

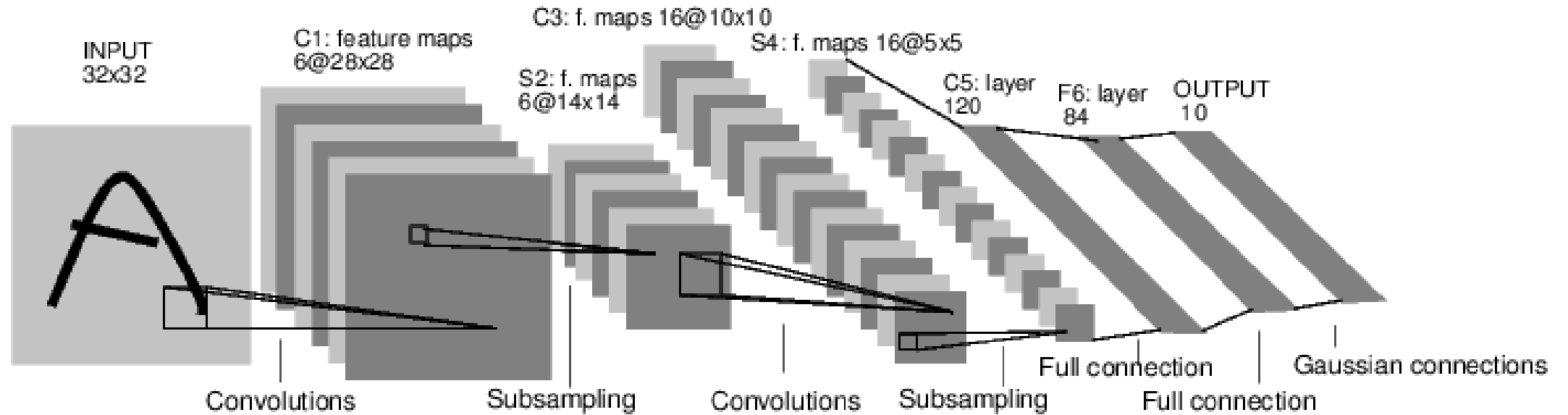
# Deep Learning for Vision & Language

Computer Vision: CNN Architecture Choices



RICE UNIVERSITY

# Convolutional Network: LeNet



Yann LeCun

TITLE

[Gradient-based learning applied to document recognition](#)

Y LeCun, L Bottou, Y Bengio, P Haffner  
Proceedings of the IEEE 86 (11), 2278-2324

CITED BY

83330

YEAR

2002

# LeNet in Pytorch

```
# LeNet is French for The Network, and is taken from Yann Lecun's 98 paper  
# on digit classification http://yann.lecun.com/exdb/lenet/  
# This was also a network with just two convolutional layers.  
class LeNet(nn.Module):  
    def __init__(self):  
        super(LeNet, self).__init__()  
        # Convolutional layers.  
        self.conv1 = nn.Conv2d(3, 6, 5)  
        self.conv2 = nn.Conv2d(6, 16, 5)  
  
        # Linear layers.  
        self.fc1 = nn.Linear(16*5*5, 120)  
        self.fc2 = nn.Linear(120, 84)  
        self.fc3 = nn.Linear(84, 10)  
  
    def forward(self, x):  
        out = F.relu(self.conv1(x))  
        out = F.max_pool2d(out, 2)  
        out = F.relu(self.conv2(out))  
        out = F.max_pool2d(out, 2)  
  
        # This flattens the output of the previous layer into a vector.  
        out = out.view(out.size(0), -1)  
        out = F.relu(self.fc1(out))  
        out = F.relu(self.fc2(out))  
        out = self.fc3(out)  
        return out
```

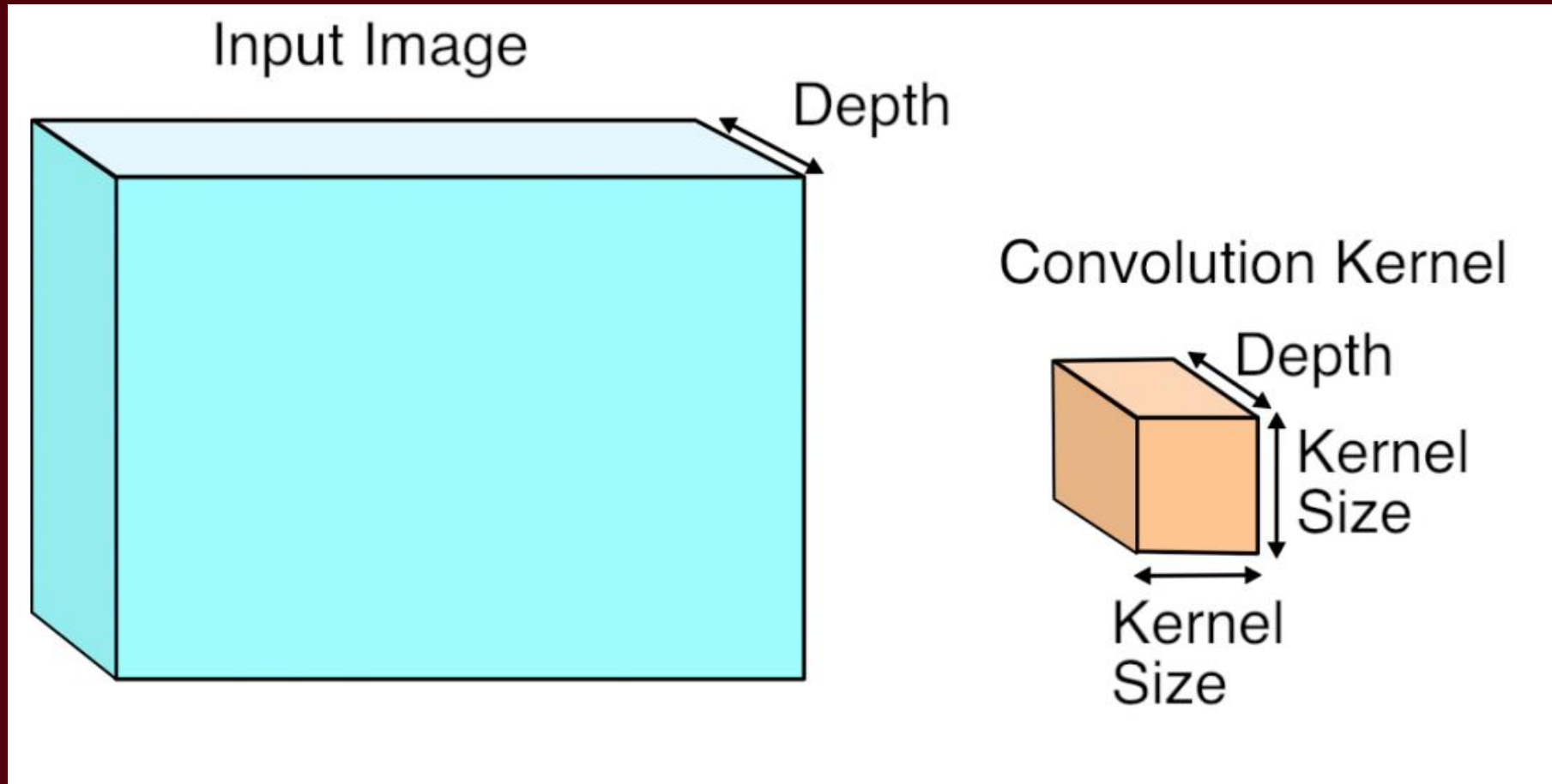
# LeNet Summary

- 2 Convolutional Layers + 3 Linear Layers
- + Non-linear functions: ReLUs or Sigmoids  
+ Max-pooling operations

