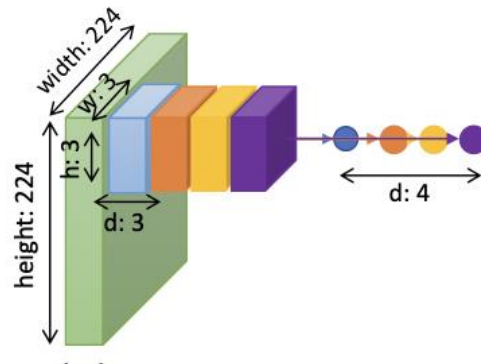
Examples time:

Input volume: **32x32x3**
10 **5x5** filters with stride 1, pad 2



Number of parameters in this layer?
each filter has $5*5*3 + 1 = 76$ params (+1 for bias)
 $\Rightarrow 76*10 = 760$

Another way to think about Conv layers



The only reason our filters have to be 3 tall (ie: span 3 channels) is because our input is 3 features tall (rgb)

Say our image is rgba (a for alpha a.k.a. brightness), we would need a filter that is 4 channels deep

A closer look at spatial dimensions

Input: $C_{in} \times H \times W$

Hyperparameters:

- **Kernel size:** $K_H \times K_W$
- **Number filters:** C_{out}
- **Padding:** P
- **Stride:** S

Weight matrix: $C_{out} \times C_{in} \times K_H \times K_W$
giving C_{out} filters of size $C_{in} \times K_H \times K_W$

Bias vector: C_{out}

Output size: $C_{out} \times H' \times W'$ where:

- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$

Common settings:

$K_H = K_W$ (Small square filters)

$P = (K - 1) / 2$ ("Same" padding)

$C_{in}, C_{out} = 32, 64, 128, 256$ (powers of 2)

$K = 3, P = 1, S = 1$ (3x3 conv)

$K = 5, P = 2, S = 1$ (5x5 conv)

$K = 1, P = 0, S = 1$ (1x1 conv)

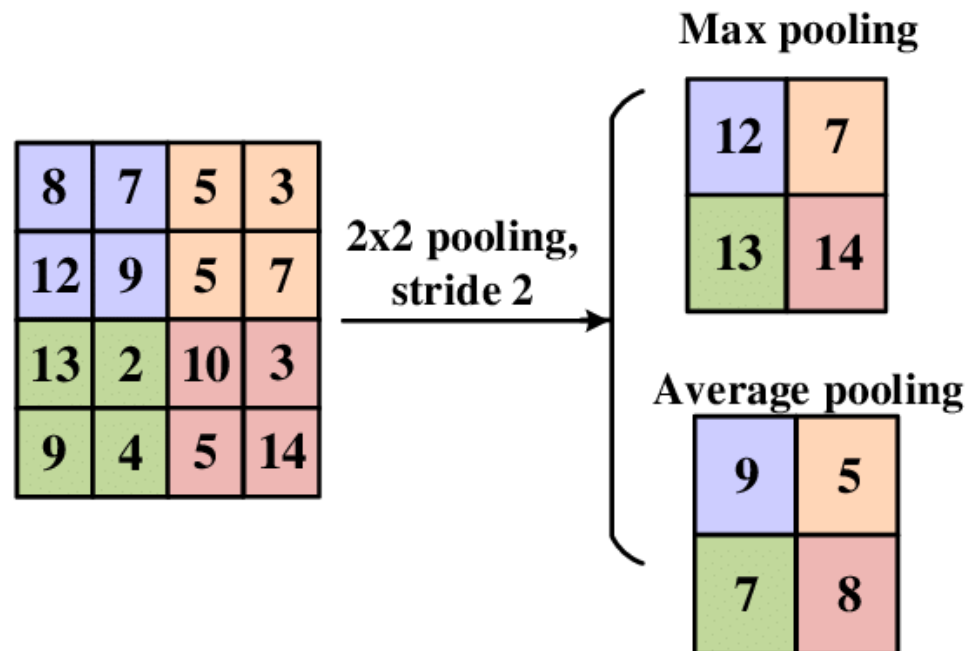
$K = 3, P = 1, S = 2$ (Downsample by 2)

Pooling Layer

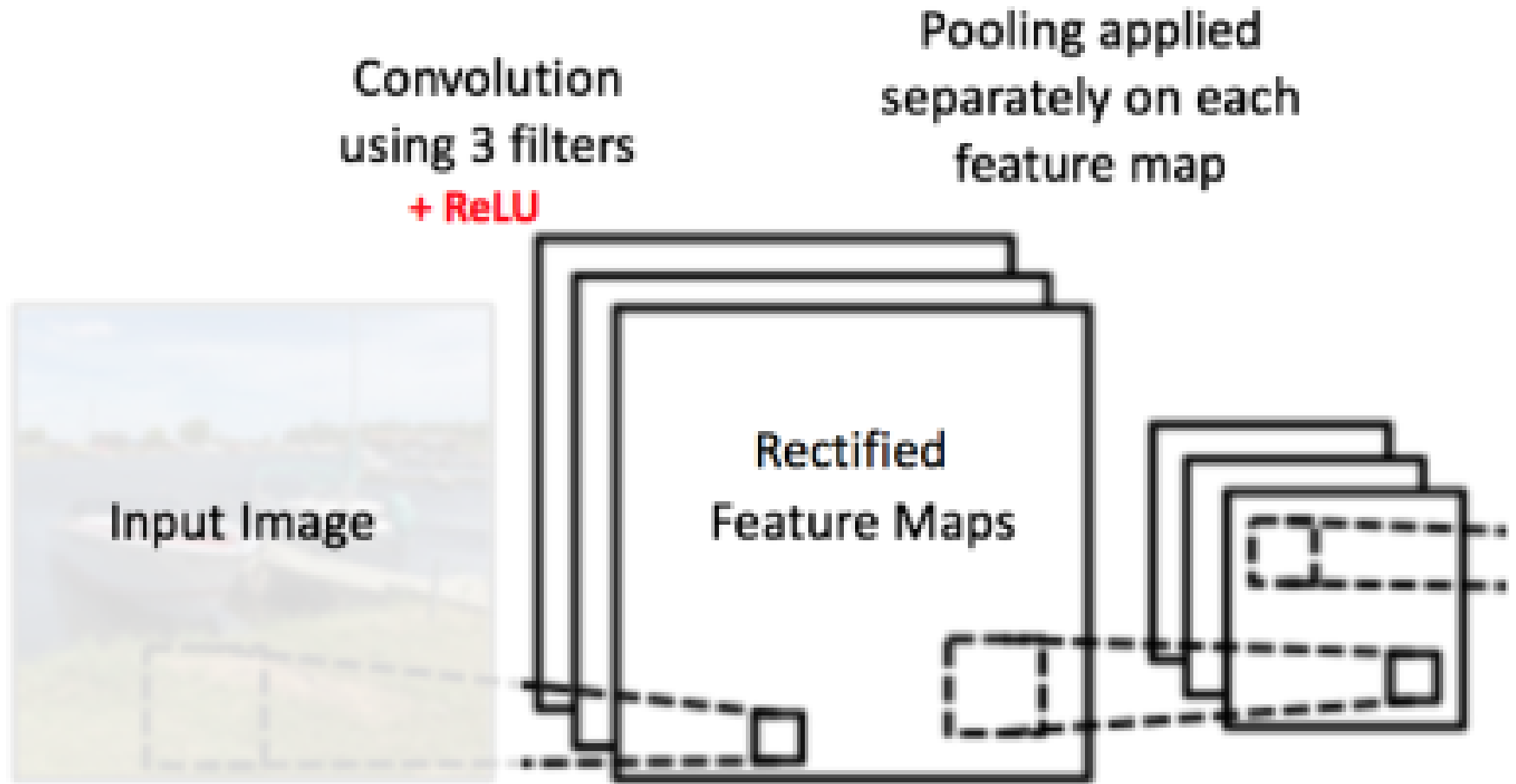
- Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information.
- Spatial Pooling can be of different types: Max, Average, Sum etc.
- In particular, pooling:
 - ▣ Makes the input representations (feature dimension) smaller and more manageable
 - ▣ Reduces the number of parameters and computations in the network, therefore, controlling overfitting
 - ▣ Makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).
 - ▣ Helps us arrive at an almost scale invariant representation of our image (the exact term is “equivariant”). This is very powerful since we can detect objects in an image no matter where they are located.

Pooling Layer

- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially.
- The most common form is a 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, discarding 75% of the activations.



Pooling applied to Rectified Feature Maps



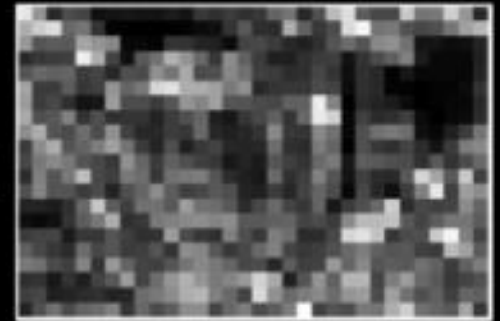
Effect of Pooling on the Rectified Feature Map



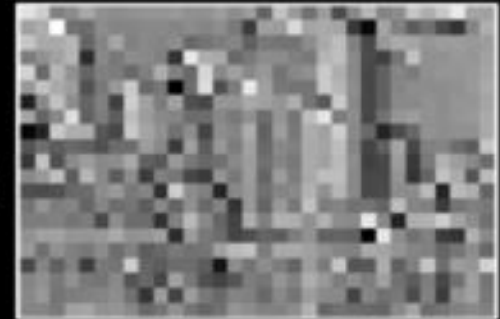
Rectified Feature Map

Pooling
→

Max

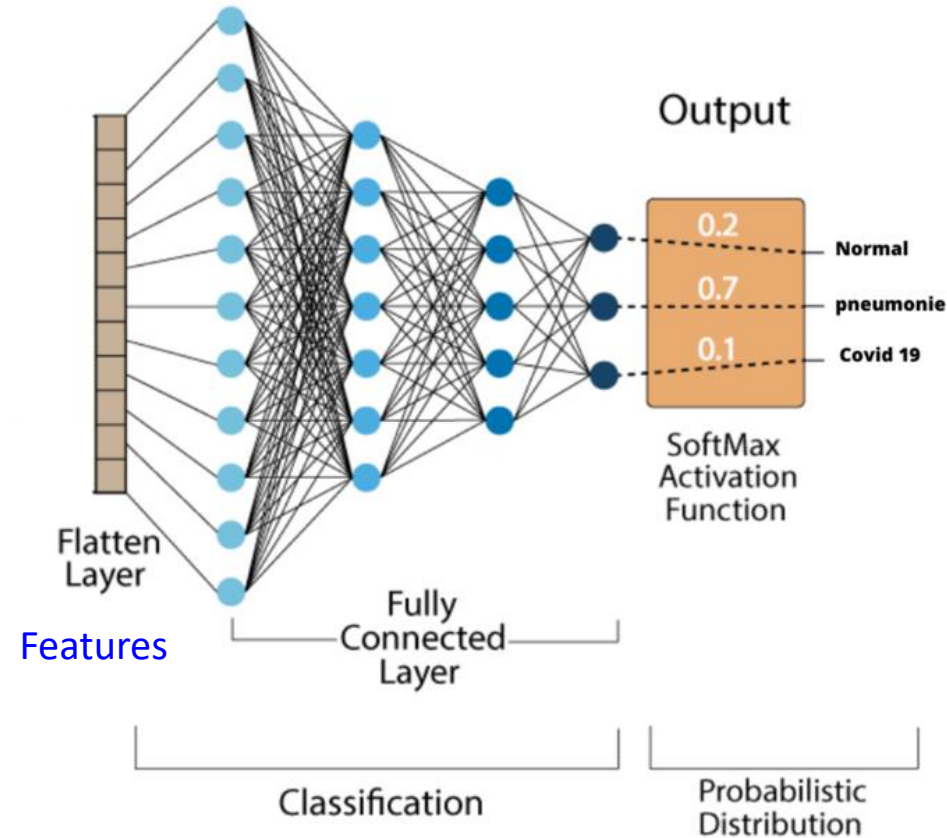


Sum



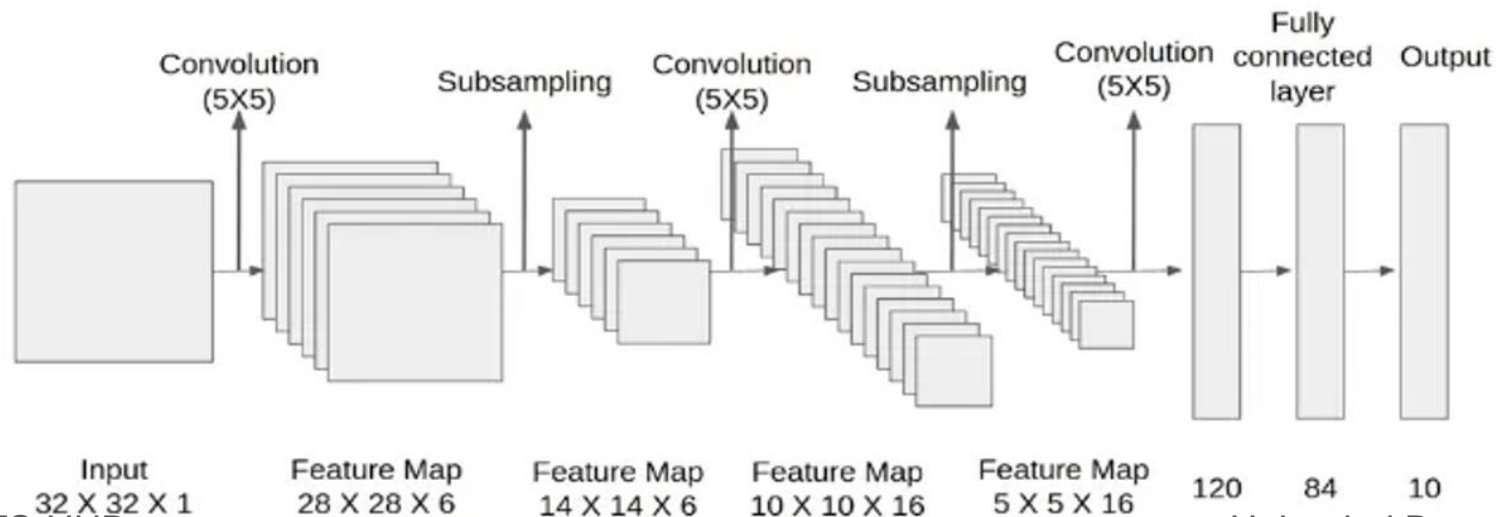
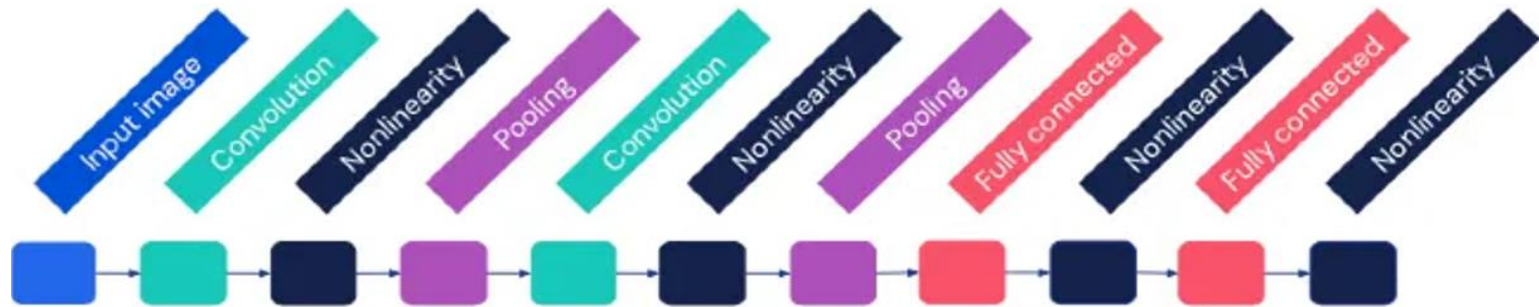
Fully Connected Layer

- The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used).
- The output from the convolutional and pooling layers represent high-level features of the input image.
- The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.
- Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better



Putting it all together – LeNet Architecture (1998)

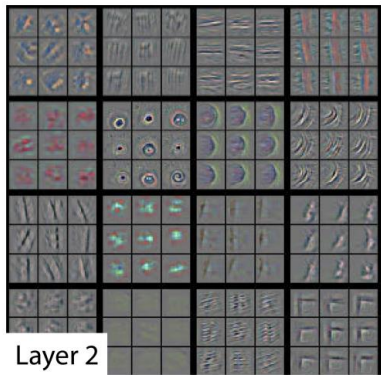
- LeNet-5 convnet for handwritten digit recognition



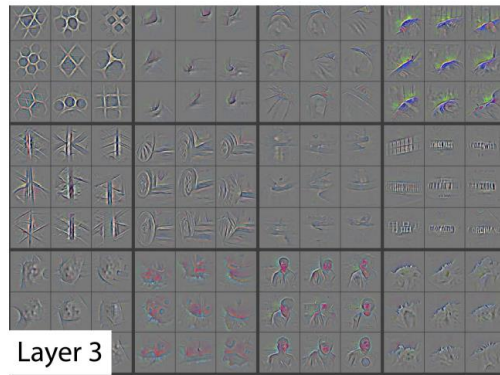
Training CNN using Backpropagation

- **Step1:** We initialize all filters and parameters / weights with random values
- **Step2:** The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
 - ▣ Lets say the output probabilities for an image are [0.2, 0.4, 0.1, 0.3]
 - ▣ Since weights are randomly assigned for the first training example, output probabilities are also random.
- **Step3:** Calculate the total error at the output layer (summation over all 4 classes)
 - ▣ **Total Error = $\sum \frac{1}{2} (\text{target probability} - \text{output probability})^2$**
- **Step4:** Use Backpropagation to calculate the *gradients* of the error with respect to all weights in the network and use *gradient descent* to update all filter values / weights and parameter values to minimize the output error.
 - ▣ The weights are adjusted in proportion to their contribution to the total error.
 - ▣ When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].
 - ▣ This means that the network has *learnt* to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.
 - ▣ Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.
- **Step5:** Repeat steps 2-4 with all images in the training set.

