

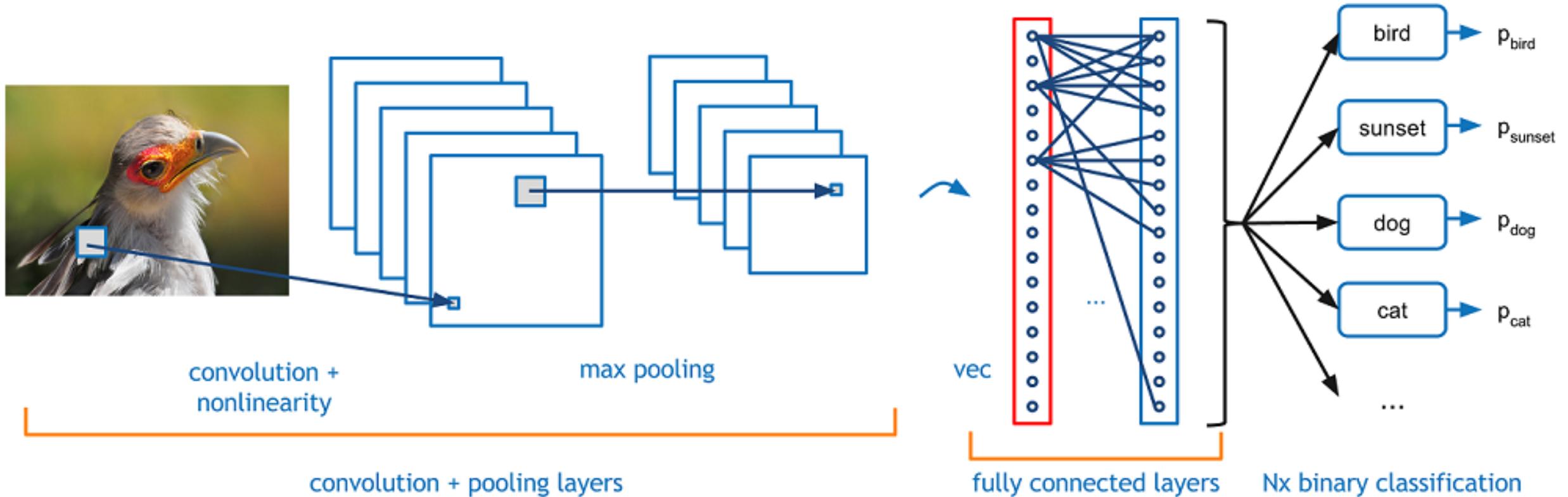
Kuan-Ting Lai
2020/3/31

Convolutional Neural Network



Convolutional Neural Networks (CNN)

- A.k.a. CNN or ConvNet



Digital Images

- Input array: an image's height × width × 3 (RGB)
- Value of each pixel: 0 - 255



What We See

08 02 22 97 38 15 00 40 00 75 04 05 07 76 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 59 41 92 36 54 22 40 40 28 66 33 13 60
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 58 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 49 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

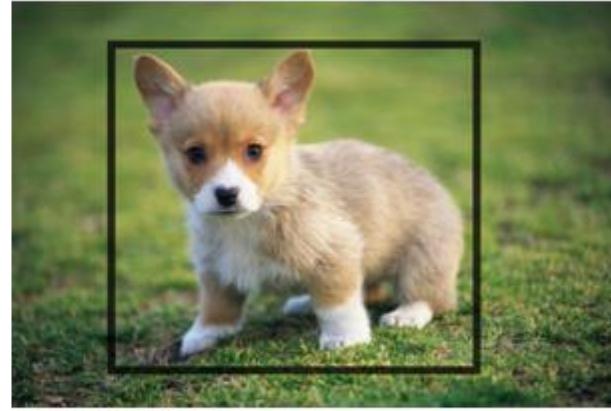
What Computers See



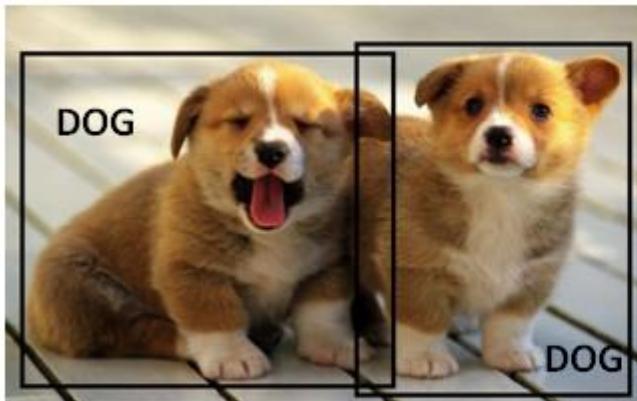
Classification, Localization, Detection, Segmentation



Object Classification is the task of identifying that picture is a dog



Object Localization involves the class label as well as a bounding box to show where the object is located.



Object Detection involves localization of multiple objects (doesn't have to be the same class).



Object Segmentation involves the class label as well as an outline of the object in interest.



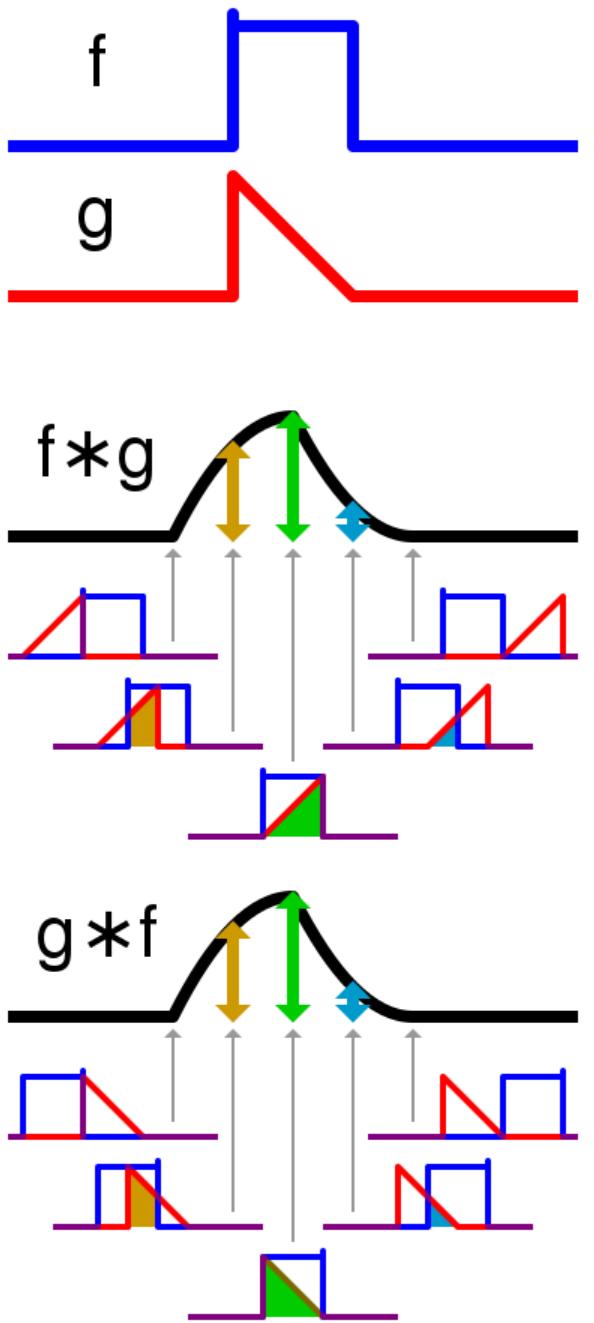
Convolution Theorem

- Fourier transform of a convolution of two signals is the pointwise product of their Fourier transforms

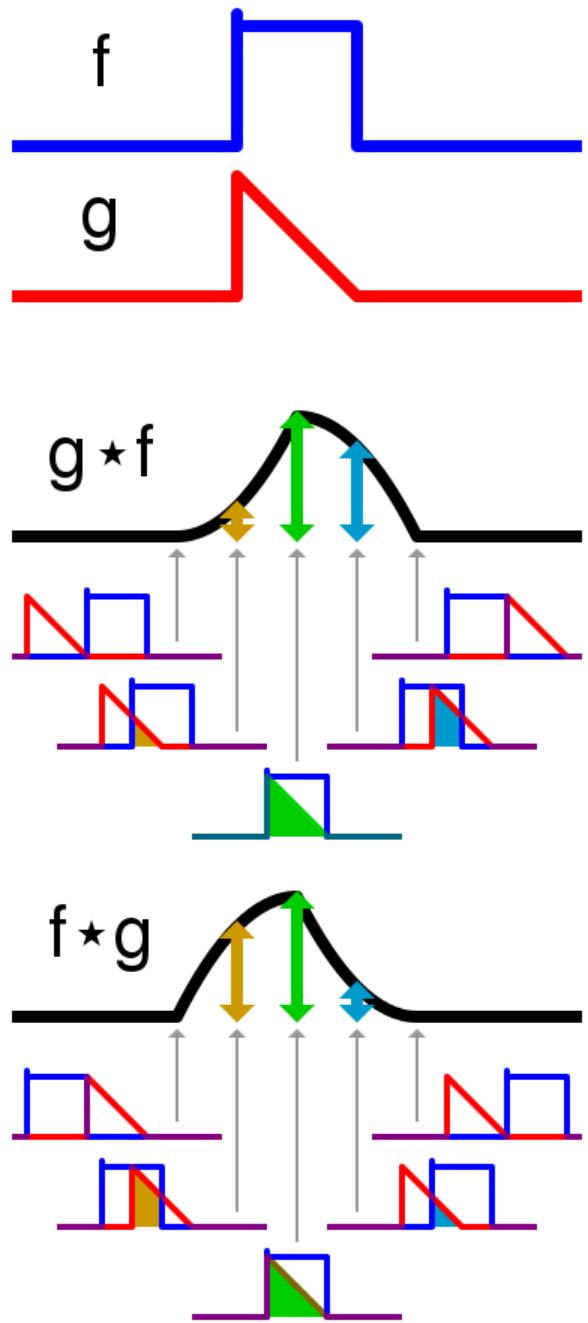
Spatial Domain	↔	Frequency Domain
$g = f * h$	↔	$G = FH$
$g = fh$	↔	$G = F * H$



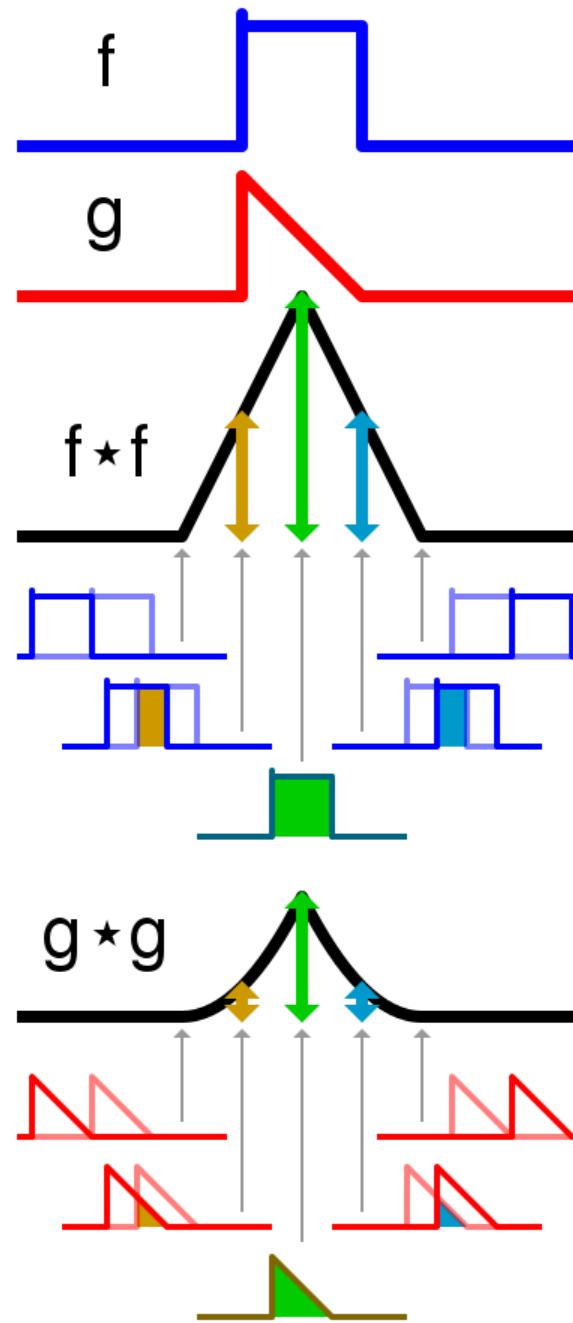
Convolution



Cross-correlation



Autocorrelation

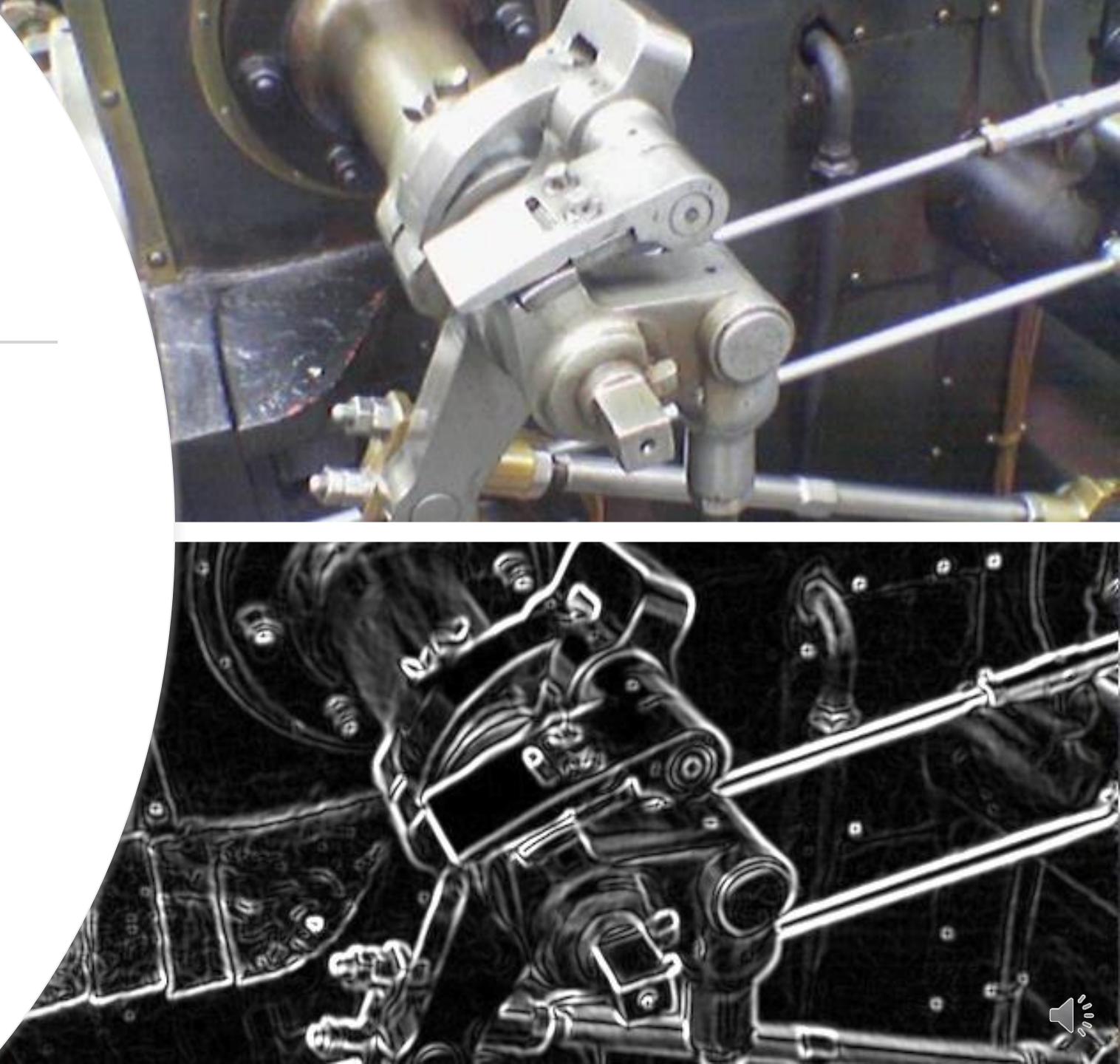


2D Convolution: Sobel Filter

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

https://en.wikipedia.org/wiki/Sobel_operator



0	0	0	0	0	0	
0	105	102	100	97	96	
0	103	99	103	101	102	
0	101	98	104	102	100	
0	99	101	106	104	99	
0	104	104	104	100	98	

Kernel Matrix

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

320						

Image Matrix

$$\begin{aligned}
 & 0 * 0 + 0 * -1 + 0 * 0 \\
 & + 0 * -1 + 105 * 5 + 102 * -1 \\
 & + 0 * 0 + 103 * -1 + 99 * 0 = 320
 \end{aligned}$$

