Deep Learning

Prof. Tiago Vieira, PhD

tvieira@ic.ufal.br

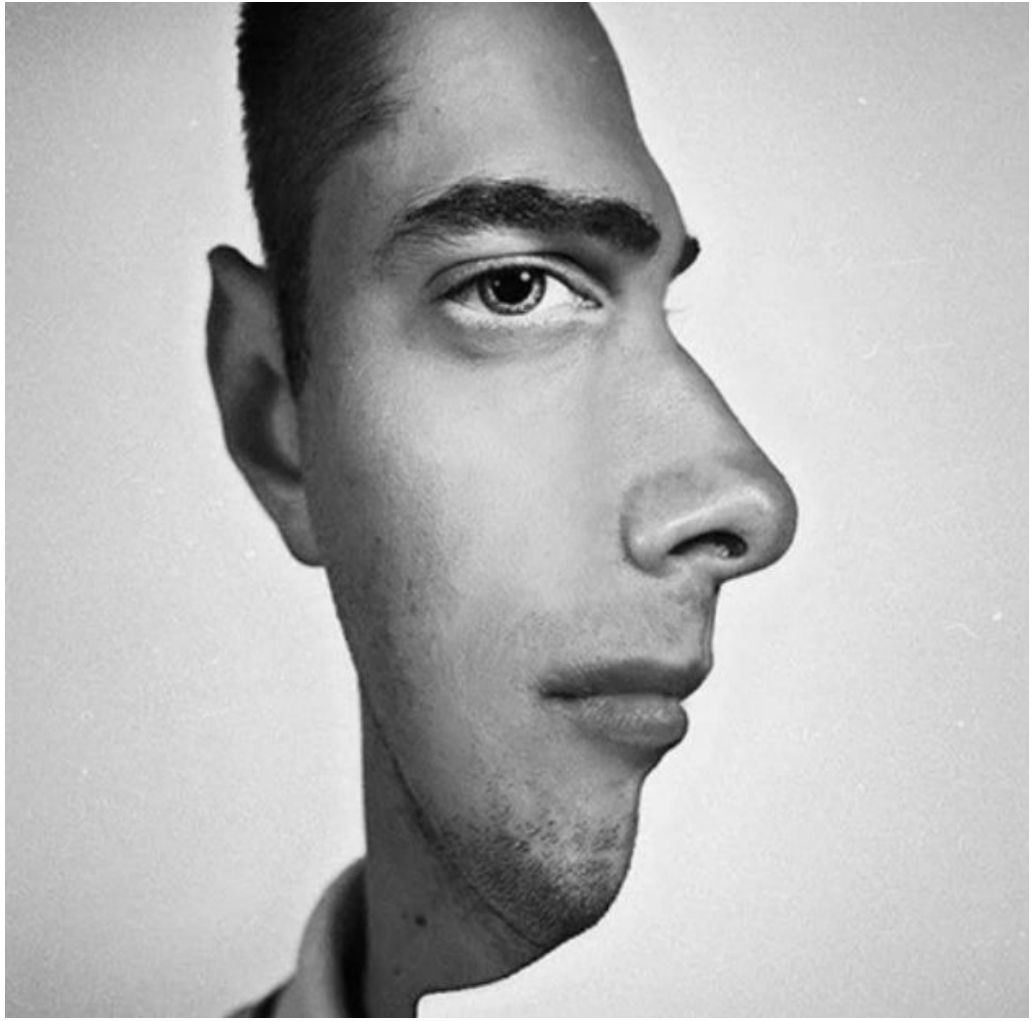
Convolutional Neural Networks (ConvNets)

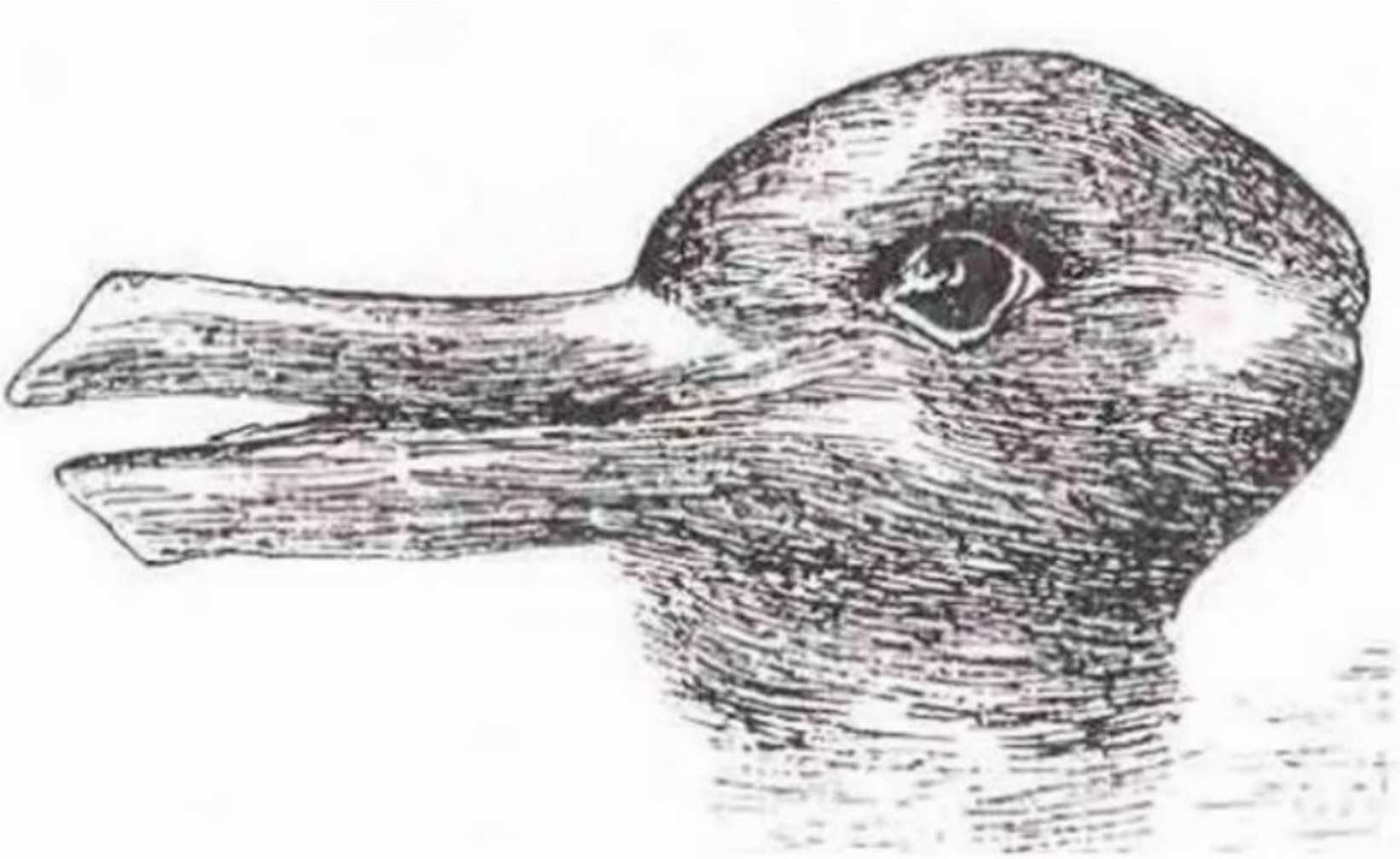
Intuition.

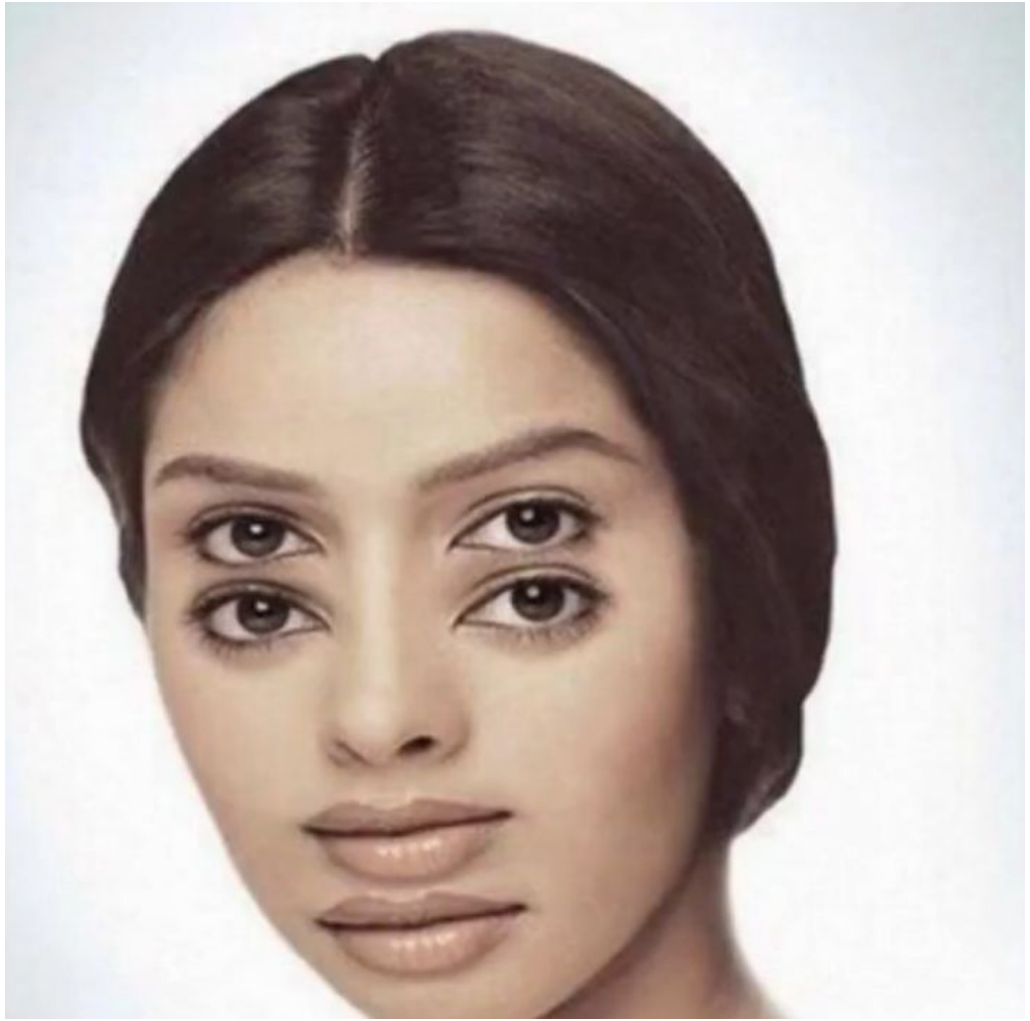
Motivation - MNIST.

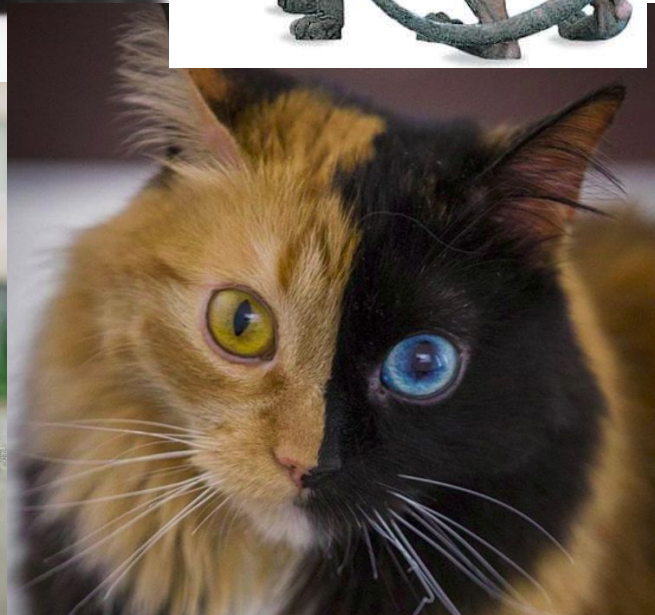
1. Convolution operation.
2. Rectifier Linear Unit activation function (ReLU).
3. Max-Polling.
4. Flattening.
5. Full Connection.

Intuition









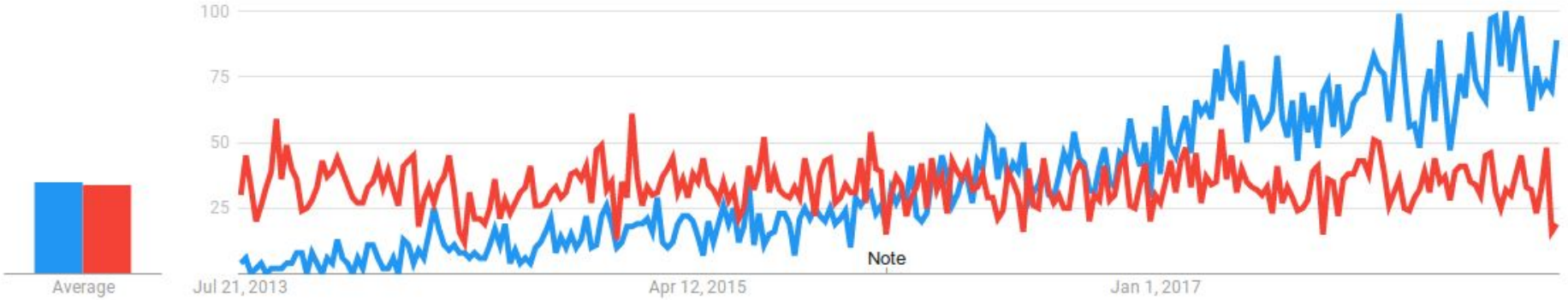
convolutional neural networks
Search term

artificial neural networks
Search term

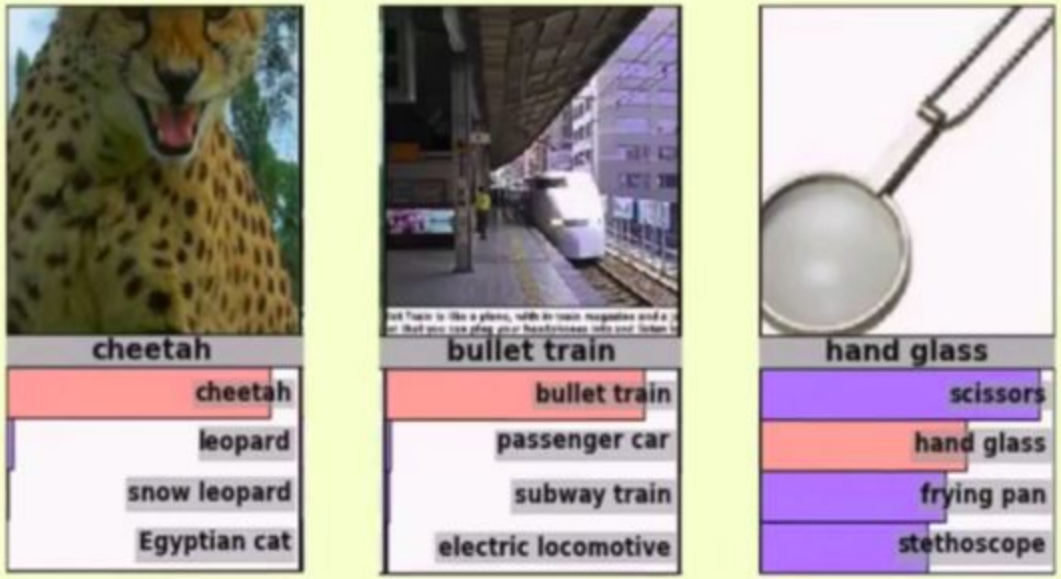
+ Add comparison

Worldwide | Past 5 years | All categories | Web Search

Interest over time



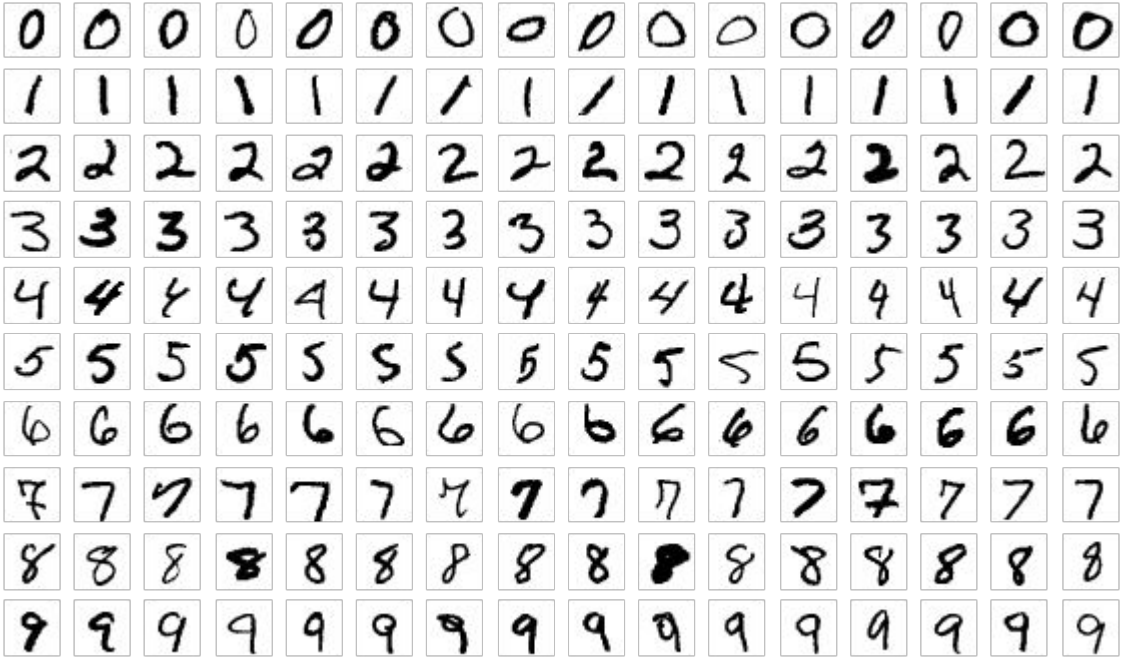
Examples from the test set
(with the network's guesses)



Motivation - MNIST

Modified National Institute of Standards and Technology DB

- 60k training.
- 10k testing.
- 28x28.



```
#%% Import modules
import keras
keras.__version__
import matplotlib.pyplot as plt
import numpy as np
```

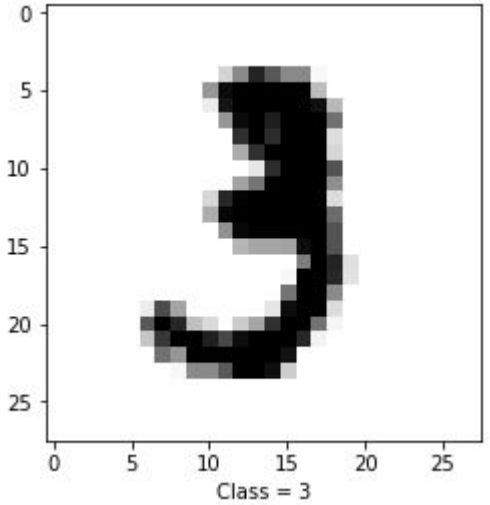
```
#%% Listing 5.1 Instantiating a small convnet
from keras import layers
from keras import models

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()
```

```
_________________________________________________________________
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
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

```
#%% Listing 5.2 Adding a classifier on top of the convnet
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

```
_________________________________________________________________
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
_________________________________________________________________
```

```
#%% Listing 5.3 Training the convnet on MNIST images
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
idx = 10
plt.imshow(255 - train_images[idx], cmap = 'gray')
plt.xlabel('Class = ' + str(train_labels[idx]))
plt.show()
```