Visualizing what a convnet learns at each Layer



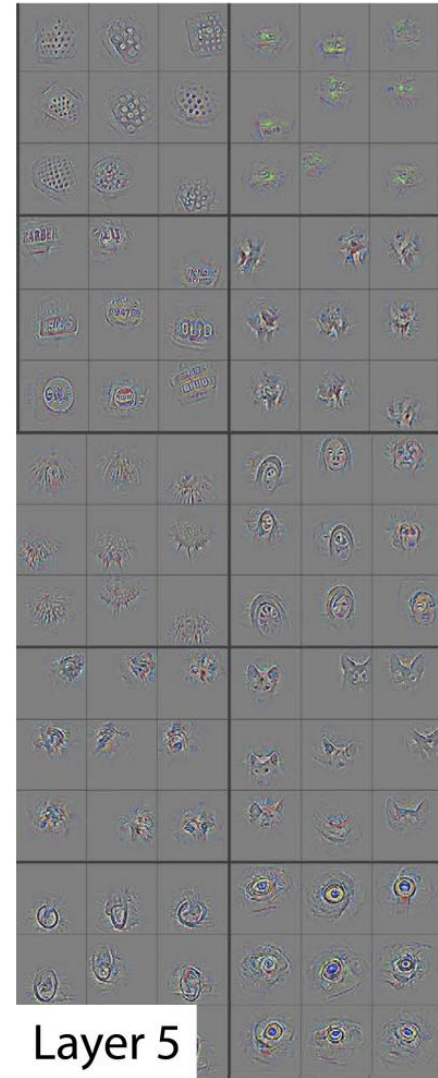
Layer 2



Layer 3



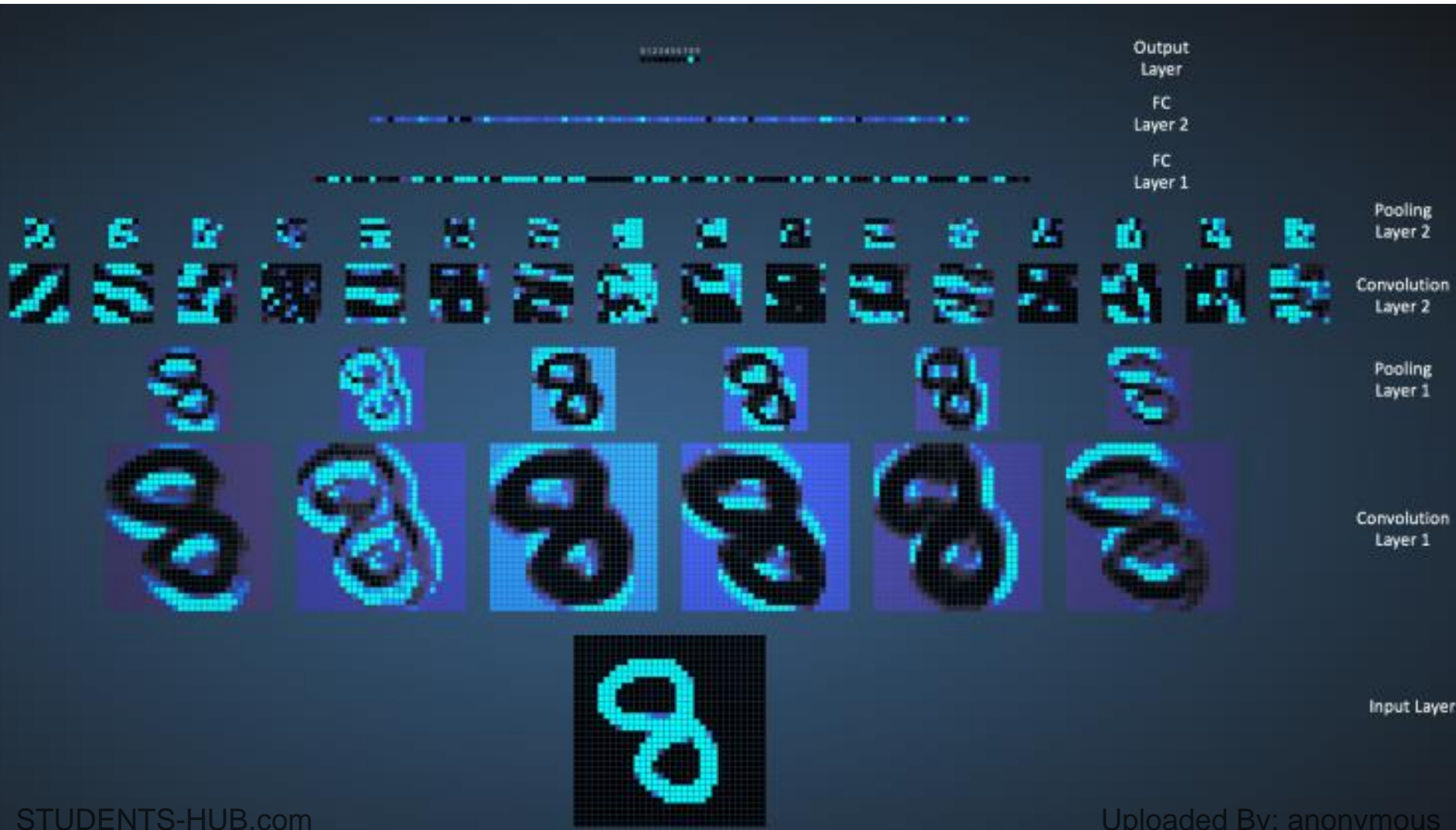
Layer 4



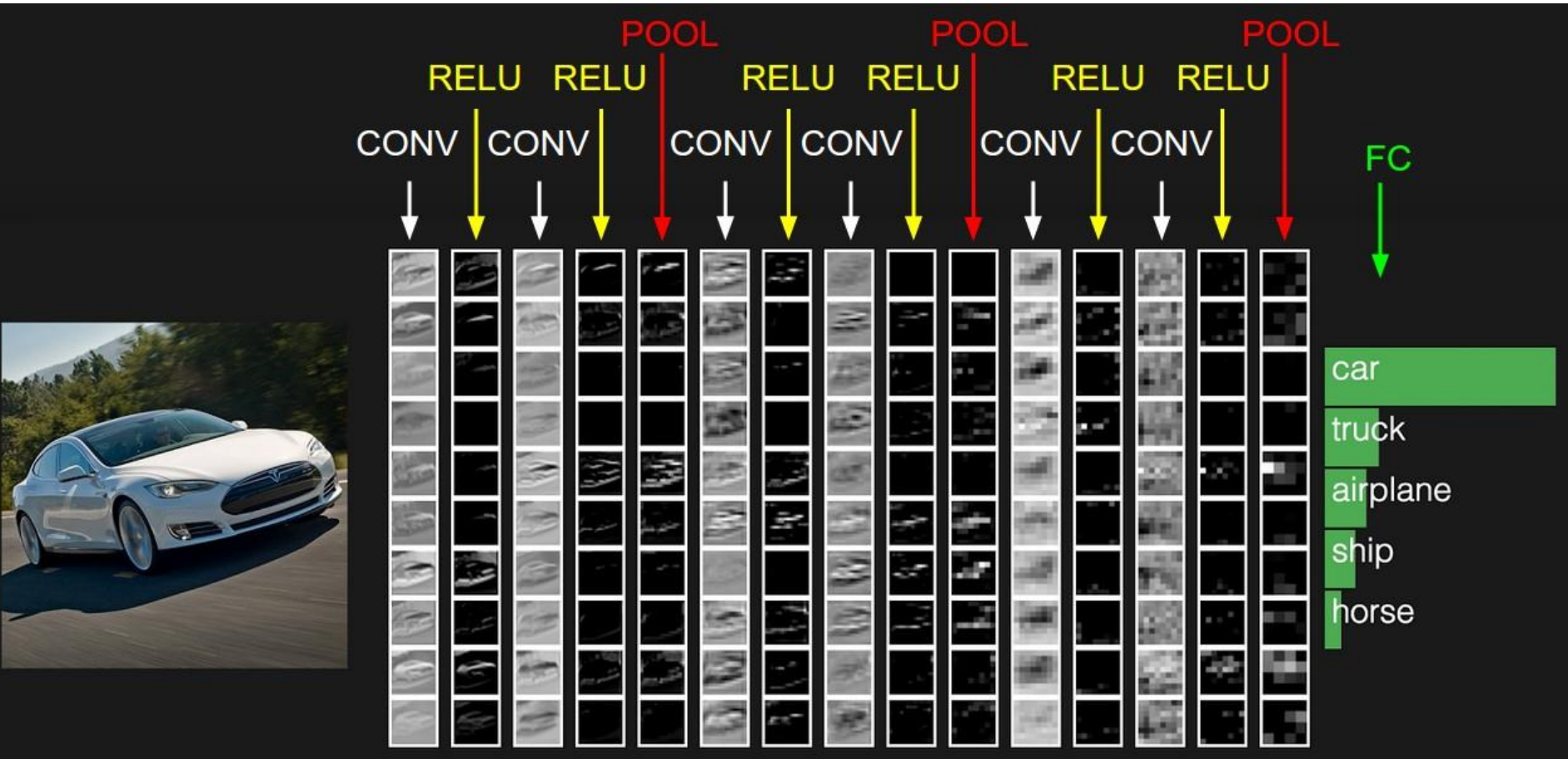
Layer 5

Visualizing what a convnet learns

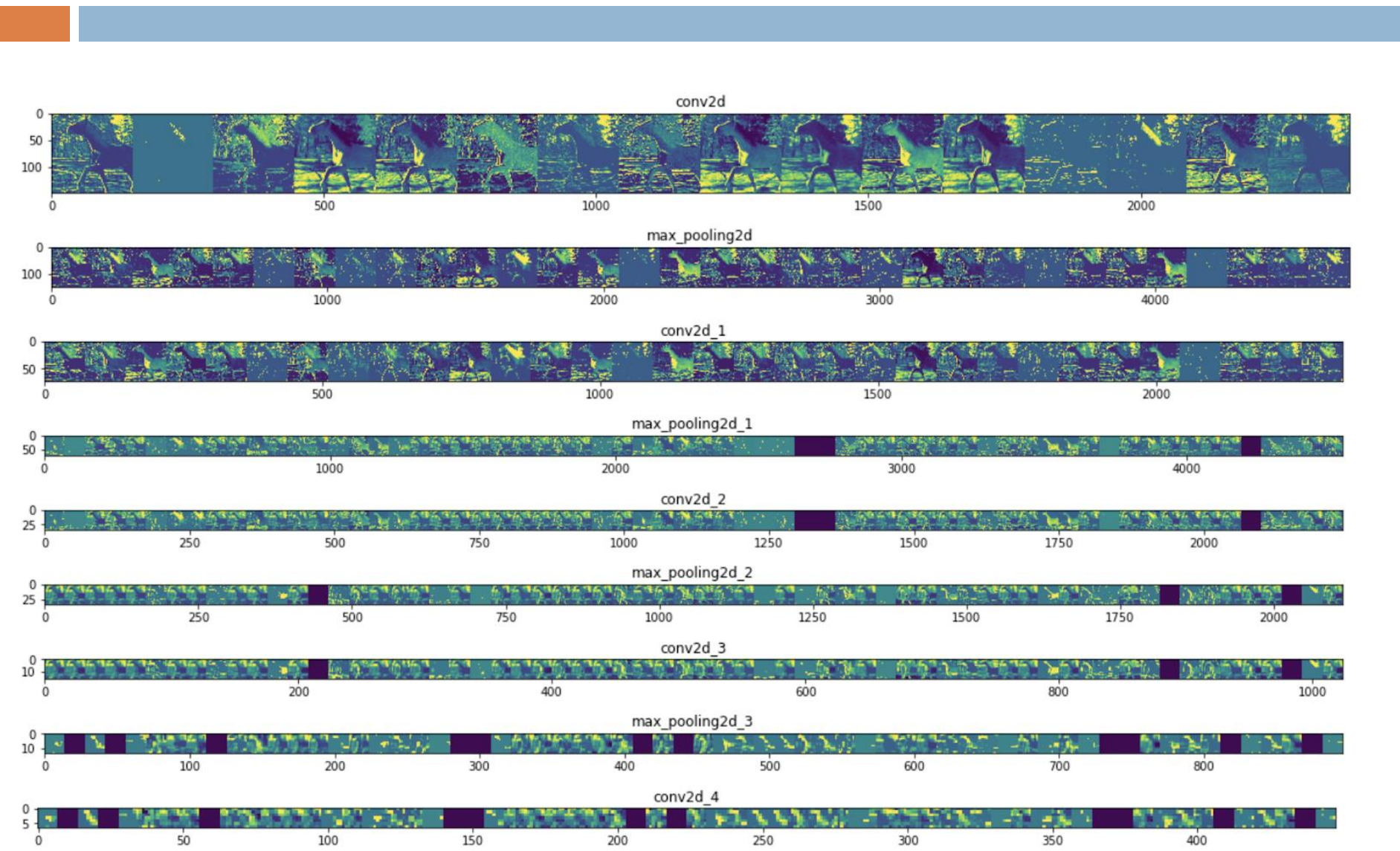
Convolutional Neural Network trained on the MNIST Database of handwritten digits