Output Matrix

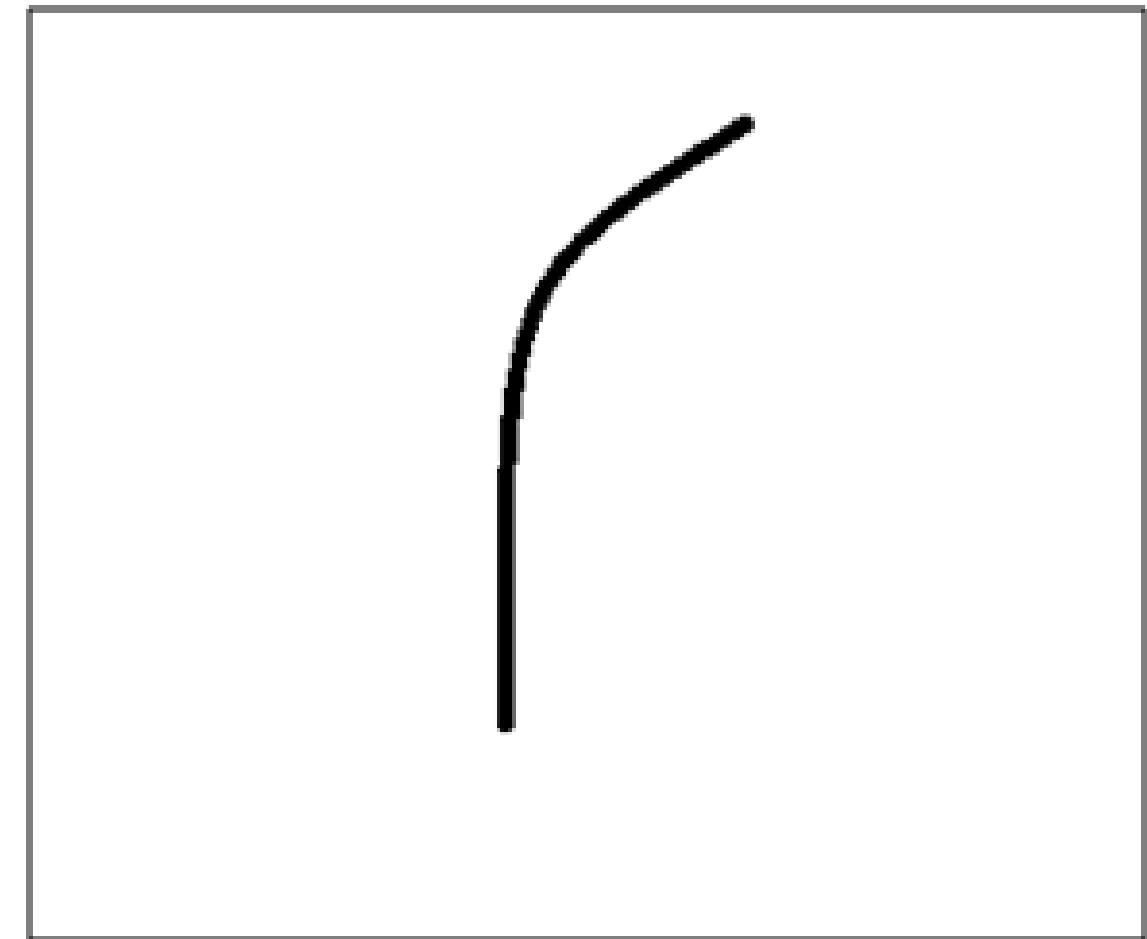


Convolution with horizontal and vertical strides = 1

Example: A Curve Filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



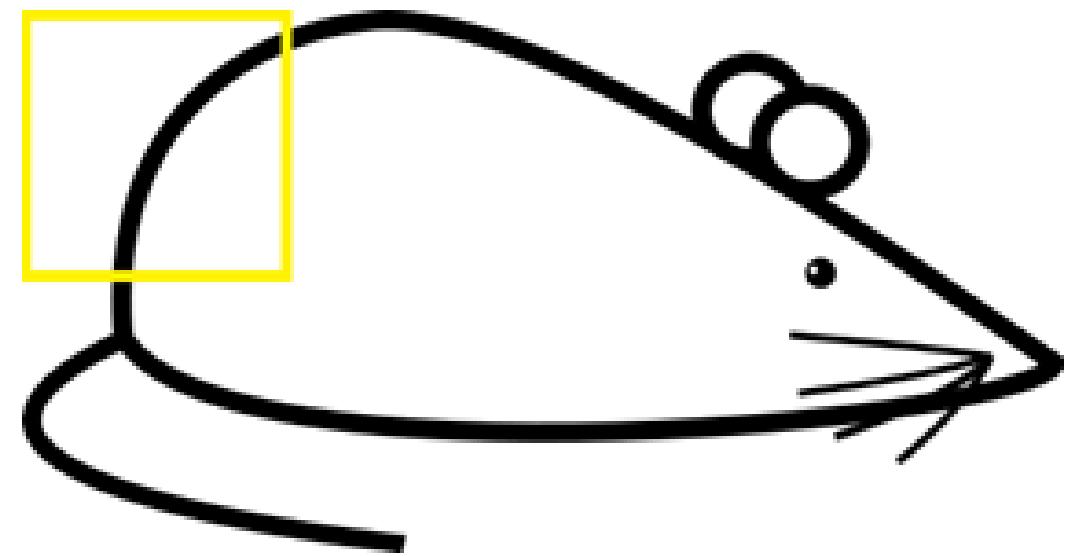
Visualization of a curve detector filter



Scan the Image to Detect an Edge



Original image



Visualization of the filter on the image



Edge Detected!



Visualization of the
receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive
field

*

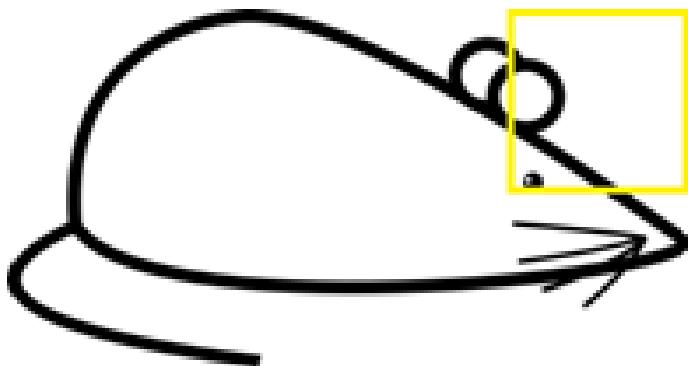
0	0	0	0	0	0	30	0
0	0	0	0	0	0	30	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

$$\text{Multiplication and Summation} = (50 * 30) + (50 * 30) + (50 * 30) + (20 * 30) + (50 * 30) = 6600 \text{ (A large number!)}$$



Continue Scanning (No edge)



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

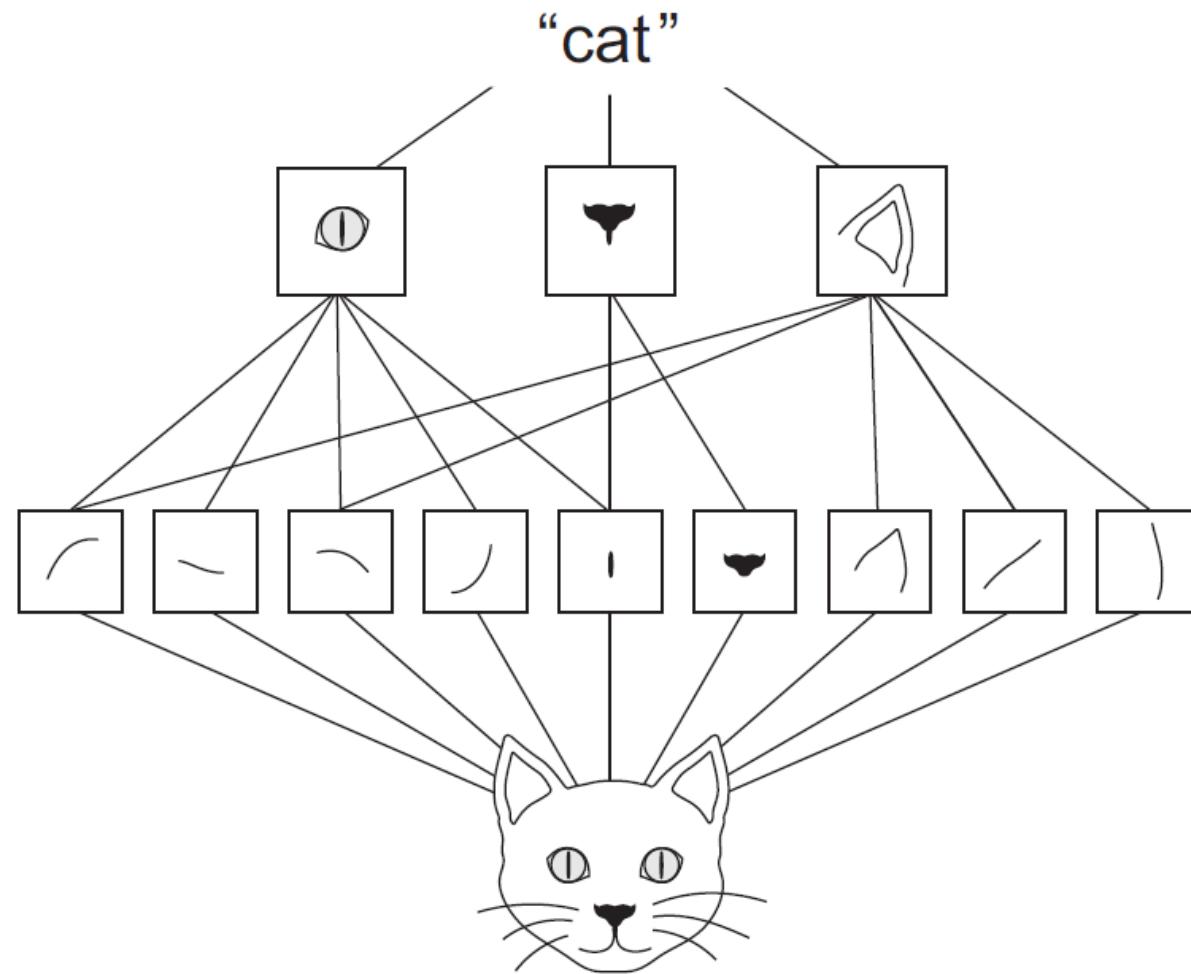
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0



Spatial Hierarchy of Features



Create First ConvNet

- Create a CNN to classify MNIST digits

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```



Model Summary

- `model.summary()`

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
maxpooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496 $= (3 \times 3 \times 32 + 1) \times 64$
maxpooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928

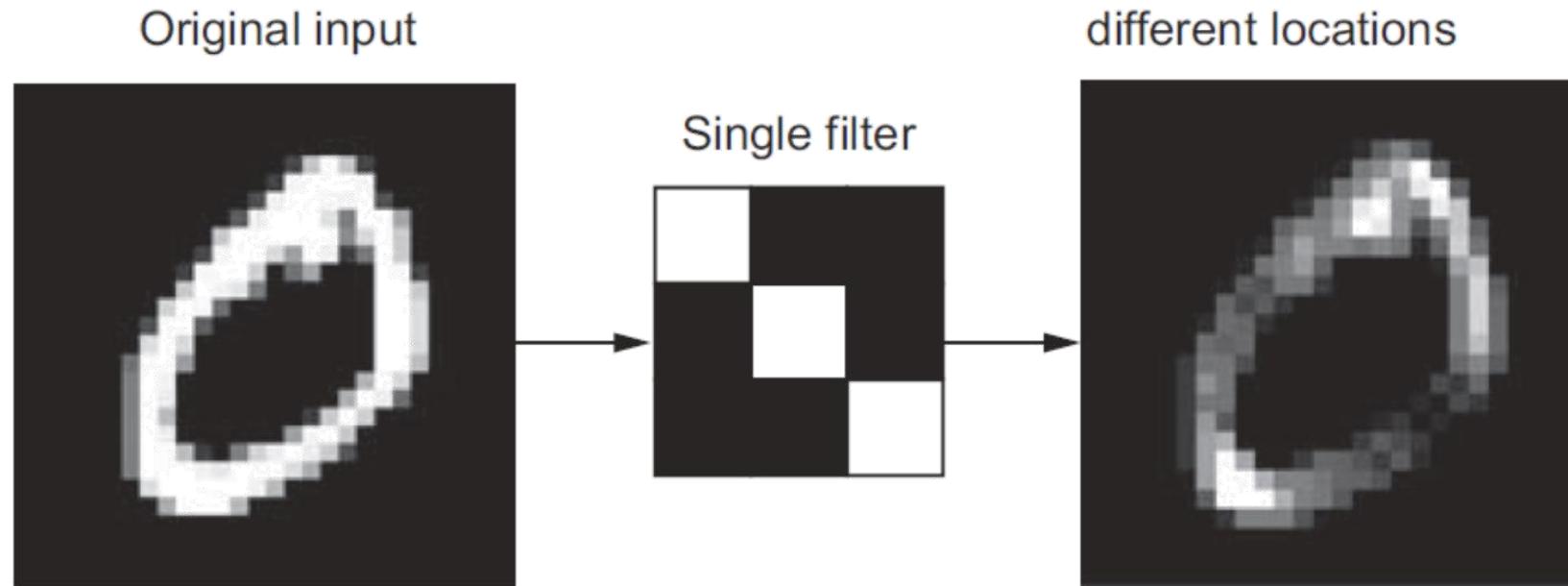


Feature Map

- Outputs of a Convolution Layer is also called as Feature Map

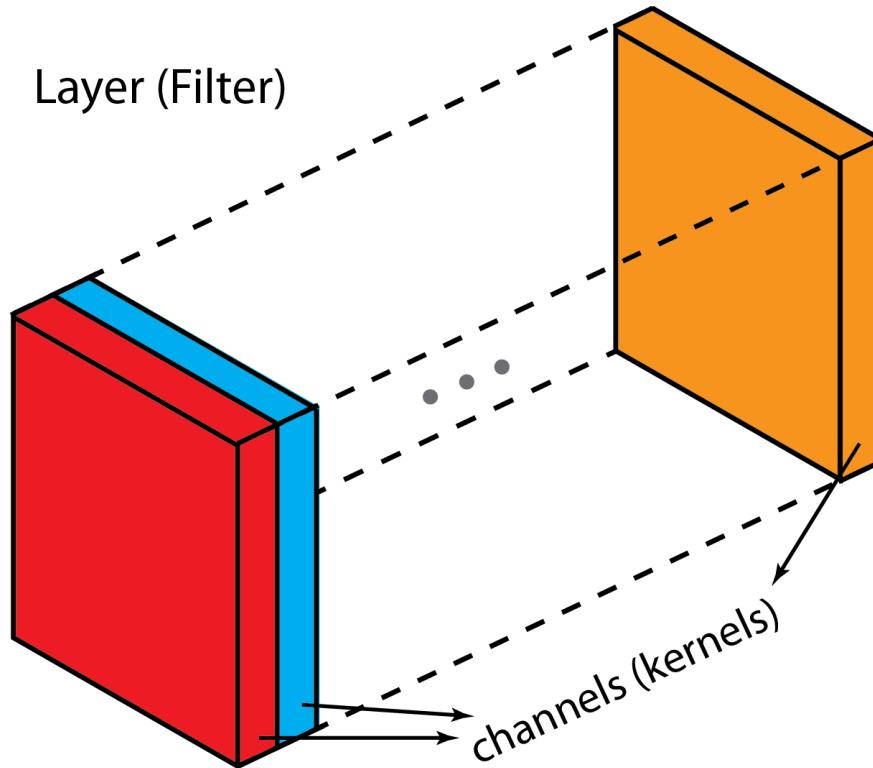
```
=>layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))
```

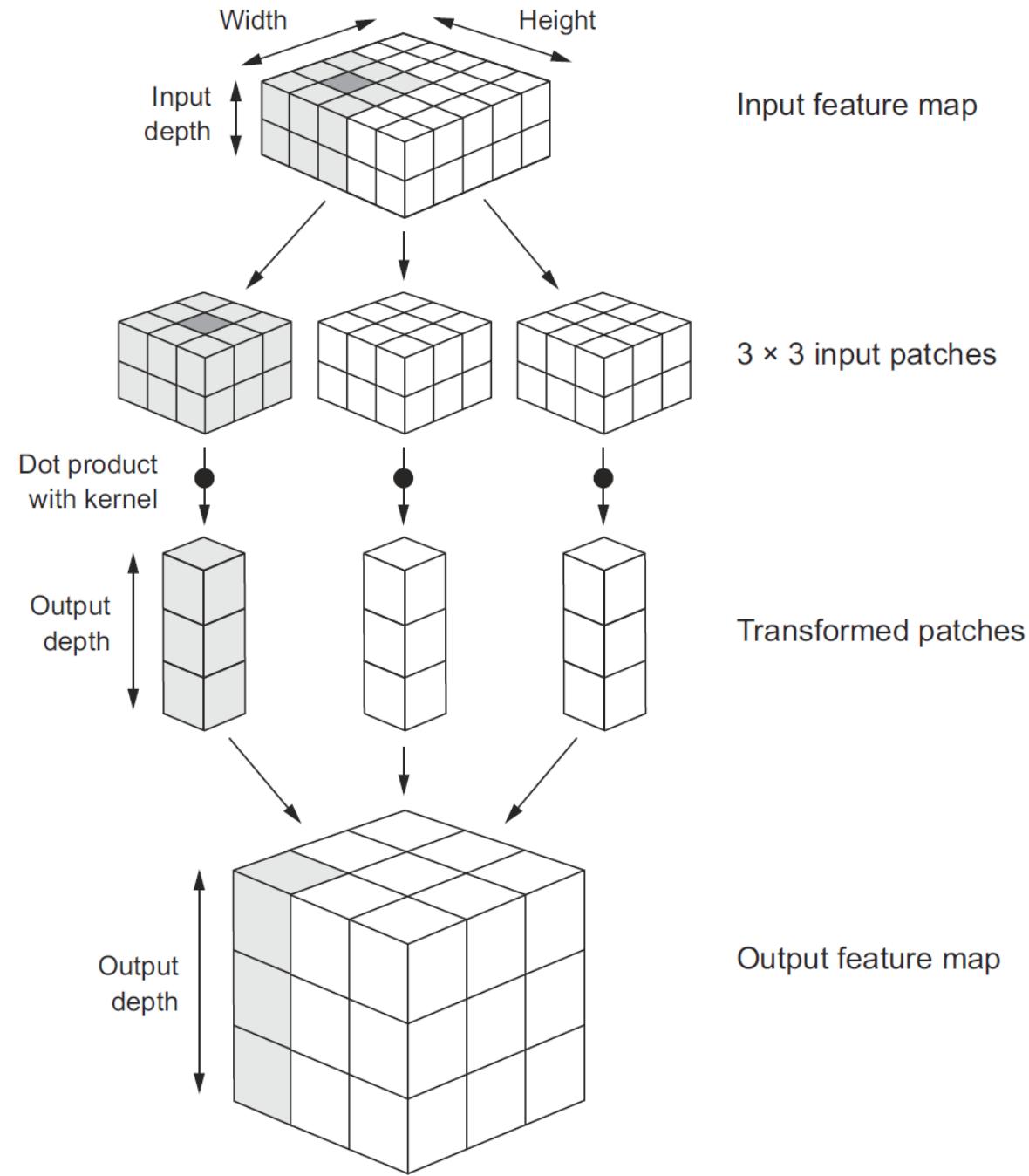
- Receive a 28x28 input image and computes 32 filters over it
- Each filter has size 3x3



Kernel and Filter in Deep Learning

- “Kernel” refers to a 2D array of weights.
- “filter” is for 3D structures of multiple kernels stacked together.