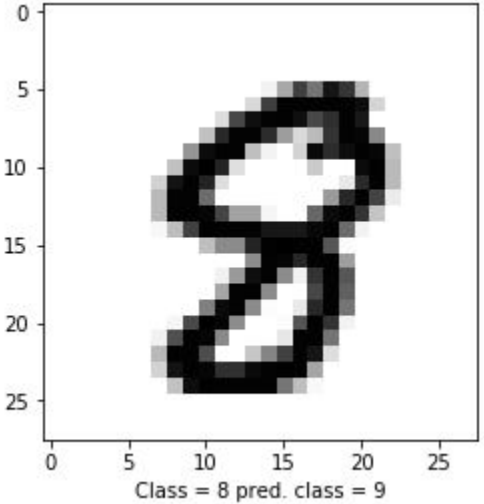


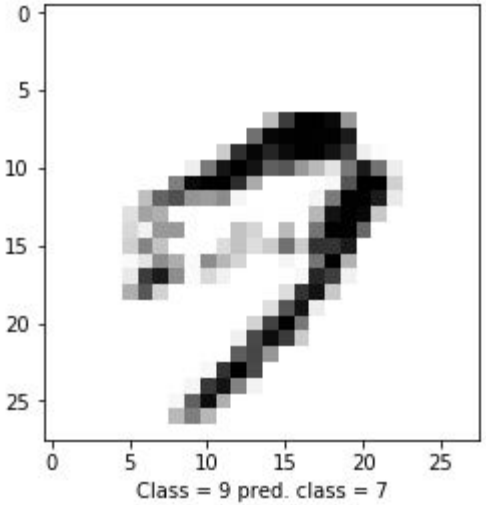
```
#%% Listing 5.3 Training the convnet on MNIST images
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
idx = 10
plt.imshow(255 - train_images[idx], cmap = 'gray')
plt.xlabel('Class = ' + str(train_labels[idx]))

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

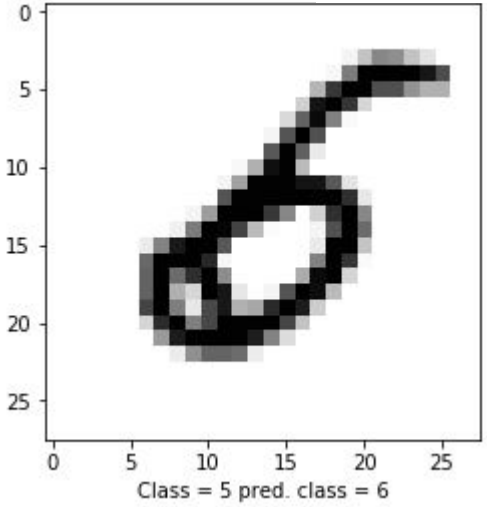
```
#%% Train and save model
#model.fit(train_images, train_labels, epochs=5,
batch_size=64)
#model.save('model-5-1.h5')
from keras.models import load_model
model = load_model('model-5-1.h5')
test_loss, test_acc = model.evaluate(test_images,
test_labels)
test_acc # 99.08
```

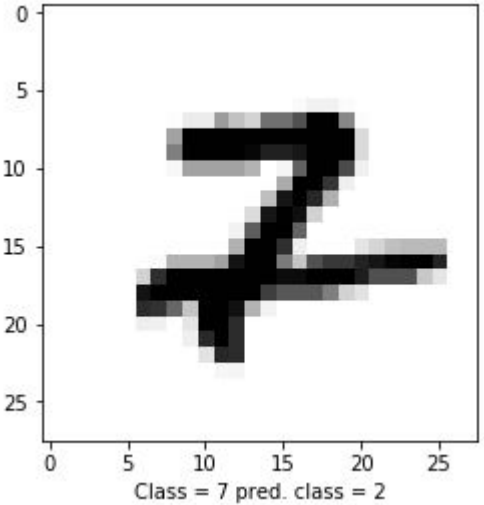
```
#%% Investigate some test samples
pred_labels = model.predict_classes(test_images)
pred_scores = model.predict(test_images)
labels = np.argmax(test_labels, axis = 1)
idxs = ~(pred_labels == labels)
ind = np.where(idxs)[0]
for i in ind:
    img = test_images[i]
    plt.imshow(255 - img[:, :, 0], cmap='gray')
    plt.xlabel('Class = ' + str(labels[i]) +
               ' pred. class = ' + str(pred_labels[i]))
    plt.show()
```

Misclassifications

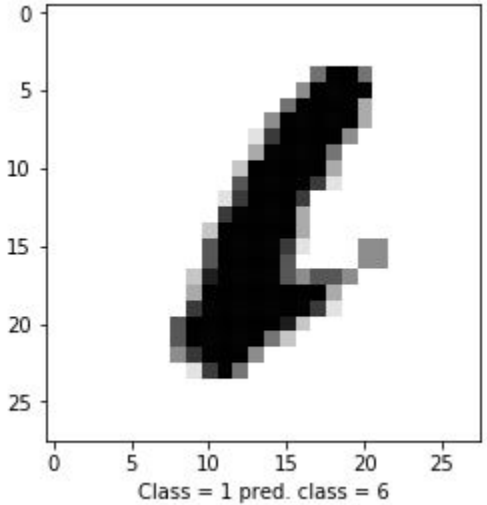


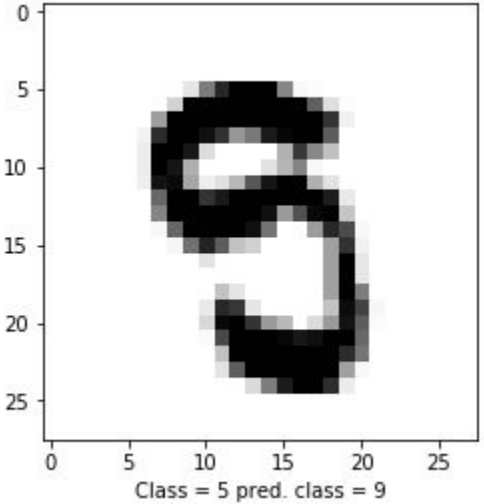


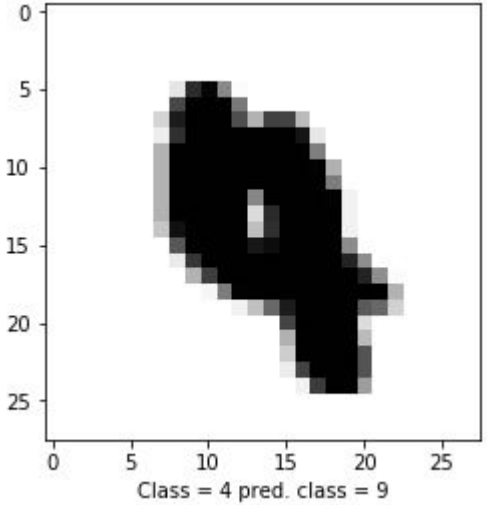




Misclassifications








Draw your number here
Downsampled drawing:
First guess: 4
Second guess: 1
Layer visibility
Input layer
Show
Convolution layer 1
Show
https://goo.gl/vEsVgx

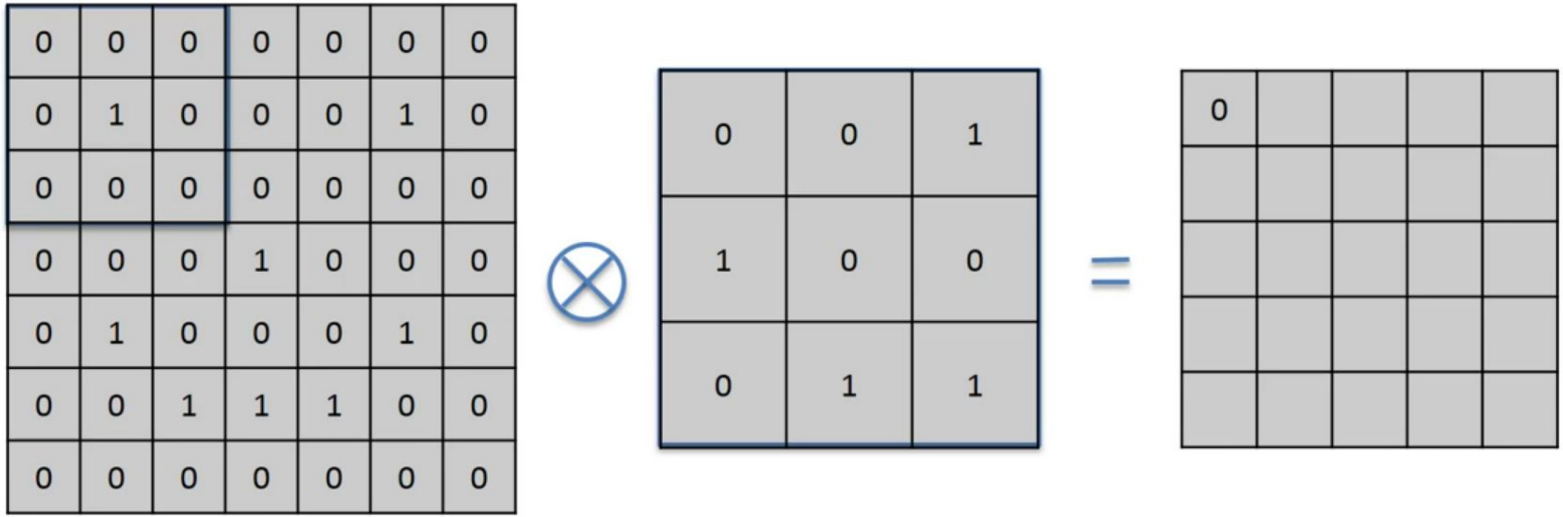
The Convolution Operation

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input Image

