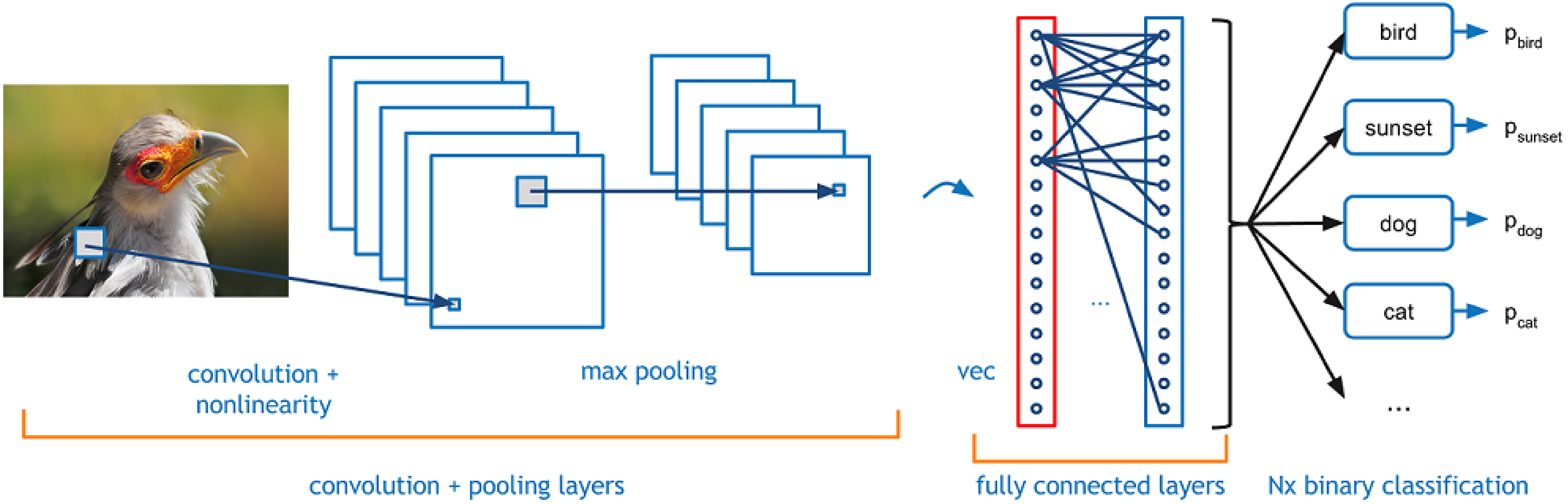
Convolutional Neural Network

Kuan-Ting Lai

2020/3/31

Convolutional Neural Networks (CNN)

- A.k.a. CNN or ConvNet



Adit Deshpande, [A Beginner's Guide To Understanding Convolutional Neural Networks]().

Digital Images

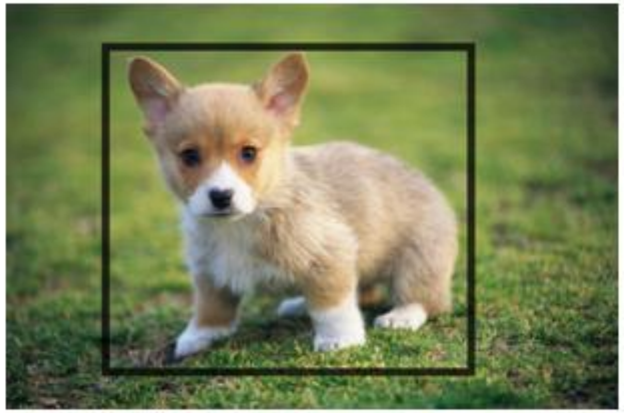
- Input array: an image's height × width × 3 (RGB)
- Value of each pixel: 0 - 255

What We See

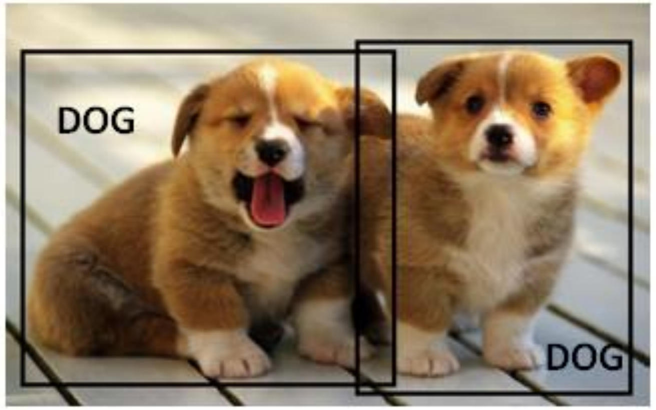
What Computers See

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.



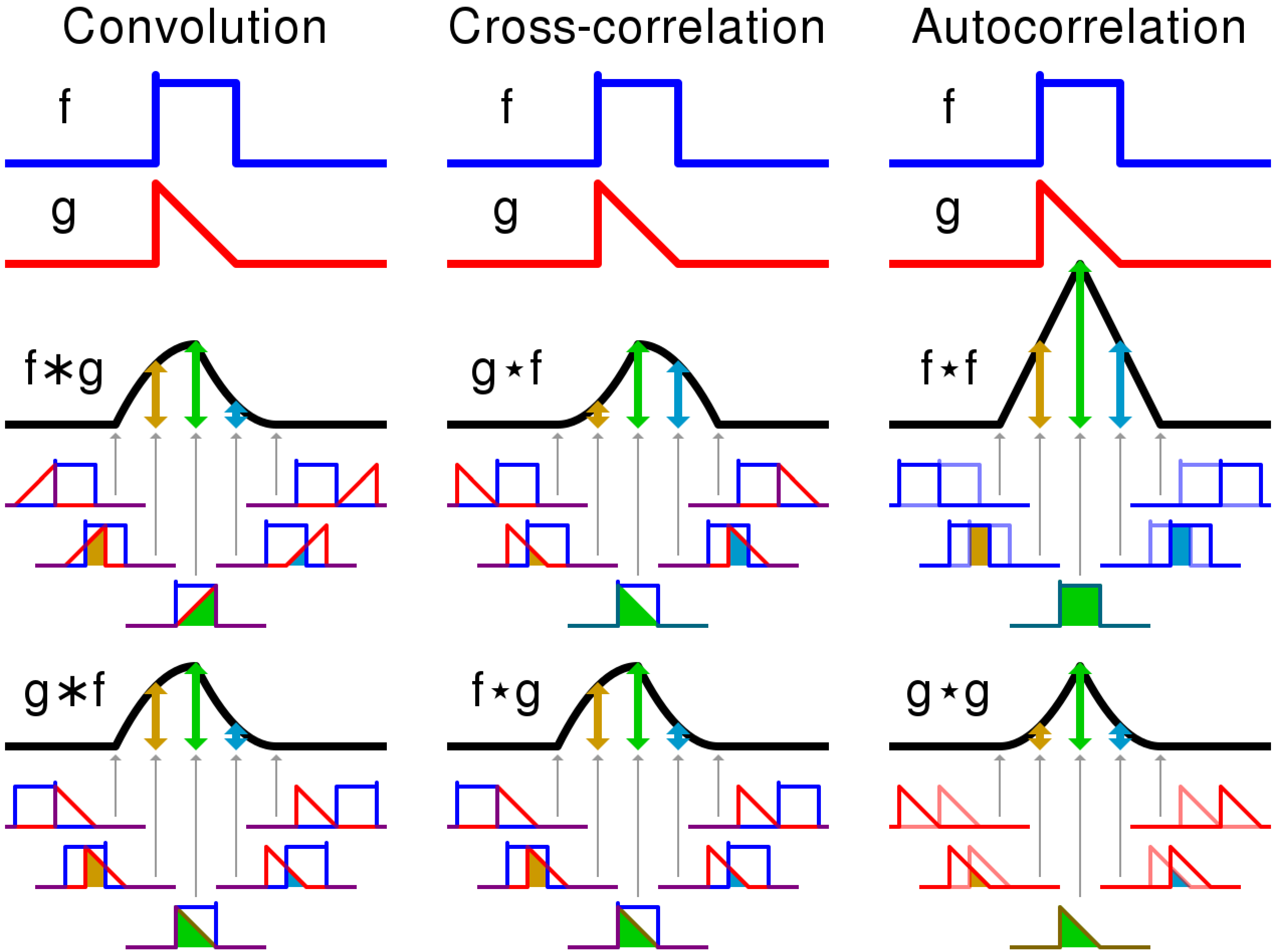
Object Detection involves localization of multiple objects (doesn't have to be the same class).

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
$g = f * h$	$\longleftrightarrow$	$G = FH$
$g = fh$	$\longleftrightarrow$	$G = F * H$

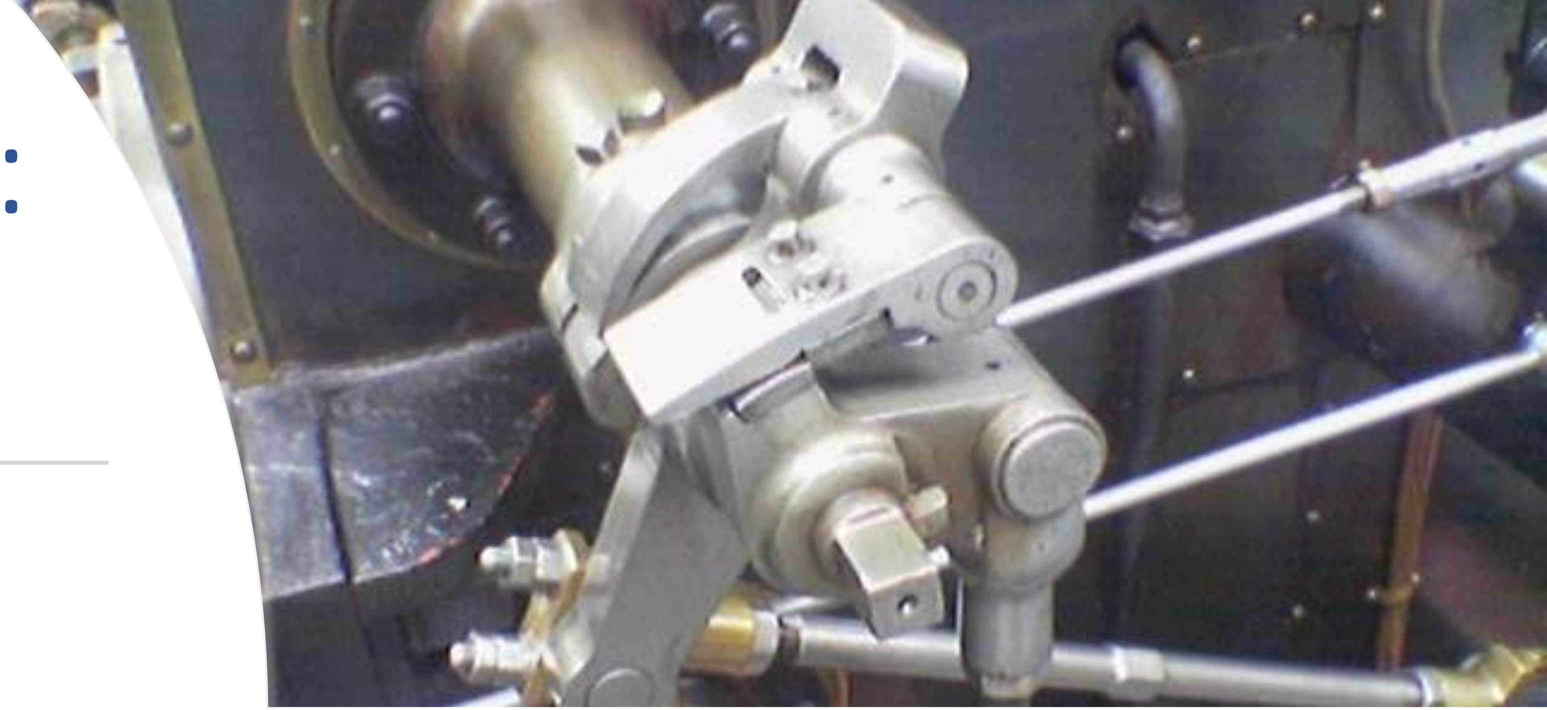

Convolution
Cross-correlation
Autocorrelation
f
g
f∗g
g∗f
g⋆f
f⋆g
f⋆f
g⋆g

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



Kernel Matrix

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

Image Matrix

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

Output Matrix

320				

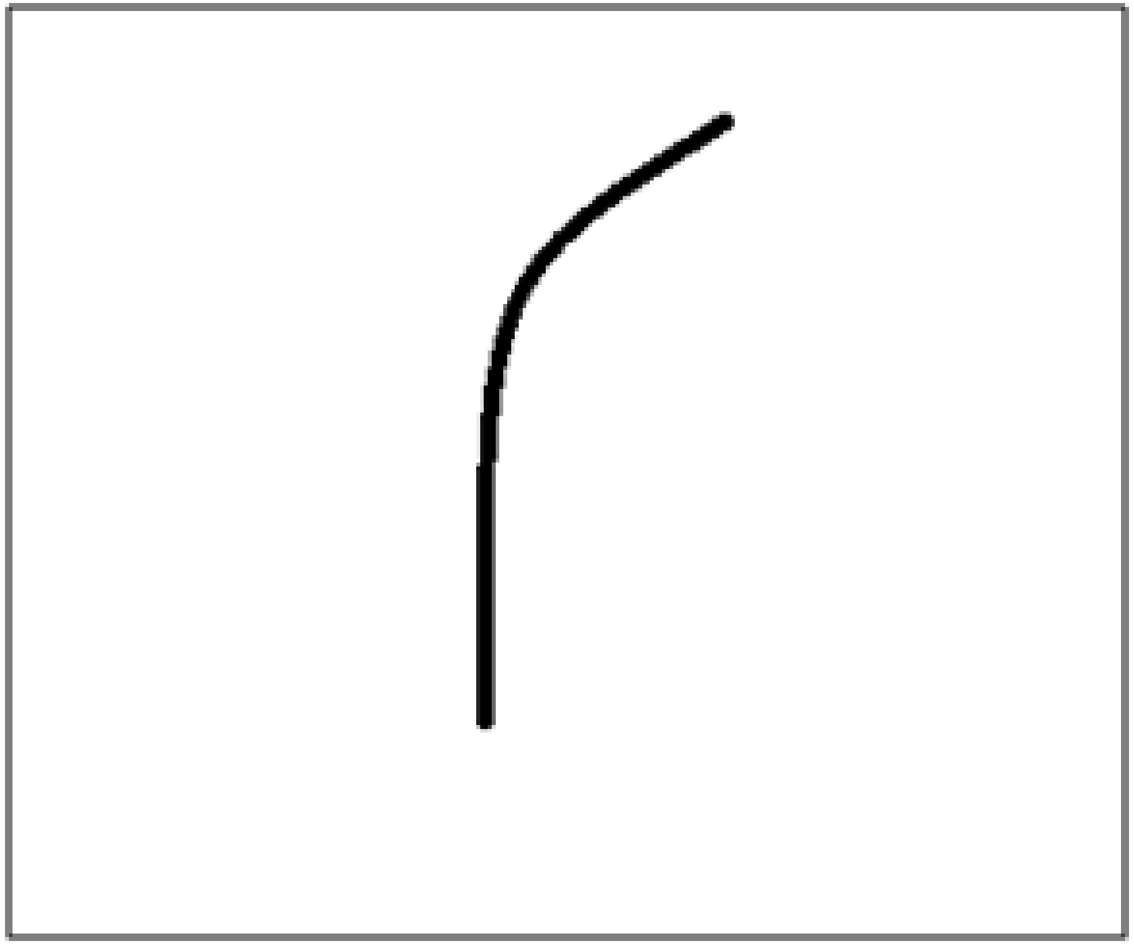
$$
\begin{aligned}
&0 * 0 + 0 * -1 + 0 * 0 \\
&+0 * -1 + 105 * 5 + 102 * -1 \\
&+0 * 0 + 103 * -1 + 99 * 0 \quad = 320
\end{aligned}
$$