Add a Classifier on Top of ConvNet

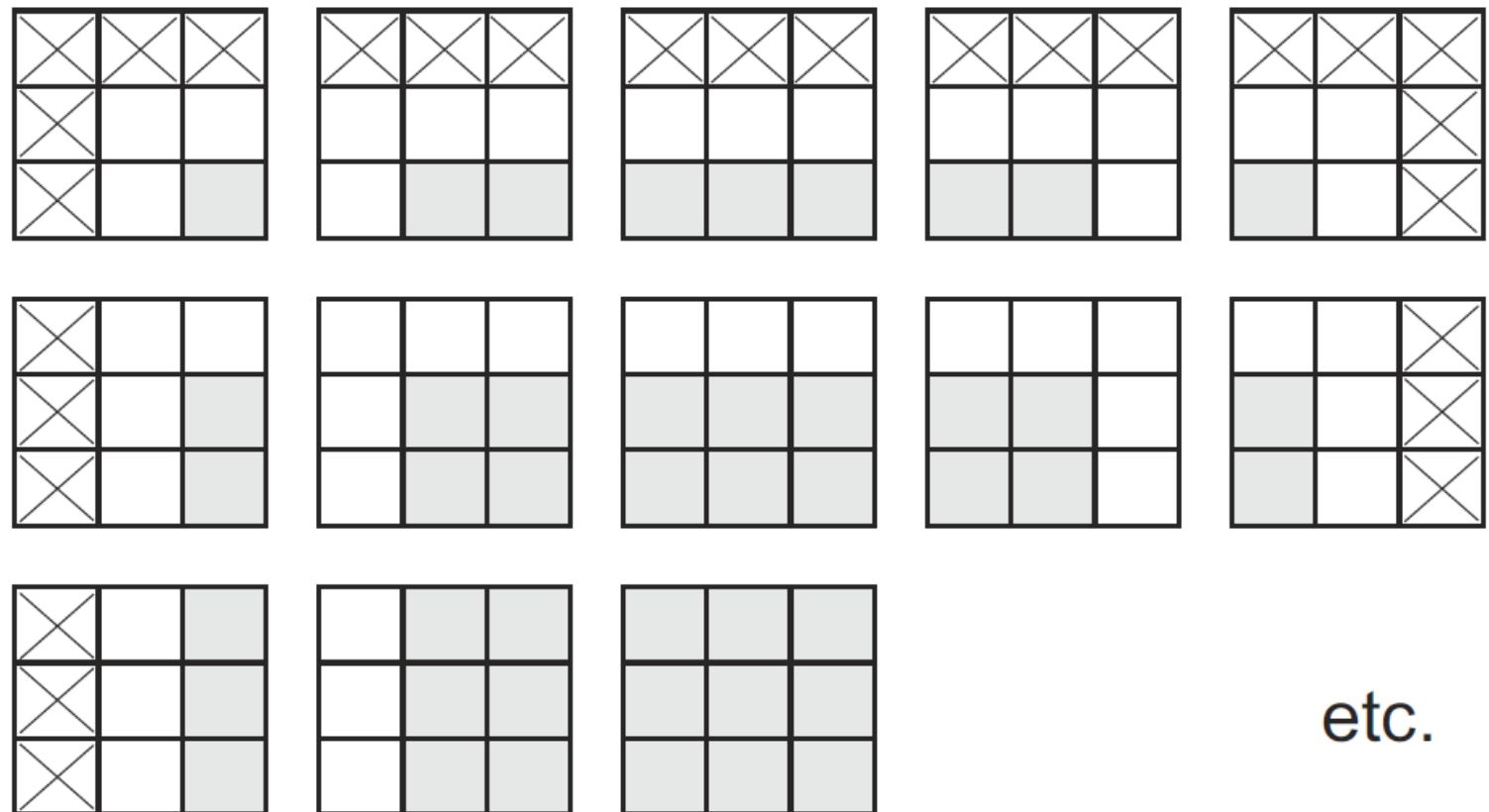
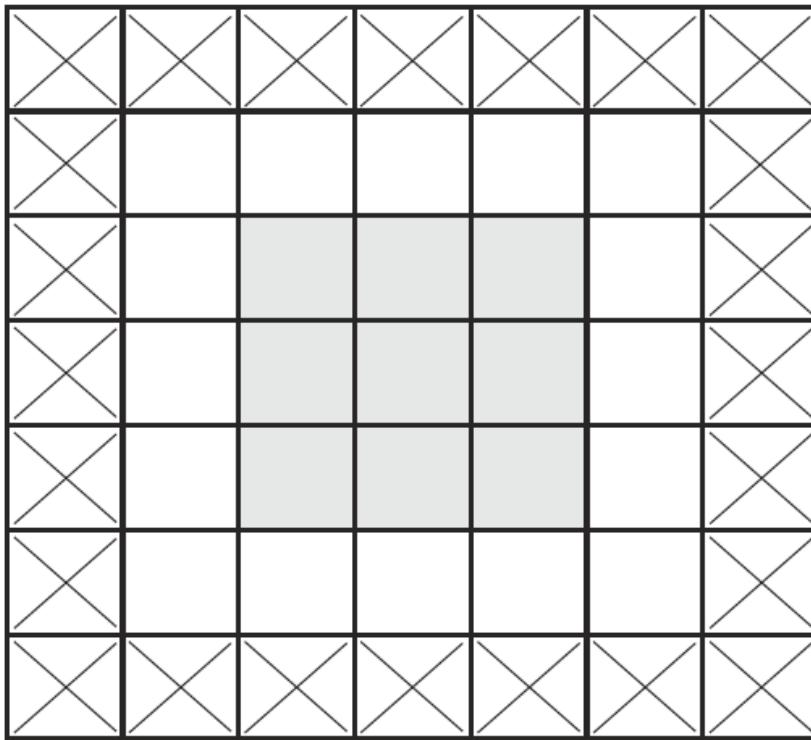
```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650
<hr/>		
Total params: 93,322 Trainable params: 93,322 Non-trainable params: 0		



Padding

- Padding a 5x5 input to extract 25 3x3 patches

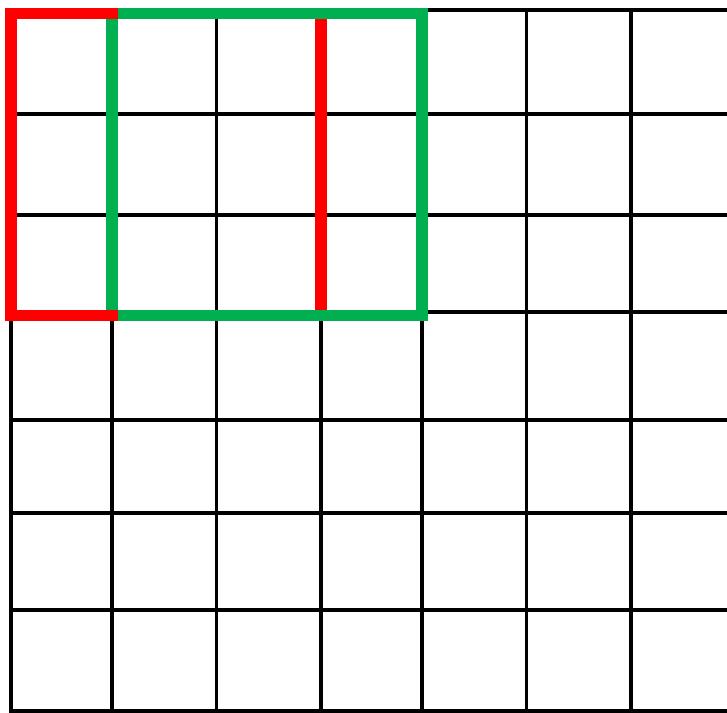


etc.

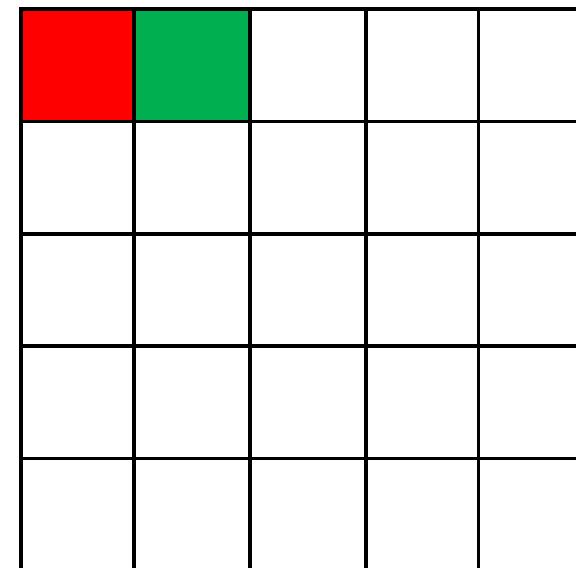


Stride=1

7 x 7 Input Volume

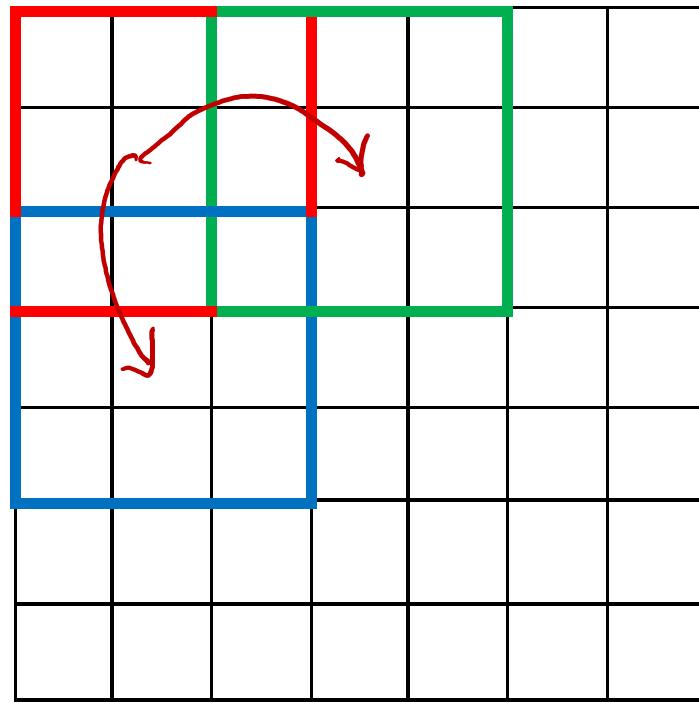


5 x 5 Output Volume

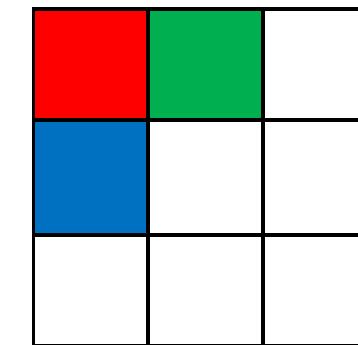


Stride=2

7 x 7 Input Volume

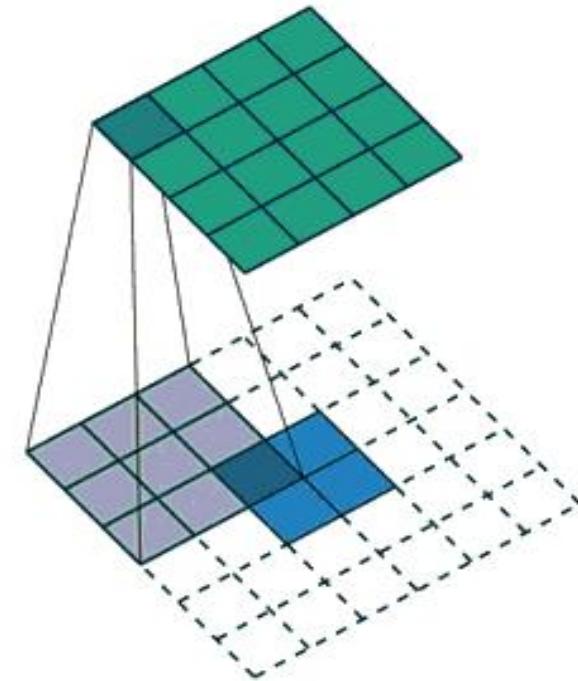
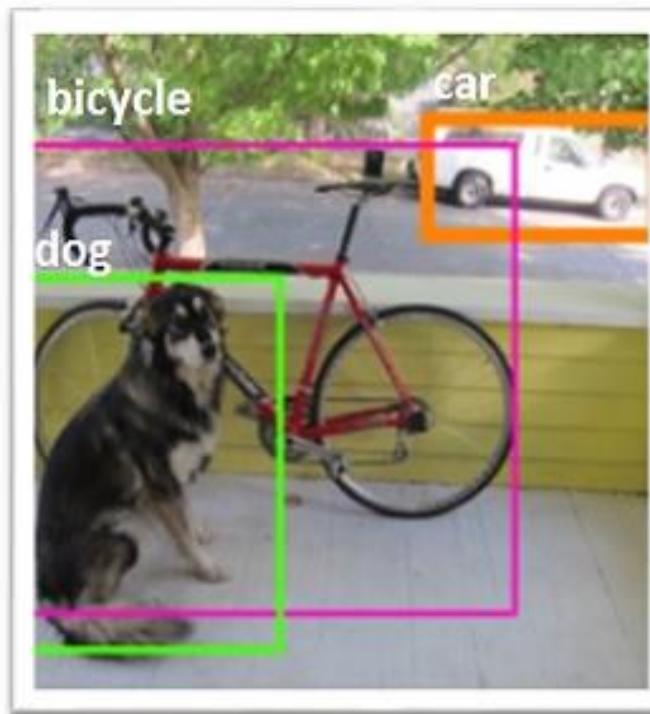
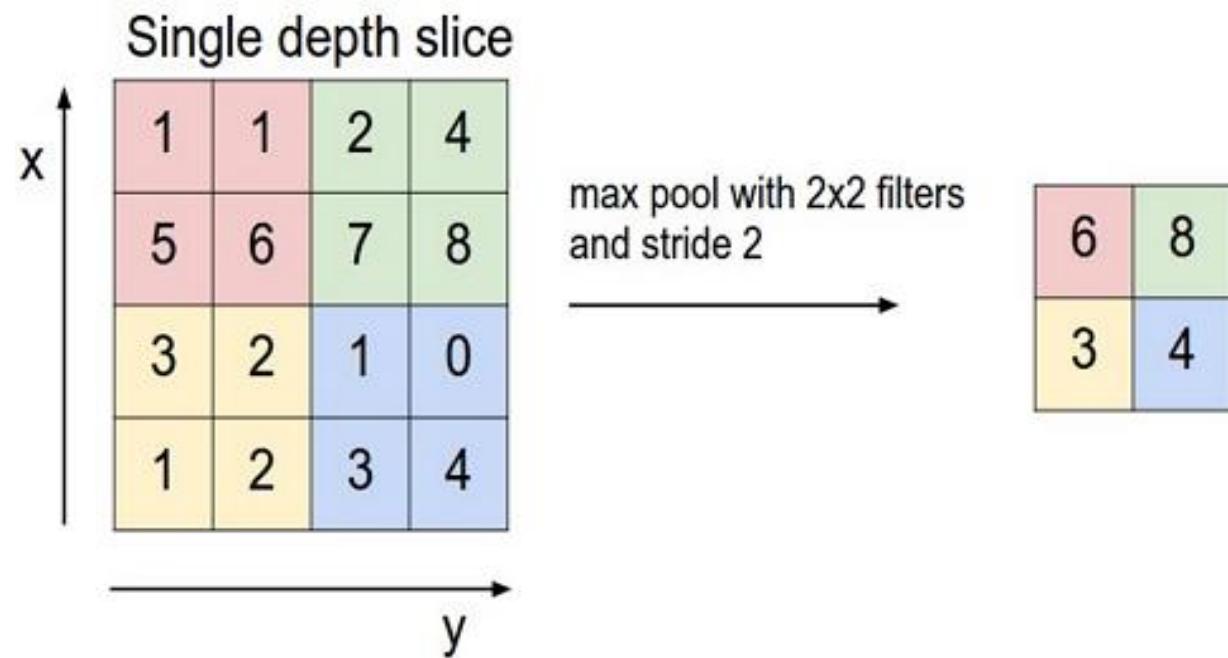


3 x 3 Output Volume



Max Pooling

- Downsampling an image
- Better than average pooling and strides



Train a Model to Classify Cats & Dogs

- www.kaggle.com/c/dogs-vs-cats/data
- 2000 cat and 2000 dog images



Create a CNN Model for Binary Classification

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



Image Generator

1. Read the picture files.
2. Decode the JPEG content to RGB grids of pixels.
3. Convert these into floating-point tensors.
4. Rescale the pixel values (between 0 and 255) to the [0, 1] interval

```
from keras.preprocessing.image import  
ImageDataGenerator  
train_datagen =  
ImageDataGenerator(rescale=1./255)  
test_datagen =  
ImageDataGenerator(rescale=1./255)  
train_generator =  
train_datagen.flow_from_directory(  
    train_dir,  
    target_size=(150, 150),  
    batch_size=20,  
    class_mode='binary')  
validation_generator =  
test_datagen.flow_from_directory(  
    validation_dir,  
    target_size=(150, 150),  
    batch_size=20,  
    class_mode='binary')
```



Python Generator

- Use *yield* operator
- Note that the generator loops endlessly

Here is an example of a generator that yields integers:

```
def generator():
    i = 0
    while True:
        i += 1
        yield i

for item in generator():
    print(item)
    if item > 4:
        break
```

It prints this:

```
1
2
3
4
5
```

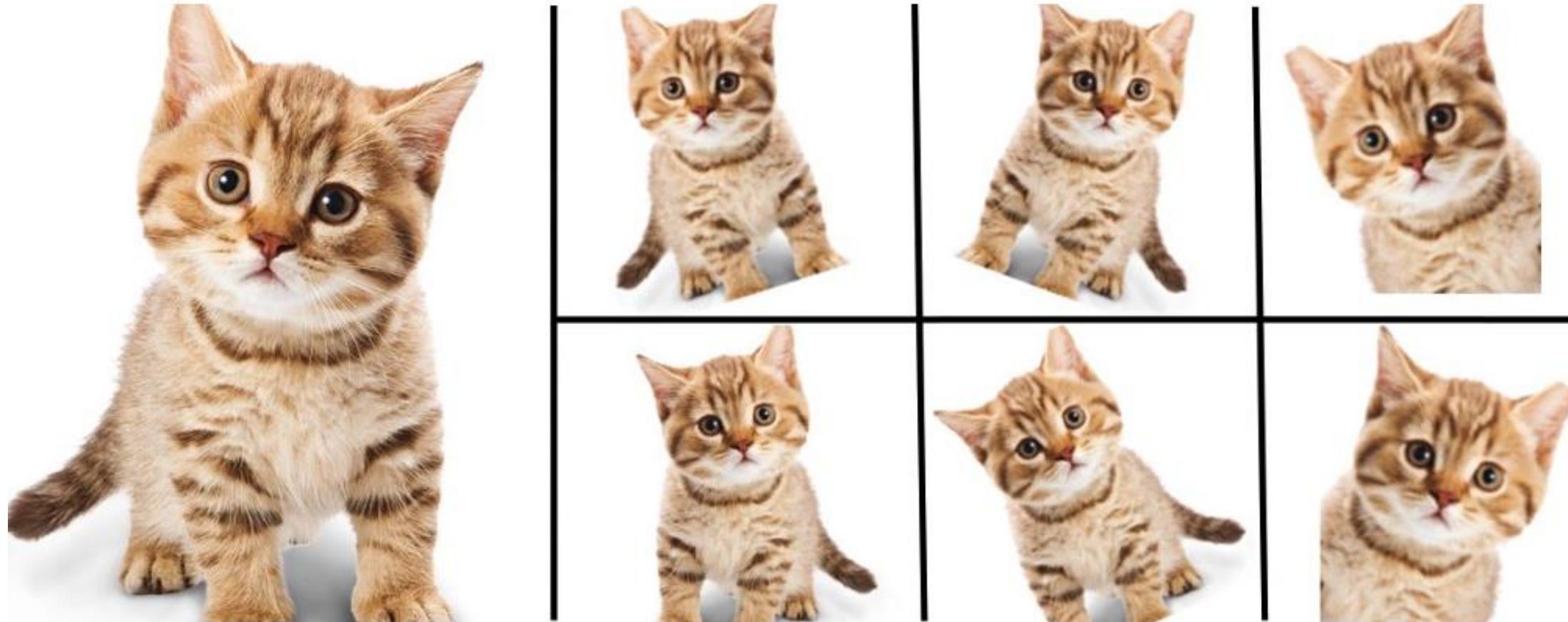


Fitting the Model using a Batch Generator

```
history = model.fit_generator(  
    train_generator,  
    steps_per_epoch=100,  
    epochs=30,  
    validation_data=validation_generator,  
    validation_steps=50)  
  
# Save the model  
model.save('cats_and_dogs_small_1.h5')
```



Data Augmentation



Data Augmentation via ImageDataGenerator

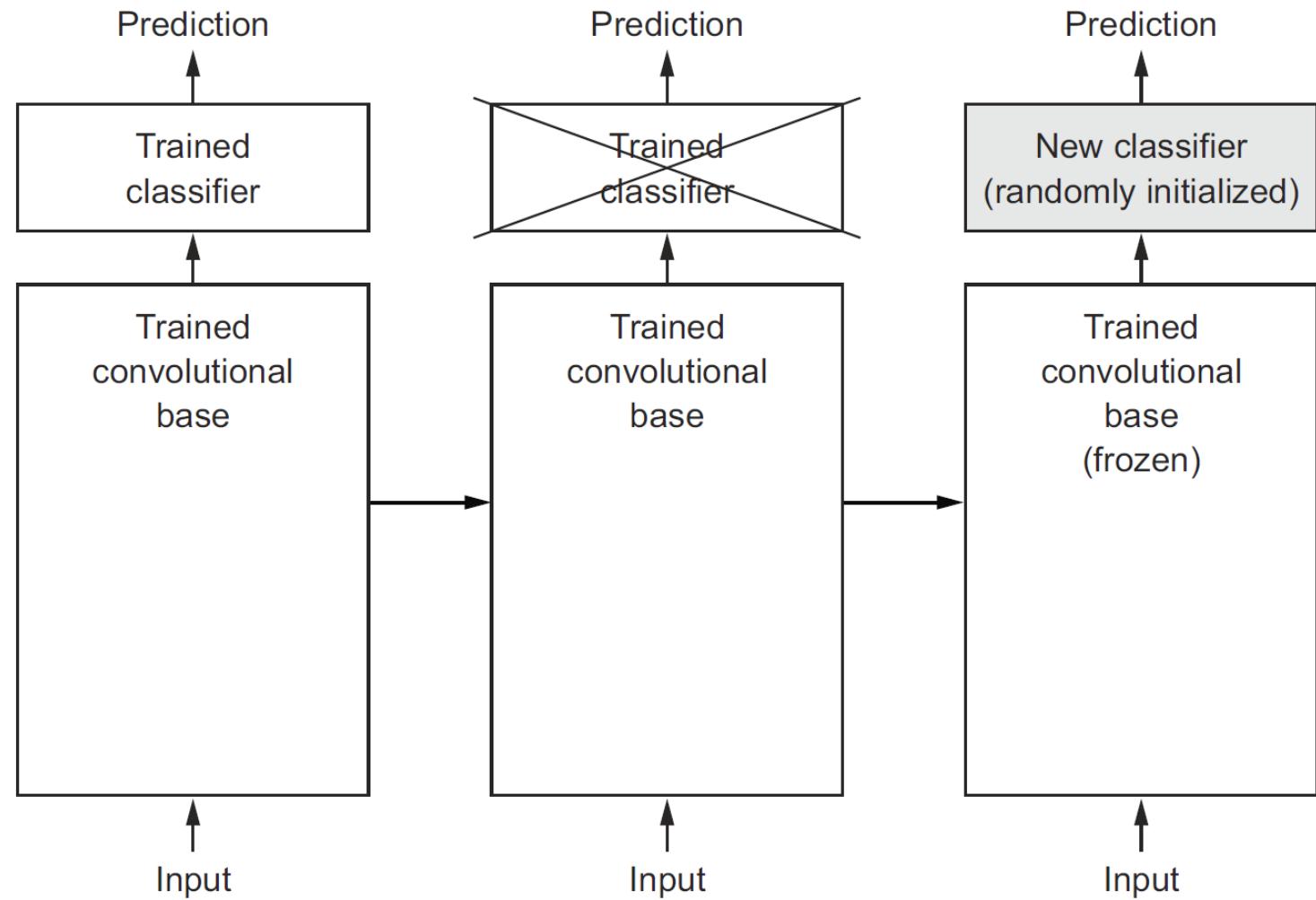
- `rotation_range` is a value in degrees (0–180)
- `width_shift` and `height_shift` are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
- `shear_range` is for randomly applying shearing transformations.
- `zoom_range` is for randomly zooming inside pictures.
- `horizontal_flip` is for randomly flipping half the images horizontally
- `fill_mode` is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

```
datagen = ImageDataGenerator(  
    rotation_range=40,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True,  
    fill_mode='nearest')
```



Using Pre-trained Models

- [Xception](#)
- [VGG16](#)
- [VGG19](#)
- [ResNet, ResNetV2, ResNeXt](#)
- [InceptionV3](#)
- [InceptionResNetV2](#)
- [MobileNet](#)
- [MobileNetV2](#)
- [DenseNet](#)
- [NASNet](#)



Example: Using Pre-trained VGG16

- `weights` specifies the weight checkpoint from which to initialize the model.
- `include_top` refers to including (or not) the densely connected classifier on top of the network (1,000 classes output).
- `input_shape` the network will be able to process inputs of any size if the argument is omitted.

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```



Adding a Classifier on Top of a Pre-trained Model

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Layer (type)	Output Shape	Param #
=====		
vgg16 (Model)	(None, 4, 4, 512)	14714688
=====		
flatten_1 (Flatten)	(None, 8192)	0
=====		
dense_1 (Dense)	(None, 256)	2097408
=====		
dense_2 (Dense)	(None, 1)	257
=====		
Total params:	16,812,353	
Trainable params:	16,812,353	
Non-trainable params:	0	



Freezing Trainable Parameters

- `conv_base.trainable = False`

```
[4] print('This is the number of trainable weights '
      'before freezing the conv base:', len(model.trainable_weights))
```

↳ This is the number of trainable weights before freezing the conv base: 30

```
[5] conv_base.trainable = False
```

```
▶ print('This is the number of trainable weights '
      'after freezing the conv base:', len(model.trainable_weights))
```

↳ This is the number of trainable weights after freezing the conv base: 4



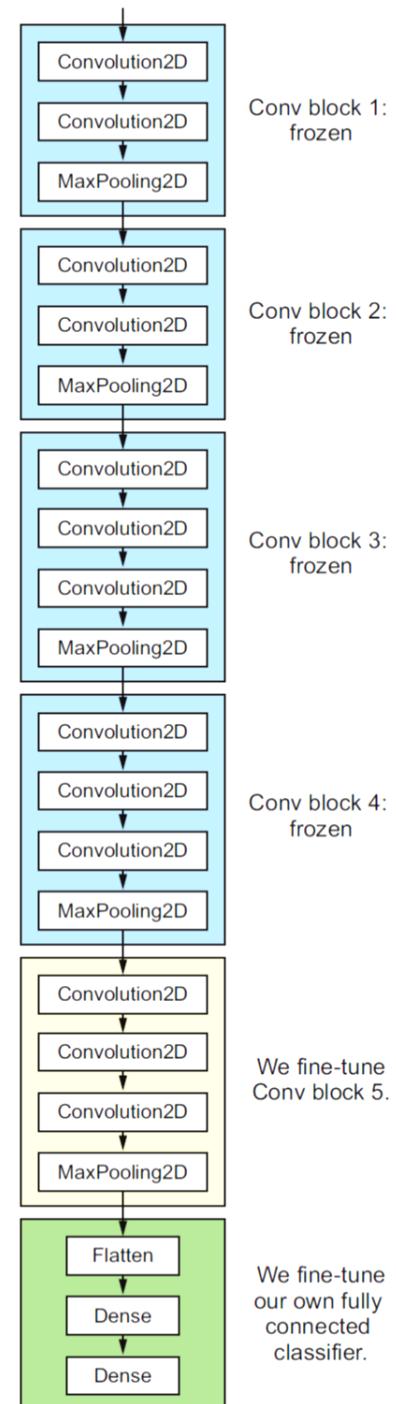
Fine-Tuning Top Few Layers

- Freezing all layers up to a specific one

```
conv_base.trainable = True
set_trainable = False

for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True

    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```



Summary

- Convnets are the best for Computer Vision (and maybe all the other tasks)
- Data augmentation is a powerful way to fight overfitting
- We can use pre-trained model for feature extraction
- We can further improve the pre-trained model on our dataset by fine-tuning



Visualizing What Convnets Learn

1. *Visualizing Intermediate ConvNet Outputs (Intermediate Activations)*
 - Understand how successive convnet layers transform their input
 - Get a first idea of the meaning of individual convnet filters
2. *Visualizing ConvNets Filters*
 - Understand precisely what visual pattern or concept each filter in a convnet is receptive to
3. *Visualizing Heatmaps of Class Activation in an Image*
 - See which parts of an image were identified as belonging to a given class
 - Can localize objects in images.



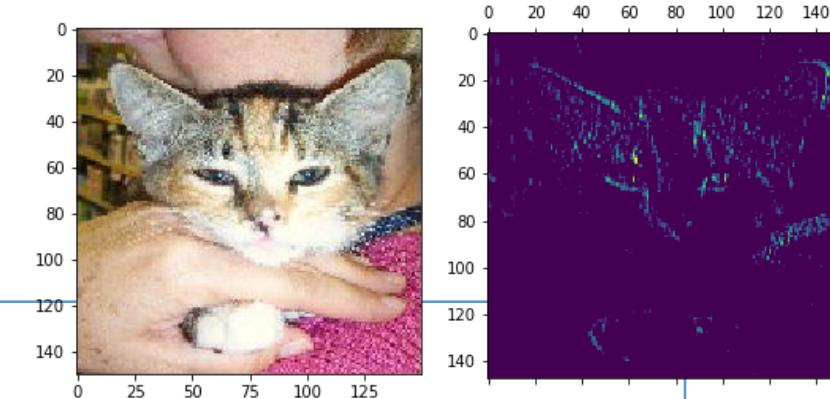
1. Visualizing Intermediate Activations

- Show the feature maps that are output by various convolution and pooling layers in a network

```
from keras.preprocessing import image
import numpy as np
img = image.load_img('./test1/1700.jpg', target_size=(150, 150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)/255.

from keras import models
model = load_model('cats_and_dogs_small_1.h5')
layer_outputs = [layer.output for layer in model.layers[:8]]
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
activations = activation_model.predict(img_tensor)
first_layer_activation = activations[0]

import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0, :, :, 3], cmap='viridis')
```



Visualizing Every Channel in Every Intermediate Activation

```
layer_names = []
for layer in model.layers[:8]:
    layer_names.append(layer.name)

images_per_row = 16

for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1]
    size = layer_activation.shape[1]
    n_cols = n_features // images_per_row
    display_grid = np.zeros((size * n_cols, images_per_row * size))

    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                              :, :,
                                              col * images_per_row + row]

            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] = channel_image

    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                      scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

Names of the layers, so you can have them as part of your plot

Displays the feature maps

The feature map has shape (l, size, size, n_features).

Number of features in the feature map

Tiles each filter into a big horizontal grid

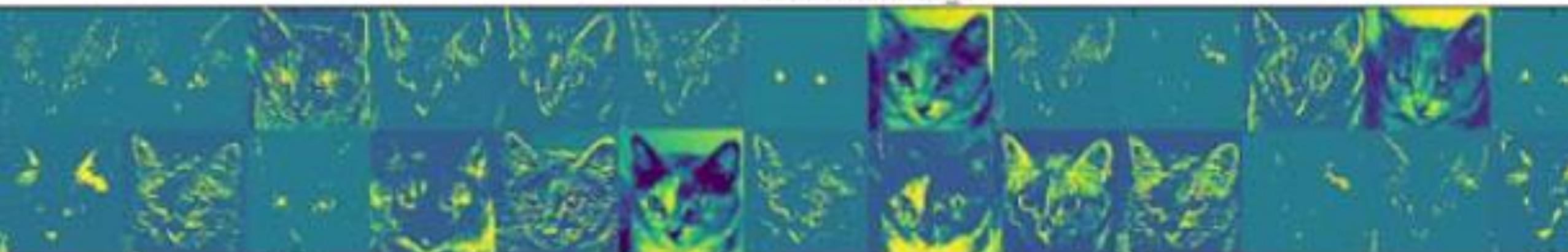
Tiles the activation channels in this matrix

Post-processes the feature to make it visually palatable

Displays the grid



convolution2d_5



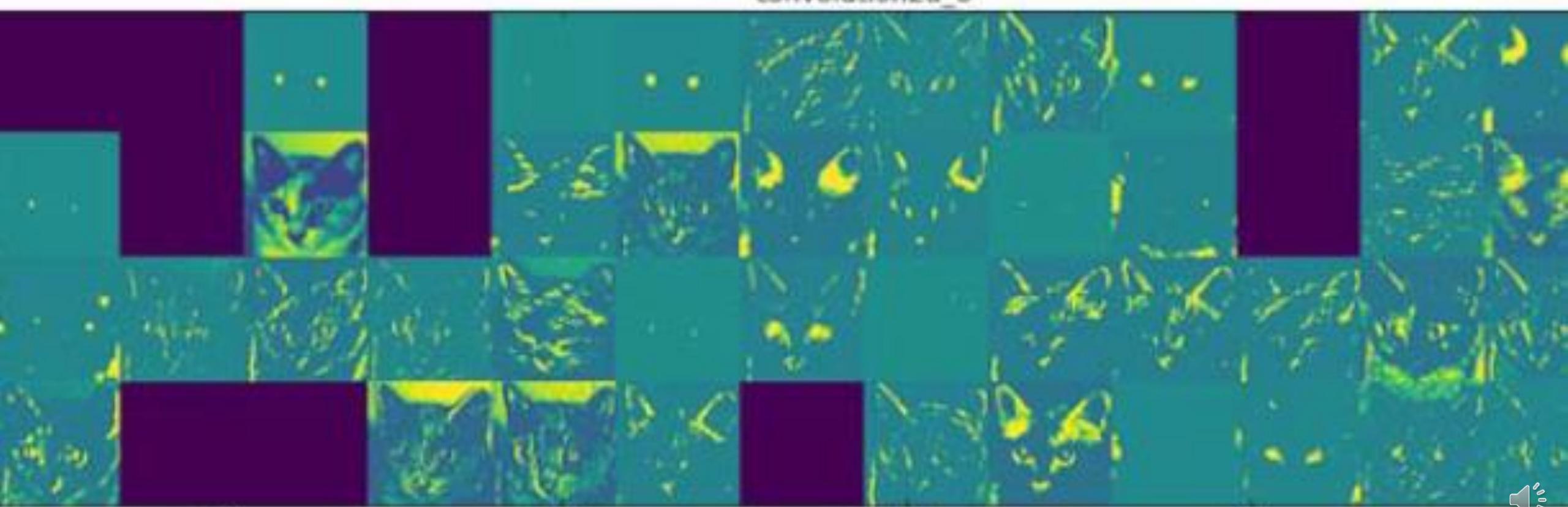
500

1000

1500

2000

convolution2d_6



200

400

600

800



Things to Note

- The first layer acts as a collection of various edge detectors
- As you go deeper, the activations become increasingly abstract and less visually interpretable
- The sparsity of the activations increases with the depth of the layer, more and more filters are blank



2. Visualizing ConvNet Filters

- Gradient ascent: applying *gradient descent* to the value of the input image of a convnet so as to *maximize* the response of a specific filter

Loss Maximization Via Stochastic Gradient Descent

Starts from a gray image with some noise

