

CSE/ISyE 6740
Computational Data Analysis

Convolutional Neural Networks

10/19/2026

Kai Wang, Assistant Professor in Computational Science and Engineering
kwang692@gatech.edu

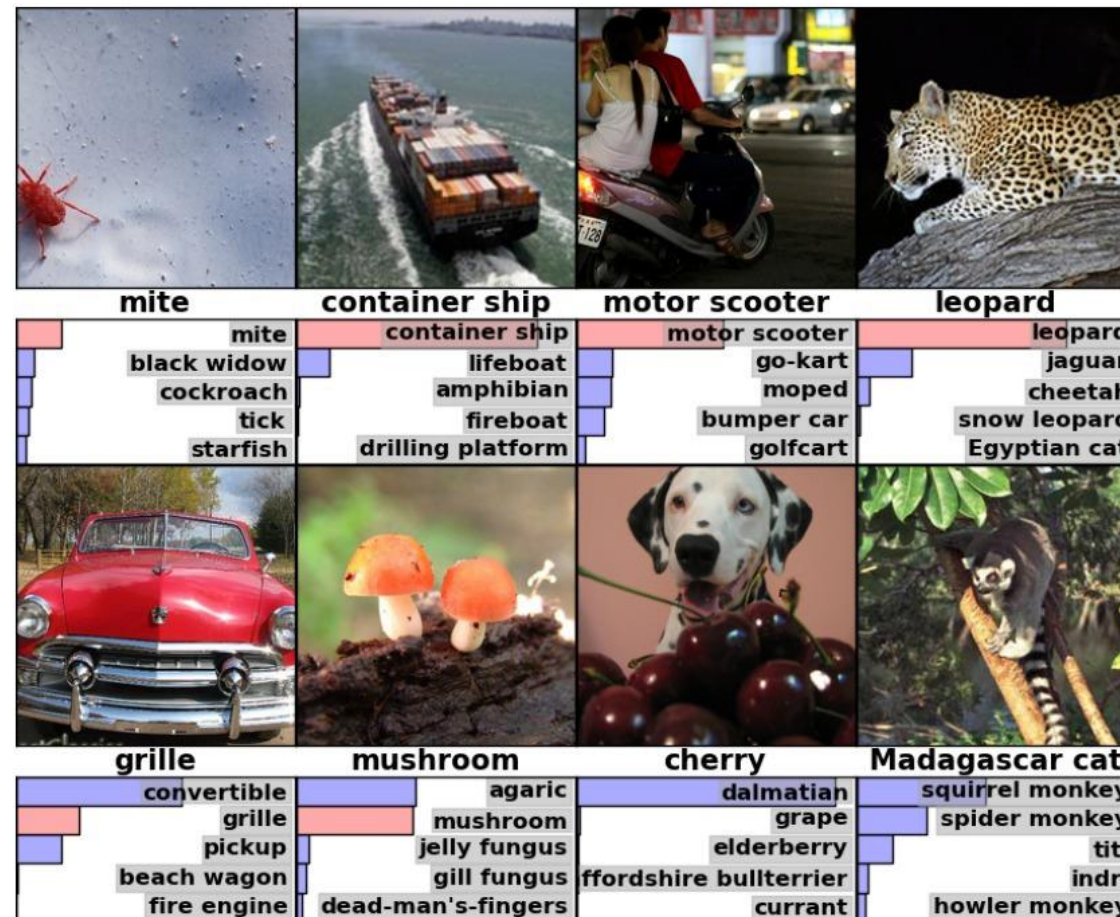
Outline

- **Supervised Learning**
 - Neural networks
 - Convolutional layer
 - Convolutional neural networks (CNNs)

Convolutional Neural Networks

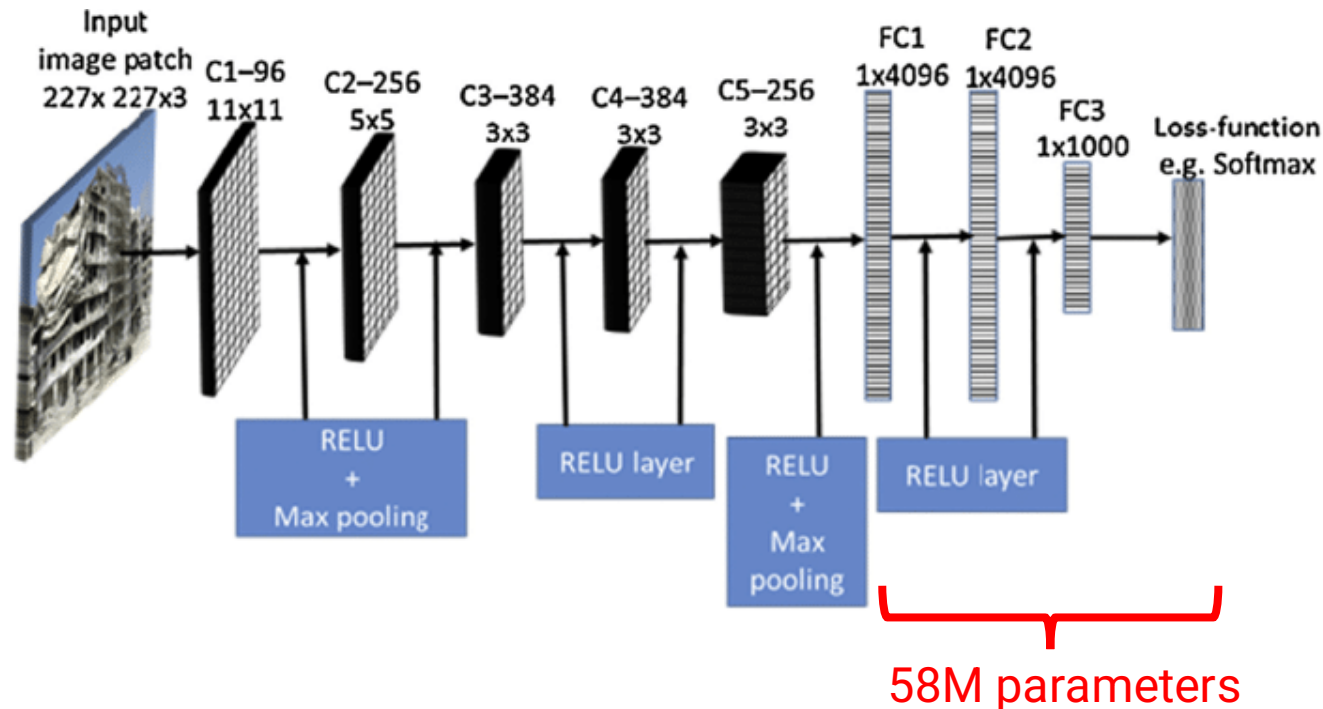
Image Classification: ImageNet

- Image classification with 1.3M color images and 1000 classes
- Need large scale non-parametric methods