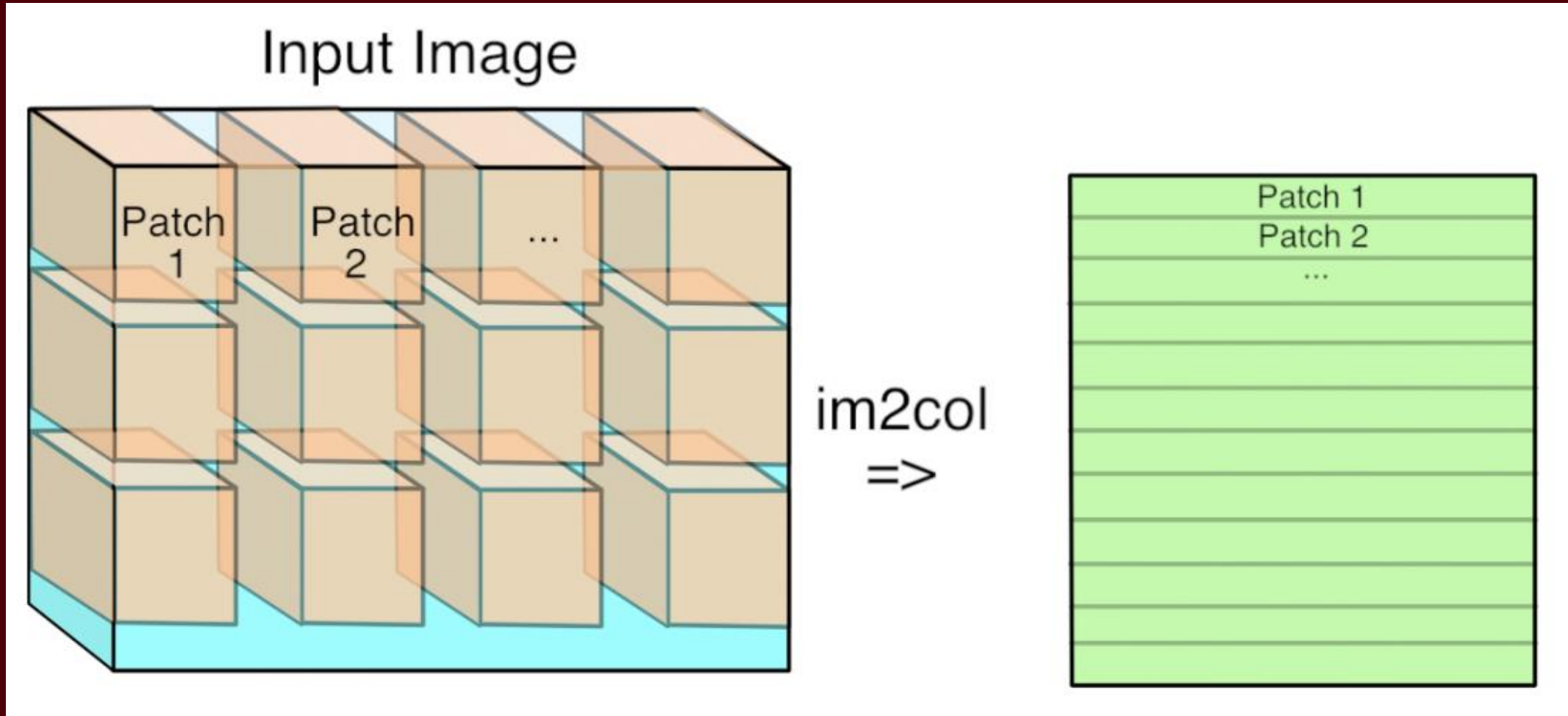
# New Architectures Proposed

- Alexnet (Krizhevsky et al NIPS 2012) [Required Reading]
- VGG (Simonyan and Zisserman 2014)
- GoogLeNet (Szegedy et al CVPR 2015)
- ResNet (He et al CVPR 2016)
- DenseNet (Huang et al CVPR 2017)

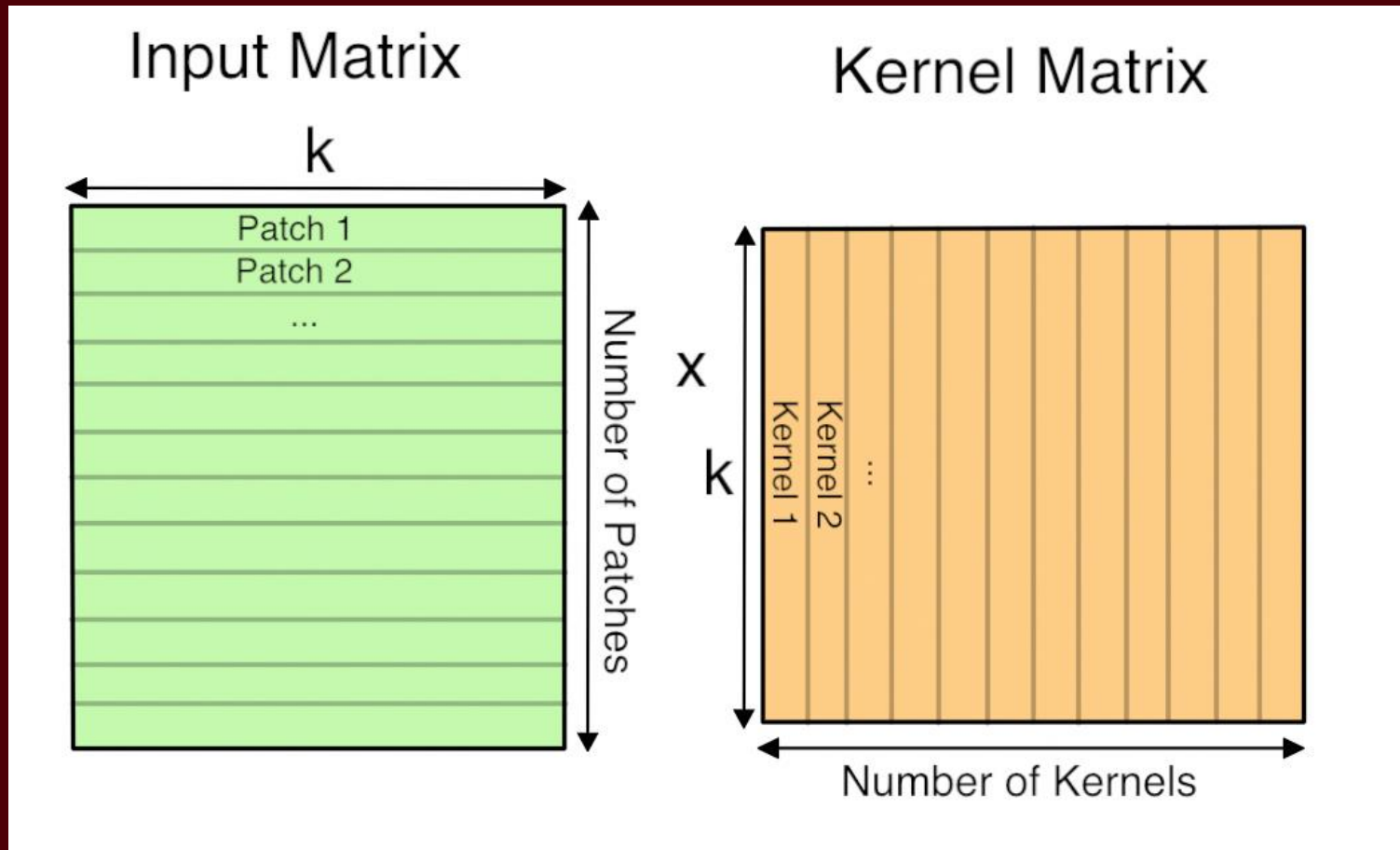
# Convolutional Layers as Matrix Multiplication



# Convolutional Layers as Matrix Multiplication



# Convolutional Layers as Matrix Multiplication



Pros?  
Cons?