Convolution with horizontal and vertical strides = 1

Example: A Curve Filter

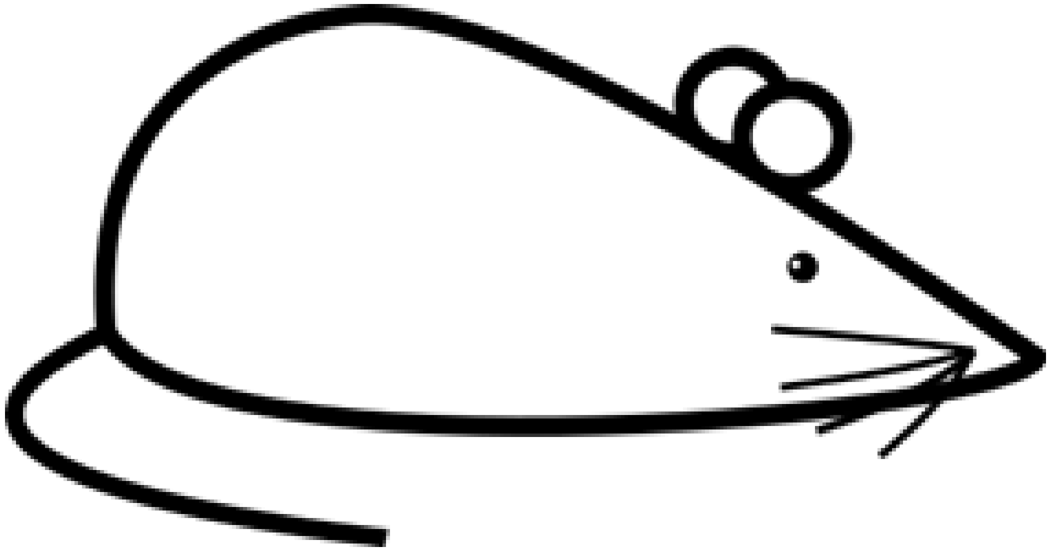
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

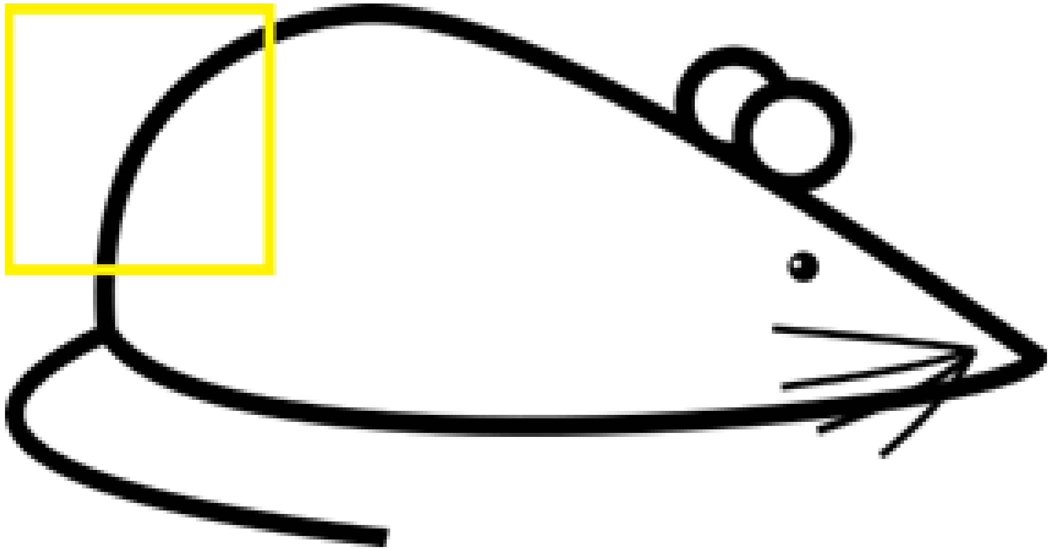


Visualization of a curve detector filter

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

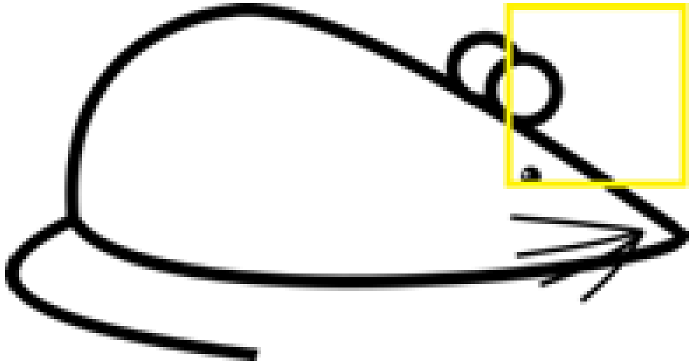
$$*$$

0	0	0	0	0	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 = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (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

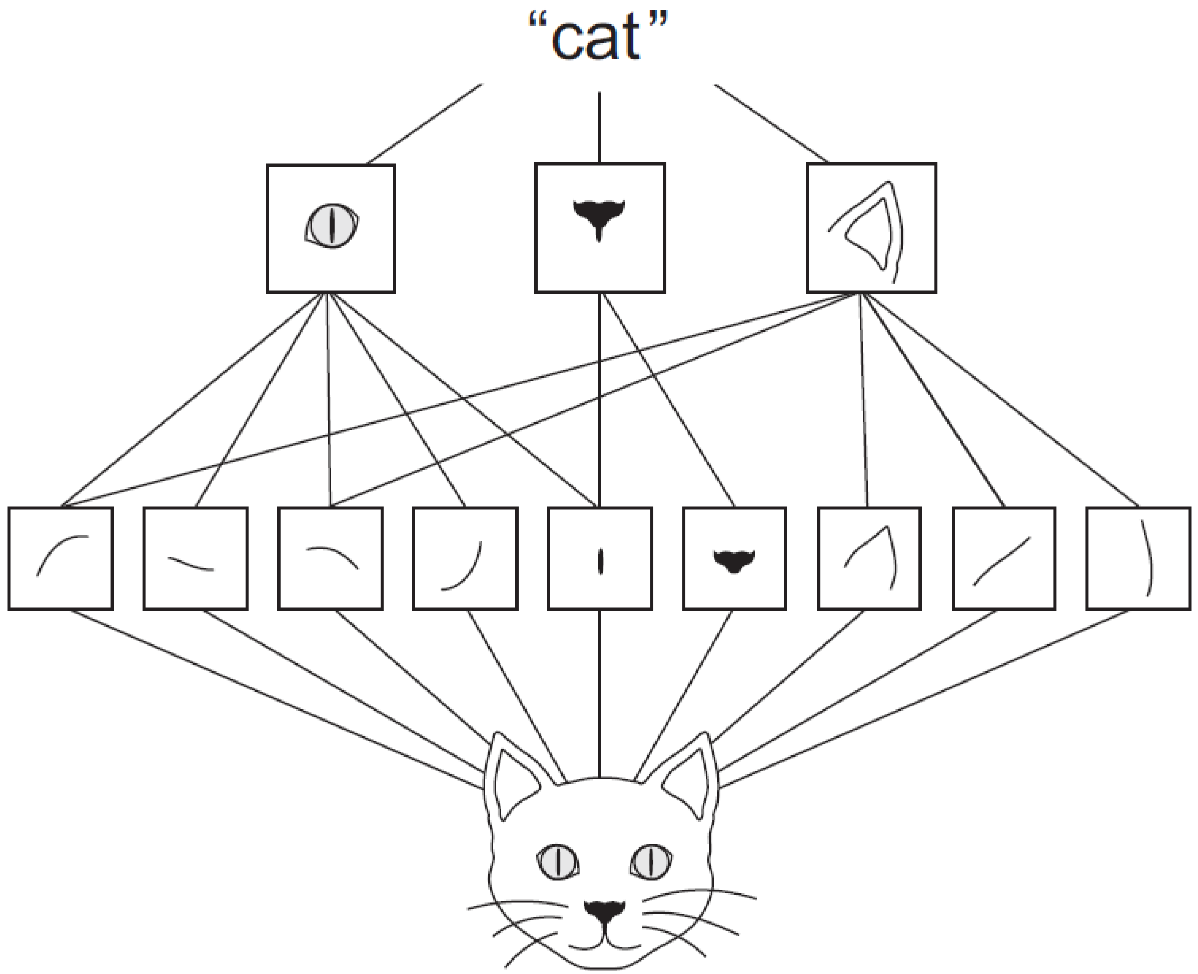
$*$

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()