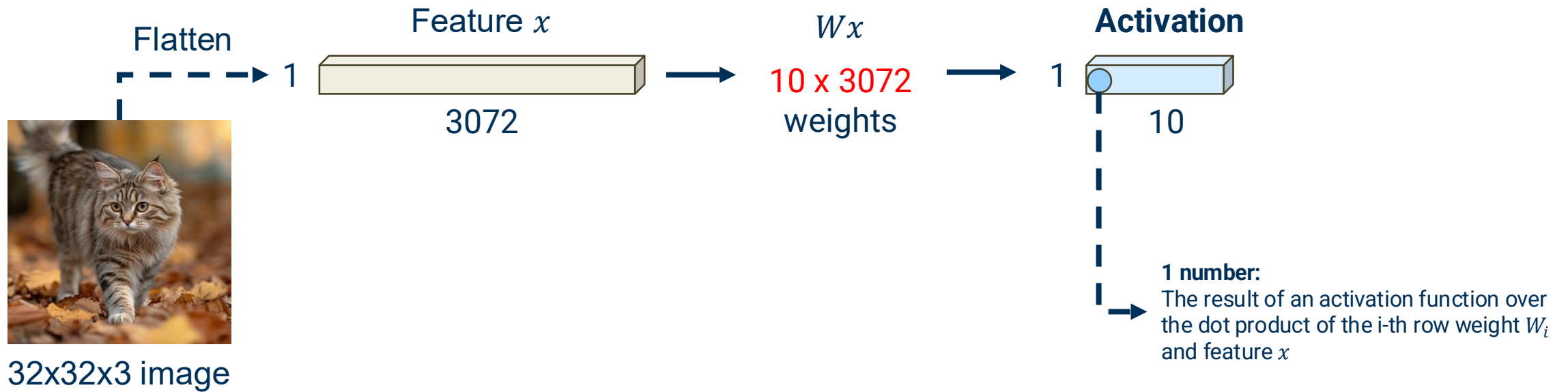
Neural Networks and Convolution Features

- 8 layer convolution neural network [[Krizhevsky et al. 2012](#)] achieved the state-of-the-art result (beating the second place by 10%).
 - **First 5 layers:** convolution + max pooling
 - **Next 2 layers:** fully connected nonlinear neurons
 - **Last layer:** multiclass logistic regression

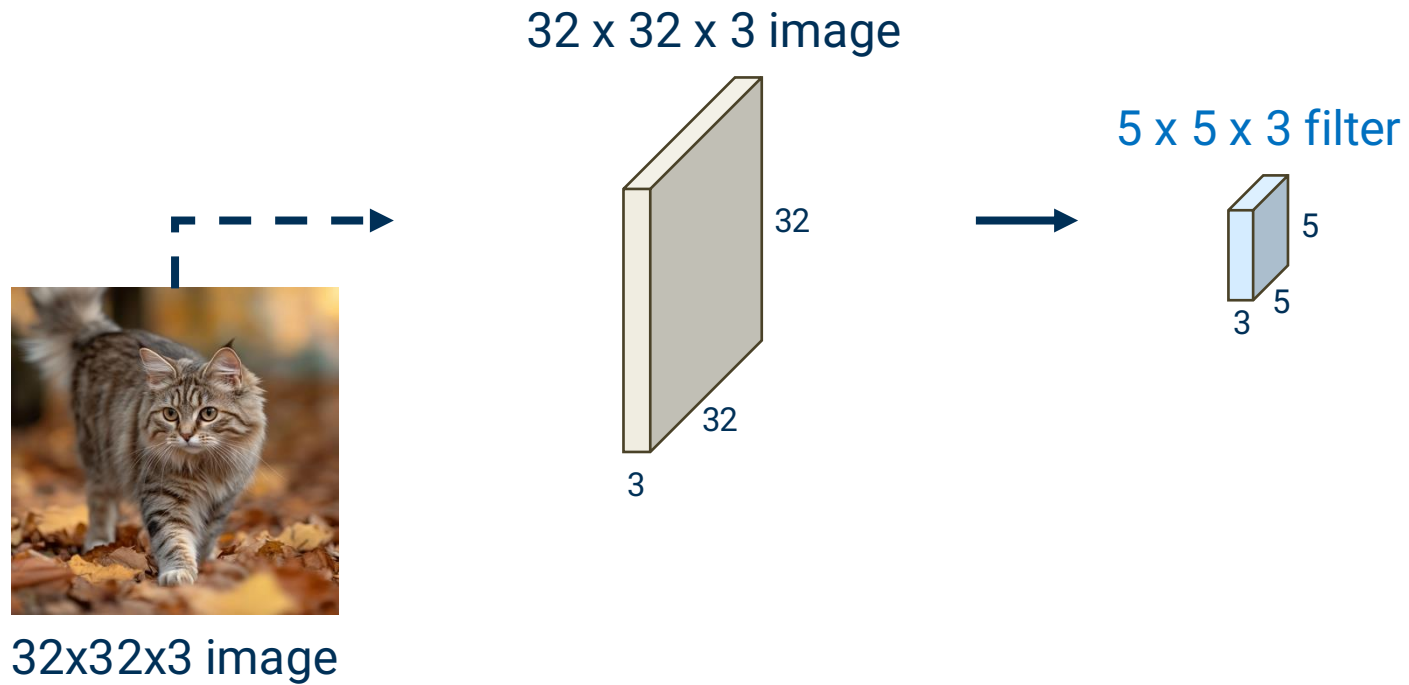


Fully Connected (FC) Layer

How to handle images using fully connected layers?



Convolution Layer

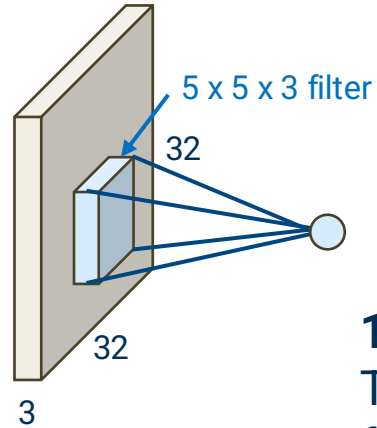


Convolution Layer



32x32x3 image

32 x 32 x 3 image



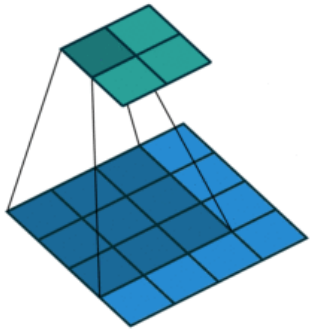
1 number:

The result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e., $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)