# CNN Computations are Computationally Expensive

- However highly parallelizable
- GPU Computing is used in practice
- CPU Computing in fact is prohibitive for training these models

# The Alexnet network (Krizhevsky et al NIPS 2012)

---

## ImageNet Classification with Deep Convolutional Neural Networks

---

**Alex Krizhevsky**  
University of Toronto  
kriz@cs.utoronto.ca

**Ilya Sutskever**  
University of Toronto  
ilya@cs.utoronto.ca

**Geoffrey E. Hinton**  
University of Toronto  
hinton@cs.utoronto.ca

# The Problem: Classification

Classify an image into 1000 possible classes:

e.g. Abyssinian cat, Bulldog, French Terrier, Cormorant, Chickadee,  
red fox, banjo, barbell, hourglass, knot, maze, viaduct, etc.



cat, tabby cat (0.71)

Egyptian cat (0.22)

red fox (0.11)

.....

# The Data: ILSVRC

Imagenet Large Scale Visual Recognition Challenge (ILSVRC): Annual Competition

1000 Categories

~1000 training images per Category

~1 million images in total for training

~50k images for validation

Only images released for the test set but no annotations,  
evaluation is performed centrally by the organizers (max 2 per week)

# The Evaluation Metric: Top K-error

True label: Abyssinian cat

Top-1 error: 1.0

Top-1 accuracy: 0.0

Top-2 error: 1.0

Top-2 accuracy: 0.0

Top-3 error: 1.0

Top-3 accuracy: 0.0

Top-4 error: 0.0

Top-4 accuracy: 1.0

Top-5 error: 0.0

Top-5 accuracy: 1.0



cat, tabby cat (0.61)

Egyptian cat (0.22)

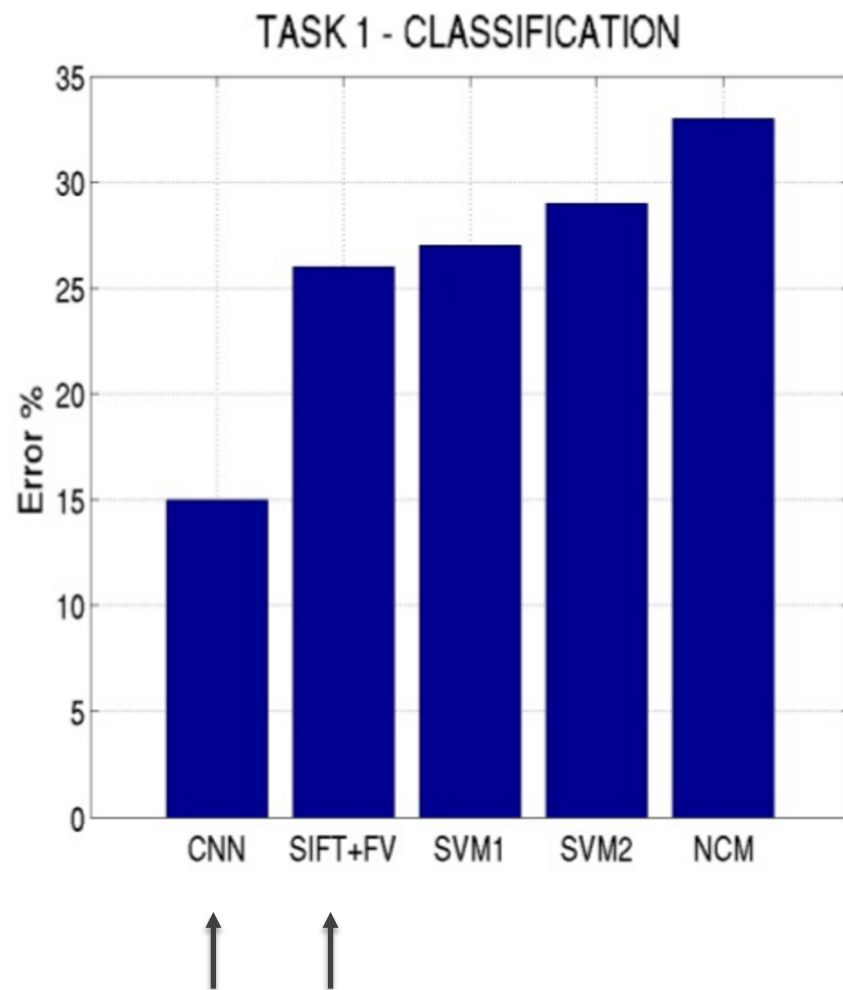
red fox (0.11)

Abyssinian cat (0.10)

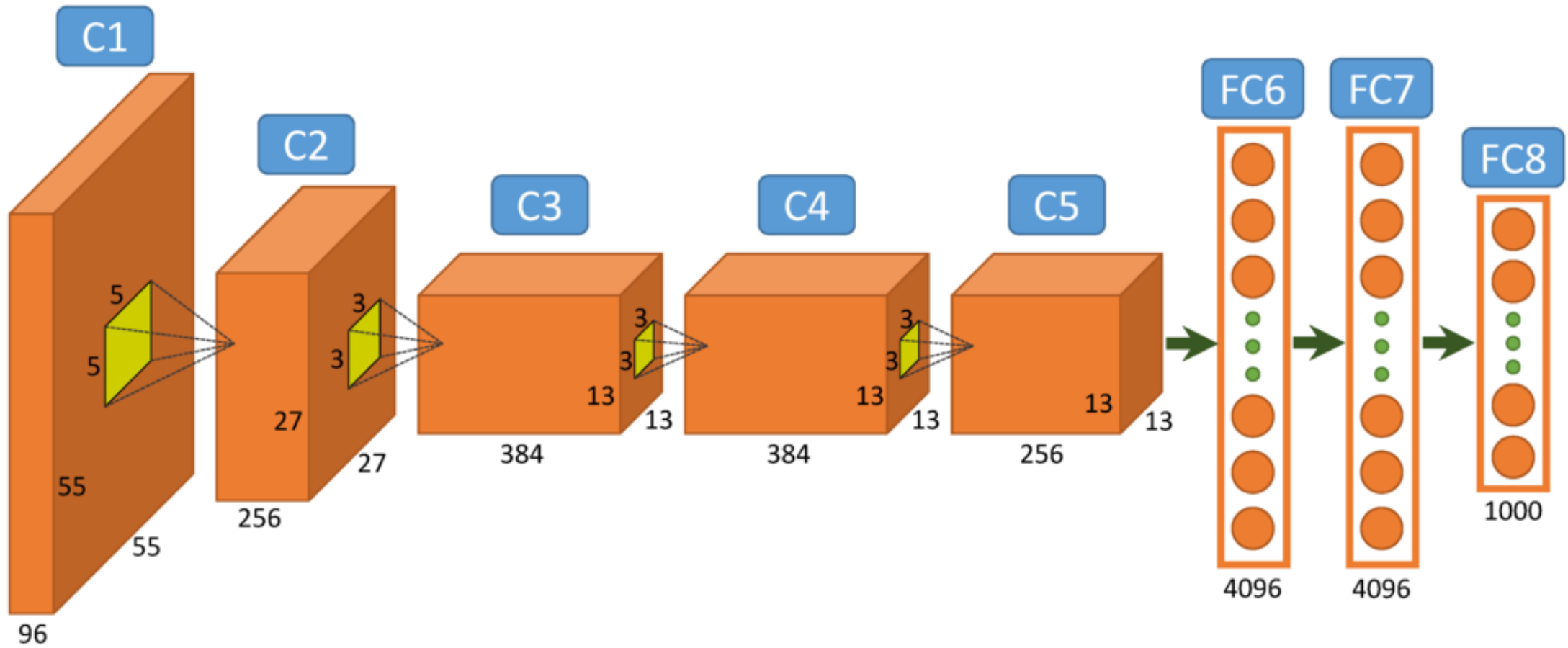
French terrier (0.03)