Visualizing what a convnet learns



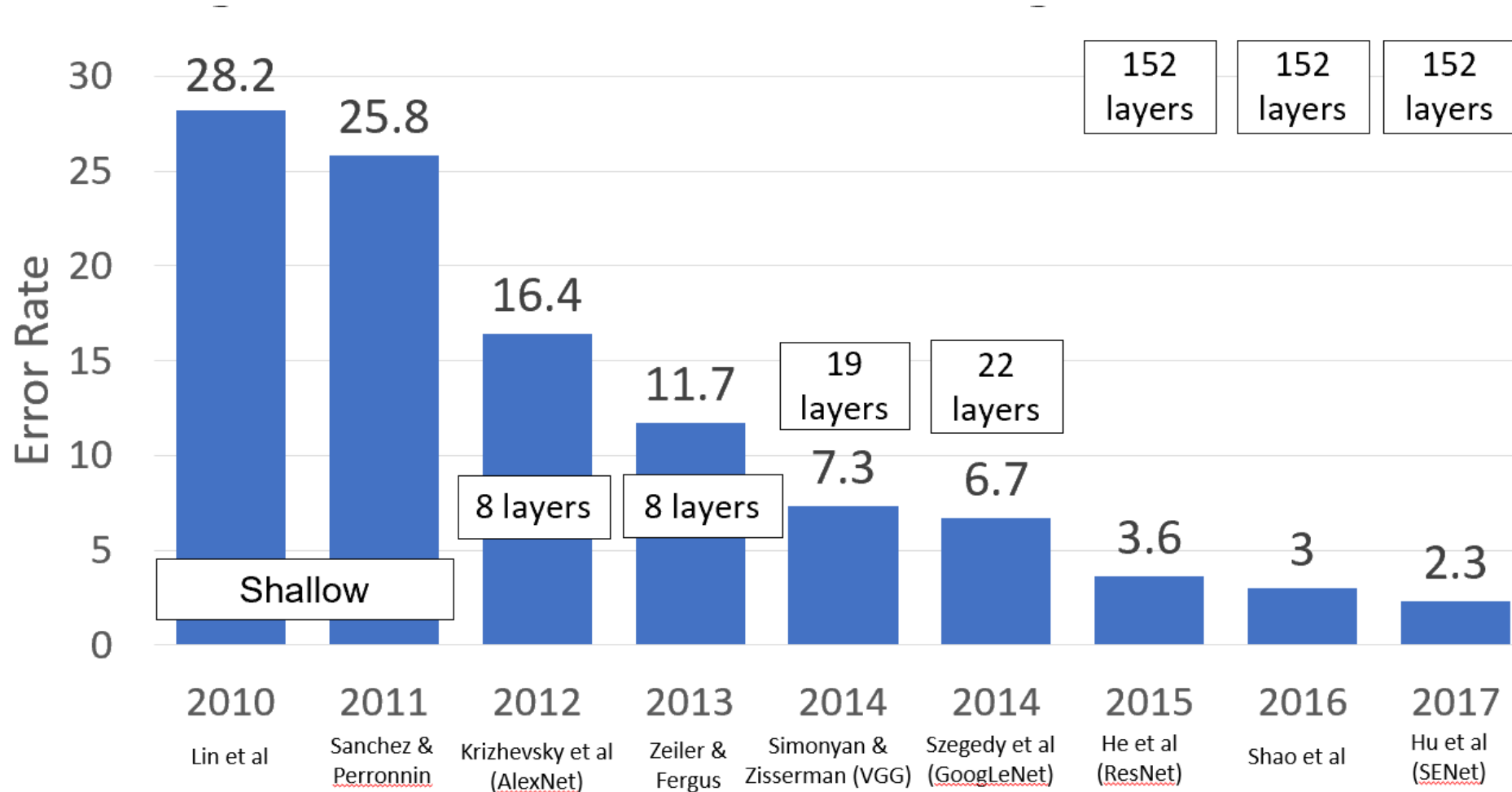
Visualizing what a convnet learns



Well Known ConvNet Architectures

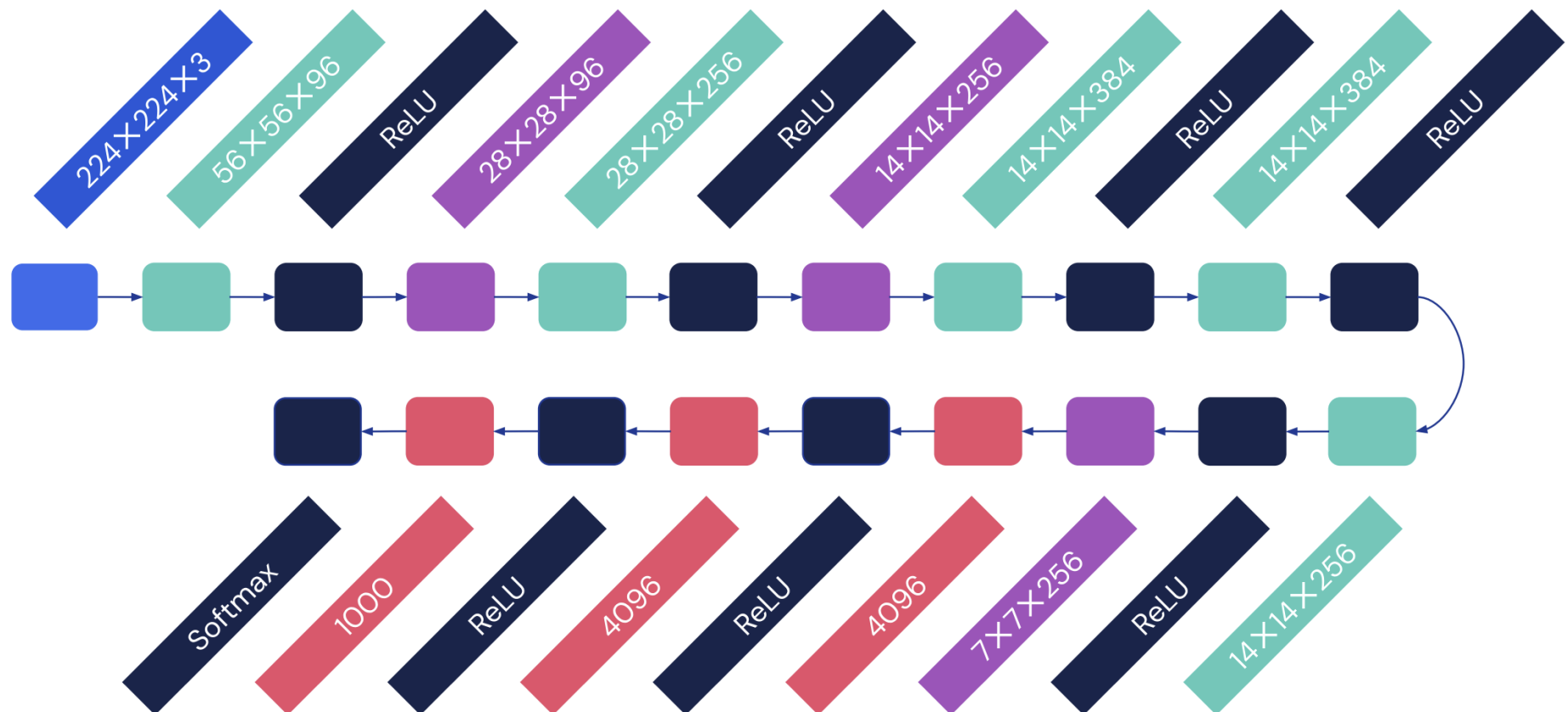
- **LeNet (1990s)**
- **AlexNet (2012)**
- **GoogLeNet (2014)**
- **VGGNet (2014)**
- **ResNets (2015)**
- **DenseNet (August 2016)**
- **MobileNet**
- **.....**

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

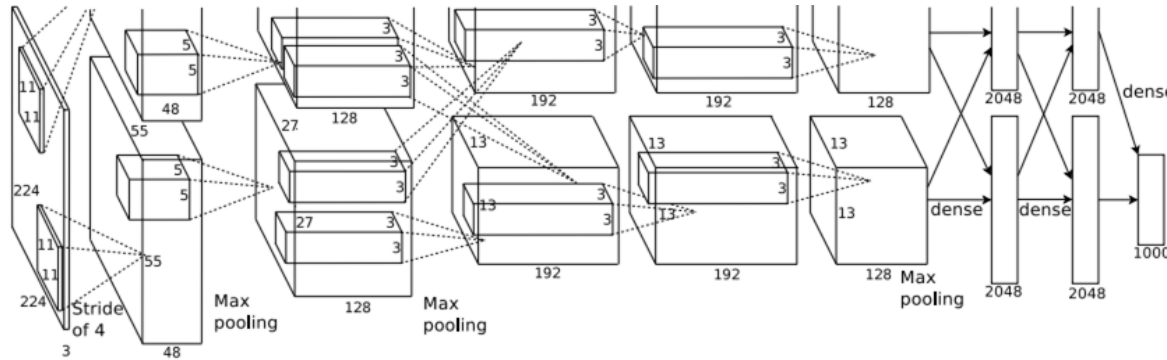


AlexNet [Krizhevsky et al. 2012]

- Has a similar architecture to LeNet-5 but was deeper and bigger with five Conv layers stacked on top of each other, followed by three fully connected layers



AlexNet



Details/Retrospectives:

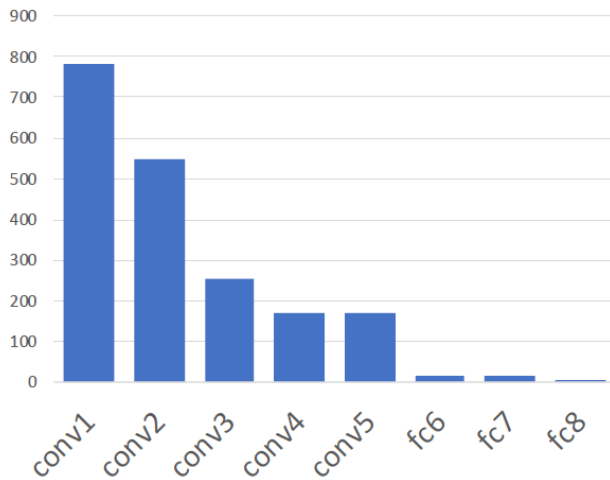
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4

Layer	Input size			Layer				Output size		memory (KB)	params (k)	flop (M)
	C	H / W		filters	kernel	stride	pad	C	H / W			
conv1	3	227		64	11	4	2	64	56	784	23	73
pool1	64	56			3	2	0	64	27	182	0	0
conv2	64	27		192	5	1	2	192	27	547	307	224
pool2	192	27			3	2	0	192	13	127	0	0
conv3	192	13		384	3	1	1	384	13	254	664	112
conv4	384	13		256	3	1	1	256	13	169	885	145
conv5	256	13		256	3	1	1	256	13	169	590	100
pool5	256	13			3	2	0	256	6	36	0	0
flatten	256	6						9216		36	0	0
fc6	9216			4096				4096		16	37,749	38
fc7	4096			4096				4096		16	16,777	17
fc8	4096			1000				1000		4	4,096	4

AlexNet

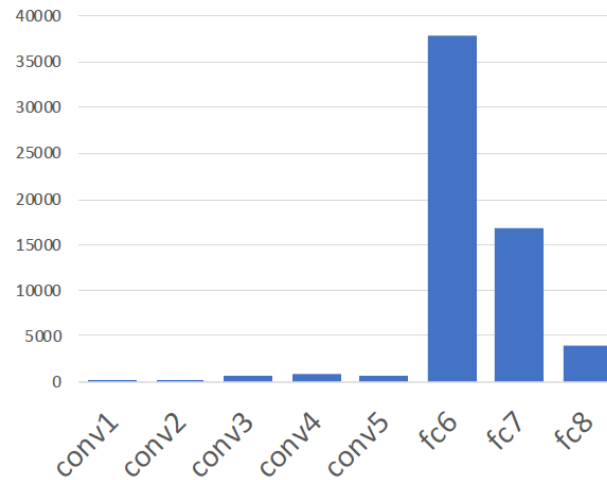
Most of the **memory usage** is in the early convolution layers

Memory (KB)



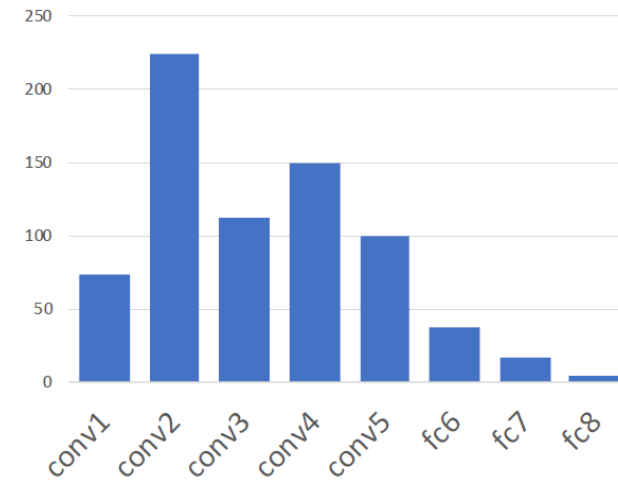
Nearly all **parameters** are in the fully-connected layers

Params (K)



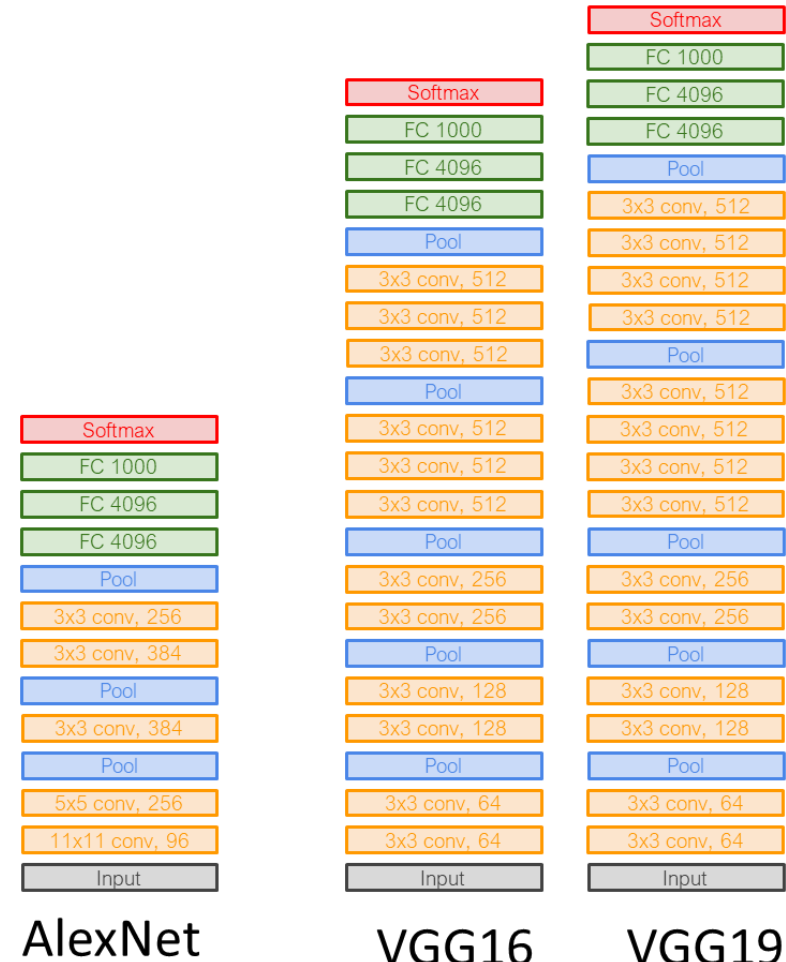
Most **floating-point ops** occur in the convolution layers

MFLOP



VGGNet [Simonyan and Zisserman, 2014]

- **VGGNet** is a very deep convnet. It stacks many convolutional layers before pooling. Moreover, it uses “same” convolutions to avoid resolution reduction.
- VGG Design rules:
 - ▣ All conv are 3x3 stride 1 pad 1
 - ▣ All max pool are 2x2 stride 2
 - ▣ After pool, double #channels
- Network has 5 convolutional **stages**:
 - ▣ Stage 1: conv-conv-pool
 - ▣ Stage 2: conv-conv-pool
 - ▣ Stage 3: conv-conv-pool
 - ▣ Stage 4: conv-conv-conv-[conv]-pool
 - ▣ Stage 5: conv-conv-conv-[conv]-pool



VGGNet

INPUT: [224x224x3] memory: $224*224*3=150\text{K}$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory: $112*112*64=800\text{K}$ params: 0

CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory: $56*56*128=400\text{K}$ params: 0

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory: $28*28*256=200\text{K}$ params: 0

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory: $14*14*512=100\text{K}$ params: 0

CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory: $7*7*512=25\text{K}$ params: 0

FC: [1x1x4096] memory: 4096 params: $7*7*512*4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096*4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096*1000 = 4,096,000$

Note:

Most memory is in early CONV

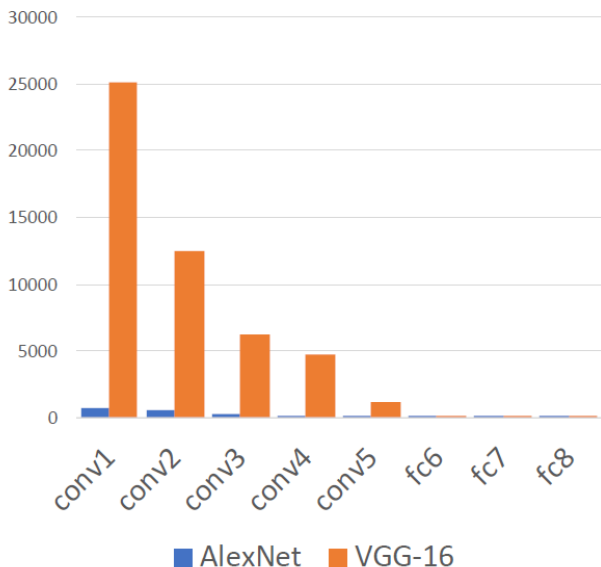
Most params are in late FC

TOTAL memory: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB}$ / image (only forward! $\sim*2$ for bwd)

TOTAL params: 138M parameters

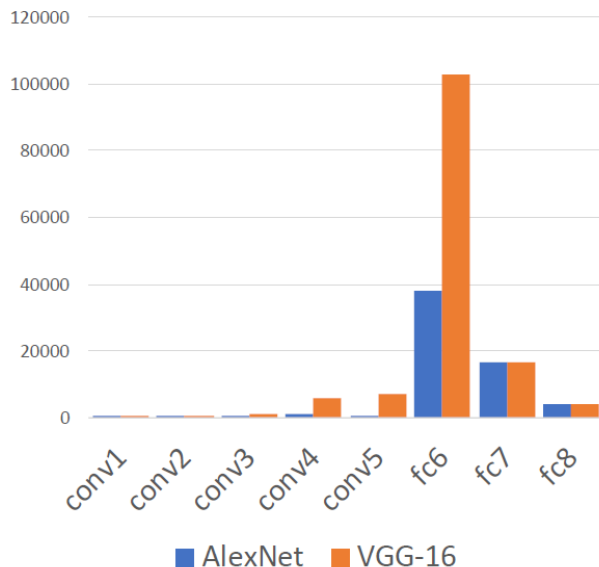
VGGNet

AlexNet vs VGG-16
(Memory, KB)



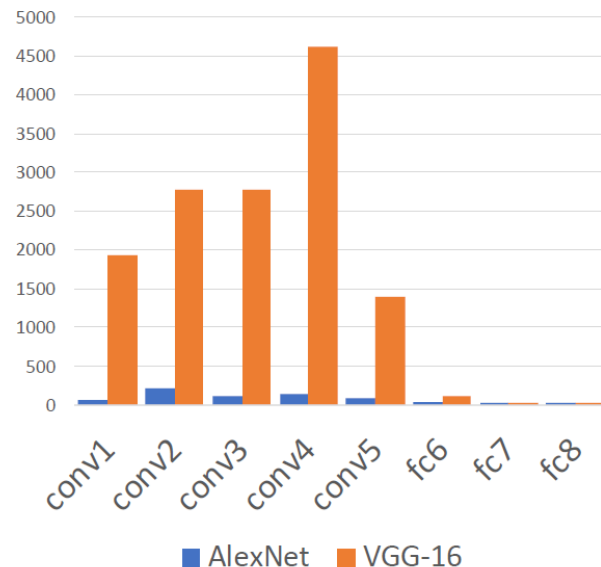
AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

AlexNet vs VGG-16
(Params, M)



AlexNet total: 61M
VGG-16 total: 138M (2.3x)

AlexNet vs VGG-16
(MFLOPs)

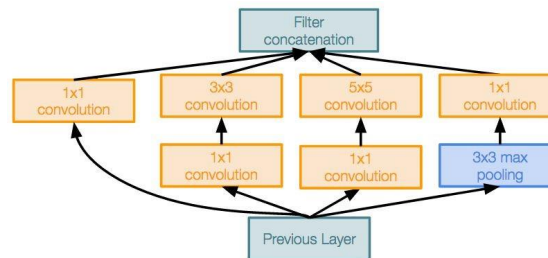


AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

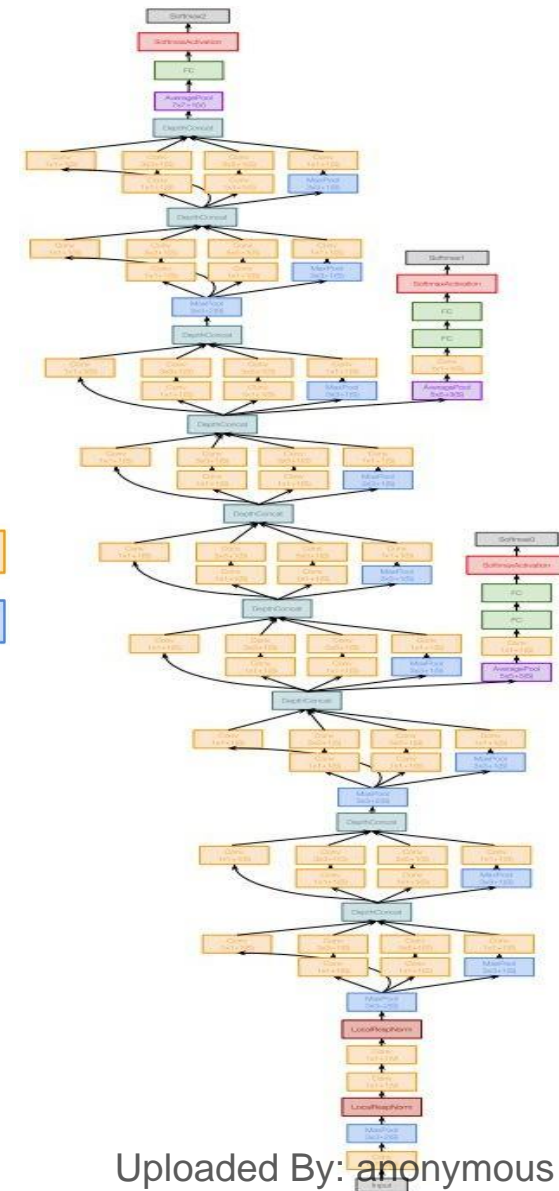
GoogLeNet (Inception-v1) [Szegedy et al., 2014]

□ Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
- 12x less than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)



Inception module



Key Features of GoogLeNet (Inception-v1)

□ Inception Modules:

- The inception module consists of multiple parallel convolutional branches of different filter sizes (1x1, 3x3, 5x5), along with a max-pooling branch.
- The outputs of these branches are concatenated along the depth dimension.
- This allows the network to capture features at multiple scales simultaneously.