0	0	1
1	0	0
0	1	1

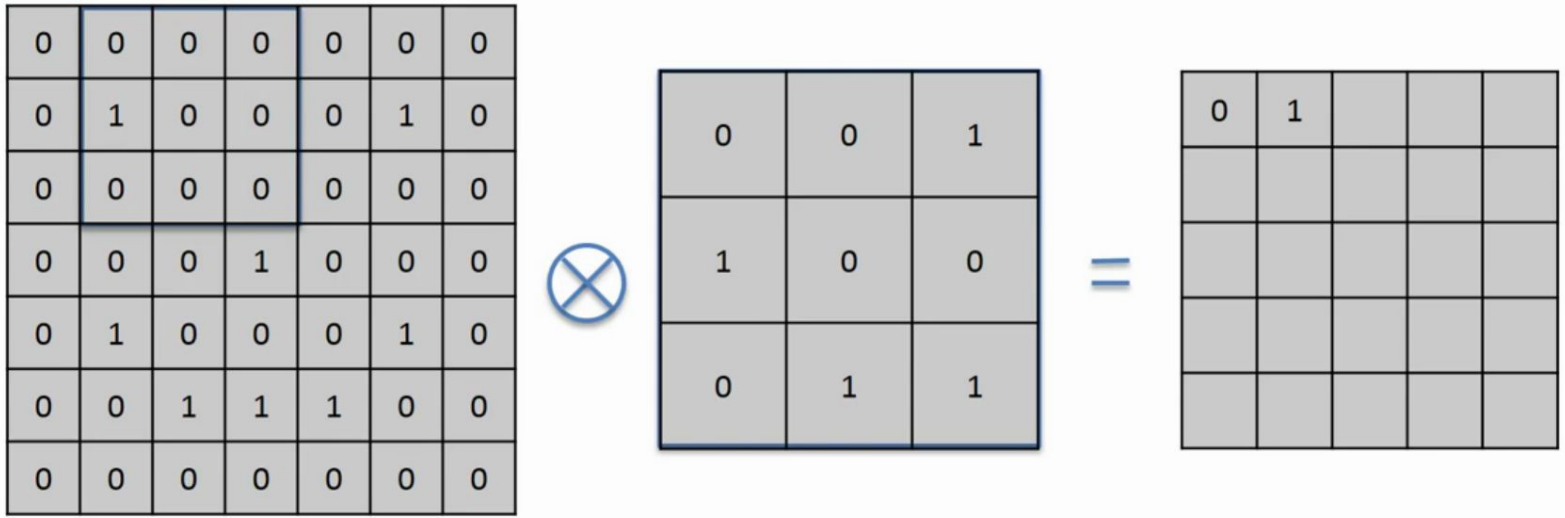
Feature Detector

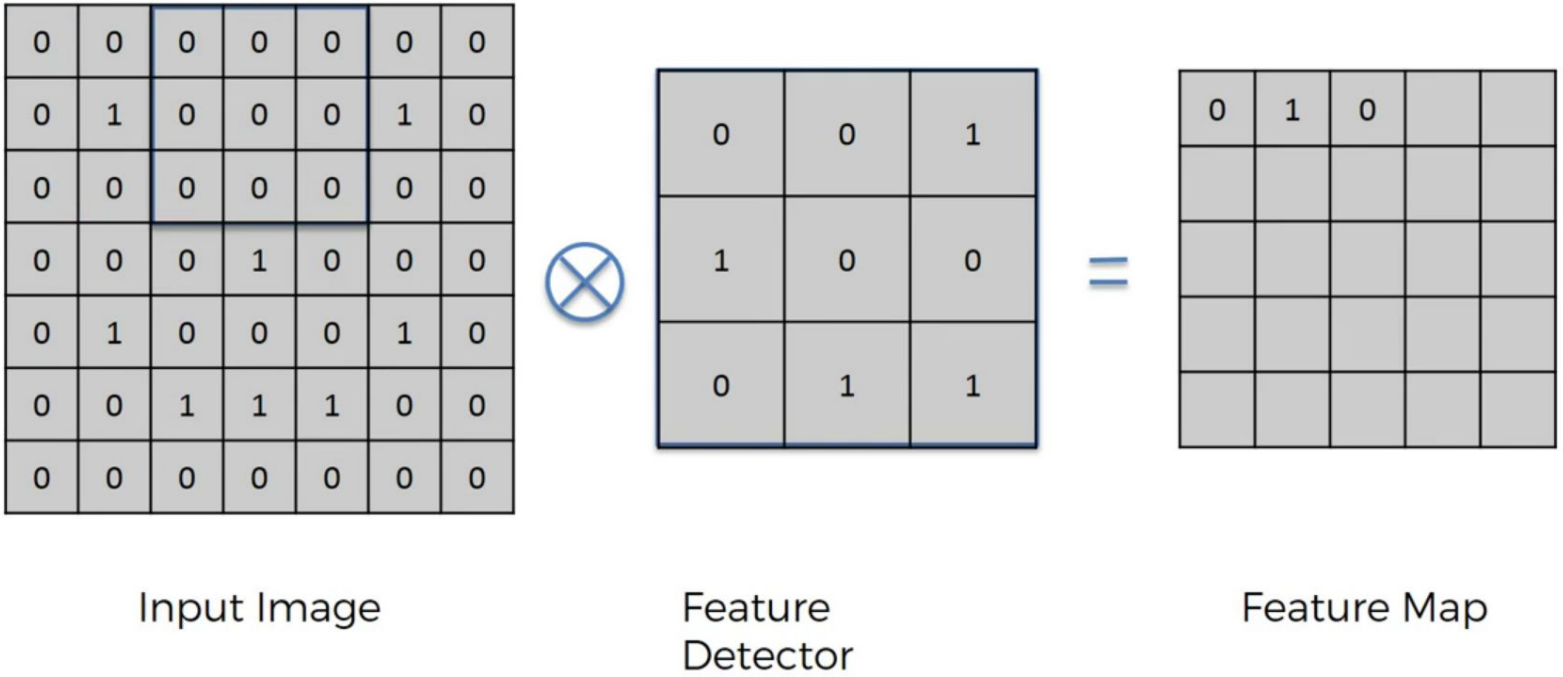


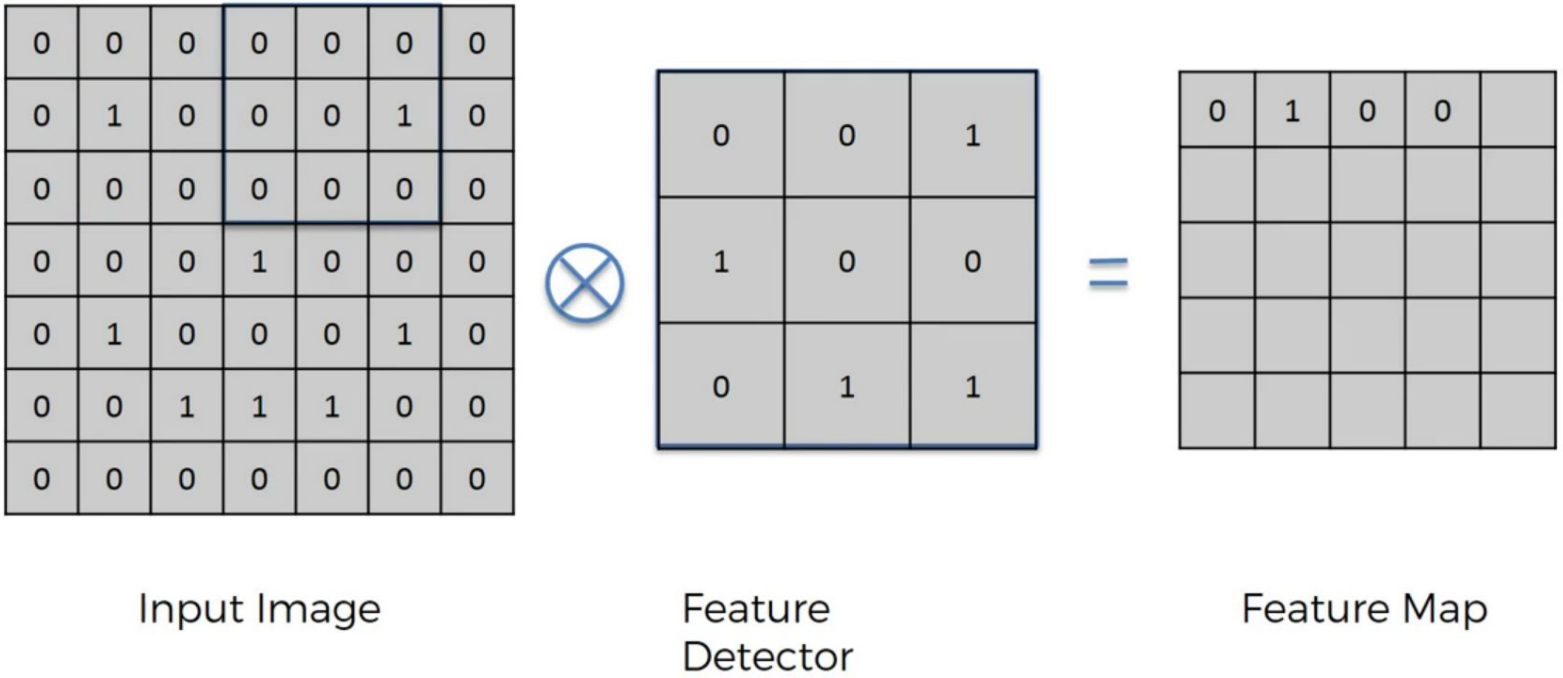
Input Image

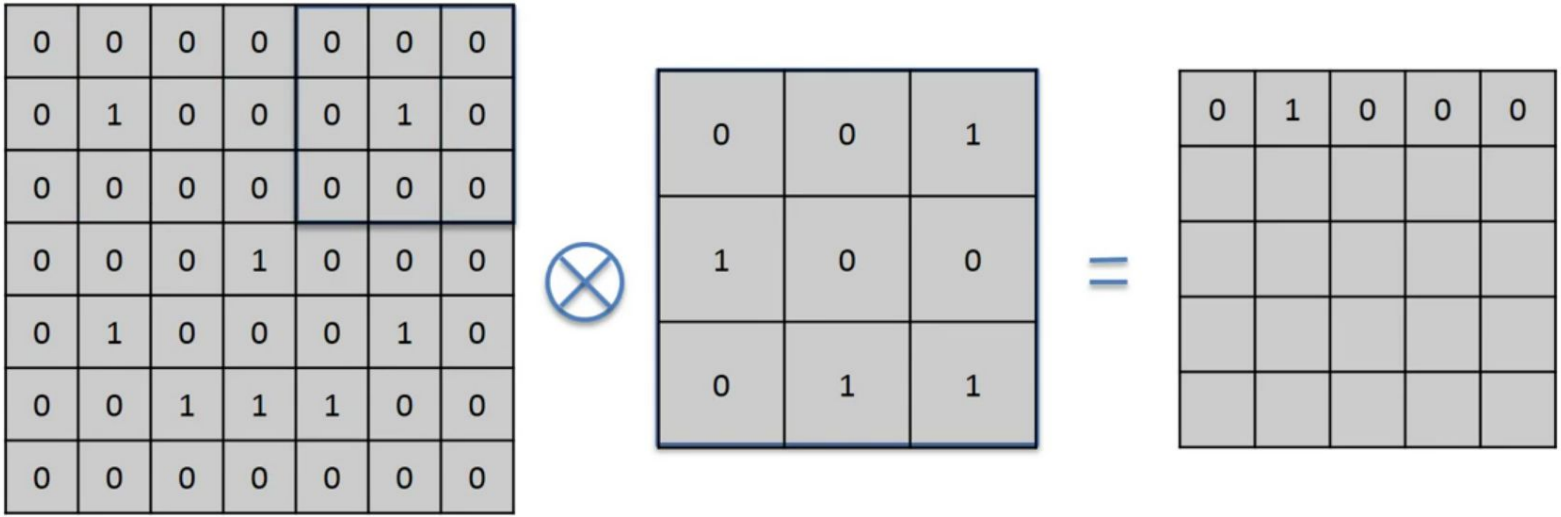
Feature Detector

Feature Map


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1
Input Image
Feature Detector
Feature Map


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0
Input Image
Feature Detector
Feature Map

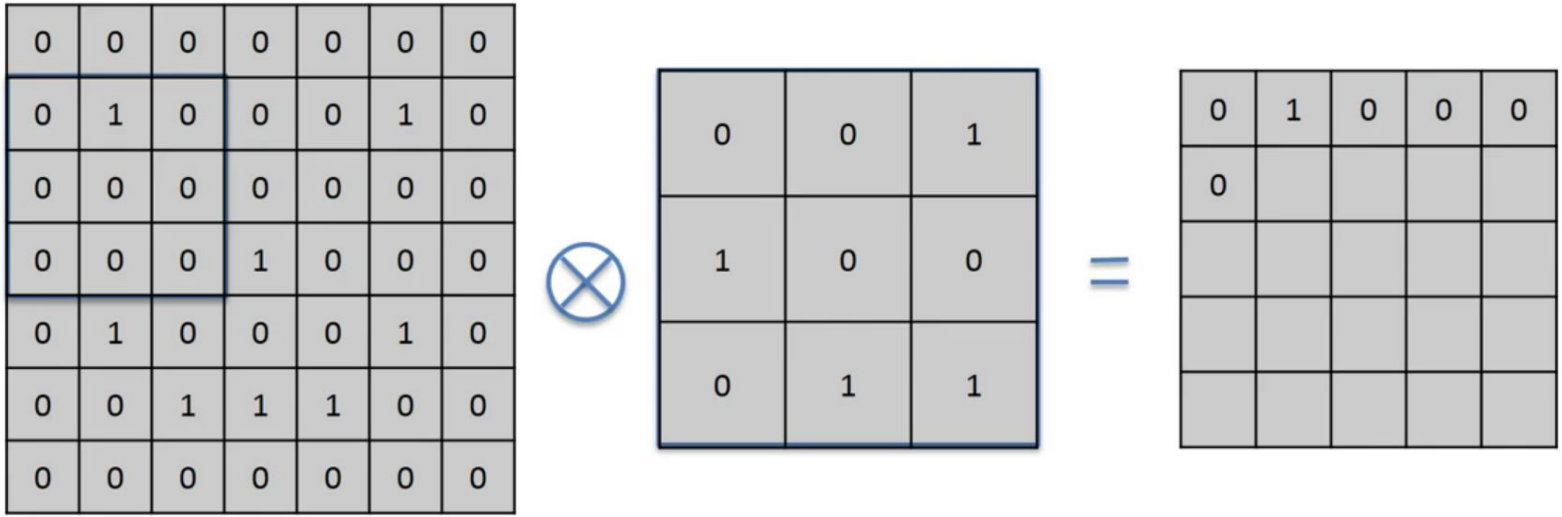

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0
Input Image
Feature Detector
Feature Map



Input Image

Feature Detector

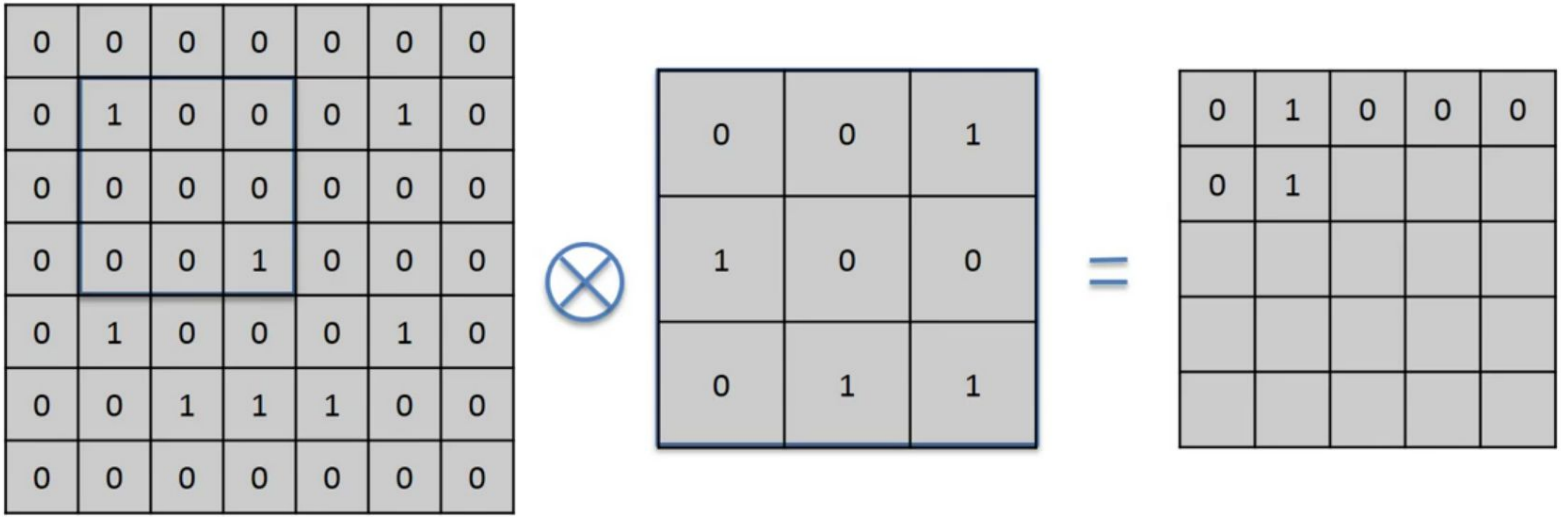
Feature Map



Input Image

Feature Detector

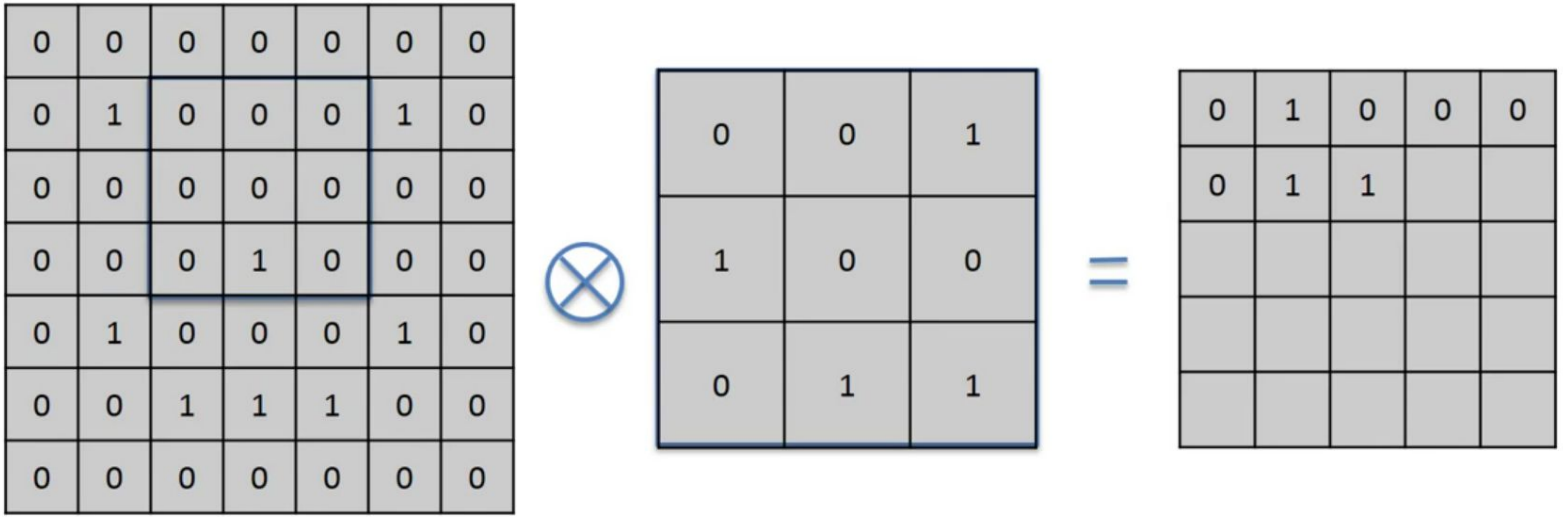
Feature Map



Input Image

Feature Detector

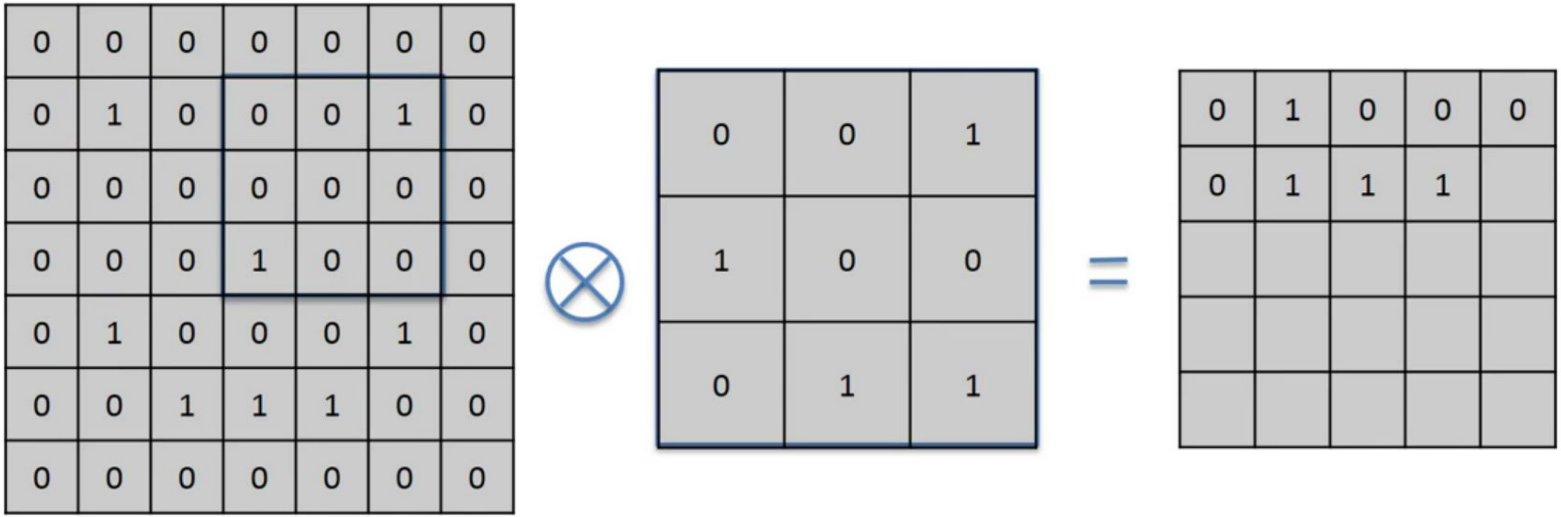
Feature Map

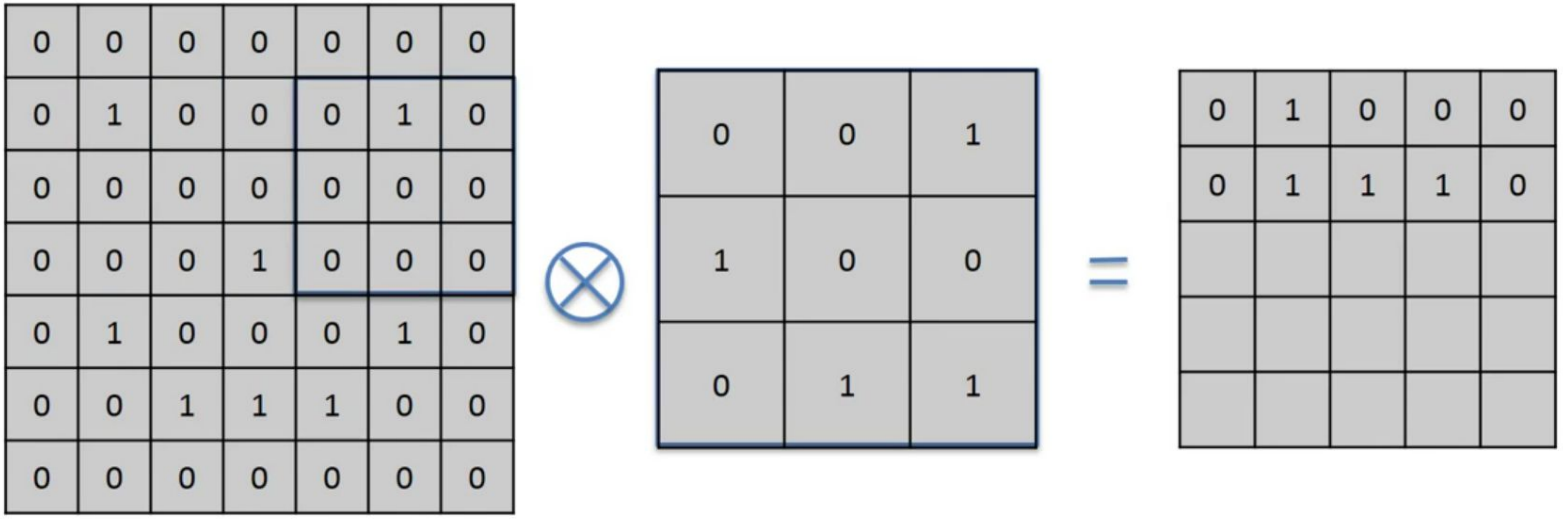


Input Image

Feature Detector

Feature Map

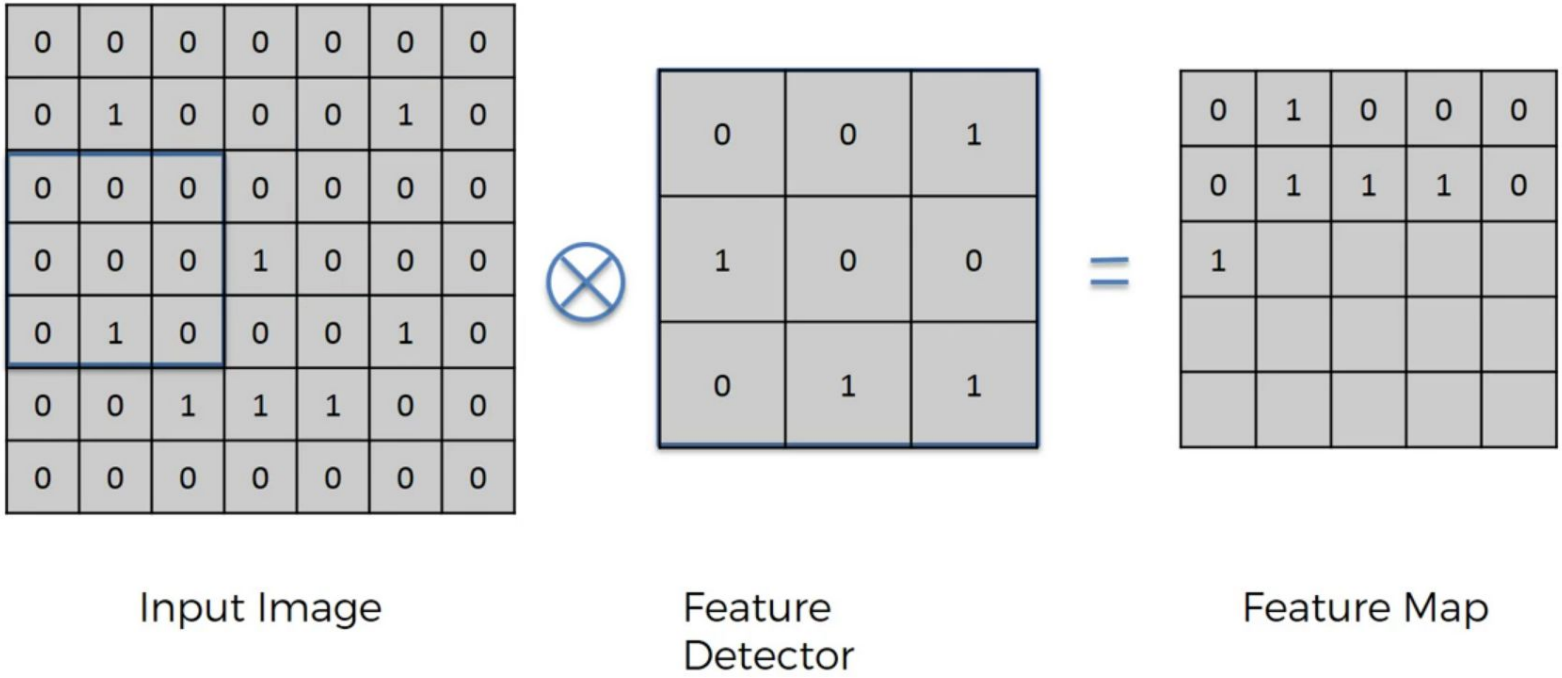

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1
Input Image
Feature Detector
Feature Map

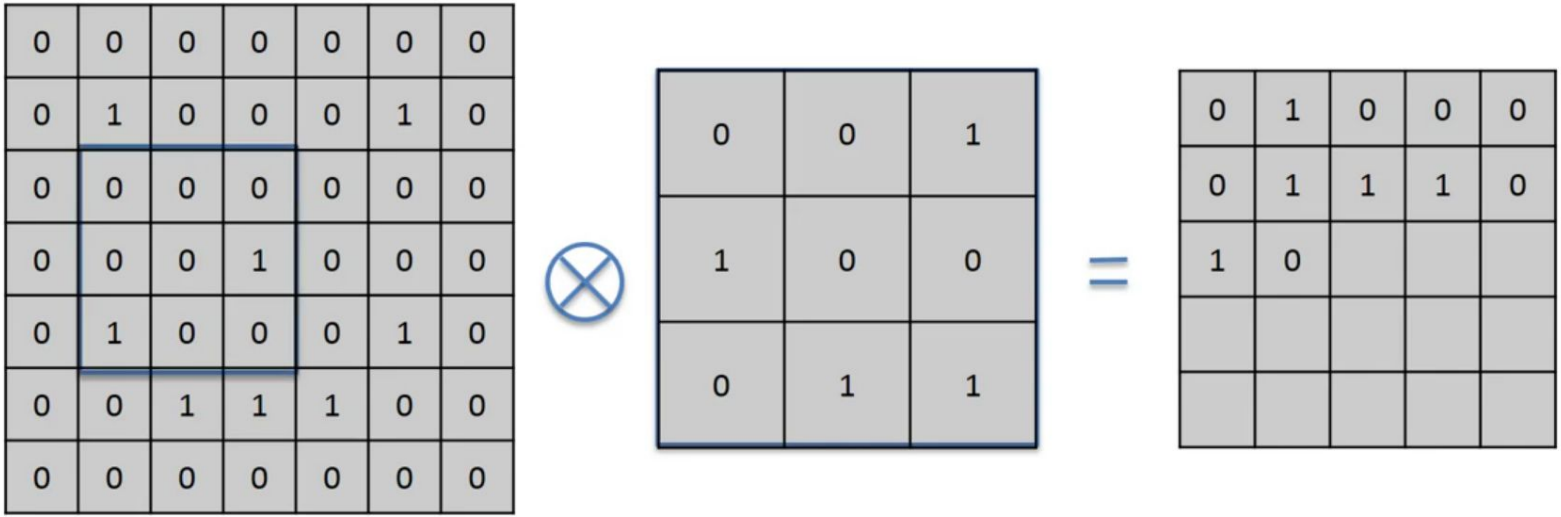


Input Image

Feature Detector

Feature Map

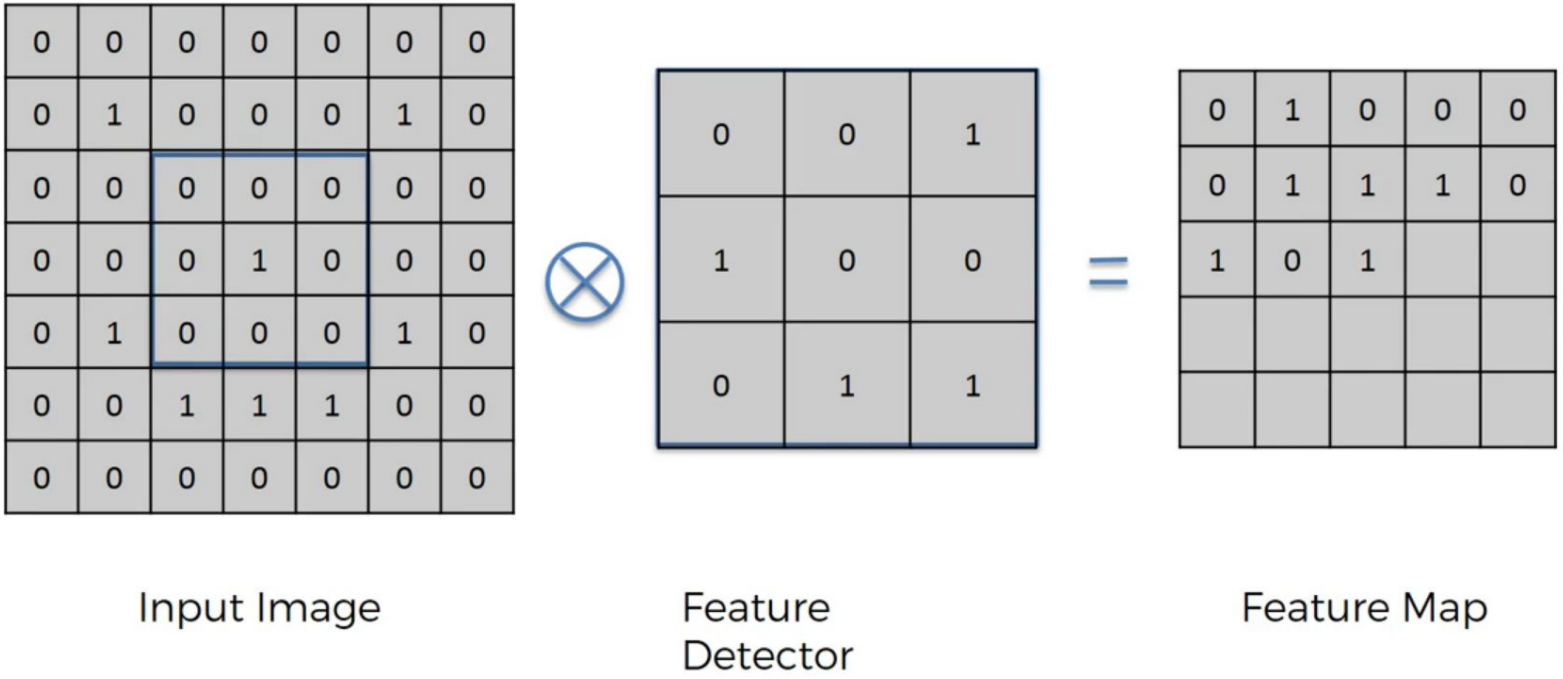

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1
Input Image
Feature Detector
Feature Map

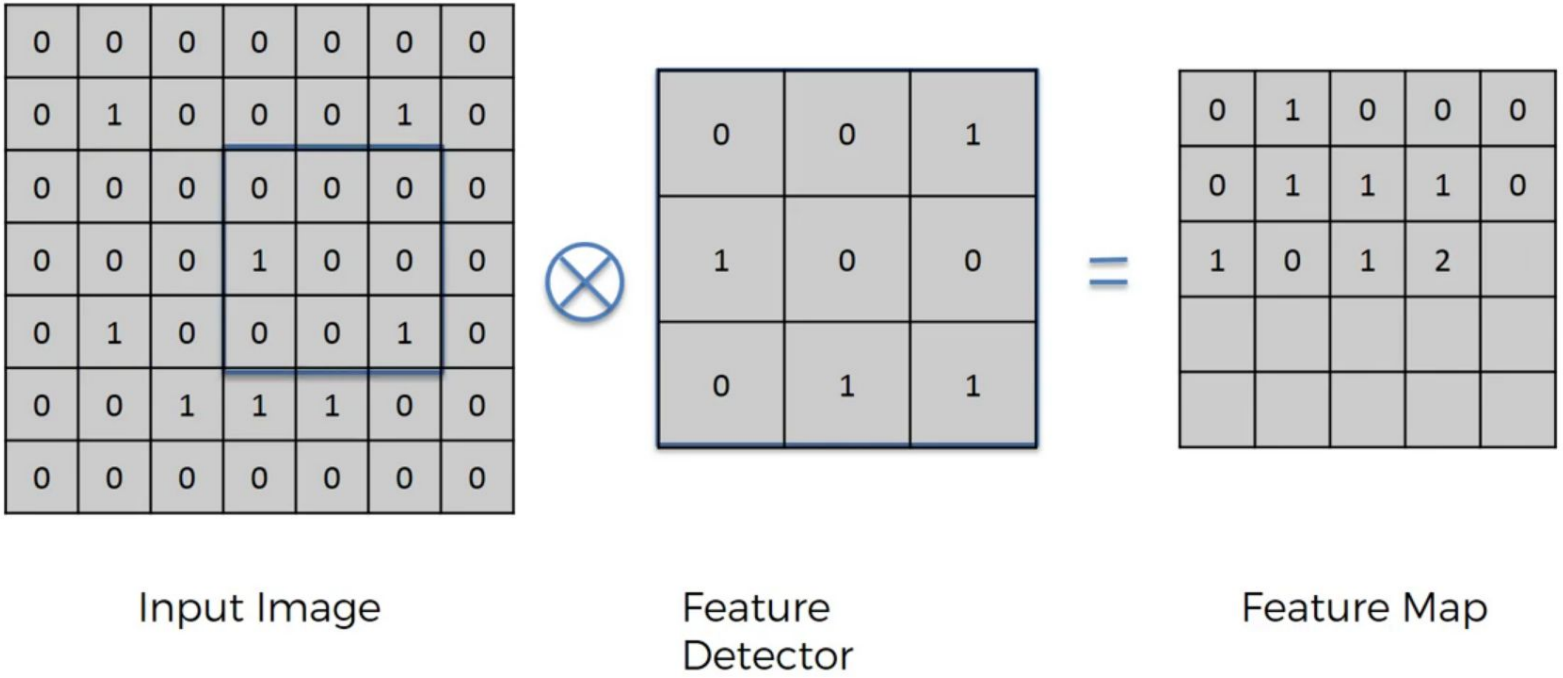


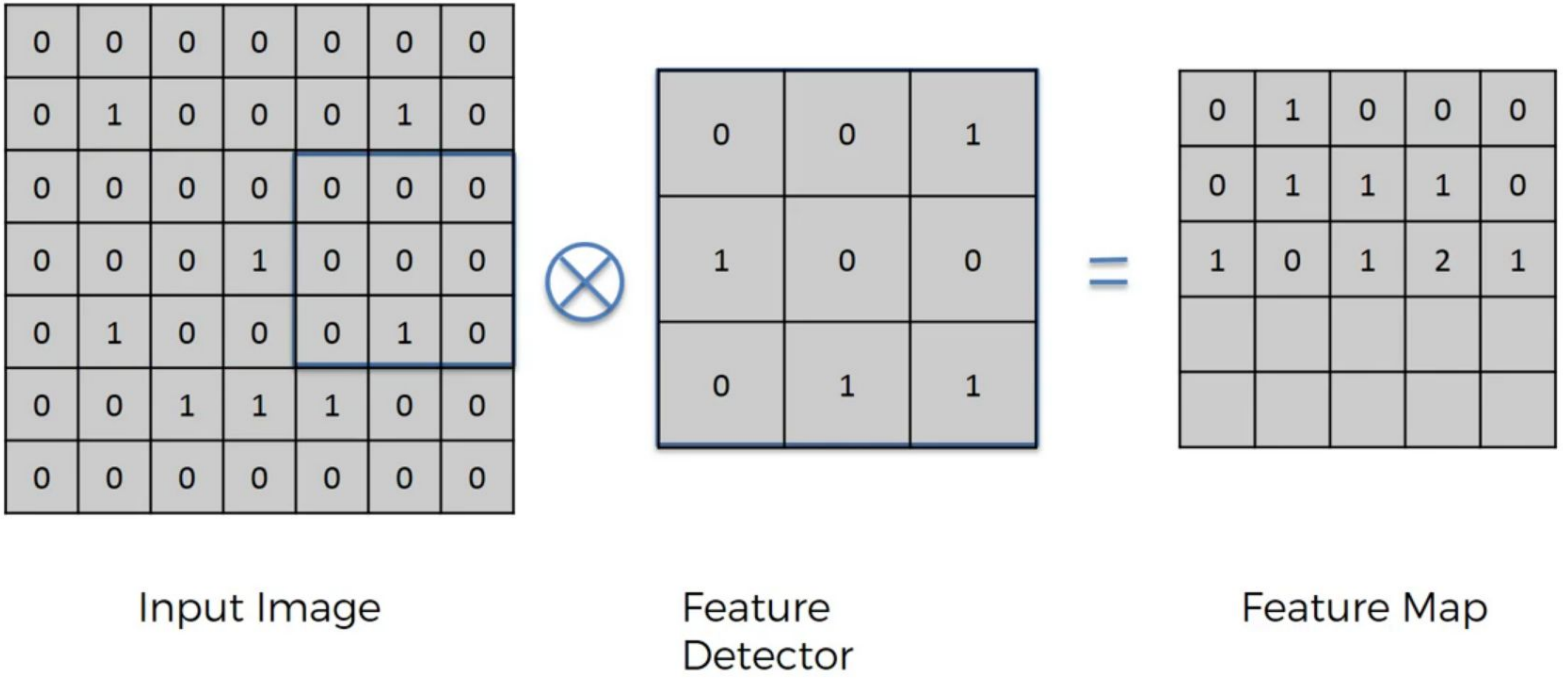
Input Image

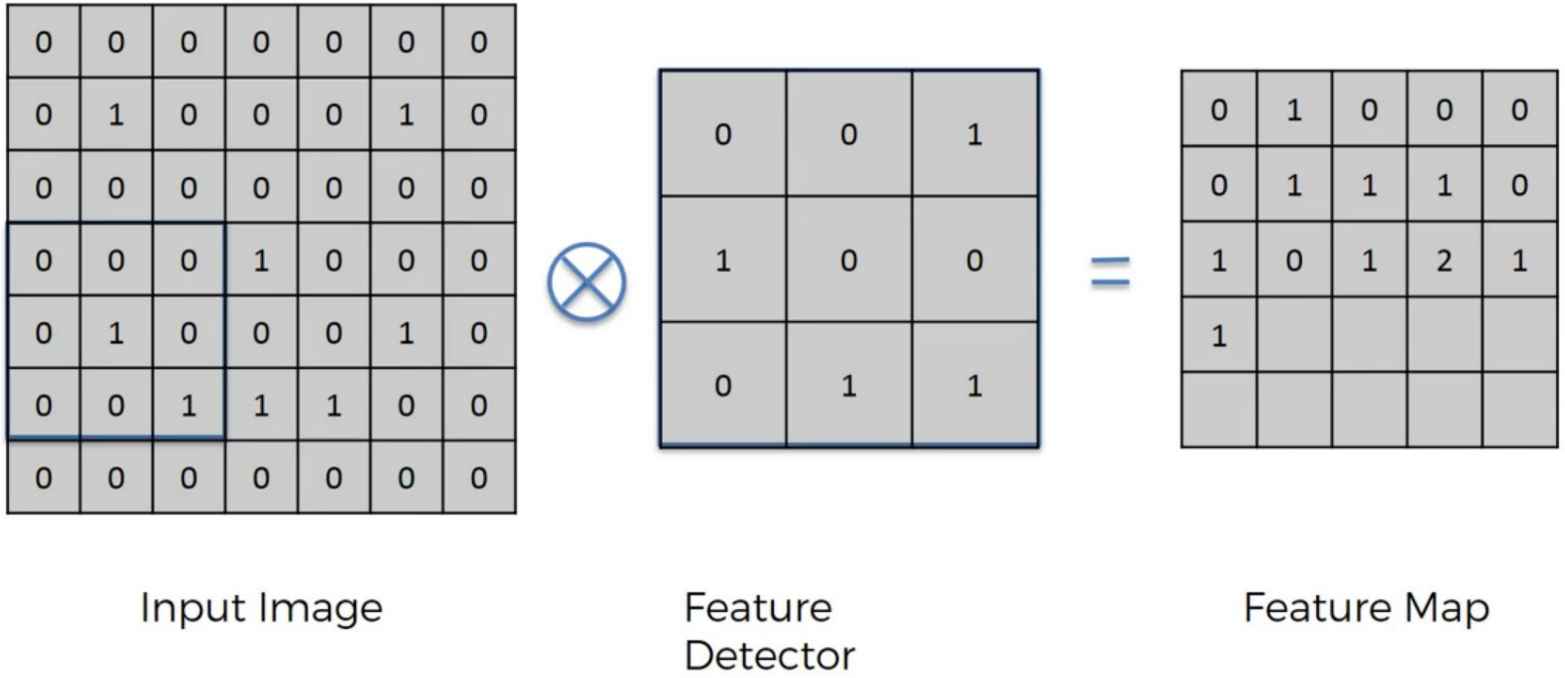
Feature Detector

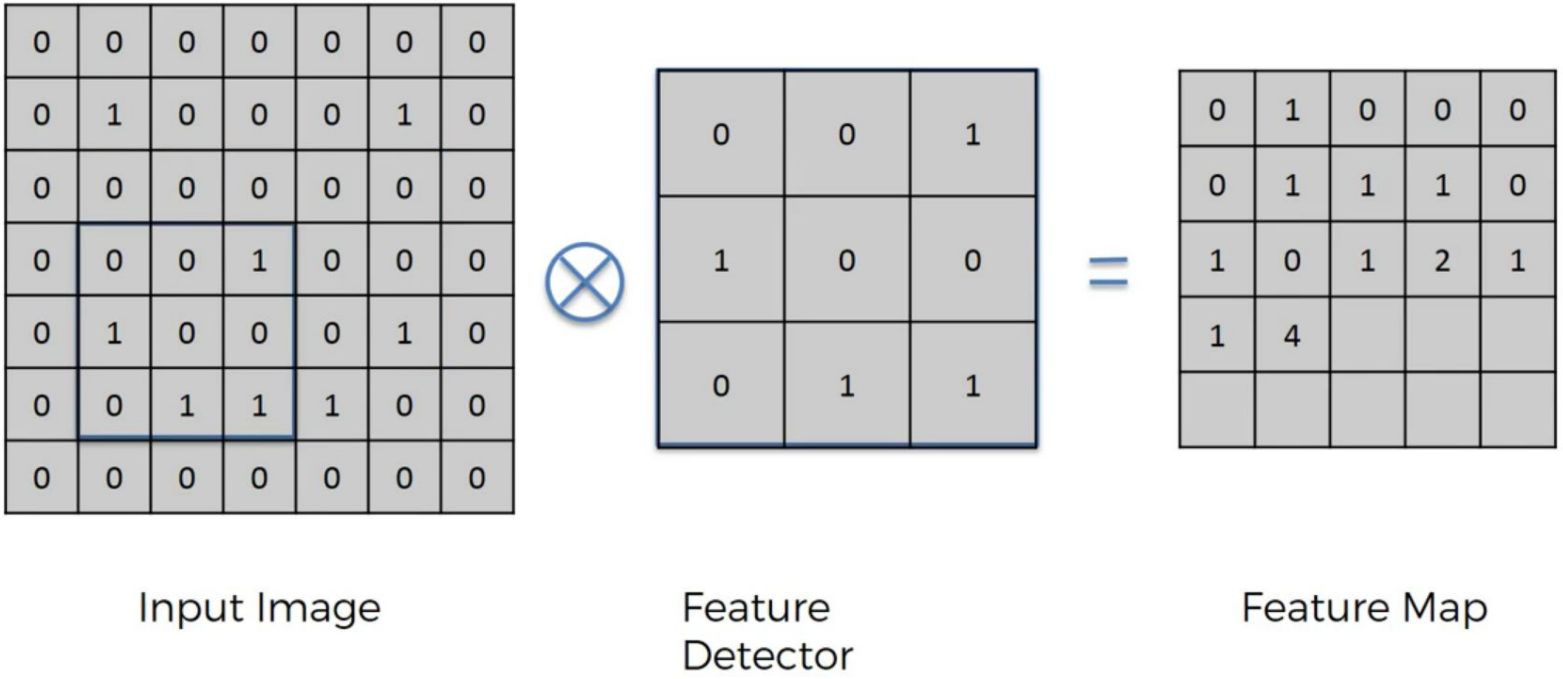
Feature Map

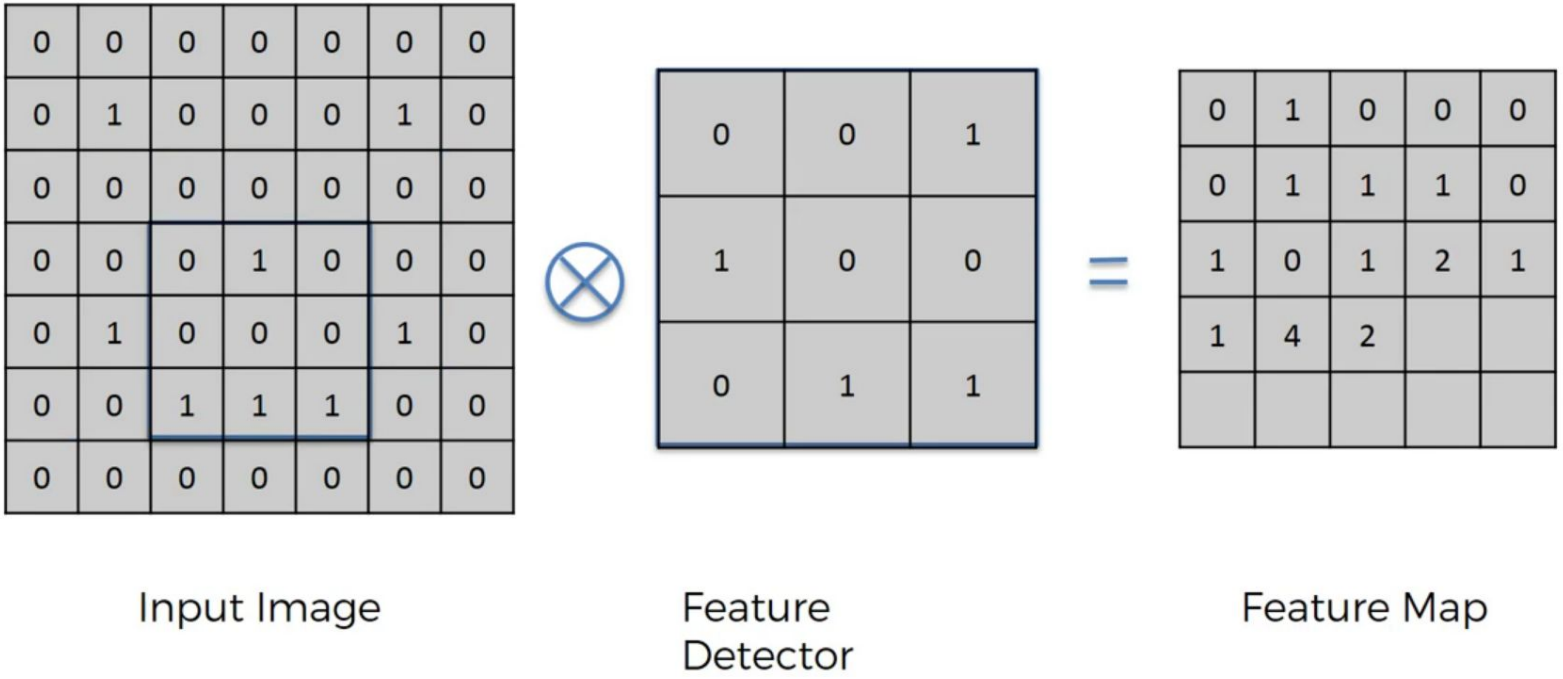

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1