.....

# Top-5 error on this competition (2012)



# Alexnet



# Pytorch Code for Alexnet

- In-class analysis

<https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py>

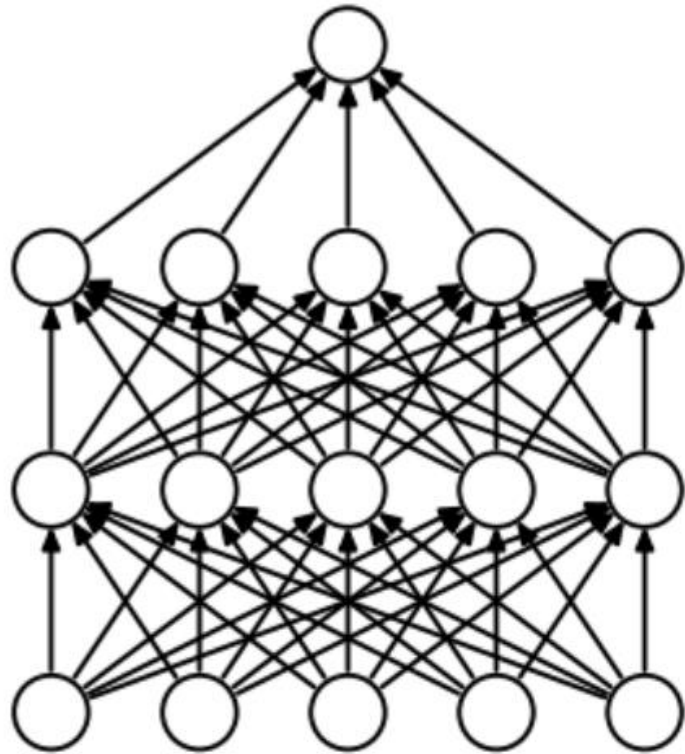
# AlexNet Pytorch

- nn.Sequential saves some coding on the forward() pass
- nn.ReLU(inplace = True) saves creating another tensor for the output.
- See how torch.flatten() works
- Why do we need self.avgpool?  
Do we need?

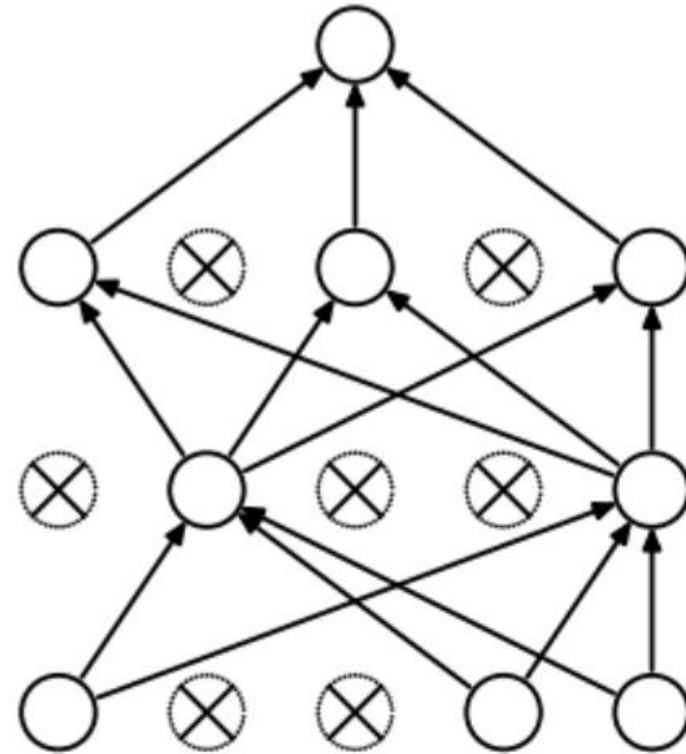
```
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super().__init__()
        _log_api_usage_once(self)
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

# Dropout Layer



(a) Standard Neural Net



(b) After applying dropout.

# Dropout Layer does not work the same during training and inference

```
import torch
import torch.nn as nn

class MyDropout(nn.Module):
    def __init__(self, p=0.5):
        super().__init__()
        self.p = p

    def forward(self, x):
        if (not self.training) or self.p == 0.0:
            return x