□ Batch Normalization:

- Batch Normalization is applied to the input of each layer, contributing to faster convergence and improved training stability.

□ Global Average Pooling:

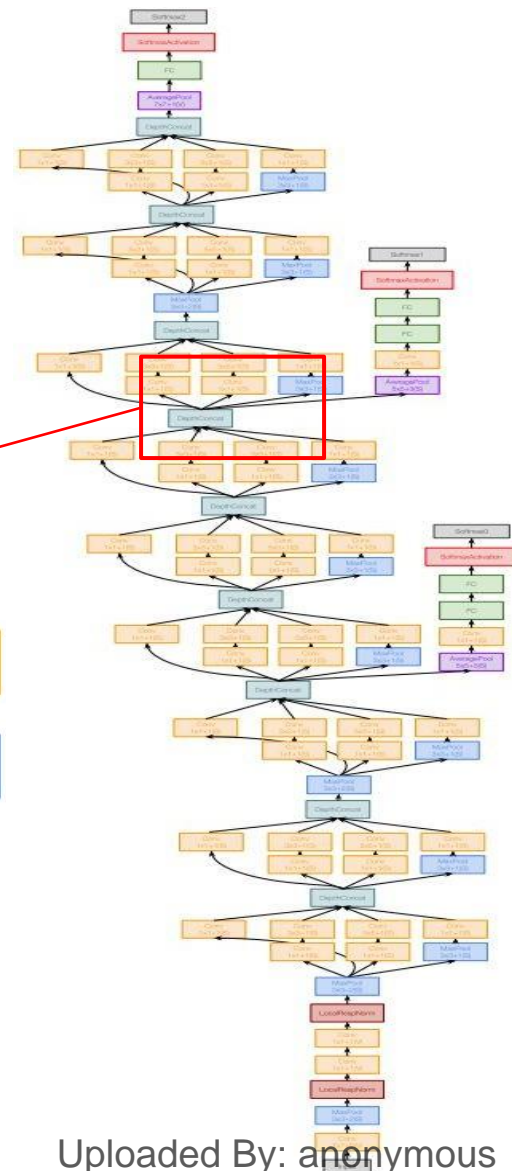
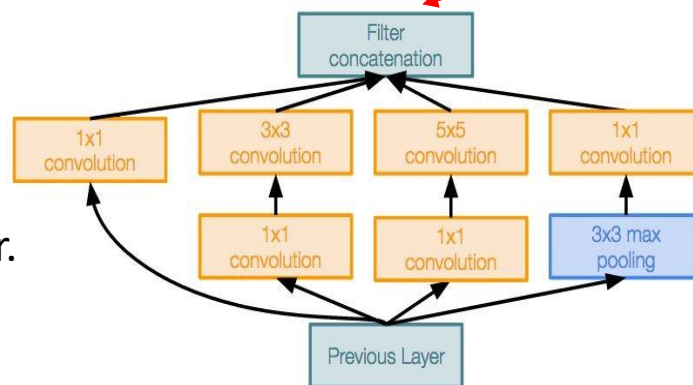
- Instead of using fully connected layers at the end of the network, GoogLeNet uses global average pooling.
- Global average pooling reduces the number of parameters and helps with model generalization.

□ Auxiliary Classifiers:

- Auxiliary classifiers, placed at intermediate layers, are introduced during training to provide additional gradient signals and combat the vanishing gradient problem.
- These auxiliary classifiers have their loss functions and contribute to the overall loss during training.

GoogLeNet: Inception Module

- The Inception Module is a fundamental building block of GoogLeNet (Inception-v1),
- The main purpose of the Inception Module is to enable the network to capture information at multiple scales by employing filters of different sizes in parallel.
- This helps the network efficiently learn both fine-grained and coarse-grained features within the same layer.
- Local unit with parallel branches
- Local structure repeated many times throughout the network



Structure of the Inception Module

1. 1x1 Convolution (Dimension Reduction):

- A 1x1 convolution is used to perform dimension reduction, reducing the number of channels:
$$\text{output}[i, j, k] = \sum(\text{input}[i, j, c] * \text{kernel}[1, 1, c, k] \text{ for } c \text{ channels})$$

 k is the index of the filter in the output.
 c is the index of the channel in the input.
 $\text{kernel}[1,1,c,k]$ is the weight associated with channel c of the input for filter k .
- This operation helps control the computational cost and provides a linear combination of features.

2. 3x3 Convolution:

- A 3x3 convolution captures features over a medium-sized receptive field.
- It helps capture spatial hierarchies within the image.

3. 5x5 Convolution:

- A 5x5 convolution captures features over a larger receptive field.
- It helps capture more global features and structures.

4. Max-Pooling:

- Max-pooling is used to capture the most important features within a local region.
- It provides some translation invariance and reduces spatial dimensions.

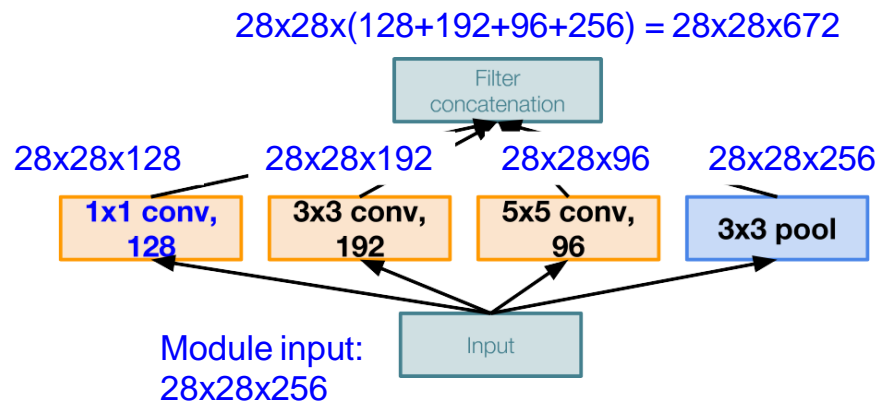
5. Concatenation:

- The outputs from all branches are concatenated along the depth dimension.

STUDENTS This creates a rich set of features that can capture information at different scales.

Structure of the Inception Module

Example:



Naive Inception module

Conv Ops:

[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 256$

[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 256$

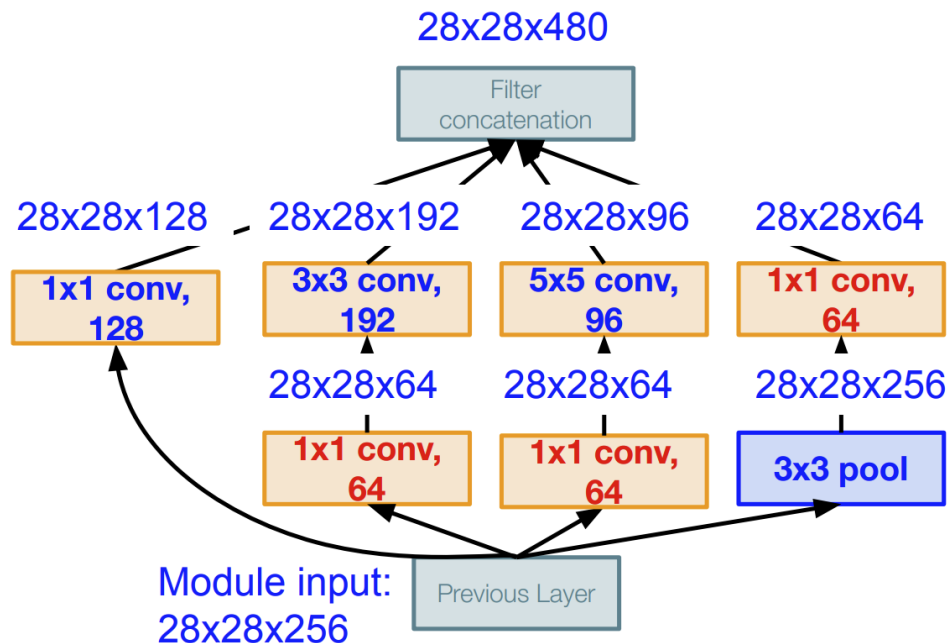
Total: 854M ops

Very expensive compute

Pooling layer preserves feature depth, which means total depth after concatenation can only grow at every layer!

Structure of the Inception Module

- Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:



Inception module with dimension reduction

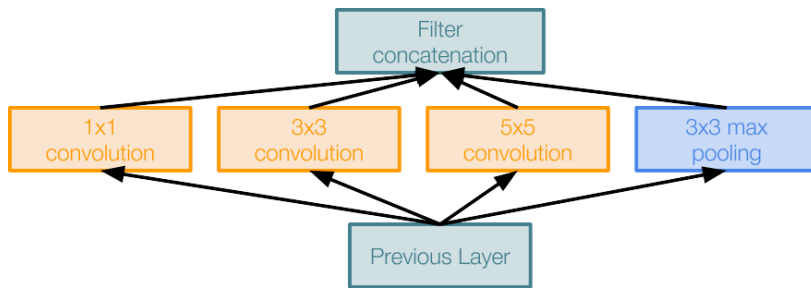
Conv Ops:

[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256

Total: 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

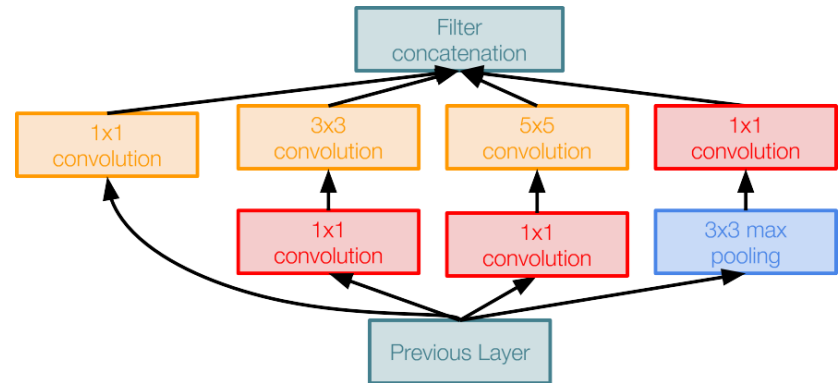
GoogLeNet: Inception Module



Naive Inception module

Total: 854M ops

1x1 conv “bottleneck” layers



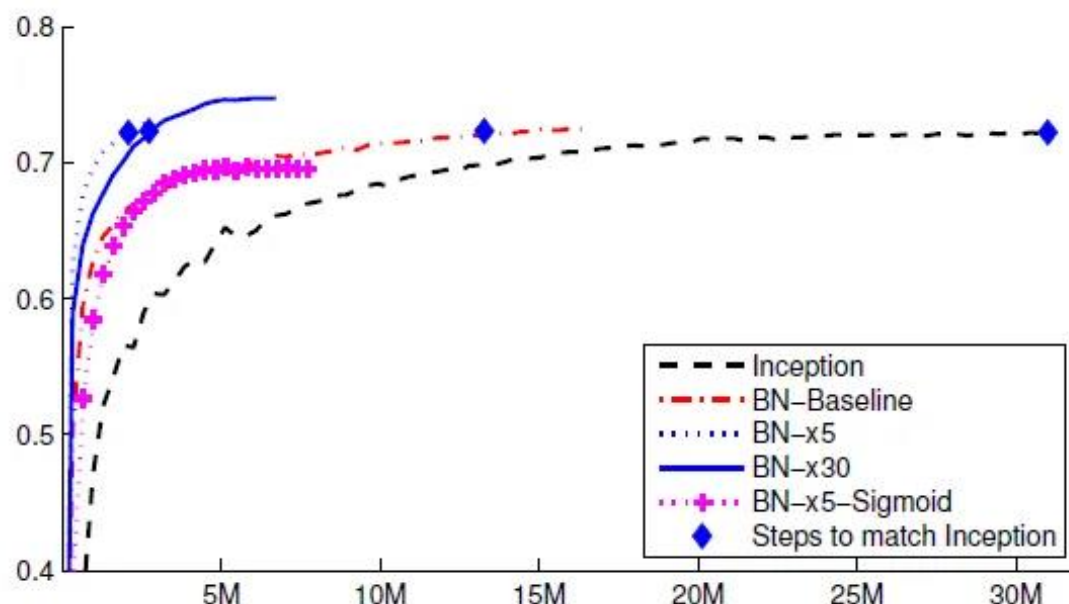
Inception module with dimension reduction

Total: 358M ops

Batch Normalization

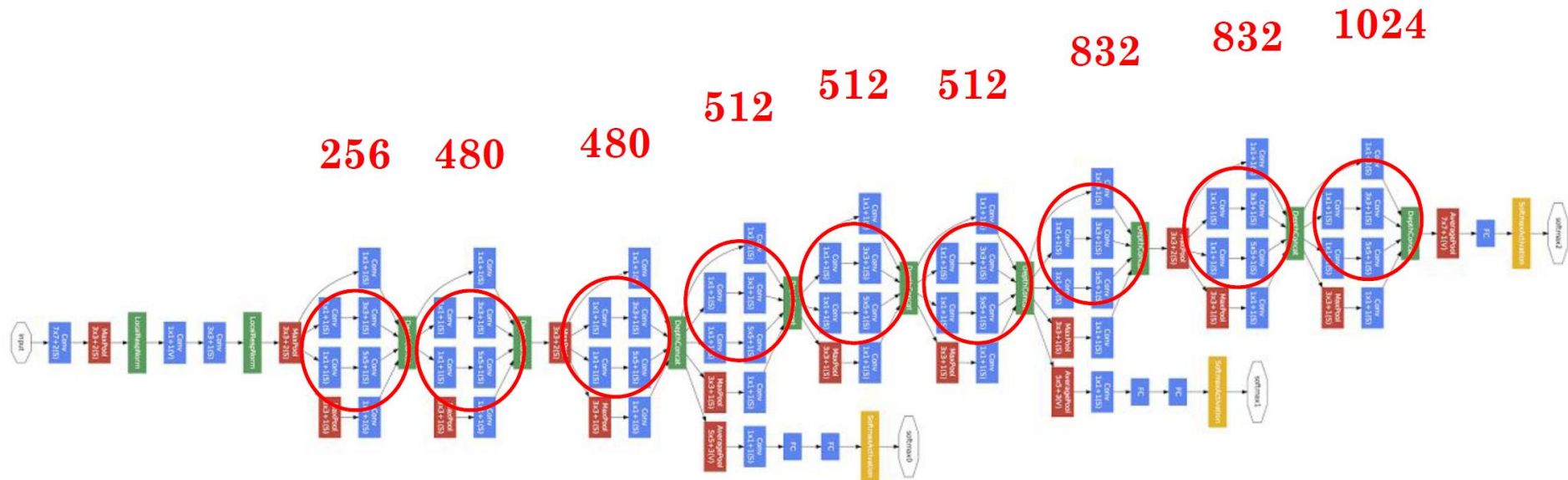
74

- BatchNorm helps mitigate the internal covariate shift. This helps stabilize and accelerate the training process by mitigating issues like vanishing/exploding gradients.



- **Inception:** Inception-v1 without BN
- **BN-Baseline:** Inception with BN
- **BN-x5:** Initial learning rate is increased by a factor of 5 to 0.0075
- **BN-x30:** Initial learning rate is increased by a factor of 30 to 0.045
- **BN-x5-Sigmoid:** BN-x5 but with Sigmoid

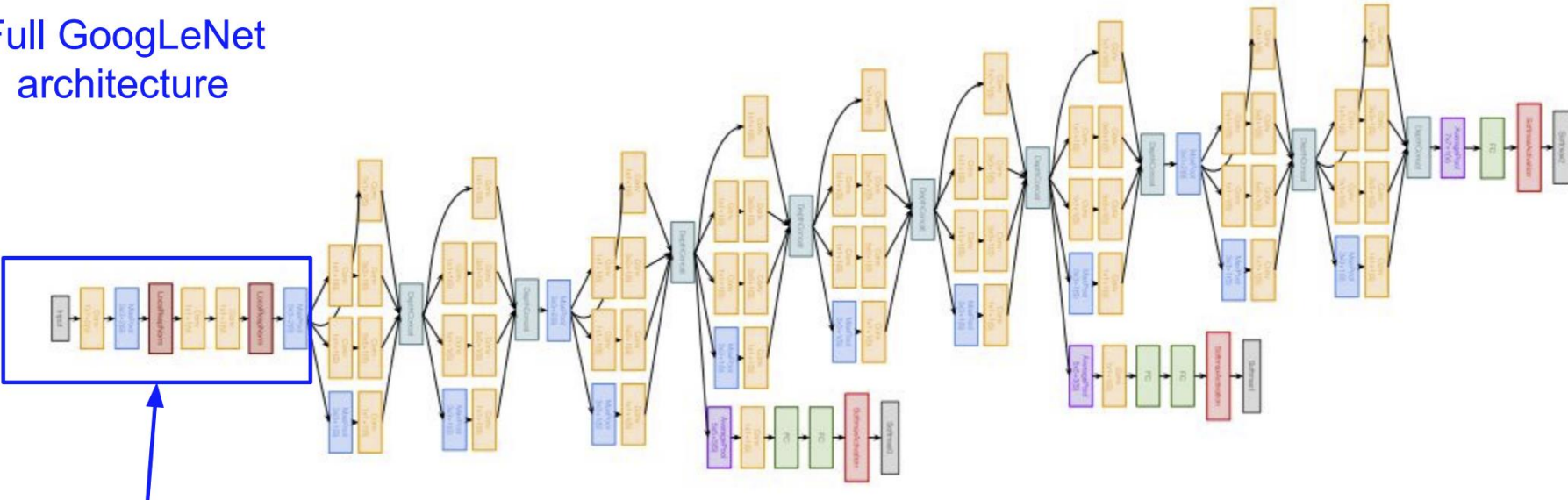
Since the introduction of GoogLeNet, Batch Normalization has become a standard component in many deep learning architectures, providing benefits in terms of training stability, convergence speed, and generalization performance.



