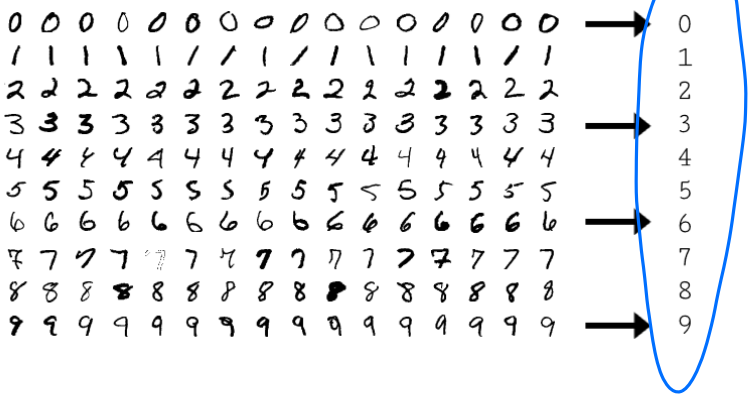
Section 14. Convolutional Neural Network

- Convolution and pooling
- Convolutional Neural Network
- CNN examples
- Pretrained Neural Network

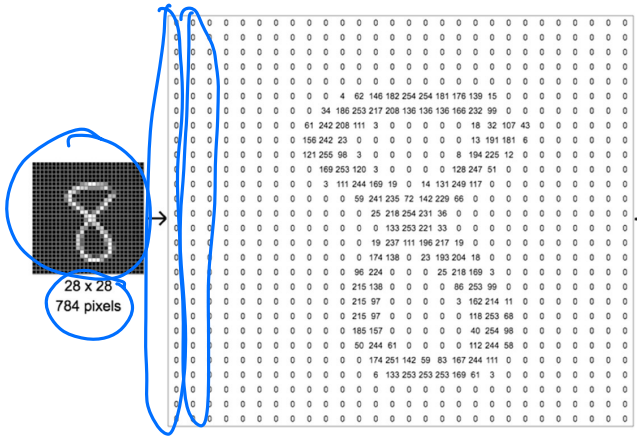
Multi classes classification. Image Representation

Example: hand-written digits taken from US zip codes.

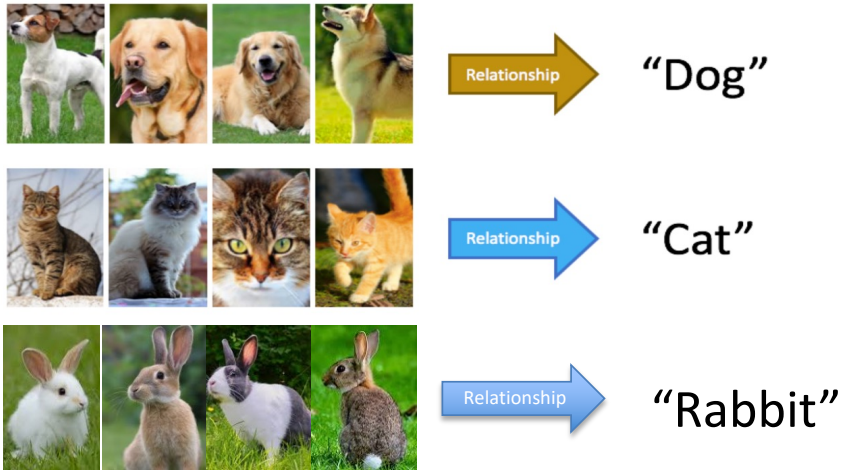
The MNIST (Modified National Institute of Standards and Technology) data set of handwritten numbers. It contains 60,000 training images and 10,000 testing images.



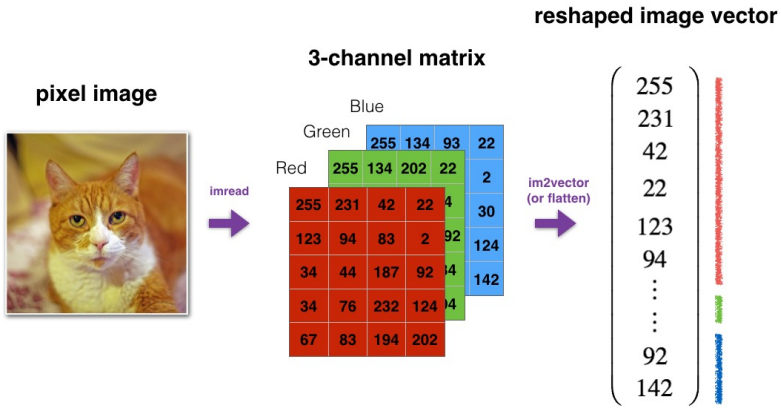
The black and white images from MNIST were normalized to fit into a 28x28 pixels.



- Multi-class classification: Image Classification



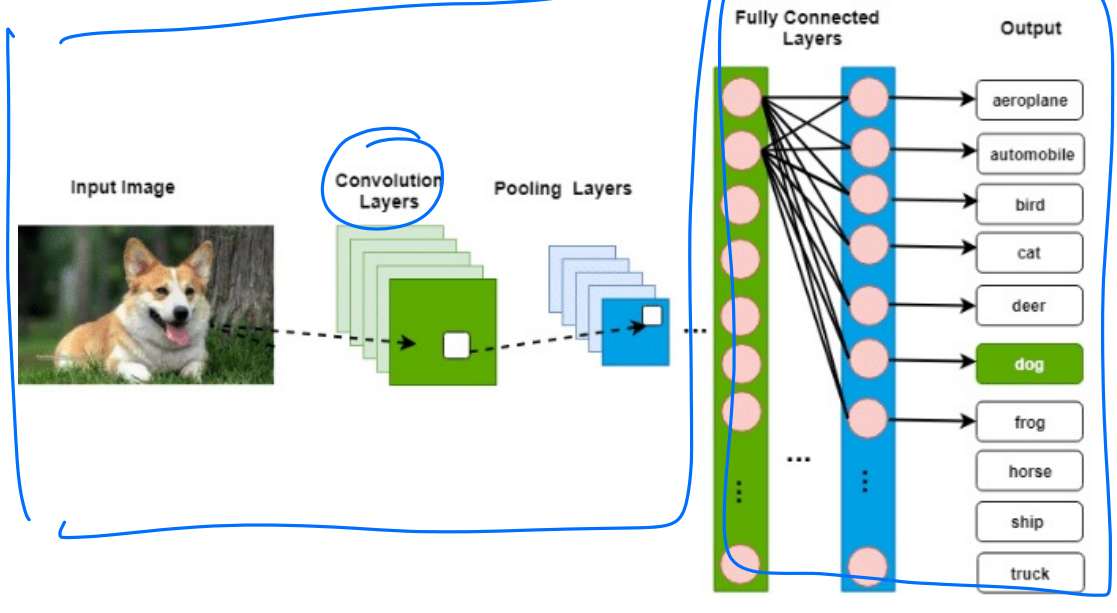
The features represent pixel values.



➢ Convolution Neural Networks (CNN)

CNNs are a specific type of neural networks that are generally composed of **convolution layers** and **pooling layers.**

Typical CNN architecture

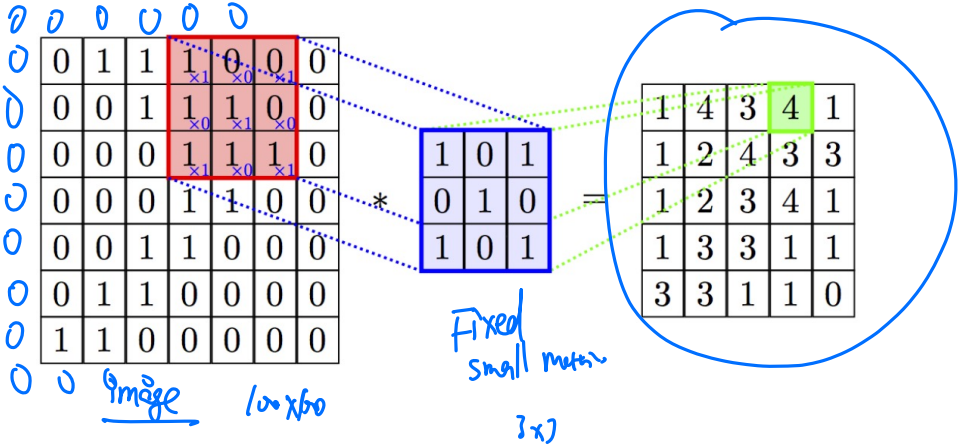


➢ Convolution layers: (generate feature maps.)

Mathematically, a **convolution** combines two matrices to make a third, by taking the **dot product** of the smaller matrix with every block of the larger.

Suppose B is an $m \times n$ matrix.

$$(A * B)_{i,j} = \sum_{s=0}^{m-1} \sum_{r=0}^{n-1} A_{i+s,j+r} B_{s,r}$$



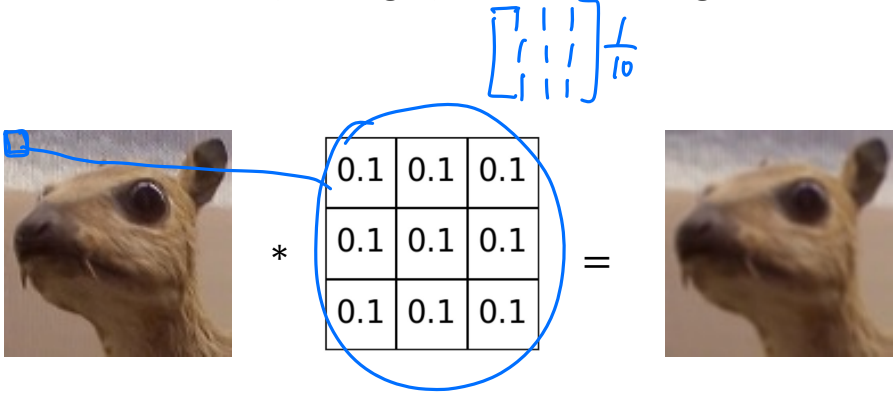
The **Stride** of a Convolutional Filter. (Filter Step Size.)