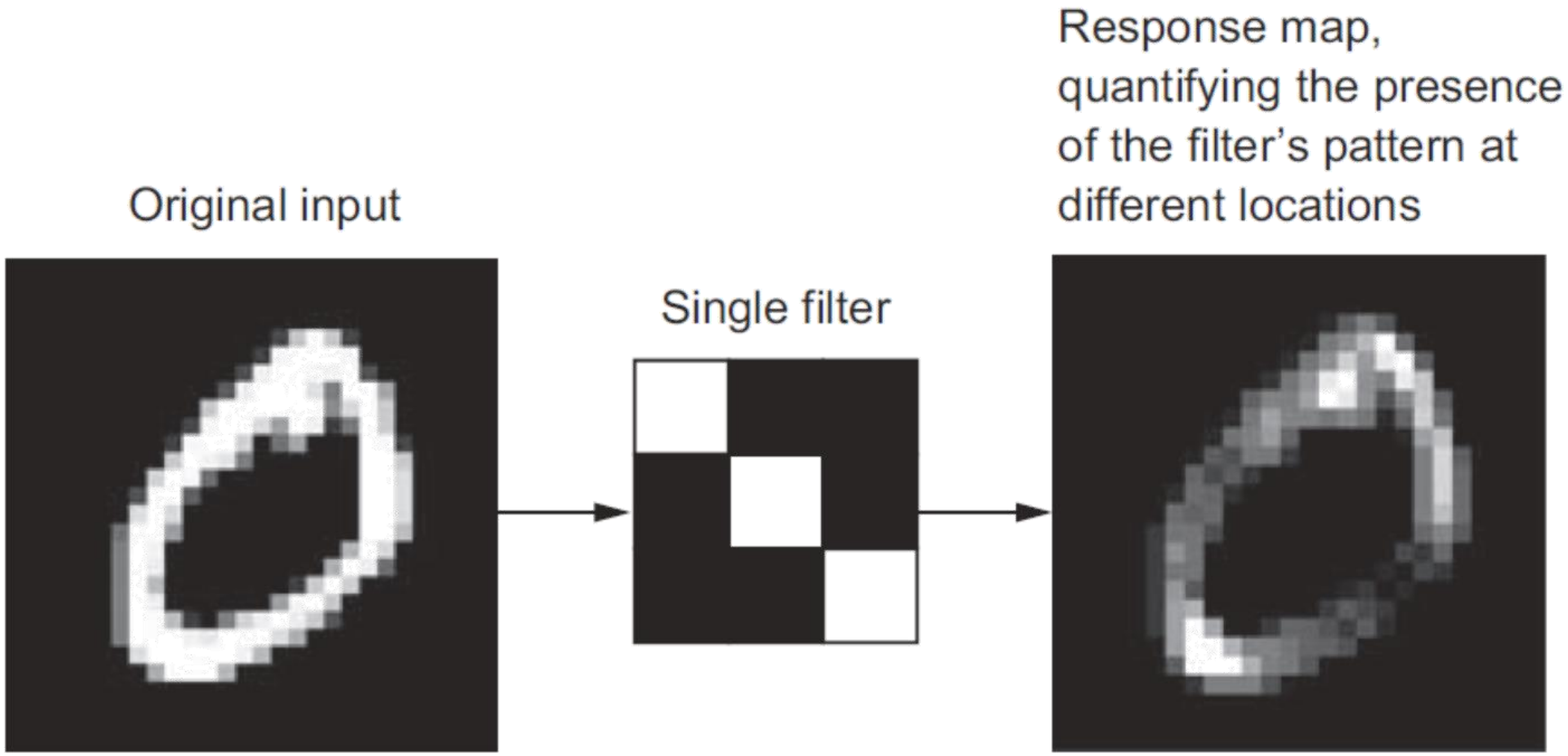
```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d_1 (Conv2D) (None, 26, 26, 32) 320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32) 0
_________________________________________________________________
conv2d_2 (Conv2D) (None, 11, 11, 64) 18496
_________________________________________________________________
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

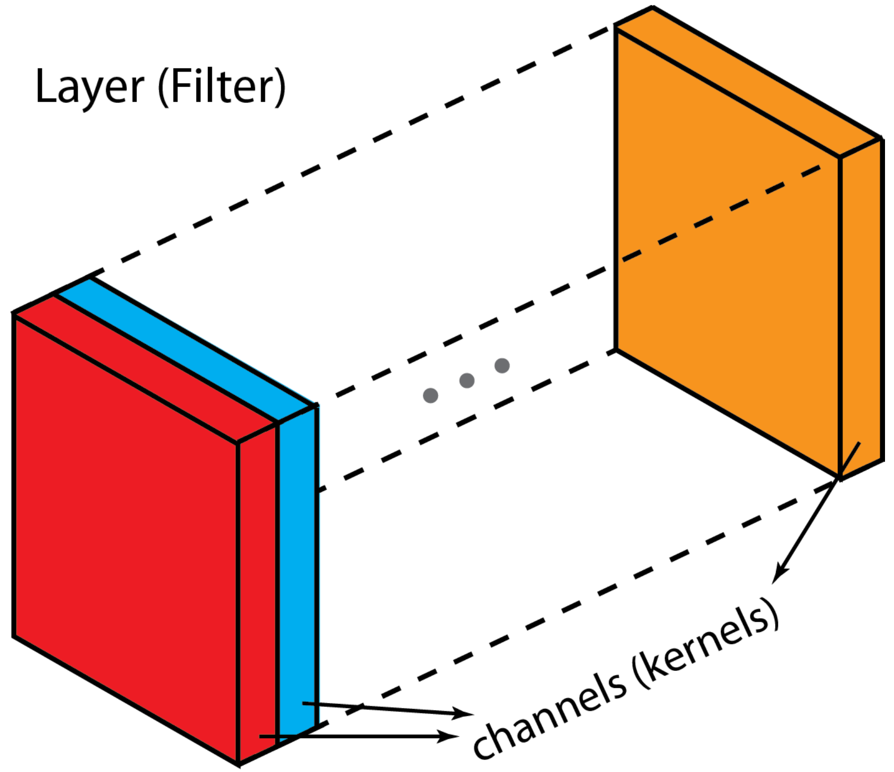
```
=>layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))
```

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

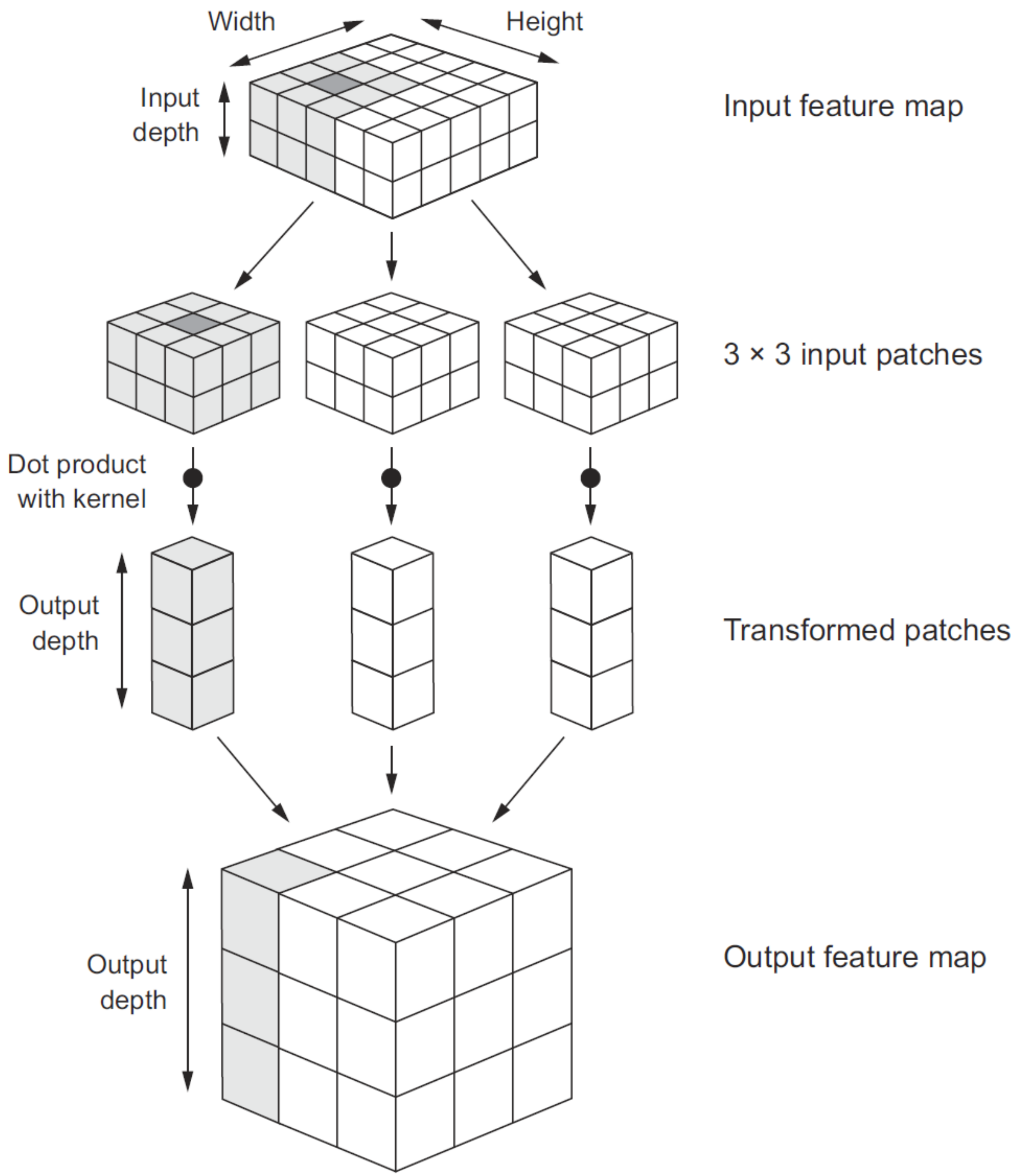


Kernel and Filter in Deep Learning

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



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215


Width
Height
Input
depth
Input feature map
3 × 3 input patches
Dot product
with kernel
Output
depth
Transformed patches
Output
depth
Output feature map

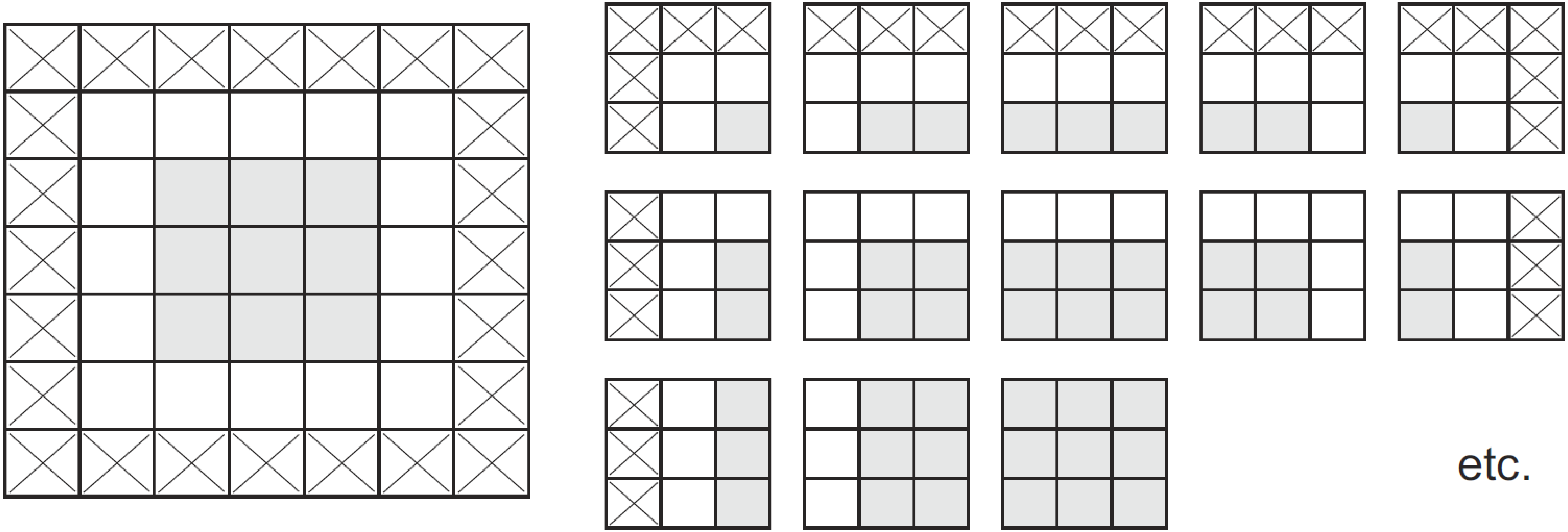
Add a Classifier on Top of ConvNet