Width of inception modules ranges from 256 filters to 1024.
Can remove fully connected layers on top completely.

Full GoogLeNet Arch.

Full GoogLeNet architecture



Stem Network:
Conv-Pool-
2x Conv-Pool

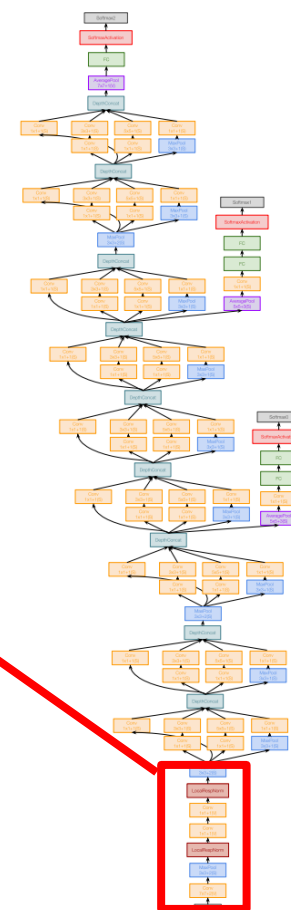
GoogLeNet: Aggressive Stem

Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

Layer	Input size		Layer				Output size				params	flop (M)
	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	(K)		
conv	3	224	64	7	2	3	64	112	3136	9	118	
max-pool	64	112		3	2	1	64	56	784	0	2	
conv	64	56	64	1	1	0	64	56	784	4	13	
conv	64	56	192	3	1	1	192	56	2352	111	347	
max-pool	192	56		3	2	1	192	28	588	0	1	

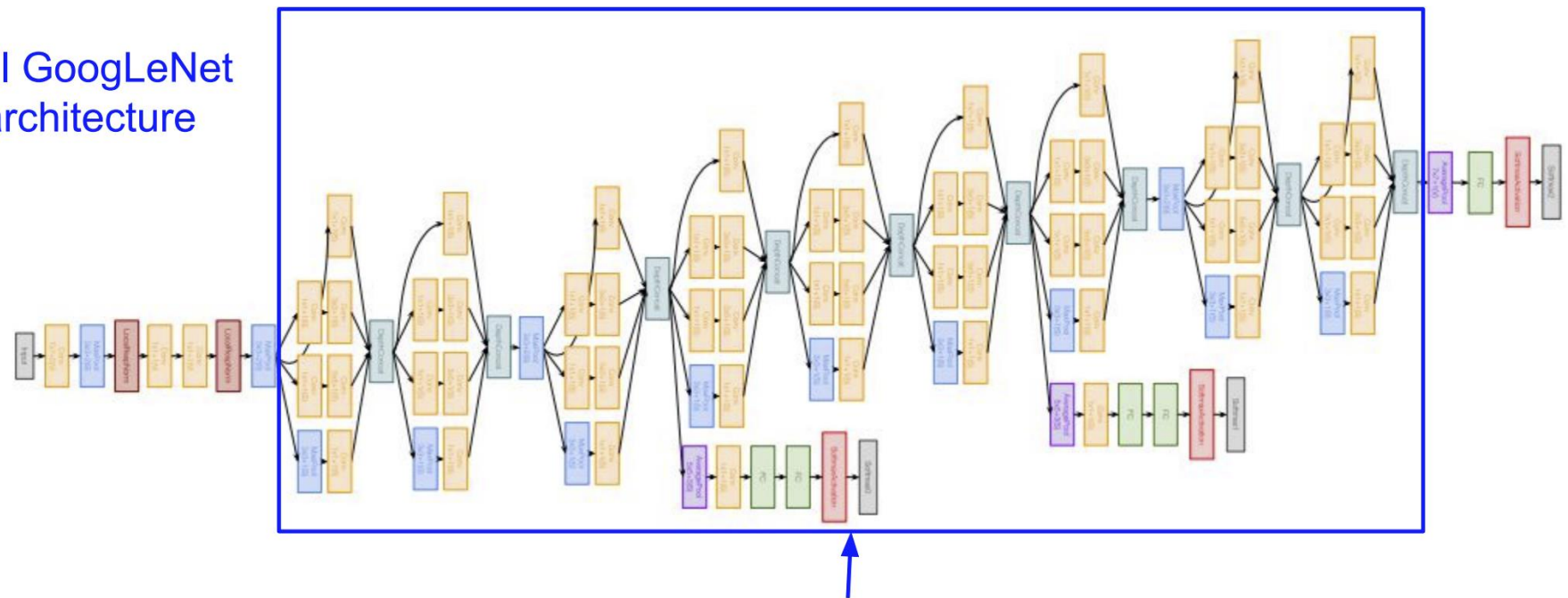
Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)



Full GoogLeNet Arch.

Full GoogLeNet architecture



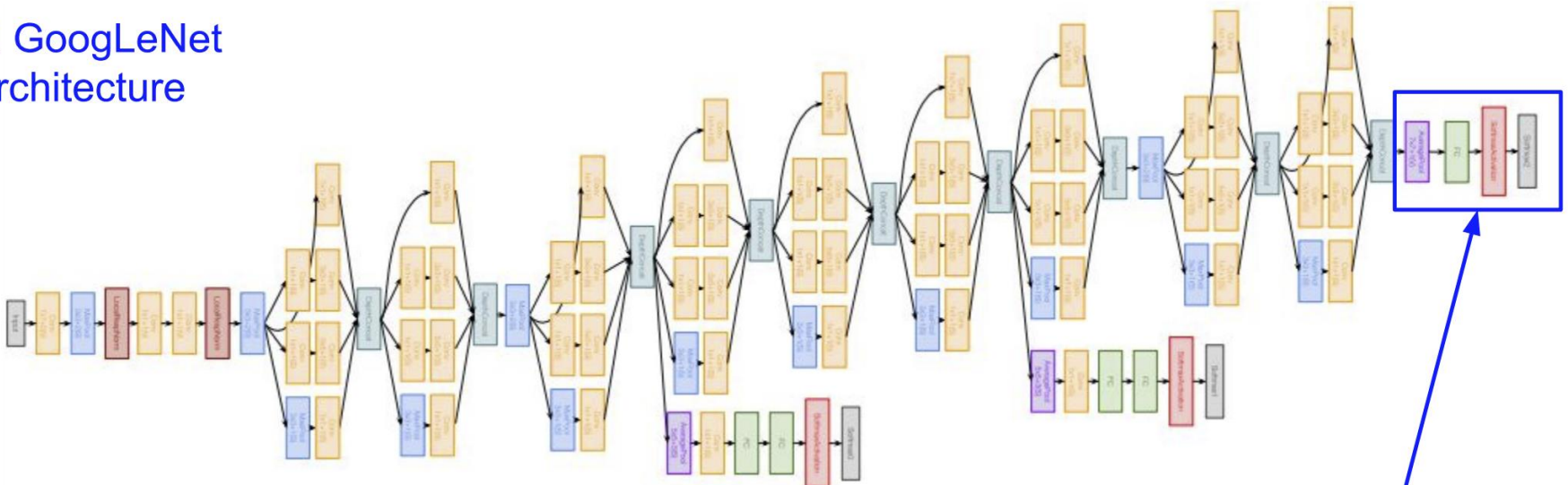
Stacked Inception Modules

Auxiliary Classifiers

- Training using loss at the end of the network didn't work well: Network is too deep, gradients don't propagate cleanly
- Auxiliary classifiers are used in GoogLeNet for two main reasons:
 - ▣ To improve the convergence of the network. Auxiliary classifiers provide additional gradients to the lower layers of the network, which can help the network to converge to a better solution more quickly.
 - ▣ To regularize the network and prevent overfitting. Auxiliary classifiers act as regularizers by forcing the network to learn to classify images at different levels of abstraction. This can help to prevent the network from overfitting to the training data.
- In GoogLeNet, the auxiliary classifiers are each composed of a global average pooling layer, a fully connected layer, and a softmax layer.
- On the ImageNet dataset, GoogLeNet with auxiliary classifiers achieves an accuracy of 93.6%, which is significantly higher than the 89.3% accuracy achieved by GoogLeNet without auxiliary classifiers.

Full GoogLeNet Arch.

Full GoogLeNet architecture



Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

Classifier output

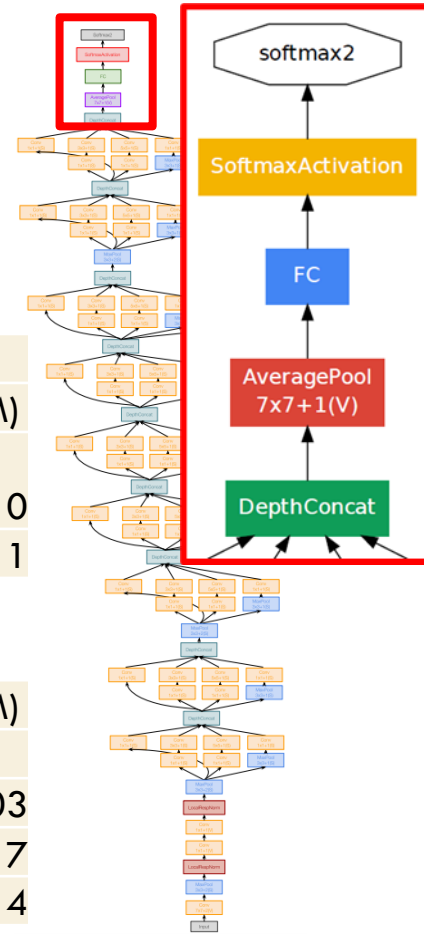
GoogLeNet: Global Average Pooling

- No large FC layers at the end! Instead uses “global average pooling” to collapse spatial dimensions, and one linear layer to produce class scores
- (Recall VGG-16: Most parameters were in the FC layers!)

Layer	Input size		Layer				Output size		memory (KB)	params (k)	flop (M)
	C	H/W	filters	kernel	stride	pad	C	H/W			
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1

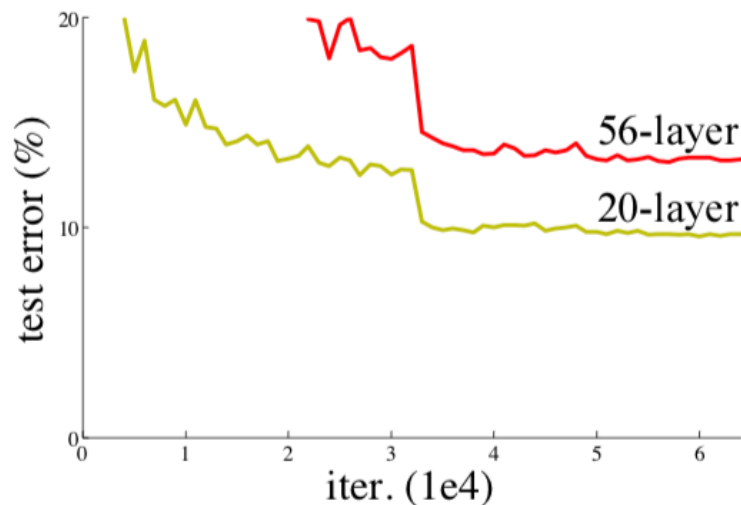
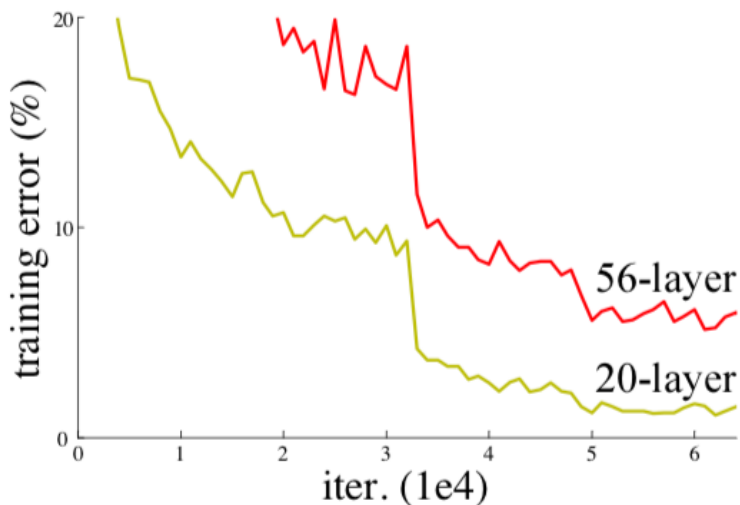
Compare with VGG-16:

Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4



Training very Deep Models

- Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?
 - ▣ Deeper model does worse than shallow model!
 - ▣ Initial guess: Deep model is **overfitting** since it is much bigger than the other model
 - ▣ In fact the deep model seems to be **underfitting** since it also performs worse than the shallow model on the training set! It is actually **underfitting**



Residual Networks

- Residual Networks, often referred to as ResNets, are a type of deep neural network architecture designed to address the challenges of training very deep networks.
- Residual networks (ResNets) are a type of neural network architecture that introduced the concept of skip connections.
- A skip connection is a connection that allows information to flow from one layer of the network to a later layer without passing through any intermediate layers.
- The basic idea behind ResNets is to learn residual functions instead of direct mappings. A residual function is a function that learns the difference between the desired output and the input.
- ResNets have been shown to achieve state-of-the-art results on a wide range of tasks, including image classification, object detection, and segmentation. They are now one of the most popular types of neural network architectures in use.

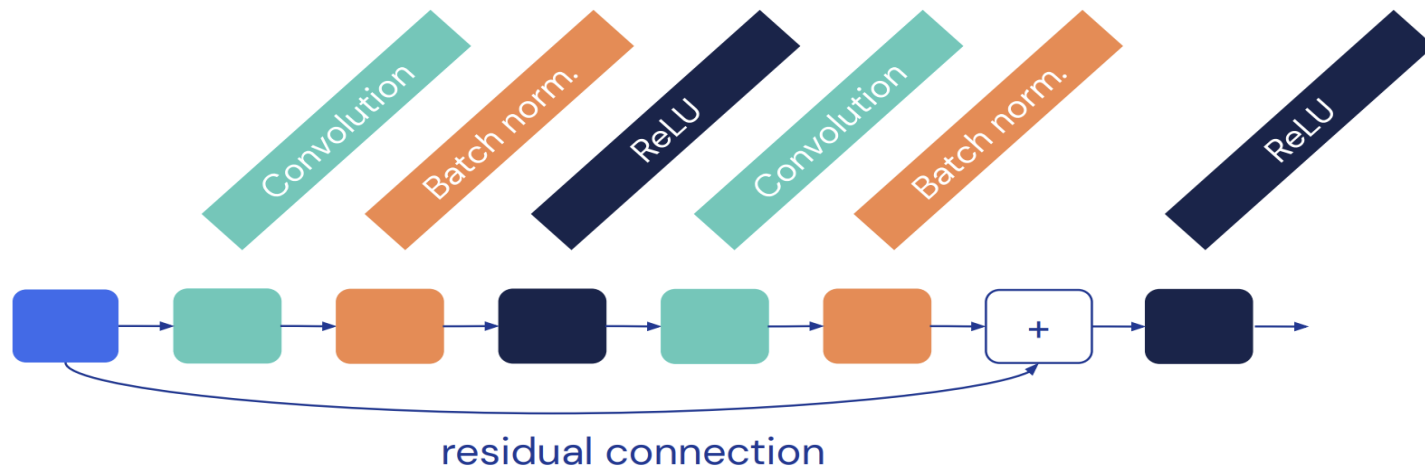
Residual Networks

□ Advantages of Residual Networks:

- It makes it easier to train deep networks.
 - Skip connections help to alleviate the vanishing gradient problem, which can make it difficult to train deep networks.
- It allows the network to learn more complex mappings.
 - By learning residual functions, the network can learn to modify the input in more subtle ways, which can lead to better performance on complex tasks.
- Scalability
 - ResNets are scalable to deep architectures, making them suitable for state-of-the-art models in computer vision and other domains.
- It makes the network more robust to noise.
 - Skip connections allow the network to bypass noisy or irrelevant features in the input, which can lead to better performance on noisy datasets.

ResNet [He et al., 2016]

- **ResNet** (2016): is the residual network which features special skip (residual) connections and a heavy use of **batch normalization** layer. The residual connections facilitate training deep networks.



- Able to train very deep networks
- Deeper networks do better than shallow networks (as expected)
- Swept 1st place in all ILSVRC and
- COCO 2015 competitions
- Still widely used today!