Input Image
Feature Detector
Feature Map

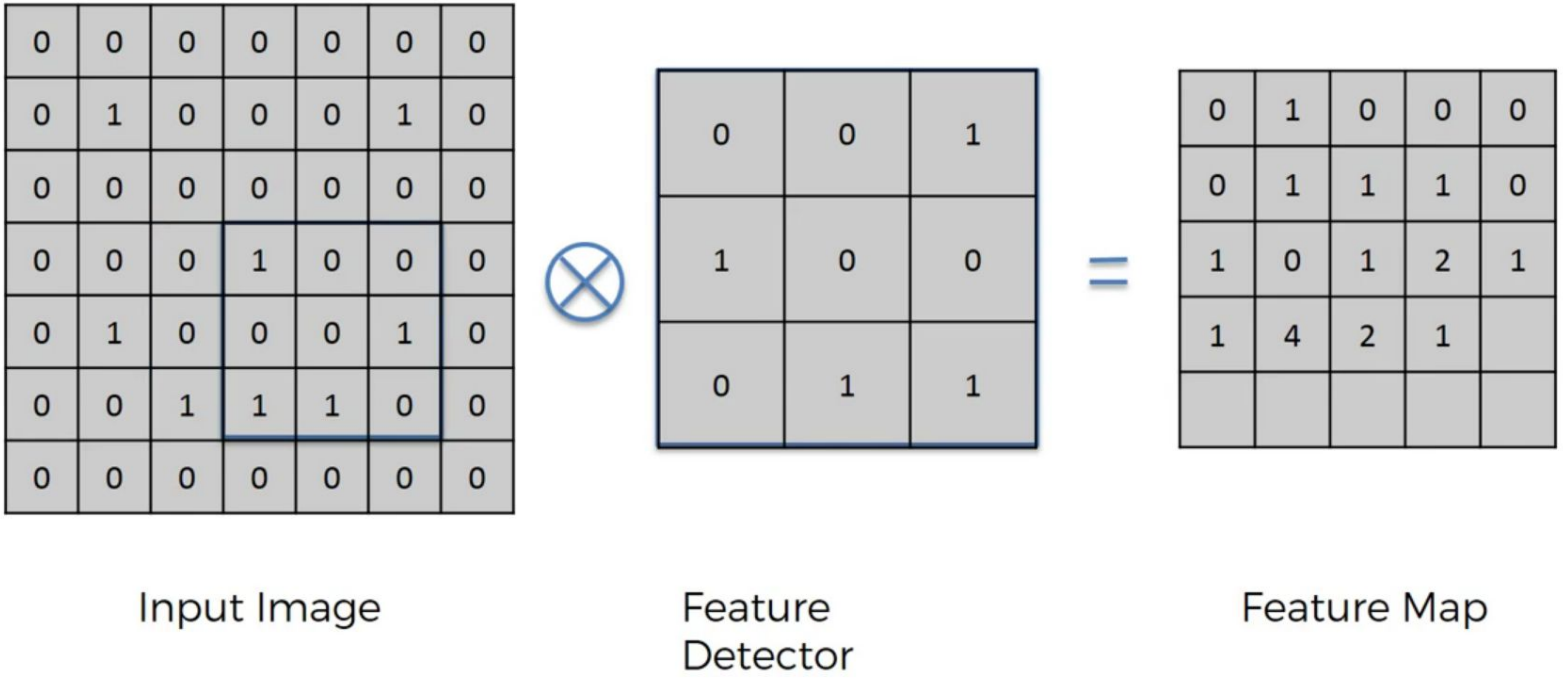

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1 2
Input Image
Feature Detector
Feature Map

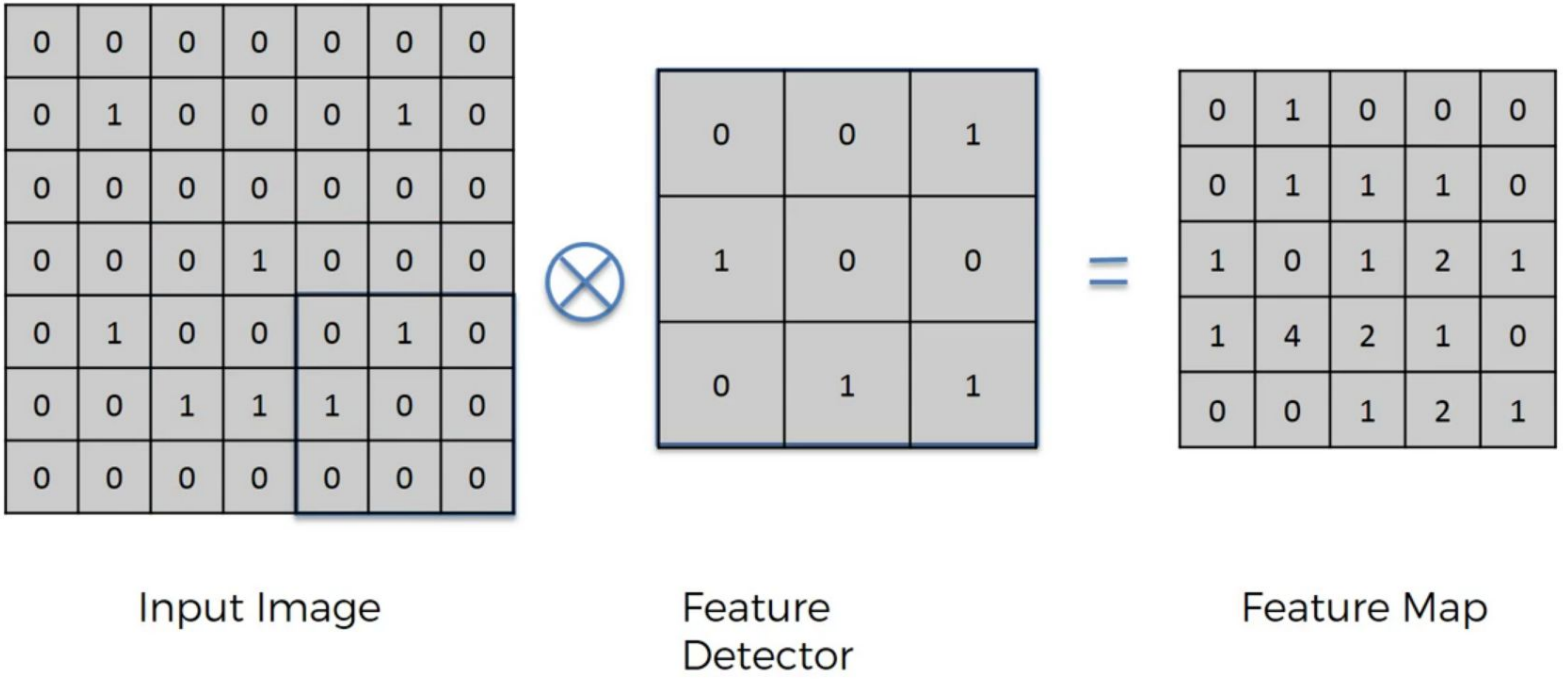

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
Input Image
Feature Detector
Feature Map

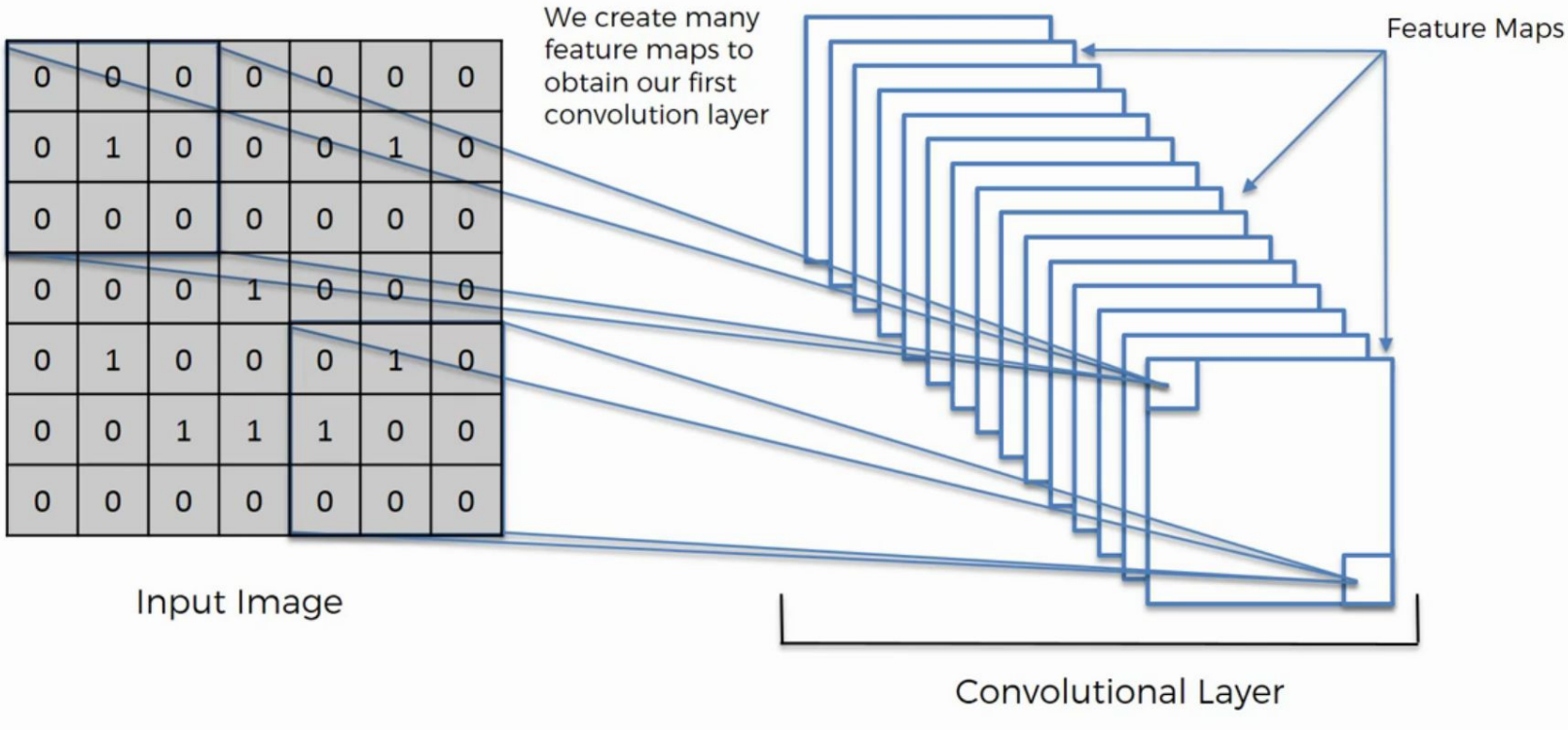

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1
Input Image
Feature Detector
Feature Map

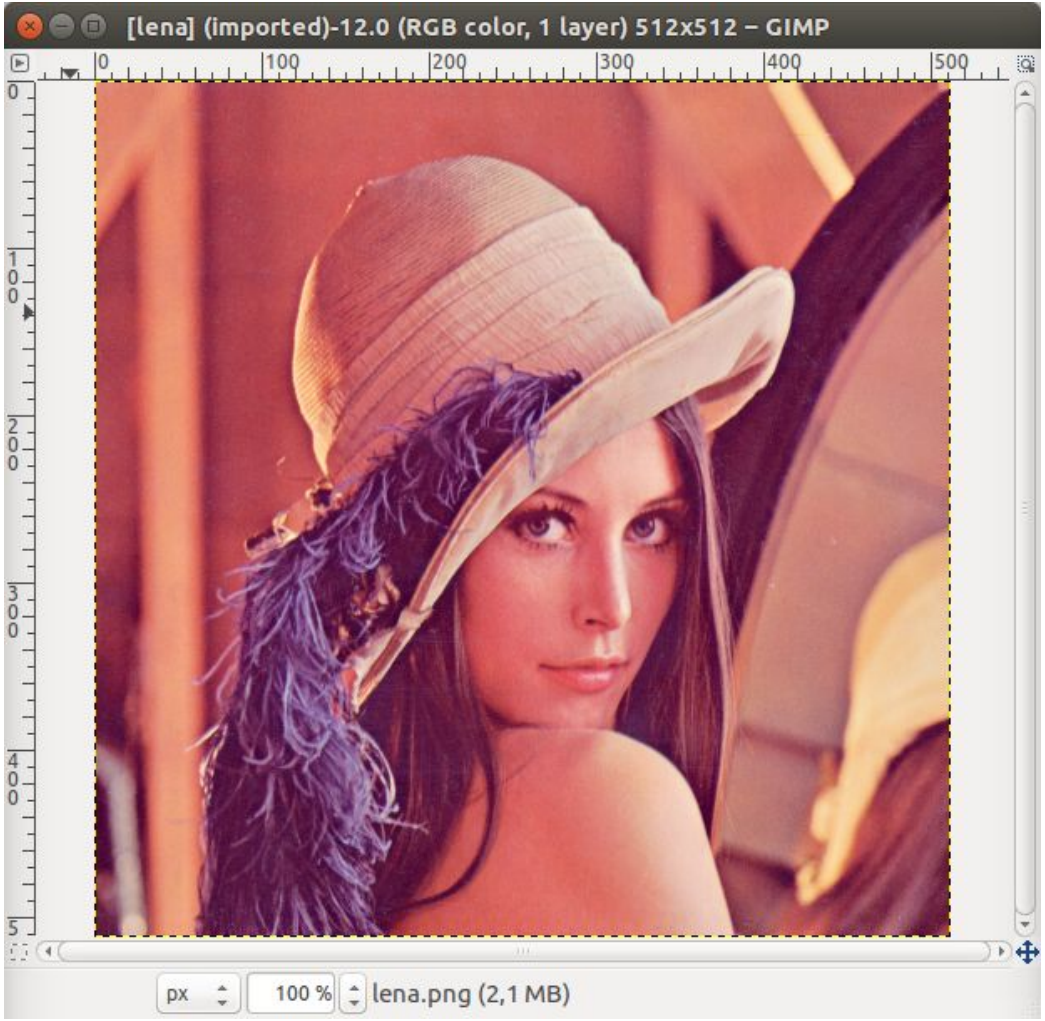

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4
Input Image
Feature Detector
Feature Map

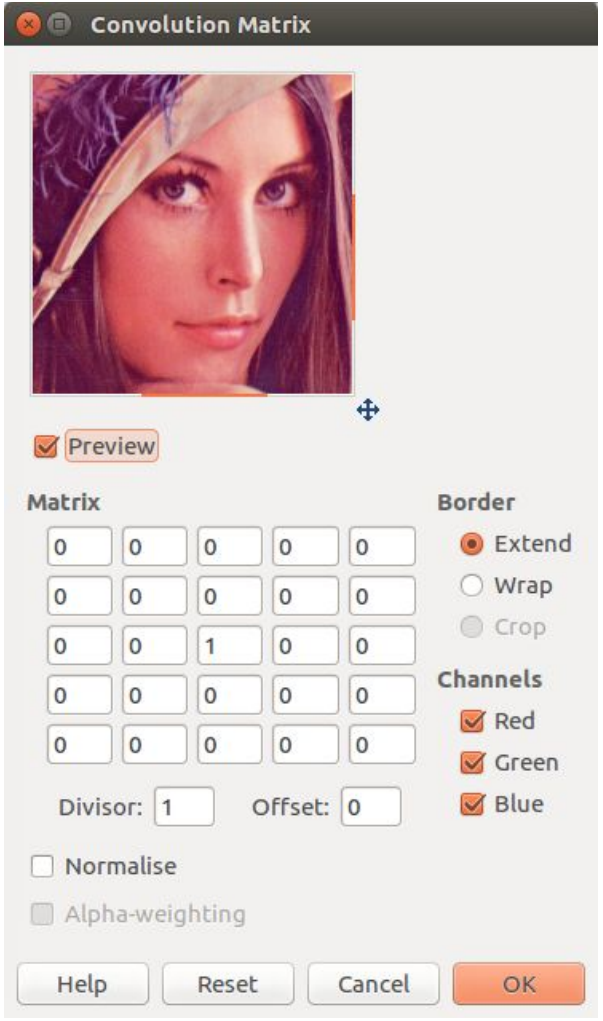

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2
Input Image
Feature Detector
Feature Map

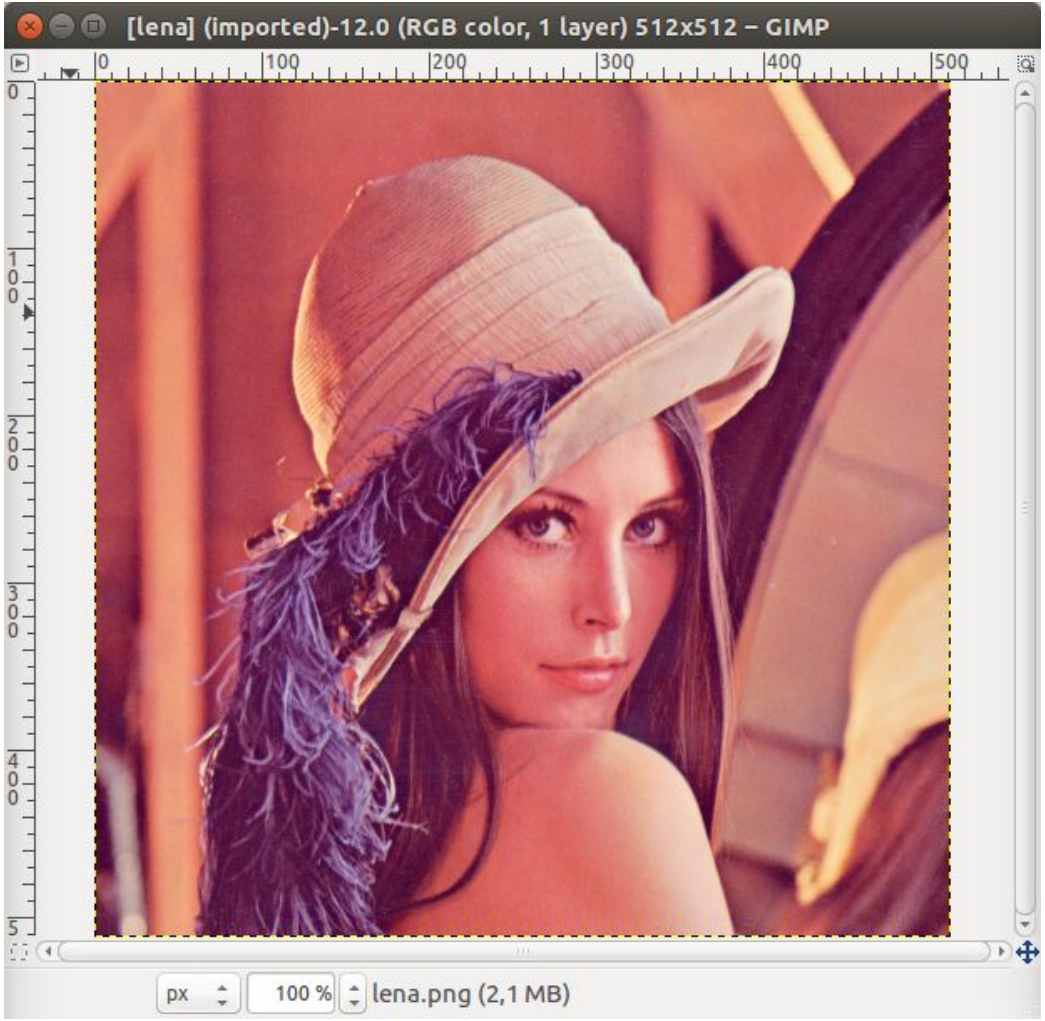

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1
Input Image
Feature Detector
Feature Map

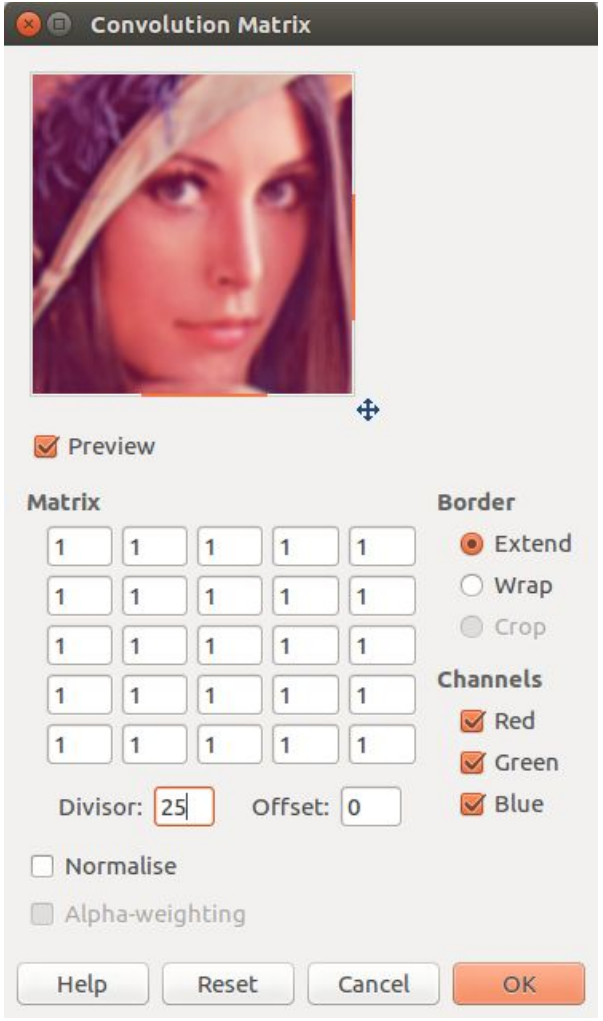

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 1
1 0 0
0 1 1
0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Input Image
Feature Detector
Feature Map

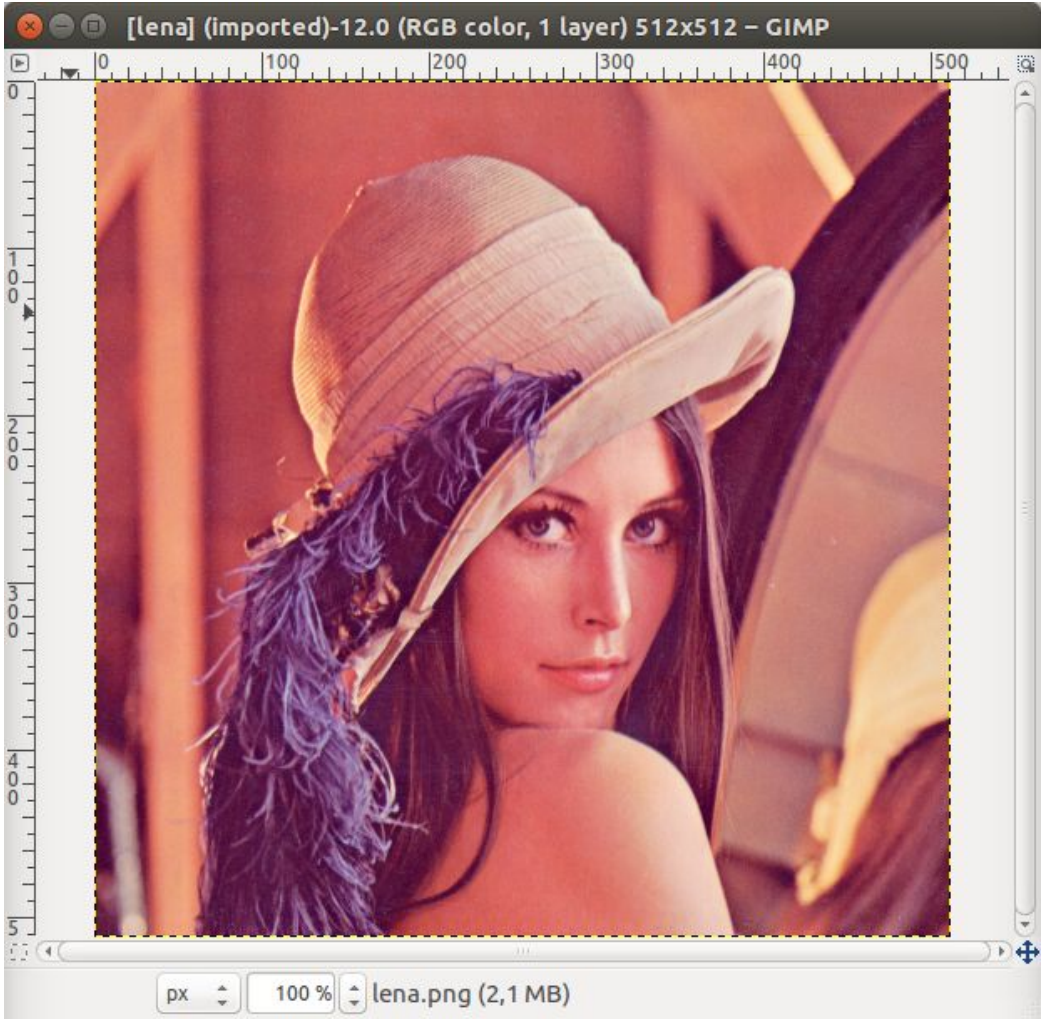

We create many feature maps to obtain our first convolution layer
Feature Maps
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolutional Layer

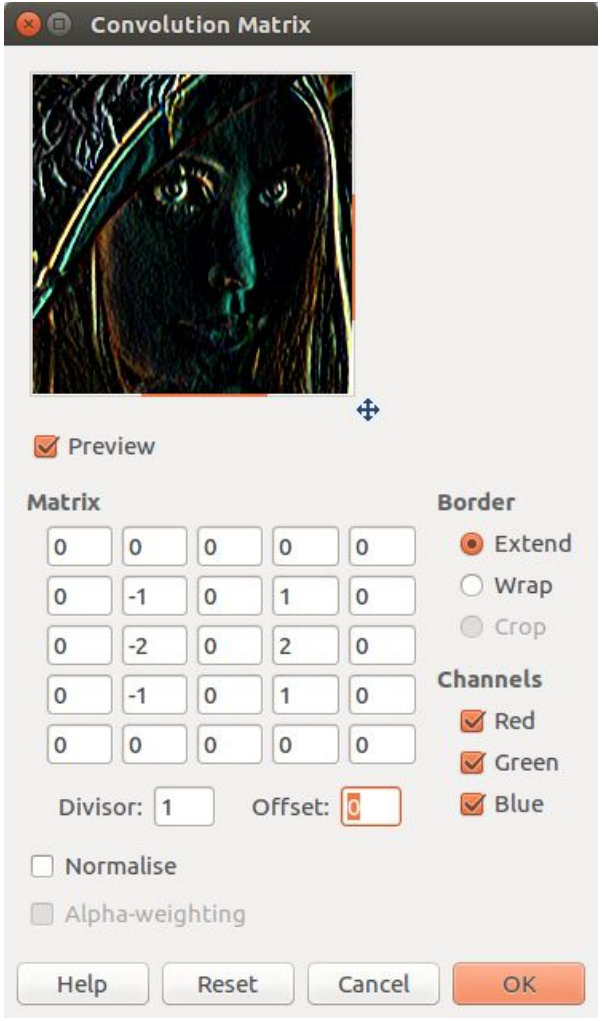

[lena] (imported)-12.0 (RGB color, 1 layer) 512x512 – GIMP
0
100
200
300
400
500
px
100 %
lena.png (2,1 MB)

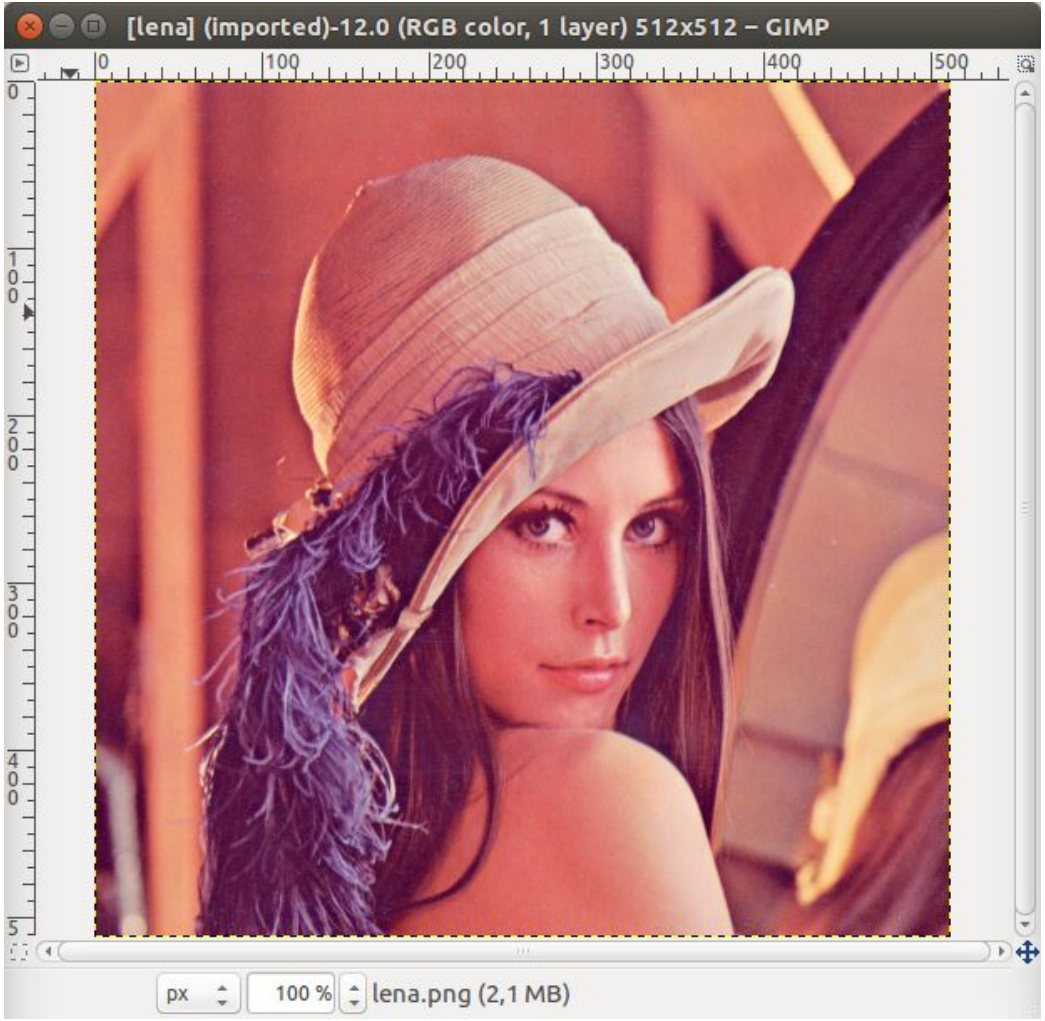

Convolution Matrix
Preview
Matrix
0 0 0 0 0
0 0 0 0 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
Divisor: 1
Offset: 0
Normalise
Alpha-weighting