```
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 (MaxPooling2 (None, 13, 13, 32) 0
_________________________________________________________________
conv2d_2 (Conv2D) (None, 11, 11, 64) 18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (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
=================================================================
Total params: 93,322 Trainable params: 93,322 Non-trainable params: 0
```

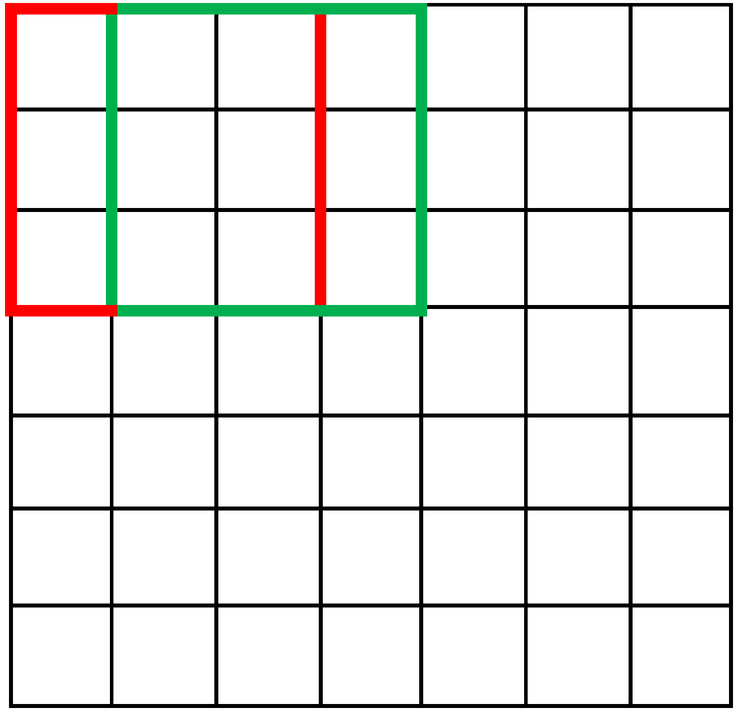
Padding

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

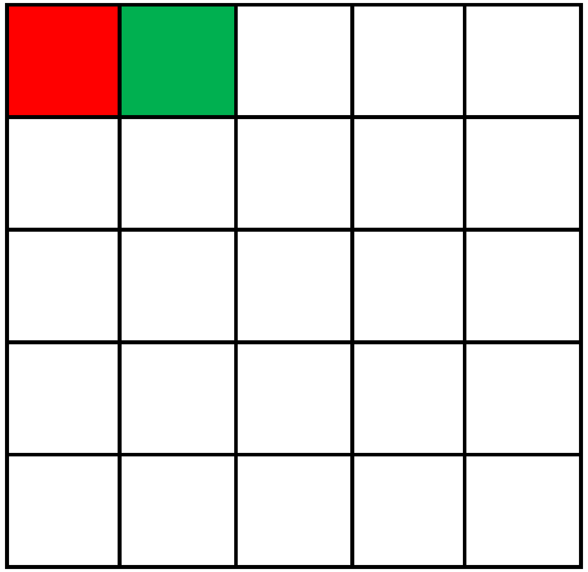


Stride=1

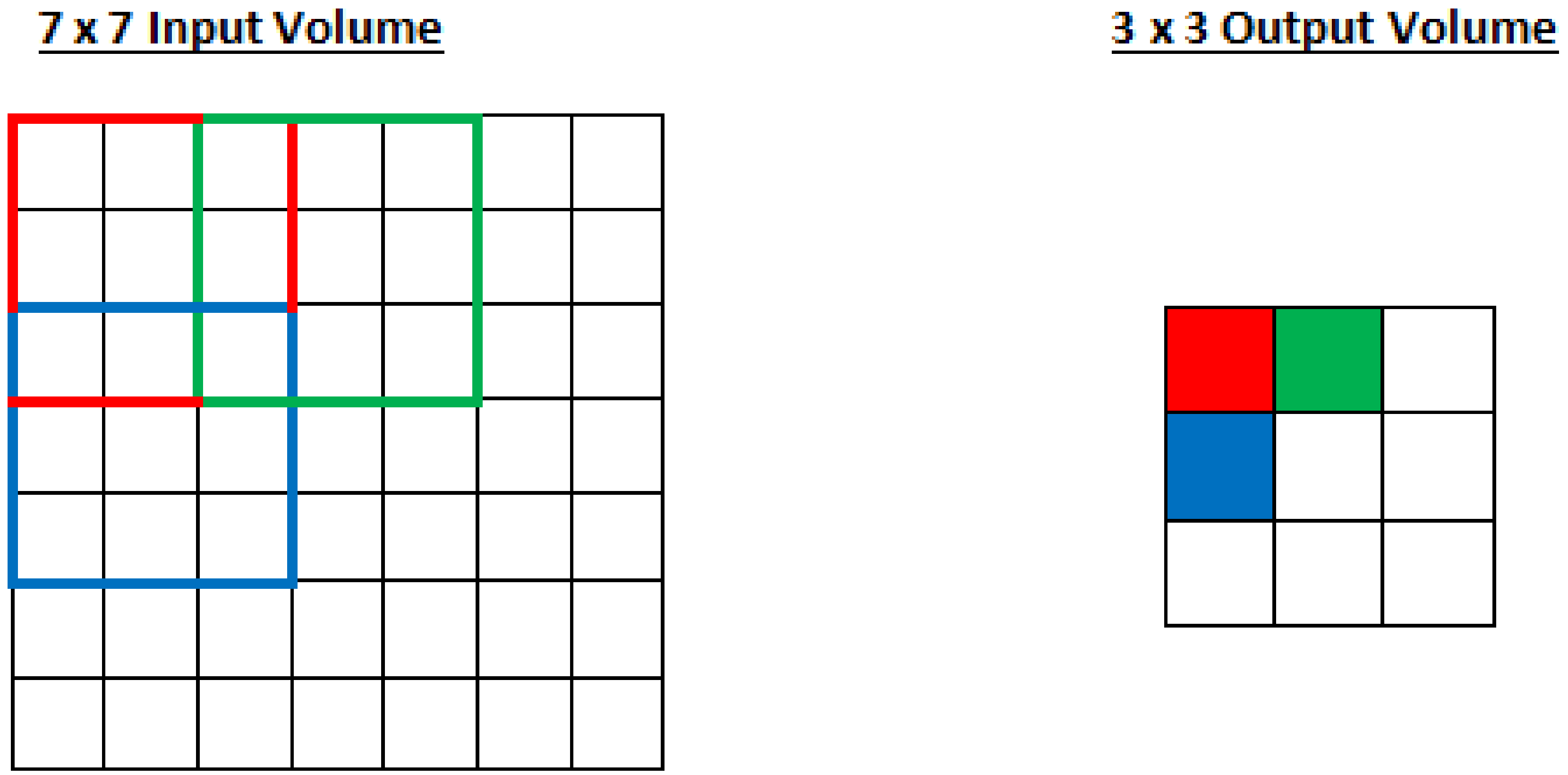
7 x 7 Input Volume



5 x 5 Output Volume

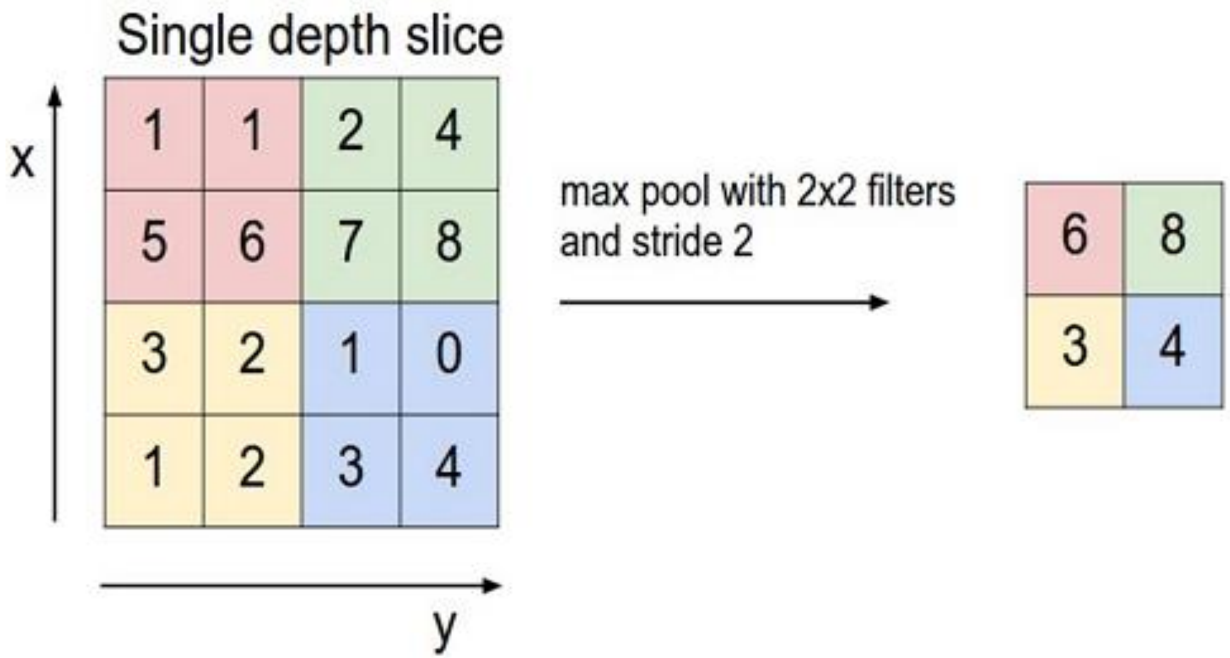


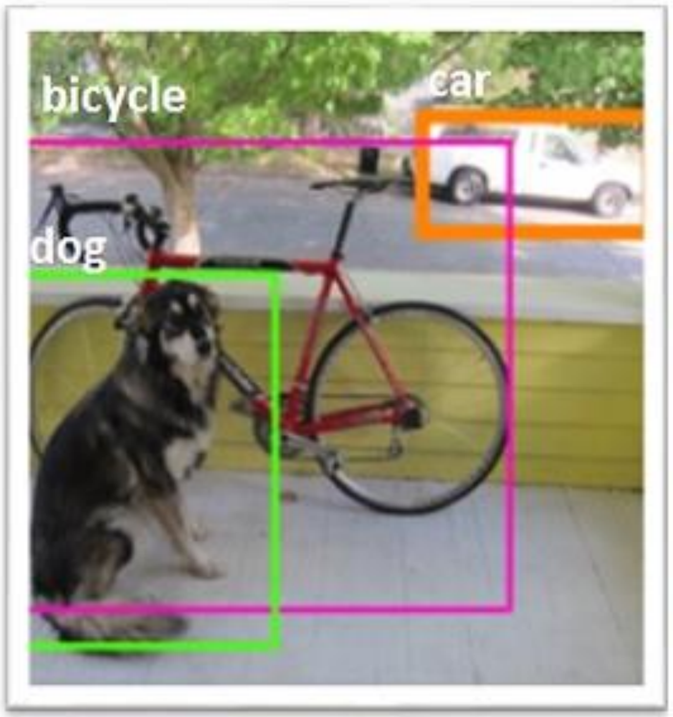
Stride=2

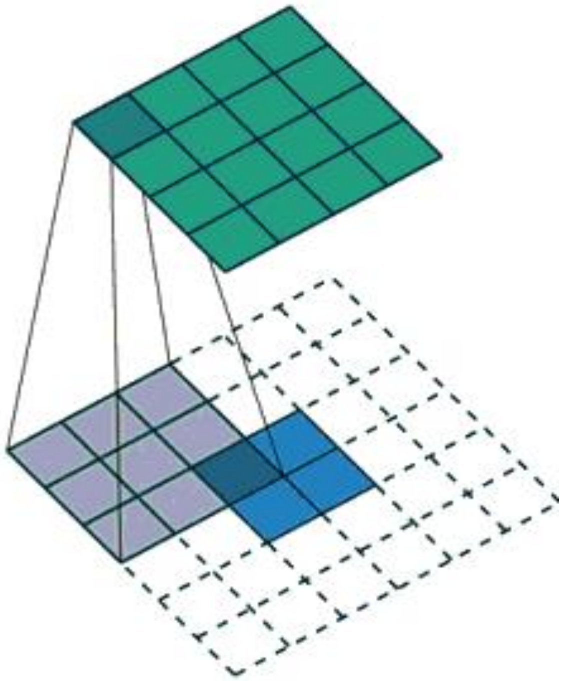


Max Pooling

- 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:

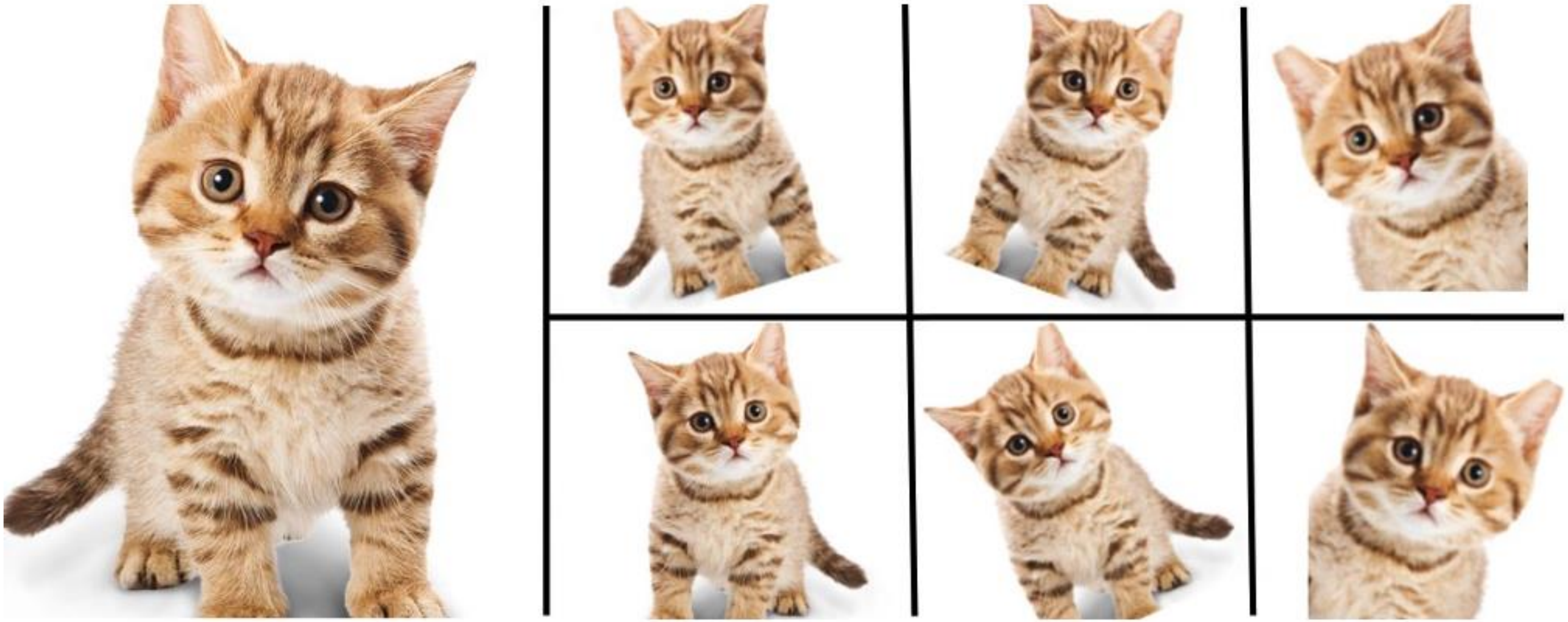
```
1
2
3
4
5
```

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



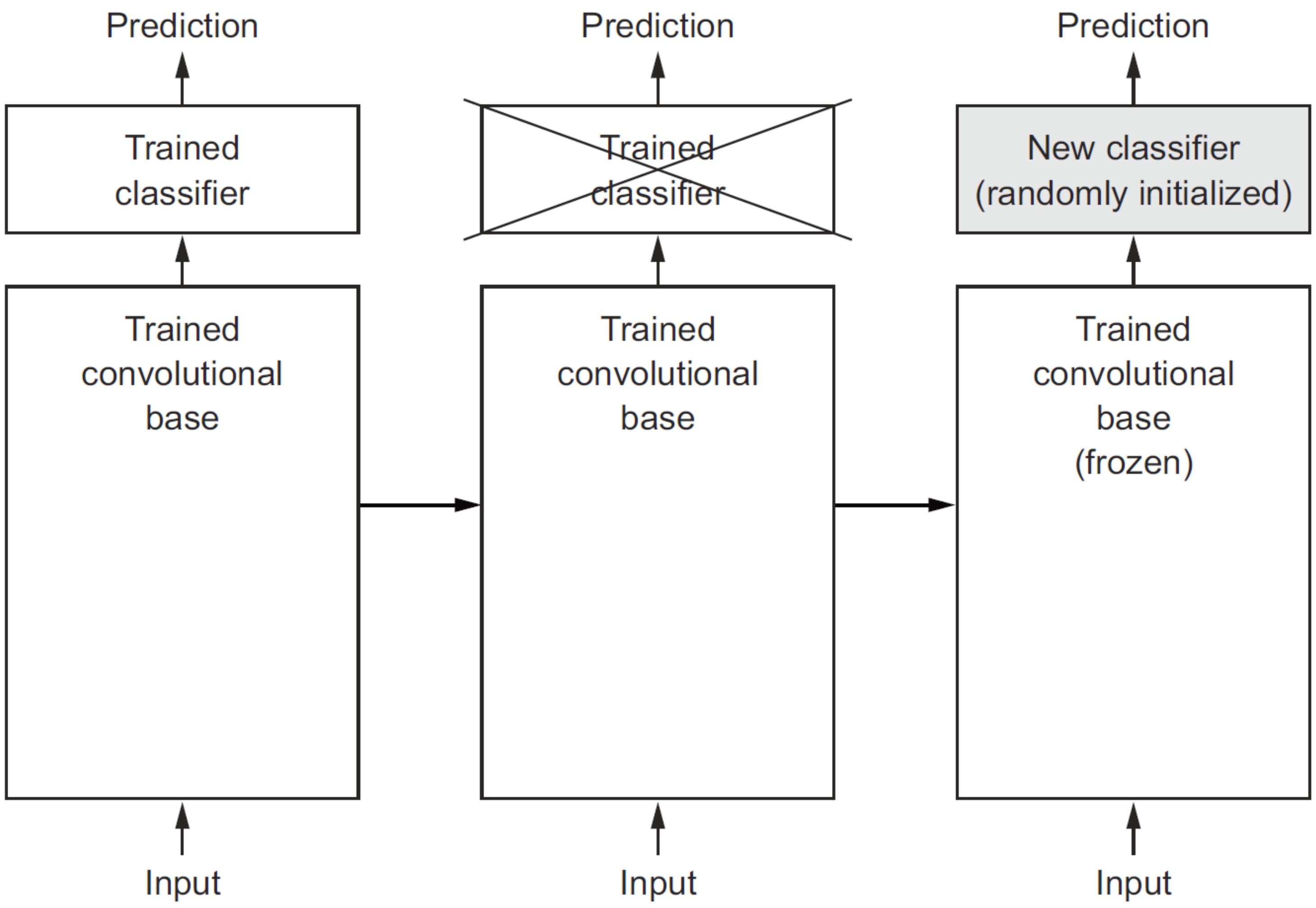
Data Augmentation via ImageDataGenerator

- 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 it 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
```

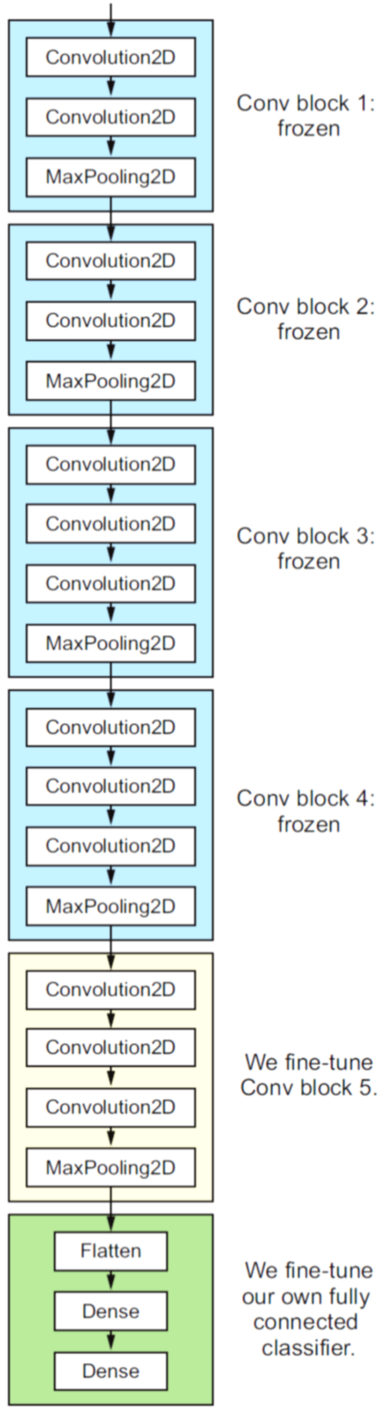
Fine-Tuning Top Few Layers

- 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

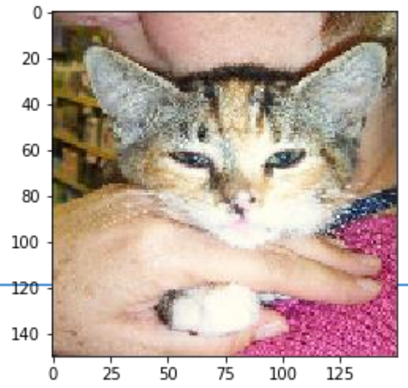
- Convnets are the best for Computer Vision (and maybe all the other tasks)
- Data augmentation is a powerful way to fight overfitting
- We can use pre-trained model for feature extraction
- We can further improve the pre-trained model on our dataset by fine-tuning