MSRA @ ILSVRC & COCO 2015 Competitions

• 1st places in all five main tracks

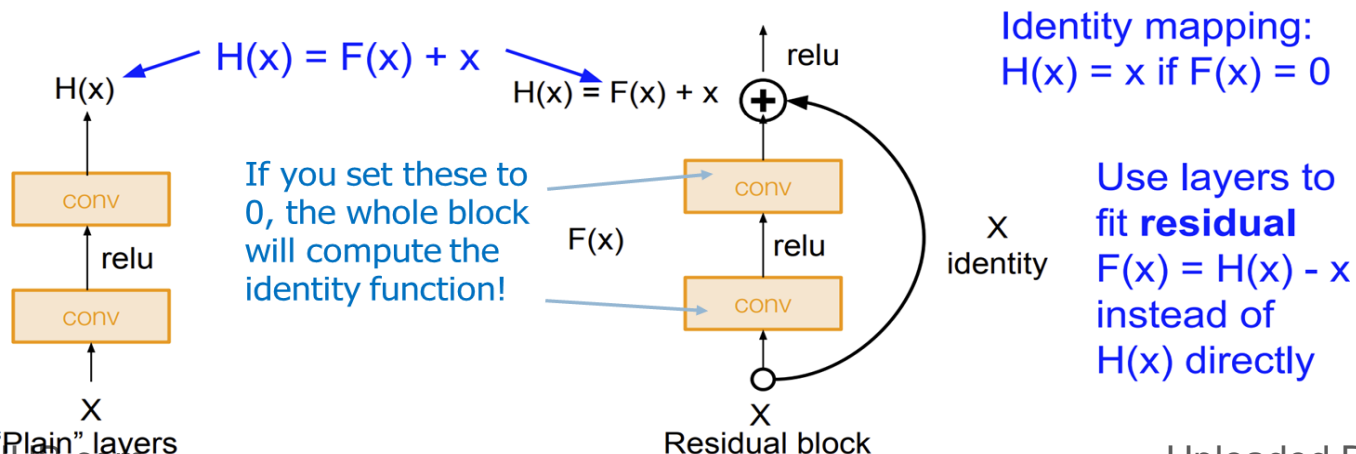
- ImageNet Classification: "Ultra-deep" (quote Yann) 152-layer nets
- ImageNet Detection: 16% better than 2nd
- ImageNet Localization: 27% better than 2nd
- COCO Detection: 11% better than 2nd
- COCO Segmentation: 12% better than 2nd

Uploaded By: anonymous

Key Concepts of Residual Networks

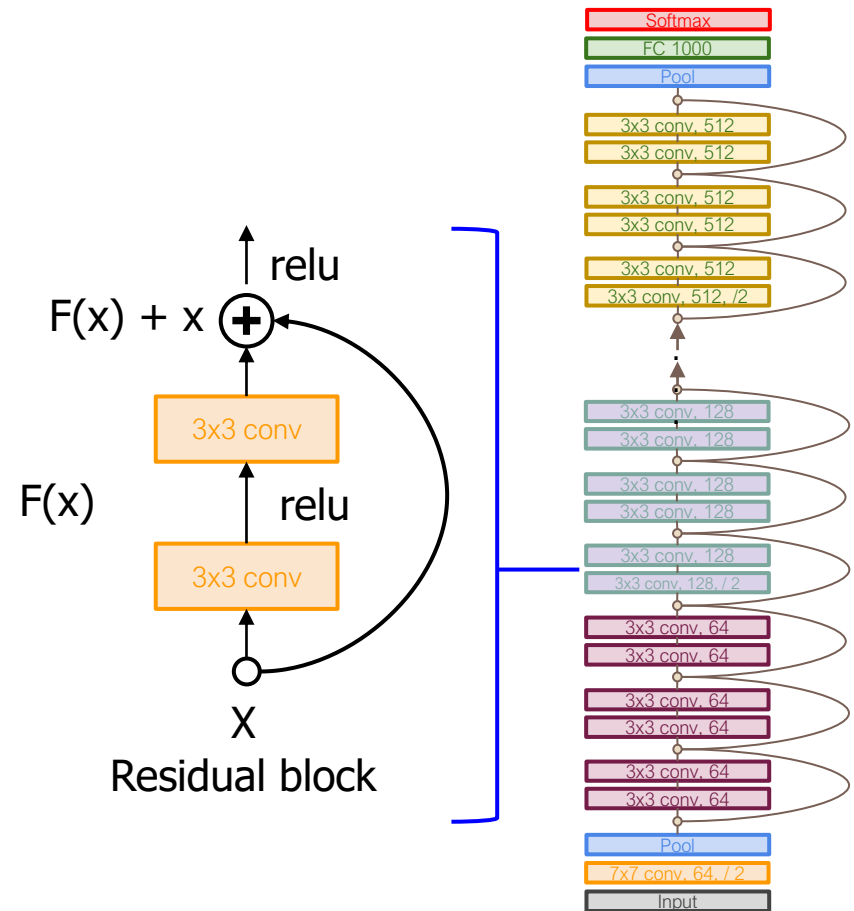
Residual Blocks:

- The basic building block of a ResNet is the residual block.
- A residual block consists of a shortcut connection (skip connection) that bypasses one or more layers and is added to the output of those layers.
- Mathematically, for a residual block with input x and output $H(x)$, the block computes $H(x)-x$, and the final output is $F(x)=H(x)+x$.
- Instead of directly learning the mapping from input to output, a residual block learns the residual (difference) between the input and the desired output.
- Residual blocks enable the learning of residual mappings, making it easier to train very deep networks.
 - This is because the residual is the difference between the desired output and the input, which is a smaller and simpler function to learn than the entire mapping.



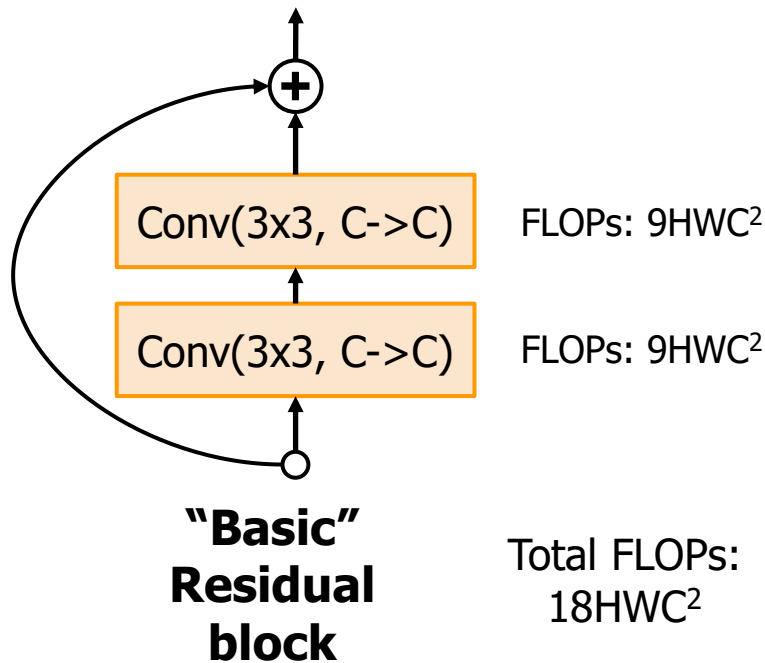
Key Concepts of Residual Networks

- A residual network is a stack of many residual blocks
- Regular design, like VGG: each residual block has two 3x3 conv
- Network is divided into stages: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels

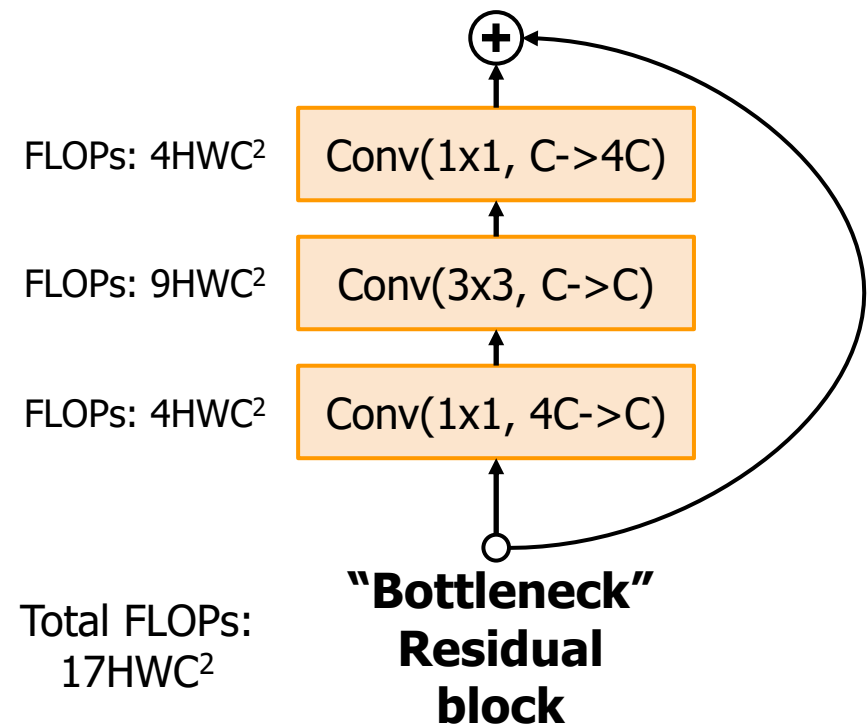


Key Concepts of Residual Networks

Bottleneck Residual block



More layers, less computational cost!



Key Concepts of Residual Networks

□ Skip Connections (Identity Shortcuts):

- Skip connections allow the gradient to flow directly through the network without passing through the intermediate layers.
- Help to alleviate the vanishing gradient problem by allowing the gradients to bypass some of the layers of the network. This allows the gradients to flow more easily through the network, which makes it easier for the network to learn the weights of all of the layers.

□ Deep Network Architecture:

- ResNets are characterized by their deep architectures, often with hundreds of layers.
- The use of residual blocks and skip connections enables the training of deeper networks without suffering from degradation in performance.

□ Batch Normalization:

- Batch normalization is commonly used in ResNets to stabilize and accelerate training.

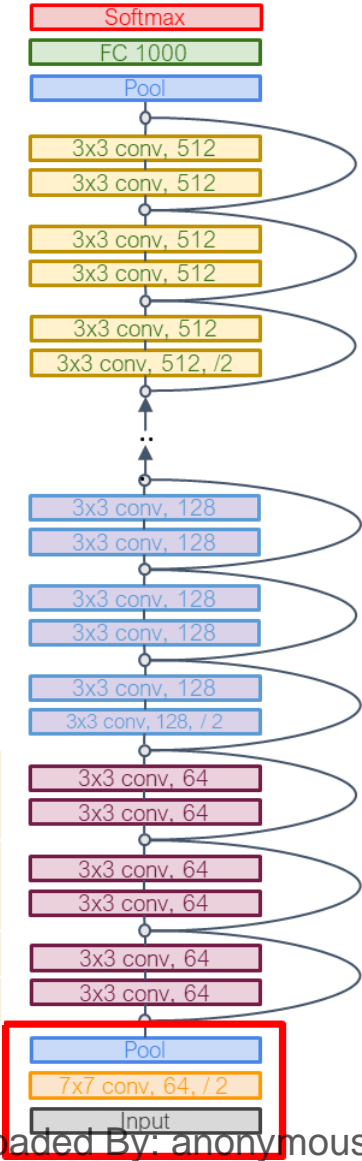
□ Global Average Pooling (GAP):

- ResNets often use Global Average Pooling (GAP) as an alternative to fully connected layers for dimensionality reduction at the end of the network.

Key Concepts of Residual Networks

- Uses the same aggressive **stem** as GoogLeNet to downsample the input 4x before applying residual blocks:
- Like GoogLeNet, no big fully-connected-layers: instead use **global average pooling** and a single linear layer at the end

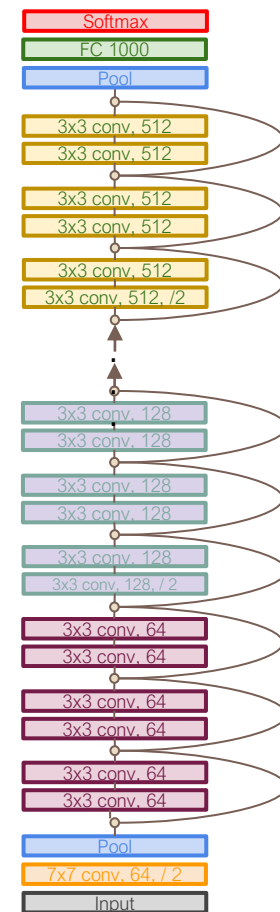
Layer	Input size		Layer				Output size		memory (KB)	params (k)	flop (M)
	C	H/W	filters	kernel	stride	pad	C	H/W			
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2



ResNet Architectures

- ResNets are scalable to different depths, and variants like ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152 have been introduced with varying numbers of layers.
- ResNet-50 is the same as ResNet-34, but replaces Basic blocks with Bottleneck Blocks. This is a great baseline architecture for many tasks even today!
- Deeper ResNet-101 and ResNet-152 models are more accurate, but also more computationally heavy

	Block type	Stem layer s	Stage 1		Stage 2		Stage 3		Stage 4		FC layers	GFLO P	ImageNet top-5 error
			Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers			
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58
ResNet-50	Bottle	1	3	9	4	12	6	18	3	9	1	3.8	7.13
ResNet-101	Bottle	1	3	9	4	12	23	69	3	9	1	7.6	6.44
ResNet-152	Bottle	1	3	9	8	24	36	108	3	9	1	11.3	5.94



Other CNN Networks

□ DenseNet

- DenseNet is another deep residual neural network architecture that is known for its efficiency and accuracy.
- DenseNet has achieved state-of-the-art results on a variety of computer vision tasks, including image classification, object detection, and semantic segmentation.

□ EfficientNet

- EfficientNet is a family of efficient CNN architectures that are designed to achieve high accuracy with minimal computational resources.
- EfficientNet has achieved state-of-the-art results on a variety of computer vision tasks, including image classification, object detection, and semantic segmentation.

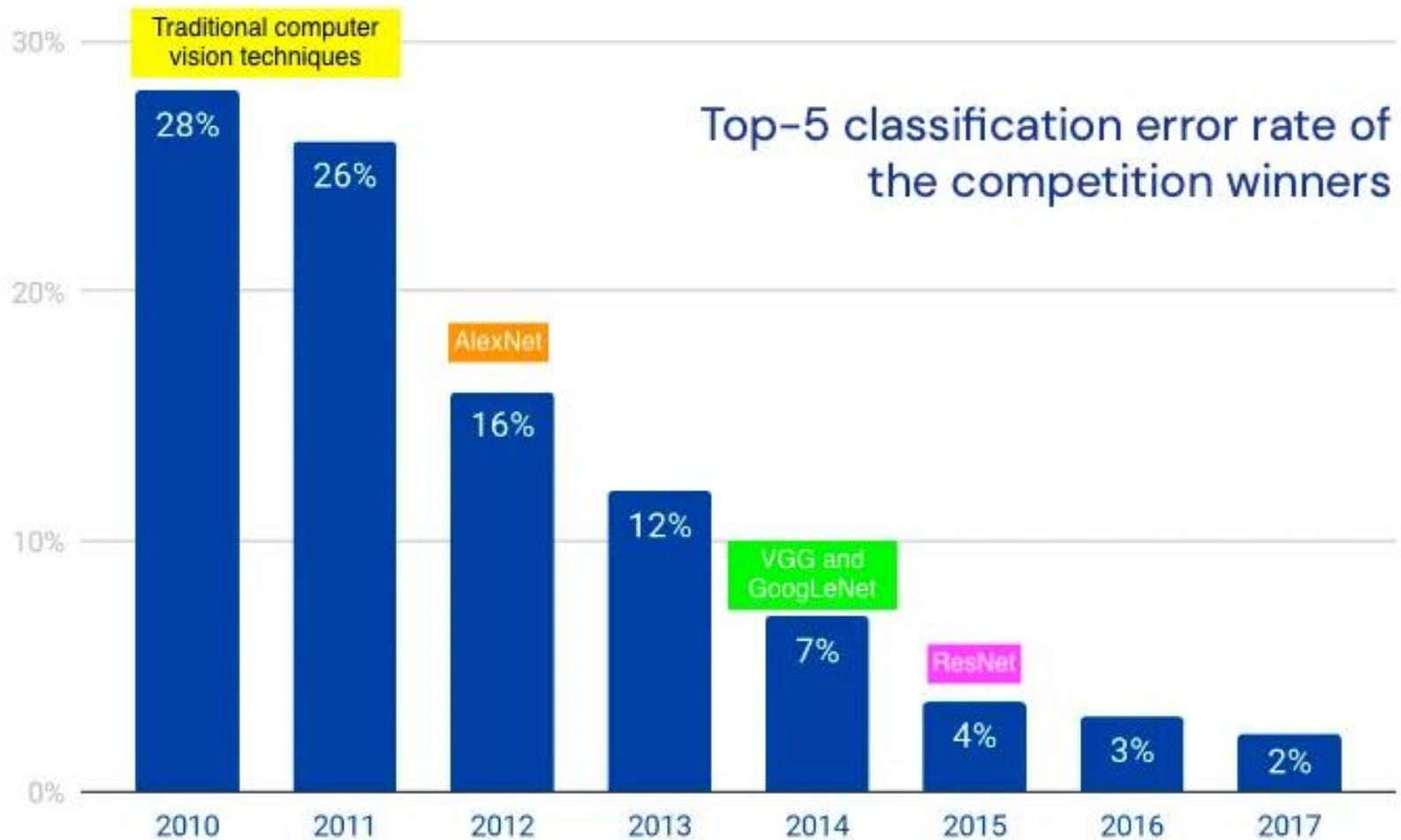
□ MobileNetV2:

- Designed for mobile and edge devices, offering a good balance between accuracy and computational efficiency.
- Uses depthwise separable convolutions to reduce the number of parameters.

