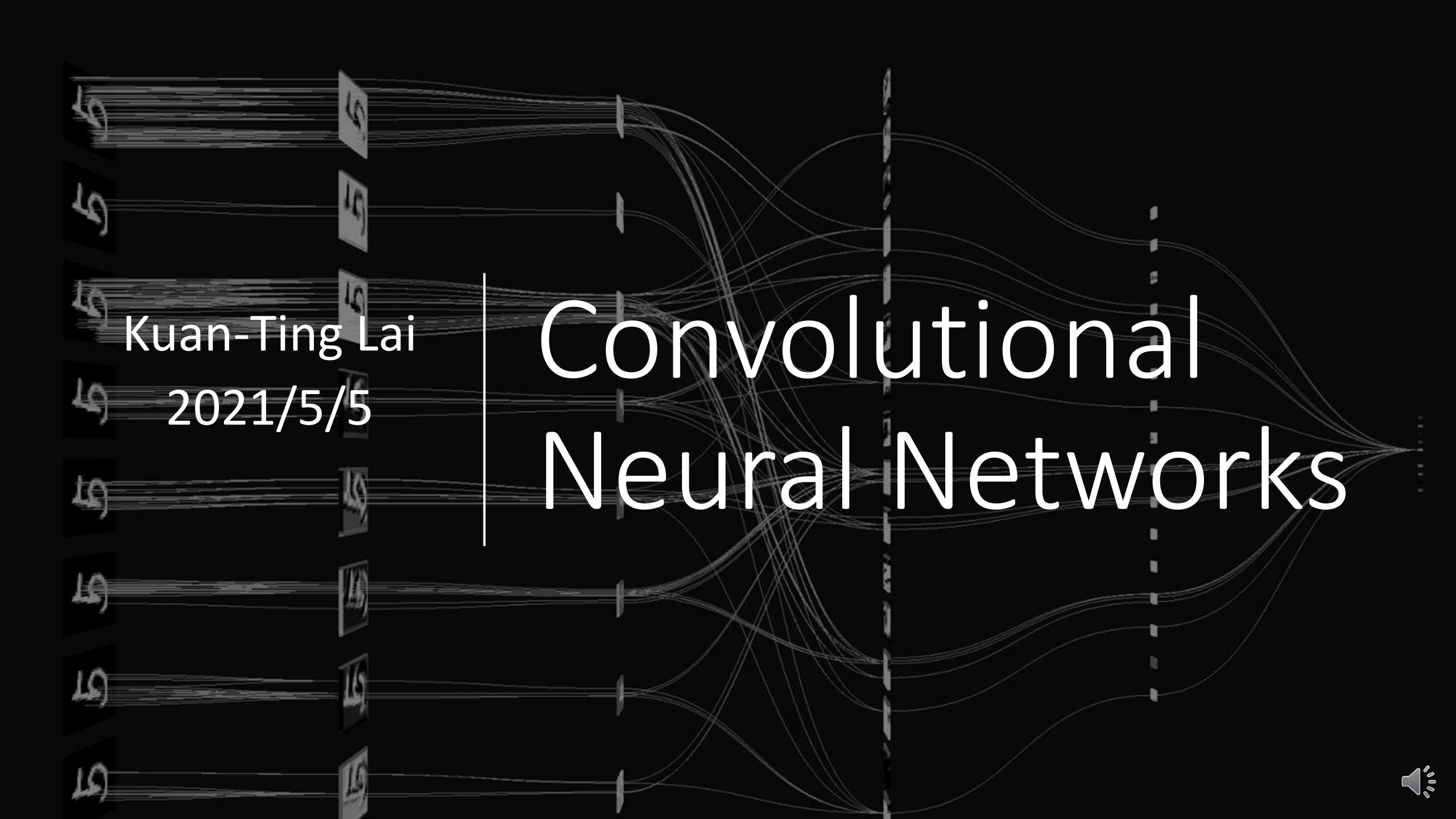
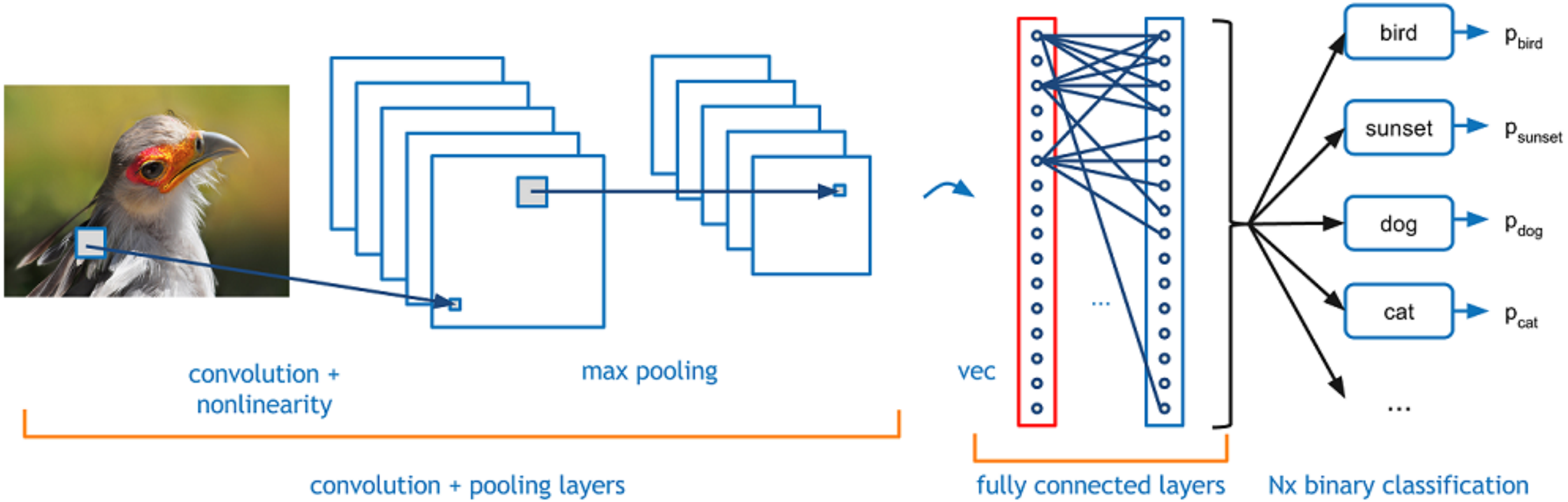


Kuan-Ting Lai
2021/5/5

Convolutional Neural Networks

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)
- Brightness of each pixel color: 0 - 255

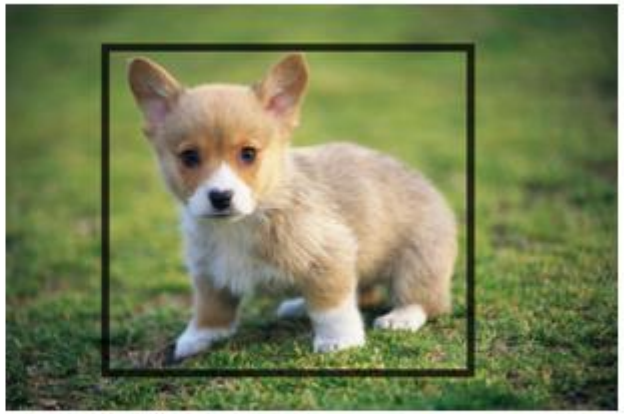
What We See

08 02 22 97 38 15 00 40 00 75 04 05 07 78 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 89 41 92 36 54 22 40 40 28 66 33 13 80
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 88 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 69 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

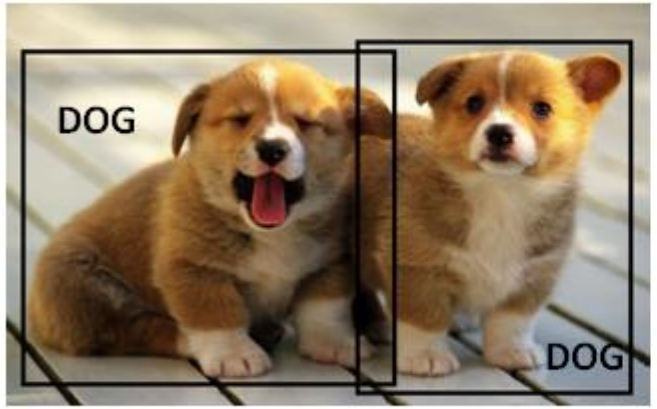
What Computers See

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.

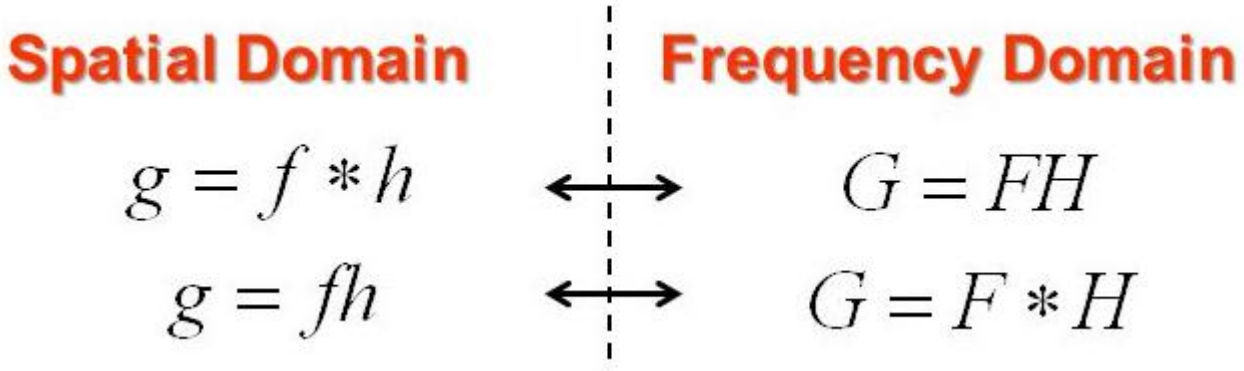


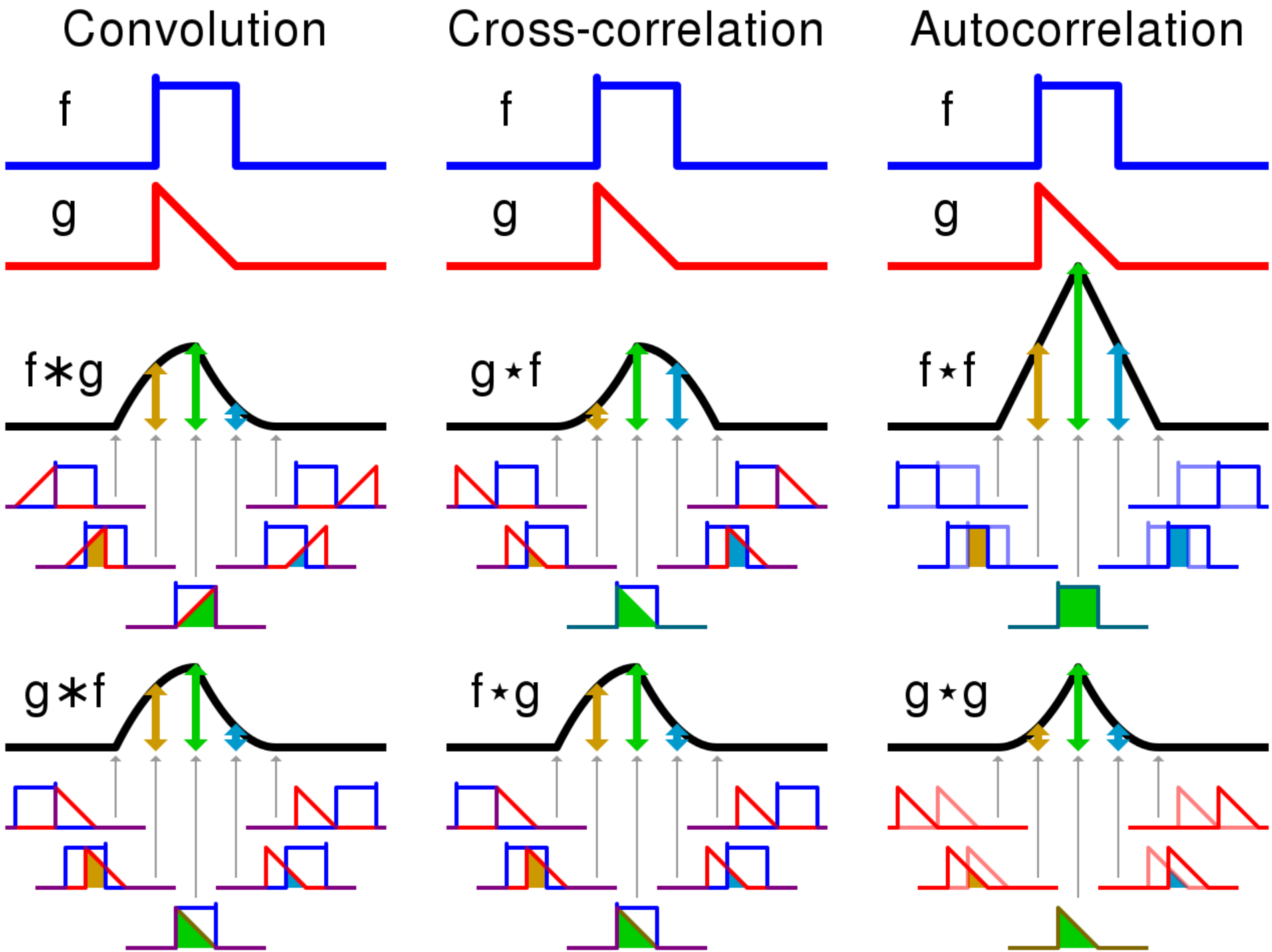
Object Detection involves localization of multiple objects (doesn't have to be the same class).

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



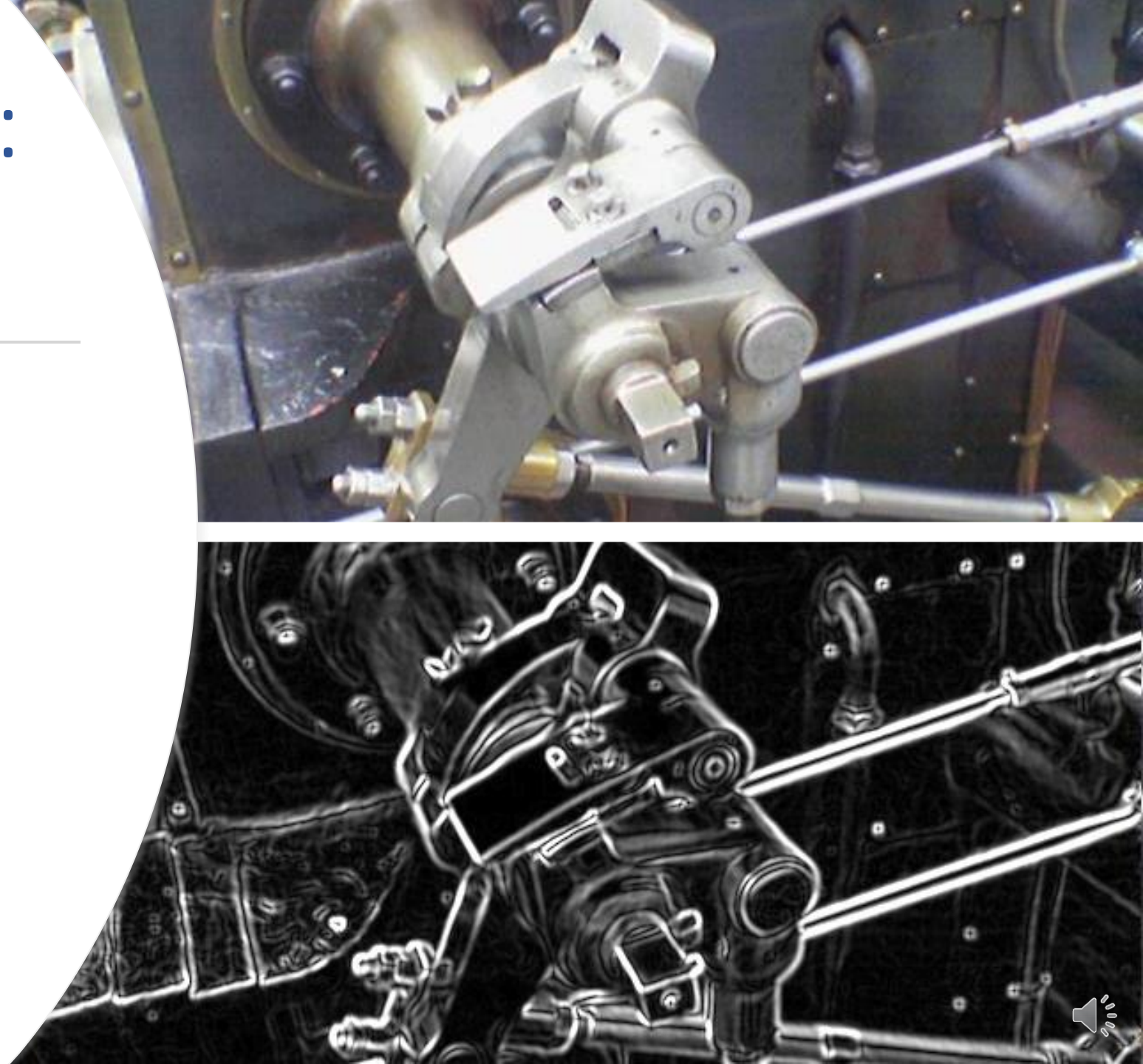

Convolution
Cross-correlation
Autocorrelation
f
g
f∗g
g⋆f
f⋆f
g∗f
f⋆g
g⋆g

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



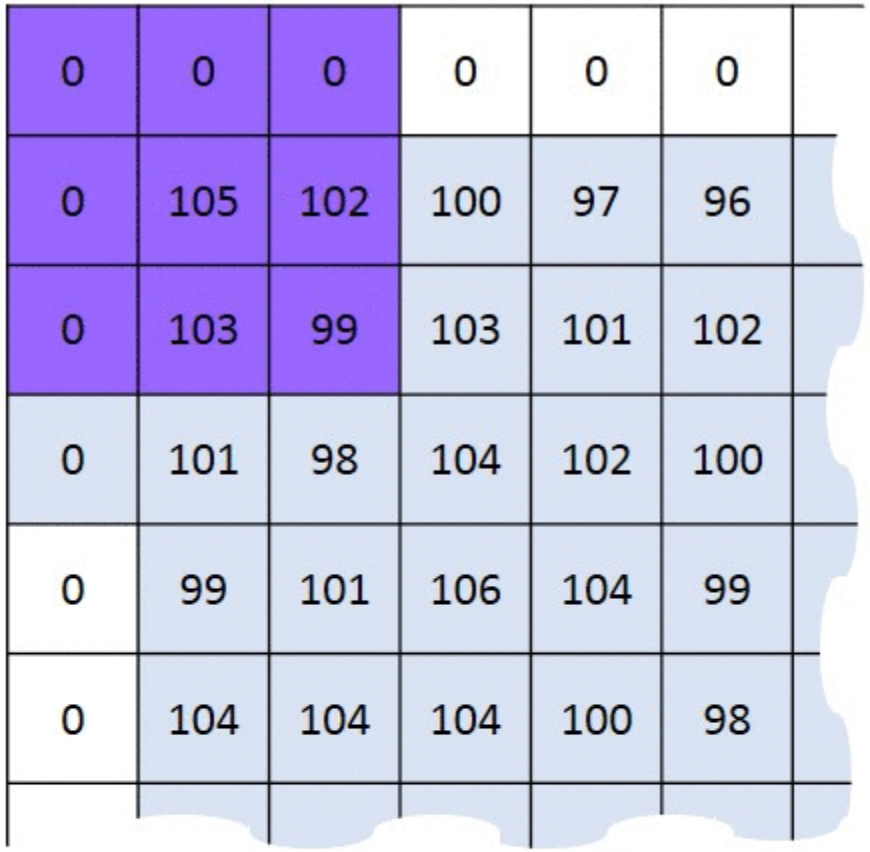
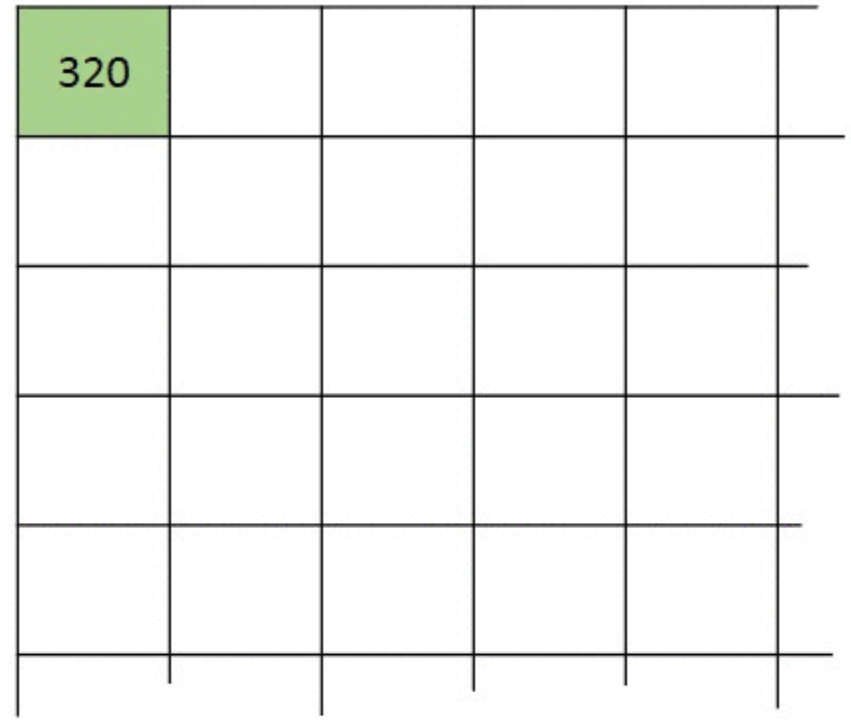