➢ Convolution examples

In image processing, this is used to create effects and extract information from images.

1. Blurring (Reduces Noise)

Mean filter: The idea of mean filtering is simply to replace each pixel value in an image with the mean ('average') value of its neighbors, including itself.

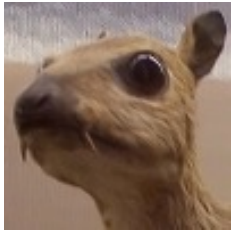


The **median filter** is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image.

2. Sharpening and Finding edges

$0 \rightarrow 2^8 = 256$ $(0, 1)$

Edge detect:

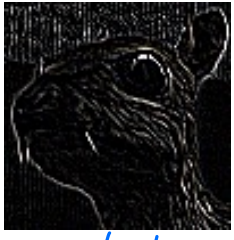


100 x 100

$*$

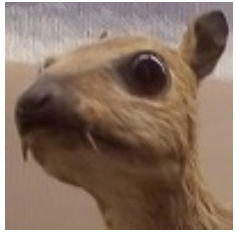
-1	-1	-1
-1	8	-1
-1	-1	-1

$=$



100 x 100

Sharpen:

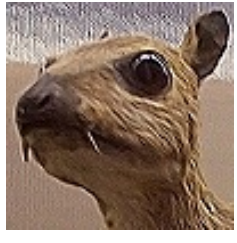


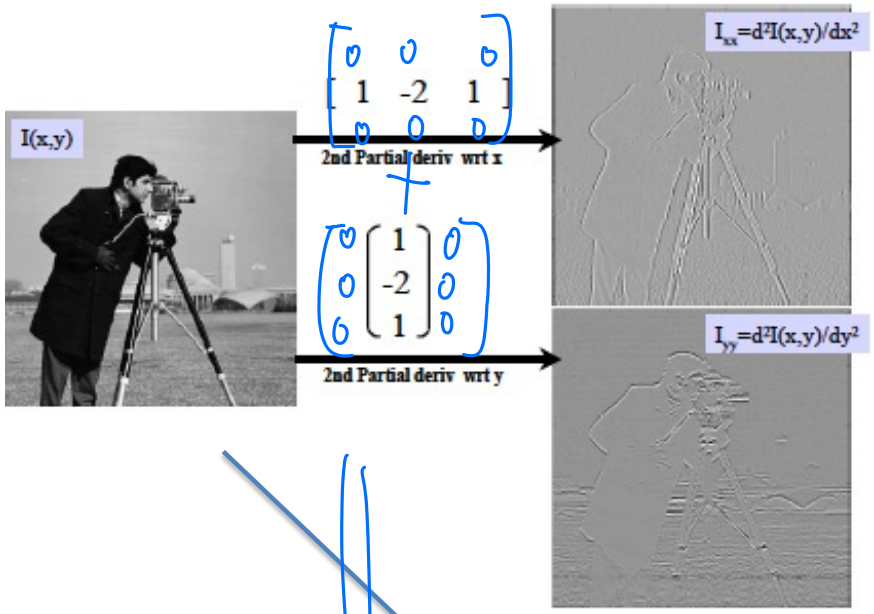
$*$

Laplacian

0	-1	0
-1	5	-1
0	-1	0

$=$





Edge detect:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * I$$

Taylor Series expansion

$$f(x+h) \approx f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + O(h^4)$$

add

$$+ \left[f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{3!}h^3 f'''(x) + O(h^4) \right]$$

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + O(h^4)$$

$$\frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \approx f''(x) + O(h^2)$$

1	-2	1

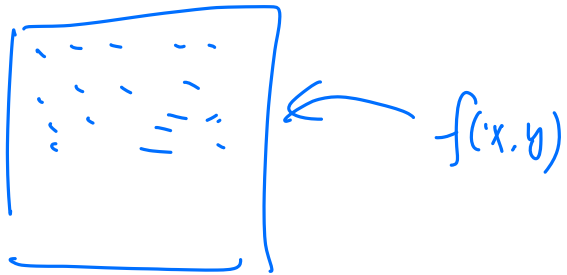
Central difference approx to second derivative

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} + O(h)$$

$$\approx f(x+1) - f(x)$$

$h = 1$

$$f(x-1) - 2f(x) + f(x+1)$$



f(x, y)

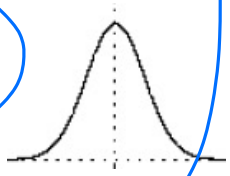
A **continuous convolution product** is defined as

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$

Gaussian filter:

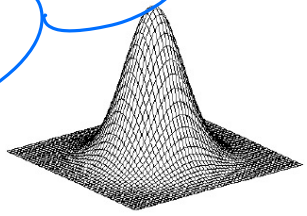
1. Used to reduce image noise and reduce detail.
2. Center pixels weighted more.
3. One dimensional Gaussian smoothing of time series, signals

$$g(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{x^2}{2\sigma^2}}$$

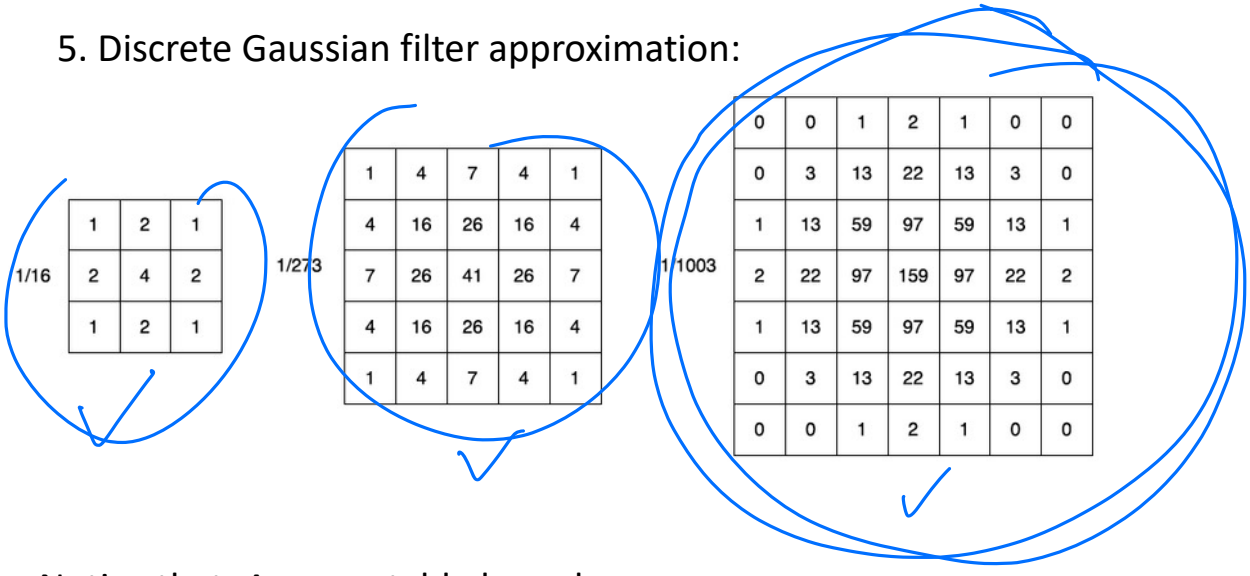


4. Two dimensional Gaussian

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$



5. Discrete Gaussian filter approximation:

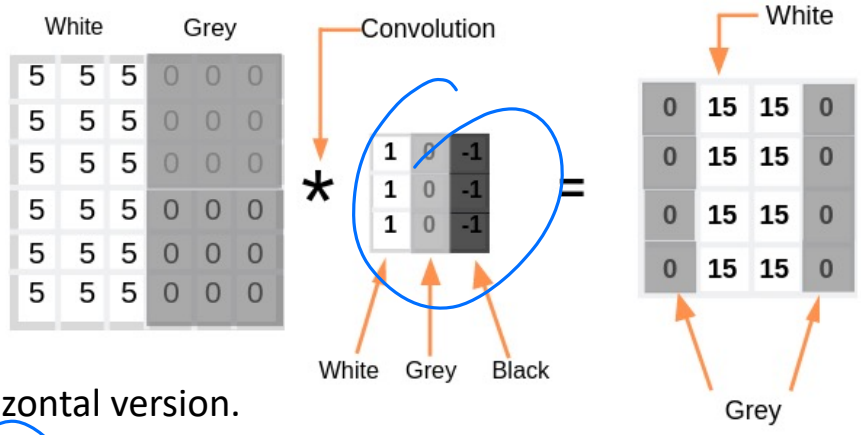


Notice that: A separatable kernel:

$$\frac{1}{256}\cdot\begin{bmatrix}1&4&6&4&1\\4&16&24&16&4\\6&24&36&24&6\\4&16&24&16&4\\1&4&6&4&1\end{bmatrix}=\frac{1}{256}\cdot\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix}\cdot\begin{bmatrix}1&4&6&4&1\end{bmatrix}$$

Vertical Prewitt Edge Detector:

1. Vertical Edge Detection
2. Noise smoothing.



Similarly, one can consider the horizontal version.