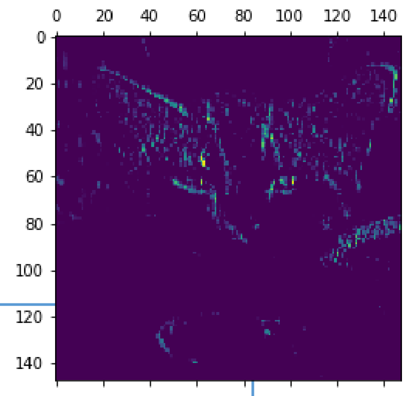
Visualizing What Convnets Learn

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

Number of features in the feature map

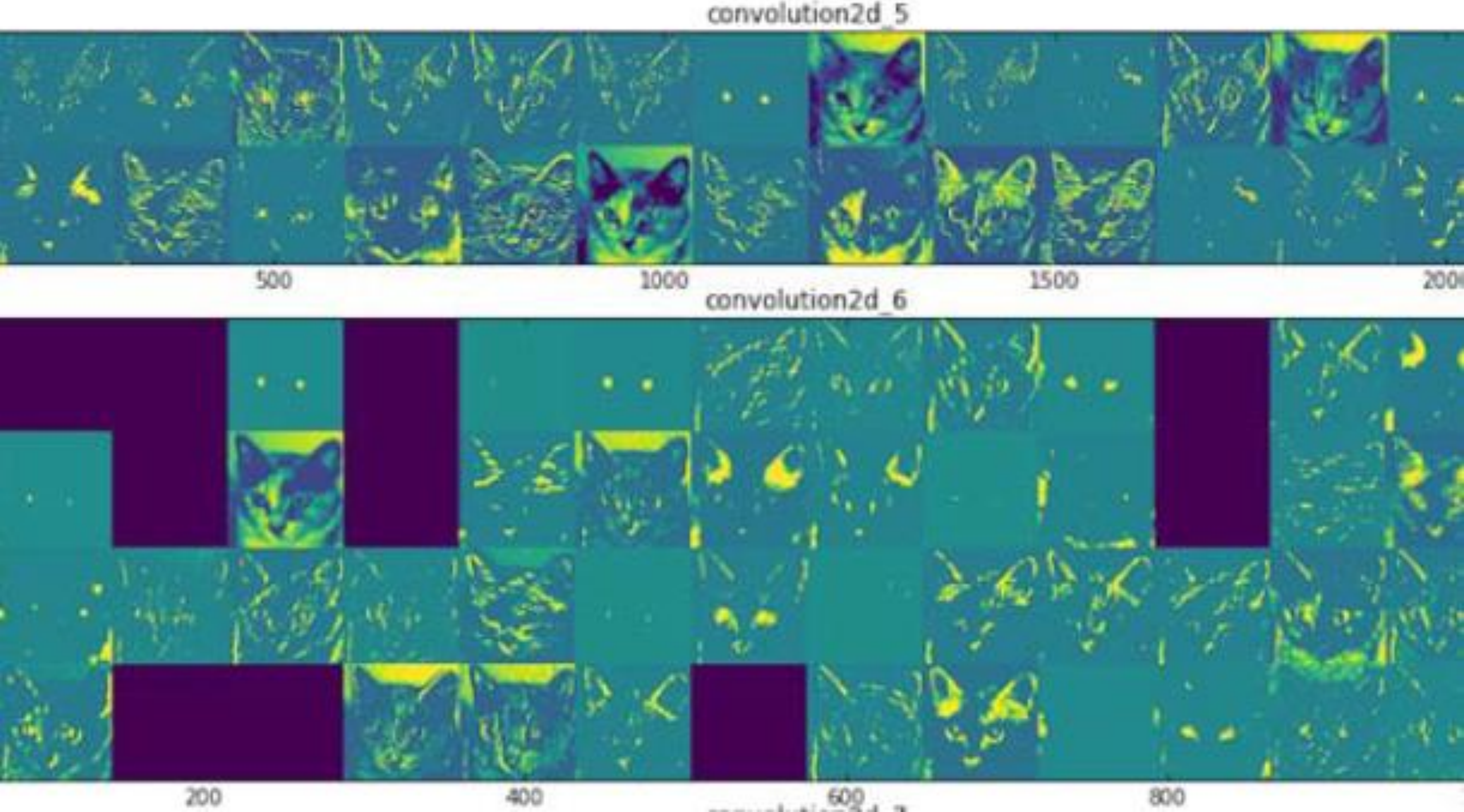
The feature map has shape (l, size, size, n_features).

Tiles the activation channels in this matrix

Tiles each filter into a big horizontal grid

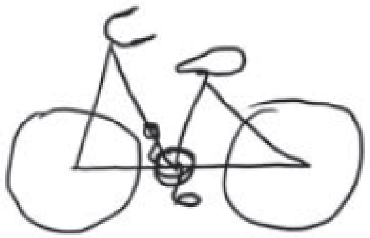
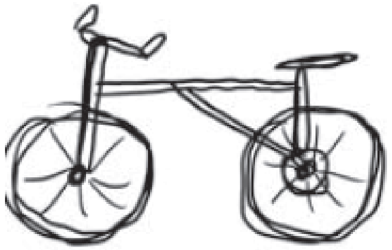
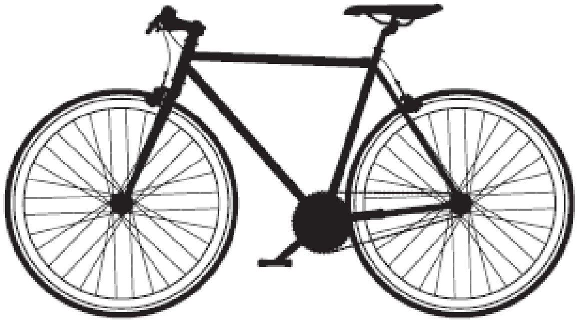
Post-processes the feature to make it visually palatable

Displays the grid


convolution2d_5
500
1000
1500
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.
for i in range(40):
    loss_value, grads_value = iterate([input_img_data])

    input_img_data += grads_value * step
```

Magnitude of each gradient update

Runs gradient ascent for 40 steps

Computes the loss value and gradient value

Adjusts the input image in the direction that maximizes the loss

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
```

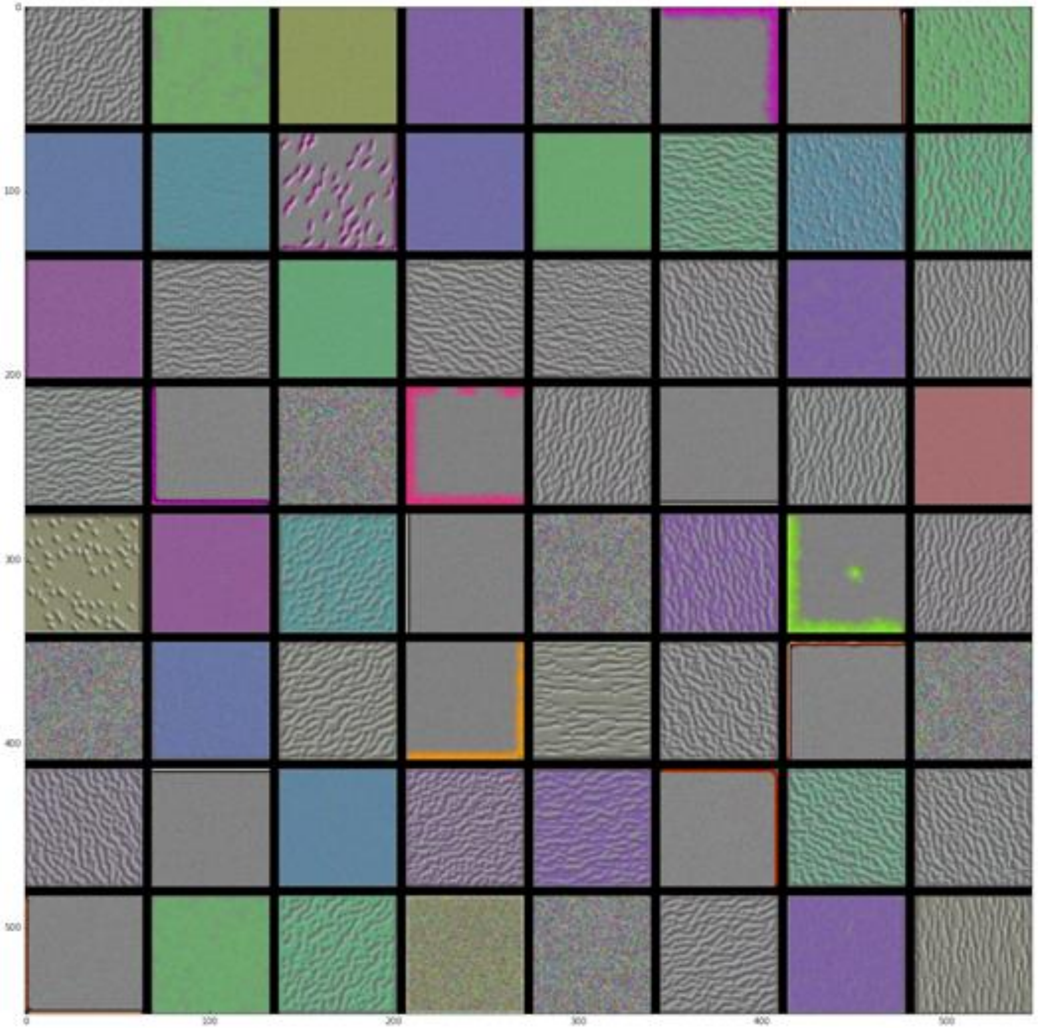
Normalizes the tensor: centers on 0, ensures that std is 0.1

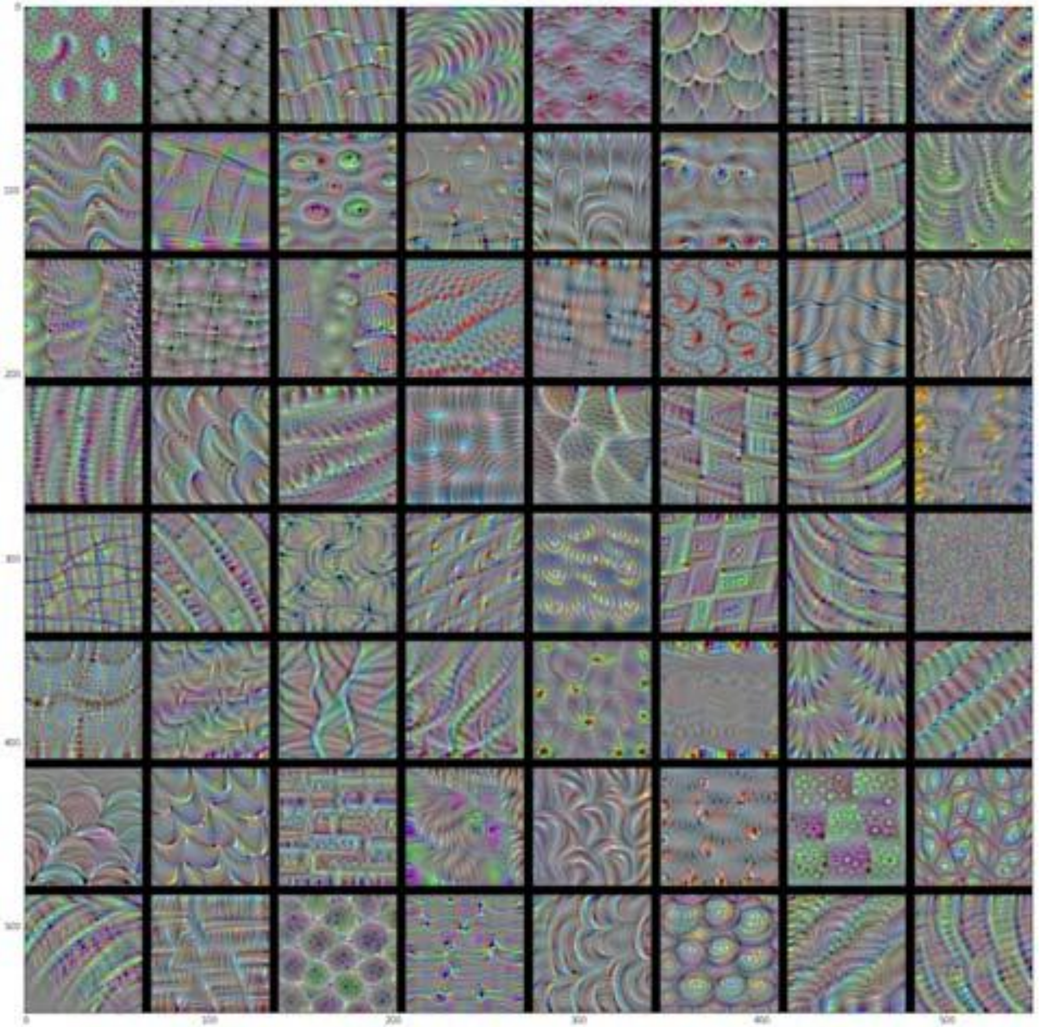
Clips to [0, 1]