Image Matrix

Kernel Matrix

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

$$0*0+0*-1+0*0$$
$$+0*-1+105*5+102*-1$$
$$+0*0+103*-1+99*0 = 320$$



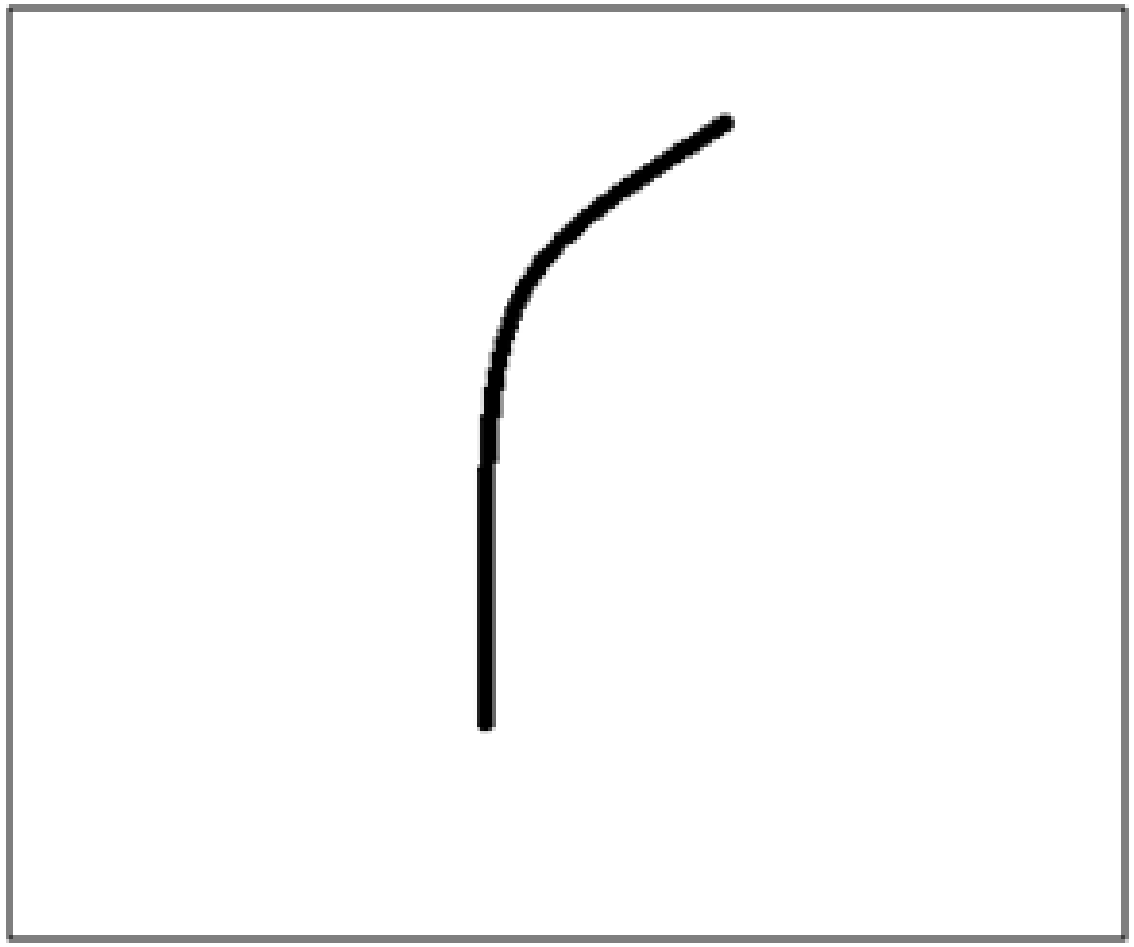
Output Matrix

Convolution with horizontal and vertical strides = 1

Example: A Curve Filter

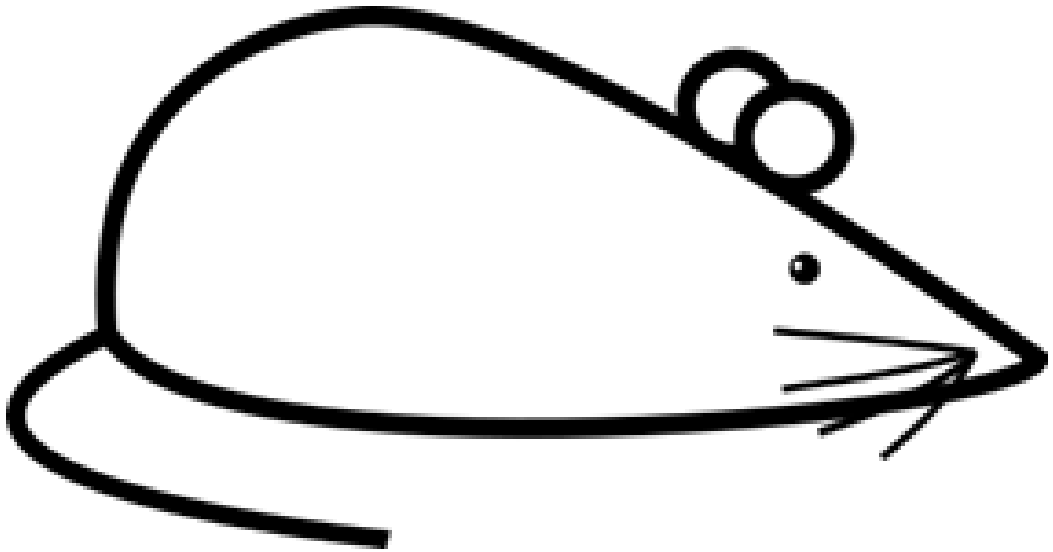
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

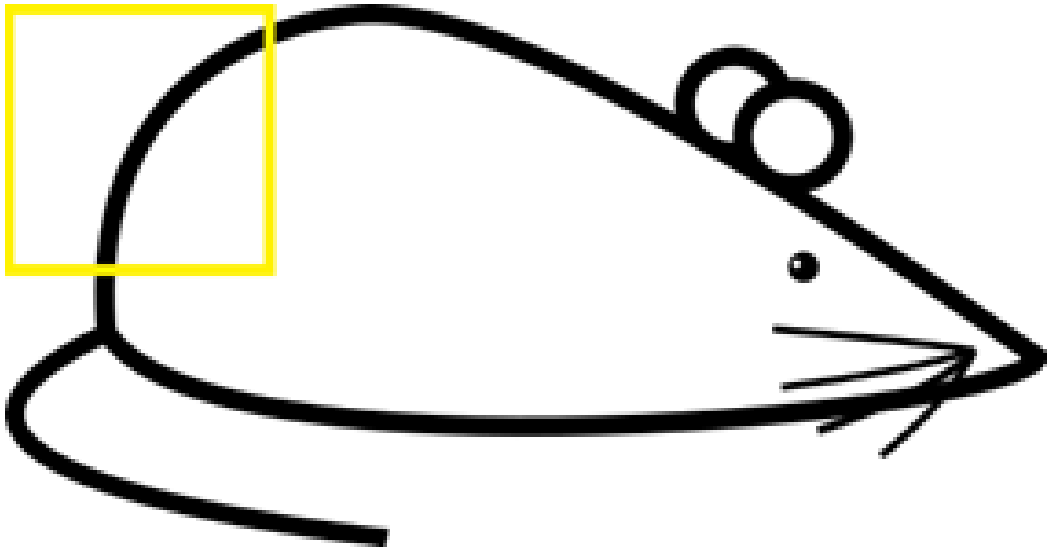


Visualization of a curve detector filter

Scan the Image to Detect an Edge

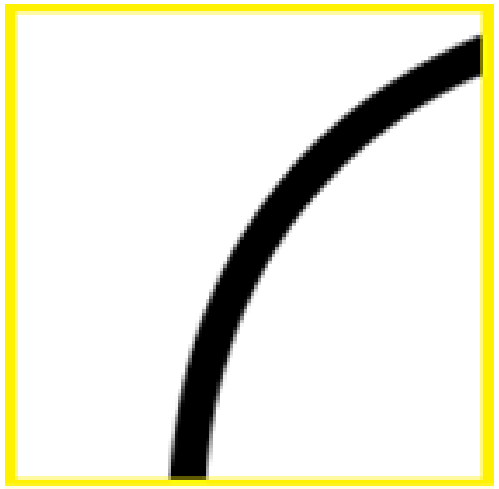


Original image



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

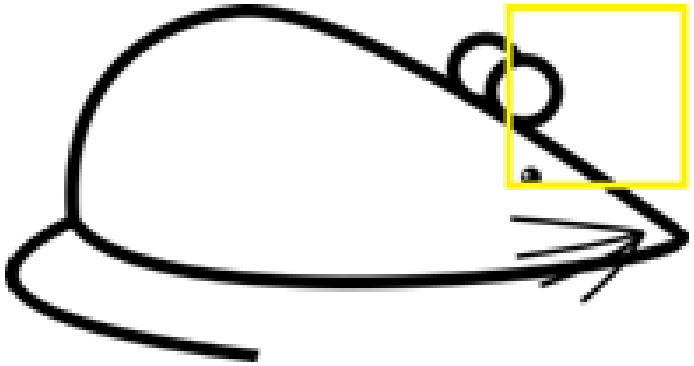
*

0	0	0	0	0	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

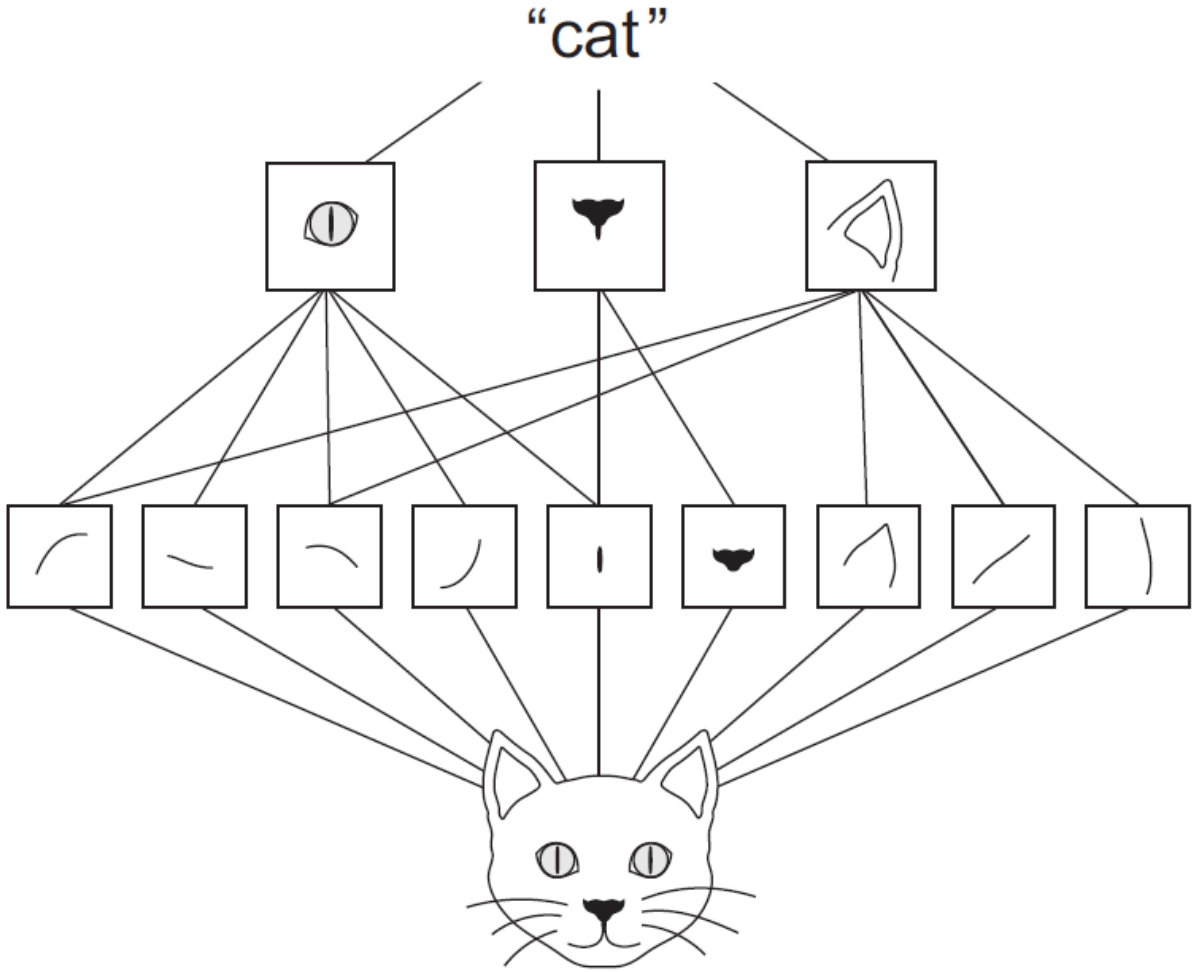
*

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()

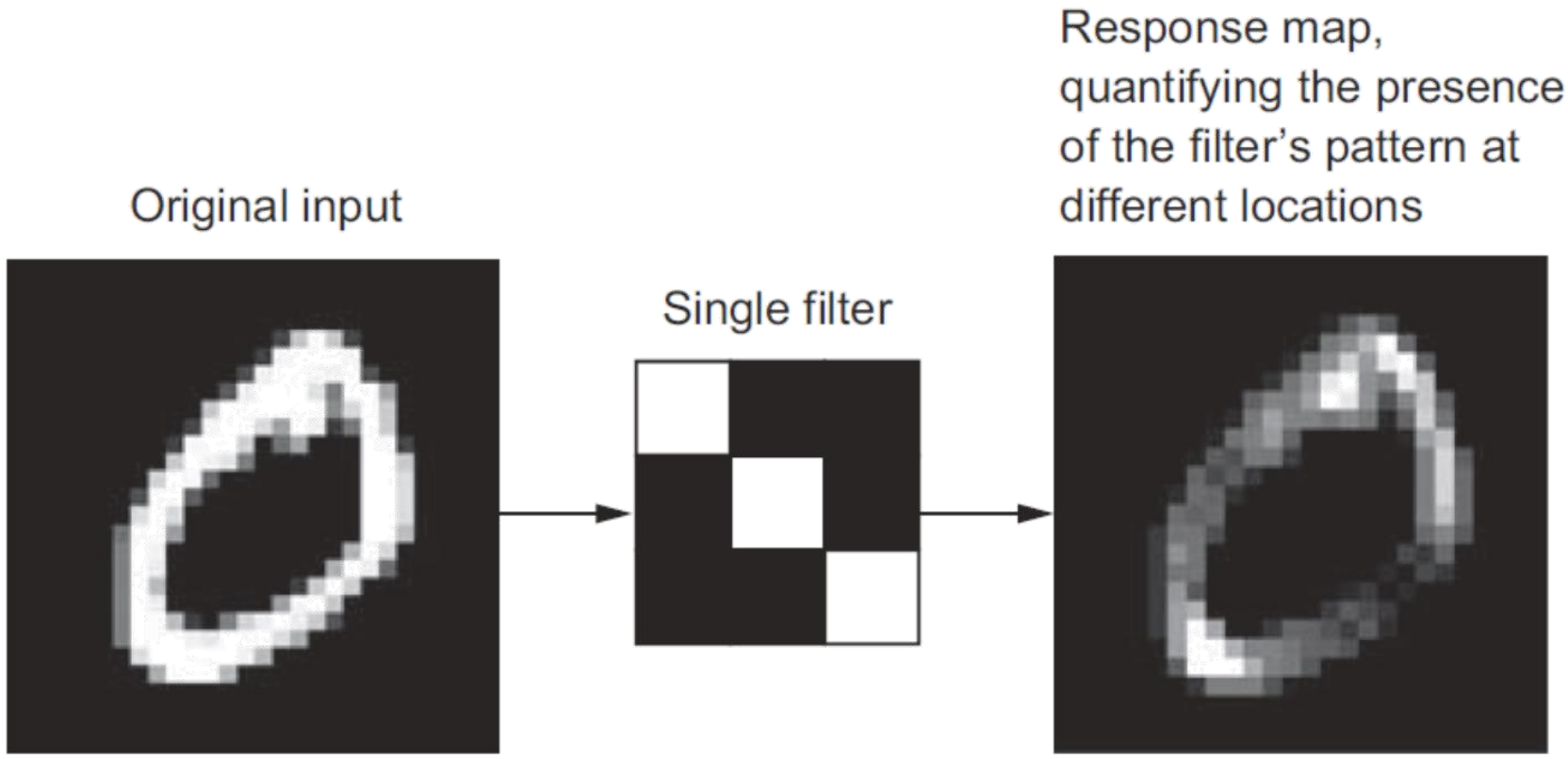
```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d_1 (Conv2D) (None, 26, 26, 32) 320 =(3x3+1)x32
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32) 0
_________________________________________________________________
conv2d_2 (Conv2D) (None, 11, 11, 64) 18496 =(3x3x32+1)x64
_________________________________________________________________
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

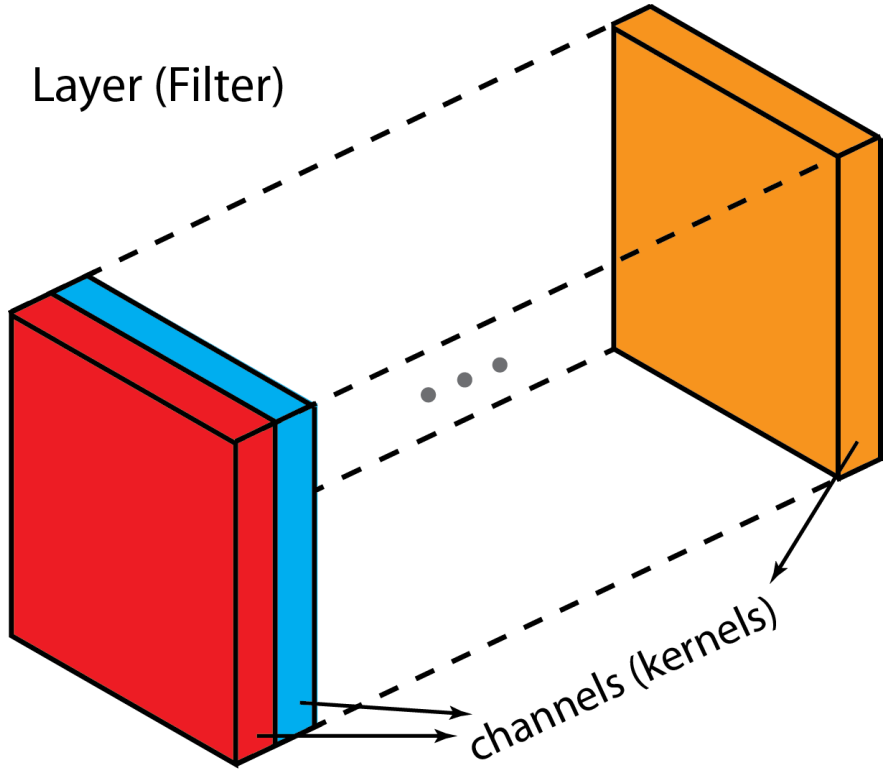
```
=>layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))
```

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



Kernel and Filter in ConvNets

- “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

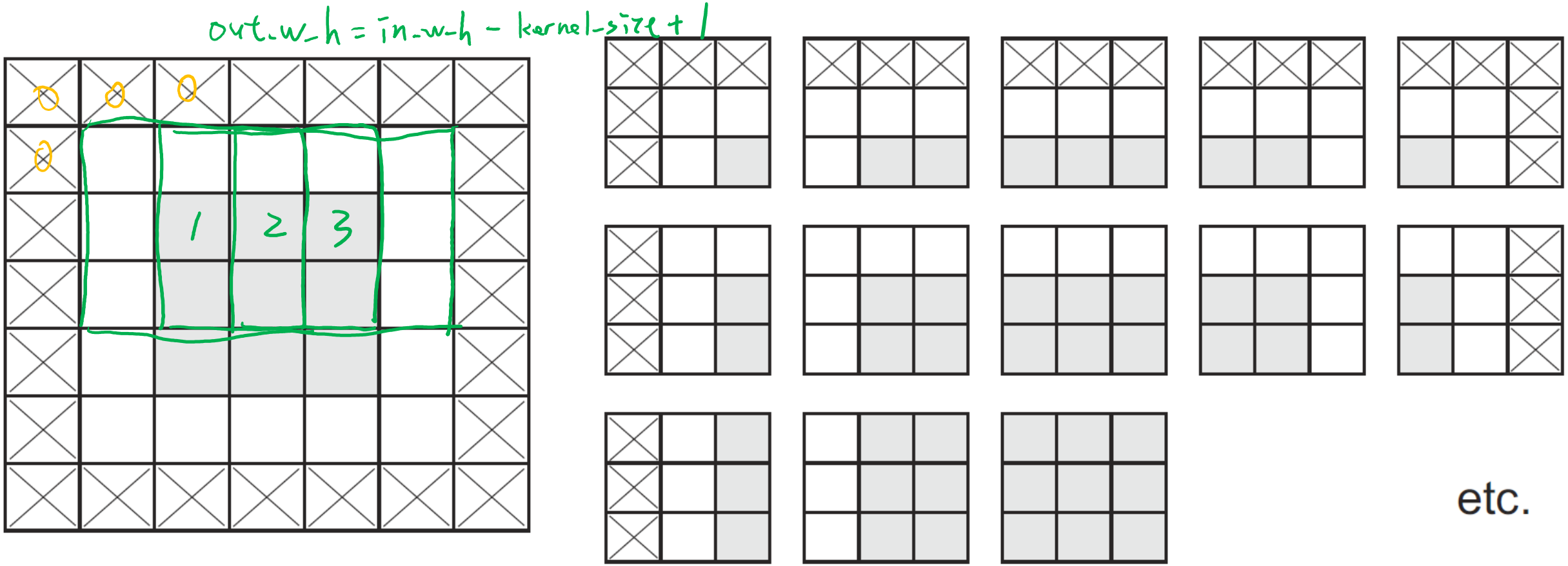
Add a Classifier on Top of ConvNet