```
→ input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.
```

```
step = 1. ← Magnitude of each gradient update
```

```
for i in range(40):
```

```
→ loss_value, grads_value = iterate([input_img_data])
```

```
input_img_data += grads_value * step ←
```

Computes the loss value and gradient value

Adjusts the input image in the direction that maximizes the loss

Runs gradient ascent for 40 steps



Convert a Tensor into a Valid Image

```
def deprocess_image(x):
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

    x += 0.5
    x = np.clip(x, 0, 1)

    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

**Normalizes the tensor:
centers on 0, ensures
that std is 0.1**

Clips to [0, 1]

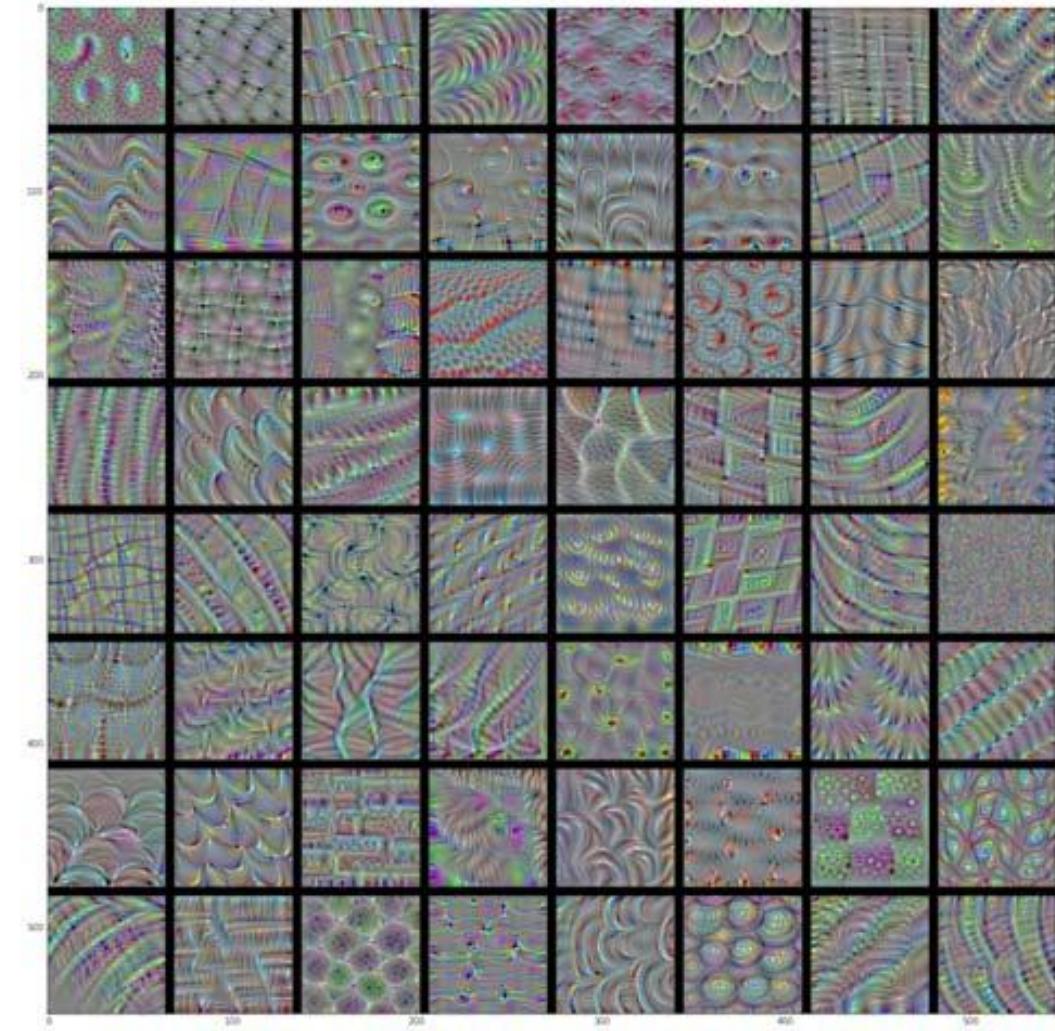
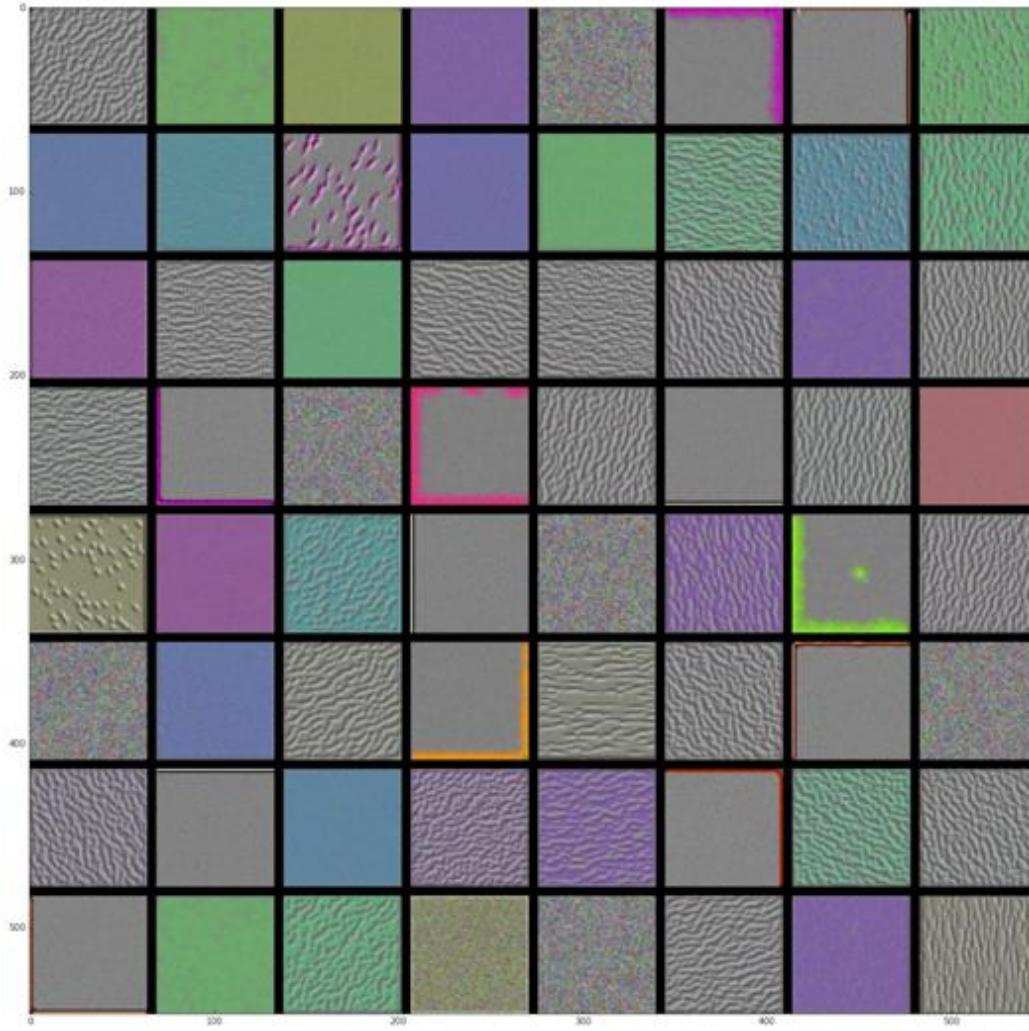
Converts to an RGB array



```
model = VGG16(weights='imagenet', include_top=False)
layer_name = 'block3_conv1'
filter_index = 0
def generate_pattern(layer_name, filter_index, size=150):
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])
    grads = K.gradients(loss, model.input)[0] # Keep only the first tensor
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5) # 1e-5 avoids divided by zero
    # Fetching Numpy output values given Numpy input values
    iterate = K.function([model.input], [loss, grads])
    loss_value, grads_value = iterate([np.zeros((1, size, size, 3))])
    # Loss maximization via stochastic gradient descent
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.
    step = 1.
    for i in range(40):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step
    img = input_img_data[0]
    return deprocess_image(img)
```



Filter Patterns for Each Layer



3. Visualizing Heatmaps of Class Activation

- Ramprasaath R. Selvaraju et al., “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization.” arXiv (2017), <https://arxiv.org/abs/1610.02391>.

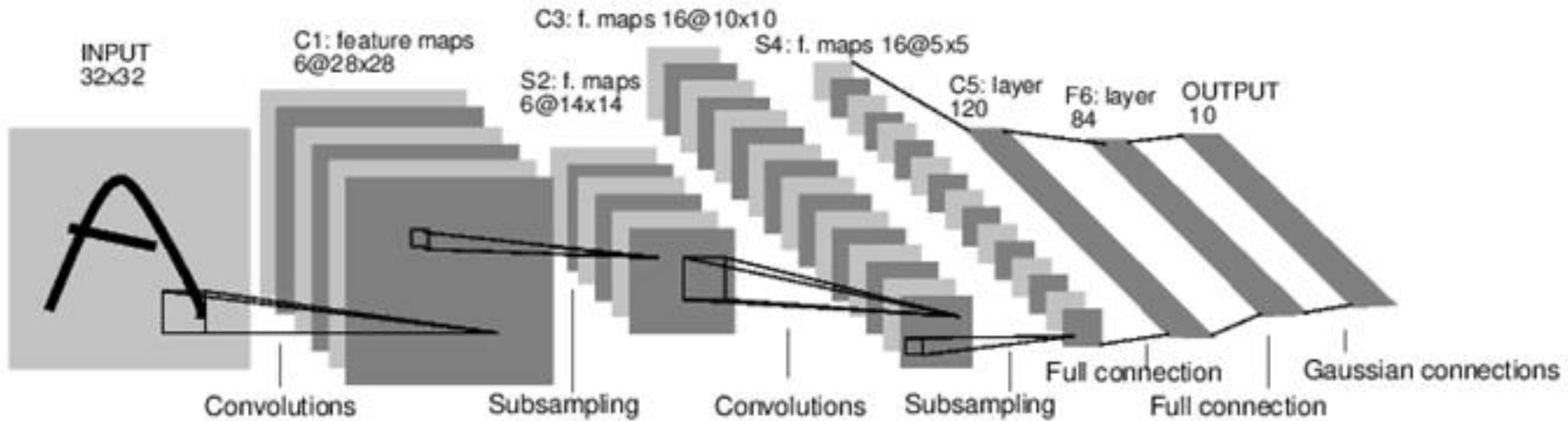


Evolution of CNN



Convolutional Neural Network (LeNet-5)

- LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.



A Full Convolutional Neural Network (LeNet)





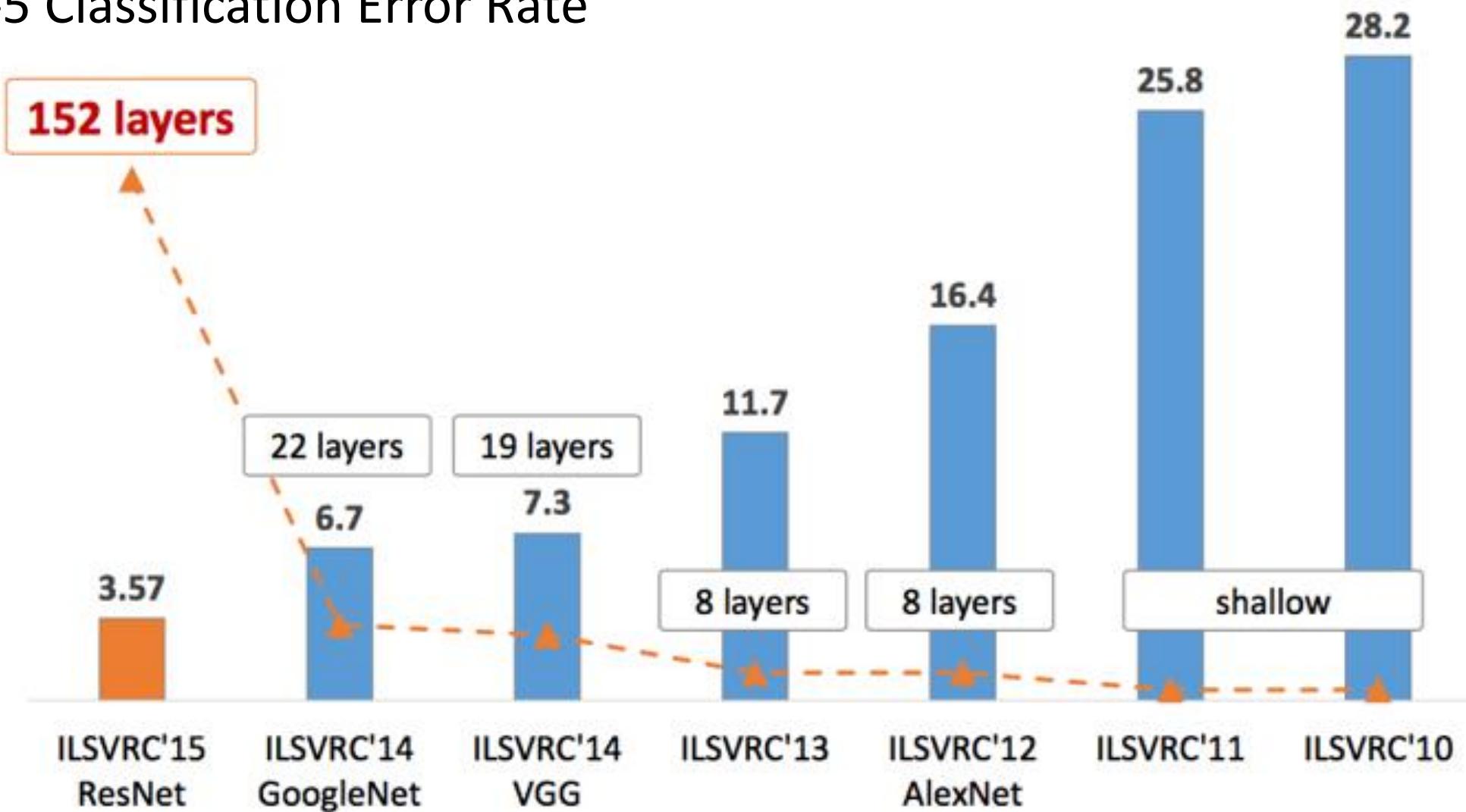
ImageNet Large Scale Visual Object Recognition Challenge (ILSVRC)

- 1000 categories
- For ILSVRC 2017
 - Training **images** for each category ranges from 732 to 1300
 - 50,000 validation **images** and 100,000 test **images**.
- Total number of images in ILSVRC 2017 is around 1,150,000