        keep = 1.0 - self.p
        mask = (torch.rand_like(x) < keep).float()
        return x * mask / keep
```

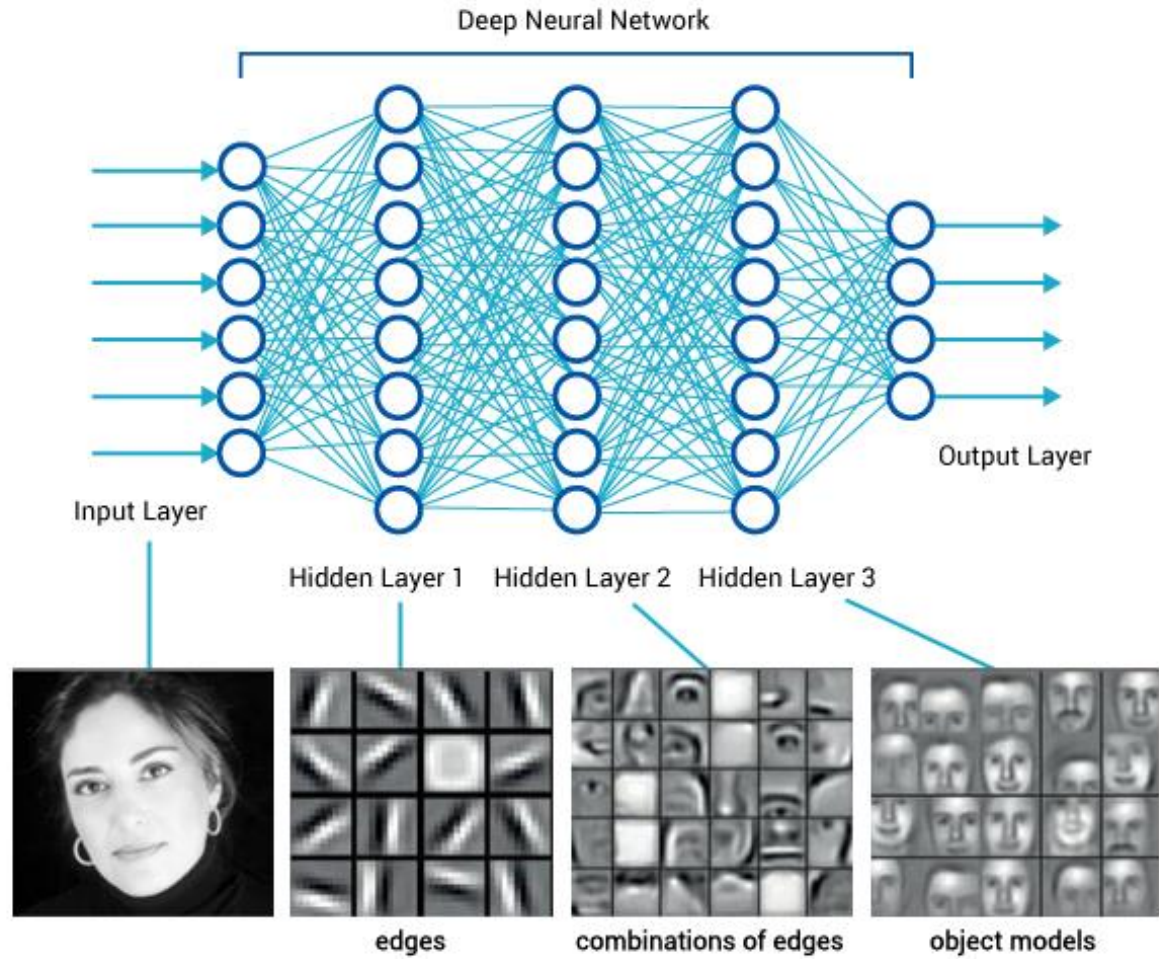
```
# quick demo
torch.manual_seed(0)
x = torch.ones(1, 8)

d = MyDropout(p=0.5)

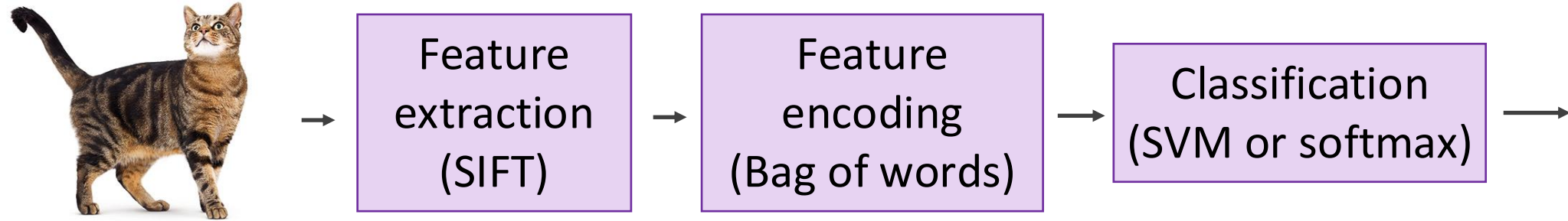
d.train()
print("train:", d(x))
print("train:", d(x))