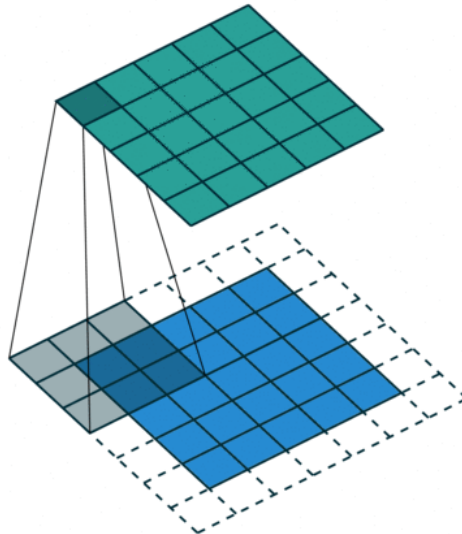
$$w^T x + b$$

Padding and Stride

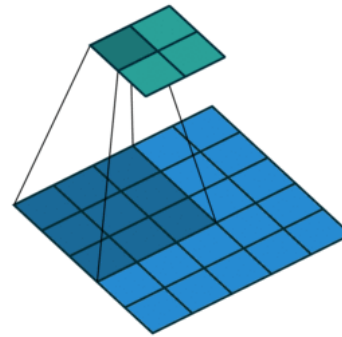
Padding = 0, stride = 1



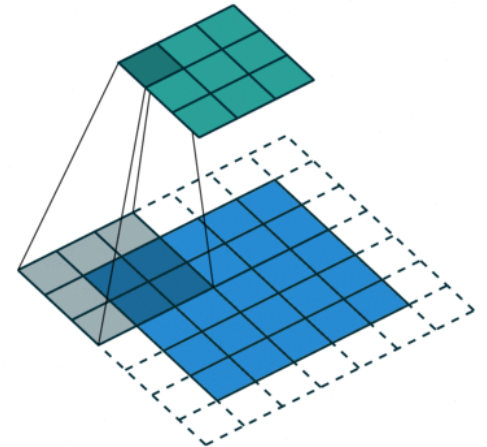
Padding = 1, stride = 1



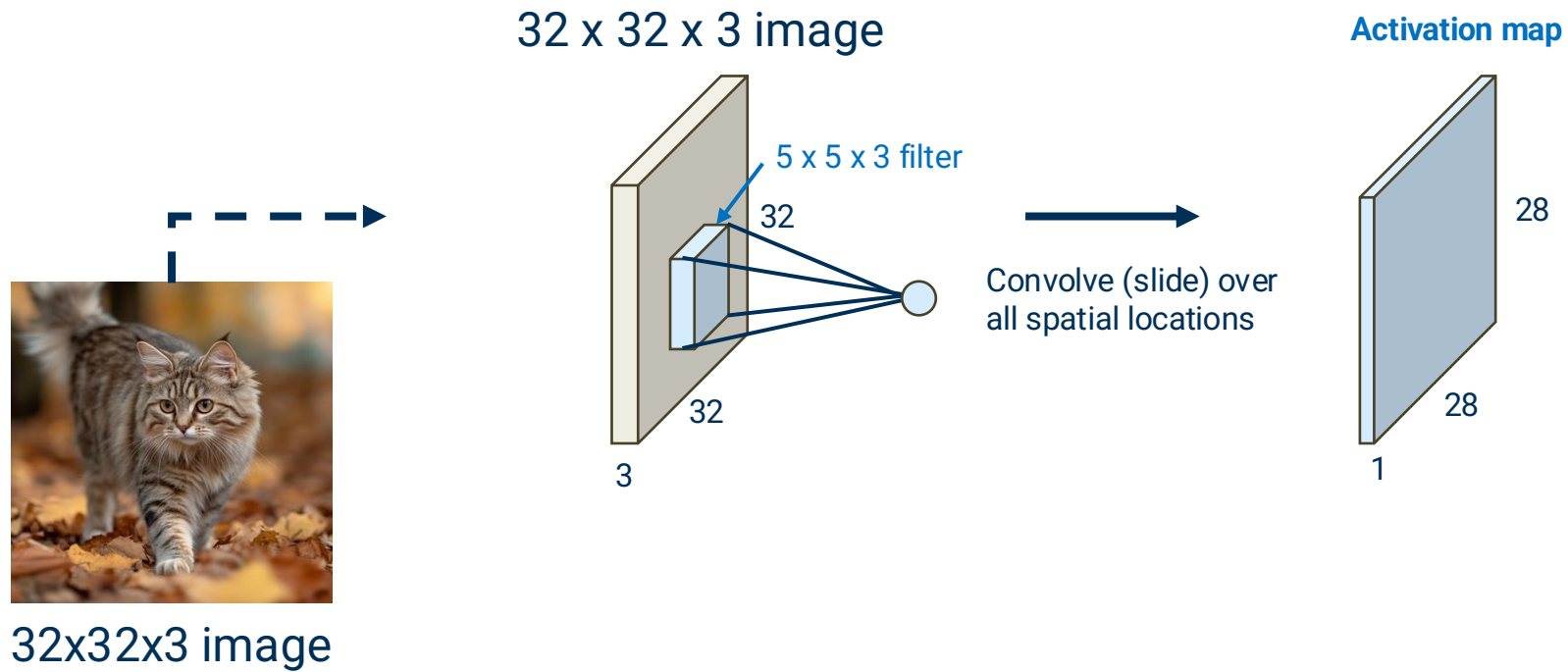
Padding = 0, stride = 2



Padding = 1, stride = 2



Convolution Layer

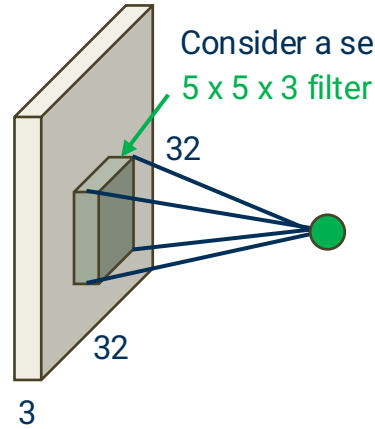


Convolution Layer



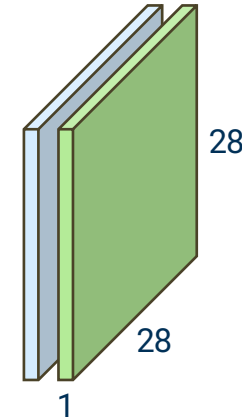
32x32x3 image

32 x 32 x 3 image



Convolve (slide) over
all spatial locations

Activation maps



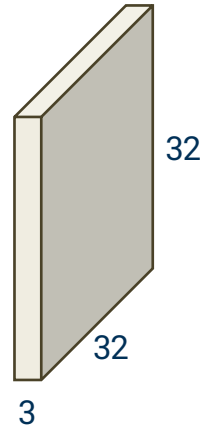
Convolution Layer



32x32x3 image

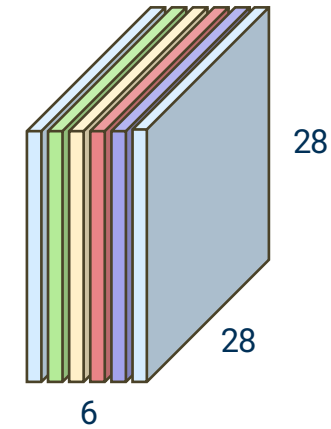


32 x 32 x 3 image



Convolve (slide) over
all spatial locations

Activation maps

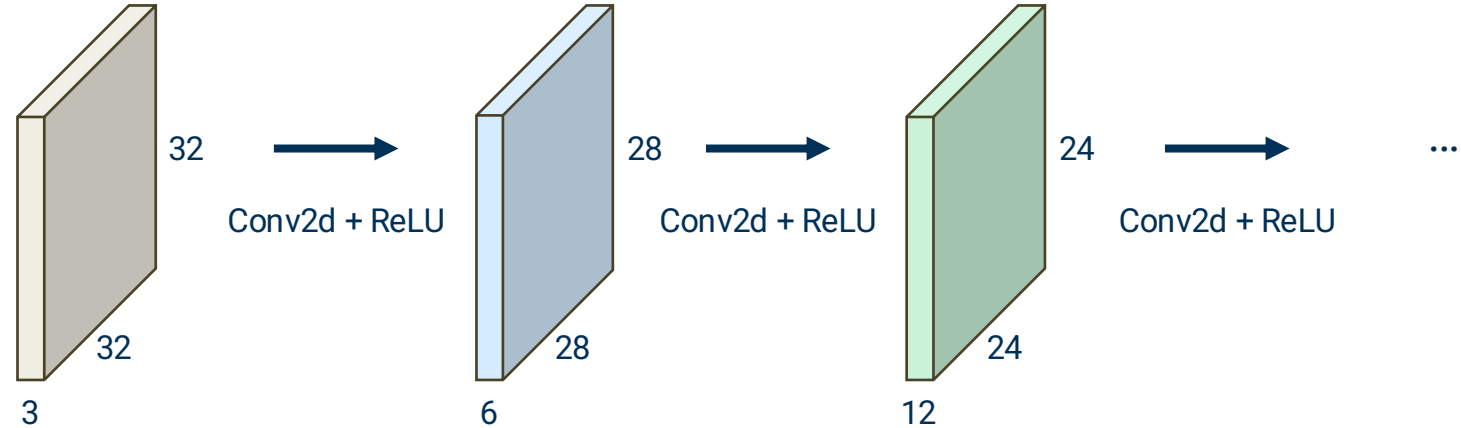


We stack all these outputs to get a 28x28x6 images.
Pick the number of layers you want.

Convolution Layer



32x32x3 image



This is how the [torch.nn.Conv2d](#) function works.

Conv2d

```
class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None) \[source\]
```

Applies a 2D convolution over an input signal composed of several input planes.