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Finite derivative Simple average

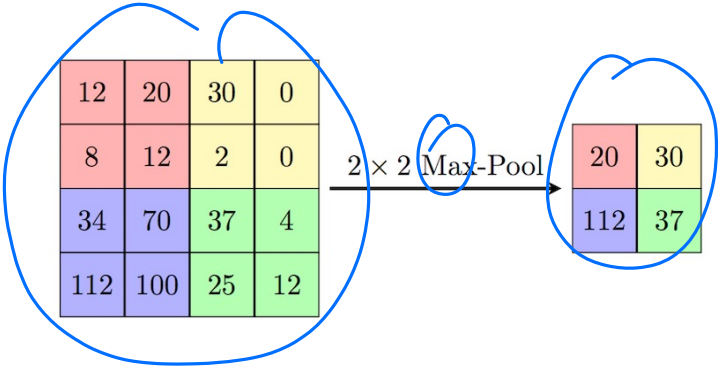
The **Sobel operator:**

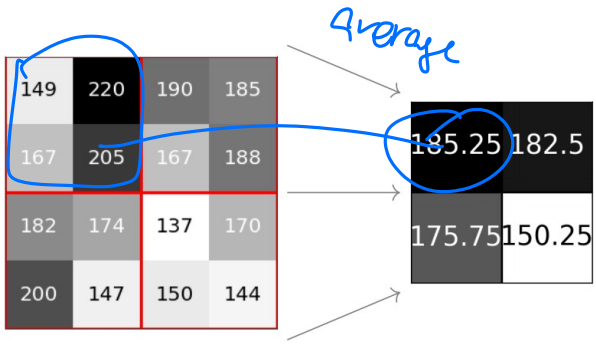
Edge detect:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

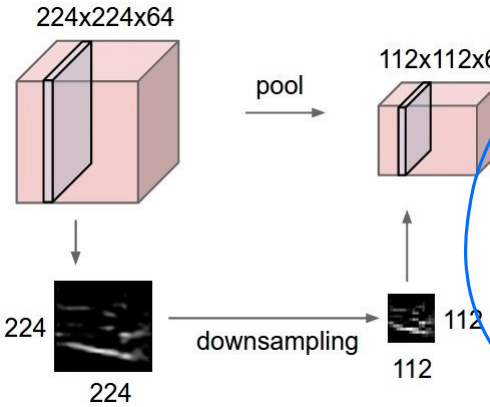
Finite derivative Simple Gaussian

➢ Pooling layers. (Downsampling)

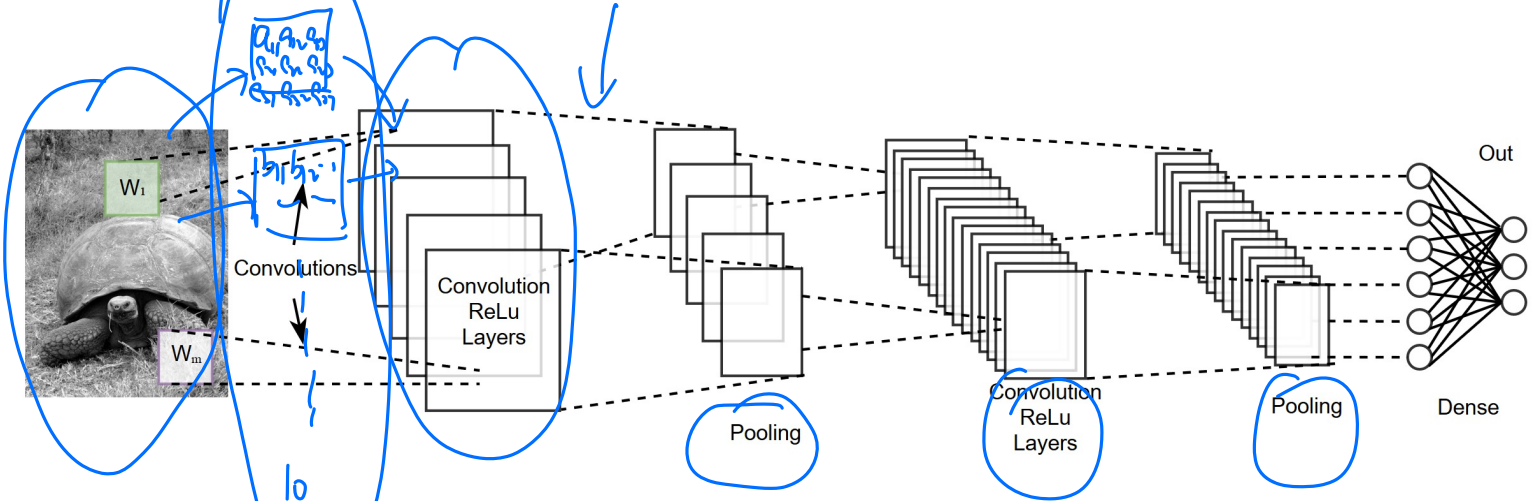




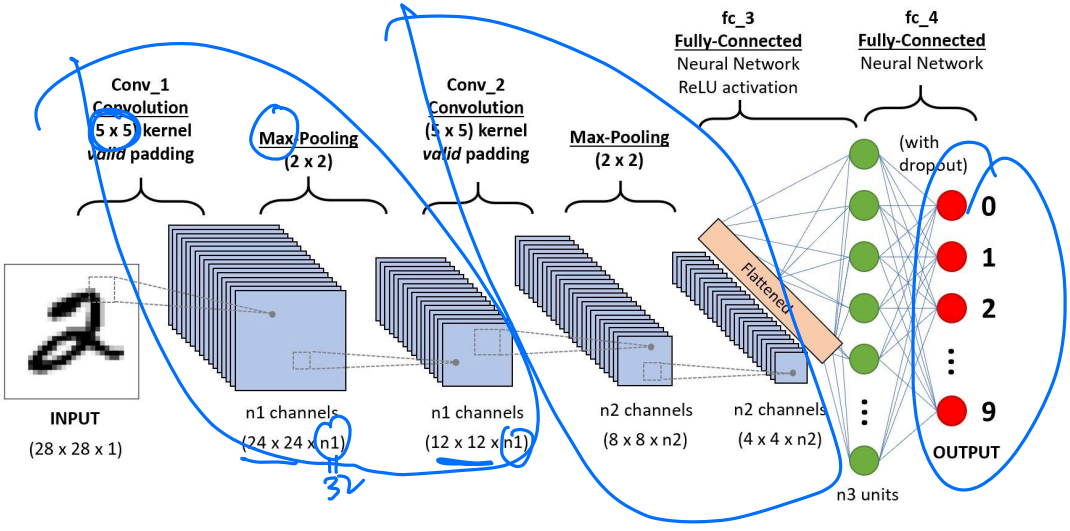
- Max (min) pooling checks if a low level feature is present and reports "yes" or "no" for each region.
- Average pooling checks to what degree a low level feature is matched.



Convolutions In Neural Networks



- Successive **convolution layers** will find higher order features (collections of lower order features)
- To group them efficiently it's common to start including pooling layers.
- A **pooling layer** down samples an image by taking the **max, average, or min** of a $n \times m$ set of pixels.



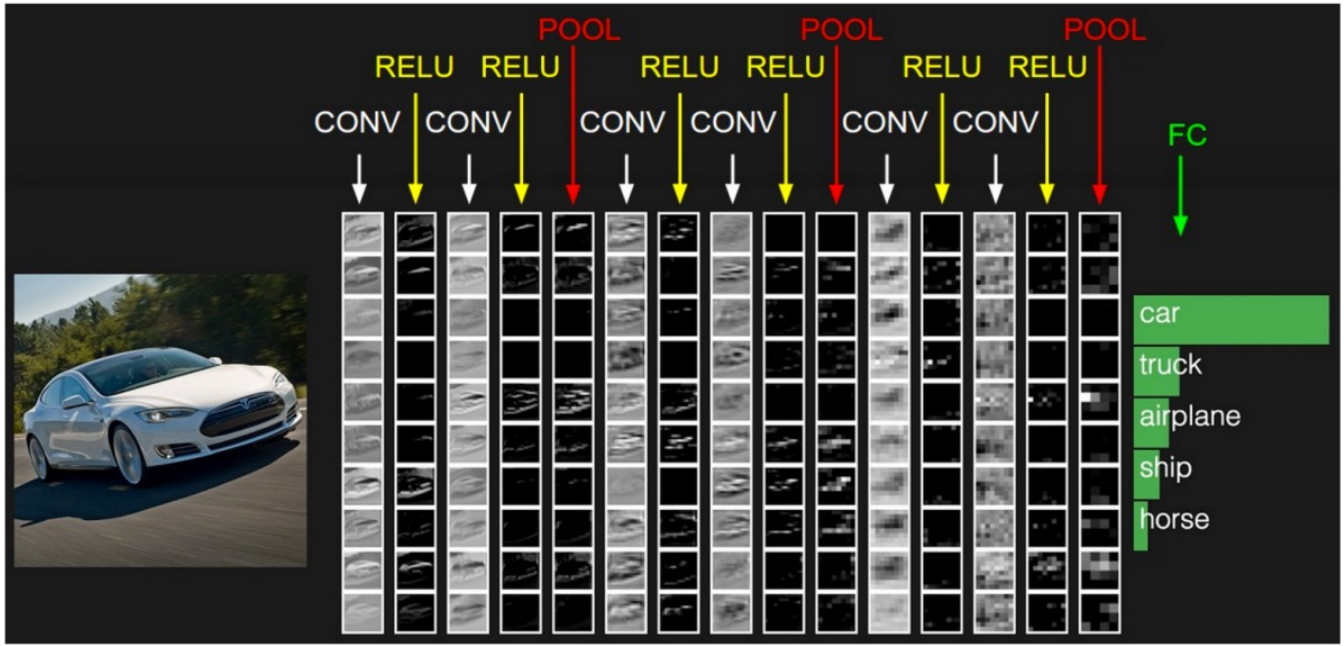

```python
import tensorflow as tf

def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # second convolutional layer
        tf.keras.layers.Conv2D(64, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # fully connected classifier
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs
    ])
    return model
```

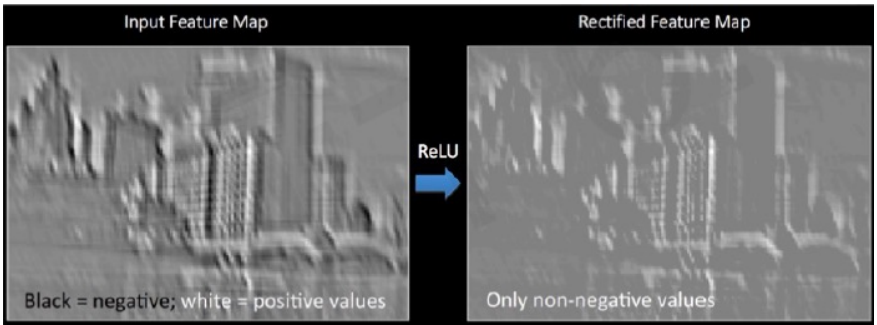
An Interactive Node-Link Visualization of Convolutional Neural Networks
https://www.cs.ryerson.ca/~aharley/vis/conv/

➢ Visualizing Deep Neural Networks

Filters in first layer of CNN are easy to visualize, while deeper ones are harder.

https://www.cs.ryerson.ca/~aharley/vis/conv/ https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html





➢ Convolutional Neural Networks.