Border
Extend
Wrap
Crop
Channels
Red
Green
Blue
Help
Reset
Cancel
OK

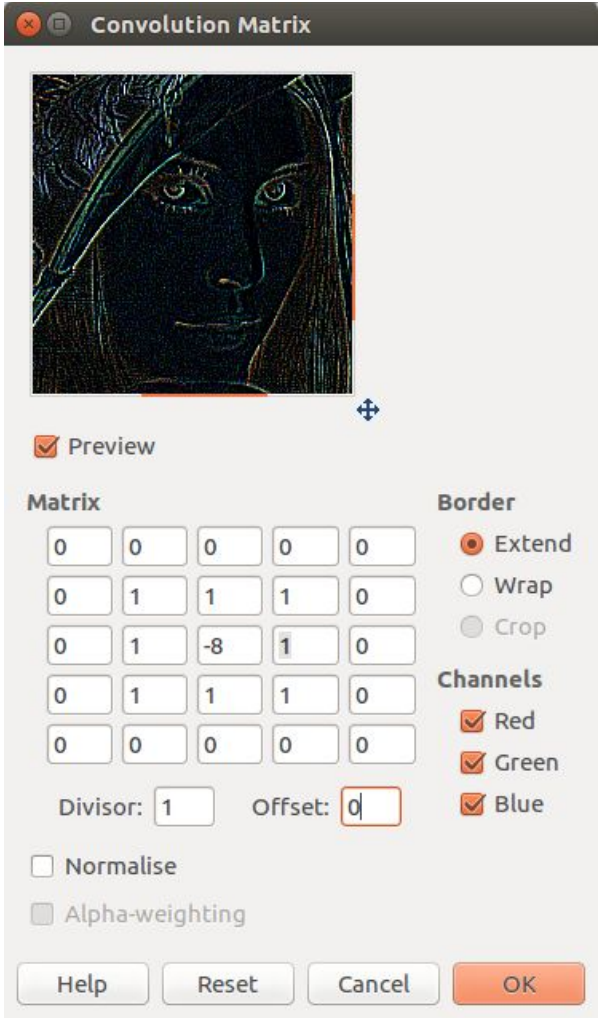

[lena] (imported)-12.0 (RGB color, 1 layer) 512x512 – GIMP
0
100
200
300
400
500
px
100 %
lena.png (2,1 MB)

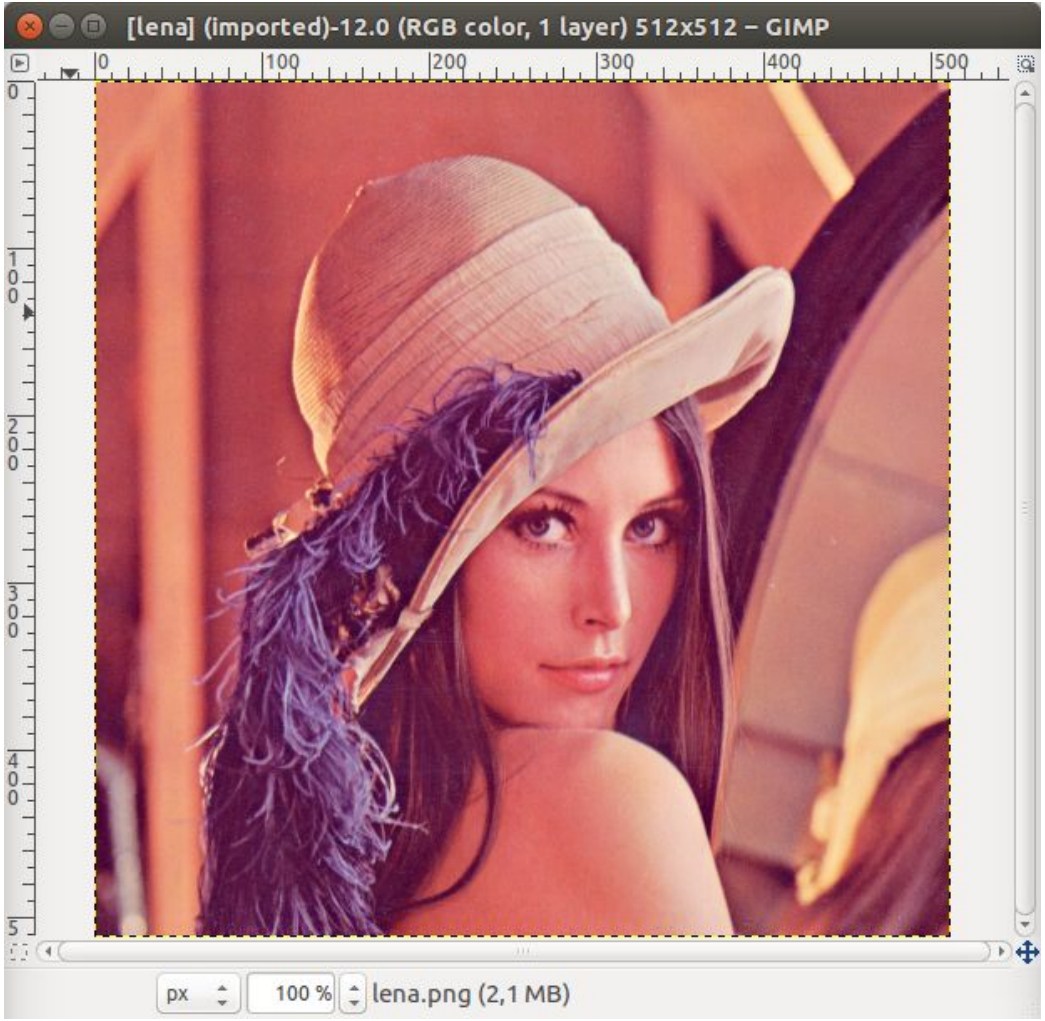

Convolution Matrix
Preview
Matrix
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
Divisor: 25
Offset: 0
Normalise
Alpha-weighting
Border
Extend
Wrap
Crop
Channels
Red
Green
Blue
Help
Reset
Cancel
OK

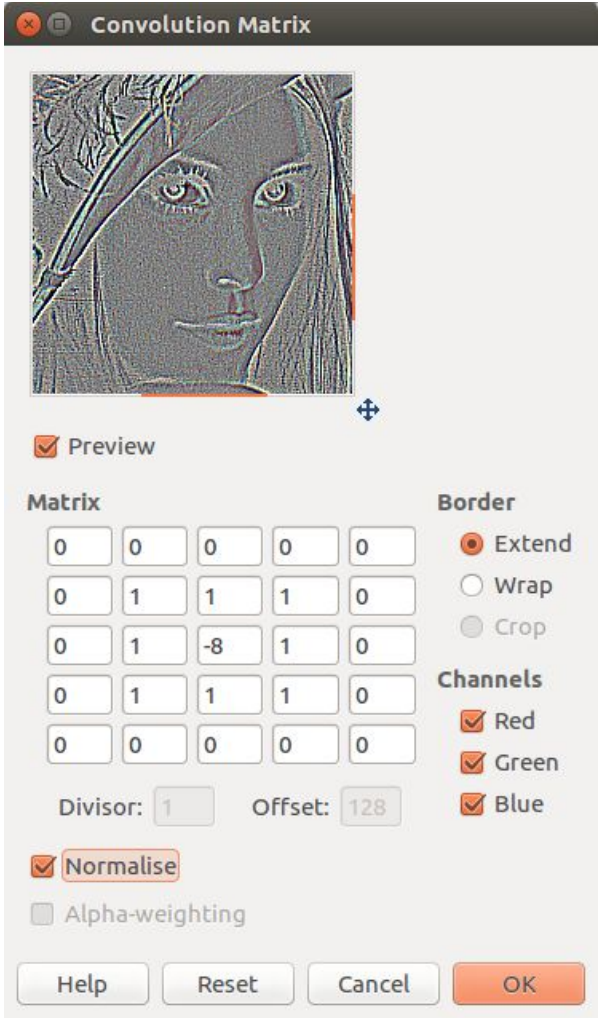

[lena] (imported)-12.0 (RGB color, 1 layer) 512x512 – GIMP
px
100 %
lena.png (2,1 MB)

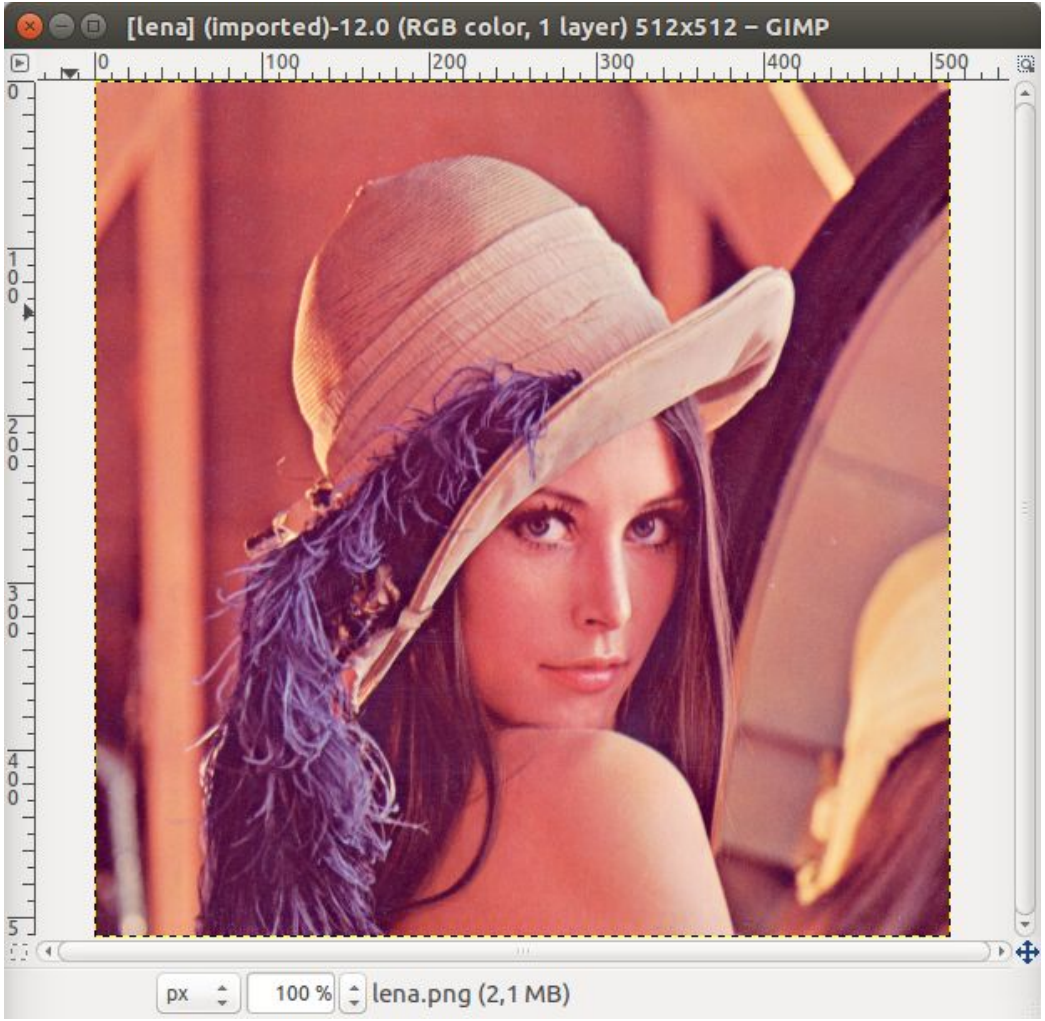

Convolution Matrix
Preview
Matrix
0 0 0 0 0
0 -1 0 1 0
0 -2 0 2 0
0 -1 0 1 0
0 0 0 0 0
Divisor: 1
Offset: 0
Normalise
Alpha-weighting
Border
Extend
Wrap
Crop
Channels
Red
Green
Blue
Help
Reset
Cancel
OK

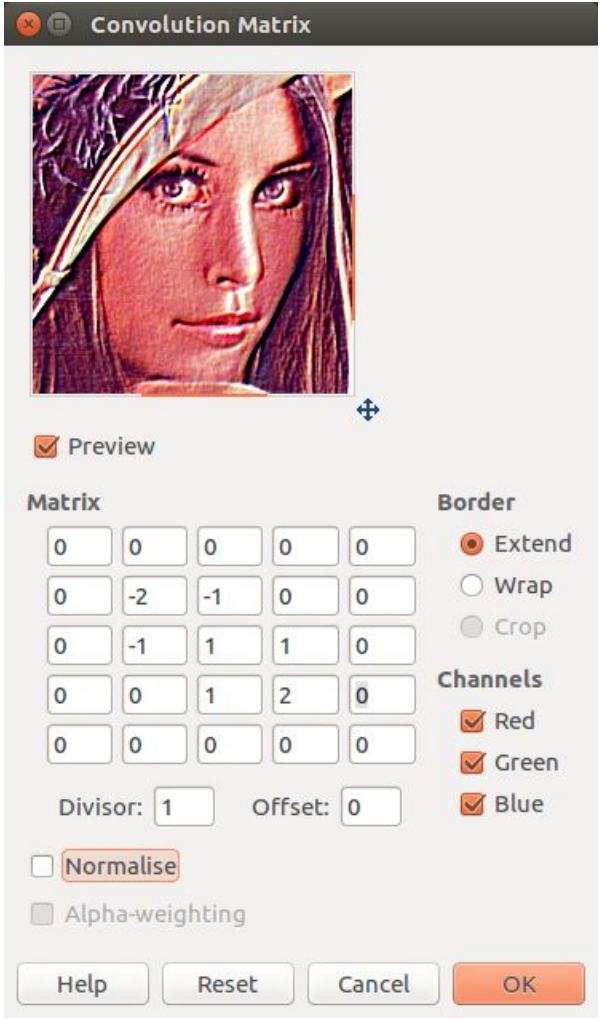

[lena] (imported)-12.0 (RGB color, 1 layer) 512x512 – GIMP
0
100
200
300
400
500
px
100 %
lena.png (2,1 MB)


Convolution Matrix
Preview
Matrix
0 0 0 0 0
0 1 1 1 0
0 1 -8 1 0
0 1 1 1 0
0 0 0 0 0
Divisor: 1
Offset: 0
Normalise
Alpha-weighting
Border
Extend
Wrap
Crop
Channels
Red
Green
Blue
Help
Reset
Cancel
OK


[lena] (imported)-12.0 (RGB color, 1 layer) 512x512 – GIMP
0
100
200
300
400
500
px
100 %
lena.png (2,1 MB)


Convolution Matrix
Preview
Matrix
0 0 0 0 0
0 1 1 1 0
0 1 -8 1 0
0 1 1 1 0
0 0 0 0 0
Divisor: 1
Offset: 128
Normalise
Alpha-weighting
Border
Extend
Wrap
Crop
Channels
Red
Green
Blue
Help
Reset
Cancel
OK


[lena] (imported)-12.0 (RGB color, 1 layer) 512x512 – GIMP
px
100 %
lena.png (2,1 MB)