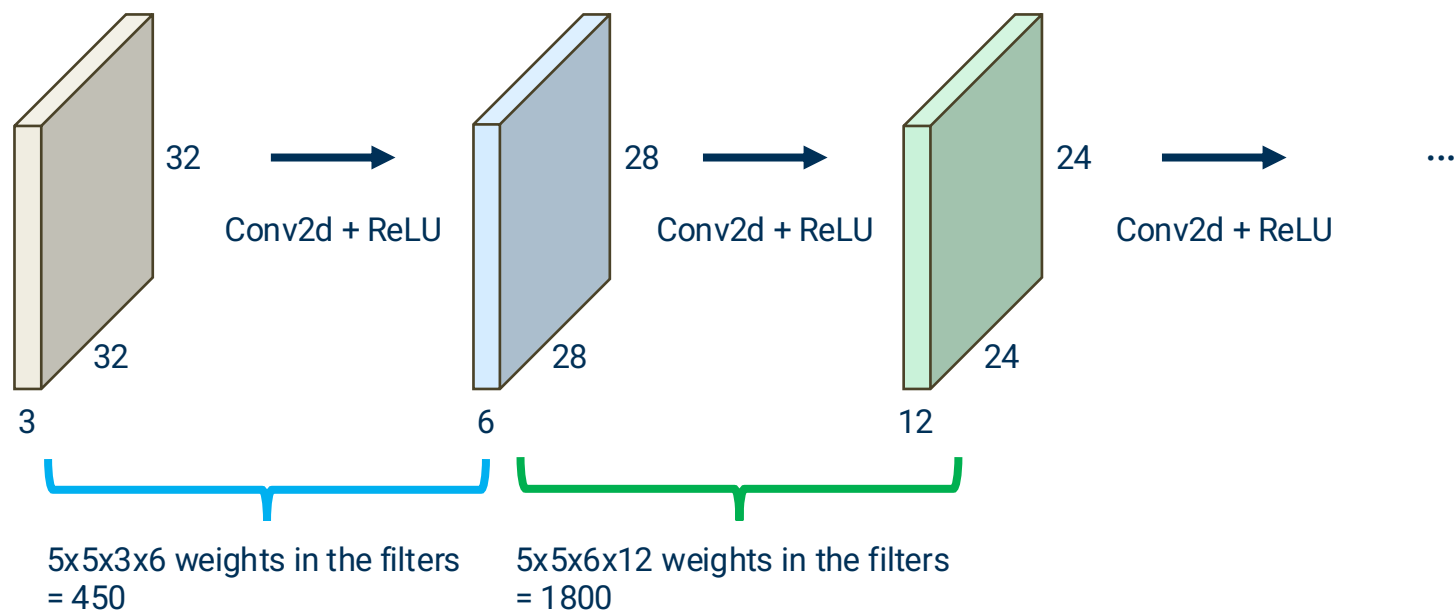
In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{out}, H_{out}, W_{out})$ can be precisely described as:

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k)$$

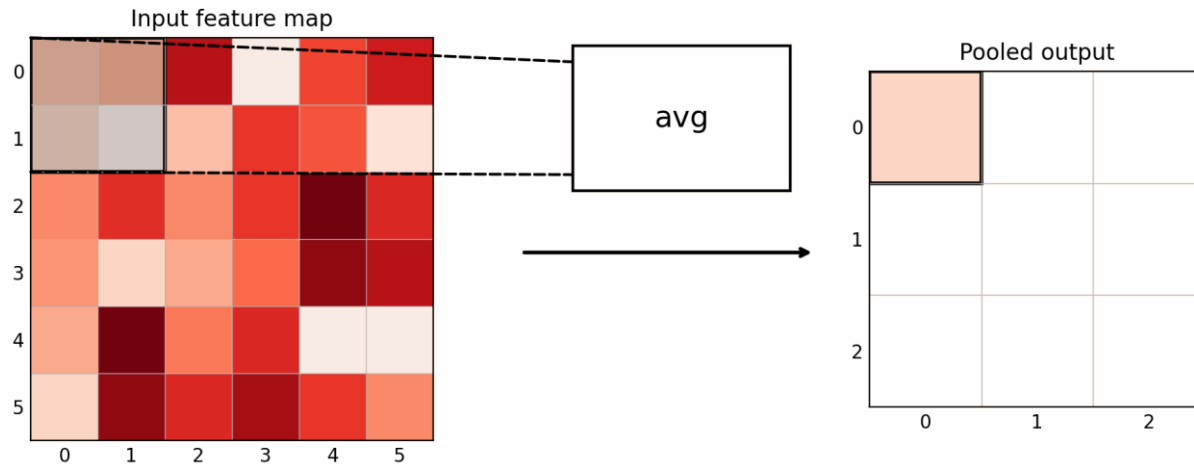
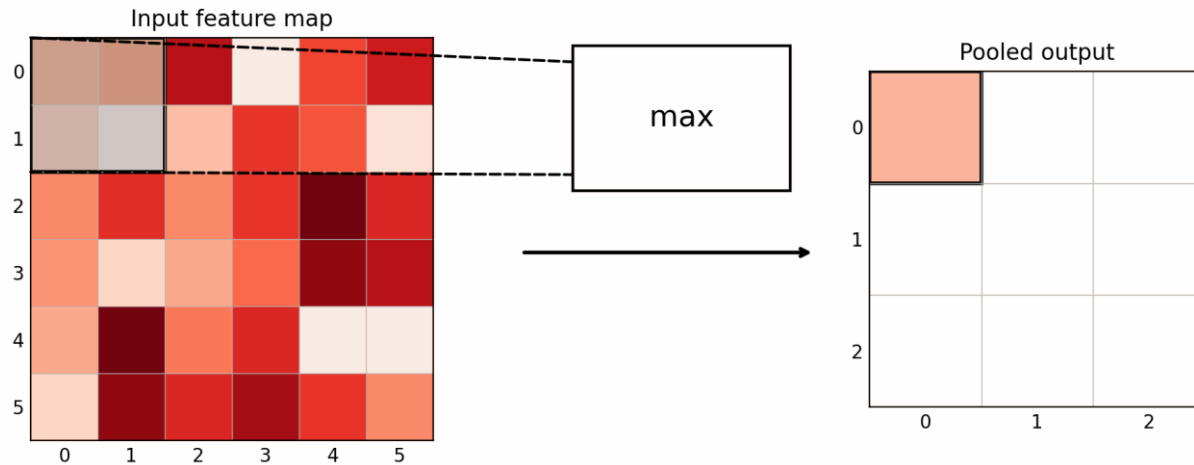
How Many Parameters?



32x32x3 image



Pooling Layer



MaxPool2d

```
class torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False)
```

[\[source\]](#)

Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W) , output (N, C, H_{out}, W_{out}) and **kernel_size** (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0, \dots, kH-1} \max_{n=0, \dots, kW-1} input(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n)$$

AvgPool2d

```
class torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False,
count_include_pad=True, divisor_override=None)
```

[\[source\]](#)

Applies a 2D average pooling over an input signal composed of several input planes.