➢ History and Examples:

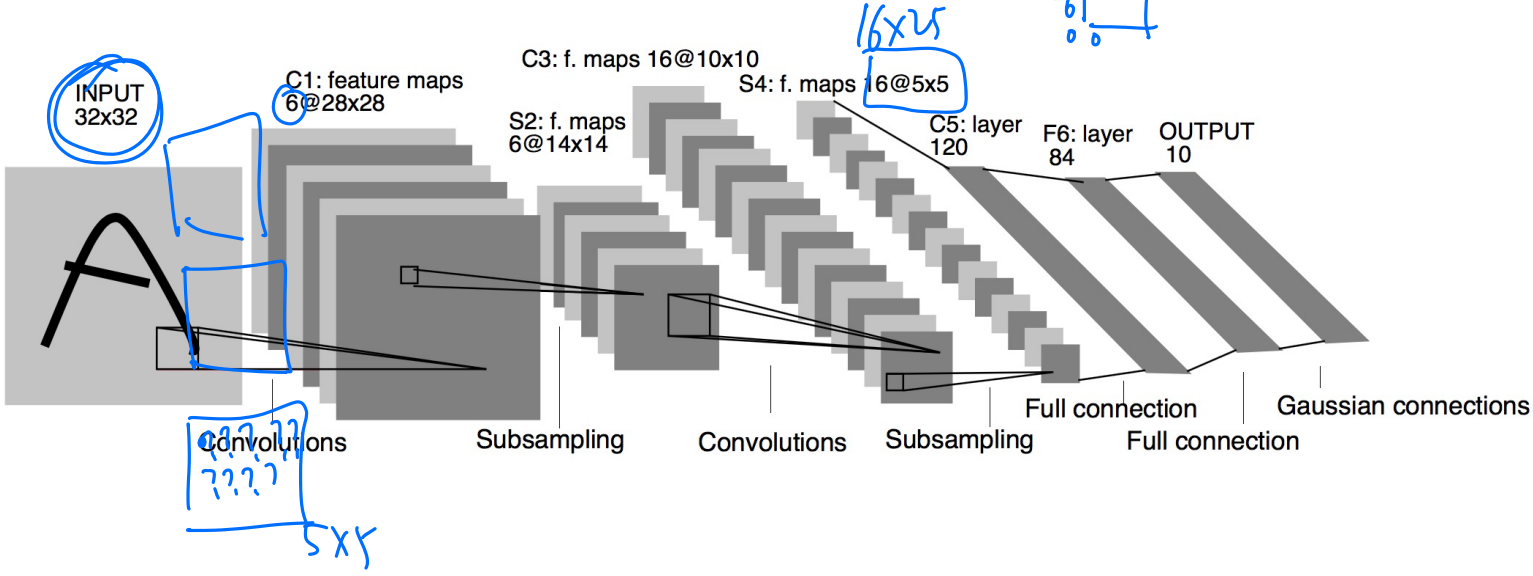
- Can be traced to Neocognitron of Kunihiko Fukushima(1979)
- Yann LeCun combined convolutional neural networks with back propagation (1989)
- Imposes shift invariance and locality on the weights
- Forward pass remains similar
- Backpropagation slightly changes – need to sum over the gradients from all spatial positions

➢ LeNet.

The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990's.

The most widely known CNN, **LeNet5**, was created by Yann LeCun in 1998 and performed a robust classification of the MNIST dataset. It now serves as the benchmark by which all other CNN architectures are compared.

The 28×28 MNIST data is padded with 0s to make each image 32×32. Yann's home page describes LeNet5 with several demonstrations
http://yann.lecun.com/exdb/lenet/index.html



Layer	Type	Size	Kernel	Stride	Activation	Maps
In	Input	32x32				1
C1	Conv	28x28	5x5	1	tanh	6
S2	Ave	14x14	2x2	2	tanh	6
C3	Conv	10x10	5x5	1	tanh	16
S4	Ave	5x5	2x2	2	tanh	16
C5	Conv	1x1	5x5	1	tanh	120
F6	Dense	84			tanh	
Out	Dense	10				

➢ Computer vision

In the mid 2000's, **ImageNet** was started as a project of Fei-Fei Li as Stanford and comprises 14 million hand annotated images for algorithms to train and test again. Since 2010, the ImageNet project has run a benchmark test for computer vision.

ImageNet ILSVRC challenge Winners:

AlexNet. 2012 The first work that popularized Convolutional Networks in Computer Vision. (60 m parameters) with 17% top 5 error rate while the second place winner had 26%.

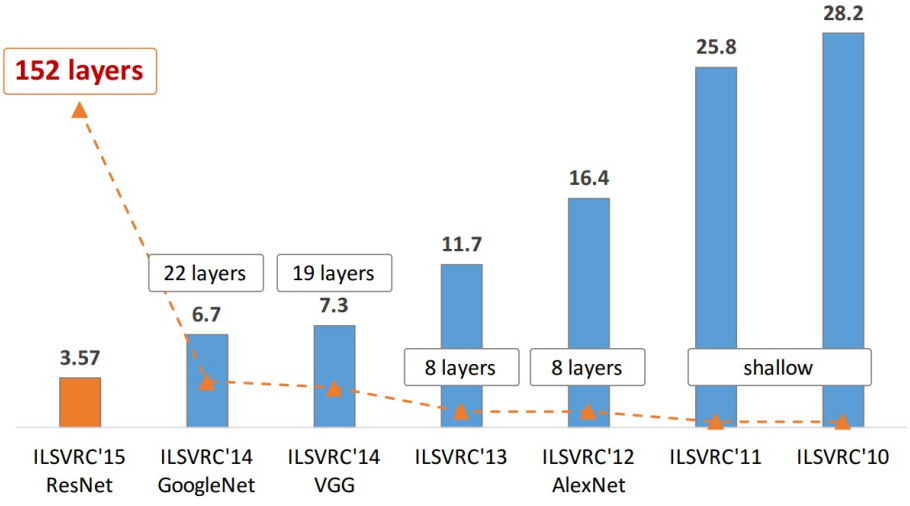
ZF Net. 2013

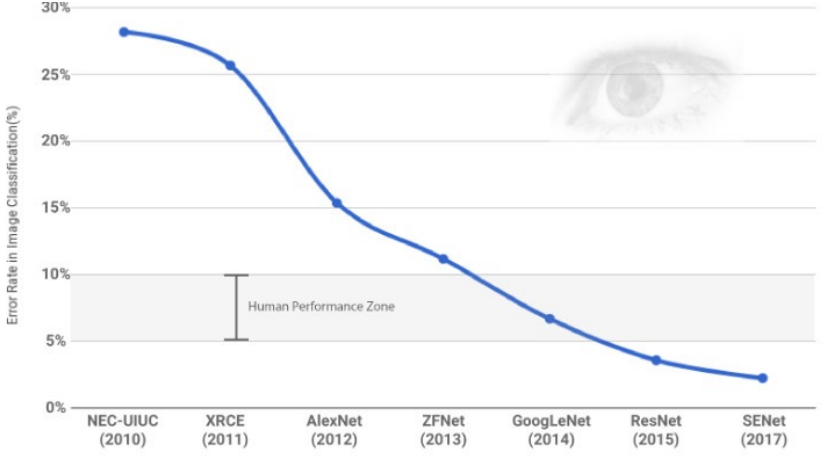
GoogLeNet.2014. (4m parameters)

ResNet 2015 (140m parameters)

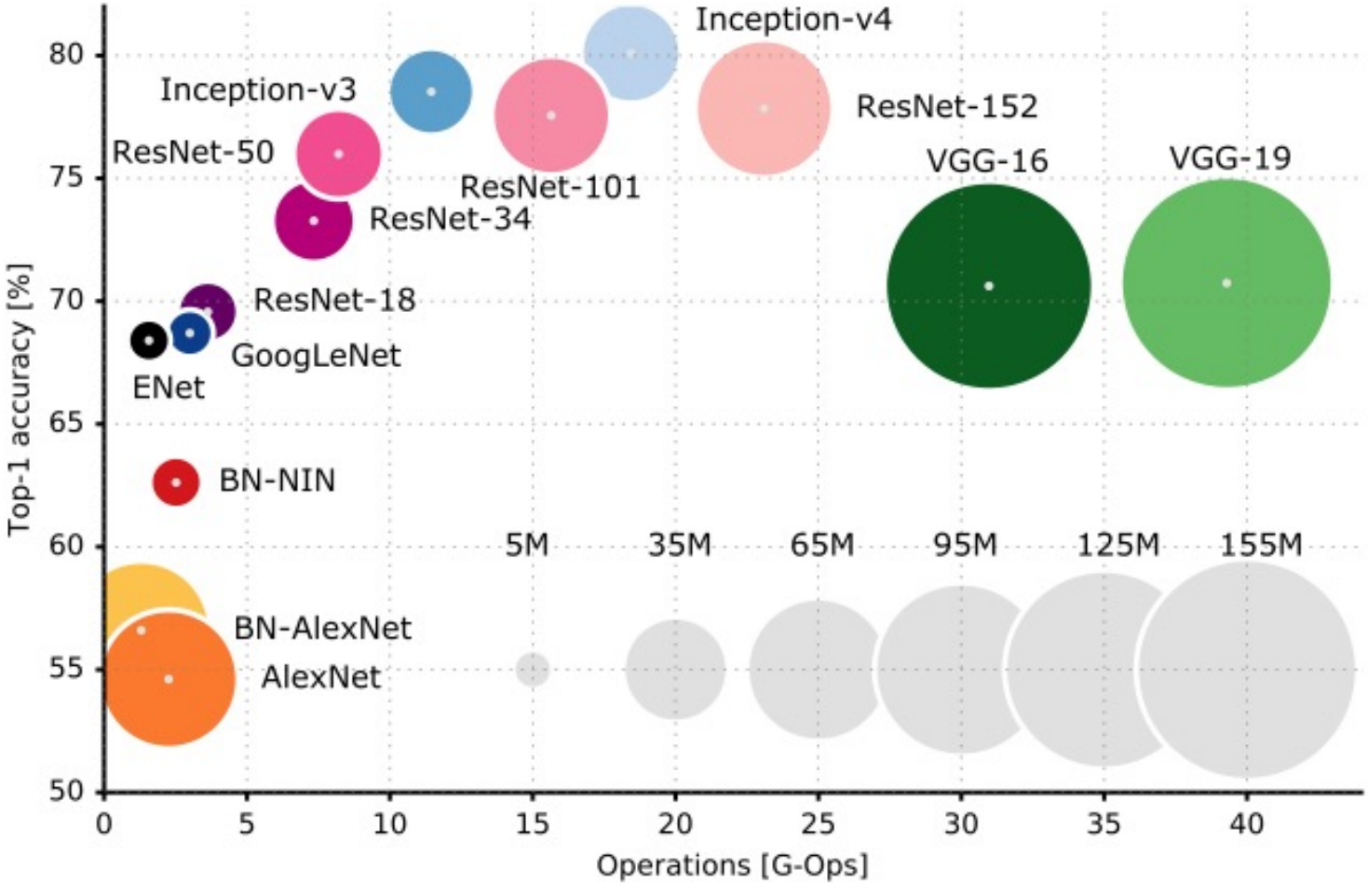
The competition has passed to Kaggle since 2017:

https://www.kaggle.com/c/imagenet-object-localization-challenge/overview

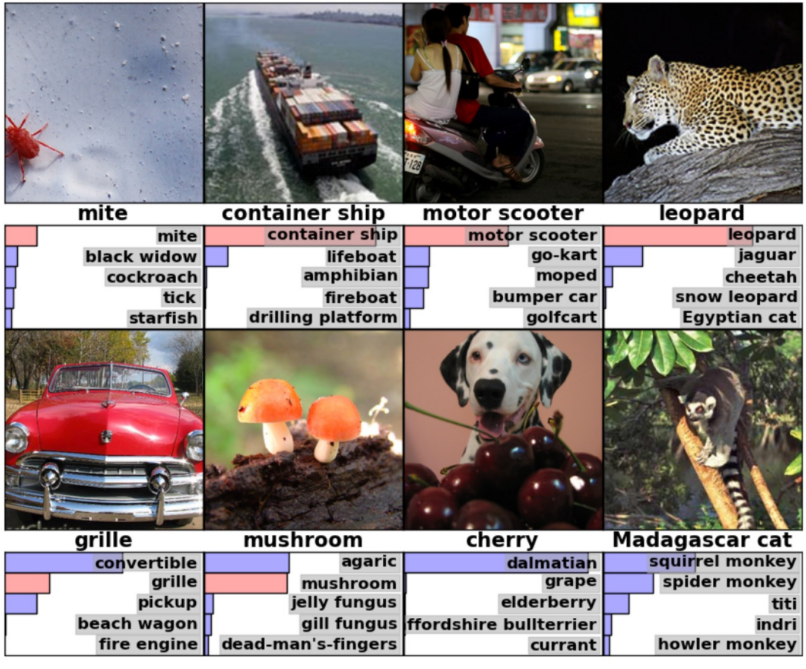




https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html


Inception-v4
Inception-v3
ResNet-152
ResNet-50
VGG-16
VGG-19
ResNet-101
ResNet-34
ResNet-18
GoogLeNet
ENet
BN-NIN
5M
35M
65M
95M
125M
155M
BN-AlexNet
AlexNet
Top-1 accuracy [%]
Operations [G-Ops]
80
75
70
65
60
55
50
0
5
10
15
20
25
30
35
40

➢ **AlexNet(2012)** [Krizhevsky et al., 2012]



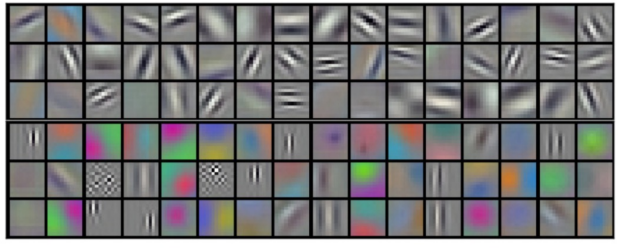
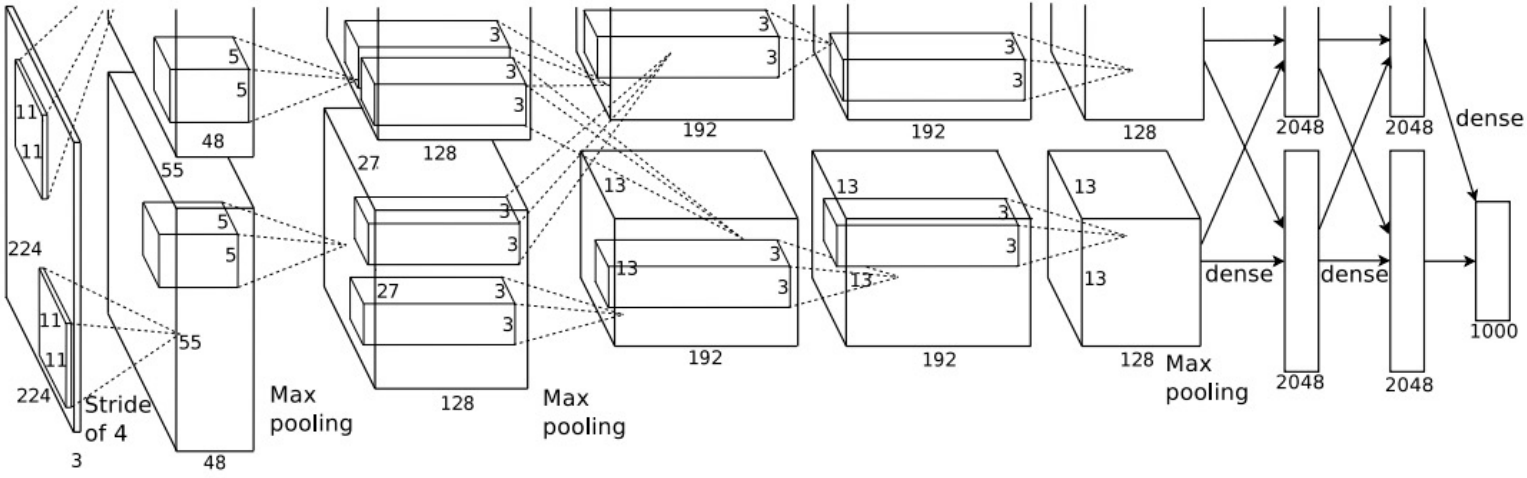


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2

AlexNet was much deeper than LeNet5, with two fully connected layers of 4,096 nodes. It also used ReLu as opposed to tanh for activation functions. Finally, to increase training AlexNet included **dropout layers**, which turn off neurons at random forcing the graph to learn the same concept in a redundant way.

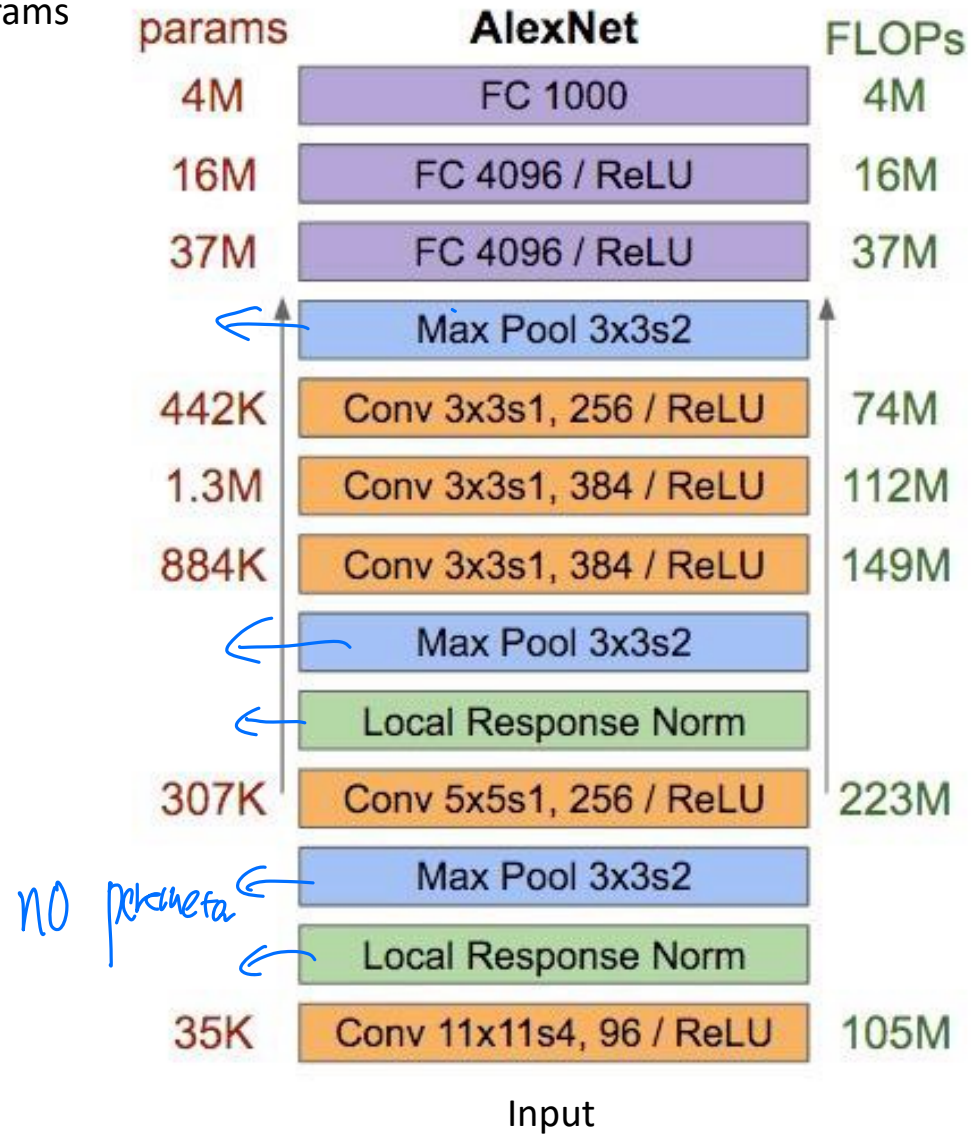
AlexNet (2012) Architecture

- 8 layers: first 5 convolutional, rest fully connected
- ReLU nonlinearity
- Local response normalization
- Max-pooling
- Dropout