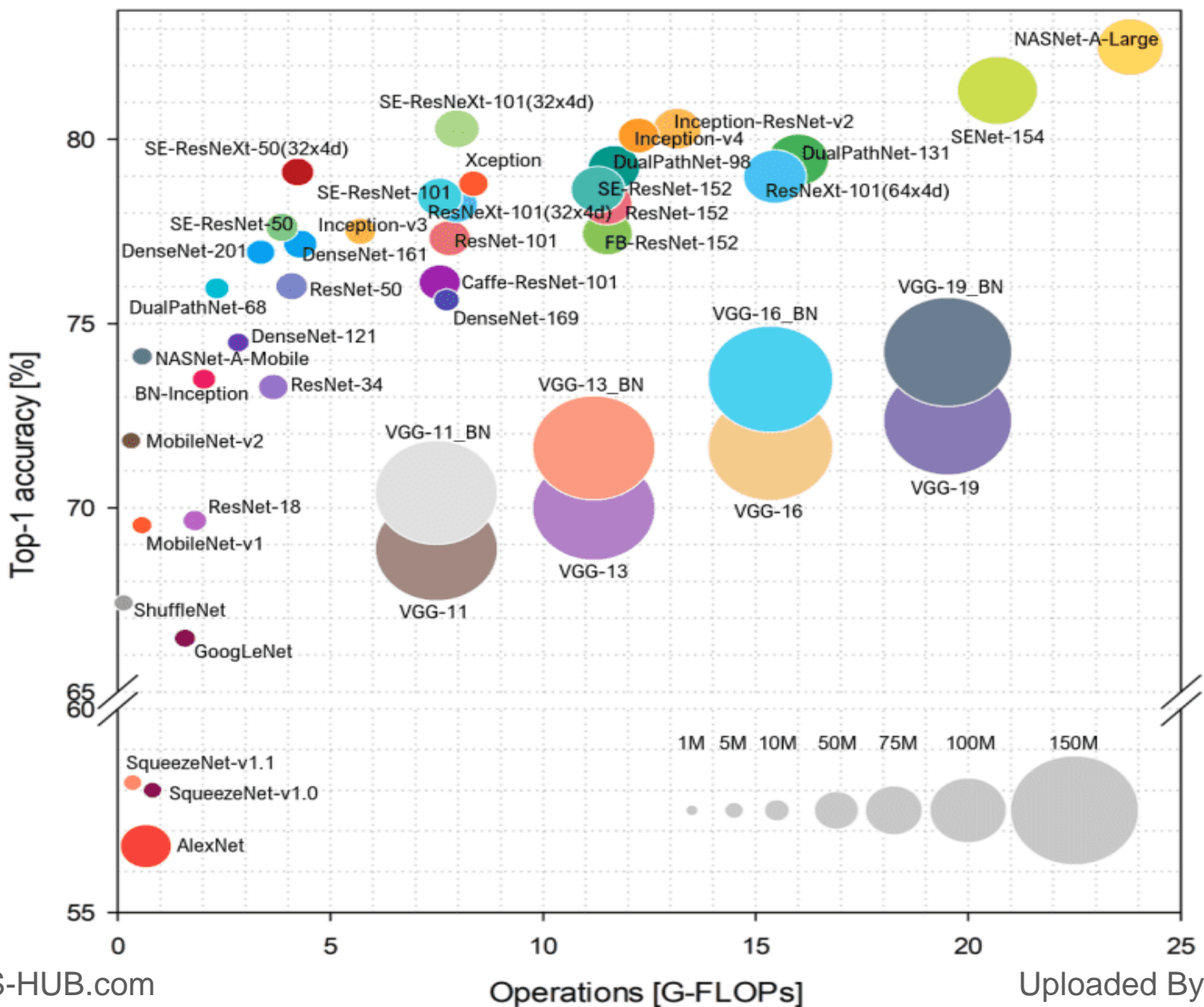
□ Vision Transformer (ViT)

- ViT is a recent CNN architecture that has achieved state-of-the-art results on a variety of computer vision tasks, including image classification, object detection, and semantic segmentation.
- ViT is based on the transformer architecture, which was originally developed for natural language processing.

ImageNet Competition Winners

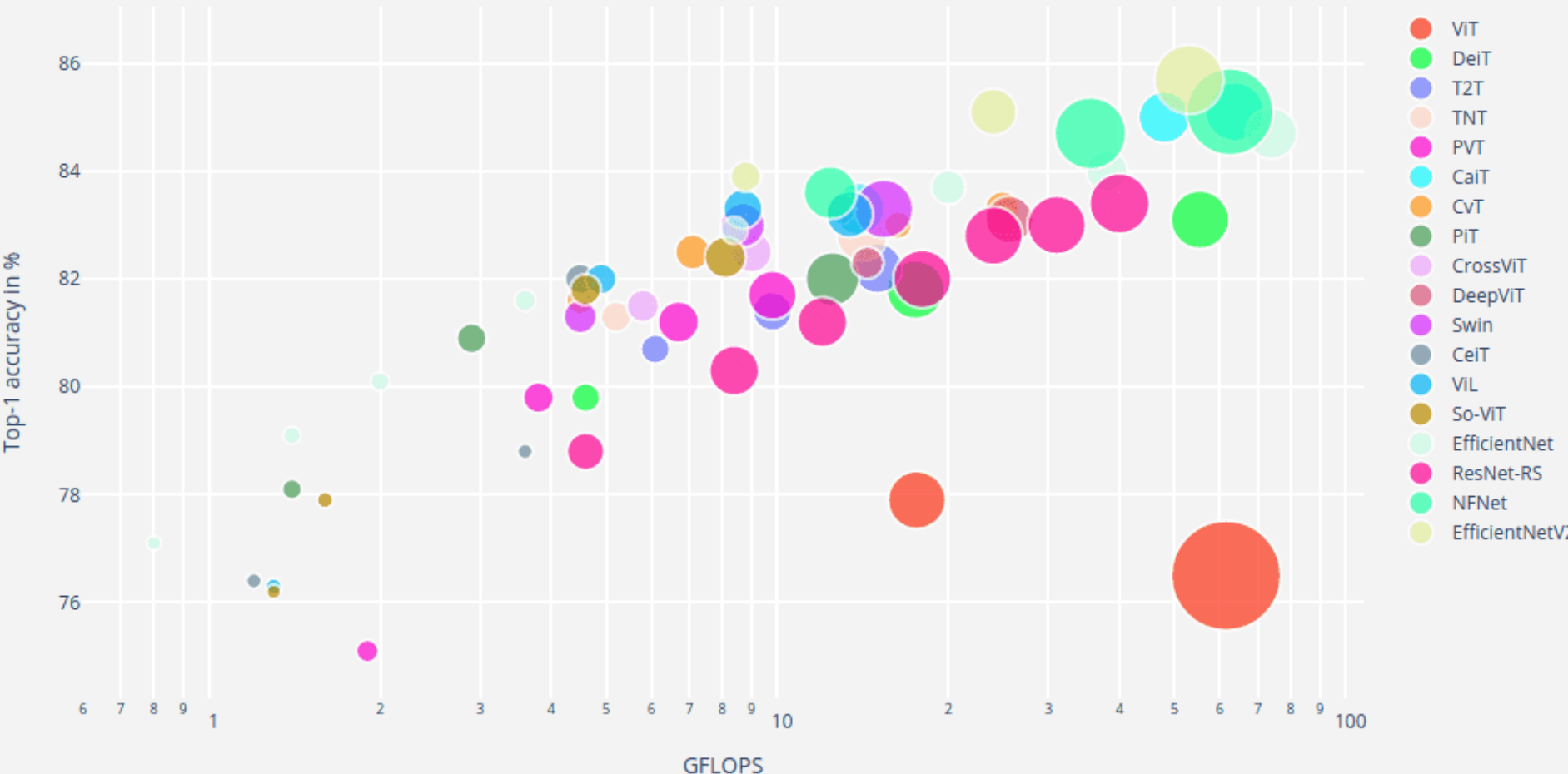


Comparing Well-Known Archs.



CNN vs ViT

ImageNet top-1 accuracy vs FLOPS vs parameters



CNN Architectures Summary

- Early work (AlexNet -> ZFNet -> VGG) shows that **bigger networks work better**
- GoogLeNet one of the first to focus on **efficiency** (aggressive stem, 1x1 bottleneck convolutions, global avg pool instead of FC layers)
- ResNet showed us how to train extremely deep networks – limited only by GPU memory! Started to show diminishing returns as networks got bigger
- After ResNet: **Efficient networks** became central: how can we improve the accuracy without increasing the complexity?
- Lots of **tiny networks** aimed at mobile devices: MobileNet, ShuffleNet, etc
- **Neural Architecture Search** promises to automate architecture design

Which Architecture should I use?

- ❑ **Don't be a hero.** For most problems you should use an off-the-shelf architecture; don't try to design your own!
- ❑ If you just care about accuracy, **ResNet-50** or **ResNet-101** are great choices
- ❑ If you want an efficient network (real-time, run on mobile, etc) try **MobileNets** and **ShuffleNets**

Data Augmentation

98

- Data augmentation is a technique used to artificially increase the size of a training dataset by applying various transformations to the existing data.
- Data augmentation encodes invariances in your model
 - ▣ Think for your problem: what changes to the image should not change the network output?
 - ▣ By design, convnets are only robust against translation.
 - ▣ Data augmentation makes them robust against other transformations: rotation, scaling, shearing, warping, ...
- Purpose:
 - ▣ **Increased Diversity:** Introduces diversity into the training set, preventing the model from memorizing specific examples.
 - ▣ **Robustness to Variability:** Trains the model to be more robust to variations and transformations that might occur in real-world scenarios.
 - ▣ **Regularization:** Acts as a form of regularization, helping prevent overfitting by making the model less sensitive to minor variations in the training data.

Data Augmentation Example

99



Original



Horizontal Flip



Vertical Flip



Horizontal + Vertical



Color Profile 1



Color Profile 2



Color Profile 3



Color Profile 4



Rotate Left



Rotate Right



Noise 1



Noise 2



Crop 1



Crop 2



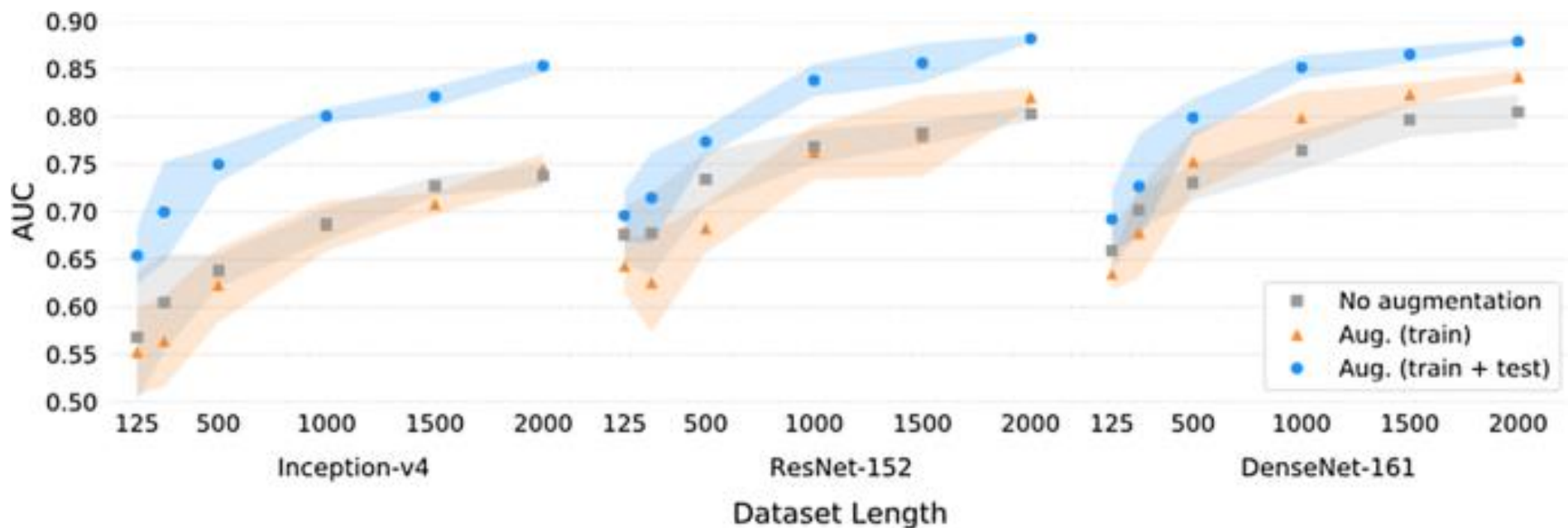
Resize 1



Resize 2

Effect on Data Augmentation

100



Transfer Learning

101

- You need a lot of a data if you want to train/use CNNs
- Training a Network From Scratch
 - ▣ Time
 - ▣ Compute
 - ▣ Training data — the more, the better. Models benefit significantly from A LOT of data — especially in computer vision, a few thousand examples usually doesn't cut it.
 - ▣ Money for all of the above
- When trained from scratch, a model's parameters are initialized randomly and then updated through some optimization algorithm like gradient descent
- So, if there are two tasks to be solved with deep learning, this process is repeated separately both times.
- However, does it really need to be? Consider the case when we humans learn something new: do we ALWAYS start from the ground up?

Transfer Learning

102

- ❑ Deep learning models are typically trained on large datasets, which can be expensive and time-consuming to collect and label.
- ❑ Transfer learning allows to leverage the knowledge that has already been learned by a pre-trained model on a different task, and apply it to a new task.
- ❑ Transfer learning with CNNs involves leveraging pre-trained models that were initially trained on large image datasets, such as ImageNet, and adapting them for new tasks.
- ❑ How transfer learning is applied with CNNs:
 - ❑ **Task 1: Pre-Trained CNN Models**
 - Models like VGG, ResNet, Inception, and MobileNet are pre-trained on massive datasets, typically for image classification tasks.
 - ❑ **Task 2: Transfer Learning**
 - **Feature Extraction**
 - **Fine-Tuning**
 - **Fine-Tuning the whole network**

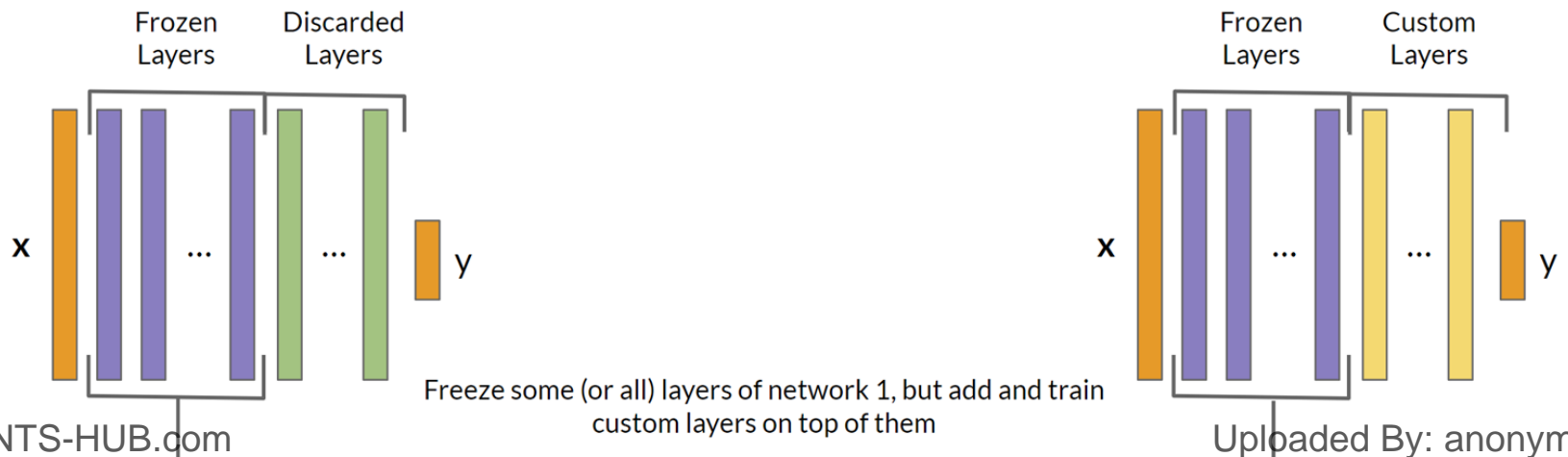
❑ Advantages of Transfer Learning:

1. **Data Efficiency:** Transfer learning enables effective learning with smaller datasets, as the model leverages knowledge from a larger dataset.
2. **Faster Training:** Training a model from scratch can be time-consuming. Transfer learning speeds up training by starting with a pre-trained model.
3. **Improved Performance:** Transfer learning often leads to improved performance compared to training a model from scratch, especially when the pre-trained model has been trained on a similar task.
4. **Generalization:** Transfer learning helps in generalizing knowledge gained from one domain to another, tend to generalize well and are less prone to overfitting, especially when applied to related tasks.
5. **Is easy to implement:** Transfer learning is a relatively simple technique to implement. There are many different transfer learning libraries available, such as TensorFlow Hub and PyTorch Torchvision.

Choosing the right transfer learning approach

104

- Choosing the right transfer learning approach depends on various factors, including the amount of data available for the target task, the similarity between the source and target tasks, and computational resources.
- Feature Extraction (Freezing Layers):**
 - When to Choose:** Choose feature extraction when you have a limited amount of data for the target task, and the features learned by the pre-trained model are expected to be relevant to the new task.
 - How it Works:** Remove the final layers of the pre-trained model, keeping the feature extraction layers. Add new layers (usually fully connected layers) for the target task. The pre-trained layers are frozen, and only the added layers are trained.