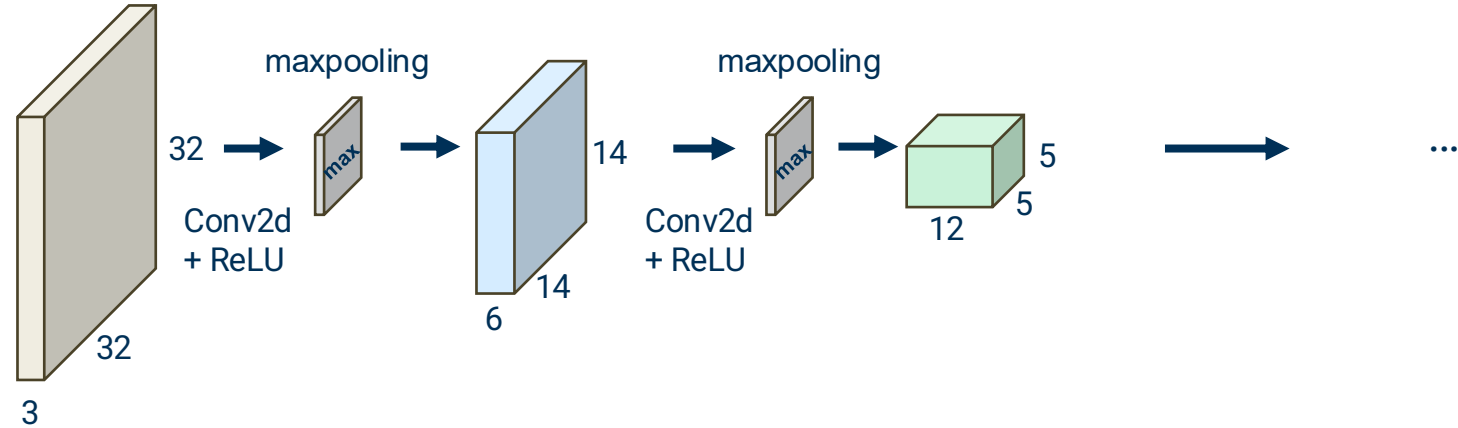
In the simplest case, the output value of the layer with input size (N, C, H, W) , output (N, C, H_{out}, W_{out}) and **kernel_size** (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \frac{1}{kH * kW} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} input(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n)$$

Adding Pooling to the Layers



32x32x3 image

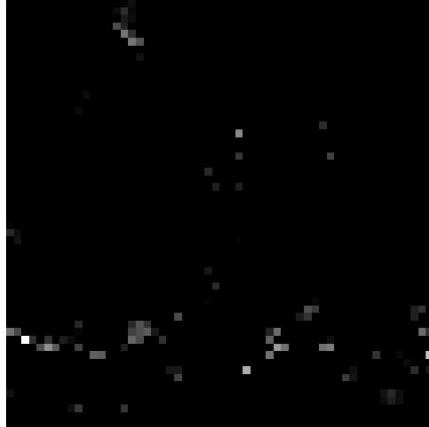


Convolutional Layers

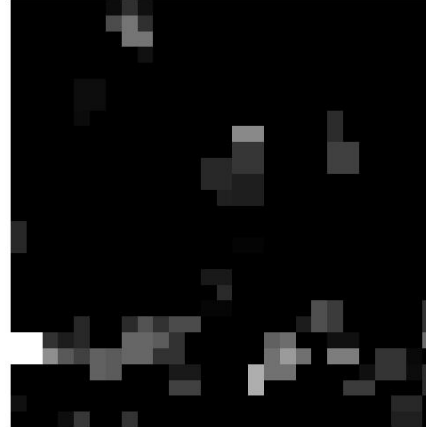
Input RGB (227×227)



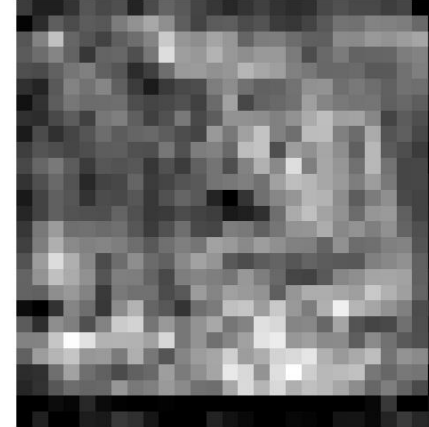
Conv1 (11×11,s4,p2) ch0 (56×56)



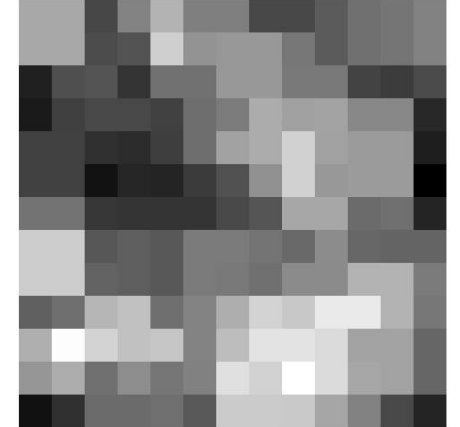
MaxPool 3×3,s2 (27×27)



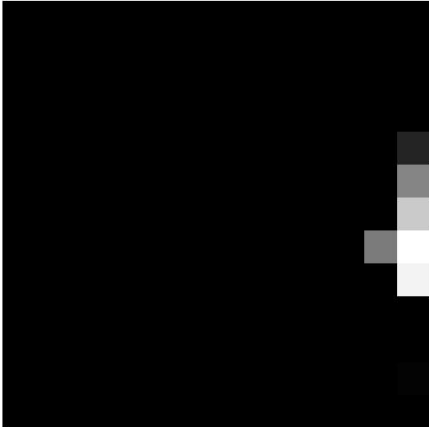
Conv2 (5×5,p2) ch0 (27×27)



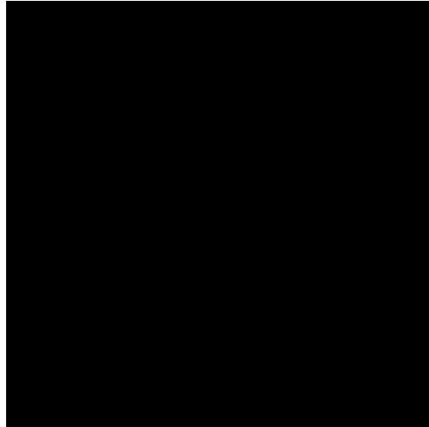
MaxPool 3×3,s2 (13×13)



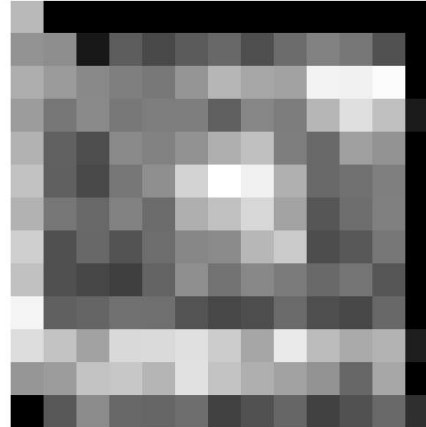
Conv3 (3×3,p1) ch0 (13×13)



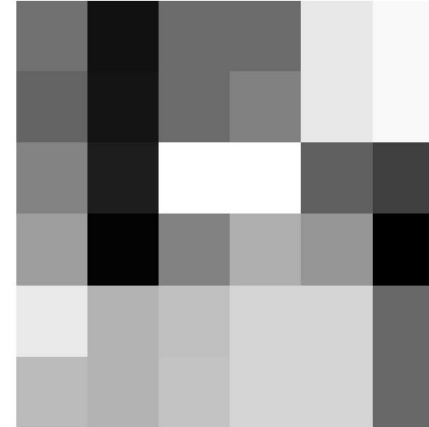
Conv4 (3×3,p1) ch0 (13×13)



Conv5 (3×3,p1) ch0 (13×13)



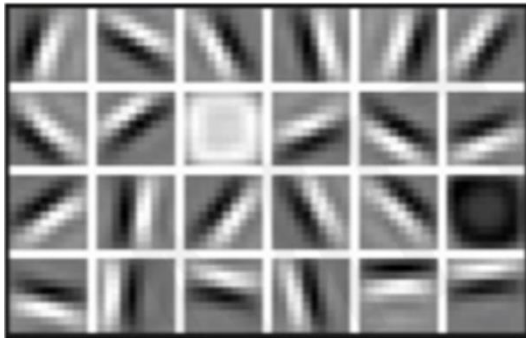
MaxPool 3×3,s2 (6×6)



Deep Learning

- Hand engineered filters are time consuming, brittle, and not scalable in practice
- Can we learn the filters directly from data?