Converts to an RGB array

```python
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, 150, 150, 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

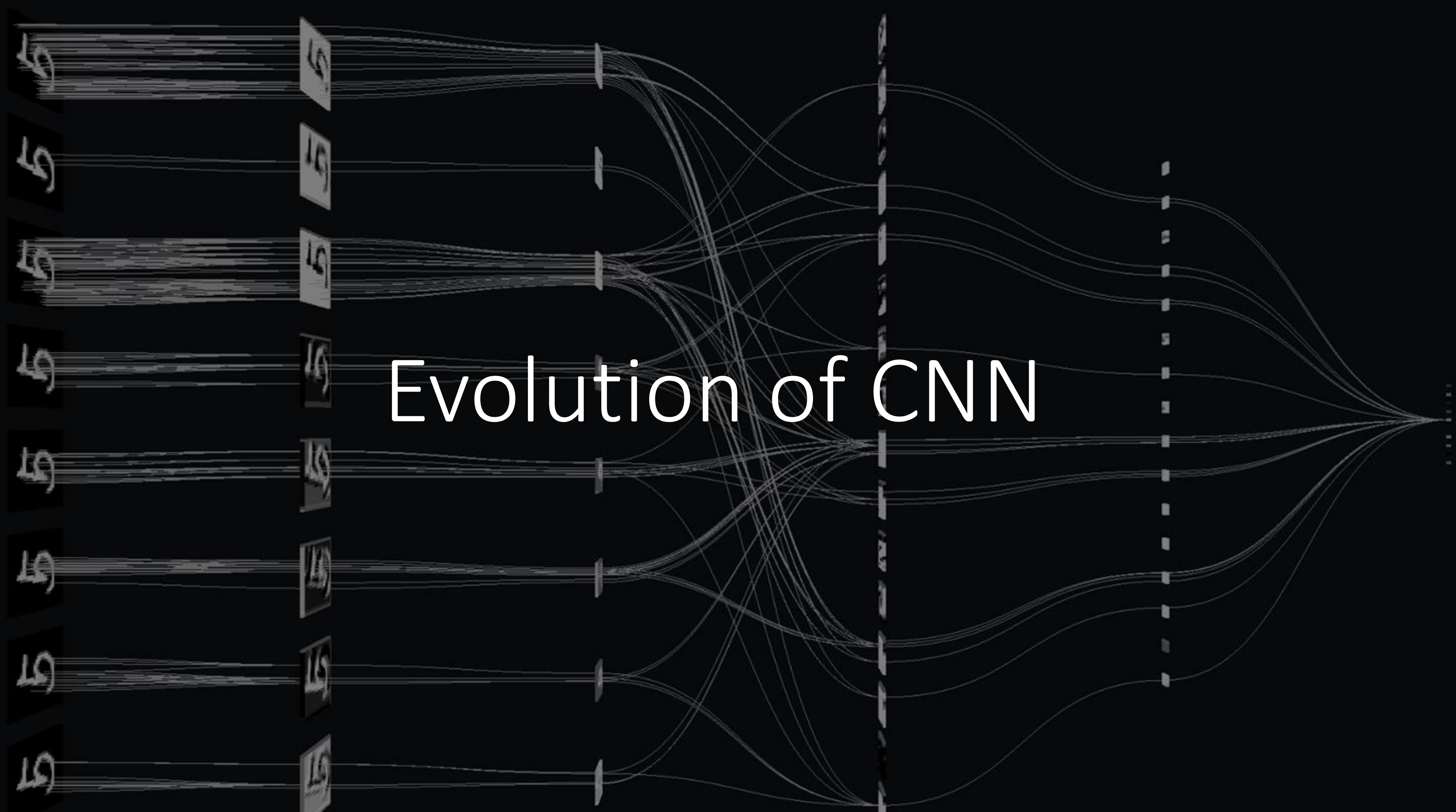




3. Visualizing Heatmaps of Class Activation

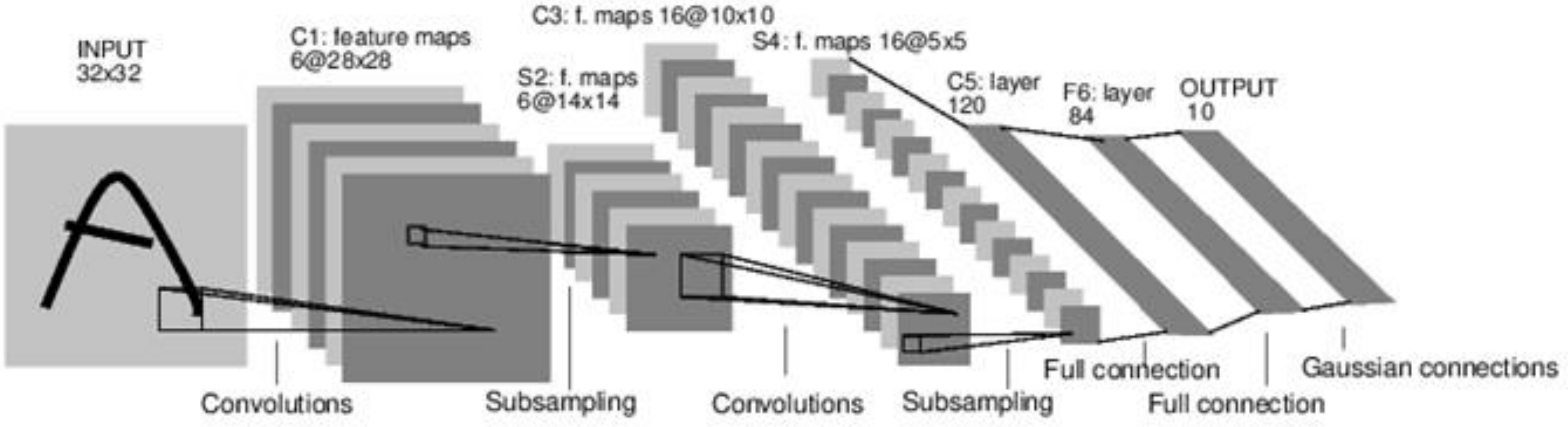
- Ramprasaath R. Selvaraju et al., "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization." arXiv (2017), https://arxiv.org/abs/1610.02391.

Evolution of CNN



Convolutional Neural Network (LeNet-5)

- https://medium.com/@sh.tsang/paper-brief-review-of-lenet-1-lenet-4-lenet-5-boosted-lenet-4-image-classification-1f5f809dbf17



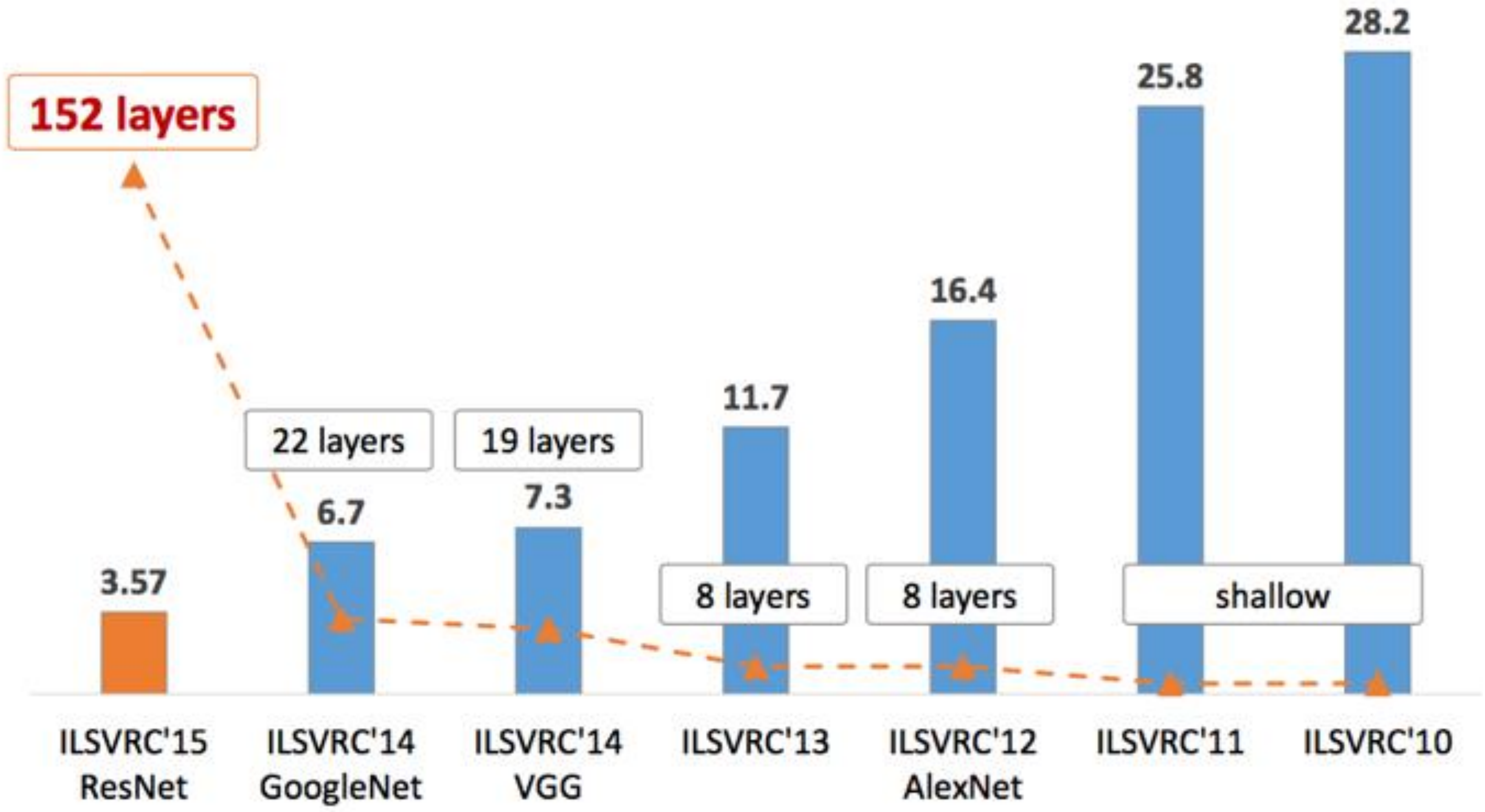
A Full Convolutional Neural Network (LeNet)


IMAGENET

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

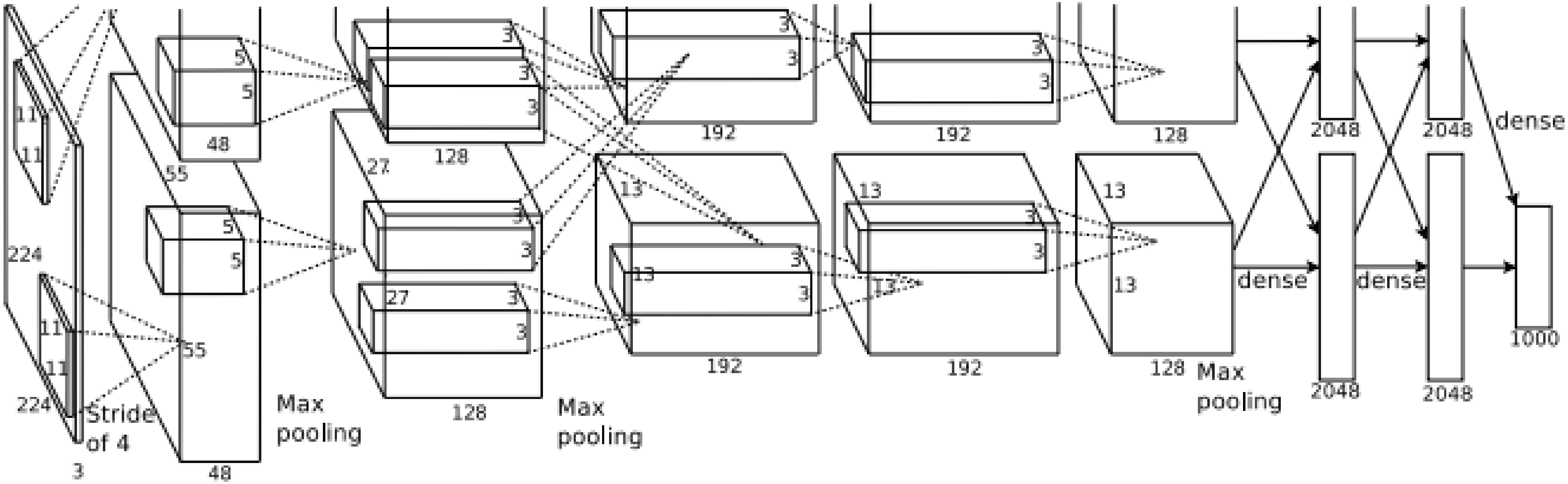
Error Rate on ImageNet Challenge




I WAS WINNING IMAGENET
UNTIL A DEEPER MODEL CAME ALONG
imgflip.com

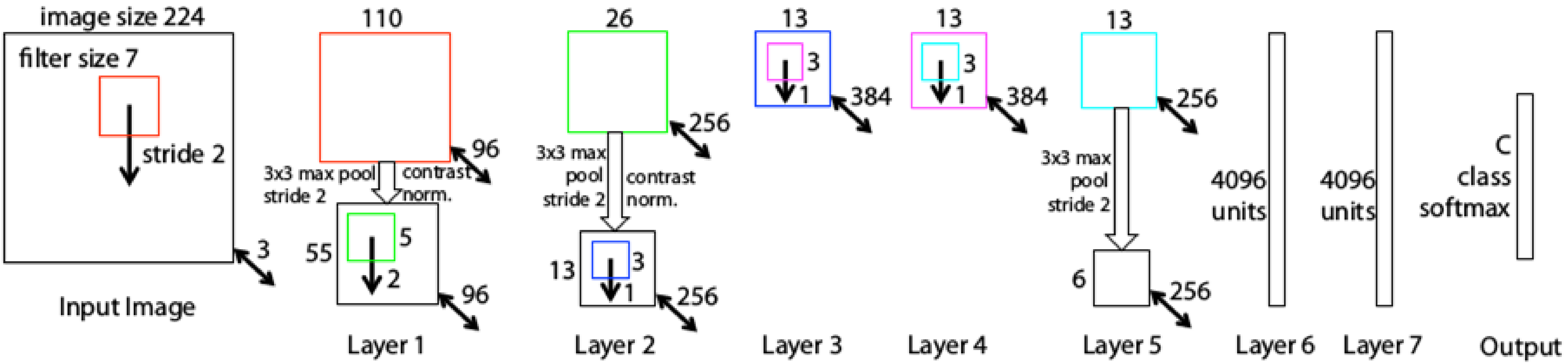