Error Rate on ImageNet Challenge

- Top-5 Classification Error Rate



I WAS WINNING
IMAGENET

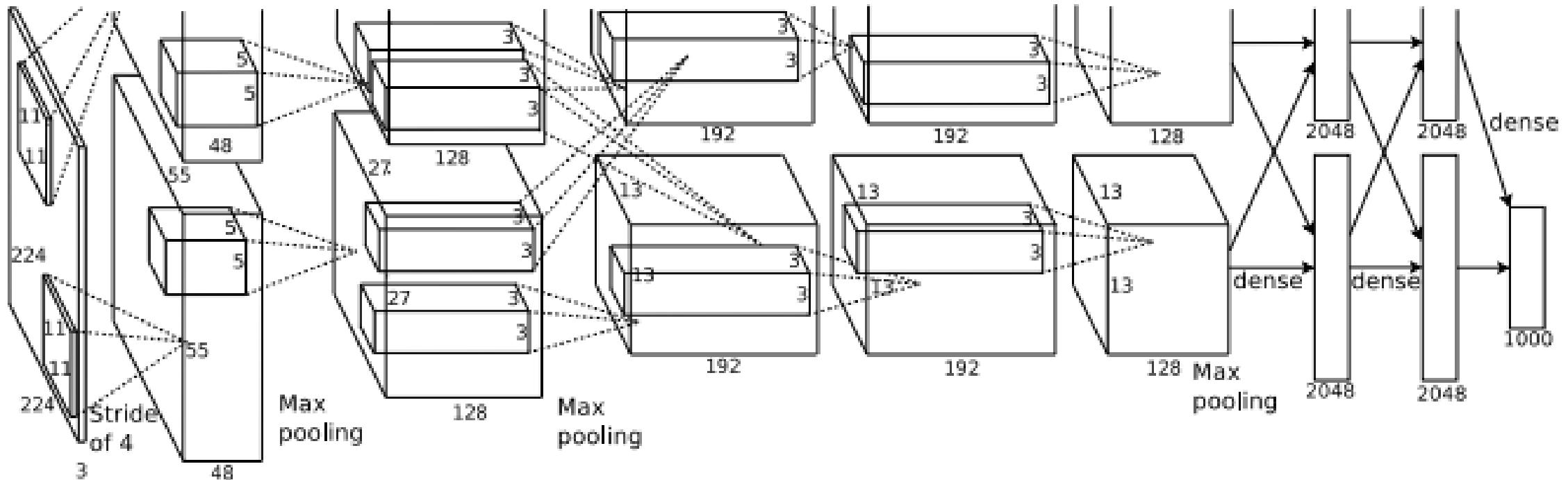


UNTIL A
DEEPER MODEL
CAME ALONG



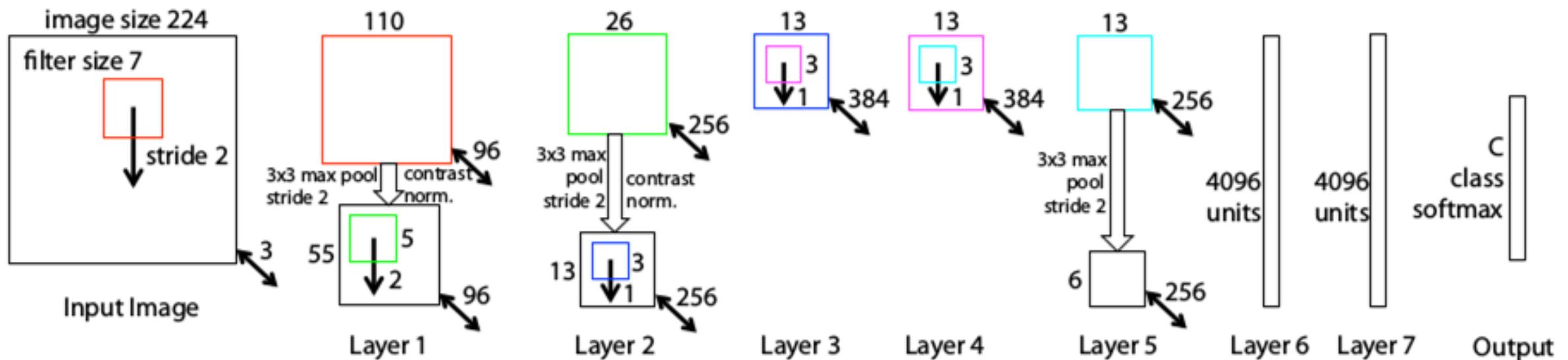
AlexNet (2012)

- AlexNet significantly outperformed previous models (e.g. SVM)
- Include convolutions, max-pooling, dropout, ReLU, SGD with momentum
- Use 2 Nvidia GeForce GTX 580 GPU



ZF Net (2013)

- Parameter tuning of AlexNet

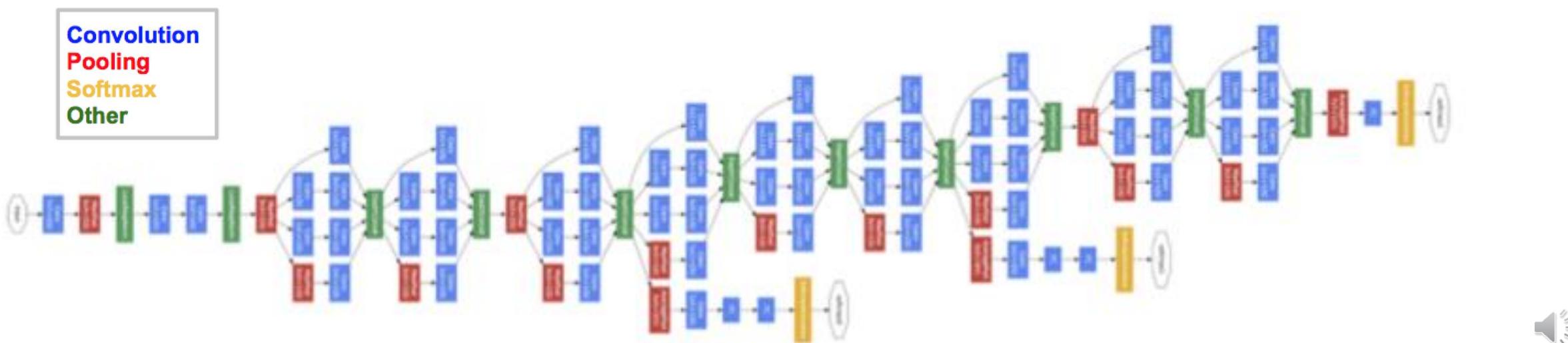


ZF Net Architecture

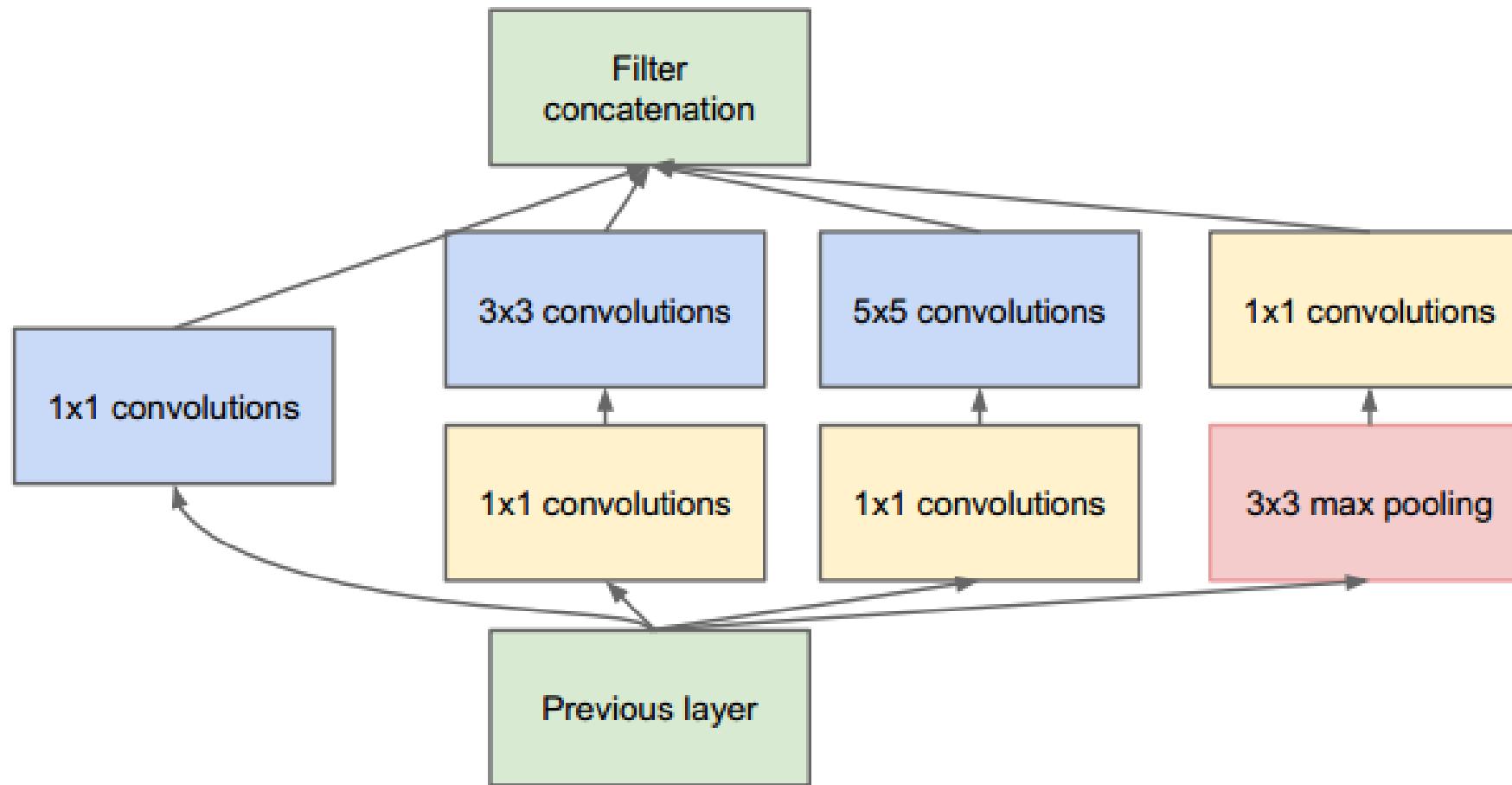


GoogLeNet (2014)

- Achieved a top-5 error rate of 6.67%! This was very close to human level performance
- Propose inception module, batch normalization, image distortions, and RMSprop
- 22 layers but reduced parameters from 60 million (AlexNet) to 4 million



Inception Module



VGG Net (2014)

- Very uniform architecture
- Preferred choice in the community for extracting features from images

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	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	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 conv1-256	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 conv1-512	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 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The 6 different architectures of VGG Net. Configuration D produced the best results



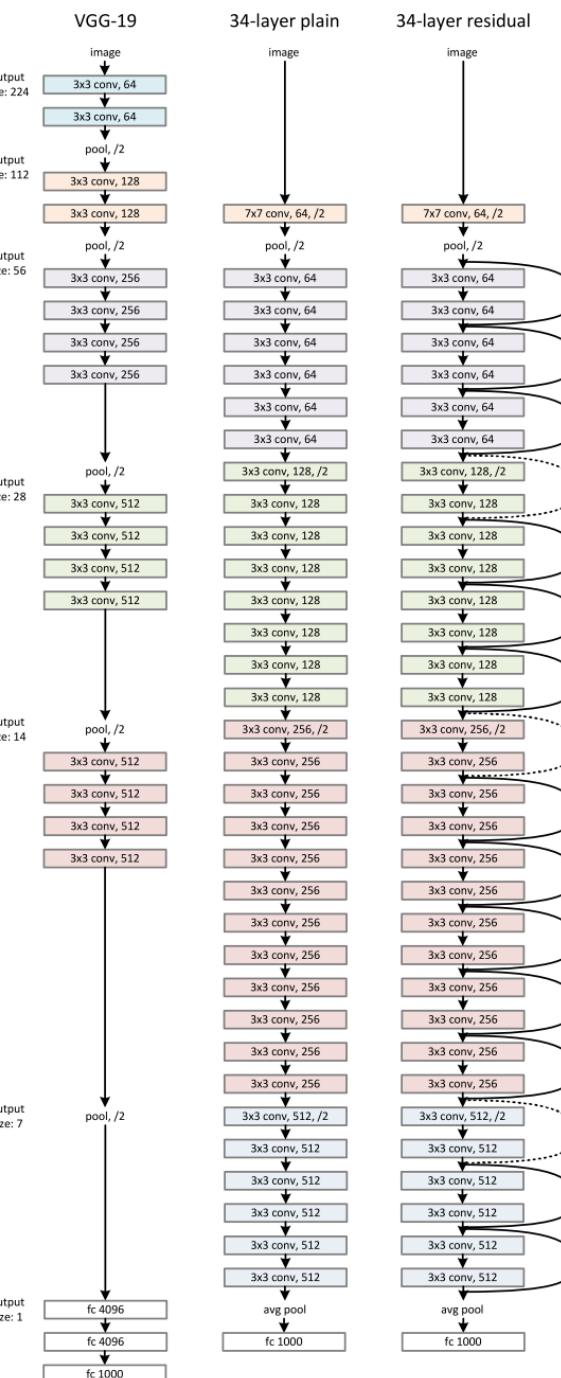
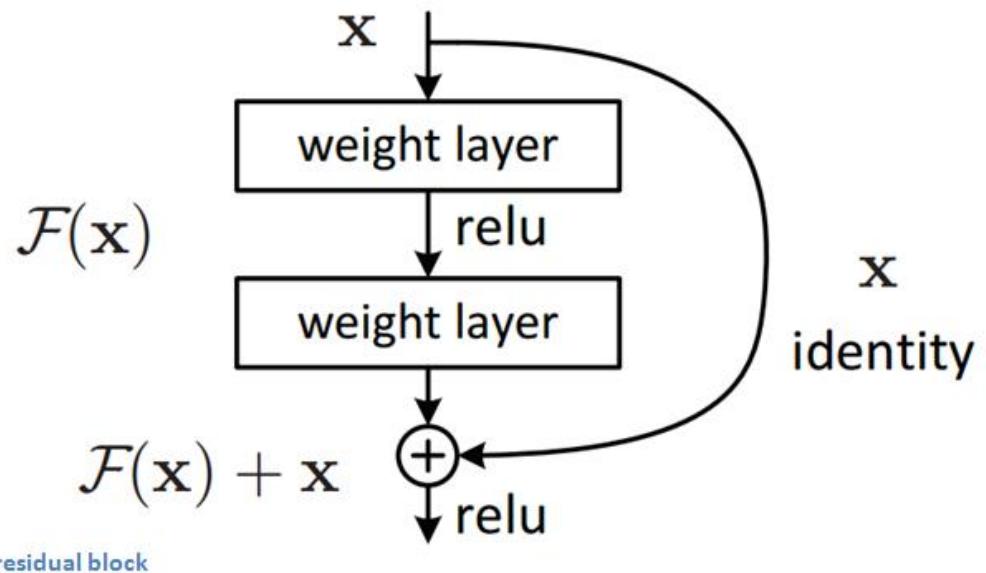
A close-up shot from the movie Inception. Two men in dark suits are seated at a table, looking down at something on it. The man on the left has his eyes closed and a slight smile. The man on the right has his eyes open and a neutral expression. The lighting is warm and focused on their faces.

WE NEED TO GO

DEEPER

ResNet (2015)

- Residual Neural Network
- Proposed “skip connection”
- 152-layer with 3.57% error rate