Low level features



Lines & edges

Mid level features



Eyes & noses & ears

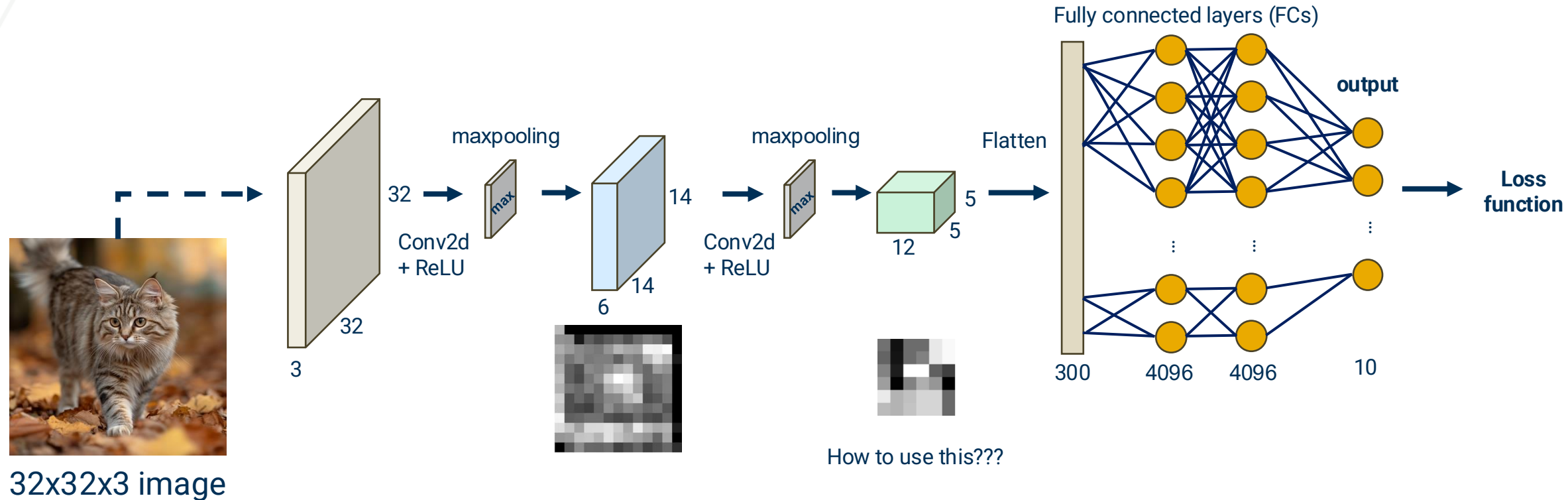
High level features



Facial structure

CNN Models

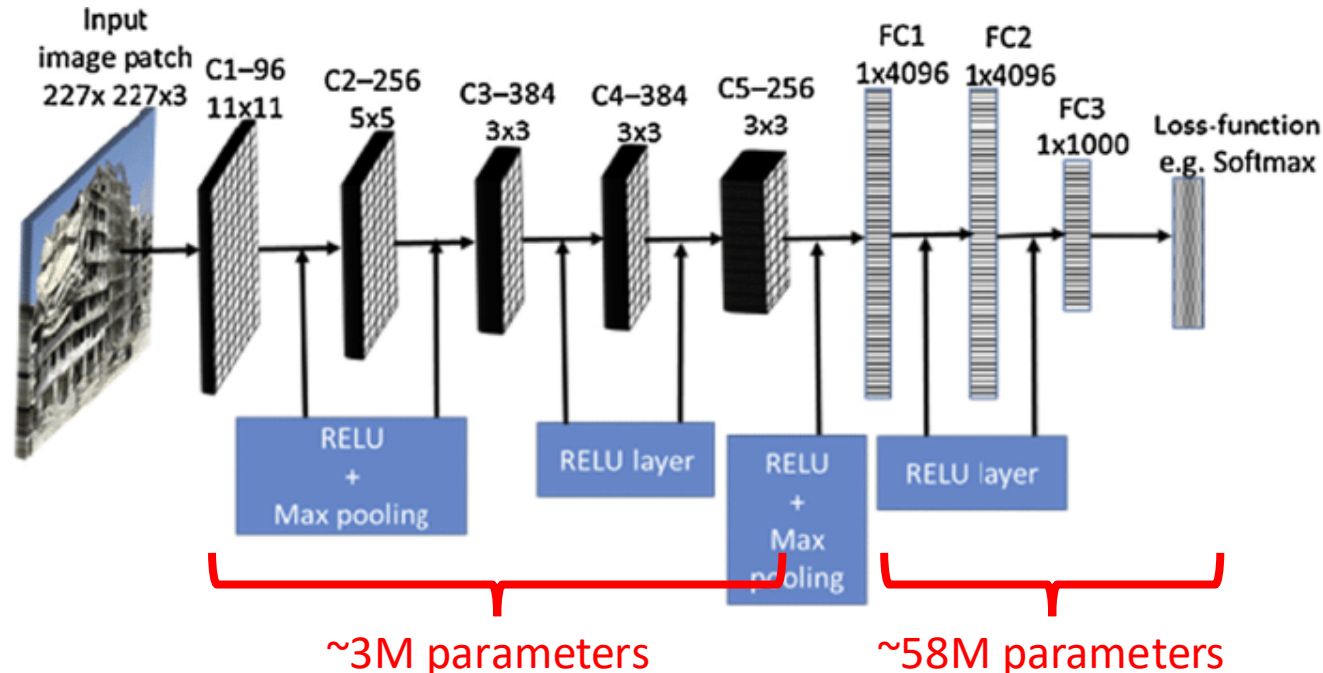
Convolutional Layers + Neural Networks



Question: what are the trainable parameters? Memory and computation cost?

AlexNet (Named after Alex Krizhevsky)

- **AlexNet** was one of the first deep convolutional on ImageNet competition with an accuracy of **84.7%** as compared to the second-best with an accuracy of **73.8%**.
- The activation used is the Rectified Linear Unit (ReLU).
- **How long did it take?**
 - The training set had 1.2 million images. It was trained for 90 epochs, which took **5-6 days** on two NVIDIA GTX 580 3GB GPUs.



VGGNet

Visual Geometry Group

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* & Andrew Zisserman*

Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen, az}@robots.ox.ac.uk

- Having **fixed size kernels (3x3)** rather than the variable used in AlexNet (11x11, 5x5, 3x3).
- Variable size kernels can be replicated by making use of multiple 3x3 kernels as building blocks.
- The deeper the better!