```
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
```

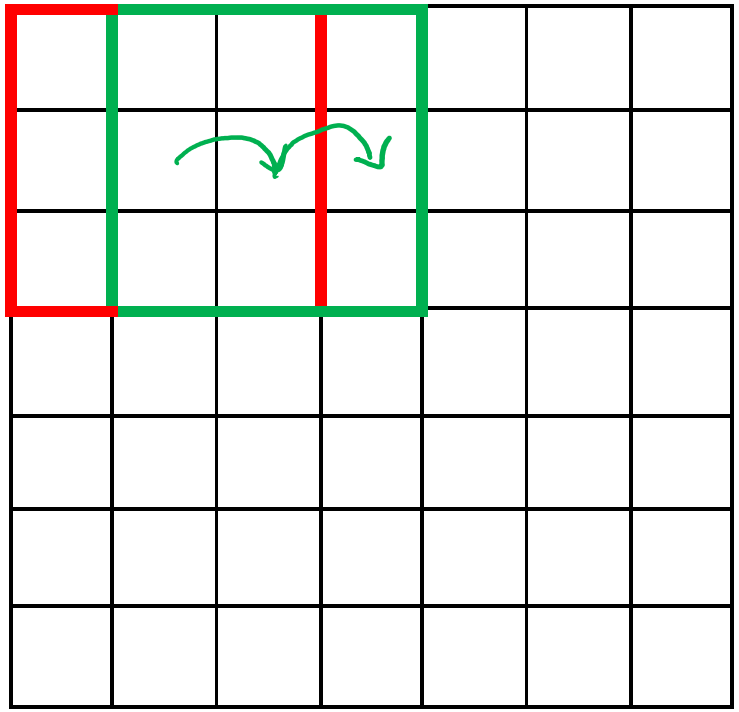
Zero Padding

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

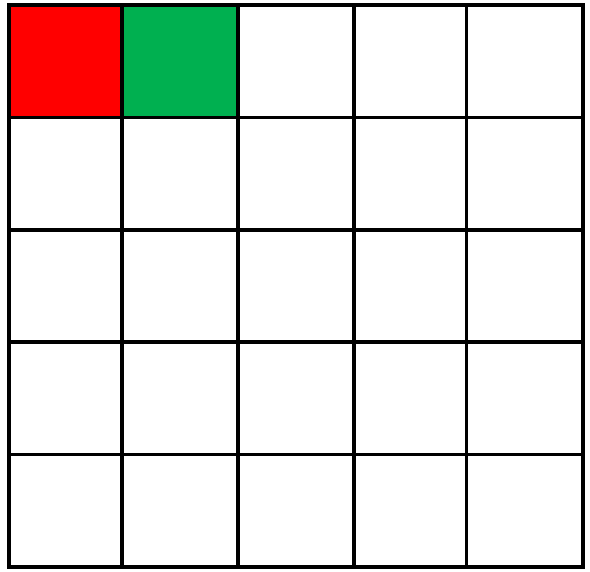


Stride=1

7 x 7 Input Volume

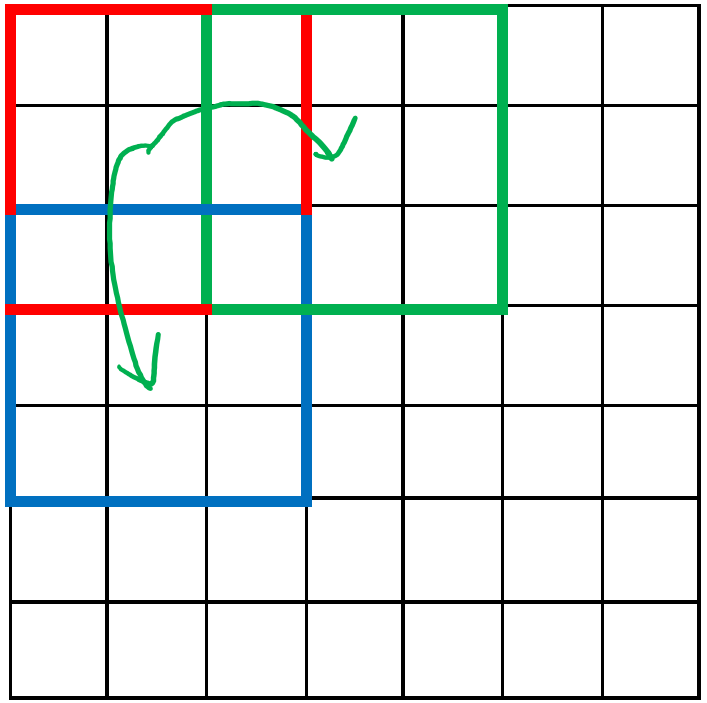


5 x 5 Output Volume

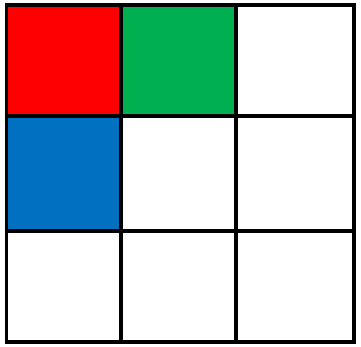


Stride=2

7 x 7 Input Volume

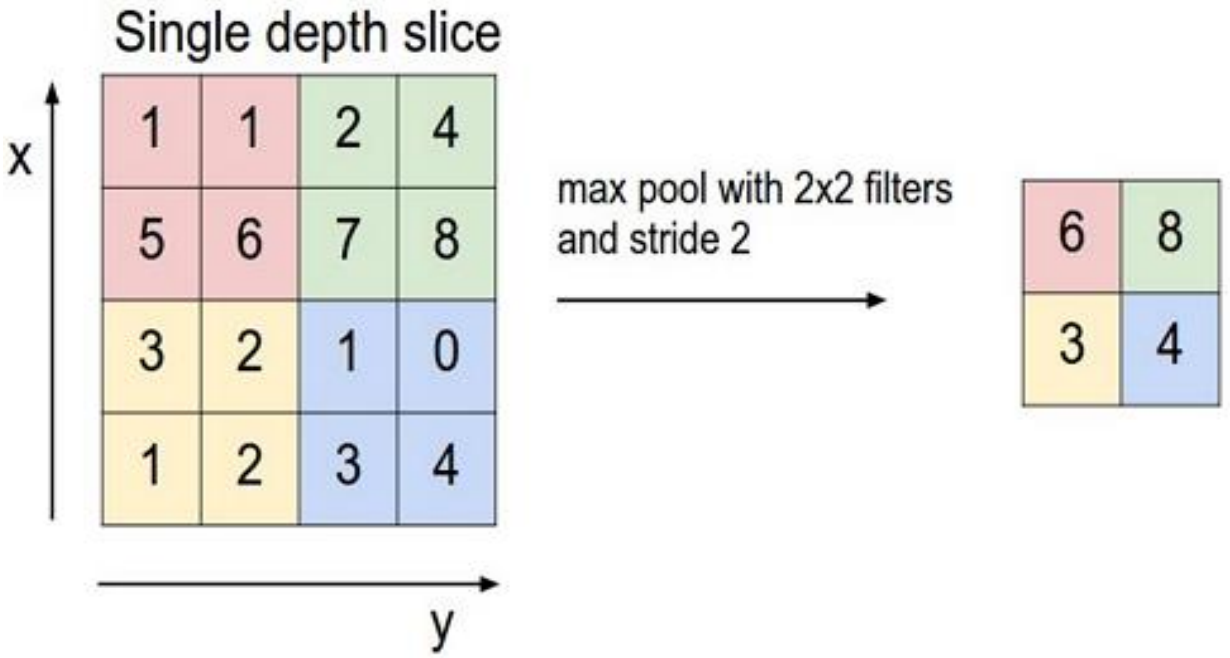


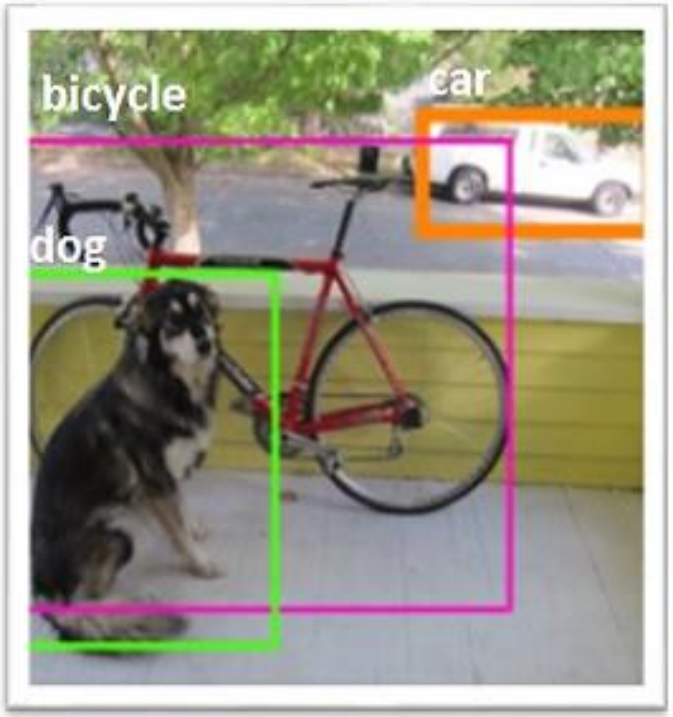
3 x 3 Output Volume

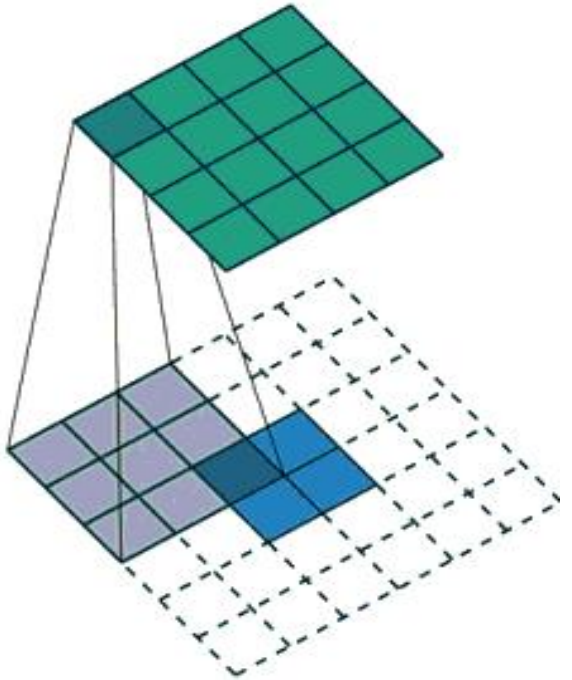


Max Pooling

- 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
```

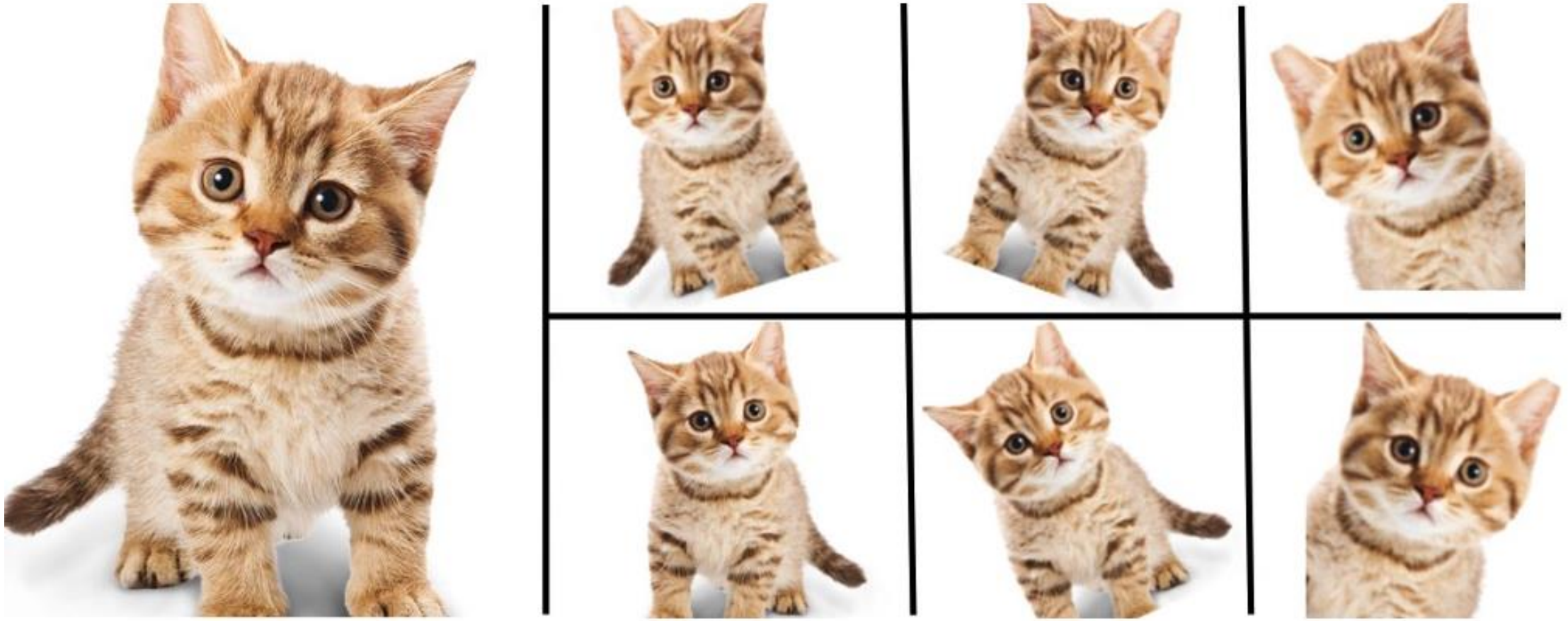
It prints this:

```
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



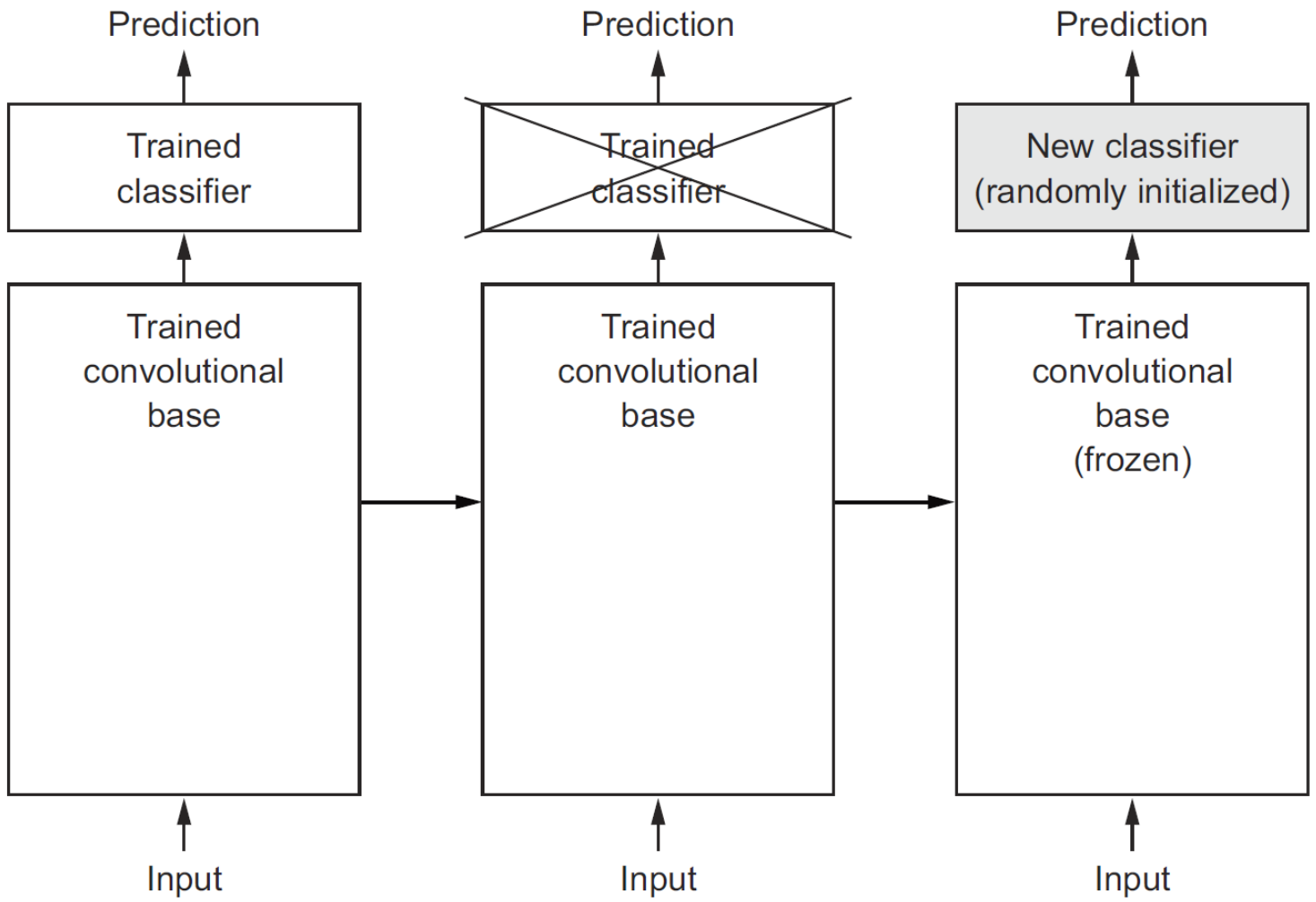
Data Augmentation via ImageDataGenerator

- 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
```