AlexNet Architecture diagrams

Layers
-Kernel sizes
-Strides
-# channels
-# kernels
-Max pooling



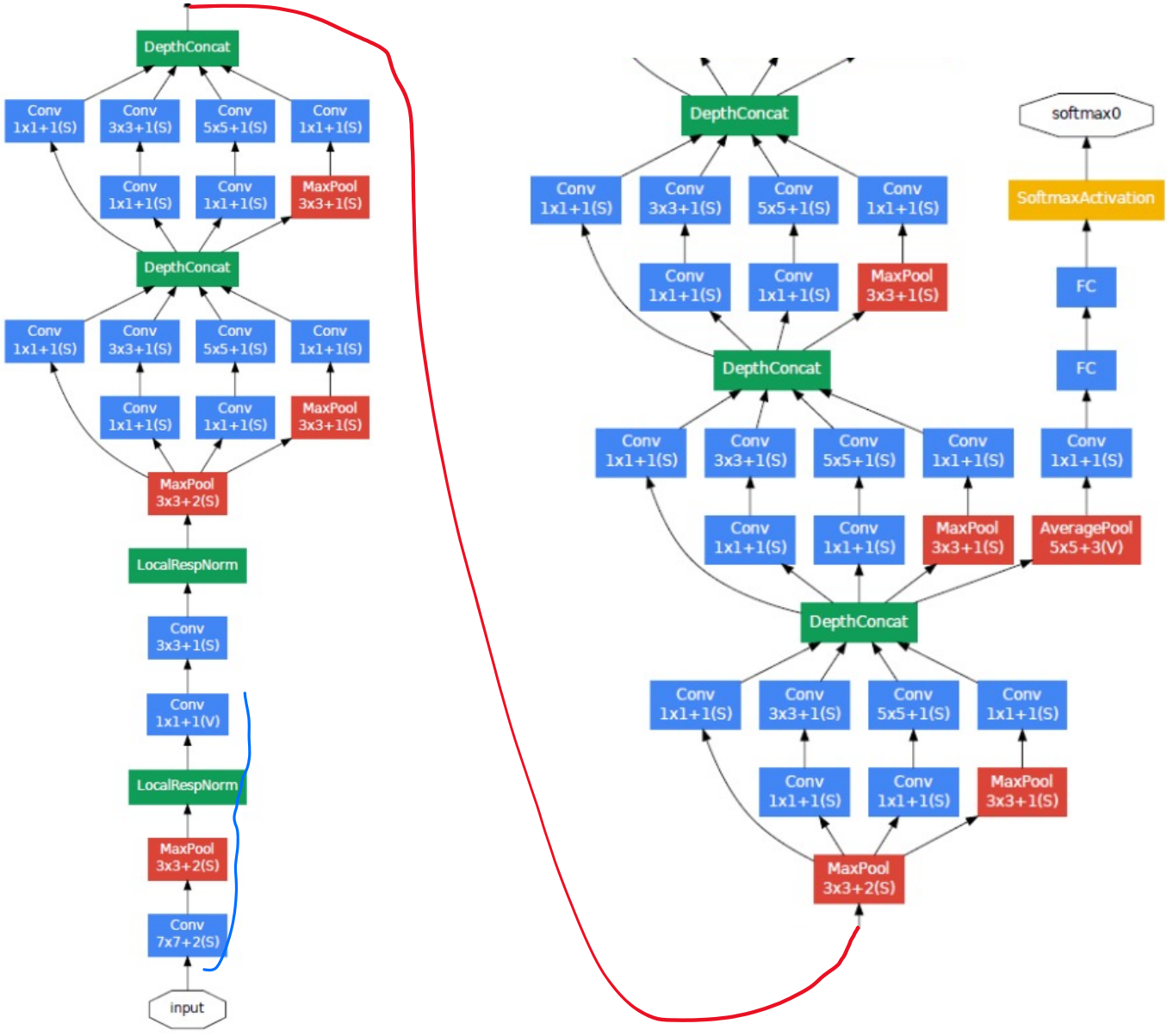
➢ ZF Net (2013)

- Very similar architecture to AlexNet, except for a few minor modifications.
- AlexNet trained on 15 million images, while ZF Net trained on only 1.3 million images.
- Instead of using 11x11 sized filters in the first layer (which is what AlexNet implemented), ZF Net used filters of size 7x7 and a decreased stride value. The reasoning behind this modification is that a smaller filter size in the first conv layer helps retain a lot of original pixel information in the input volume. A filtering of size 11x11 proved to be skipping a lot of relevant information, especially as this is the first conv layer.
- As the network grows, we also see a rise in the number of filters used.
- Used **ReLUs** for their activation functions, **cross-entropy loss** for the error function, and trained using **batch stochastic gradient descent**.
- Trained on a GTX 580 GPU for **twelve days**.
- Developed a visualization technique named **Deconvolutional Network**, which helps to examine different feature activations and their relation to the input space. Called “deconvnet” because it maps features to pixels (the opposite of what a convolutional layer does).

➢ **GoogLeNet**. 2014.

GoogLeNet is a type of convolutional neural network based on the **Inception architecture**. It utilizes Inception modules, which allow the network to choose between multiple convolutional filter sizes in each block. An Inception network stacks these modules on top of each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								


input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
SoftmaxActivation
FC
AveragePool 5x5+3(V)


Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
softmax1
SoftmaxActivation
FC
FC
Conv 1x1+1(S)
softmax2
SoftmaxActivation
FC
AveragePool 7x7+1(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat

Main Points of GoogLeNet

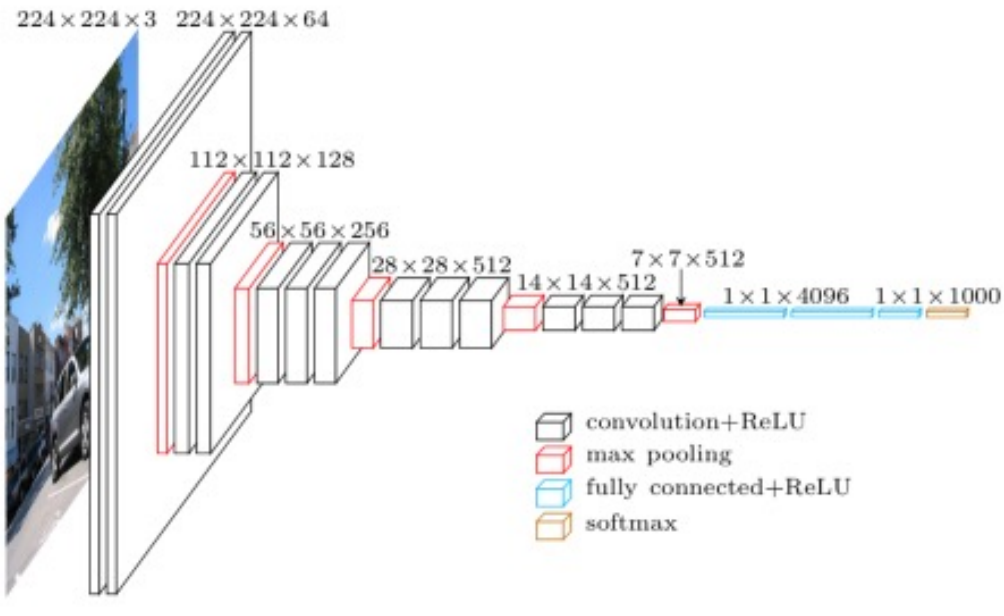
- Used 9 Inception modules in the whole architecture, with over 100 layers in total!
- No use of fully connected layers! They use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.
- Uses **12x fewer** parameters than AlexNet.
- During testing, multiple crops of the same image were created, fed into the network, and the softmax probabilities were averaged to give us the final solution.
- Dropout Regularization with dropout ratio = 0.7
- ReLU activation
- Utilized concepts from R-CNN for their detection model.
- There are updated versions to the Inception module (Versions 6 and 7).
- A softmax classifier with 1000 classes output similar to the main softmax classsifier.
- Trained on "a few high-end GPUs **within a week**".