Convolution Matrix
Preview
Matrix
0 0 0 0 0
0 -2 -1 0 0
0 -1 1 1 0
0 0 1 2 0
0 0 0 0 0
Divisor: 1
Offset: 0
Normalise
Alpha-weighting
Border
Extend
Wrap
Crop
Channels
Red
Green
Blue
Help
Reset
Cancel
OK

Border

When the initial pixel is on a border, a part of kernel is out of image. You have to decide what filter must do:

From left: source image, Extend border, Wrap border, Crop border

Extend

This part of kernel is not taken into account.

Wrap

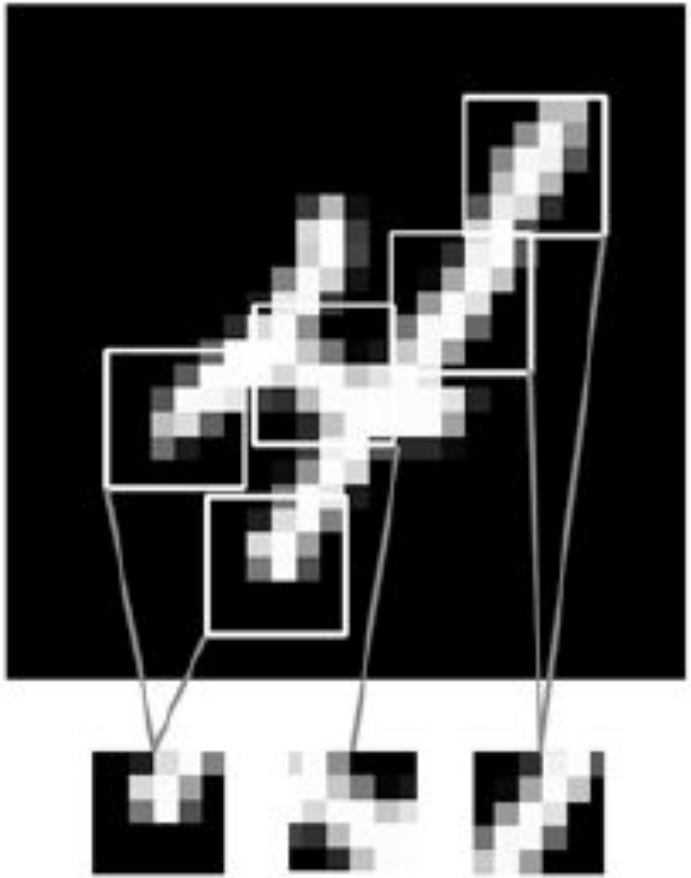
This part of kernel will study pixels of the opposite border, so pixels disappearing from one side reappear on the other side.

Crop

Pixels on borders are not modified, but they are cropped.

https://docs.gimp.org/en/plug-in-convmatrix.html

- Dense layers → global patterns.
- Convolution layers → local patterns.

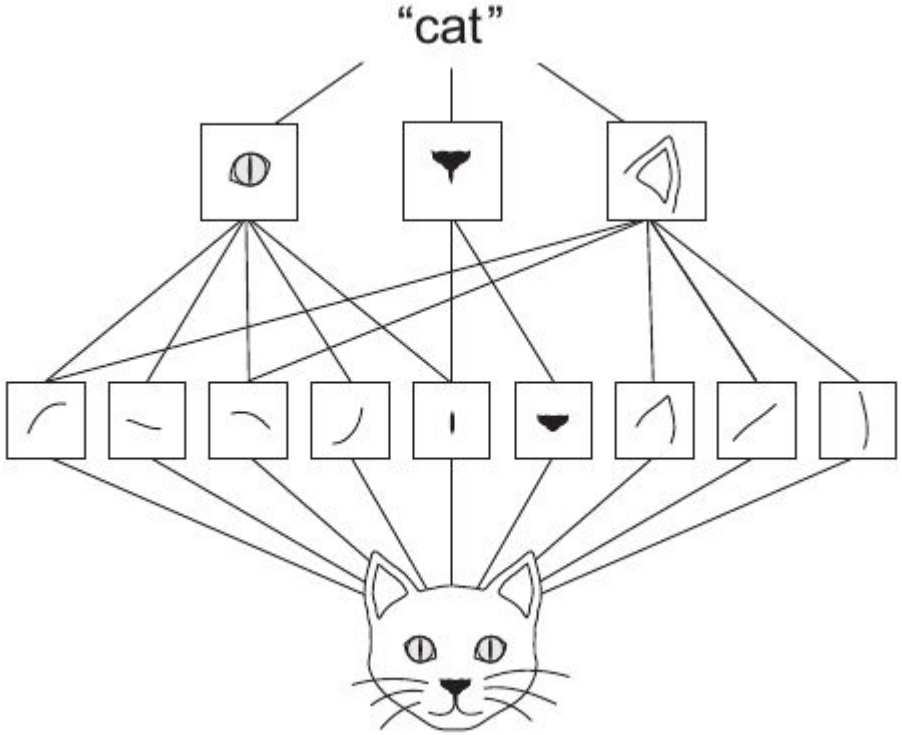


Convnets properties - Translation invariance

Cat

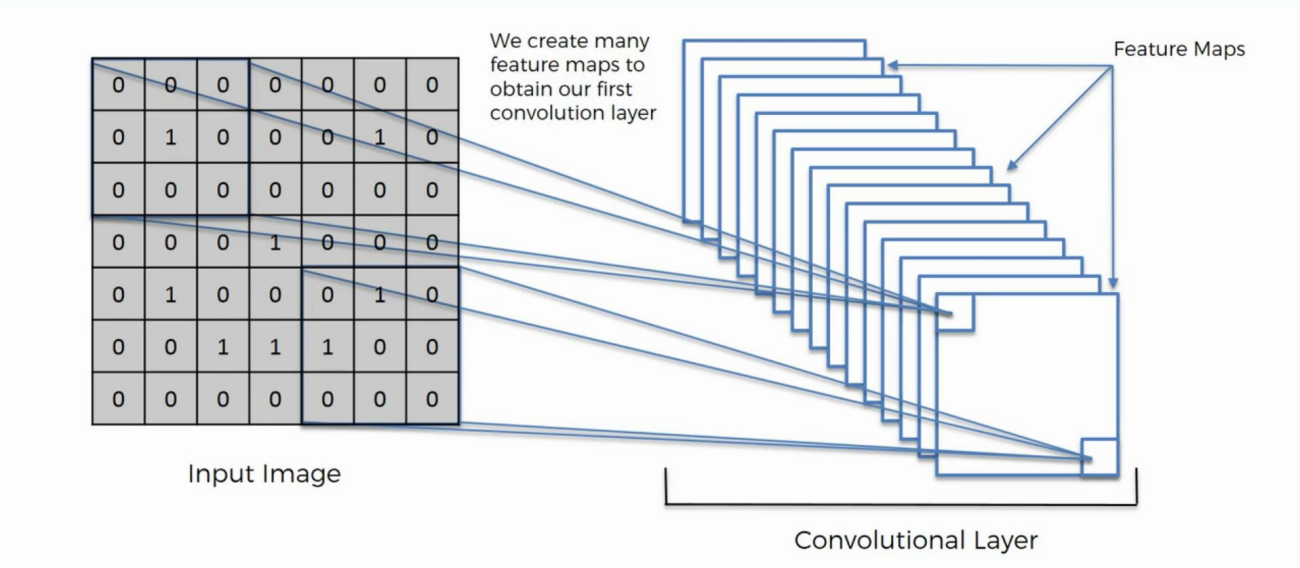
Cat

Convnets properties - Hierarchy



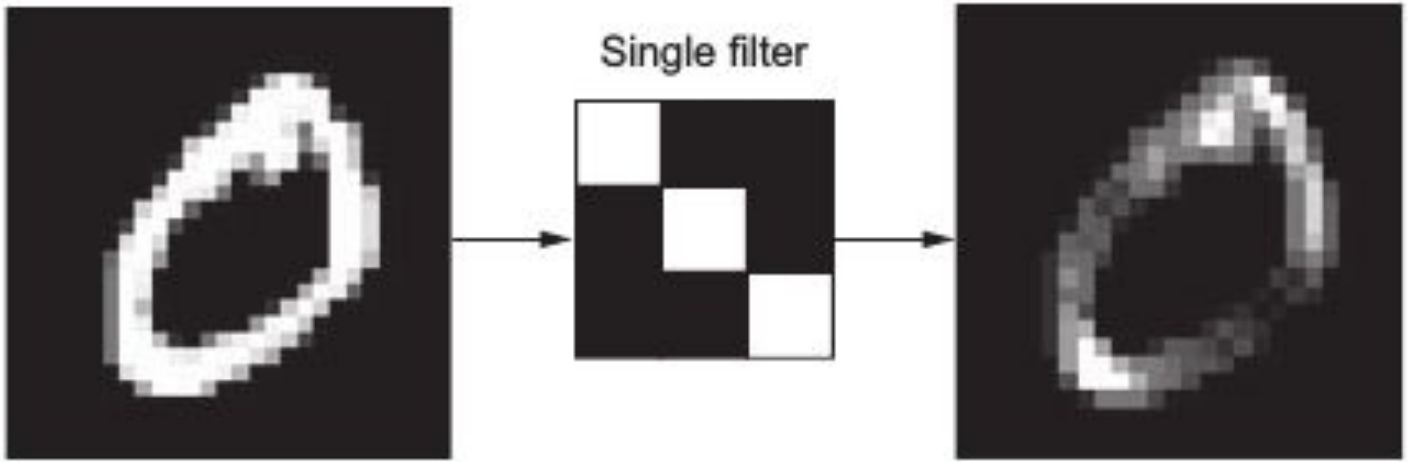
Convnets properties - operation

- Convolutions operate over 3D tensors (feature maps):
  - Height.
  - Width.
  - Depth.
- Output depth depends on the layer = filters.



Convnets properties - operation

- MNIST.
- 1st layers:
  - (28, 28, 1) → (26, 26, 32)



Convnets properties - definition

1. Size of the patches extracted from the inputs.
    - Typically 3 × 3 or 5 × 5.
2. Depth of the output feature map.
    - Number of filters computed by the convolution. The example started with a depth of 32 and ended with a depth of 64.

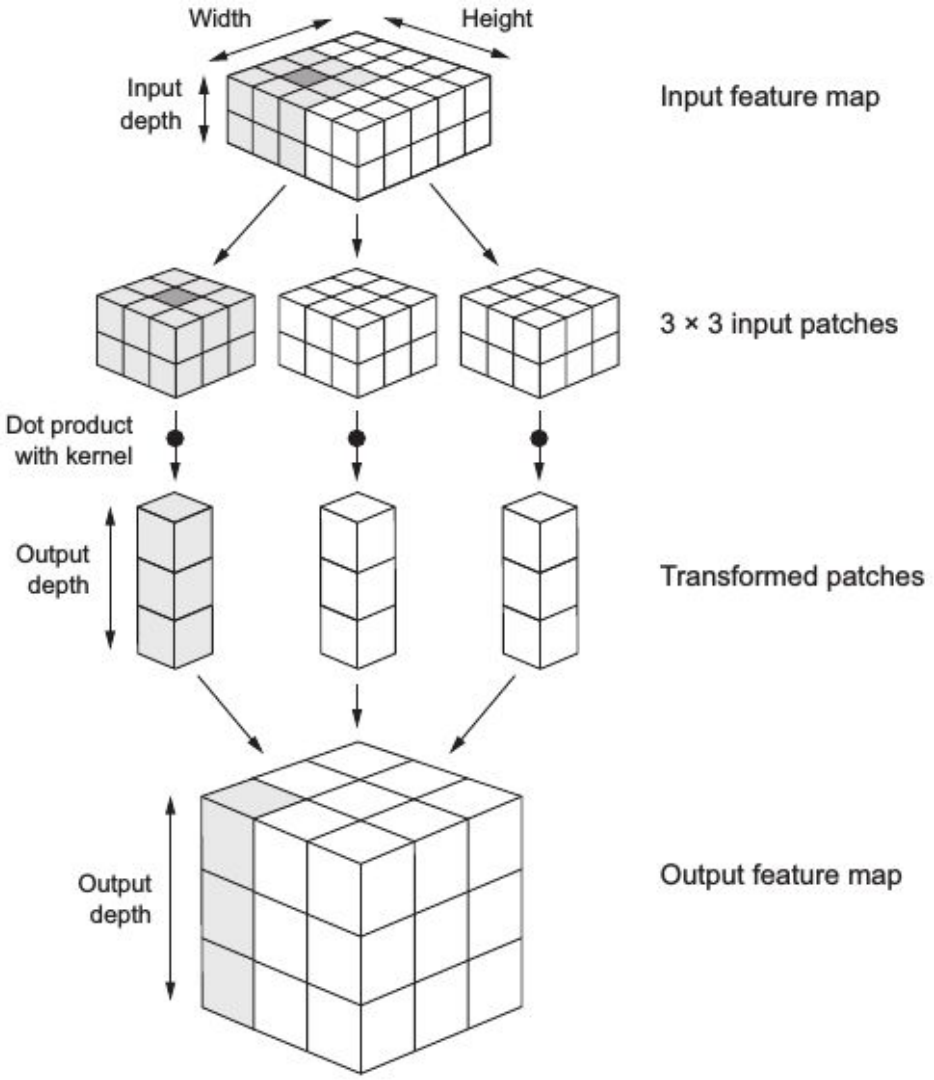
In KERAS:

```
Conv2D(output_depth, (window_height, window_width))
```

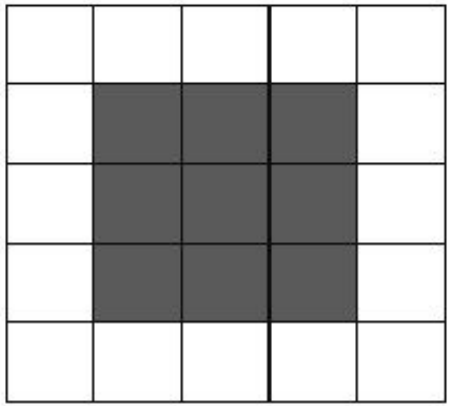
Convnets

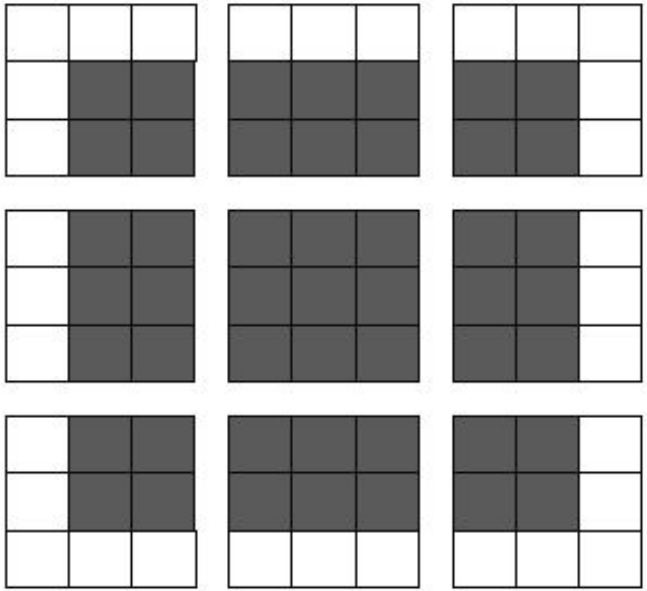
Output width and height may differ:

1. Border effects.
2. Strides.

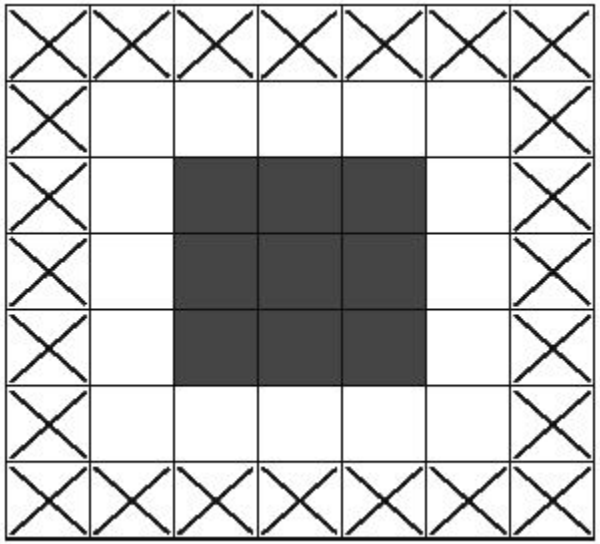


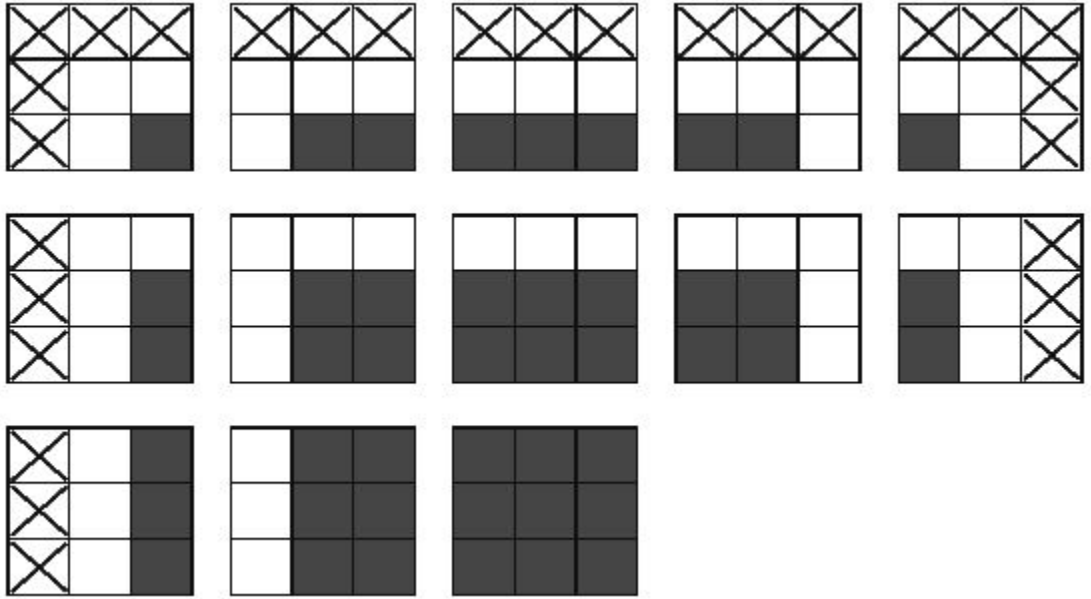
Border effects





Border effects - Padding





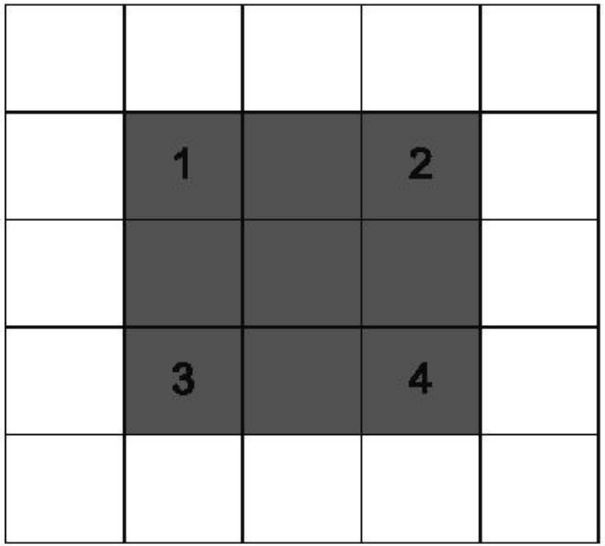
Border effects - Padding

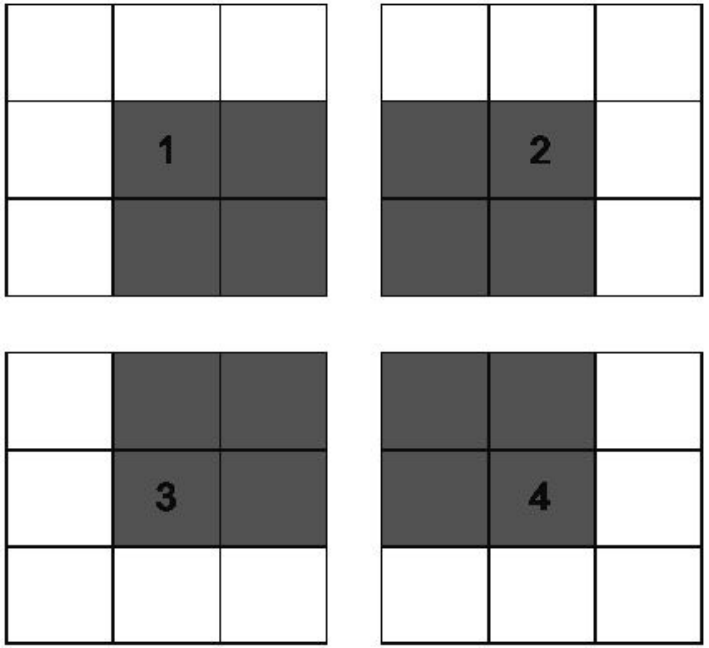
`Conv2D:`

1. “Valid” (default) - no padding (only valid window locations are used).
2. “Same” - “pad in such a way as to have an output with the same width and height as the input.

Strides

- 5x5 input.
- 3x3 convolution.
- 2x2 strides.





Max-Pooling

Image Source: Wikipedia

Image Source: Wikipedia

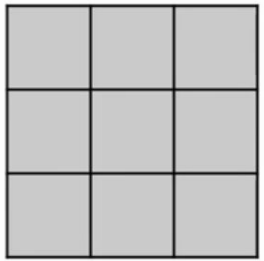
0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

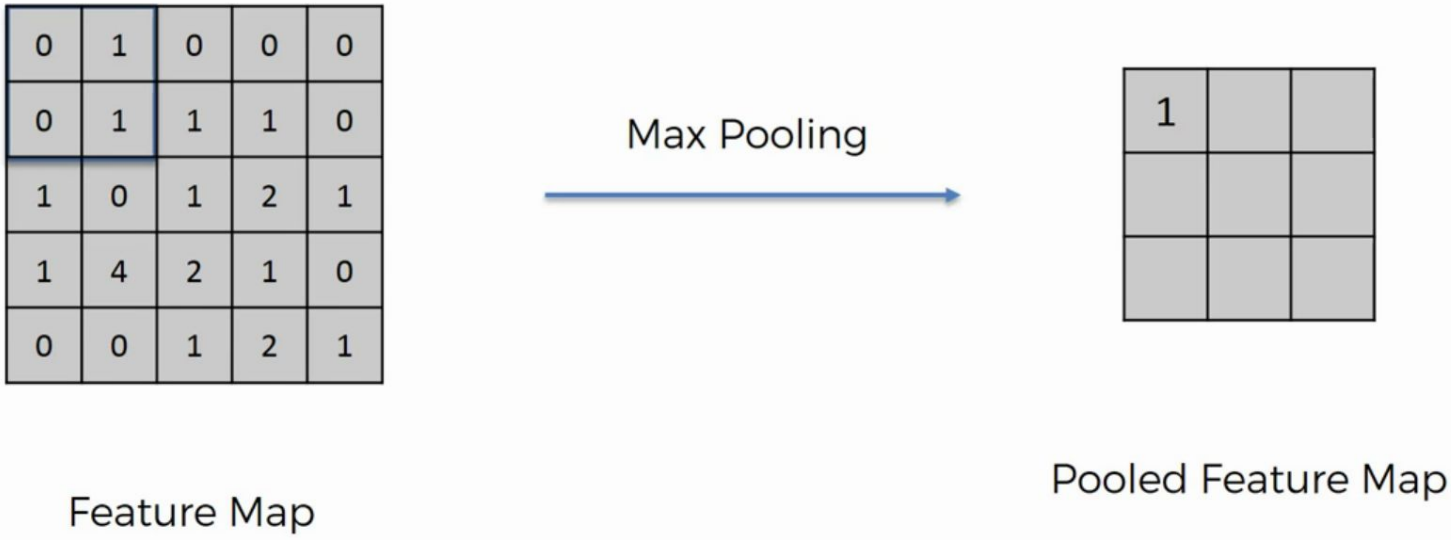
0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

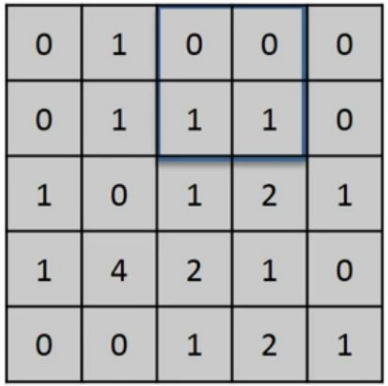
Feature Map

Max Pooling



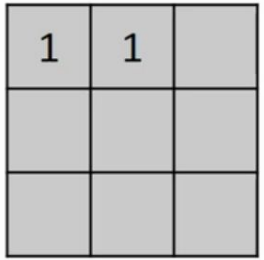
Pooled Feature Map


0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Max Pooling
1
Feature Map
Pooled Feature Map

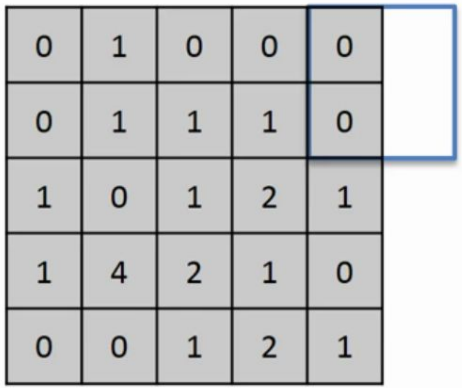


Feature Map

Max Pooling

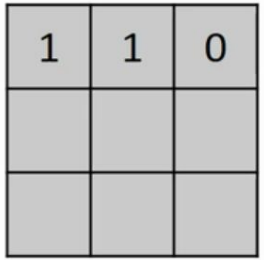


Pooled Feature Map

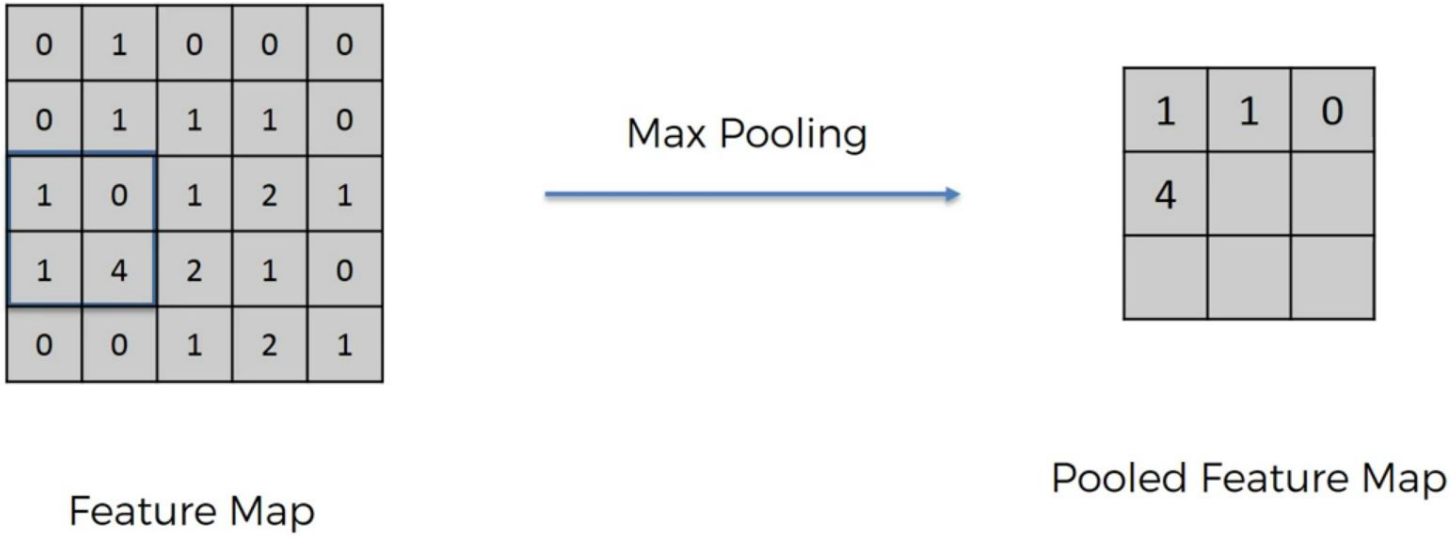


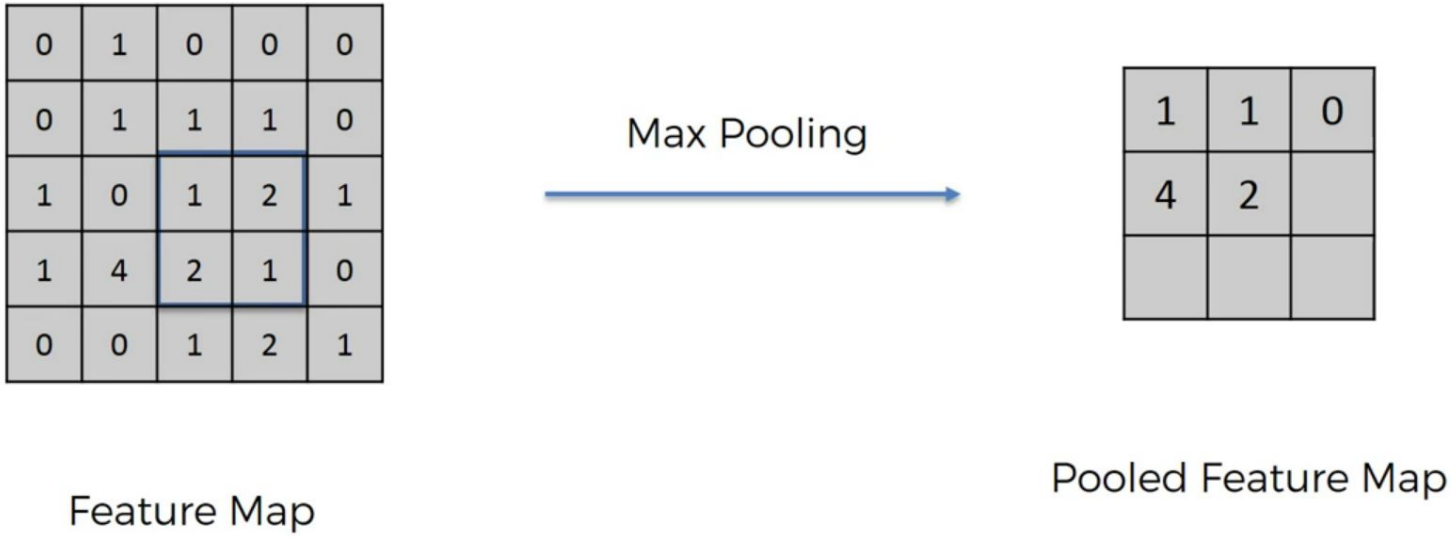
Feature Map

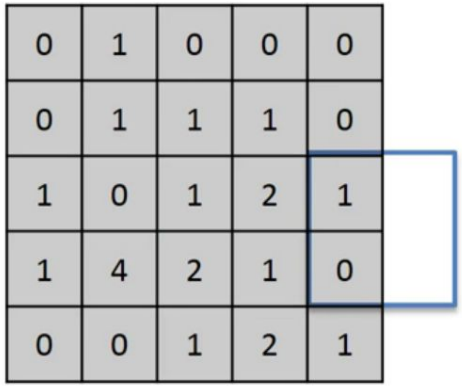
Max Pooling



Pooled Feature Map

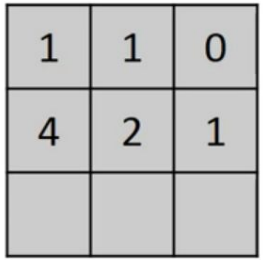

0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Max Pooling
1 1 0
4
Feature Map
Pooled Feature Map


0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Max Pooling
1 1 0
4 2
Feature Map
Pooled Feature Map

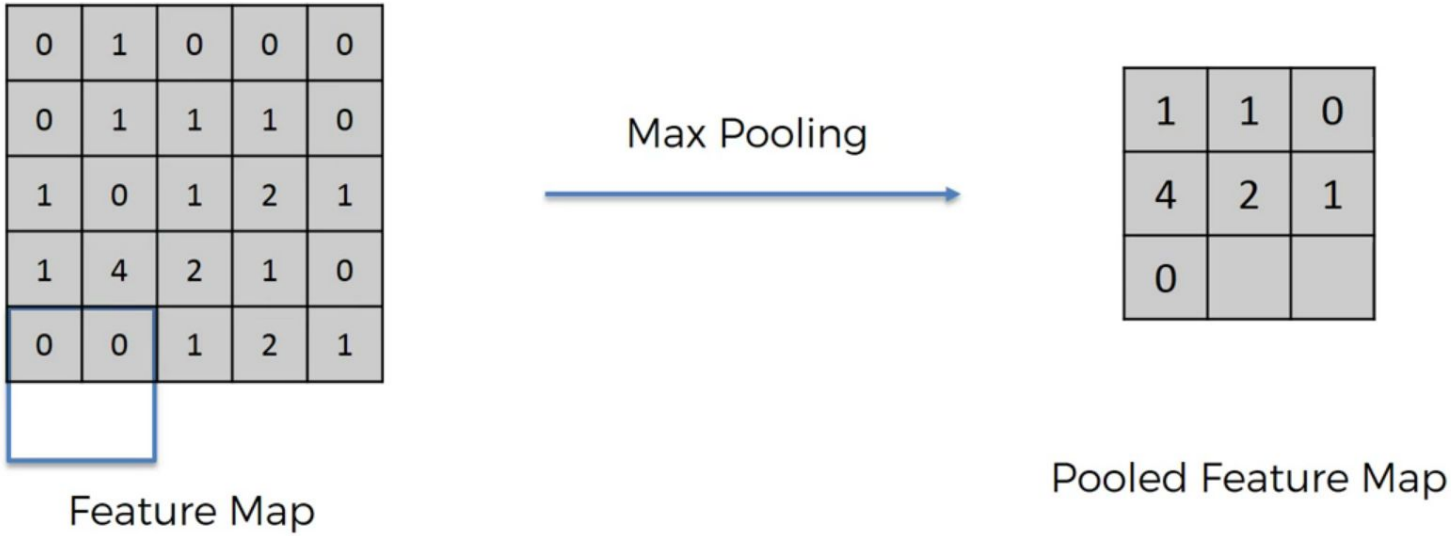


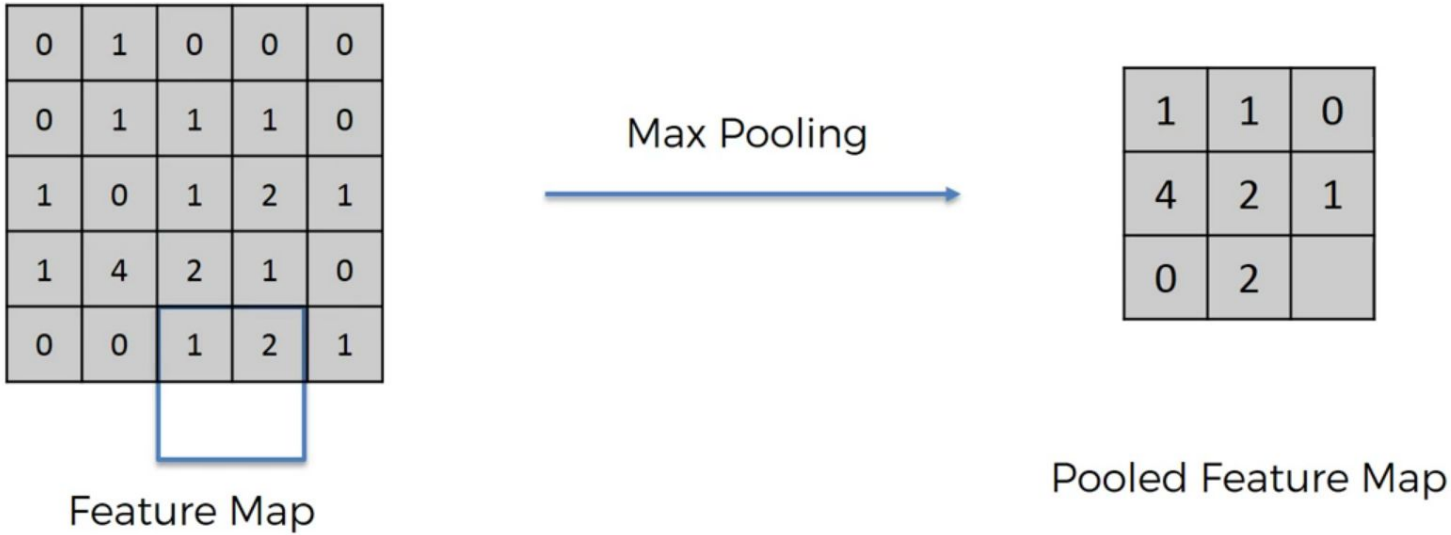
Feature Map

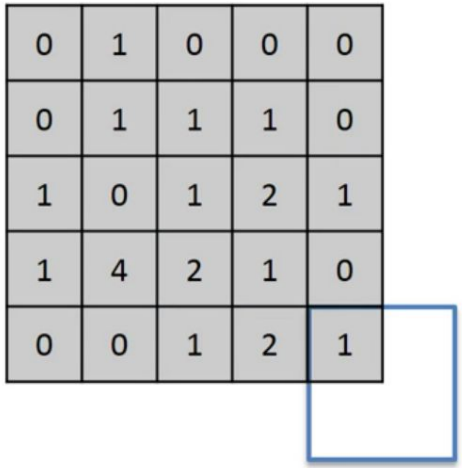
Max Pooling



Pooled Feature Map


0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Feature Map
Max Pooling
1 1 0
4 2 1
0
Pooled Feature Map


0 1 0 0 0
0 1 1 1 0
1 0 1 2 1
1 4 2 1 0
0 0 1 2 1
Feature Map
Max Pooling
1 1 0
4 2 1
0 2
Pooled Feature Map

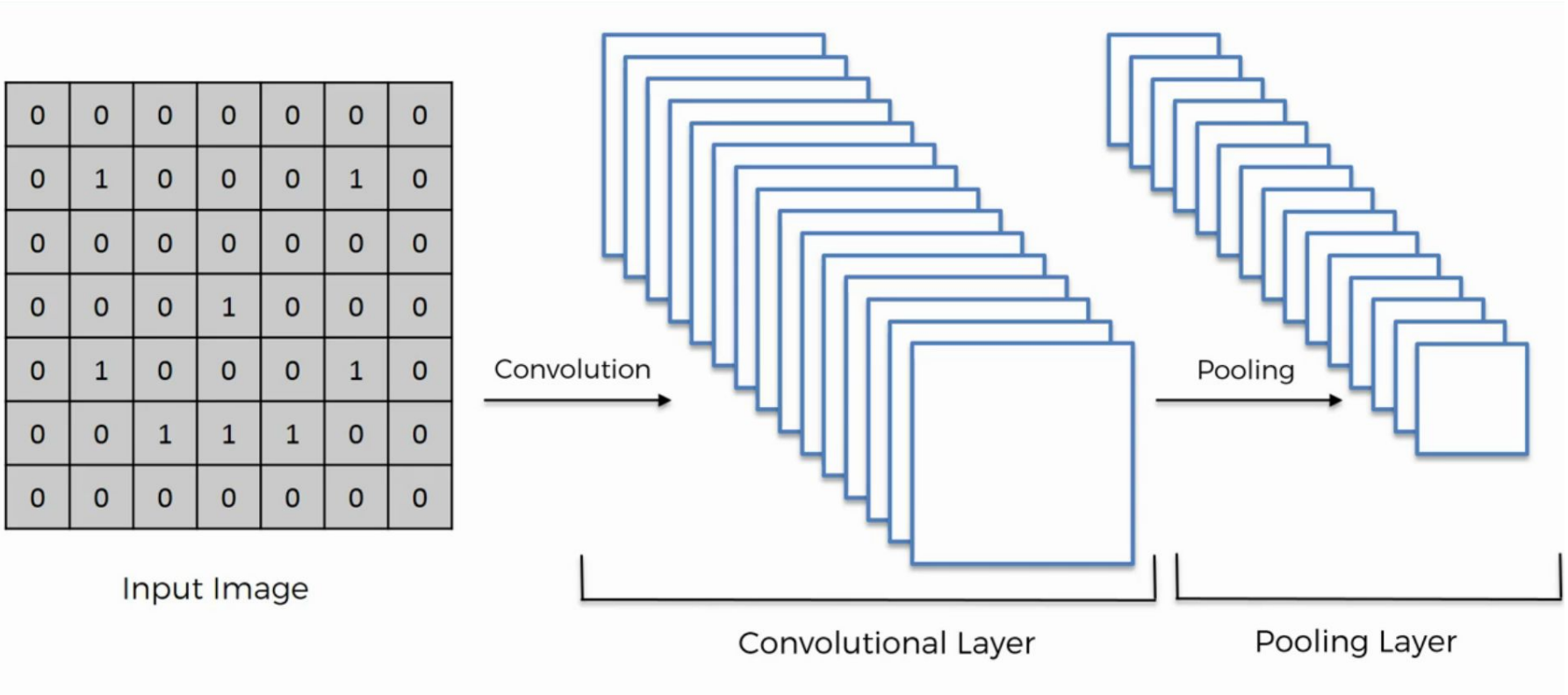


Feature Map

Max Pooling

1	1	0
4	2	1
0	2	1

Pooled Feature Map


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolution
Pooling
Convolutional Layer
Pooling Layer

Max-Polling Objectives

- Aggressively downsample feature maps.
- max tensor operation.
- 2x2.
- Stride = 2.

Max-Polling Objectives

- Aggressively downsample feature maps.
- max tensor operation.
- 2x2.
- Stride = 2.

Whereas convolution:

- 3x3.
- Stride = 1.

Max-Polling Objectives

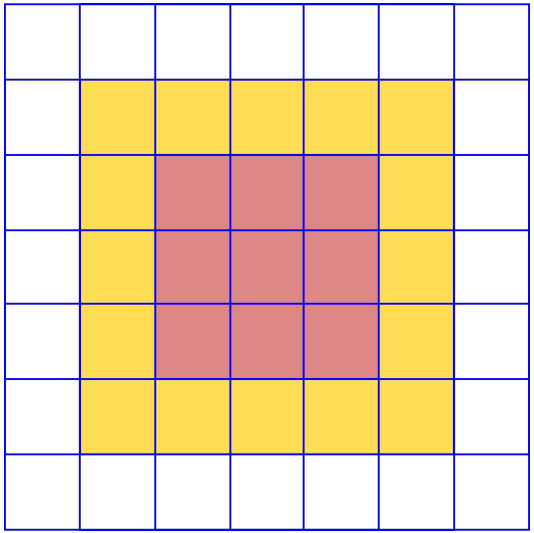
```
from keras import models, layers
model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu',
                                    input_shape=(28, 28, 1)))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 22, 64)        36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