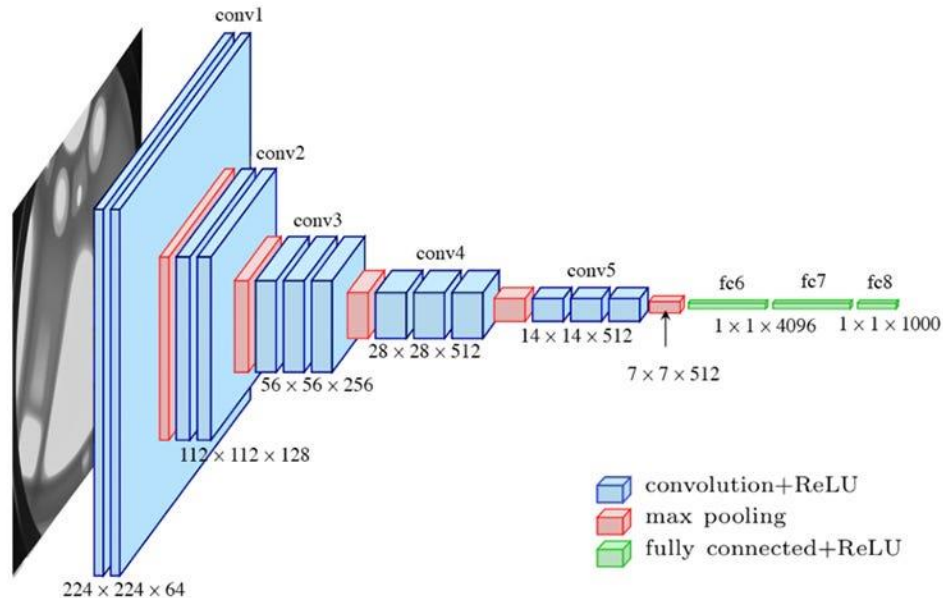


Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as “conv<receptive field size>-<number of channels>”. The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

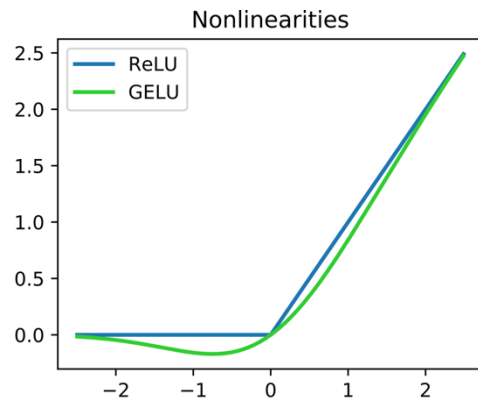
138M parameters

Vanishing Gradient Issue

- When models go deeper, gradient can vanish in the backpropagation...

- $$\frac{dl}{dw} = \frac{dl}{dz_{99}} \frac{dz_{99}}{dz_{98}} \frac{dz_{98}}{dz_{97}} \dots \frac{dz_1}{dz_0} \frac{dz_0}{dw}$$

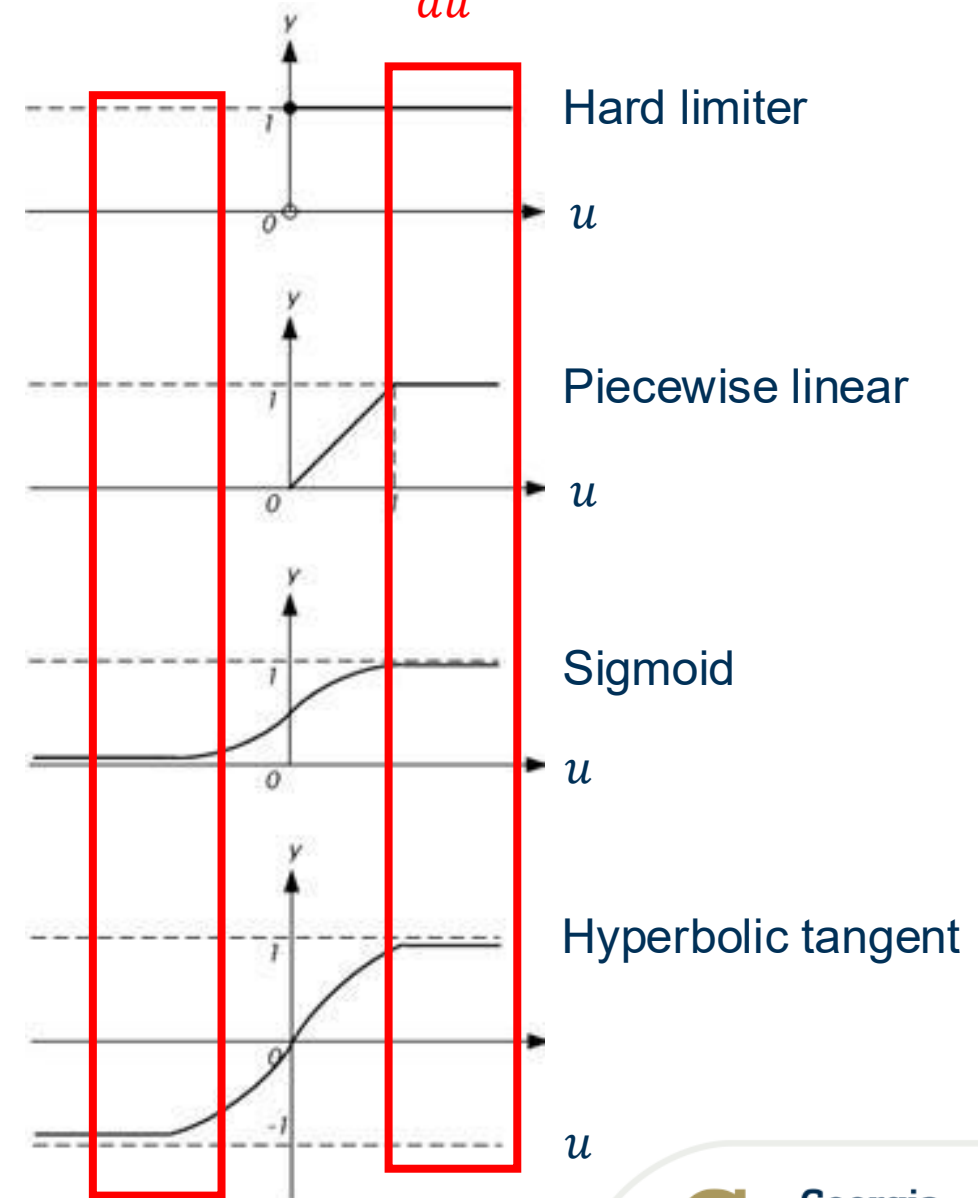
- Any term that is nearly 0 will ruin the entire chain rule.
- Activation functions play an important role.



ReLU (Rectified linear unit)

- Easy to backprop
- Preserve gradients (one-sided)
- Some variants to preserve more gradients

$$\frac{d\sigma(u)}{du} \approx 0$$



ResNet

- ResNet architecture makes use of shortcut connections to solve the vanishing gradient problem.
- Since the **vanishing gradient problem** was taken care of (more about it in the how part), CNN started to get deeper and deeper