d.eval()
print("eval :", d(x))
print("eval :", d(x))
```

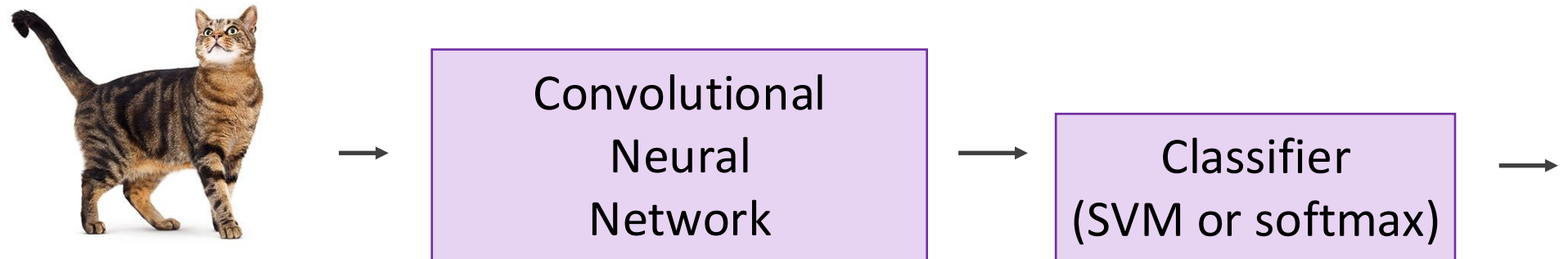
# What is happening?



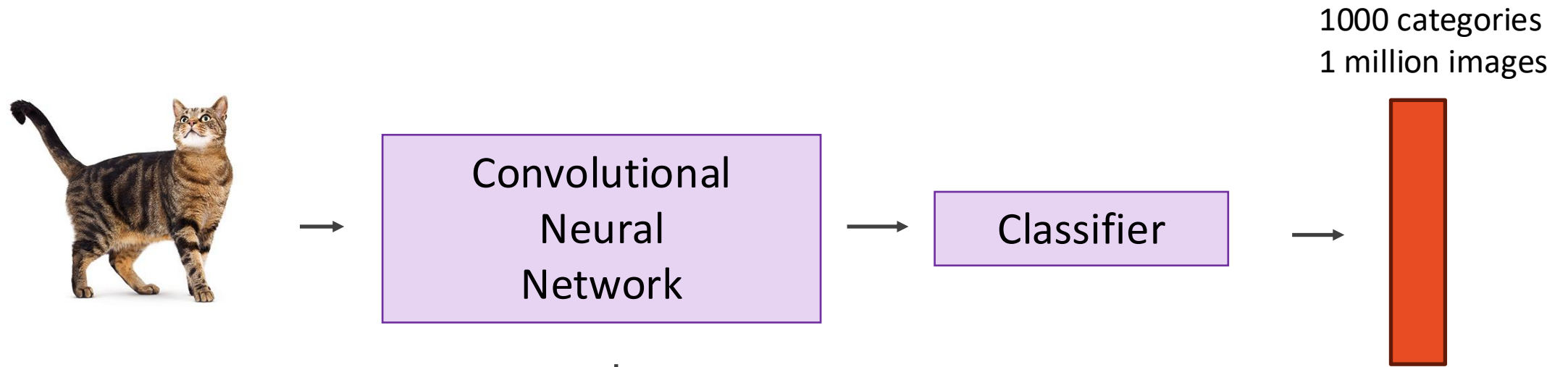
## SIFT + FV + SVM (or softmax)



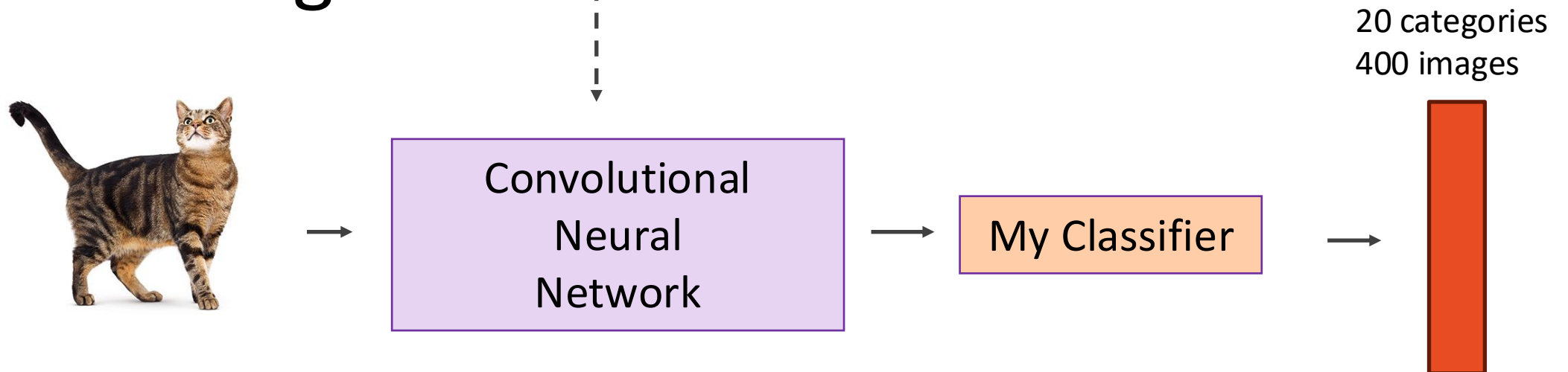
## Deep Learning



# Pre-training



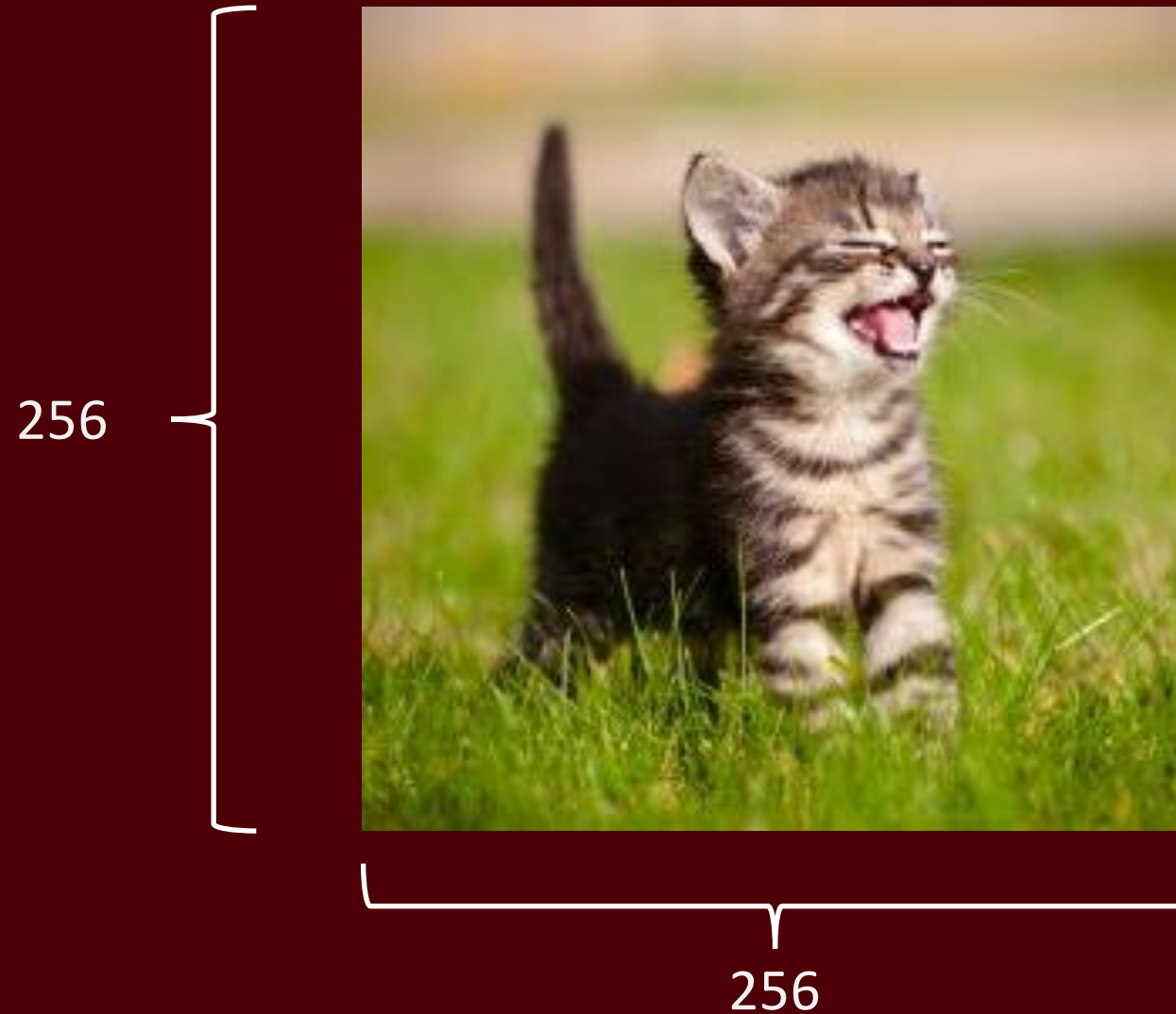
# Fine-tuning



# Preprocessing and Data Augmentation



# Preprocessing and Data Augmentation



# Preprocessing and Data Augmentation

224x224



# Preprocessing and Data Augmentation

224x224





True label: Abyssinian cat

# Other Data Augmentation options

Pytorch: <https://pytorch.org/vision/stable/transforms.html>

- **Basic:** RandomCrop, RandomResize, RandomHorizontalFlip
- **Geometric:** RandomRotation, RandomPerspective, RandomAffine, etc.
- **Color:** RandomAutoContrast, RandomEqualize, RandomPosterize, etc.
- **Other:** RandomErasing (<https://arxiv.org/abs/1708.04896>)

# Advanced Data Augmentation

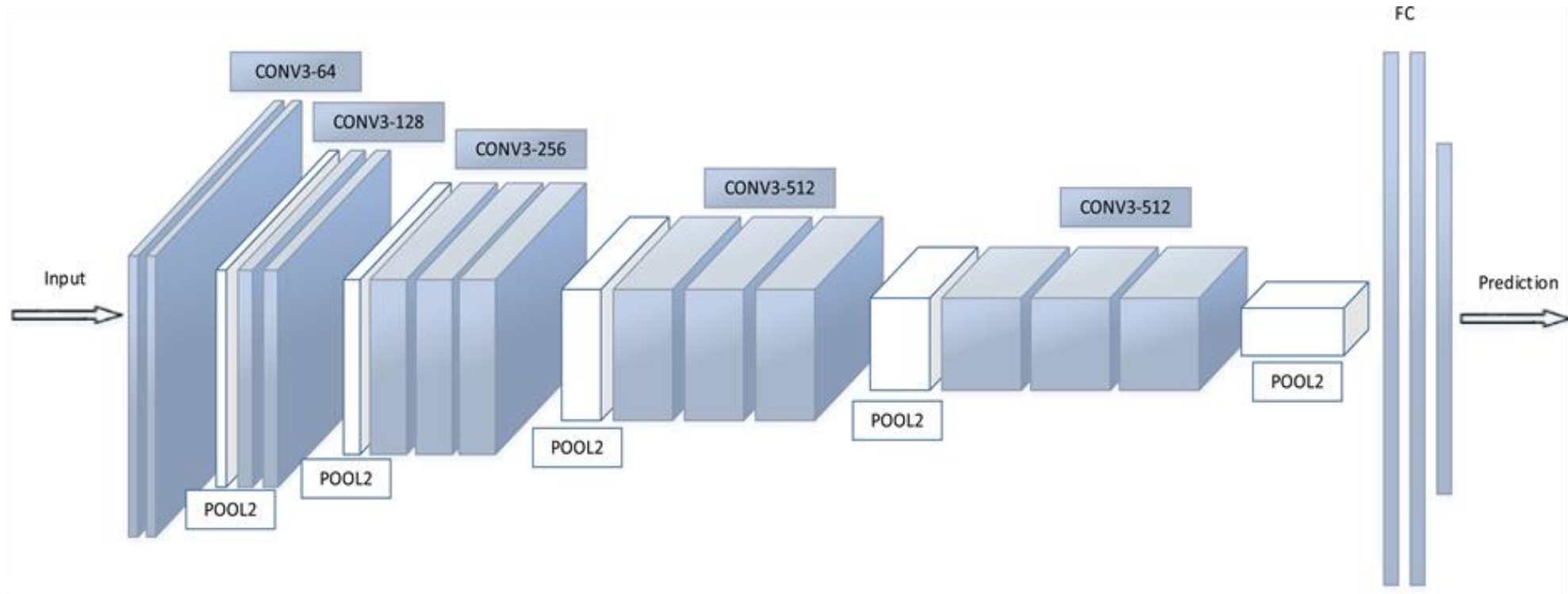
- MixUp – 2017 (<https://arxiv.org/abs/1710.09412>)
- AutoAugment - 2018 (<https://arxiv.org/abs/1805.09501>)
- RandAugment - 2019 (<https://arxiv.org/abs/1909.13719>)
- CutMix – 2019 (<https://arxiv.org/abs/1905.04899>)
- AugMix – 2019 (<https://arxiv.org/abs/1912.02781>)
- TrivialAugment - 2021 (<https://arxiv.org/abs/2103.10158>)

# Other Important Aspects

- Using ReLUs instead of Sigmoid or Tanh