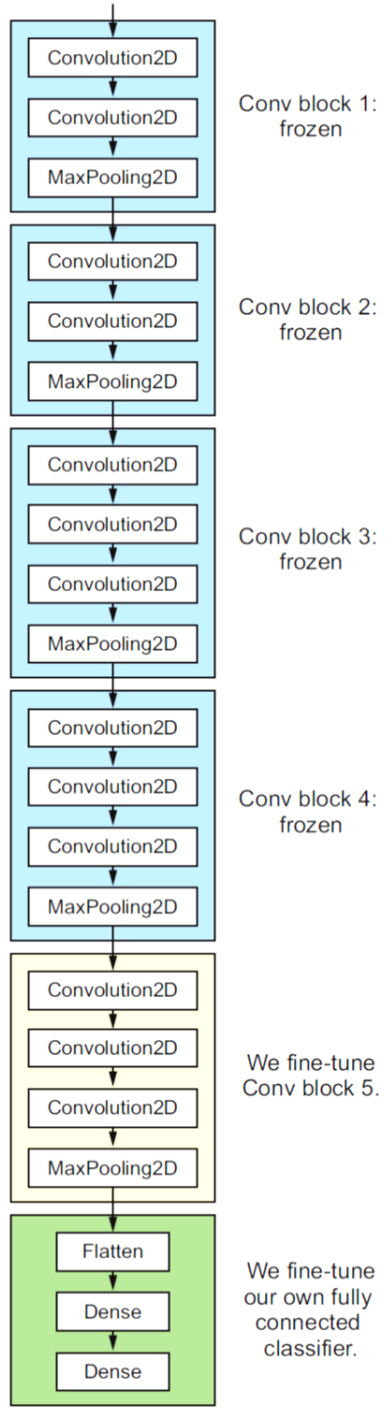
```
print('This is the number of trainable weights '
'after freezing the conv base:', len(model.trainable_weights))
```

```
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

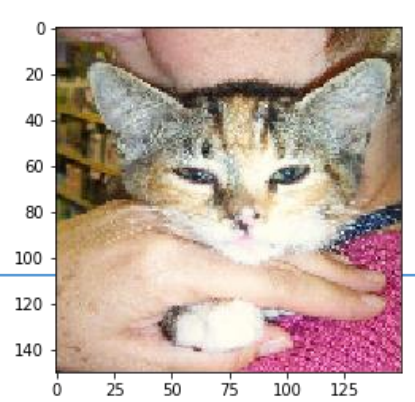
- ConvNets are the best for Computer Vision so far (Attention and Transformer may be better in the future)
- Data augmentation is a powerful way to fight overfitting
- We can use pre-trained model for feature extraction
- We can further improve the pre-trained model on our dataset by fine-tuning

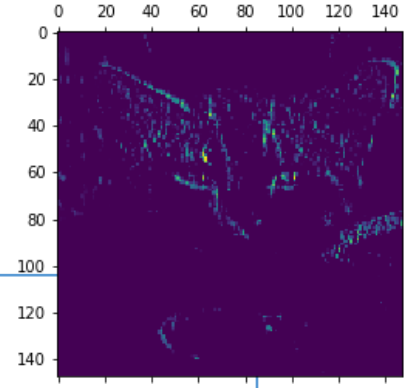
Visualizing What Convnets Learn

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



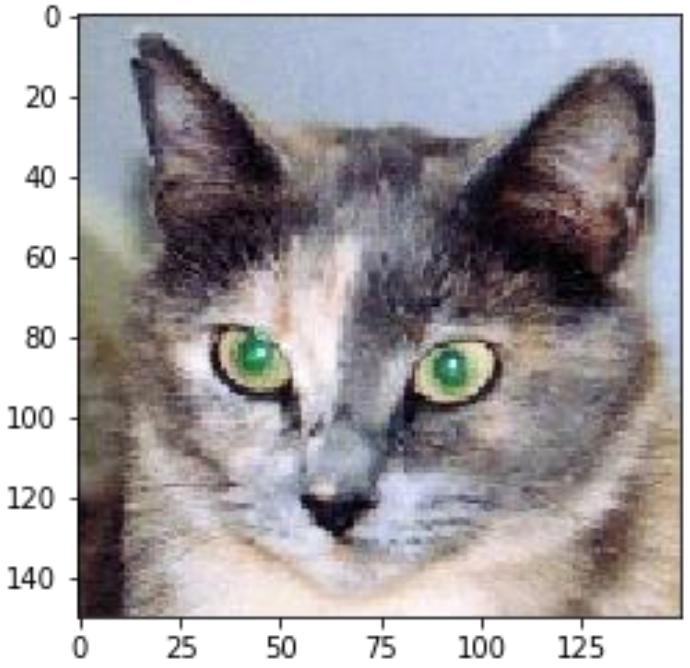



```
from keras.preprocessing import image
import numpy as np
img = image.load_img('./test1/1700.jpg', target_size=(150, 150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)/255.

from keras import models
model = load_model('cats_and_dogs_small_2.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')
```

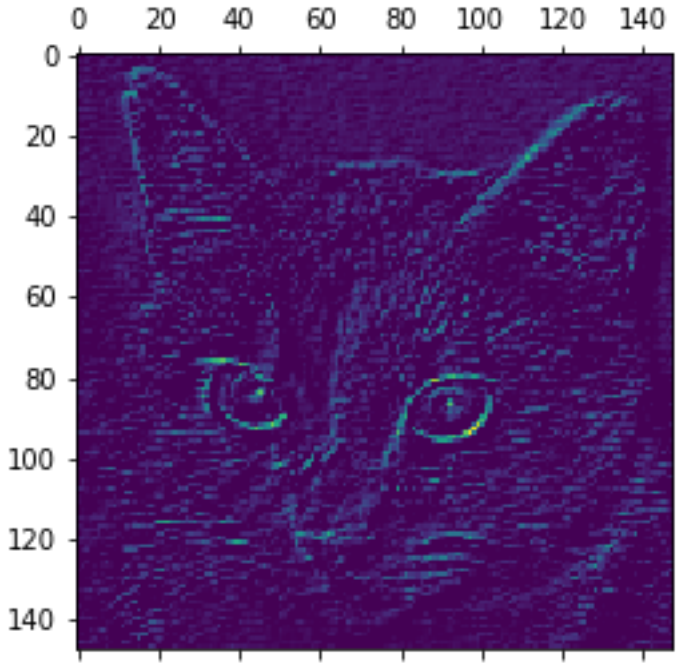
https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.4-visualizing-what-convnets-learn.ipynb

Activations of 1st Layer (Channel 3 & 30)

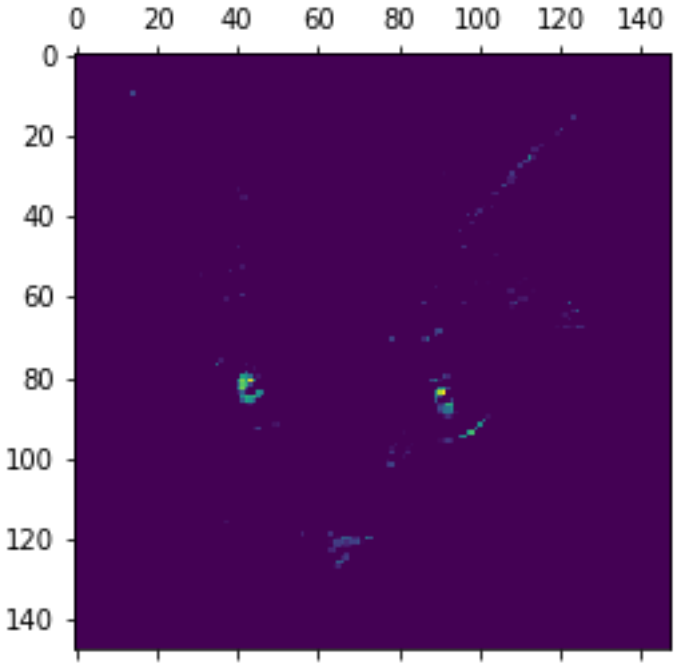
```
plt.imshow(img_tensor[0])
plt.show()
```




```
plt.matshow(first_layer_activation
[0, :, :, 3], cmap='viridis')
plt.show()
```




```
plt.matshow(first_layer_activation
[0, :, :, 30], cmap='viridis')
plt.show()
```



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

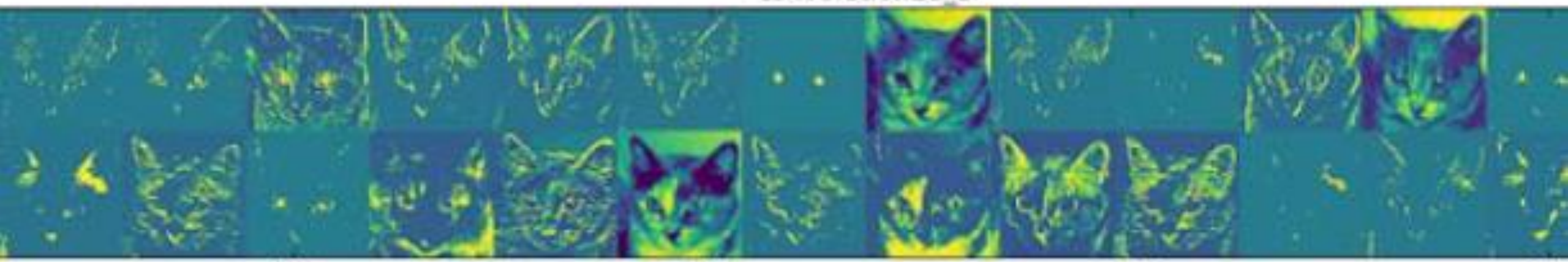
Model Architecture

- summary() of the model that will be visualized

```
from keras.models import load_model

model = load_model('cats_and_dogs_small_2.h5')
model.summary()  # As a reminder.
```

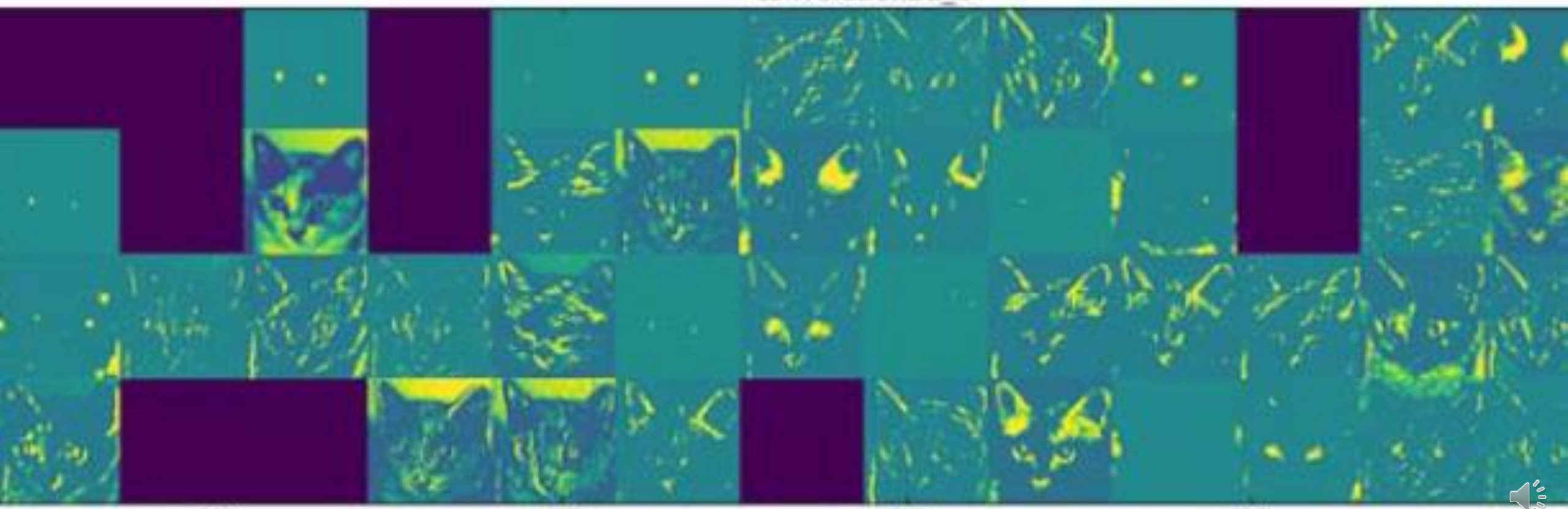
Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_5 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_6 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_6 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_7 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_7 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_8 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_8 (MaxPooling2	(None, 7, 7, 128)	0
flatten_2 (Flatten)	(None, 6272)	0
dropout_1 (Dropout)	(None, 6272)	0
dense_3 (Dense)	(None, 512)	3211776

convolution2d_5

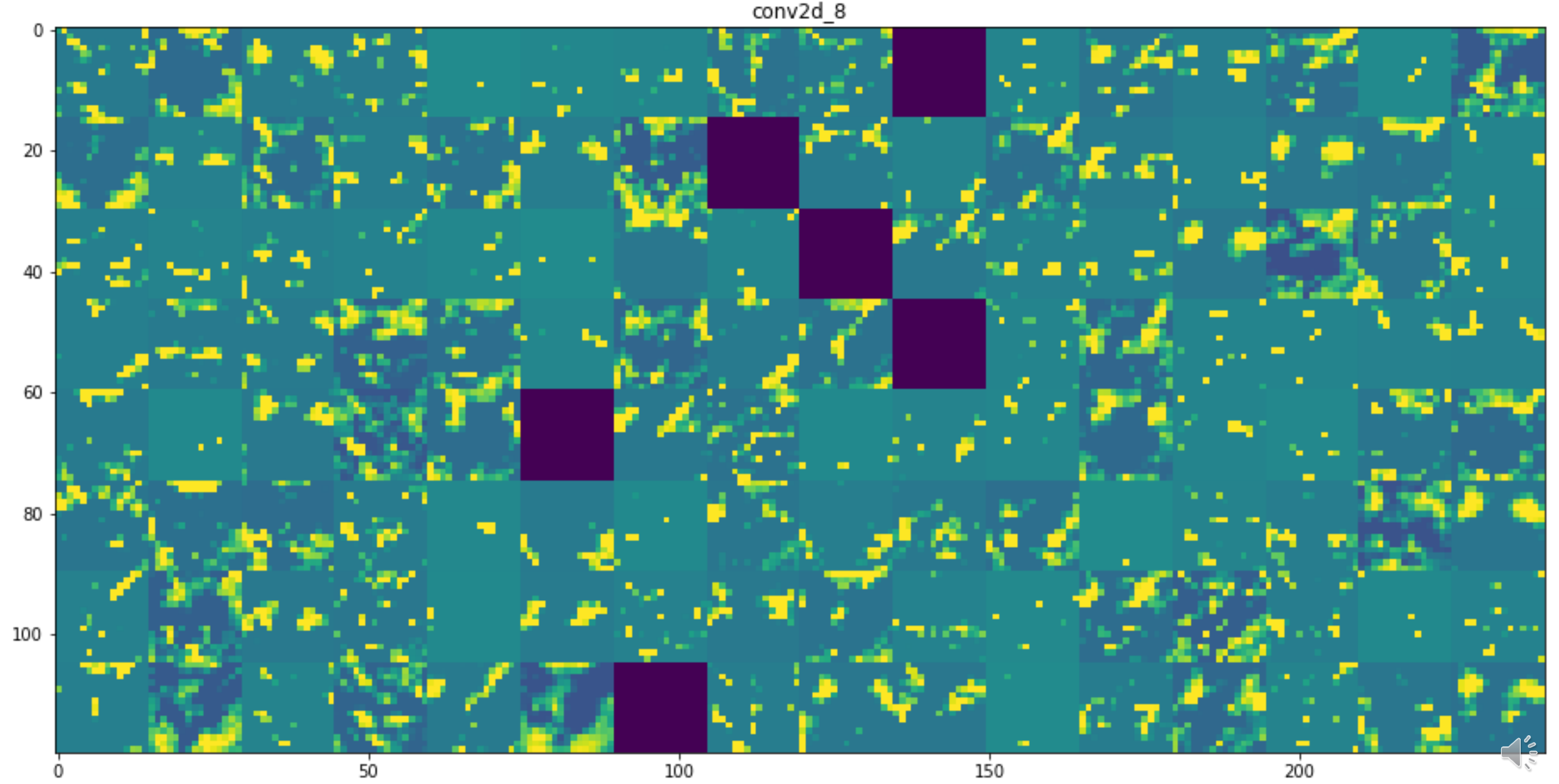


500 1000 1500 200

convolution2d_6

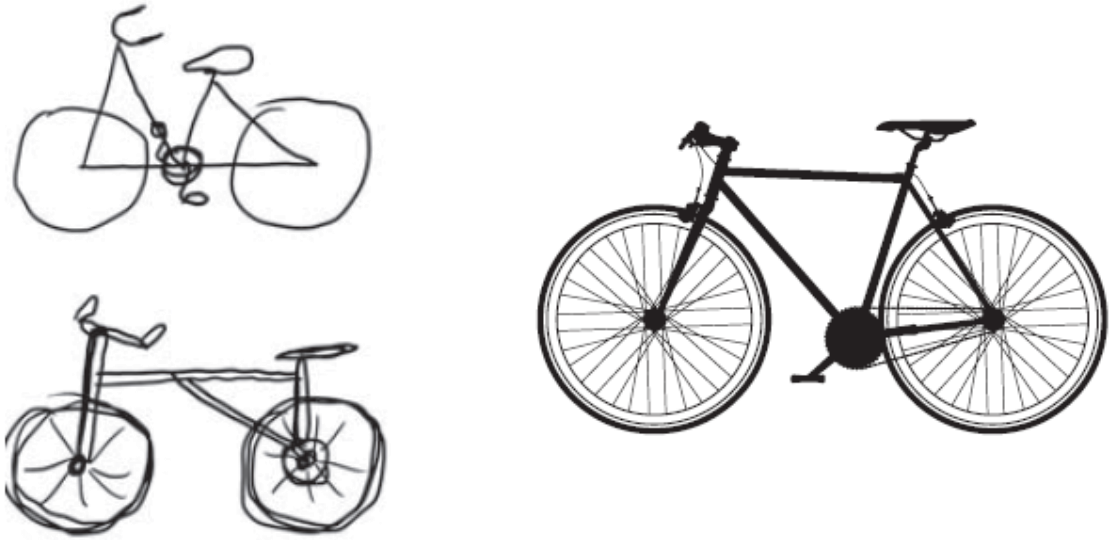


200 400 600 800


conv2d_8
0
20
40
60
80
100
0
50
100
150
200

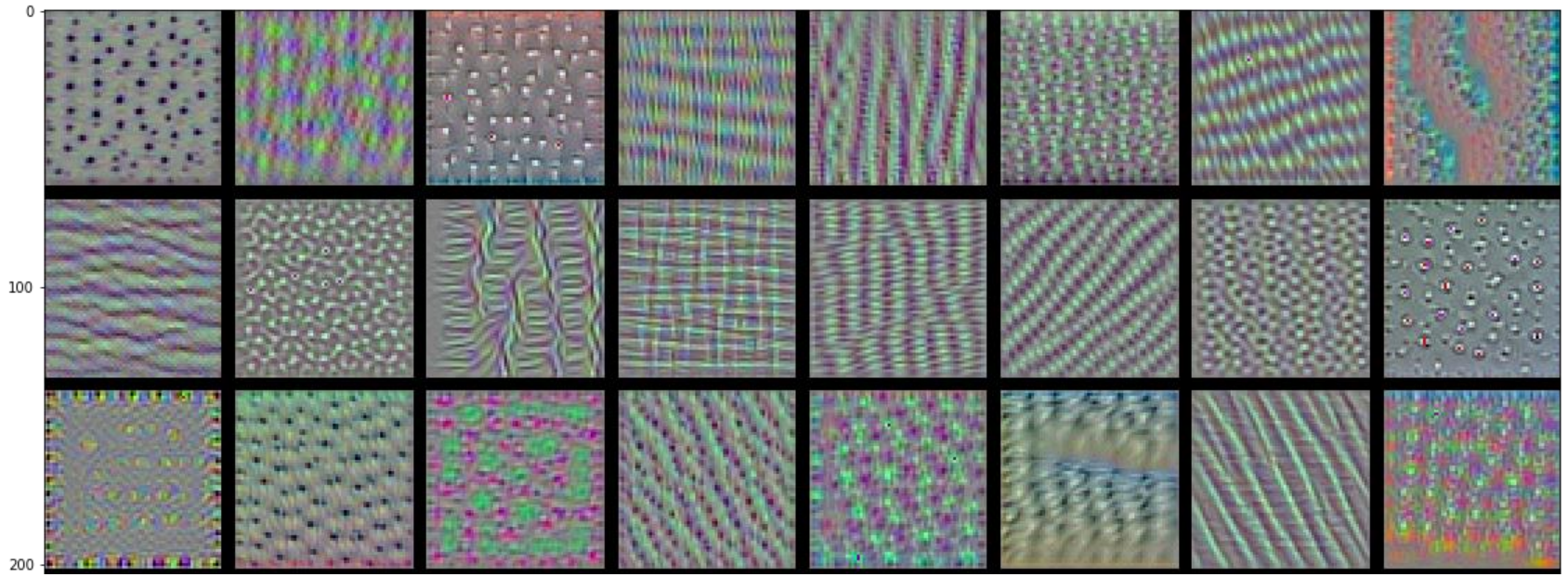
Things to Note

- 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

- Show the visual pattern that each filter is meant to respond to
  - Start with grayscale input image
  - Use gradient ascent to maximize filter responses



Visualizing Filter Patterns in VGG16