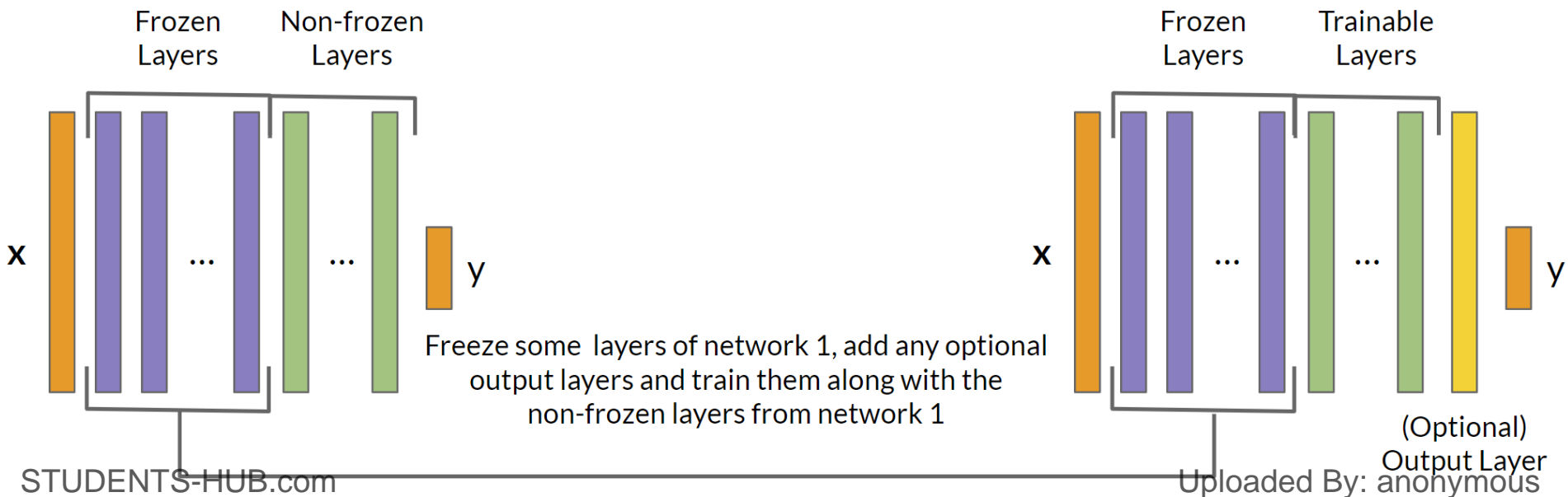


Choosing the right transfer learning approach

105

□ Fine-Tuning:

- **When to Choose:** Choose fine-tuning when you have a moderate data for the target task, and you expect the pre-trained model to adapt well to the new task.
- **How it Works:** Instead of discarding the non-frozen layers of the **pretrained** network, they are **fine-tuned**, i.e., simultaneously trained further (starting with the same pretrained weights from task 1) on the new data for task 2.



Choosing the right transfer learning approach

106

□ **Fine-Tuning the whole network**

- ▣ **When to Choose:** Choose fine-tuning when you have a big data for the target task, and you expect the pre-trained model to adapt well to the new task.
- ▣ In some cases, it might be favorable to fine-tune the entire pretrained network rather than some subset of layers. This can equivalently be viewed as initializing the second network's parameters to the pretrained network's parameters, instead of the usual random initialization. In other words, a good pretrained network gives you a "head-start" in the training process.

□ **Data Augmentation:** Regardless of the chosen approach, consider incorporating data augmentation techniques to artificially increase the size of the target dataset.

Choosing the right transfer learning approach

107

- Choosing the number of frozen, fine-tunable and custom layers greatly depends on the problem at hand. Nonetheless, here are some suggestions for four common scenarios in which transfer learning is generally applicable:

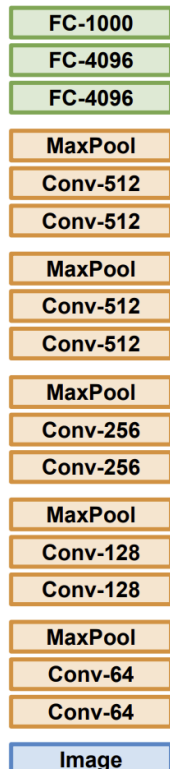
	Large Task 2 Dataset	Small Task 2 Dataset
Task 2 dataset similar to task 1 dataset	Should be ok to fine-tune the entire network.	No need to fine-tune. Fix most of the initial layers and train a linear classifier on top of them.
Task 2 dataset different from task 1 dataset	Should be ok to fine-tune the entire network.	Don't fine-tune. Fix some of the initial layers and train a custom network on top of them.

Transfer Learning with CNNs

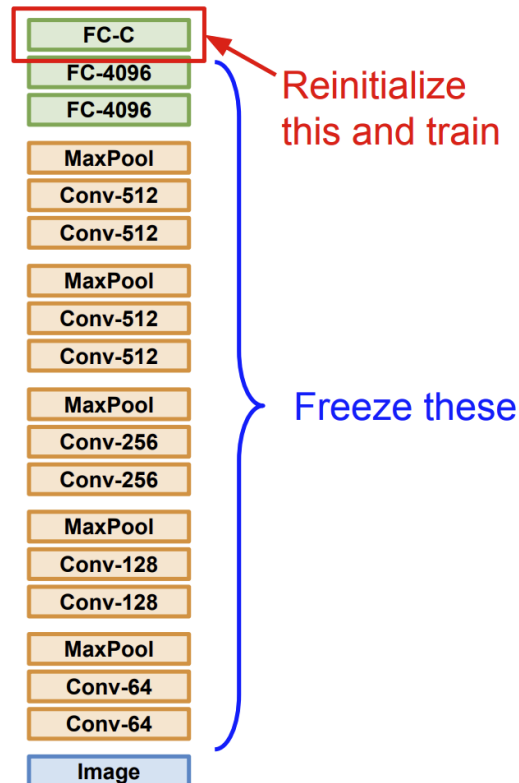
108

Transfer learning with CNNs has become a pervasive and highly effective technique in the field of deep learning, particularly in computer vision tasks.

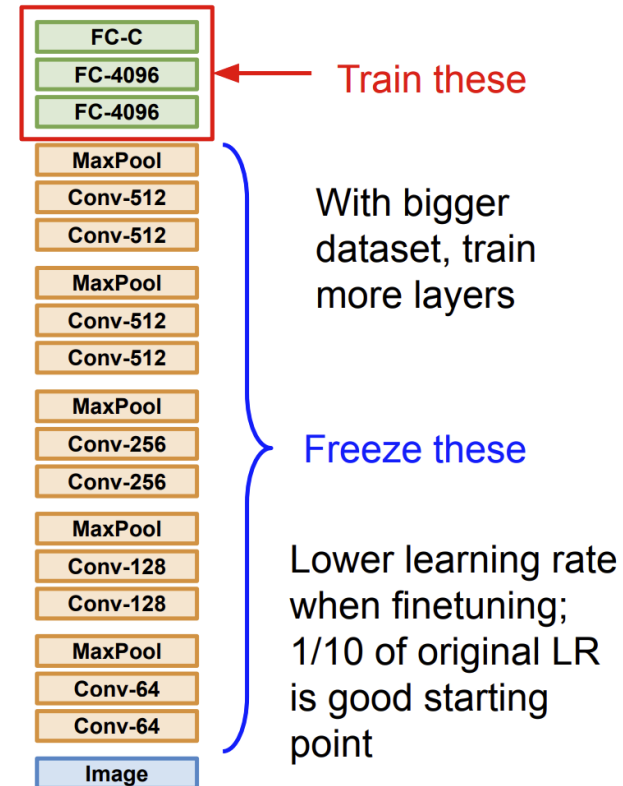
1. Train on Imagenet



2. Small Dataset (C classes)

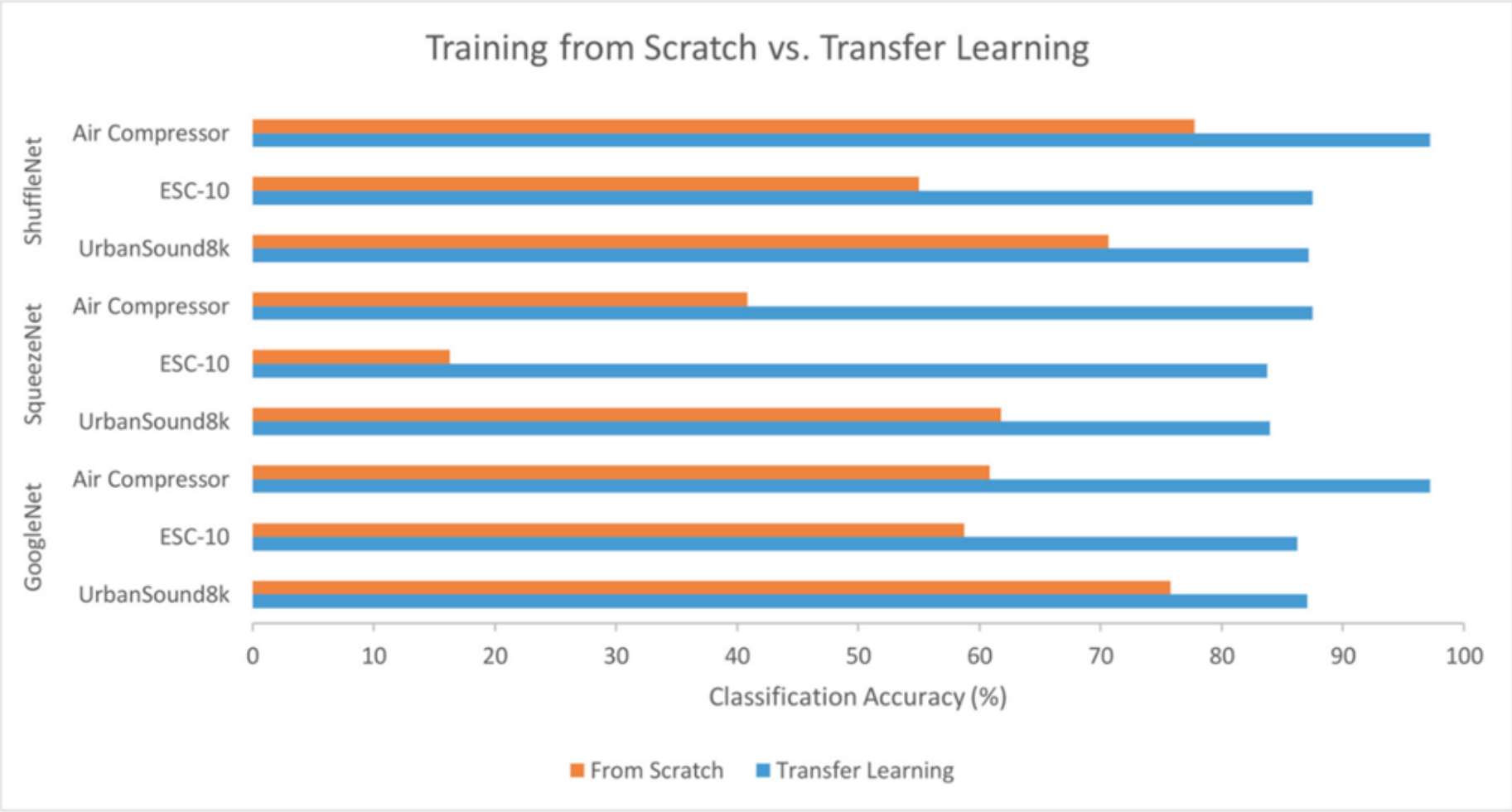


3. Bigger dataset



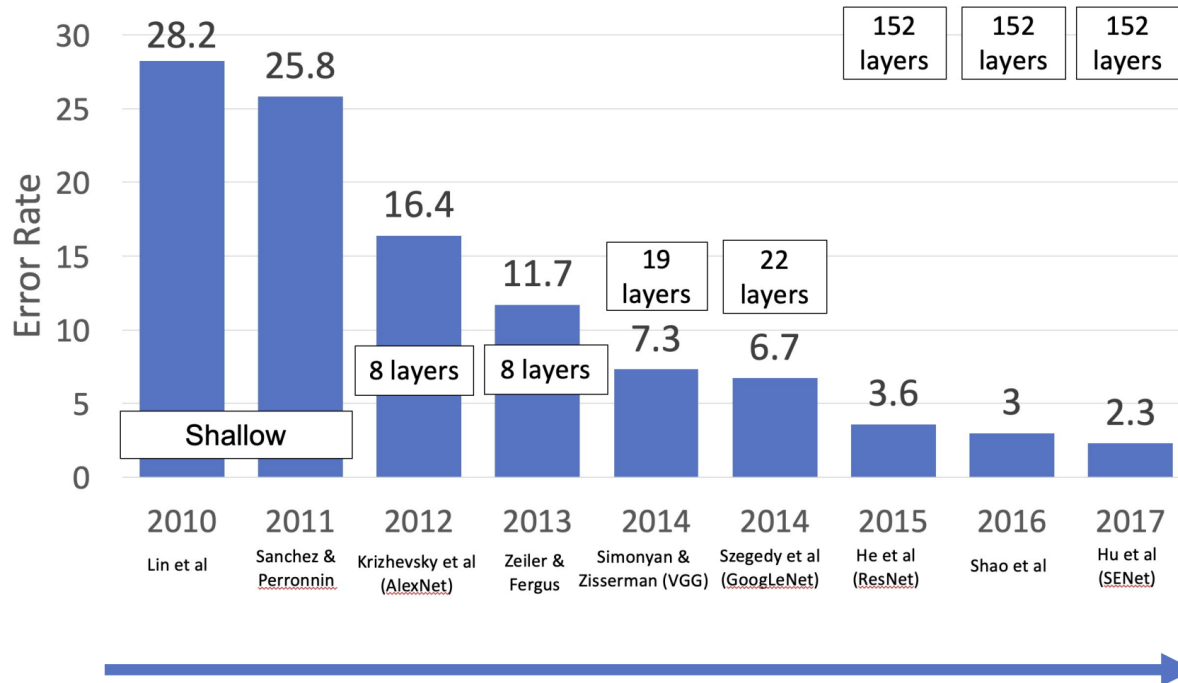
Does Transfer Learning Work?

109



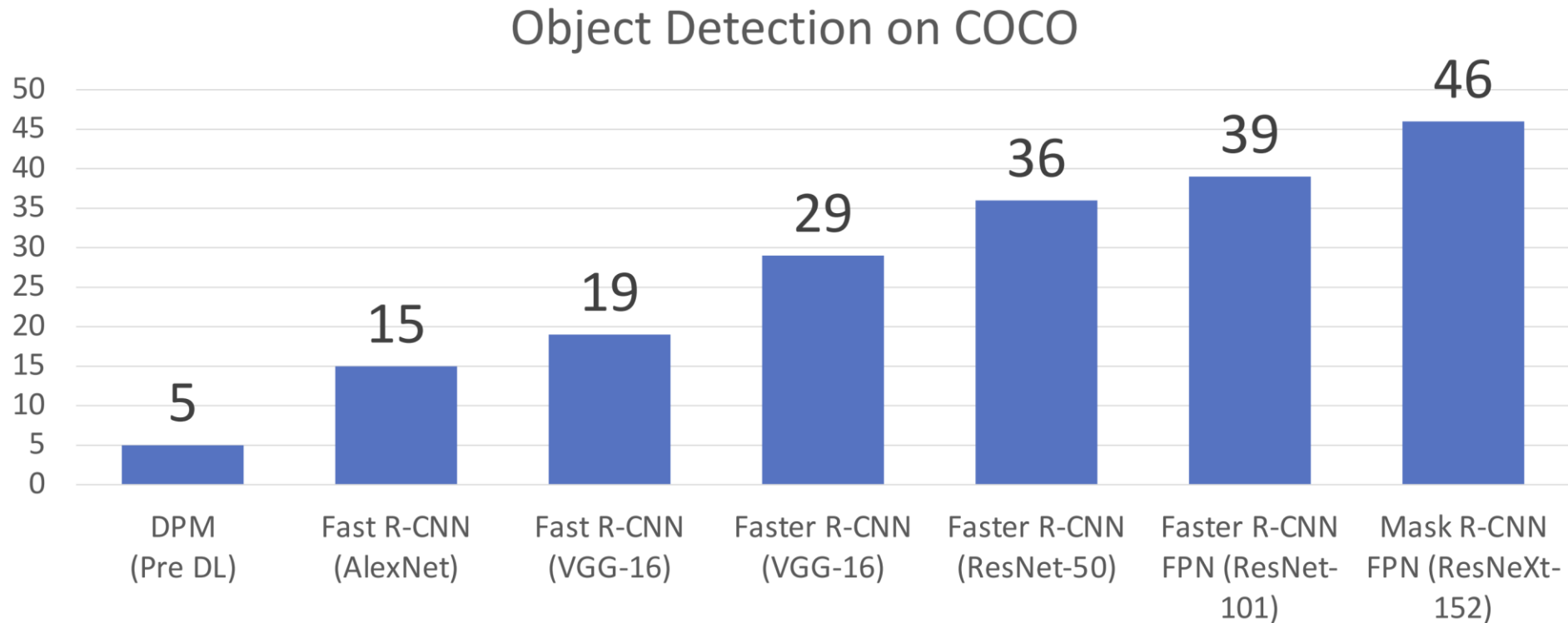
Transfer Learning with CNNs: Architecture Matters!

ImageNet Classification Challenge



Improvements in CNN architectures lead to improvements in many downstream tasks thanks to transfer learning!

Transfer Learning with CNNs: Architecture Matters!



Acknowledgement

113

- The material in these slides are based on:
 - ▣ Digital Image Processing: Rafael C. Gonzalez, and Richard
 - ▣ Forsythe and Ponce: Computer Vision: A Modern Approach
 - ▣ Rick Szeliski's book: Computer Vision: Algorithms and Applications
 - ▣ cs131@ Stanford University
 - ▣ cs131n@ Stanford University
 - ▣ CS198-126@ University of California, Berkely
 - ▣ CAP5415@ University of Central Florida
 - ▣ CSW182 @ University of California, Berkely
 - ▣ Deep Learning Lecture Series @UCL
 - ▣ EECS 498.008 @ University of Michigan
 - ▣ CSE576 @ Washington University
 - ▣ 11-785@ Carnegie Mellon University
 - ▣ CSCI1430@ Brown University
 - ▣ Computer Vision@ Bonn University
 - ▣ ICS 505@ KFUPM
 - ▣ Digital Image Processing@ University of Jordan