- Momentum + Weight Decay
- Dropout (Randomly sets Unit outputs to zero during training)
- GPU Computation!

<b>Model</b>	<b>Top-1</b>	<b>Top-5</b>
<i>Sparse coding [2]</i>	47.1%	28.2%
<i>SIFT + FVs [24]</i>	45.7%	25.7%
<b>CNN</b>	<b>37.5%</b>	<b>17.0%</b>

# VGG Network

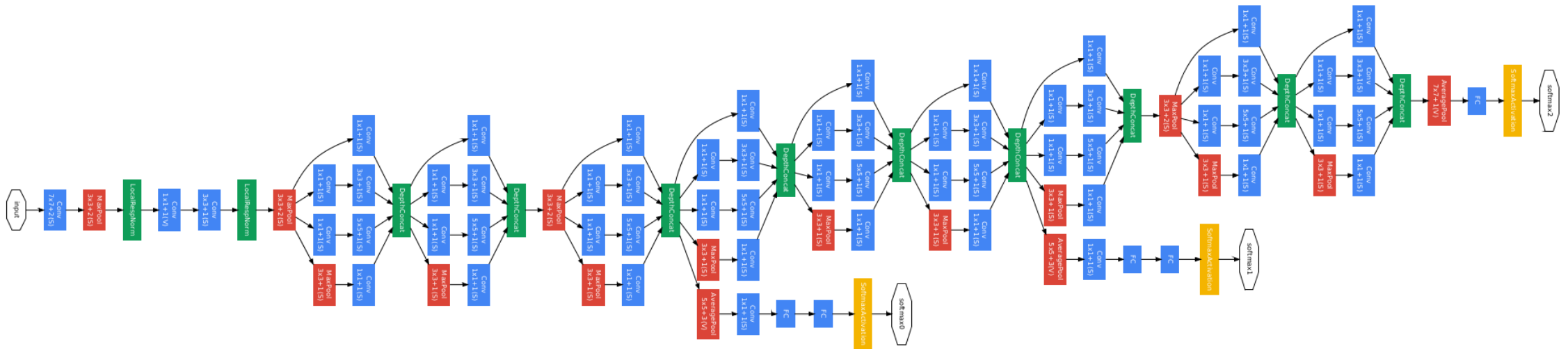


<https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py>

Simonyan and Zisserman, 2014.

<https://arxiv.org/pdf/1409.1556.pdf>

# GoogLeNet

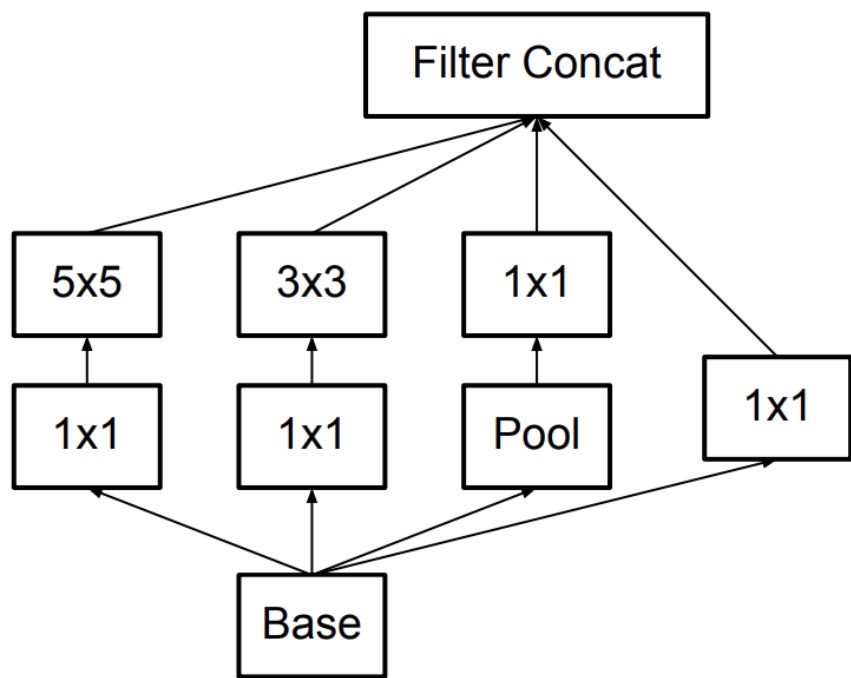


<https://github.com/kuangliu/pytorch-cifar/blob/master/models/googlenet.py>

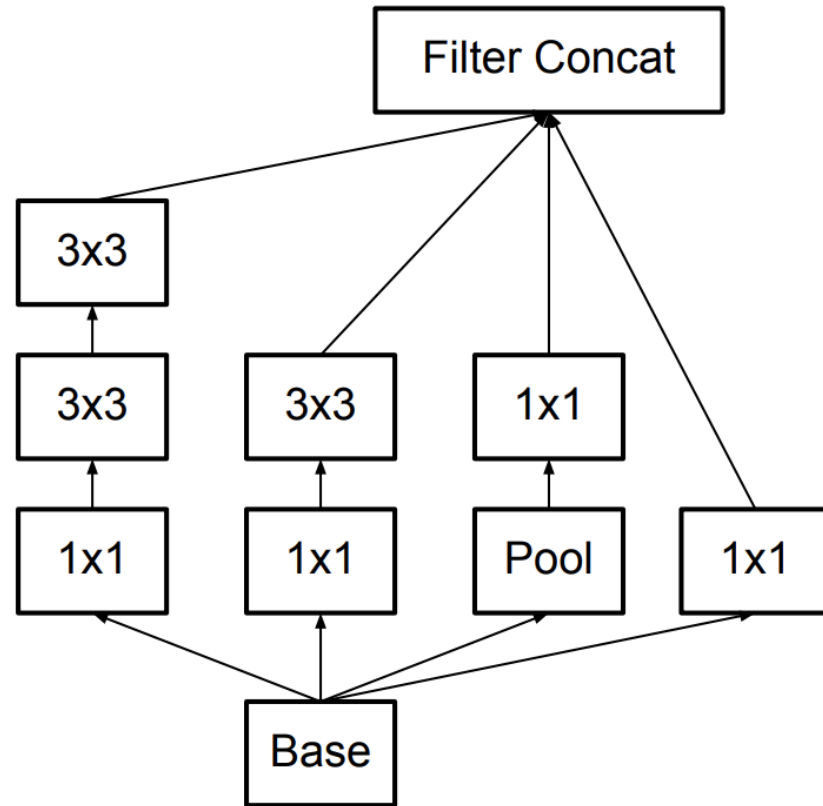
Szegedy et al. 2014

<https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf>

# Further Refinements – Inception v3, e.g.



GoogLeNet (Inceptionv1)



Inception v3

# Batch Normalization Layer

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

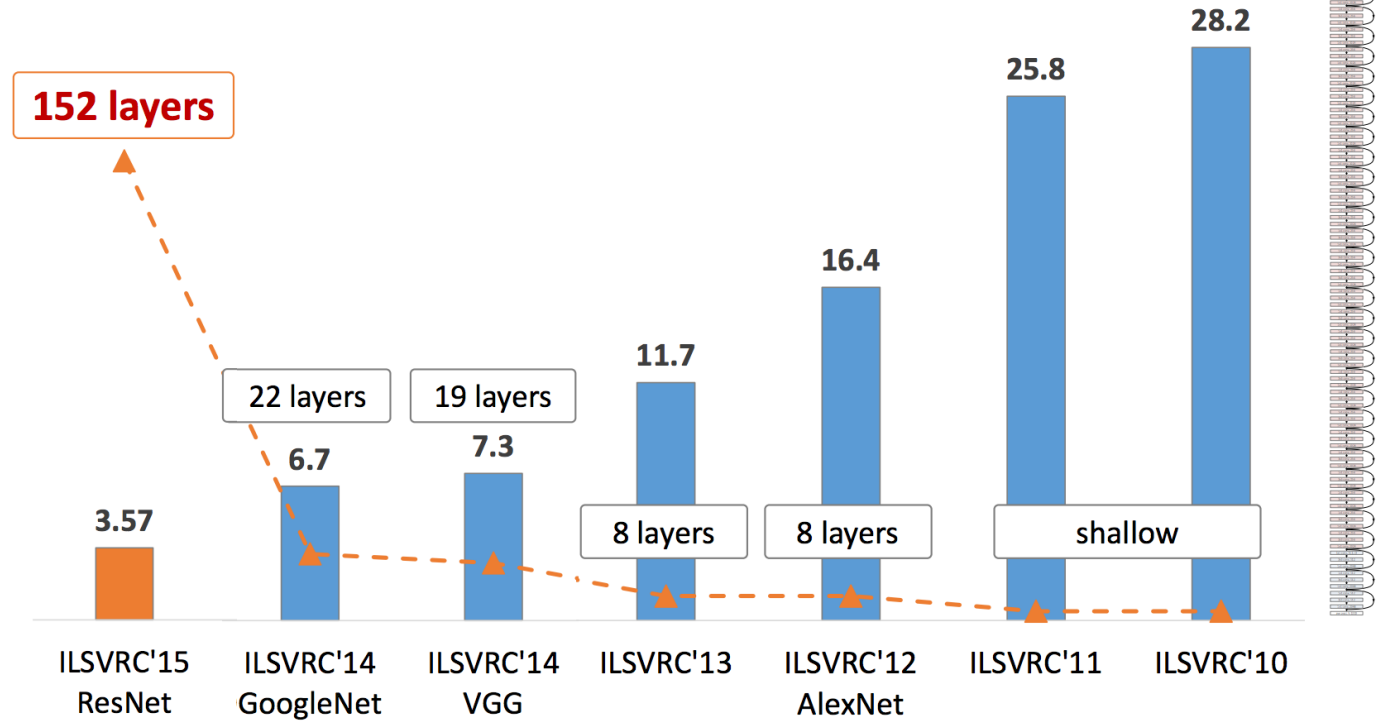
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

# Revolution of Depth

AlexNet, 8 layers  
(ILSVRC 2012)

VGG, 19 layers  
(ILSVRC 2014)

ResNet, 152 layers  
(ILSVRC 2015)



# ResNet (He et al CVPR 2016)

Sorry, does not fit in slide.

<http://felixlaumon.github.io/assets/kaggle-right-whale/resnet.png>

<https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py>

# ResNet: Residual Blocks