```
from keras.applications import VGG16
from keras import backend as K

model = VGG16(weights='imagenet',include_top=False)

layer_name = 'block3_conv1'
filter_index = 0

layer_output = model.get_layer(layer_name).output
loss = K.mean(layer_output[:, :, :, filter_index])

# The call to `gradients` returns a list of tensors (of size 1 in this case)
# hence we only keep the first element -- which is a tensor.
grads = K.gradients(loss, model.input)[0]

# We add 1e-5 before dividing so as to avoid accidentally dividing by 0.
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
```

Compute Loss Tensor and Gradient Tensor

- Use a Keras backend function iterate()
  - Iterate() takes a Numpy tensor (as a list of tensors of size 1) and returns a list of two Numpy tensors: the loss value and the gradient value.

```
iterate = K.function([model.input], [loss, grads])

# Let's test it:
import numpy as np
loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])
```

Compute Stochastic Gradient Ascent in a Loop

```
# We start from a gray image with some noise
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.

# Run gradient ascent for 40 steps
step = 1.  # this is the magnitude of each gradient update
for i in range(40):
    # Compute the loss value and gradient value
    loss_value, grads_value = iterate([input_img_data])

    # Here we adjust the input image in the direction that maximizes the loss
    input_img_data += grads_value * step
```

Convert a Tensor into a Valid Image

```
def deprocess_image(x):
    # normalize tensor: center on 0., ensure std is 0.1
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

    # clip to [0, 1]
    x += 0.5
    x = np.clip(x, 0, 1)

    # convert to RGB array
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

def generate_pattern():

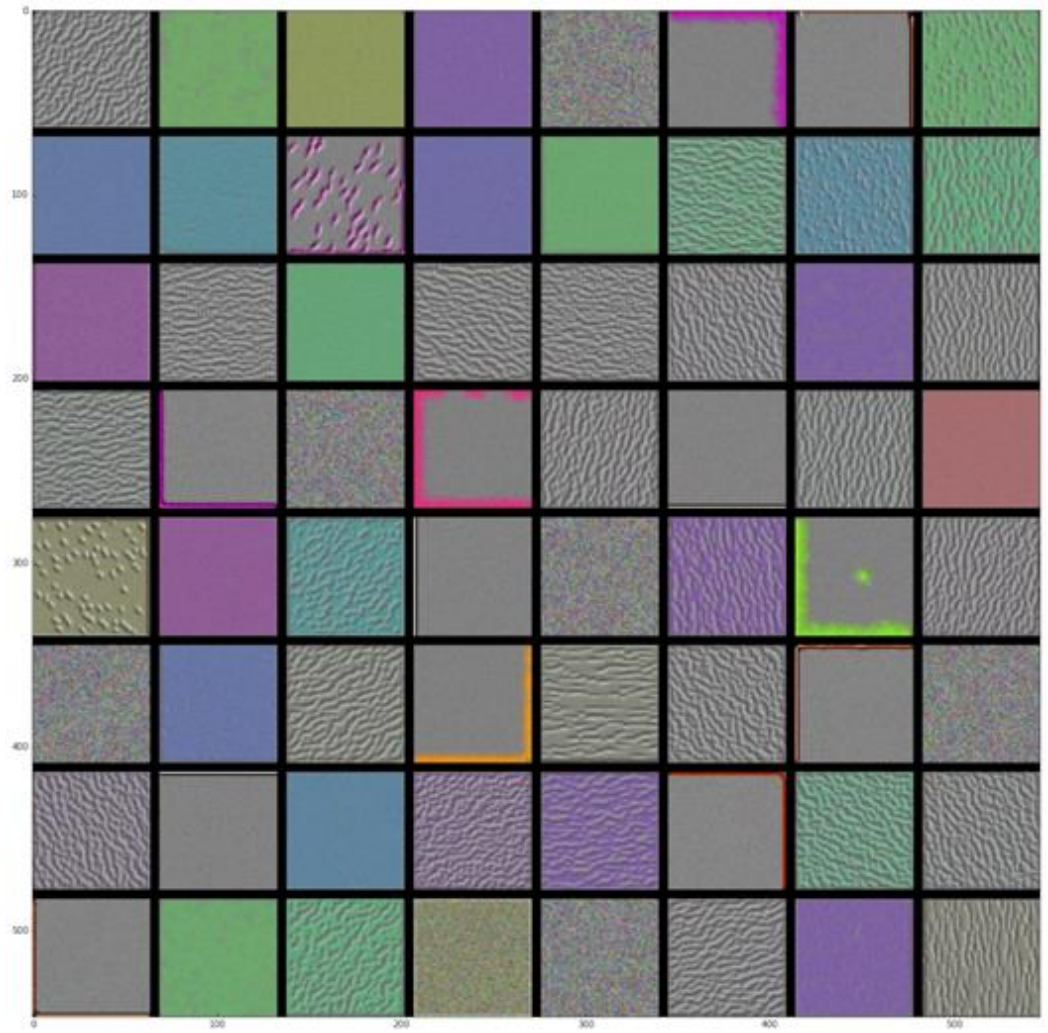
```
def generate_pattern(layer_name, filter_index, size=150):
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])
    grads = K.gradients(loss, model.input)[0] # Keep only the first tensor
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5) # 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)
```

Visualizing 4 conv-layers in VGG16

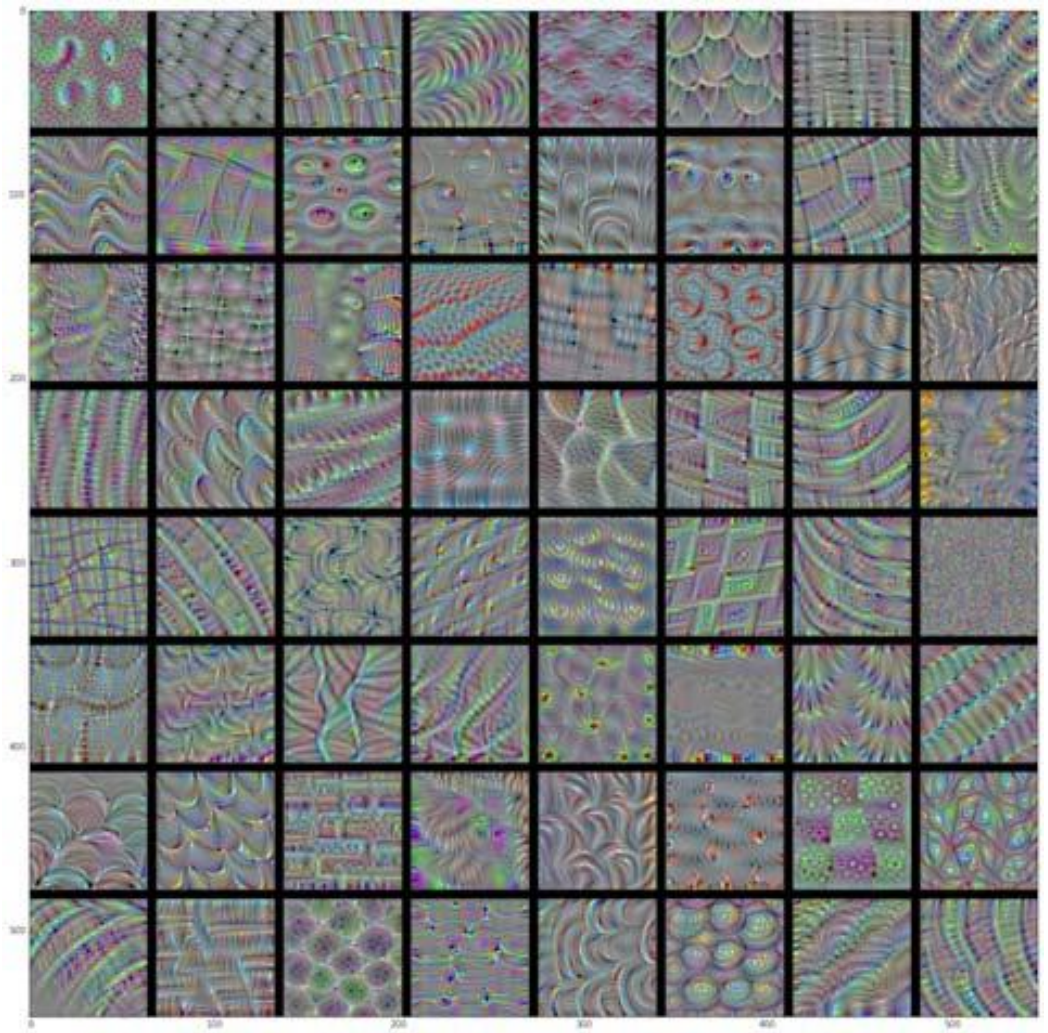
```
for layer_name in ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1']:
    size = 64
    margin = 5

    # This a empty (black) image where we will store our results.
    results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3))
    for i in range(8):  # iterate over the rows of our results grid
        for j in range(8):  # iterate over the columns of our results grid
            # Generate the pattern for filter `i + (j * 8)` in `layer_name`
            filter_img = generate_pattern(layer_name, i + (j * 8), size=size)

            # Put the result in the square `(i, j)` of the results grid
            horizontal_start = i * size + i * margin
            horizontal_end = horizontal_start + size
            vertical_start = j * size + j * margin
            vertical_end = vertical_start + size
            results[horizontal_start: horizontal_end, vertical_start: vertical_end, :] = filter_img

    # Display the results grid
    plt.figure(figsize=(20, 20))
    plt.imshow(results)
    plt.show()
```

Filter Patterns of Layers in VGG

block1_conv1



block4_conv1



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.

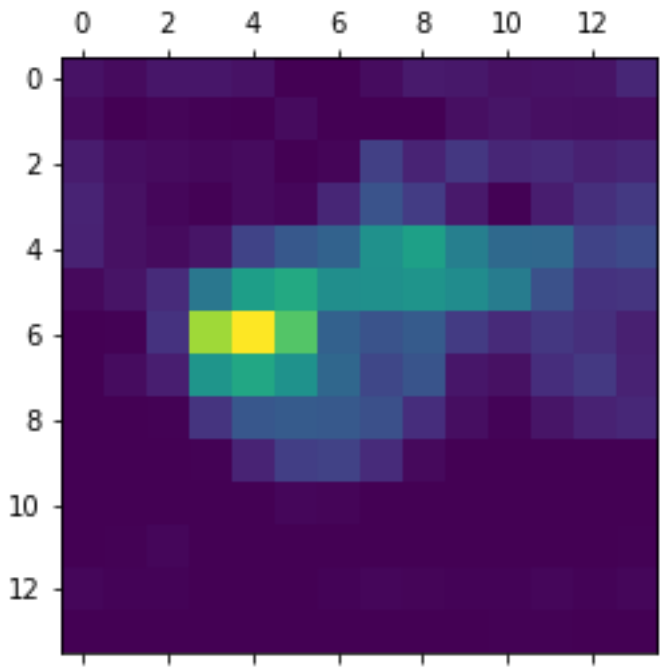
https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.4-visualizing-what-convnets-learn.ipynb

```
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16, preprocess_input, decode_predictions
import numpy as np

model = VGG16(weights='imagenet')

# The local path to our target image
img_path = '/Users/fchollet/Downloads/creative_commons_elephant.jpg'

img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)

# We add a dimension to transform our array into a "batch" of size (1, 224, 224, 3)
x = np.expand_dims(x, axis=0)

# Finally we preprocess the batch(this does channel-wise color normalization)
x = preprocess_input(x)

preds = model.predict(x)
print('Predicted:', decode_predictions(preds, top=2)[0])
# Predicted: [('n02504458','African_elephant',0.90942144), ('n01871265','tusker',0.08618243)
np.argmax(preds[0])
# 386
```

```
# This is the "african elephant" entry in the prediction vector
african_elephant_output = model.output[:, 386]

# Output feature map of the `block5_conv3` layer, the last convolutional layer in VGG16
last_conv_layer = model.get_layer('block5_conv3')

# The gradient of the "african elephant" class about the output feature map of `block5_conv3`
grads = K.gradients(african_elephant_output, last_conv_layer.output)[0]

# This is  mean intensity of the gradient  of shape (512,)
pooled_grads = K.mean(grads, axis=(0, 1, 2))

# This function allows us to access the values of the quantities we just defined:
# `pooled_grads` and the output feature map of `block5_conv3`, given a sample image
iterate = K.function([model.input], [pooled_grads, last_conv_layer.output[0]])
pooled_grads_value, conv_layer_output_value = iterate([x])

# We multiply each channel in the feature map array
# by "how important this channel is" with regard to the elephant class
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]

# The channel-wise mean of the resulting feature map is our heatmap of class activation
heatmap = np.mean(conv_layer_output_value, axis=-1)
```

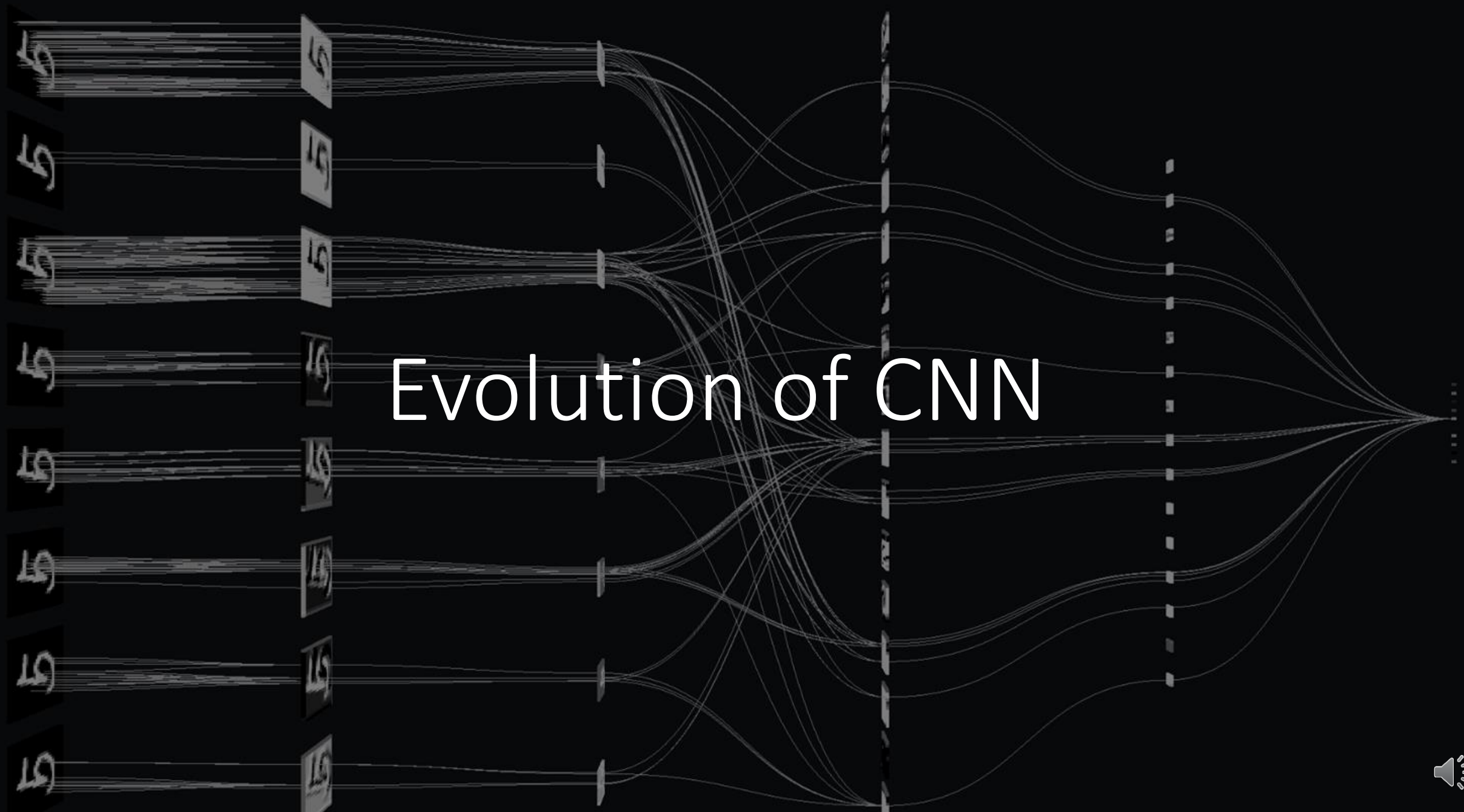
HeatMap

```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
plt.show()
```