AlexNet (2012)

- 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)

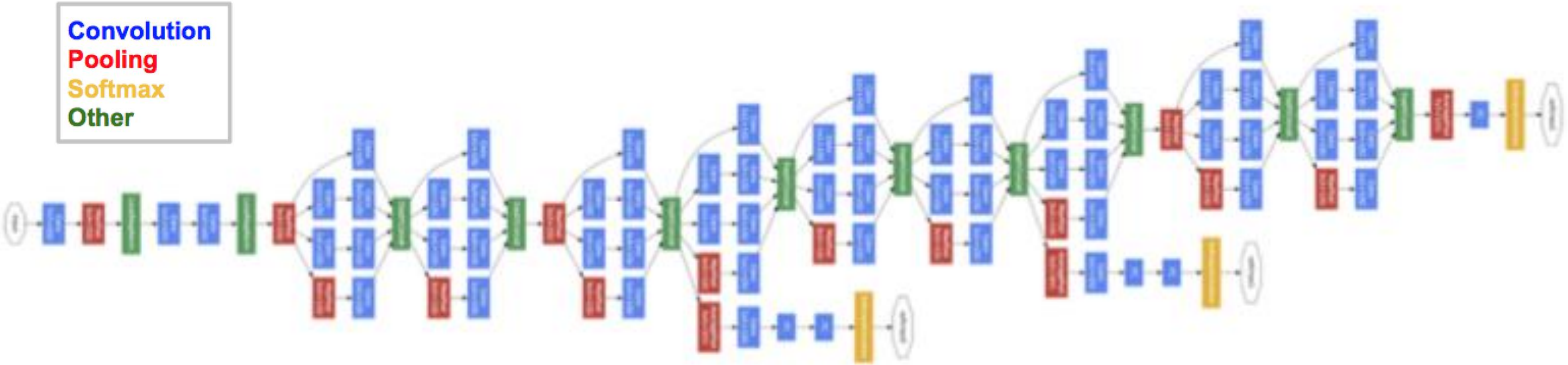
- Parameter tuning of AlexNet



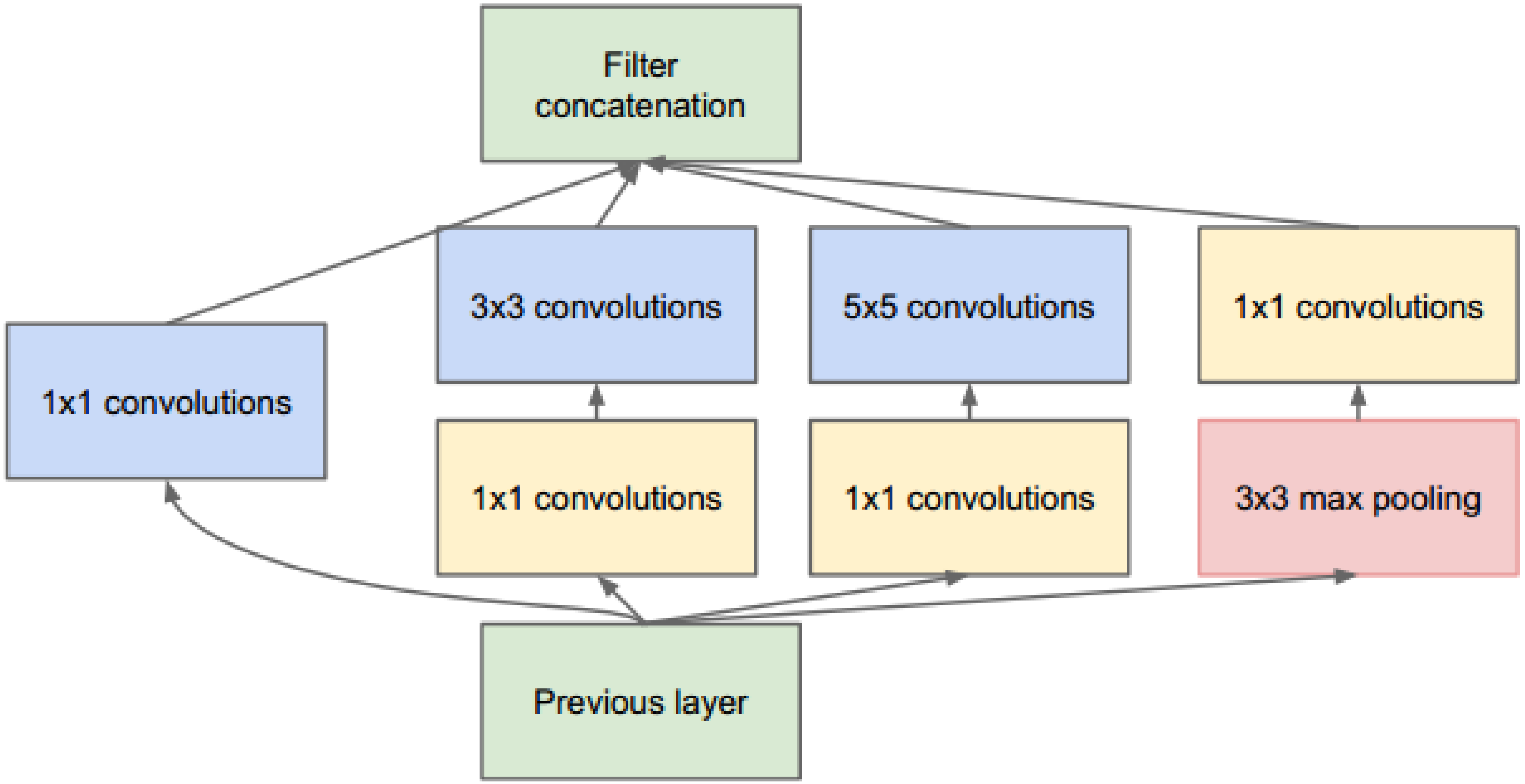
ZF Net Architecture

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

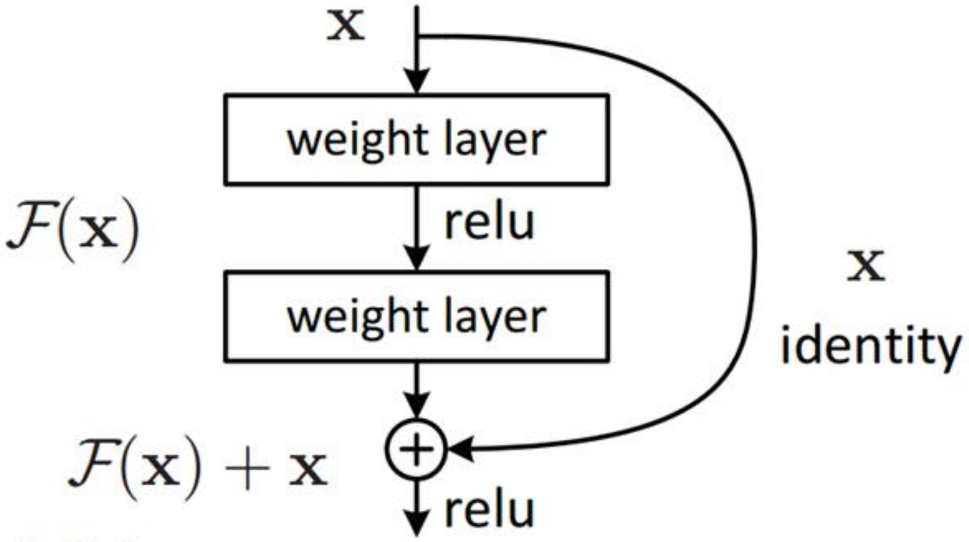
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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 architecures of VGG Net. Configuration D produced the best results

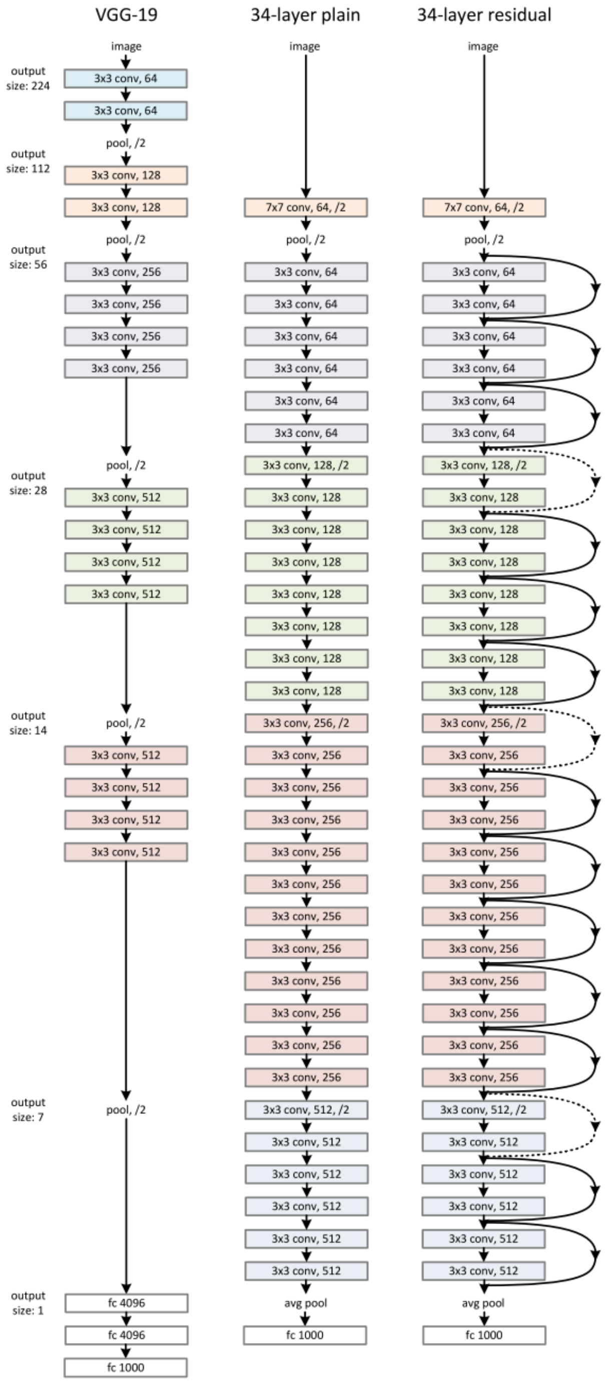

WE NEED TO GO
DEEPER
memegenerator.net

ResNet (2015)

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

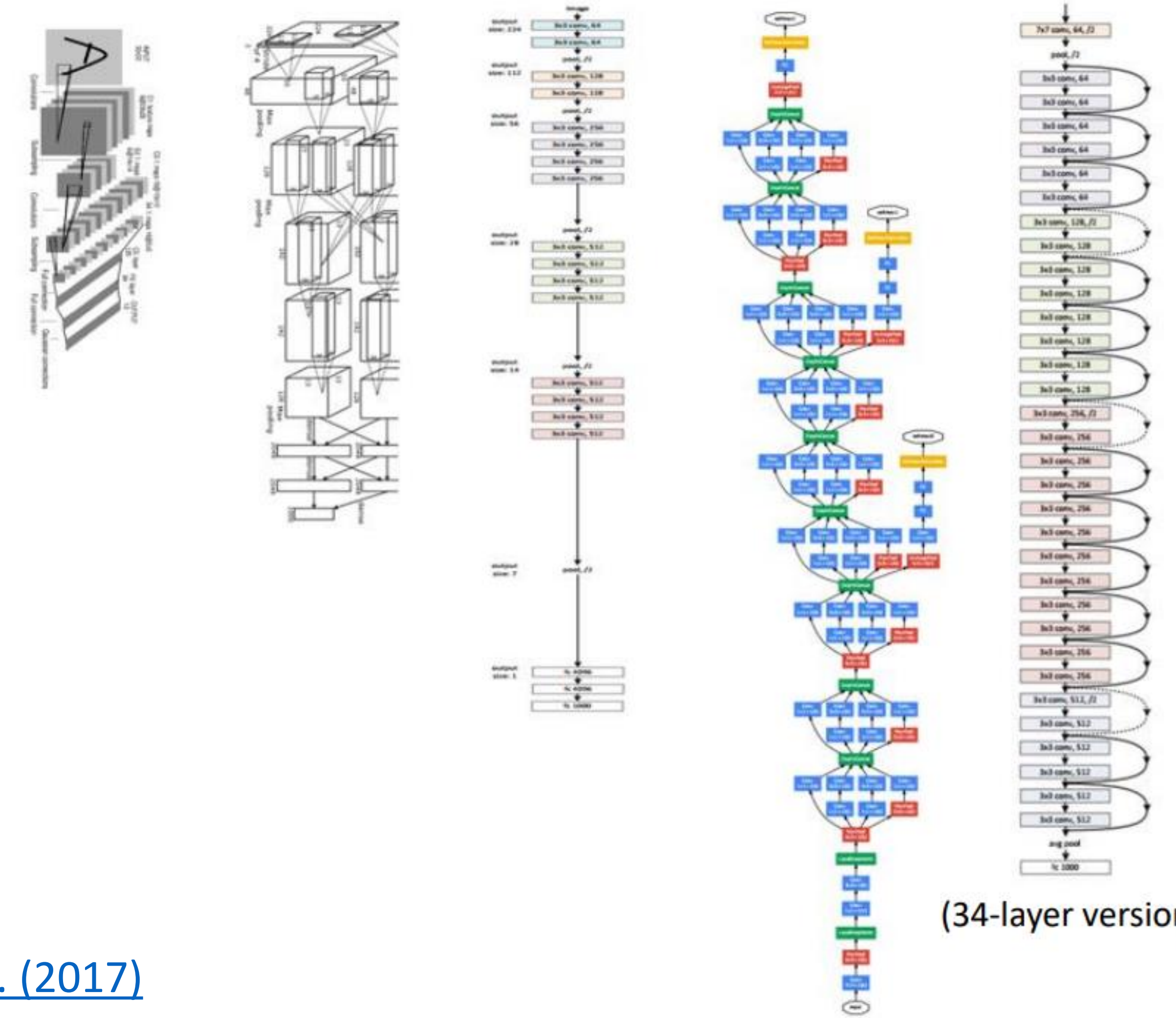


A residual block



Visualizing CNN Side-by-Side

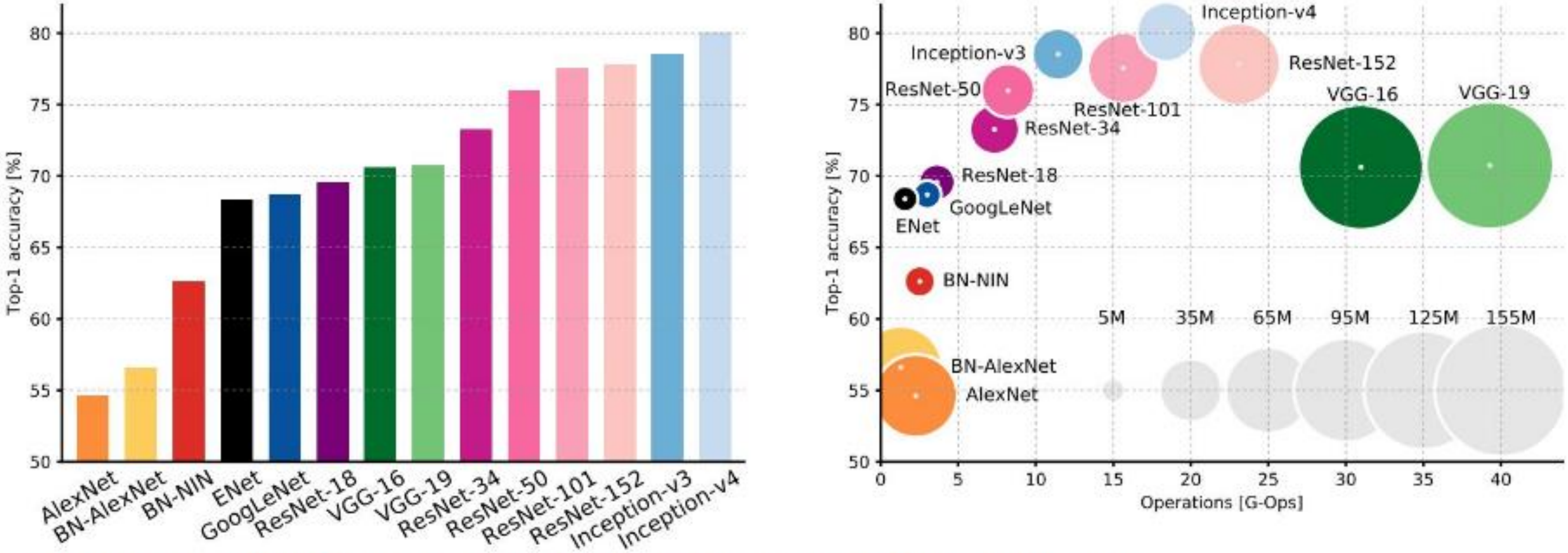
LeNet	AlexNet	VGGNet	GoogLeNet	ResNet
(1998)	(2012)	(2014)	(2014)	(2015)
7 Layers	**8** Layers	**19** Layers	**22** Layers	**152** Layers



(34-layer version)