➢ **VGG** (2014 runner up) [Simonyan-Zisserman'14]

- Deeper than AlexNet: 11-19 layers versus 8
- No local response normalization
- Number of filters multiplied by two every few layers
- Spatial extent of filters 3×3 in all layers
- Instead of 7×7 filters, use three layers of 3×3 filters
  - Gain intermediate nonlinearity
  - Impose a regularization on the 7×7 filters

· Relu ← normalize

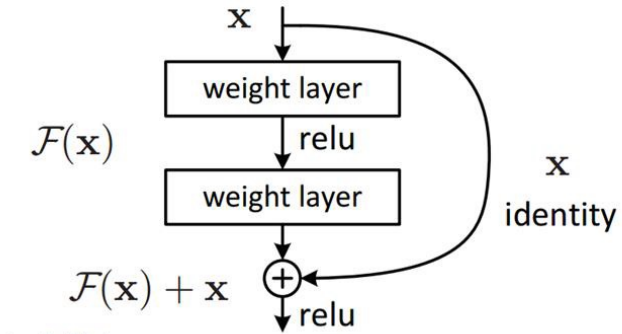
· Dropout



➢ **ResNet** (2015) [HGRS-15]

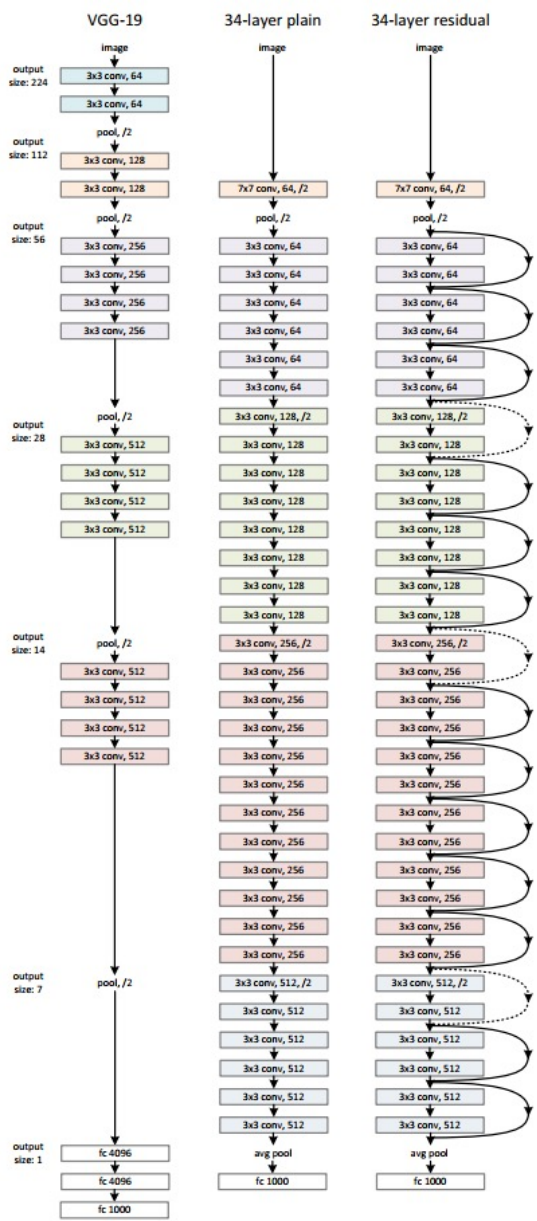
- Solves problem by adding
- skip connections
- Very deep: 152 layers
- No dropout
- Stride
- Batch normalization

Residual Block:

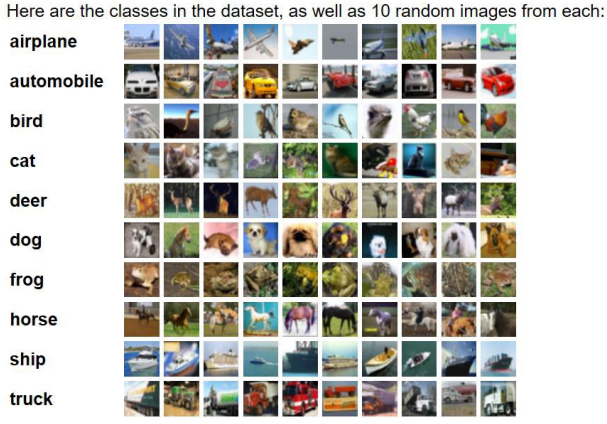


A residual block

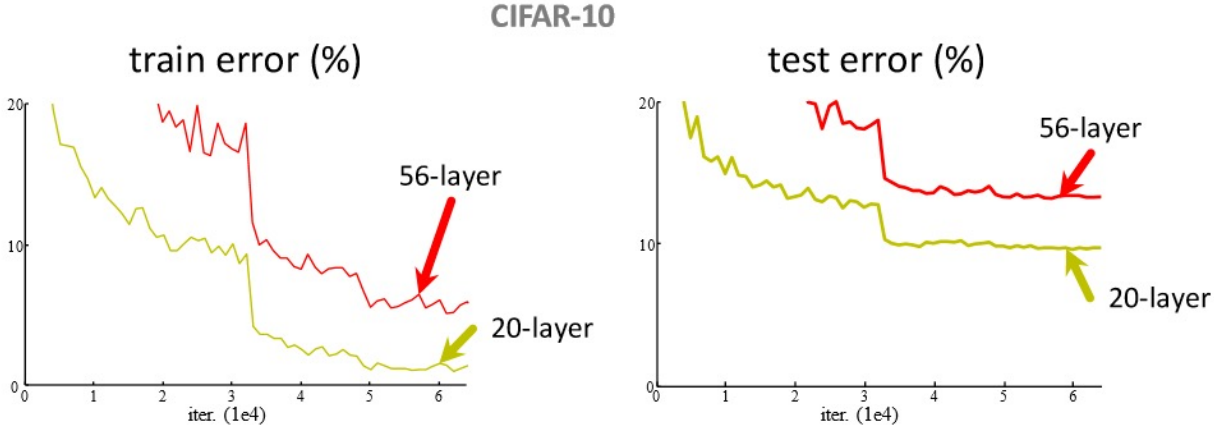
allowing the network to train a rough structure before the ne structure was trained.



CIFAR-10 data set:
60,000 32x32 color images, 10 classes



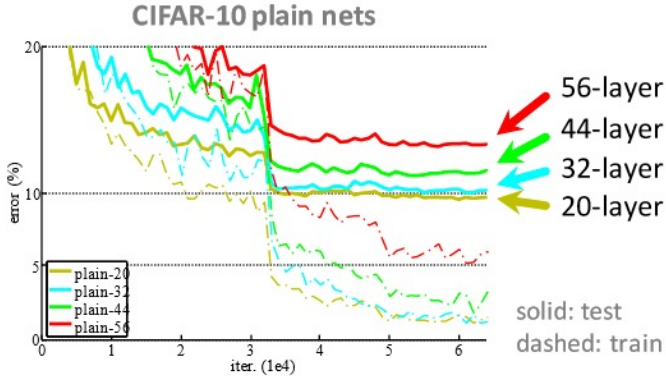
Simply stacking layers?

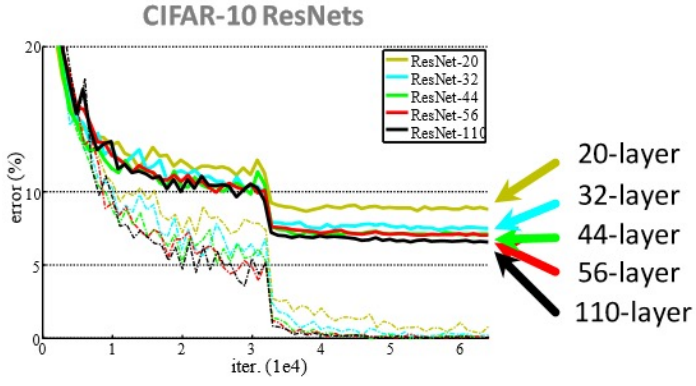


- *Plain* nets: stacking 3x3 conv layers…
- 56-layer net has **higher training error** and test error than 20-layer net

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

CIFAR-10 experiments





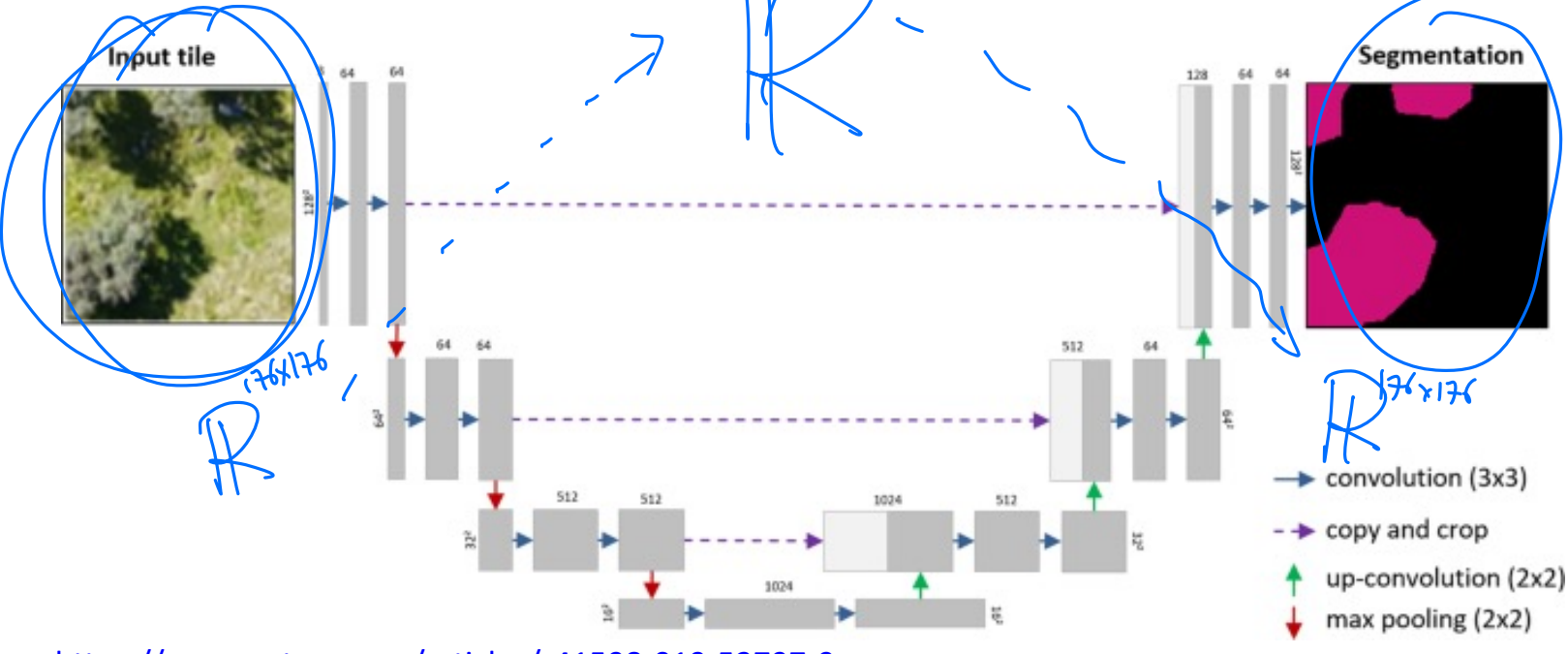
- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