Use OpenCV to Superimpose Heatmap with Original Image

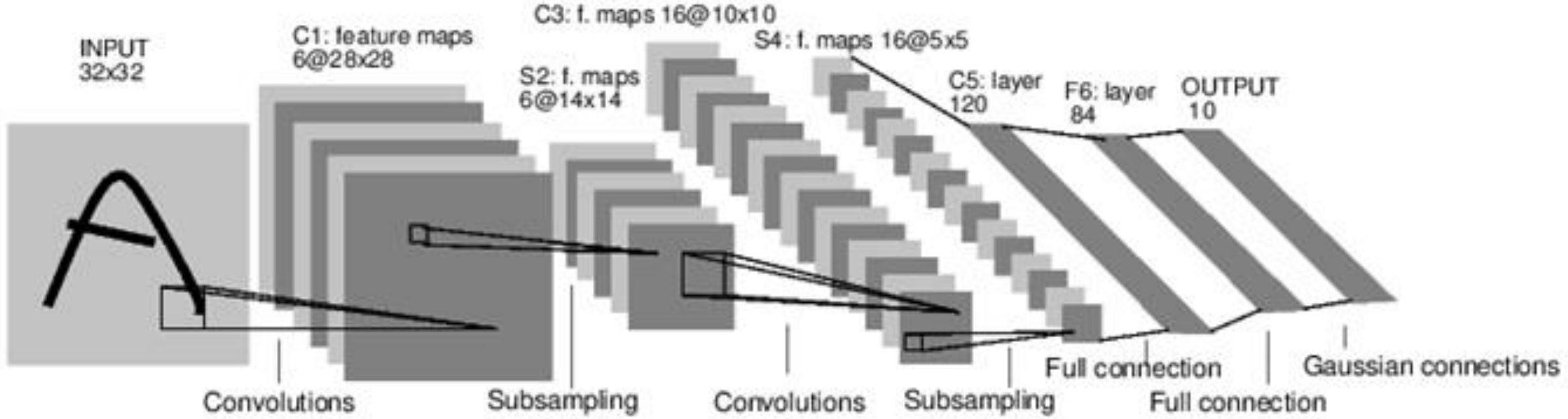
```
import cv2

# We use cv2 to load the original image
img = cv2.imread(img_path)

# We resize the heatmap to have the same size as the original image
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

# We convert the heatmap to RGB
heatmap = np.uint8(255 * heatmap)

# We apply the heatmap to the original image
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

# 0.4 here is a heatmap intensity factor
superimposed_img = heatmap * 0.4 + img

# Save the image to disk
cv2.imwrite('/Users/fchollet/Downloads/elephant_cam.jpg', superimposed_img)
```



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)

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

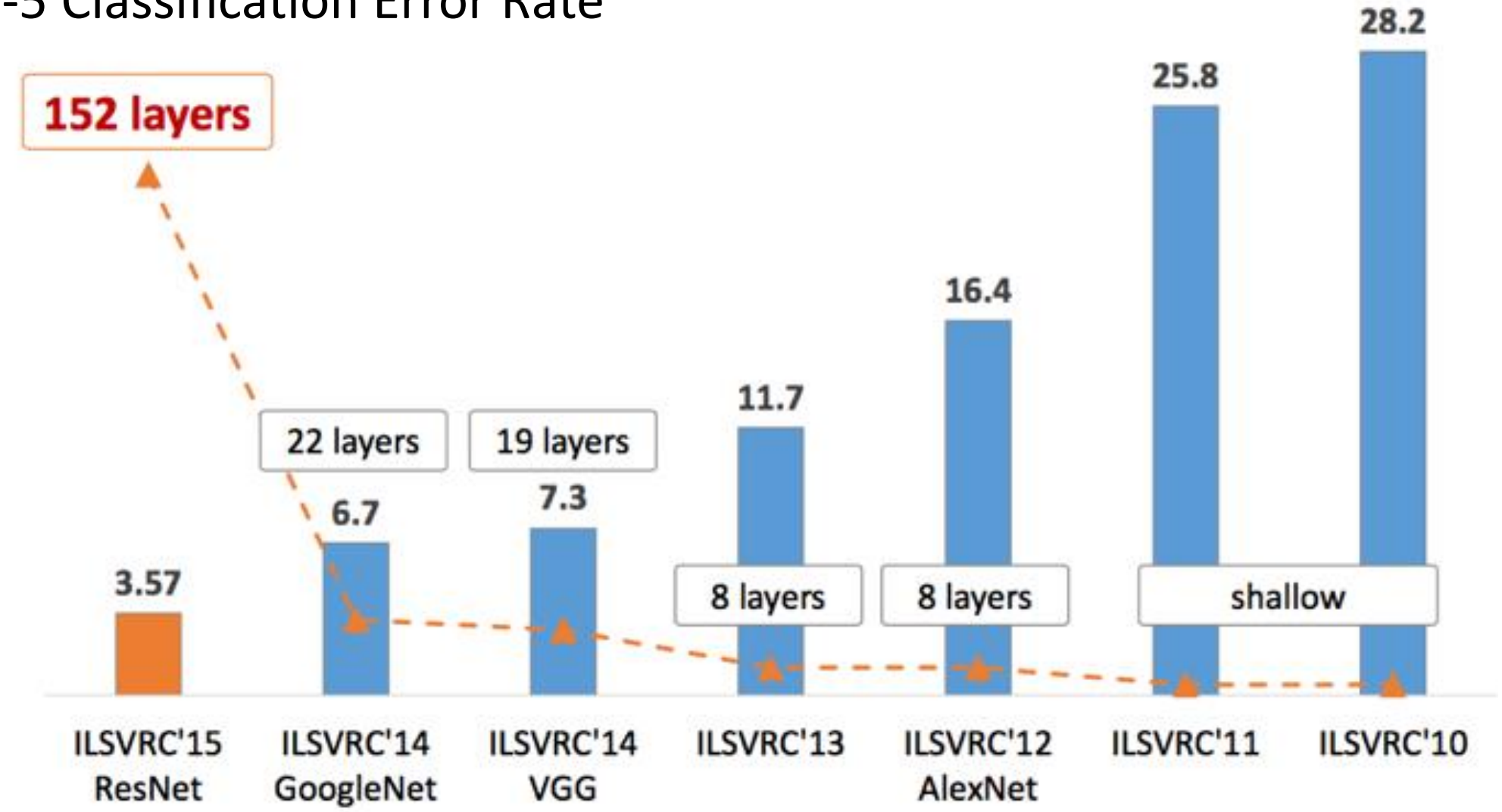

INTERNET

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

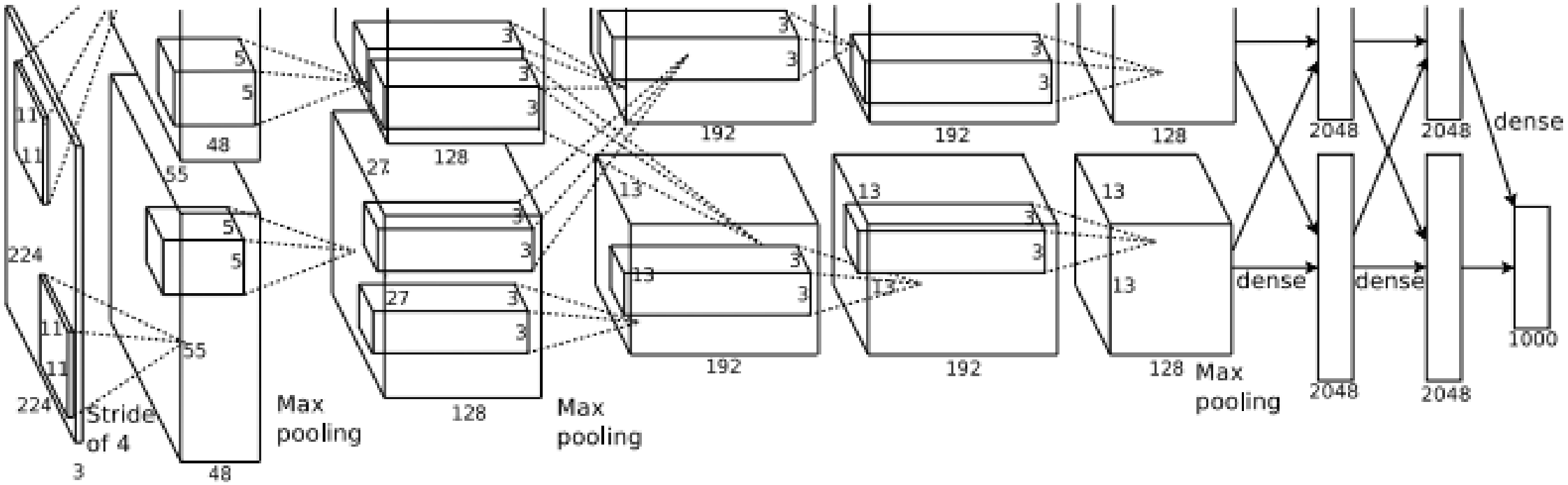
- Top-5 Classification Error Rate




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

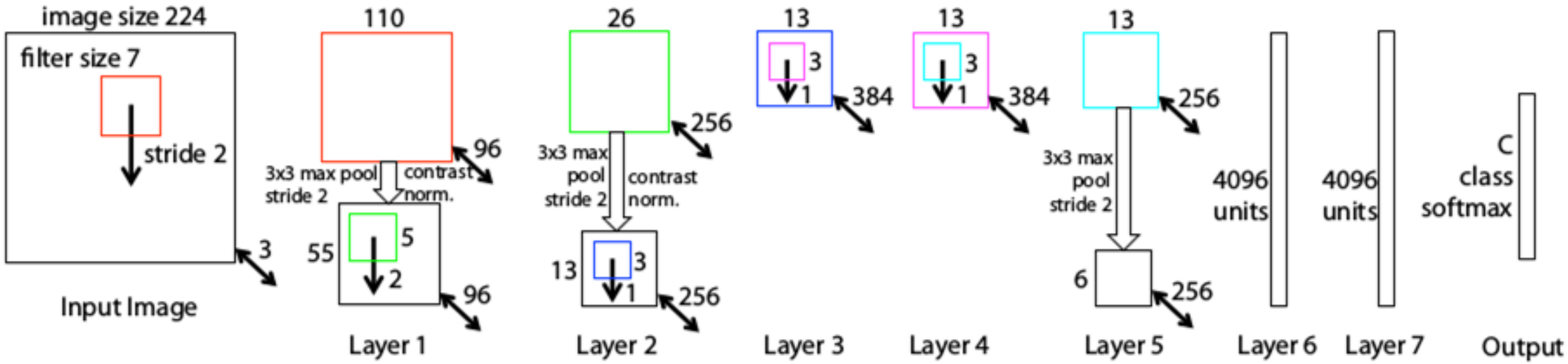
AlexNet (2012)

- AlexNet significantly outperformed previous models and started DL era
- Include convolutions, max-pooling, dropout, ReLU, SGD with momentum
- Use 2 Nvidia GeForce GTX 580 GPU



ZF Net (2013)

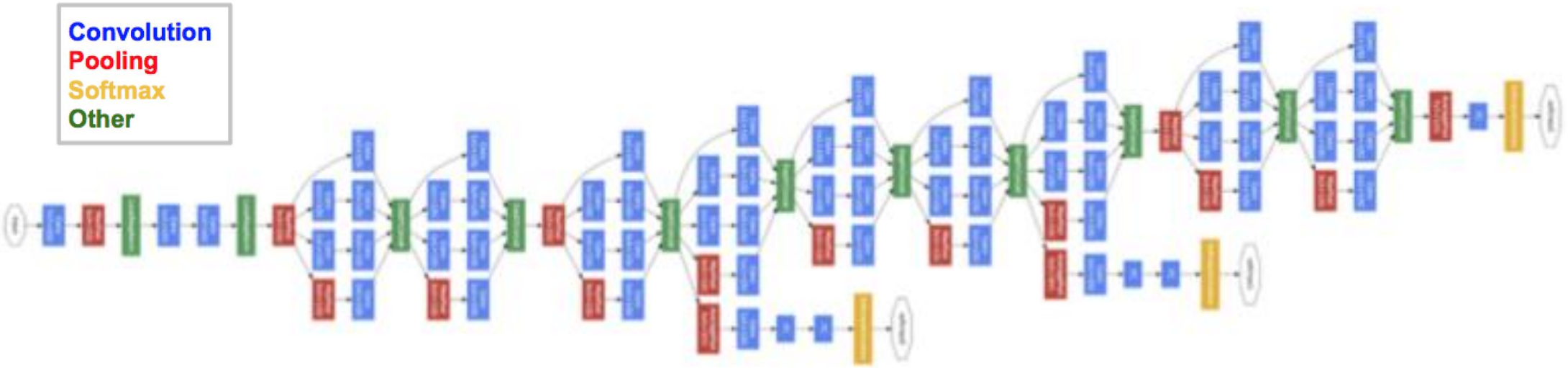
- Parameter tuning of AlexNet



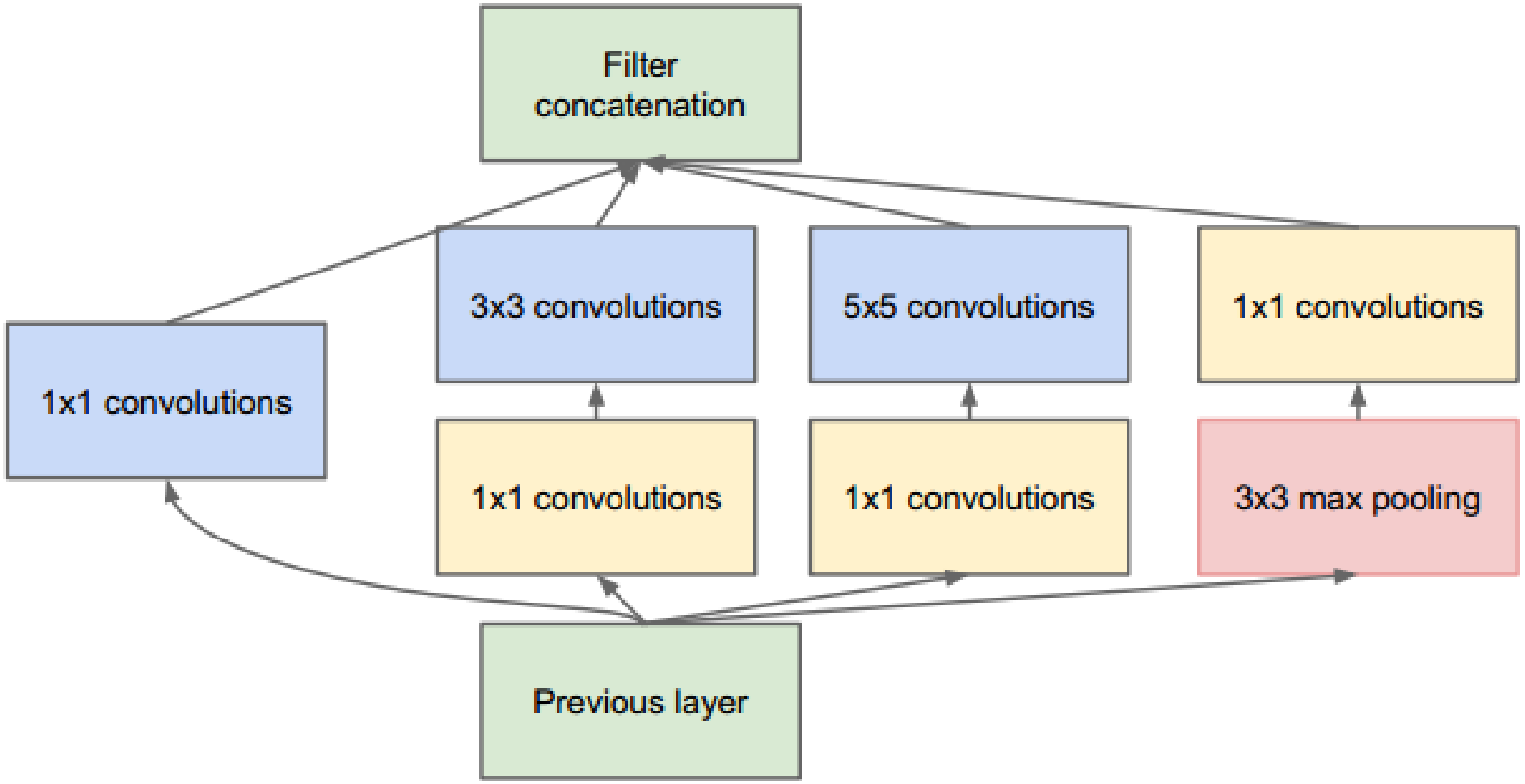
ZF Net Architecture

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 (2015)

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

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

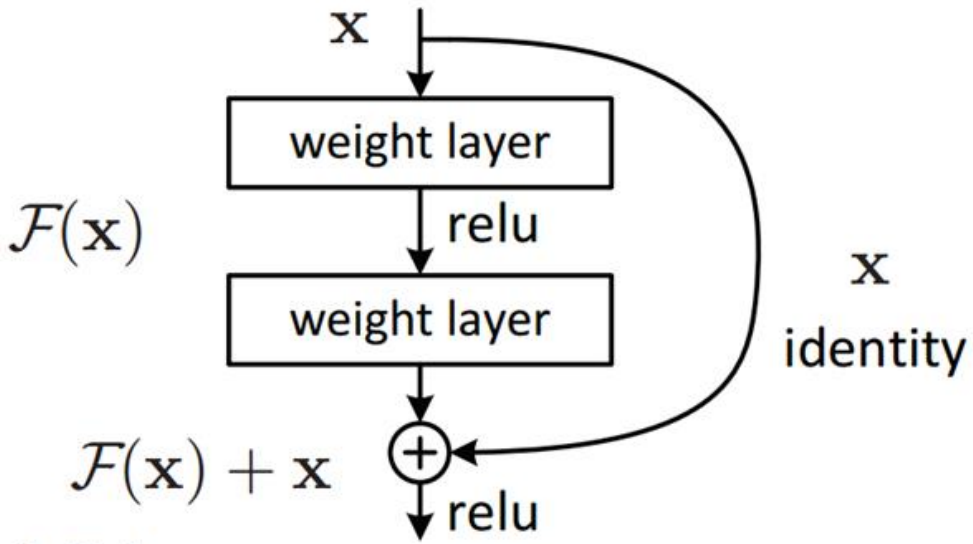
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB imag)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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

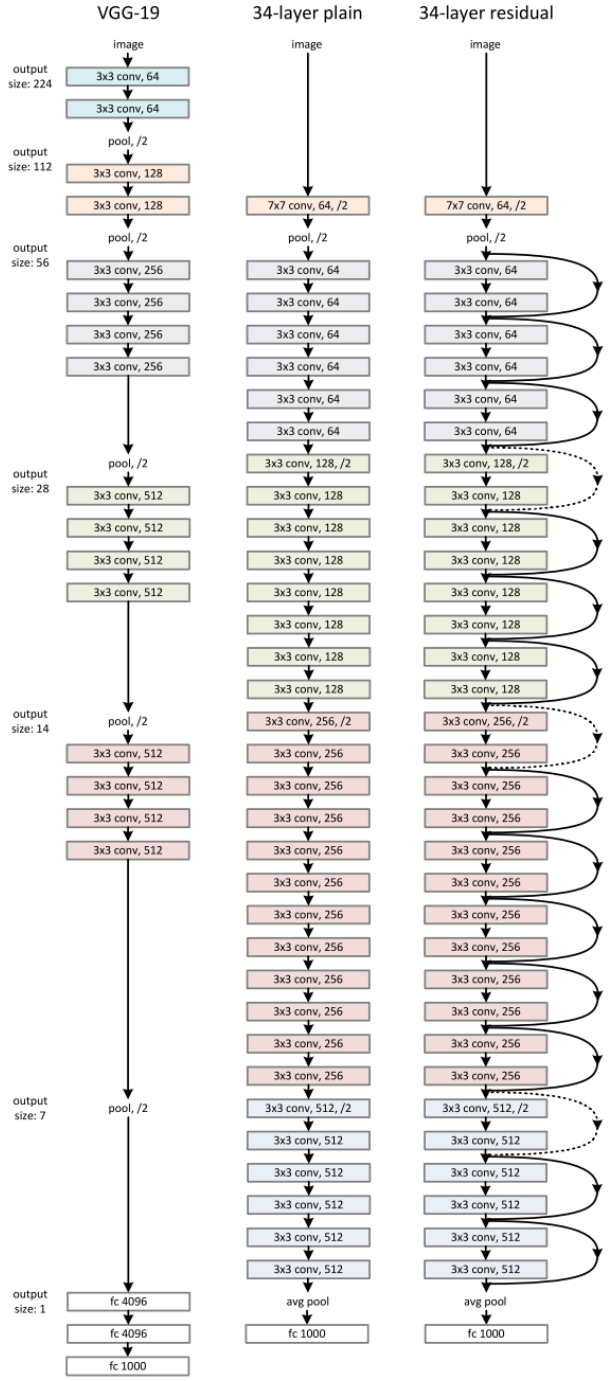

WE NEED TO GO
DEEPER
memegenerator.net

ResNet (2015)