```
import torch.nn as nn
import torch.nn.functional as F

class ResBlock(nn.Module):
    #  $y = \text{ReLU}(F(x) + x)$  (same shape in/out)
    def __init__(self, c):
        super().__init__()
        self.f = nn.Sequential(
            nn.Conv2d(c, c, 3, padding=1, bias=False),
            nn.BatchNorm2d(c),
            nn.ReLU(inplace=True),          # ReLU inside F(x) is fine as a module
            nn.Conv2d(c, c, 3, padding=1, bias=False),
            nn.BatchNorm2d(c),
        )

    def forward(self, x):
        return F.relu(self.f(x) + x)      # functional ReLU after the skip add
```

# More Recent CNNs you can learn

- DenseNet - 2016 (Similar to ResNet but all layers connect to all next and previous layers) <https://arxiv.org/abs/1608.06993>
- ResNext – 2016 (Combines ideas from ResNet and InceptionNets) <https://arxiv.org/abs/1611.05431v2>
- NASNet – 2017 (Search the space of possible architectures) <https://arxiv.org/abs/1707.07012>
- EfficientNet – 2019 (Scales up both width and depth of CNNs using a more principled approach) <https://arxiv.org/abs/1905.11946>
- RegNet – 2020 (Paper has a more principled formula to decide on widths and depths) <https://arxiv.org/abs/2003.13678>
- ConvNext – 2022 (Combines many tricks both at the architecture and optimization level) <https://arxiv.org/abs/2201.03545>

# Questions?