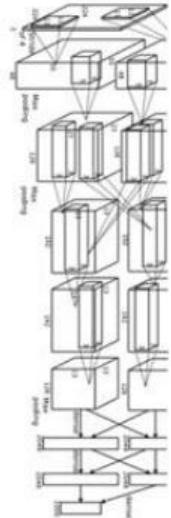


Visualizing CNN Side-by-Side

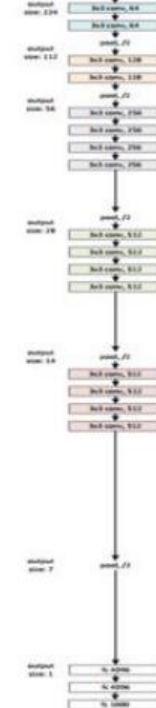
LeNet
(1998)
7 Layers



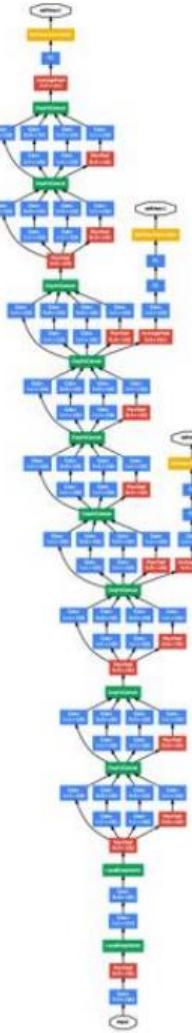
AlexNet
(2012)
8 Layers



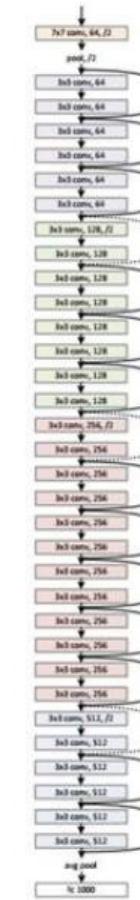
VGGNet
(2014)
19 Layers



GoogLeNet
(2014)
22 Layers



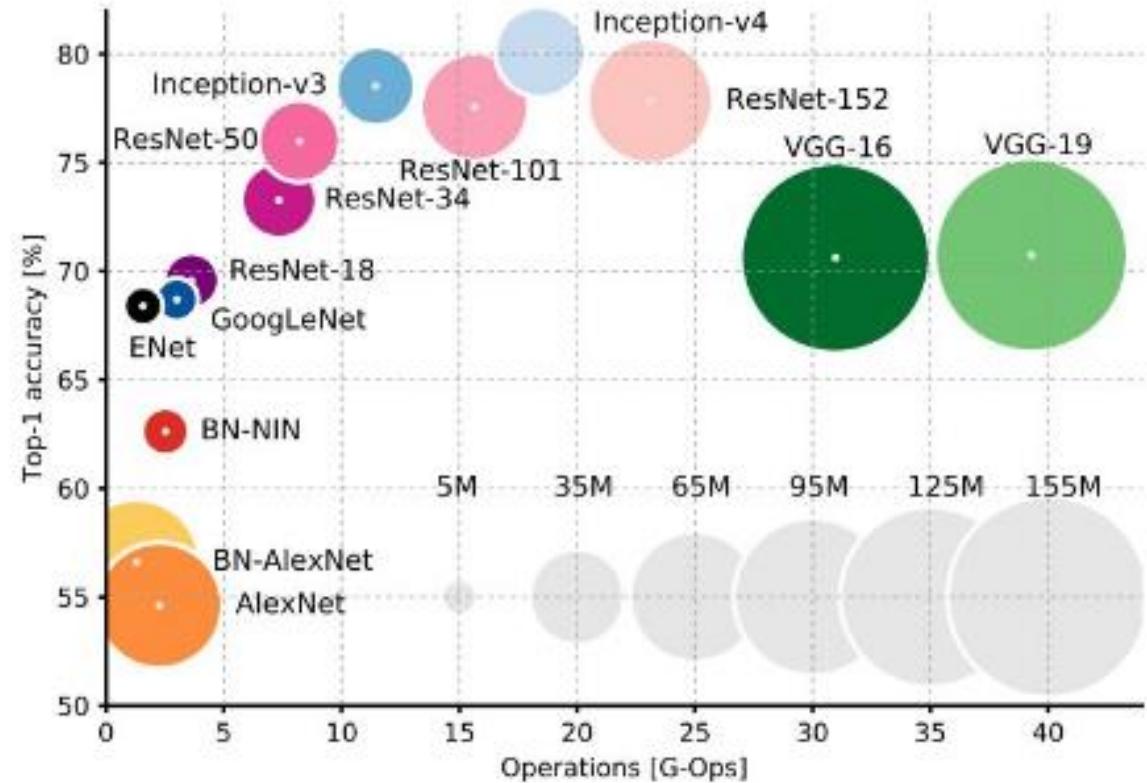
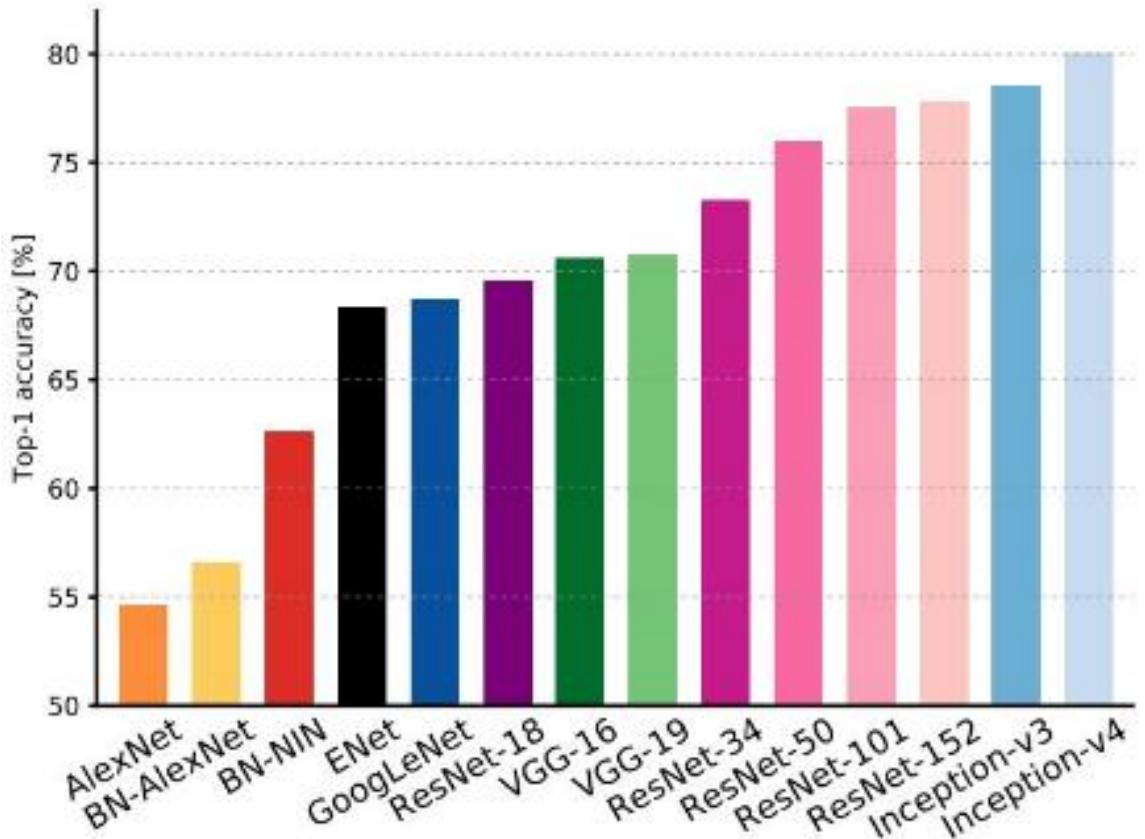
ResNet
(2015)
152 Layers



(34-layer version)



Statistics



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

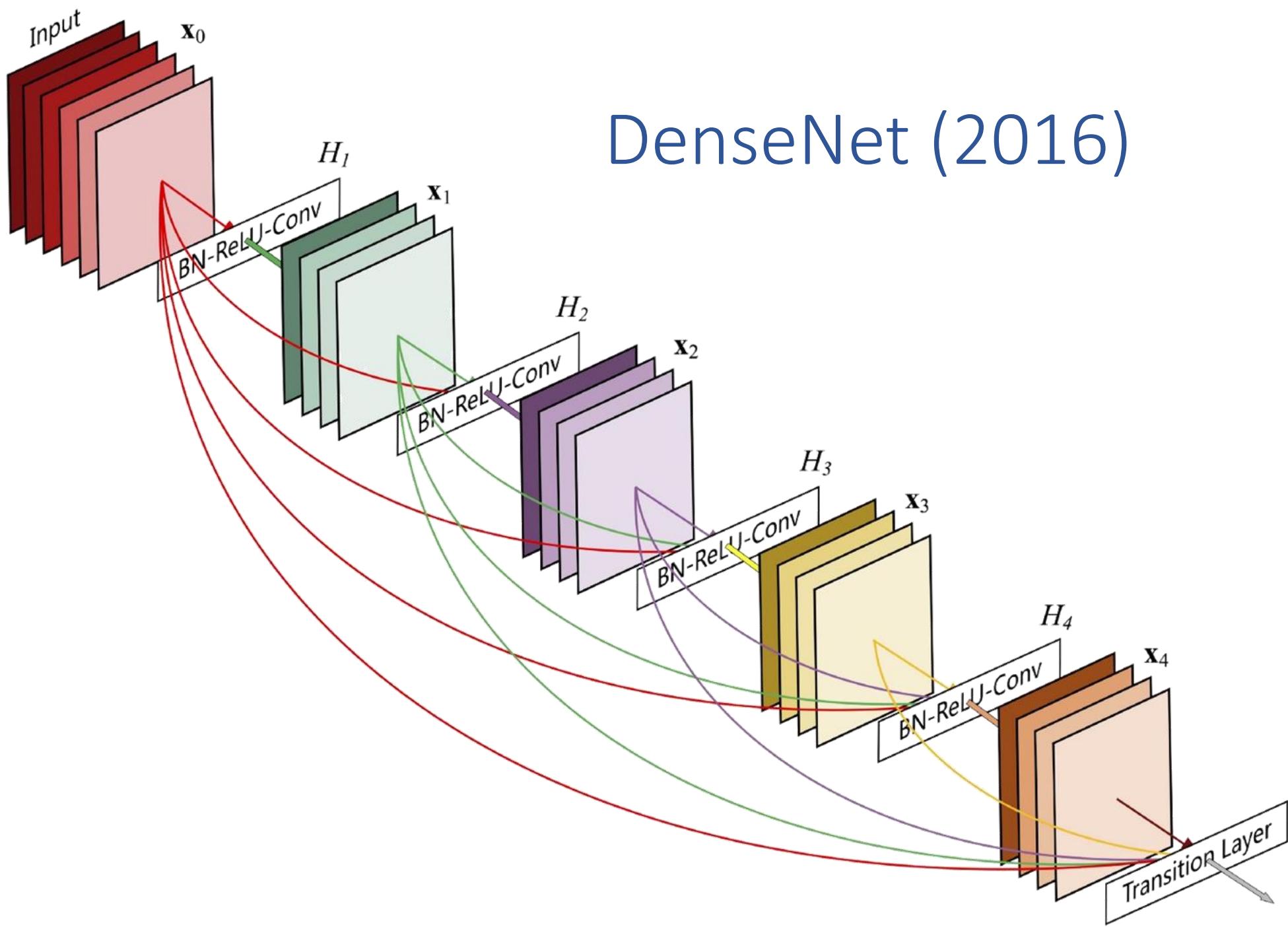


Summary Table

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

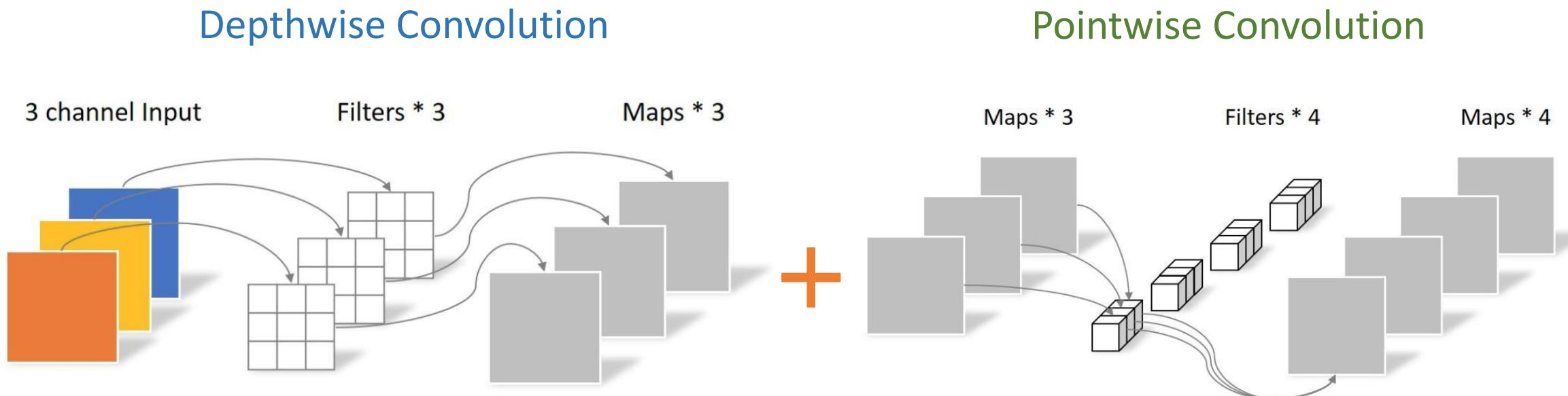


DenseNet (2016)



Xception - Separable Convolution (2017)

- Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” *CVPR*, 2017
- Example

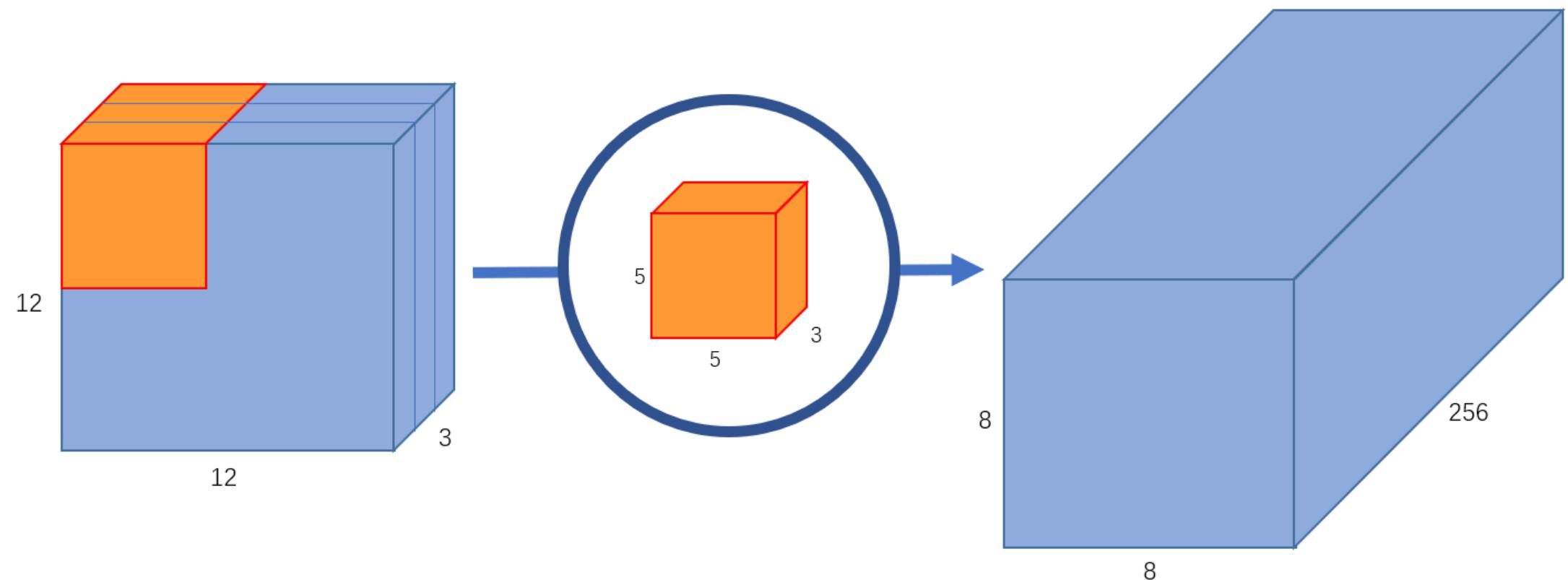


<https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728>



Normal Convolution vs. Depthwise Separable (1)

- Normal Convolution



<https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728>



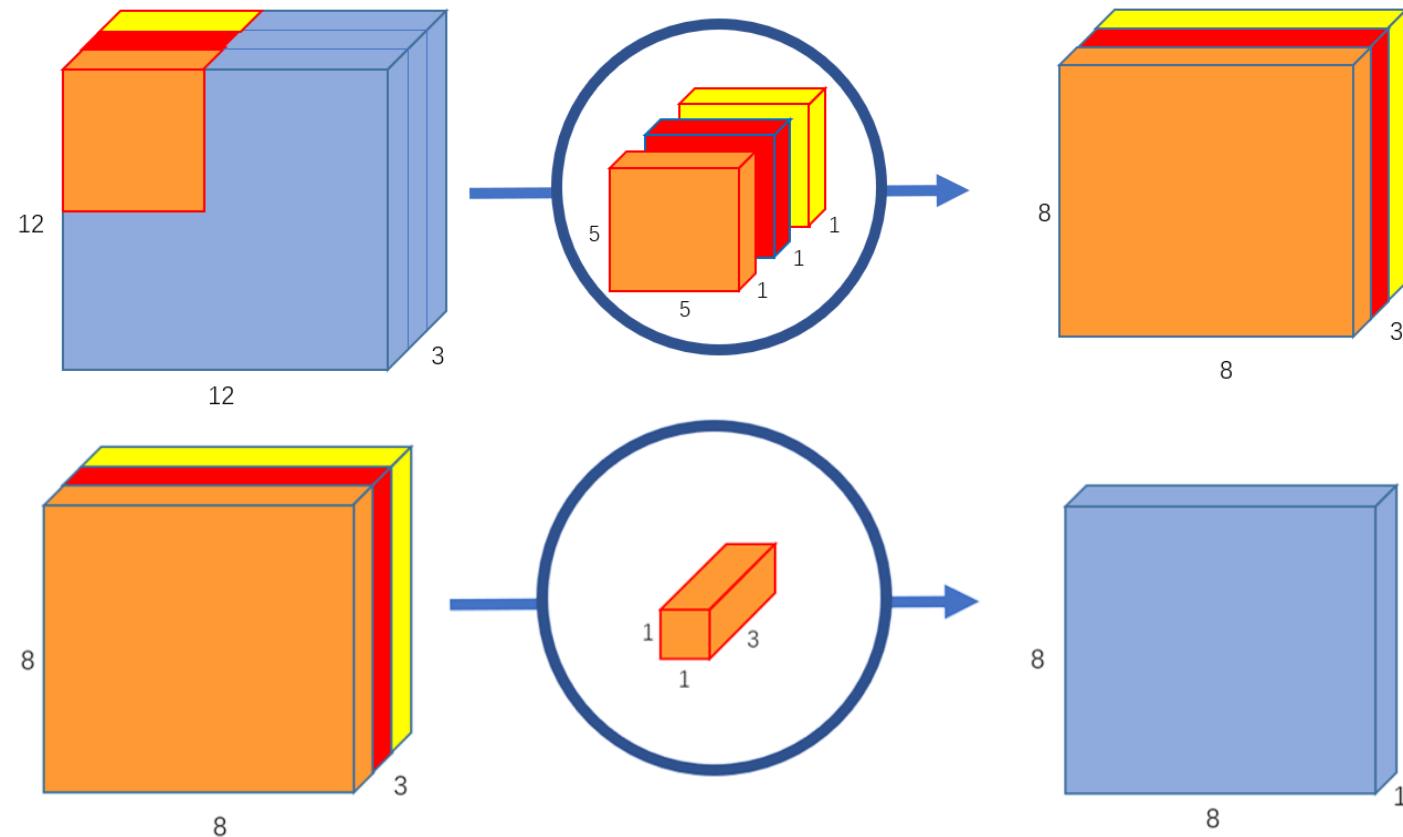
Normal Convolution vs. Depthwise Separable (2)

- Two steps: depthwise + pointwise

Depthwise

+

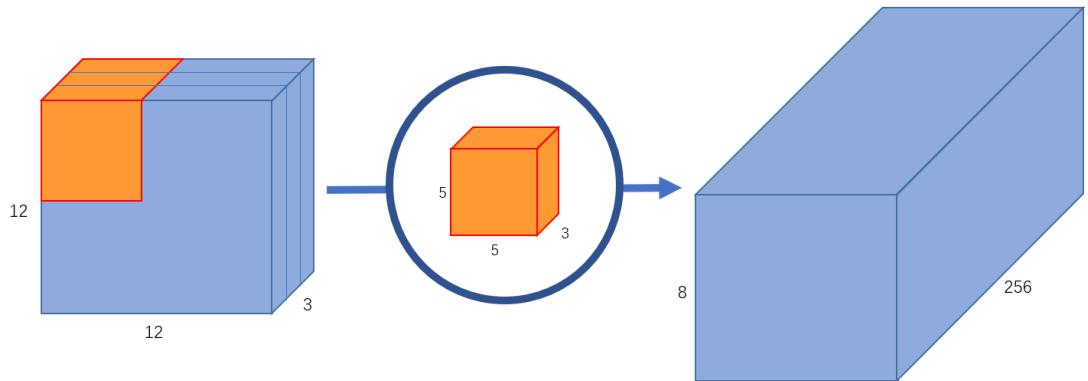
Pointwise



Normal Convolution vs. Depthwise Separable (2)