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

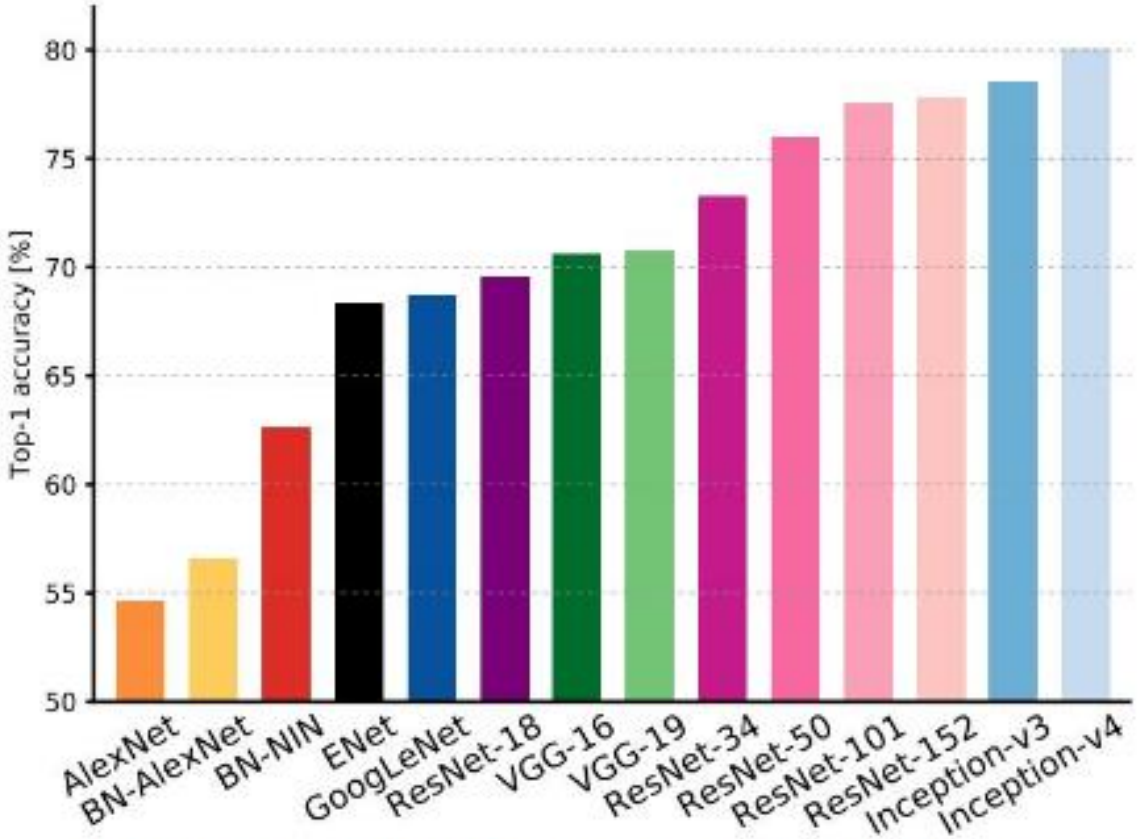


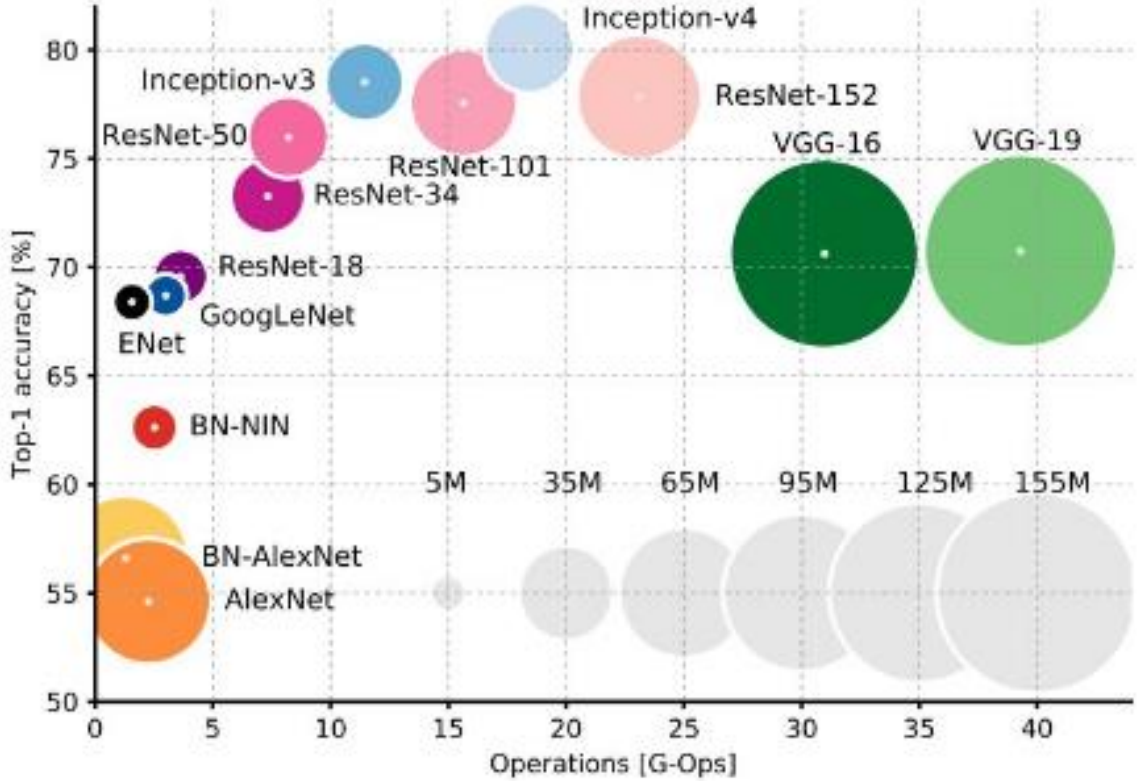
A residual block



Metrics	LeNet-5	AlexNet	VGG-16	GoogLeNet (v1)	ResNet-50
Top-5 error	n/a	16.4	7.4	6.7	5.3
Input Size	28x28	227x227	224x224	224x224	224x224
# of CONV Layers	**2**	**5**	**16**	**21 (depth)**	**49**
Filter Sizes	5	3, 5,11	3	1, 3 , 5, 7	1, 3, 7
# of Channels	1, 6	3 - 256	3 - 512	3 - 1024	3 - 2048
# of Filters	6, 16	96 - 384	64 - 512	64 - 384	64 - 2048
Stride	1	1, 4	1	1, 2	1, 2
# of Weights	2.6k	2.3M	14.7M	6.0M	23.5M
# of MACs	283k	666M	15.3G	1.43G	3.86G
# of FC layers	**2**	**3**	**3**	**1**	**1**
# of Weights	58k	58.6M	124M	1M	2M
# of MACs	58k	58.6M	124M	1M	2M
Total Weights	**60k**	**61M**	**138M**	**7M**	**25.5M**
Total MACs	**341k**	**724M**	**15.5G**	**1.43G**	**3.9G**

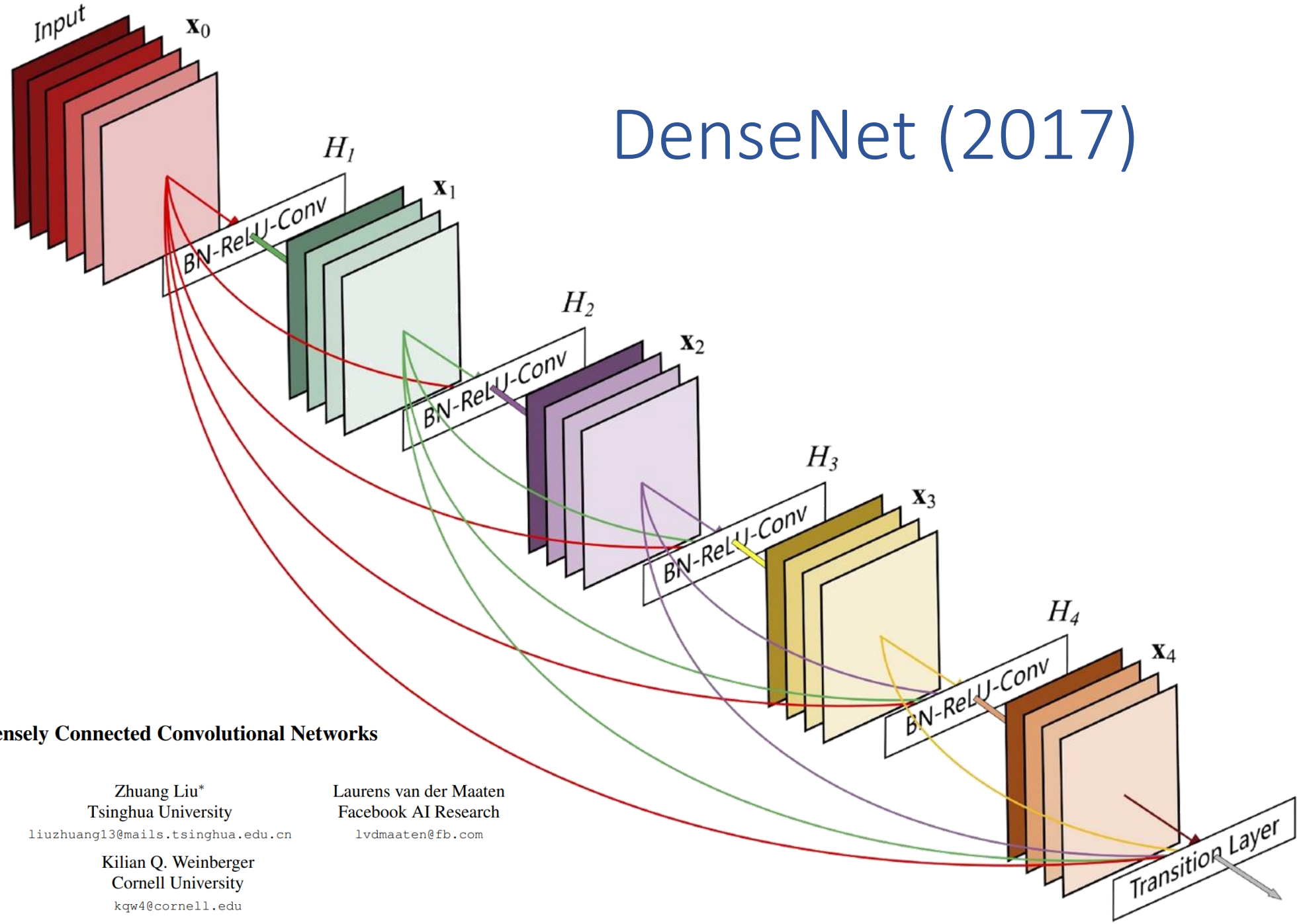
Analysis of Deep CNN Models





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

DenseNet (2017)



Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

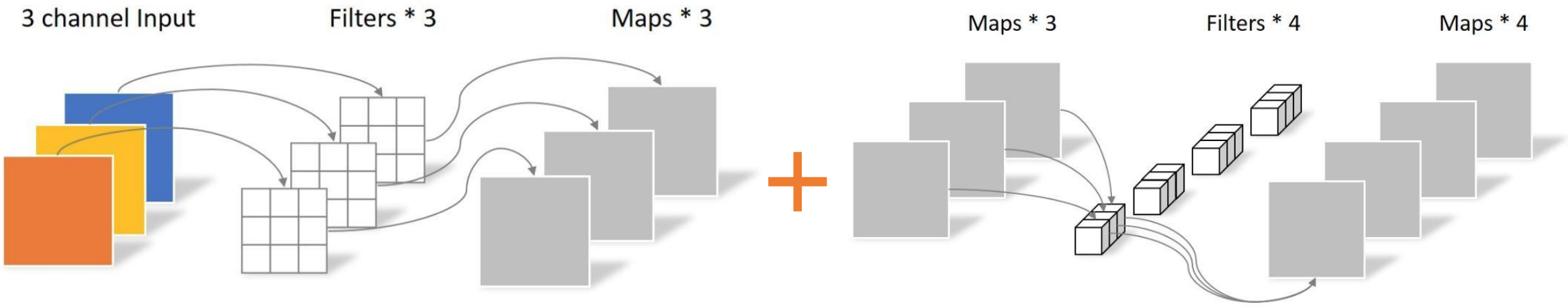
Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

Xception - Separable Convolution (2017)

- Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” *CVPR*, 2017
- Assume that cross-channel correlations and spatial correlations can be mapped completely separately

Depthwise Convolution

Pointwise Convolution



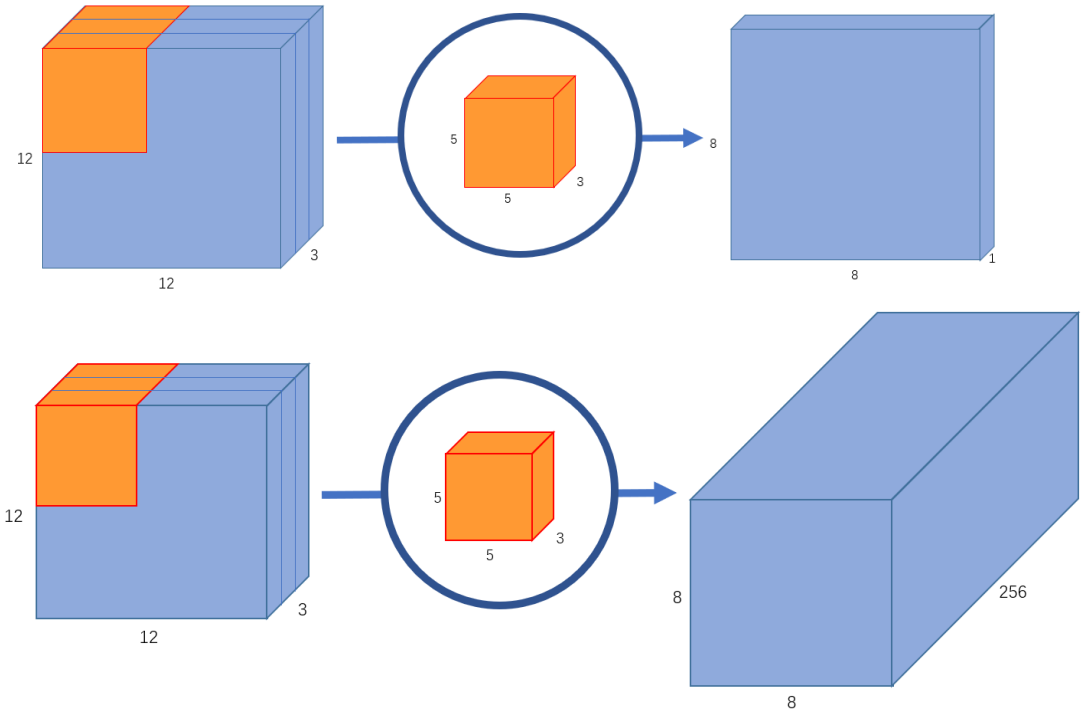
https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

Normal Convolution vs. Depthwise Separable

- Normal filer size:

  5*5*3*256

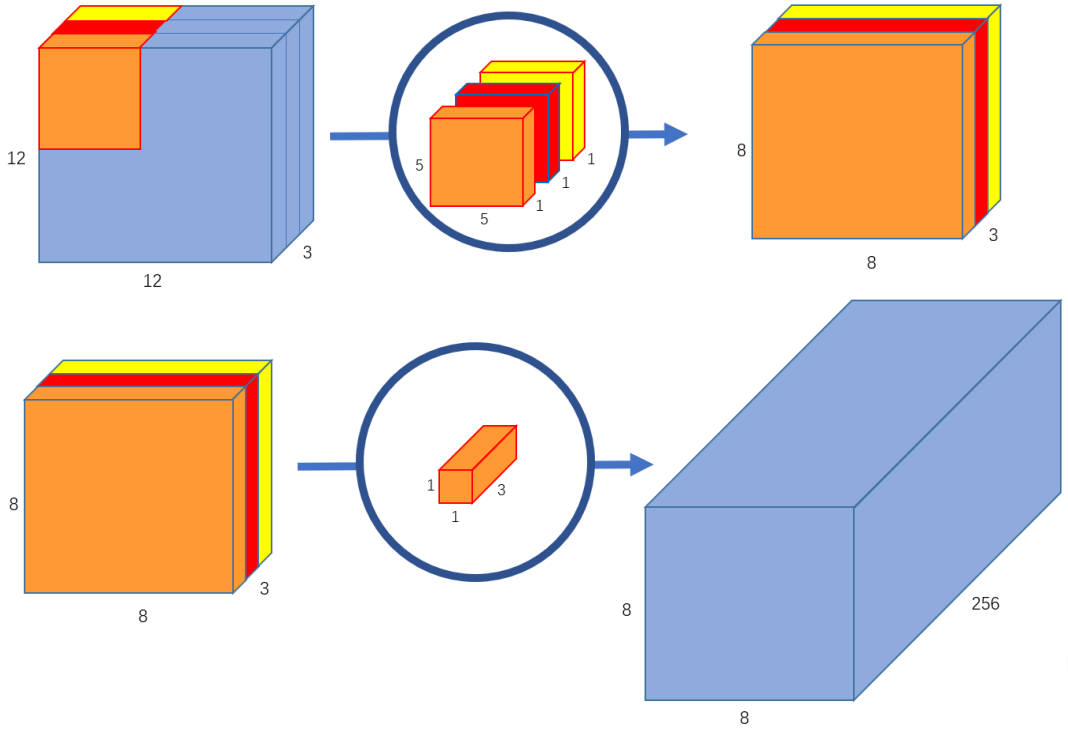
  =19200



- Depthwise filter size:

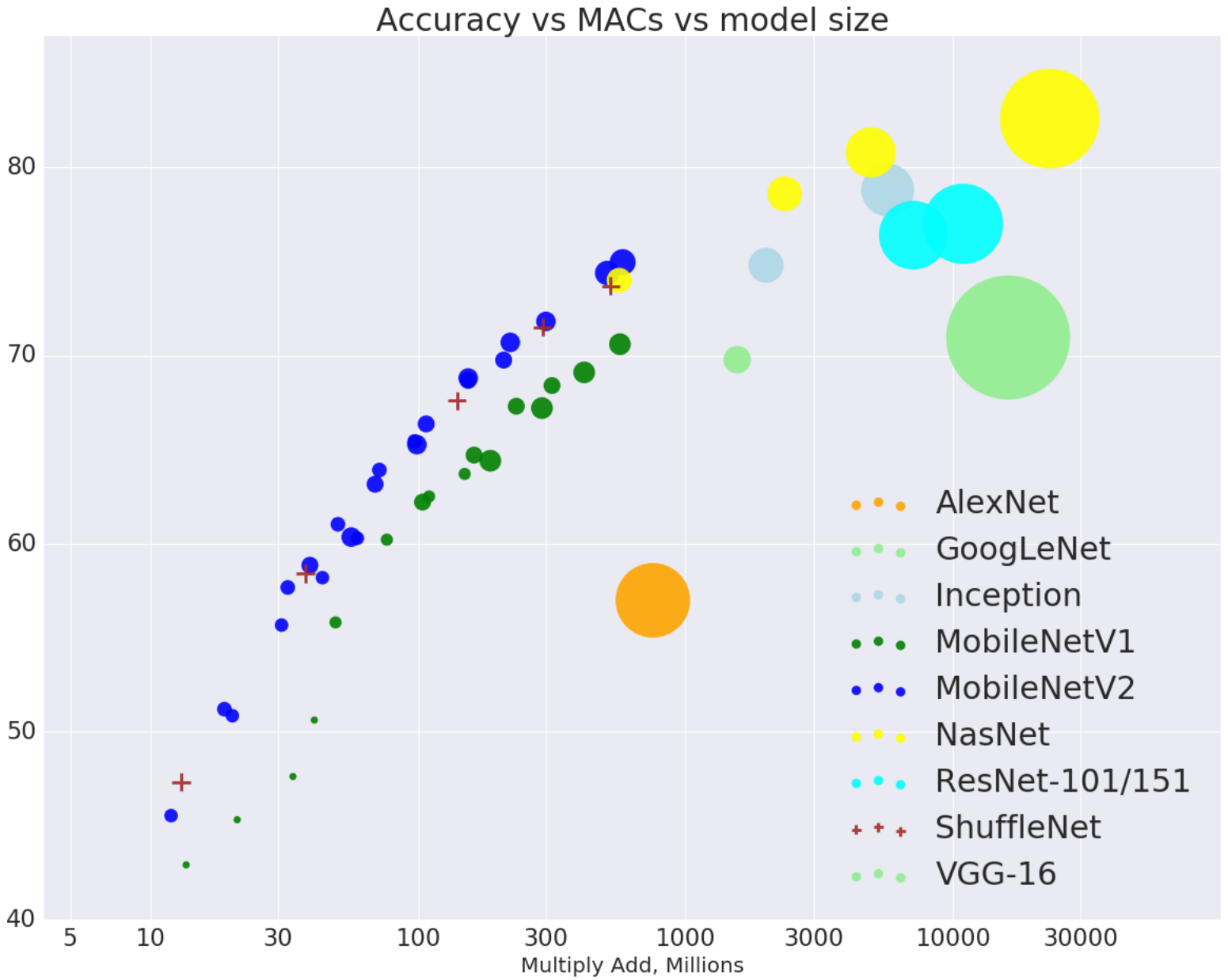
  (5*5*1)*3 + 1*1*3*256

  =843



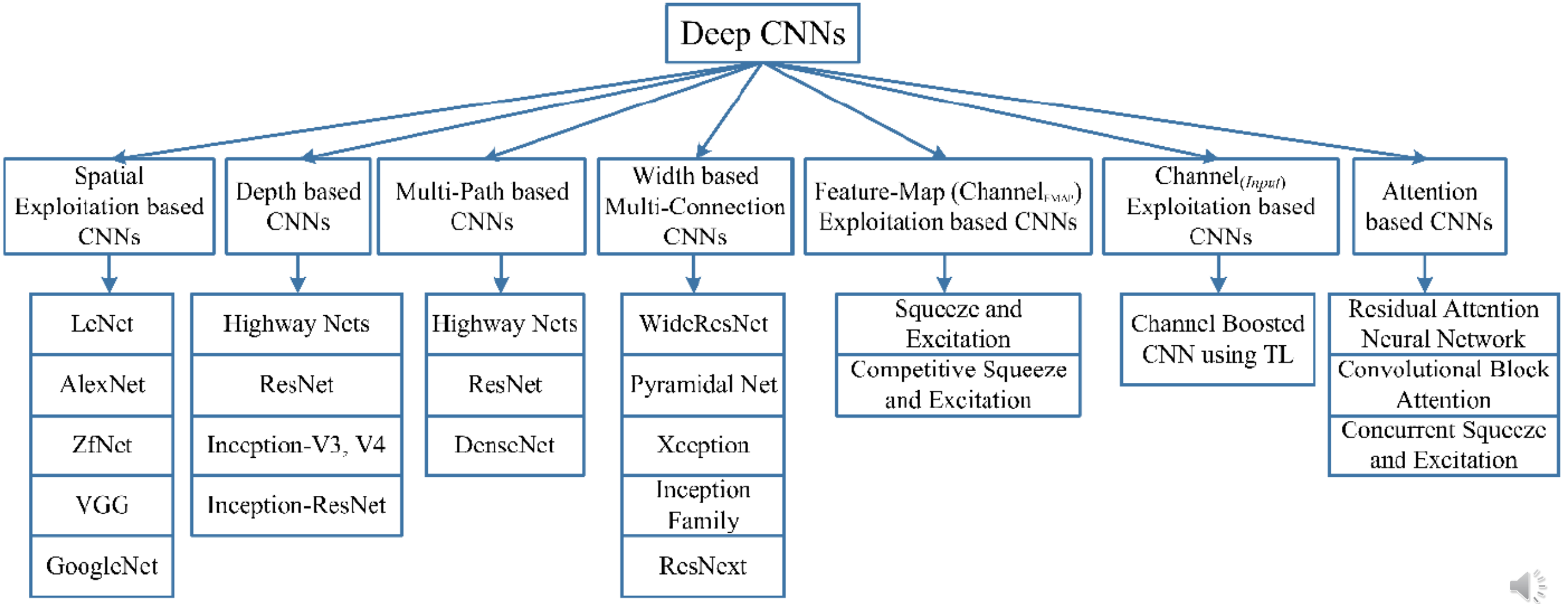
Embedded Neural Networks

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



Survey of CNN Architectures (2020)

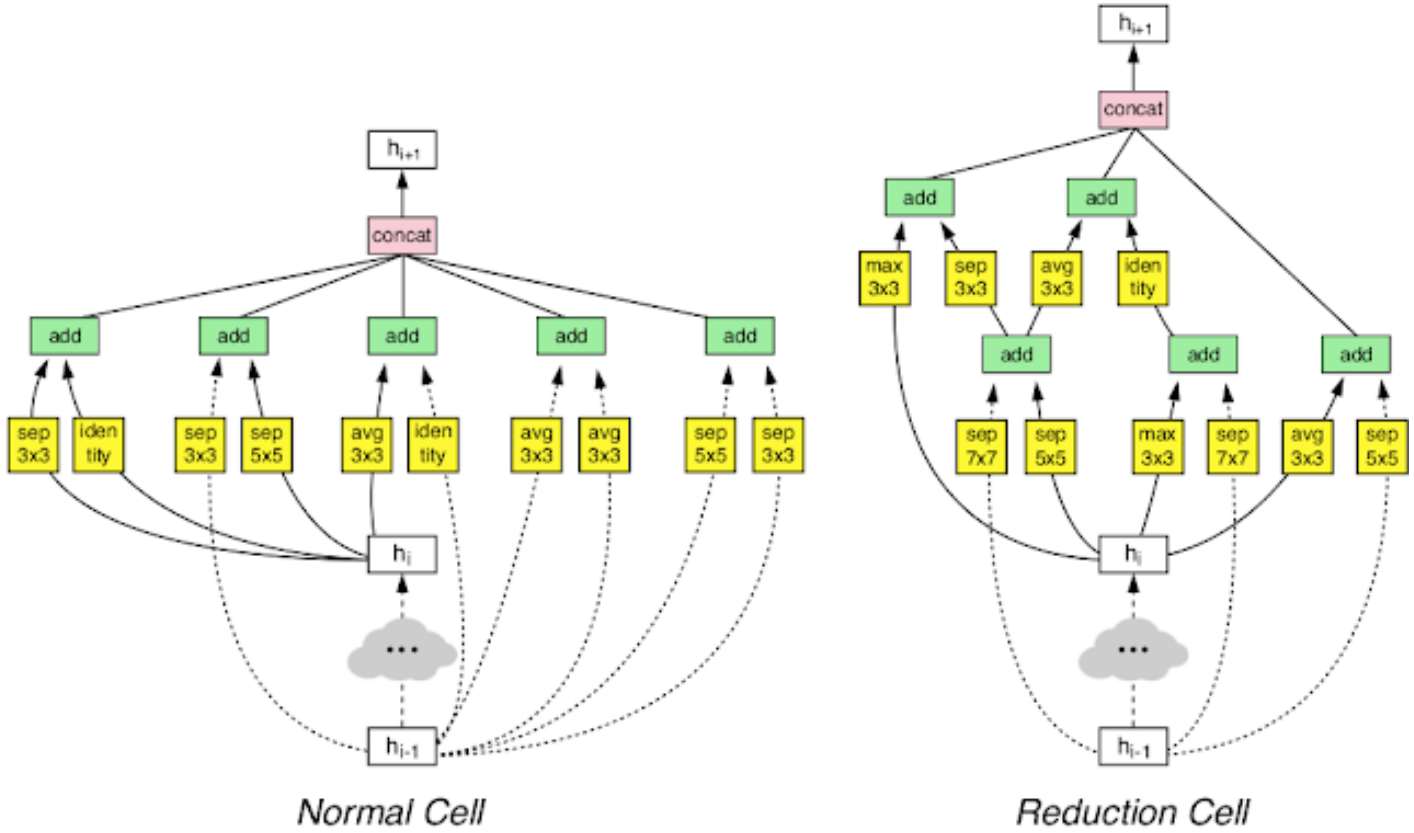
- A. Khan, "A survey of the recent architectures of deep convolutional neural networks," *Artif Intell Rev* **53,** 5455–5516, 2020
- https://arxiv.org/ftp/arxiv/papers/1901/1901.06032.pdf



Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
LeNet	1998	- First popular CNN architecture	0.060 M	[dist]MNIST: 0.8 MNIST: 0.95	5	Spatial Exploitation	(LeCun et al. 1995)
AlexNet	2012	- Deeper and wider than the LeNet - Uses Relu, dropout and overlap Pooling - GPUs NVIDIA GTX 580	60 M	ImageNet: 16.4	8	Spatial Exploitation	(Krizhevsky et al. 2012)
ZfNet	2014	-Visualization of intermediate layers	60 M	ImageNet: 11.7	8	Spatial Exploitation	(Zeiler and Fergus 2013)
VGG	2014	- Homogenous topology - Uses small size kernels	138 M	ImageNet: 7.3	19	Spatial Exploitation	(Simonyan and Zisserman 2015)
GoogLeNet	2015	- Introduced block concept - Split transform and merge idea	4 M	ImageNet: 6.7	22	Spatial Exploitation	(Szegedy et al. 2015)
Inception-V3	2015	- Handles the problem of a representational bottleneck - Replace large size filters with small filters	23.6 M	ImageNet: 3.5 Multi-Crop: 3.58 Single-Crop: 5.6	159	Depth + Width	(Szegedy et al. 2016b)
Highway Networks	2015	- Introduced an idea of Multi-path	2.3 M	CIFAR-10: 7.76	19	Depth + Multi-Path	(Srivastava et al. 2015a)
Inception-V4	2016	- Split transform and merge idea Uses asymmetric filters	35 M	ImageNet: 4.01	70	Depth +Width	(Szegedy et al. 2016a)
Inception-ResNet	2016	- Uses split transform merge idea and residual links	55.8M	ImageNet: 3.52	572	Depth + Width + Multi-Path	(Szegedy et al. 2016a)
ResNet	2016	- Residual learning - Identity mapping based skip connections	25.6 M 1.7 M	ImageNet: 3.6 CIFAR-10: 6.43	152 110	Depth + Multi-Path	(He et al. 2015a)
WideResNet	2016	- Width is increased and depth is decreased	36.5 M	CIFAR-10: 3.89 CIFAR-100: 18.85	28 -	Width	(Zagoruyko and Komodakis 2016)
Xception	2017	- Depth wise convolution followed by point wise convolution	22.8 M	ImageNet: 0.055	126	Width	(Chollet 2017)
Residual Attention Neural Network	2017	- Introduced an attention mechanism	8.6 M	CIFAR-10: 3.90 CIFAR-100: 20.4 ImageNet: 4.8	452	Attention	(Wang et al. 2017a)
ResNeXt	2017	- Cardinality - Homogeneous topology - Grouped convolution	68.1 M	CIFAR-10: 3.58 CIFAR-100: 17.31 ImageNet: 4.4	29 - 101	Width	(Xie et al. 2017)
Squeeze & Excitation Networks	2017	- Models interdependencies between feature-maps	27.5 M	ImageNet: 2.3	152	Feature-Map Exploitation	(Hu et al. 2018a)

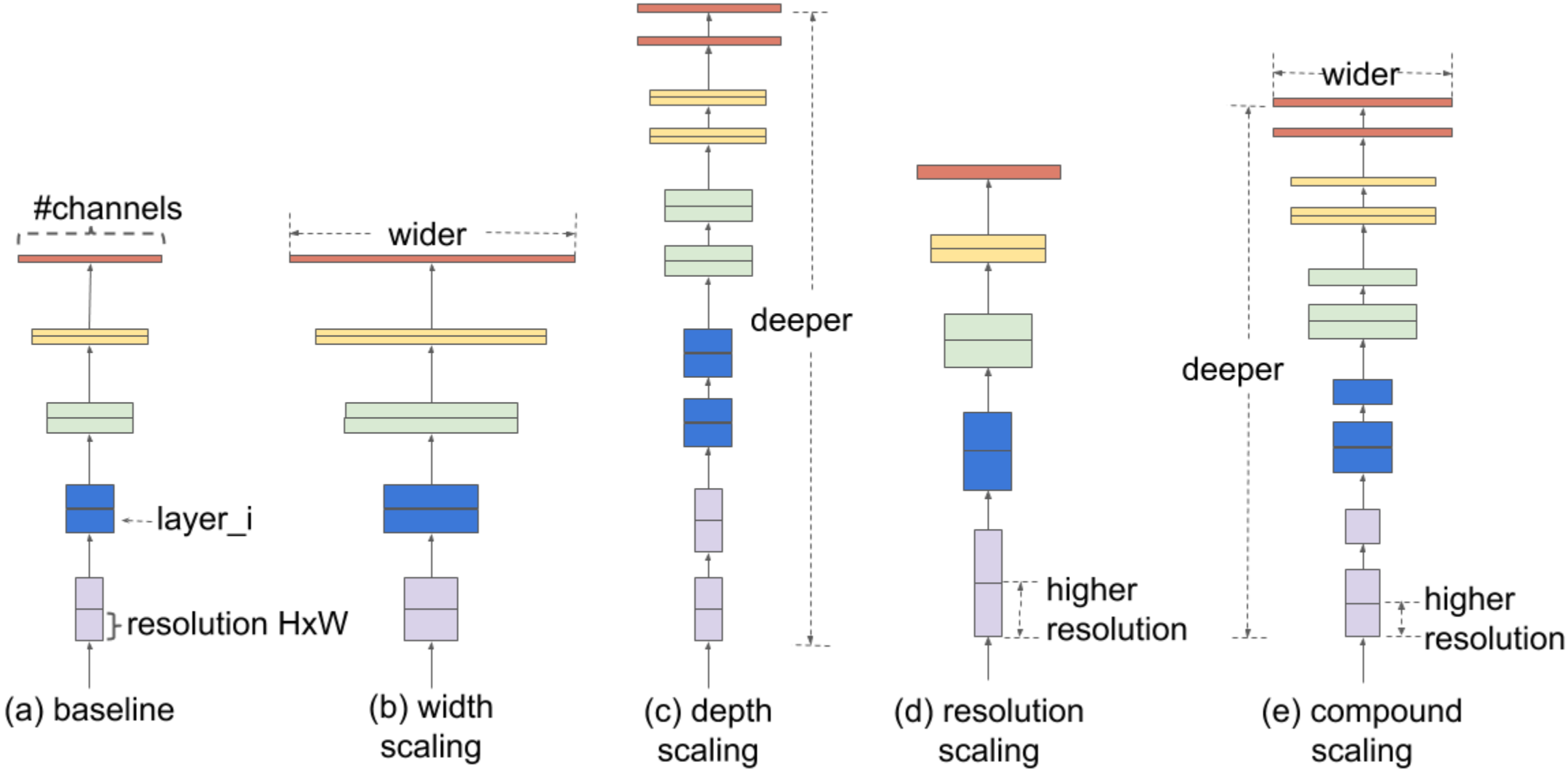
Network Architecture Search Network (NASNet)

- Learning neural network cells automatically (Google, 2017)

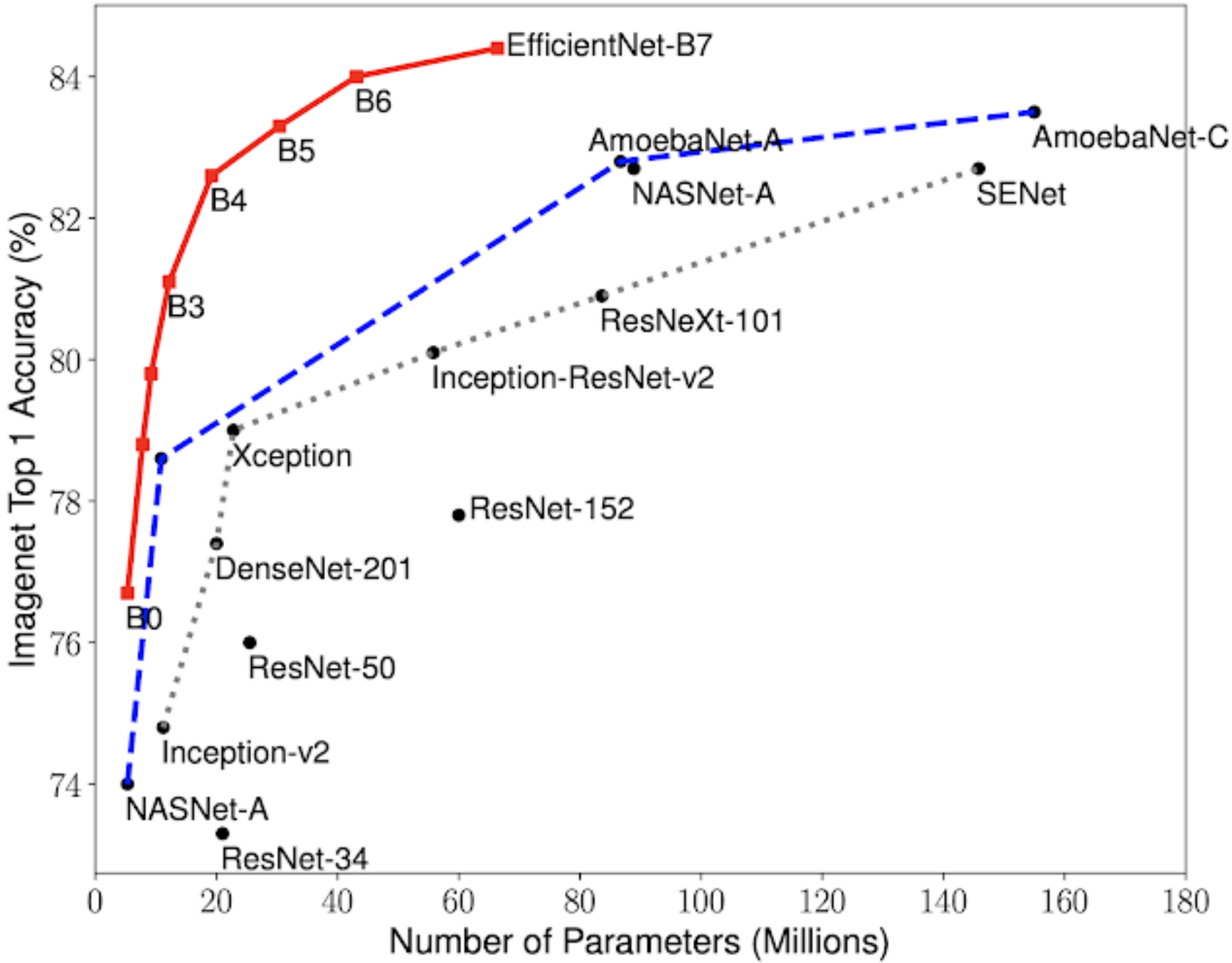


https://ai.googleblog.com/2017/11/automl-for-large-scale-image.html

EfficientNet (May, 2019)



https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html


EfficientNet-B7
B6
B5
B4
B3
B2
B1
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)
84
82
80
78
76
74
0
20
40
60
80
100
120
140
160
180

Recap

AlexNet	First large CNN model trained on GPU with ReLU, max-pooling, dropout, SGD with momentum.
Inception	Propose inception module that apply multiple filters to one input feature map.
ResNet	Use residual connection (shortcut) to avoid vanishing gradient problem and successfully train very deep networks
Xception	Reduce number of Conv. filter weights significantly using Depthwise and Pointwise Conv. filters
NASNet & EfficientNet	Use AutoML to find the CNN architecture

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
- A. Khan, “A Survey of the Recent Architectures of Deep Convolutional Neural Networks,” ArXiv, 2019