More example:

CNN-based segmentation approach (U-net)

https://arxiv.org/abs/1505.04597



https://www.nature.com/articles/s41598-019-53797-9

Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks

https://arxiv.org/pdf/1705.03820.pdf

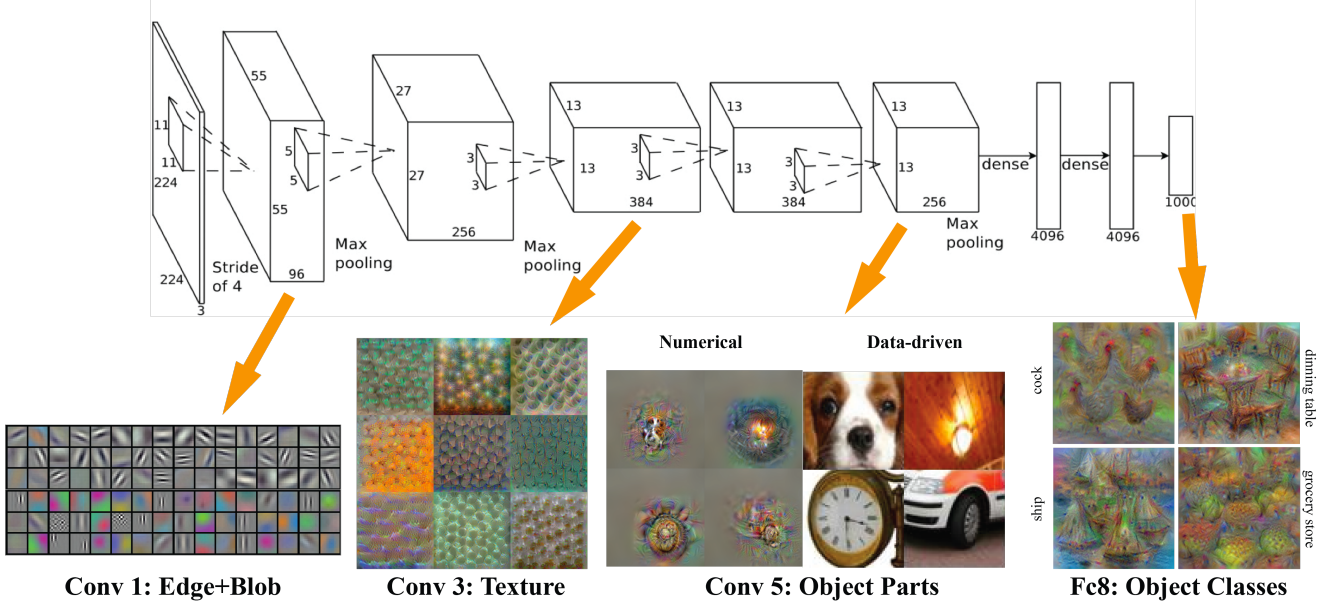
Transfer Learning?

- Filters learned in first layers of a network are transferable from one task to another.
- When solving another problem, no need to retrain the lower layers, just fine tune upper ones.
- Is this simply due to the large amount of images in ImageNet?
- Does solving many classification problems simultaneously result in features that are more easily transferable?

For example, we could use the **pretrained convolution weights** of AlexNet, feeding them into a **new set of dense layers** tuned to a new classification program.

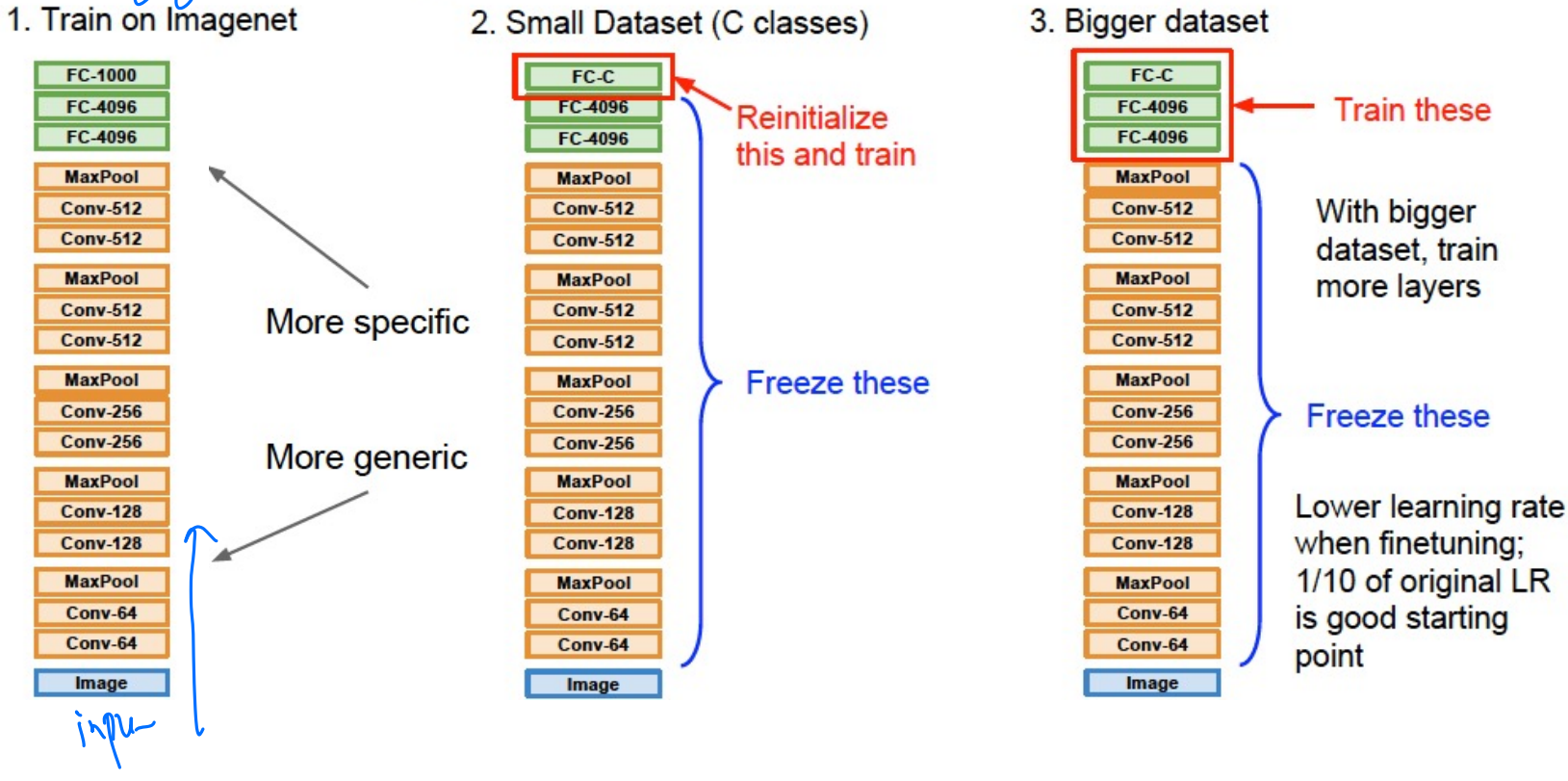
➢ Using Pretrained CNN's:

- The high degree of redundancy in the convolutional layers of a CNN makes them excellent candidates for transfer learning.
- In transfer learning, layers from a network trained on a certain task are reused for a different task.
- It turns out that many low level visual features are roughly the same.



Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014



- Takeaway for your projects and beyond

Have some dataset of interest but it has smaller set of images.

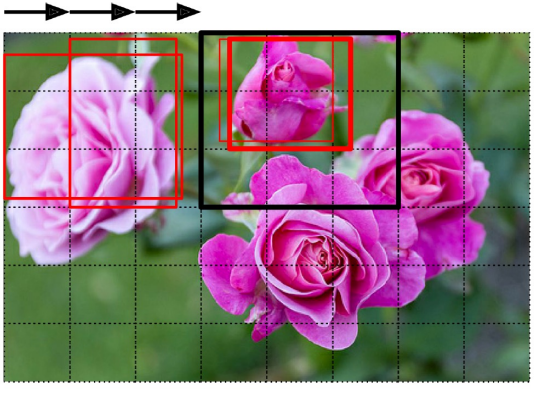
1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models, so you don’t need to train your own.

Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision

Further topics on CNN:

1. Localization with CNN's
2. Object Detection with CNN's
3. Semantic Segmentation



Output:

taxi: (x1, y1, w1, h1)

person: (x2, y2, w2, h2)

person: (x3, y3, w3, h3)

....

➢ Disadvantages of CNN:

Deep Learning may be fragile against noise!

Raining weather☺

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

[Goodfellow et al., 2014]

- Small but malicious perturbations can result in severe misclassification
- Malicious examples generalize across different architectures
- What is source of instability?
- Can we robustify network?

Memory Requirements: A problem with CNNs is that the convolutional layers require a huge amount of RAM. This is especially true during training, because the reverse pass of backpropagation requires all the intermediate values computed during the forward pass.

If training crashes because of an **out-of-memory error**, you can try **reducing the mini-batch size**. Alternatively, you can try **reducing dimensionality** using a **stride**, or **removing a few layers**. Or you can try using 16-bit floats instead of 32-bit floats. Or you could distribute the CNN across multiple devices.

Yann LeCun CVPR'15, invited talk: What's wrong with deep learning?
One important piece: **missing some theory**!

http://techtalks.tv/talks/whats-wrong-with-deep-learning/61639/