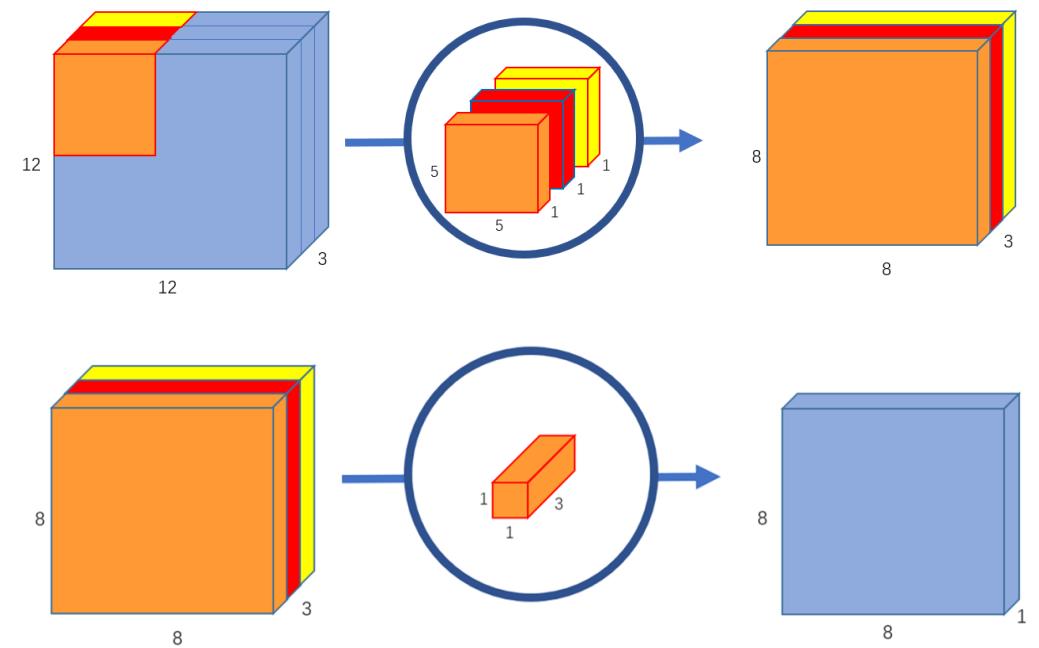
- Normal Filter size:

$$5*5*3*256 \\ =19200$$



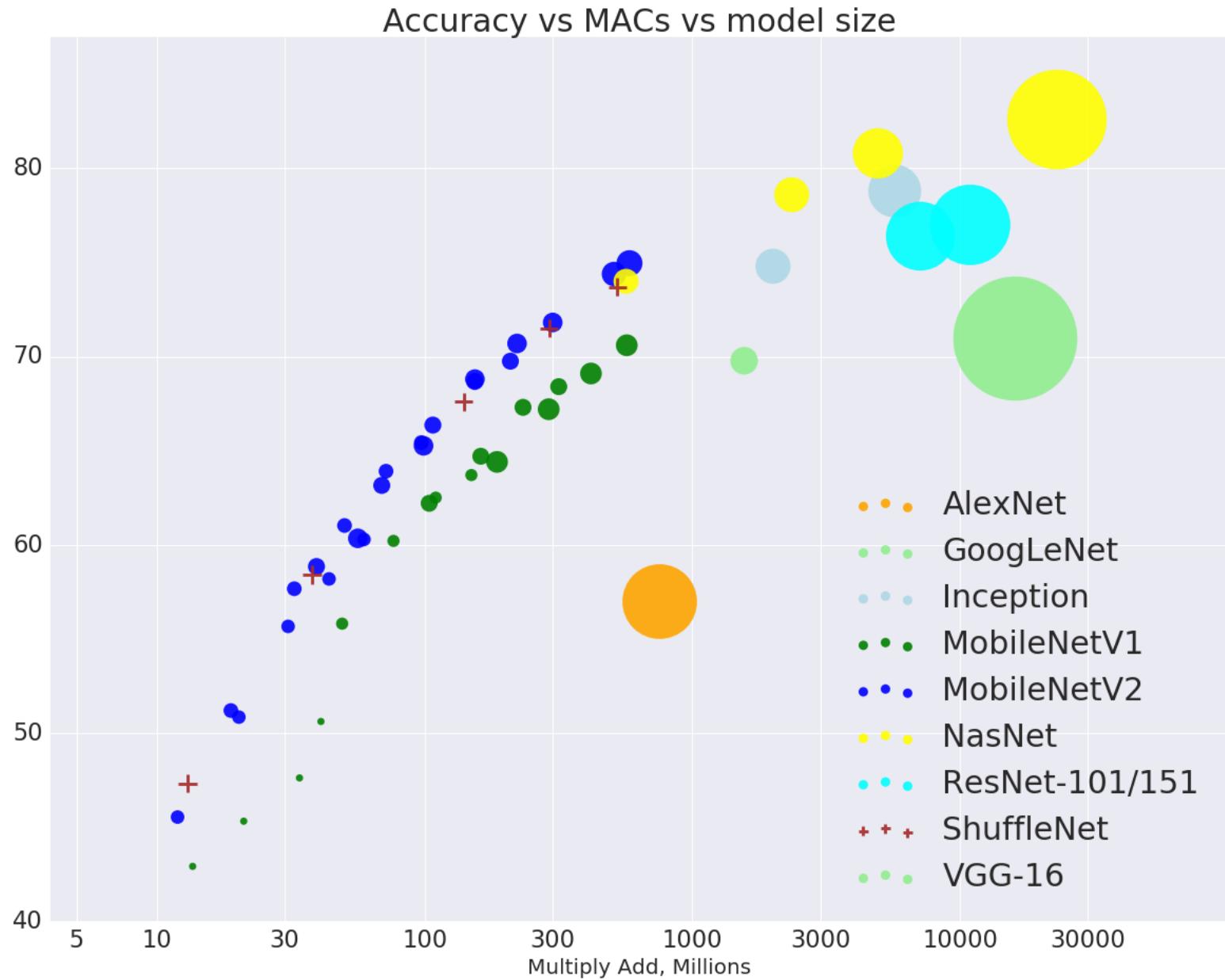
- Depthwise Filter size:

$$(5*5*1)*3 + 1*1*3*256 \\ =843$$



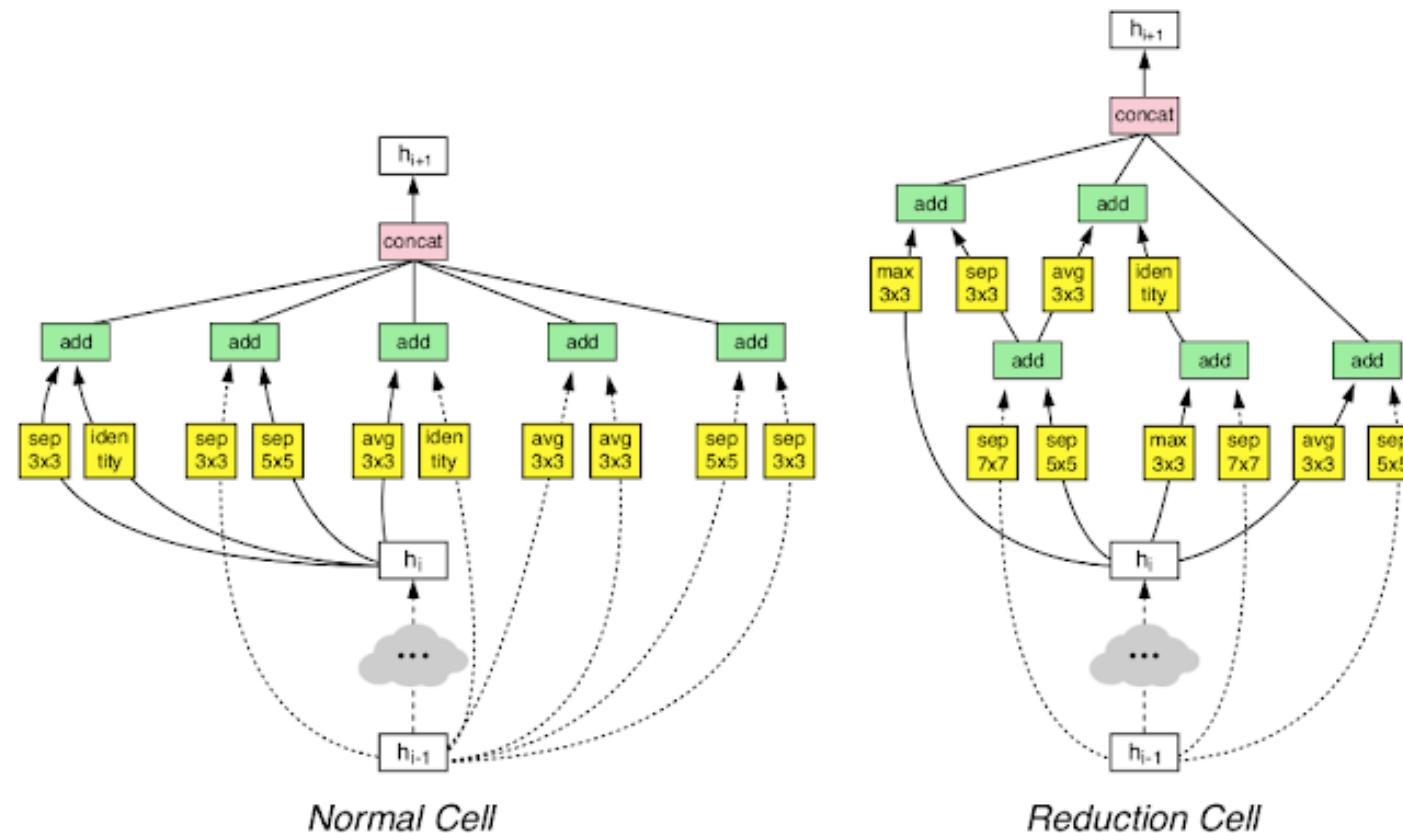
Embedded Neural Networks

- Pruning and Quantization
- Howards et al.,
“MobileNets: Efficient
Convolutional Neural
Networks for Mobile
Vision Applications”,
2017

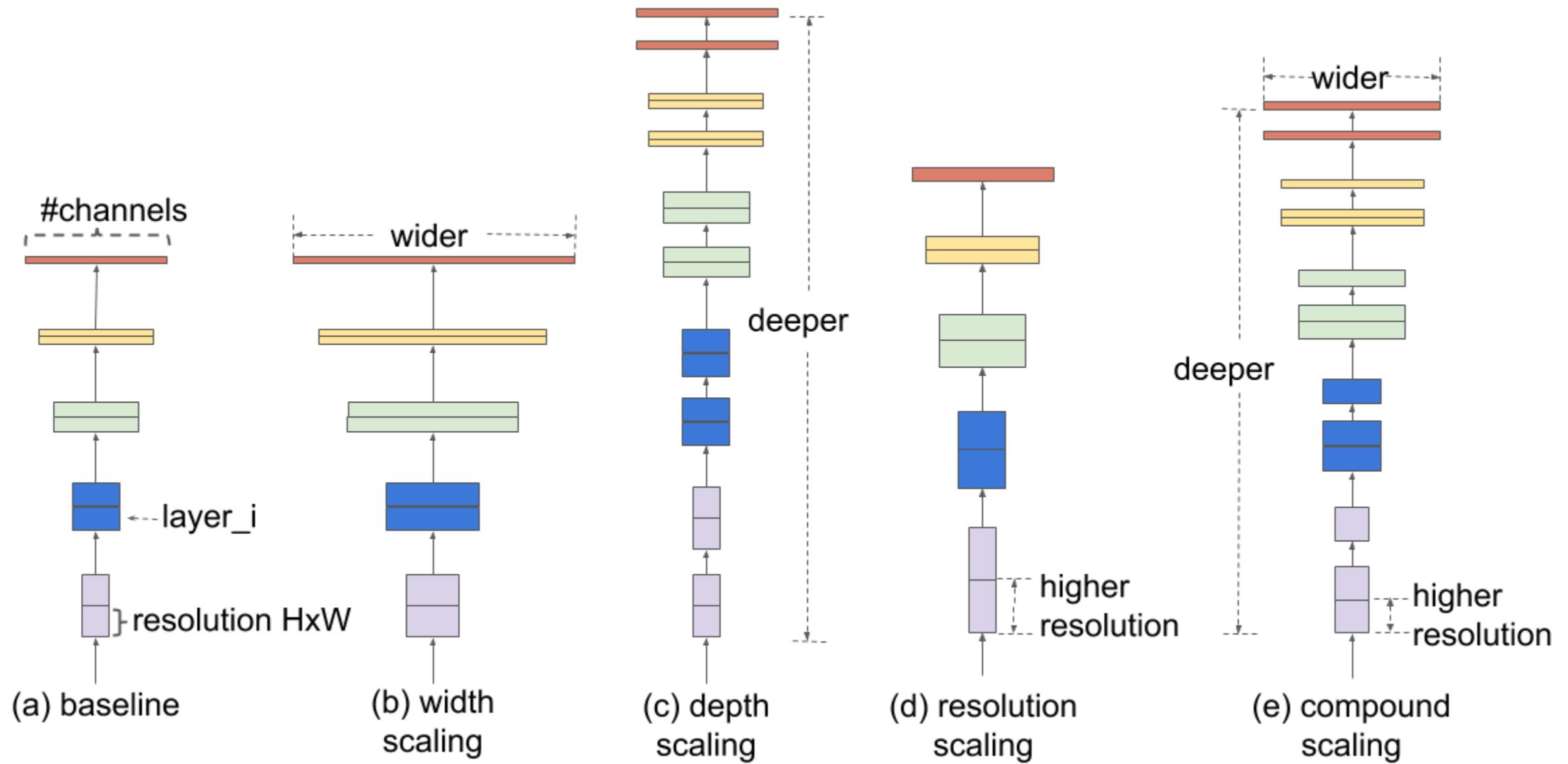


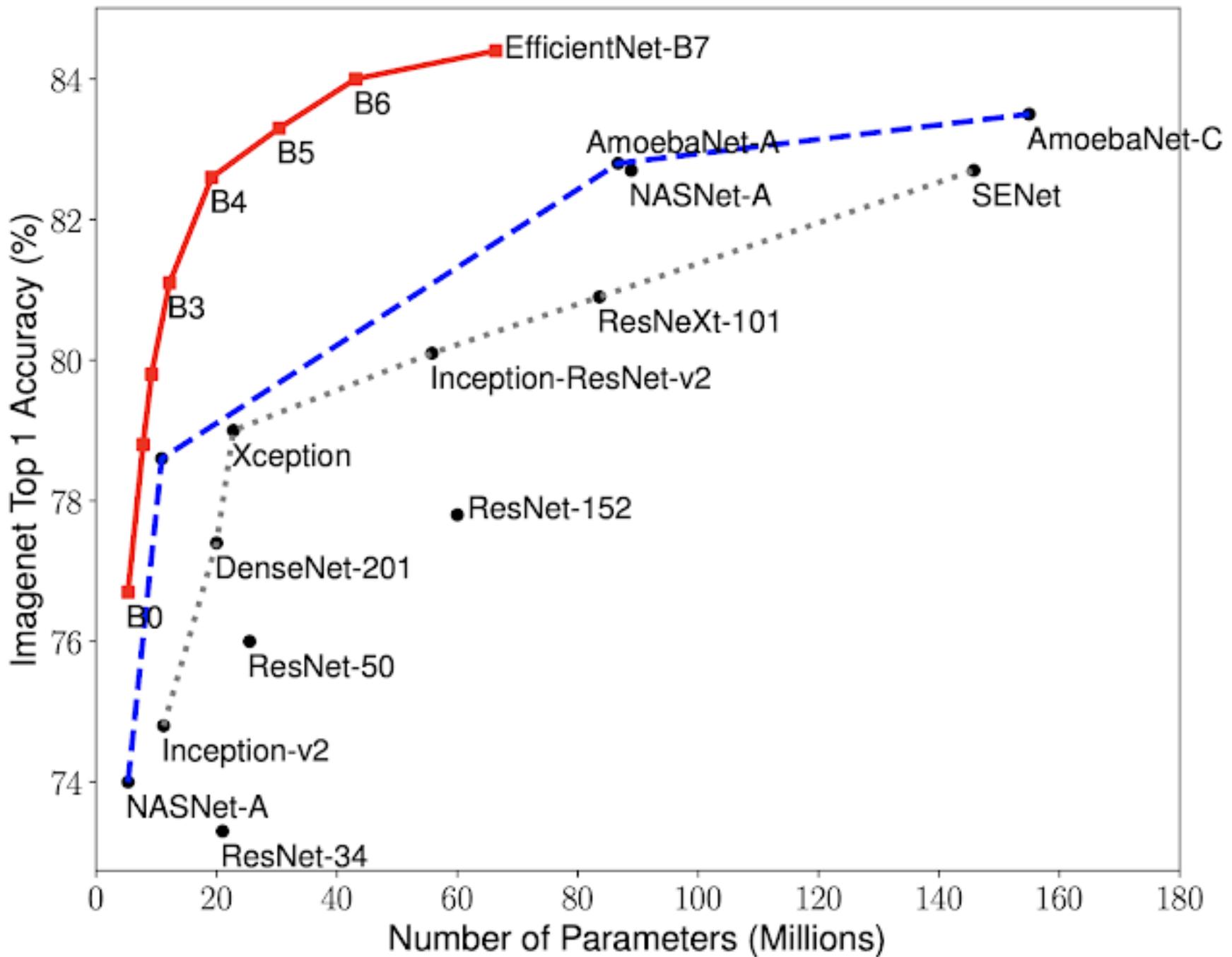
Network Architecture Search Network (NASNet)

- Learning neural network cells automatically (Google, 2017)



EfficientNet (May, 2019)





References

- Francois Chollet, “Deep Learning with Python,” Chapter 5.
- Adit Deshpande, [A Beginner's Guide To Understanding Convolutional Neural Networks.](#)
- Machine Learning Guru. [Understanding Convolutional Layers in Convolutional Neural Networks \(CNNs\)](#)
- [CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more](#)
- Wikipedia. [Convolution](#)
- <https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/>
- <http://neuralnetworksanddeeplearning.com/>
- Stanford’s CS231N
- Kunlun Bai, [A Comprehensive Introduction to Different Types of Convolutions in Deep Learning](#)