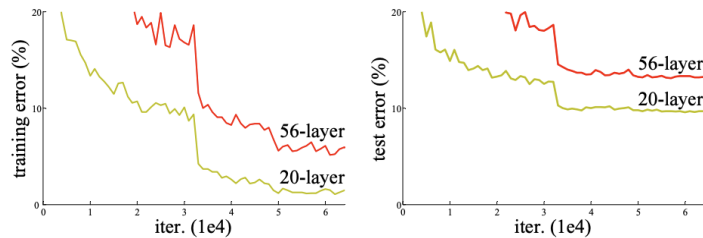
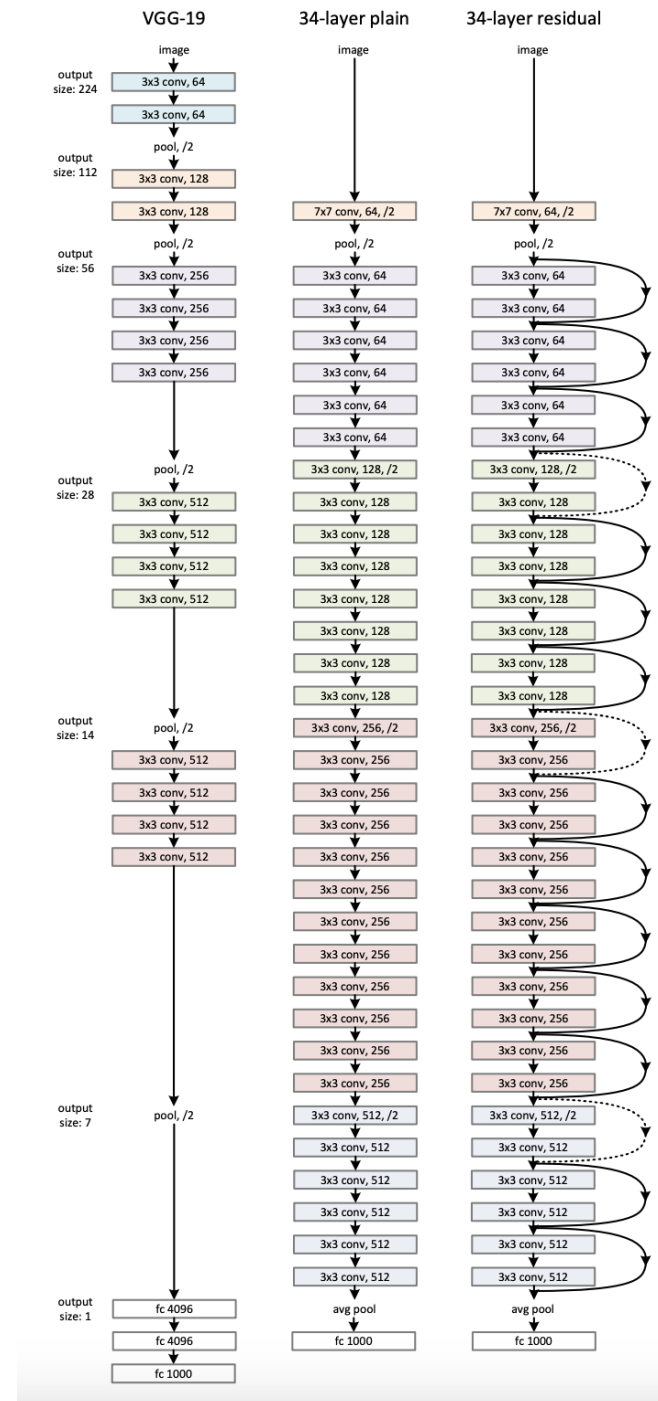
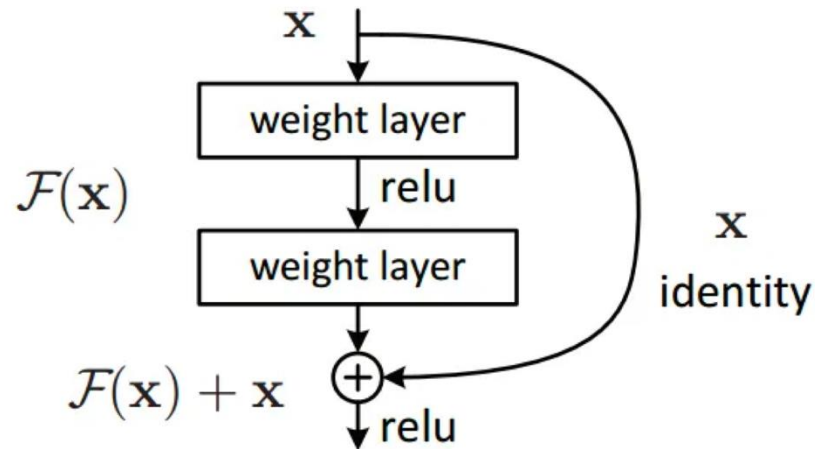


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.



Batch Normalization

- **Batch normalization** was proposed to normalize the input of any layer in neural networks.

$$\{x^{(i)}\}_{i \in \{1, 2, \dots, B\}} \in \mathbb{R}^{B \times d}$$

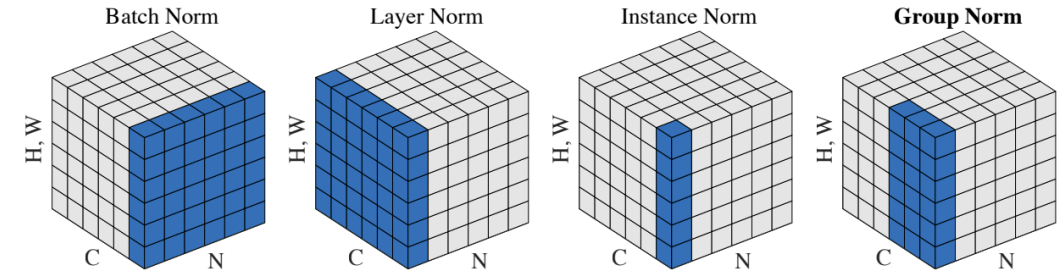
Compute the elementwise mean and variance

- $\mu_B = \frac{1}{B} \sum_i x^{(i)} \in \mathbb{R}^d$
- $\sigma_B^2 = \frac{1}{B} \sum_i (x^{(i)} - \mu_B)^2 \in \mathbb{R}^d$

Return batch-normalized features

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \in \mathbb{R}^d \quad \forall i$$

- It empirically works very well.
- However, there is no consensus why. Sometimes it may cause gradient instability too.



- [“Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, Lofe and Szegedy 2015](#)
- [“Group Normalization”, We and He 2018](#)

BatchNorm2d

```
class torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True,  
track_running_stats=True, device=None, dtype=None)
```

[\[source\]](#)

Applies Batch Normalization over a 4D input.

4D is a mini-batch of 2D inputs with additional channel dimension. Method described in the paper [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#).

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

Nowadays Deep Learning

Leaderboard

Dataset

View

Top 1 Accuracy

▼

by

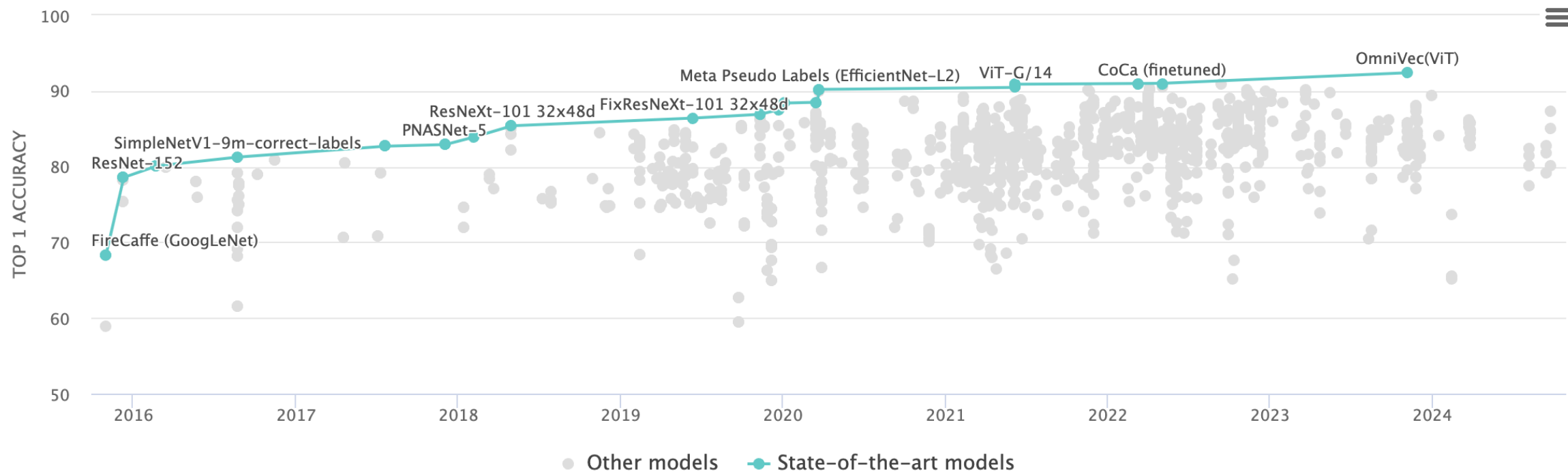
Date

▼

for

All models

▼



Nowadays Deep Learning