[Teerapittayanon et al. (2017)](#)

Statistics

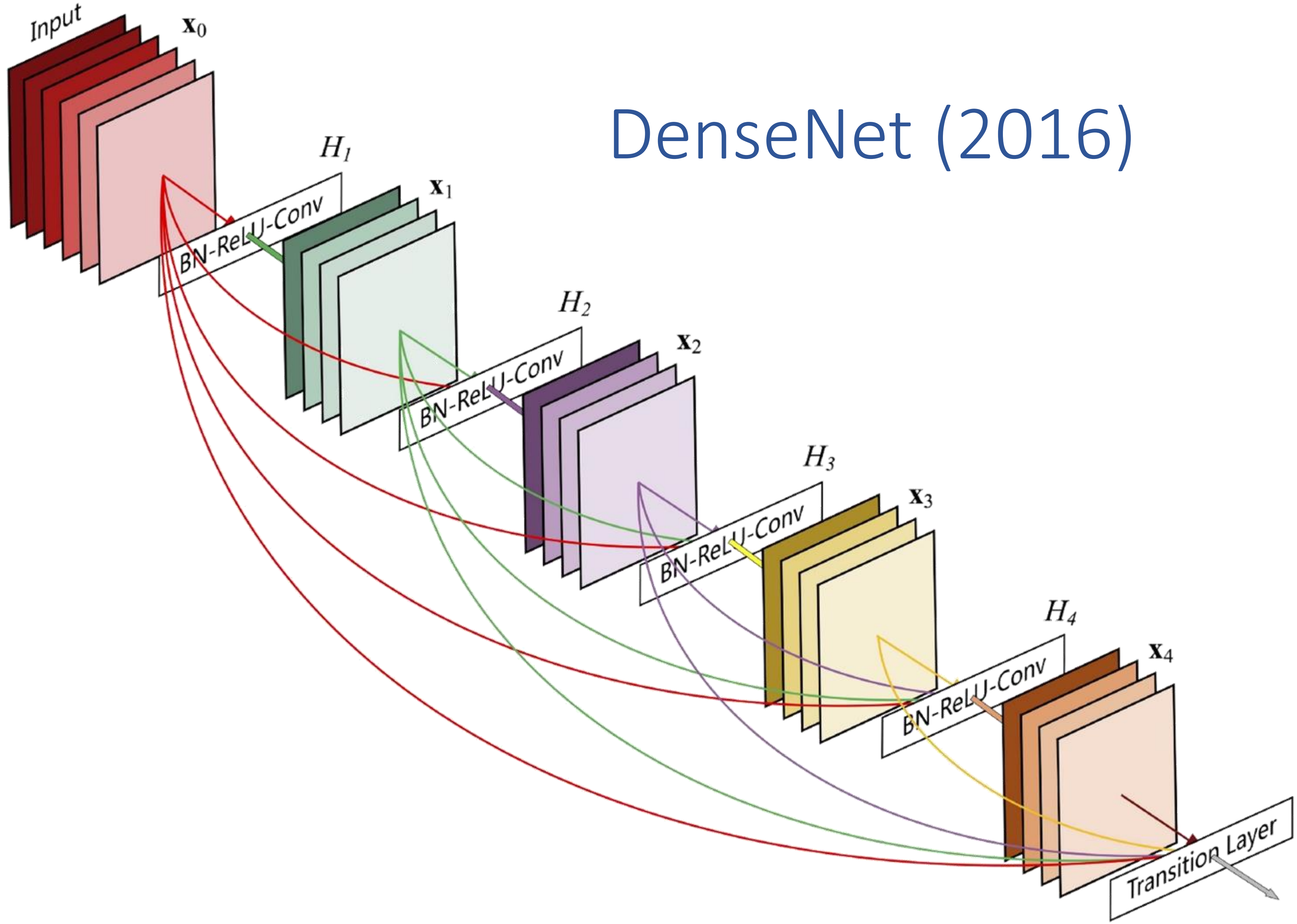


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Summary Table

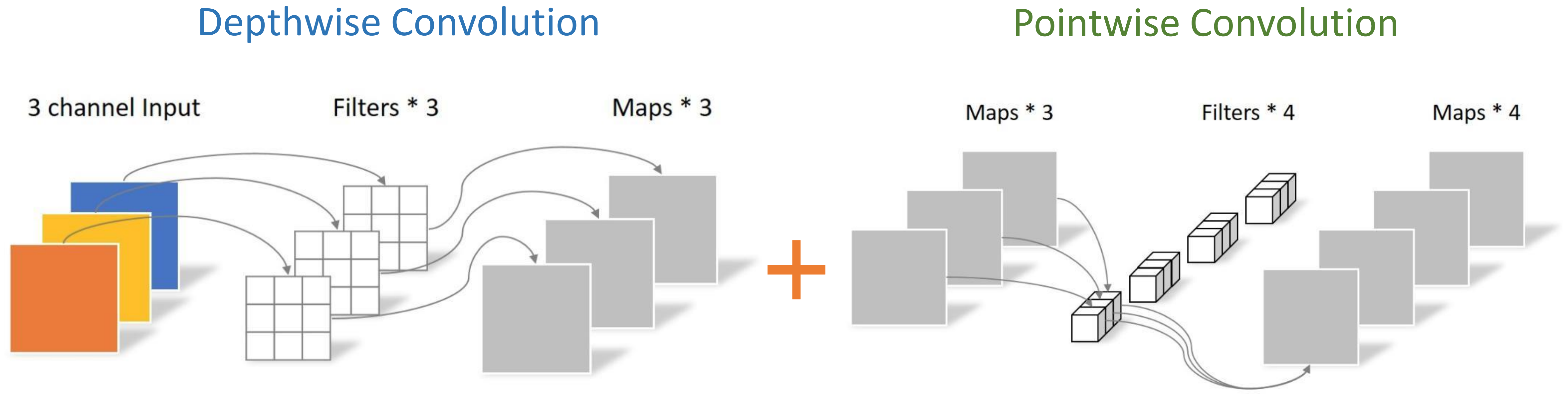
Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG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

DenseNet (2016)



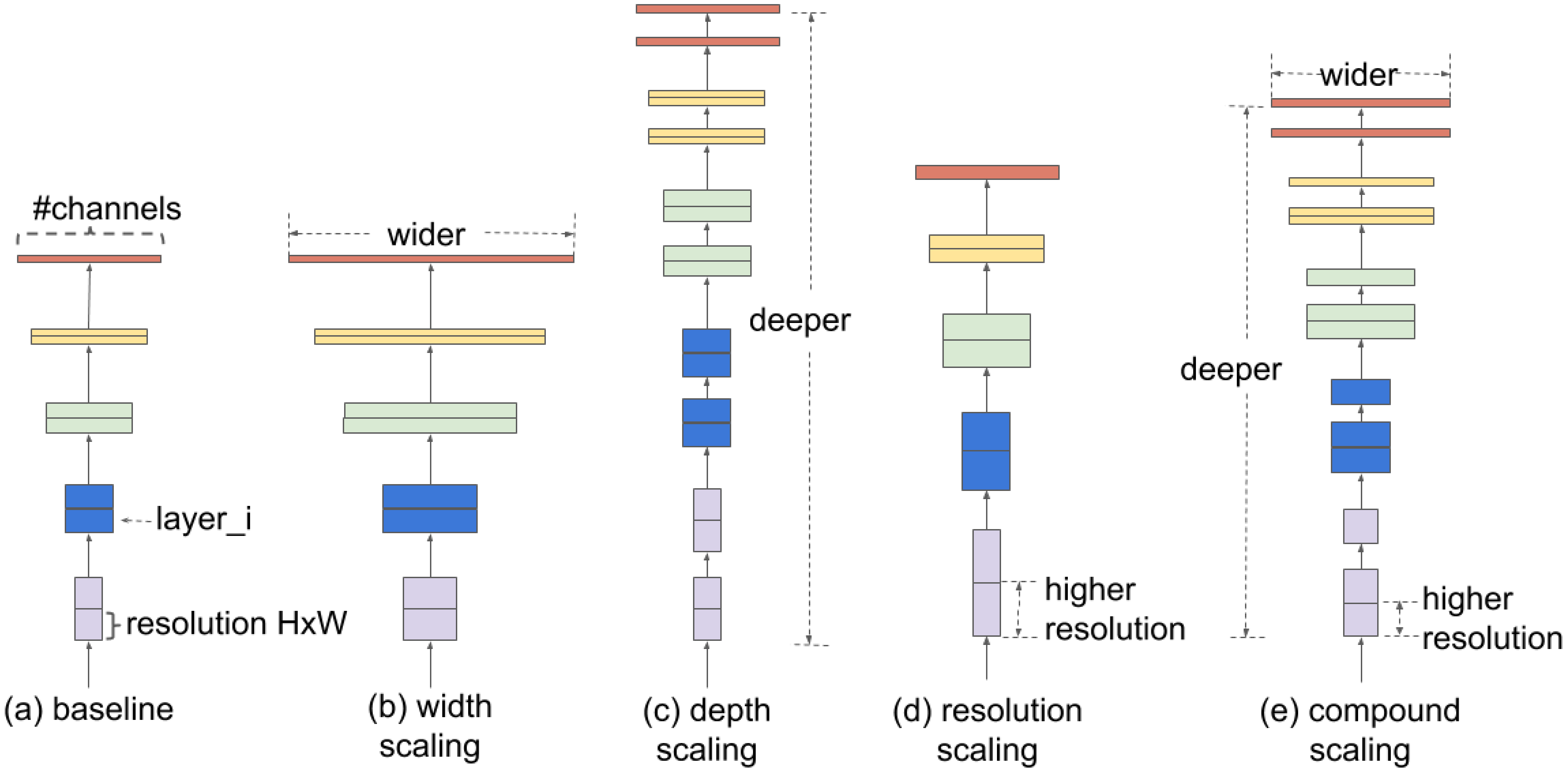
Xception - Separable Convolution (2017)

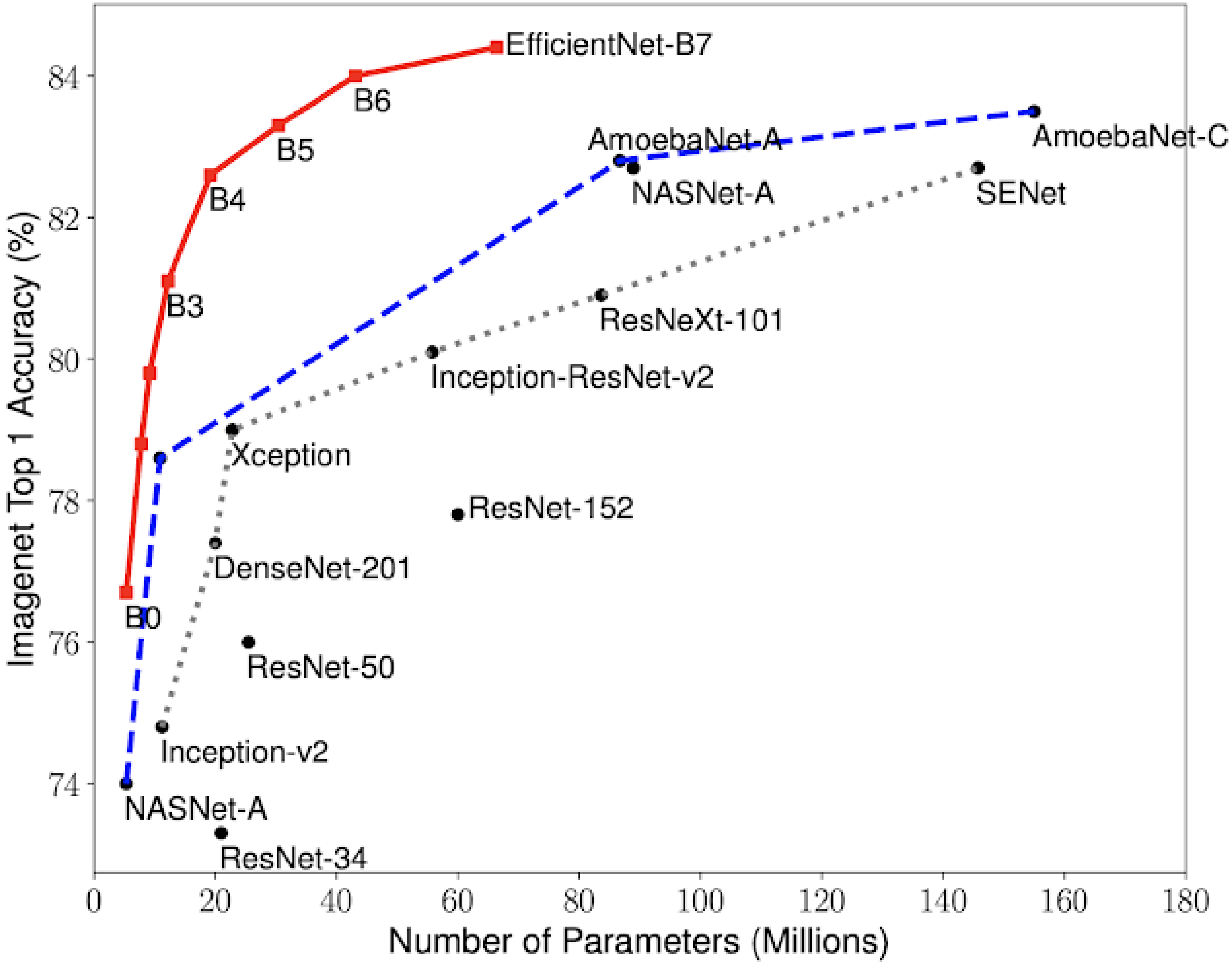
- Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *CVPR*, 2017
- Example



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

EfficientNet (May, 2019)




EfficientNet-B7
B6
B5
B4
B3
B0
AmoebaNet-A
AmoebaNet-C
NASNet-A
SENet
ResNeXt-101
Inception-ResNet-v2
Xception
ResNet-152
DenseNet-201
ResNet-50
Inception-v2
NASNet-A
ResNet-34
Imagenet Top 1 Accuracy (%)
Number of Parameters (Millions)
74
76
78
80
82
84
0
20
40
60
80
100
120
140
160
180

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