Max-Polling Objectives

Problems:

- No spatial hierarchy of features.



Max-Polling Objectives

Problems:

- Final feature map has 22 x 22 x 64 = 30976 total coefficients per sample.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 22, 64)        36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

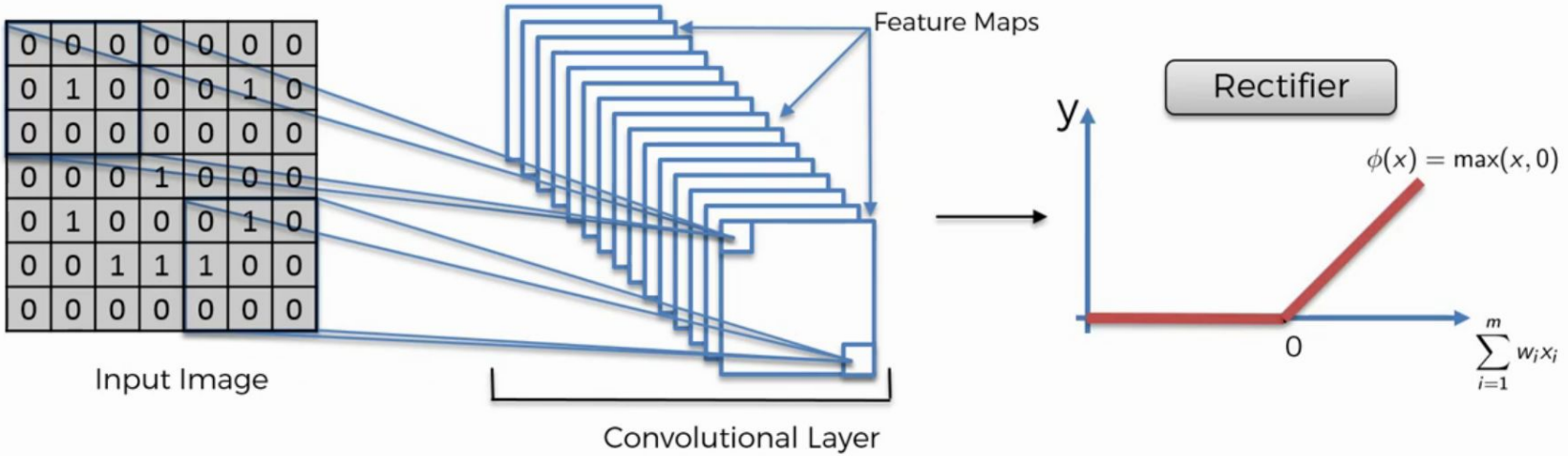
Max-Polling Objectives

- Reduce the number of feature-map coefficients to process.
- Induce spatial-filter hierarchies by making successive convolution layers look at increasingly large windows (in terms of the fraction of the original input they cover).

Other approaches:

- Strides in the prior convolution layer.
- Average pooling instead of max pooling.
- However, max pooling tends to work better:
  - Features tend to encode the spatial presence of some pattern.
  - it’s more informative to look at the maximal presence of different features than at their average presence.

Rectifier Linear Unit activation function (ReLU)


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Feature Maps
Convolutional Layer
Rectifier
y
$\phi(x) = \max(x, 0)$
0
$\sum_{i=1}^{m} w_i x_i$

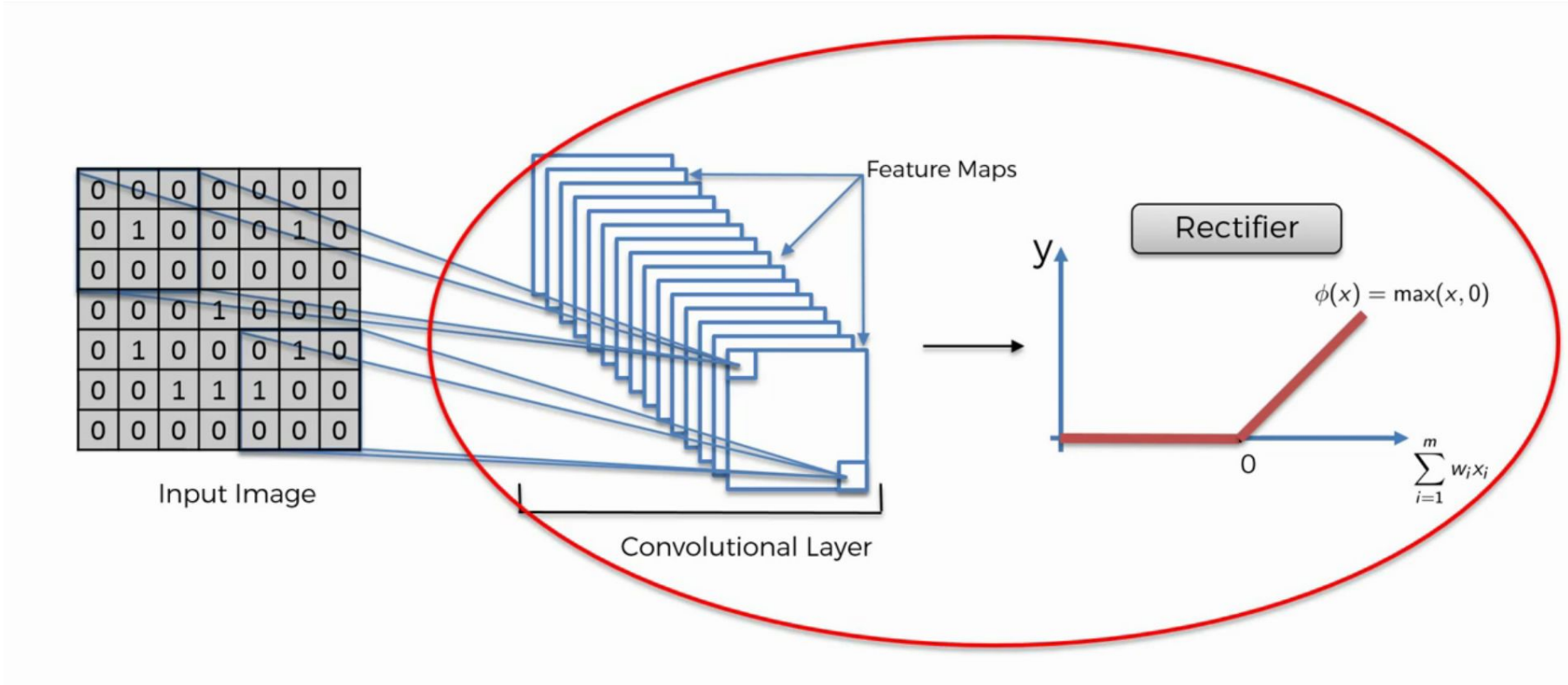

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Feature Maps
Convolutional Layer
Rectifier
y
$\phi(x) = \max(x, 0)$
0
$\sum_{i=1}^{m} w_i x_i$

Image Source: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

Image Source: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

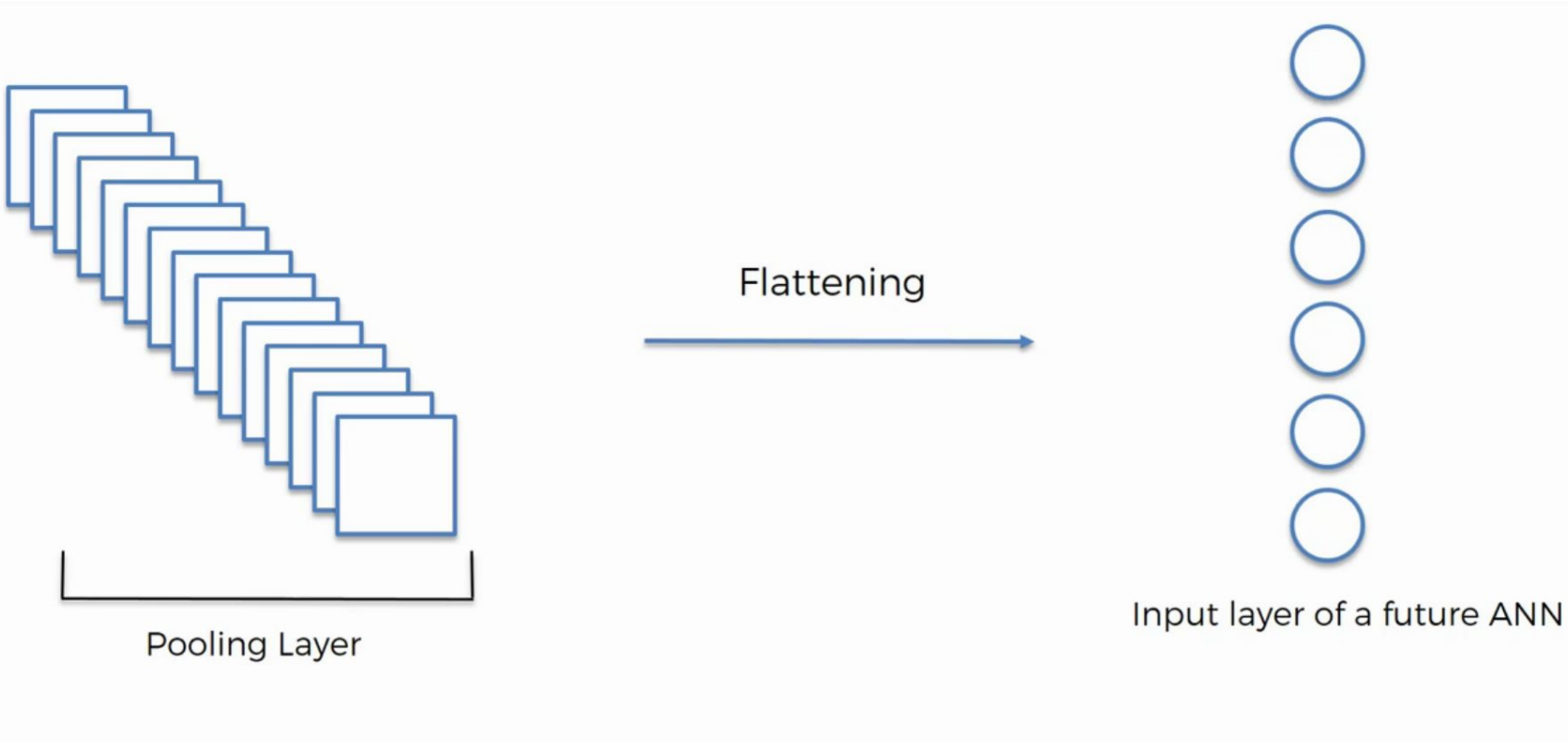
Image Source: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

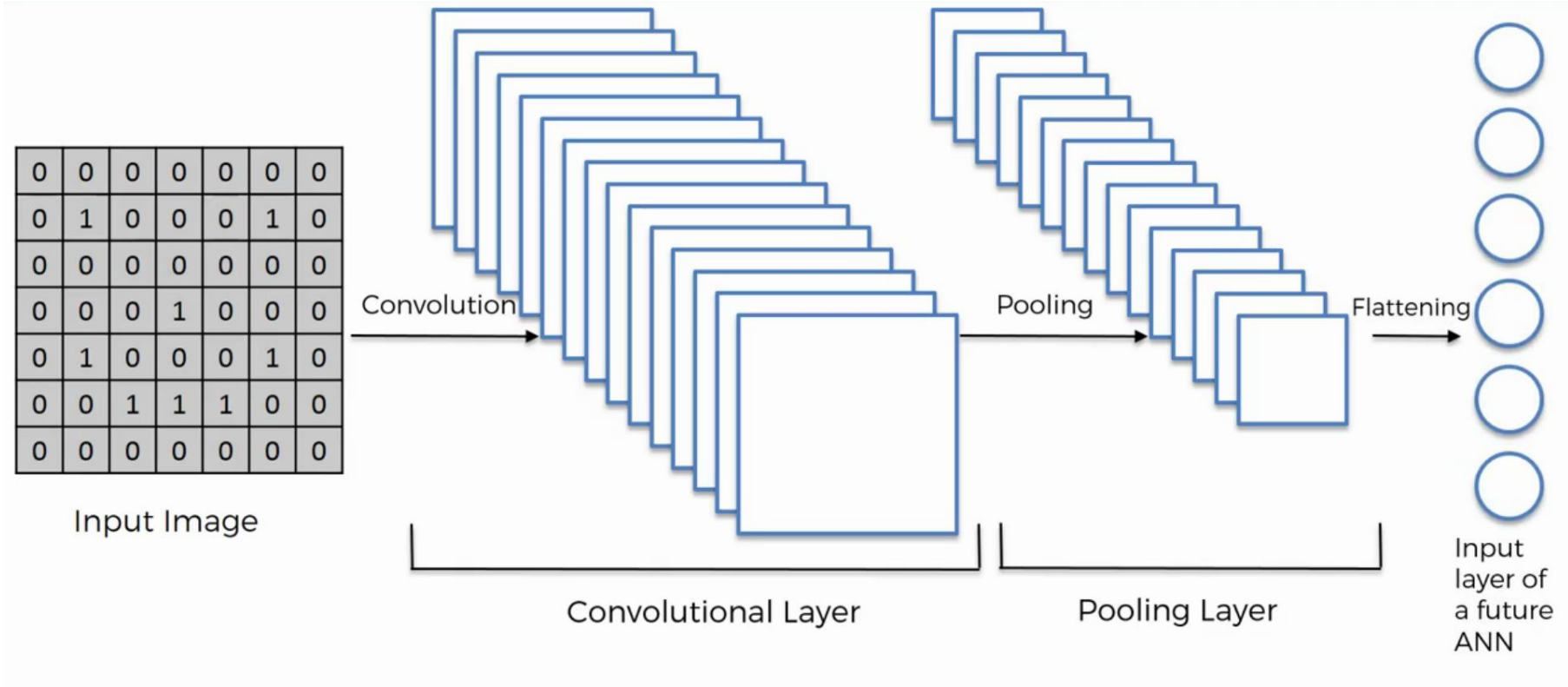
Flattening

1	1	0
4	2	1
0	2	1

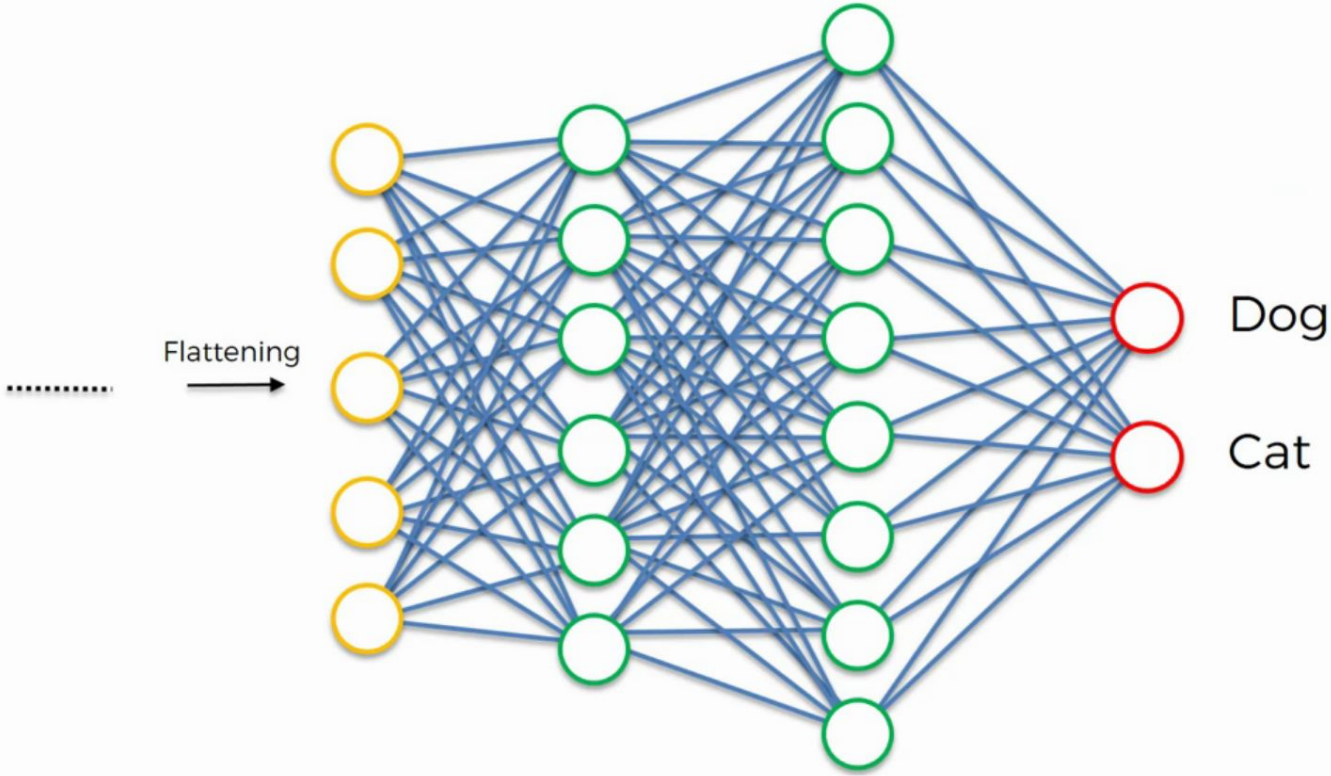
Pooled Feature Map

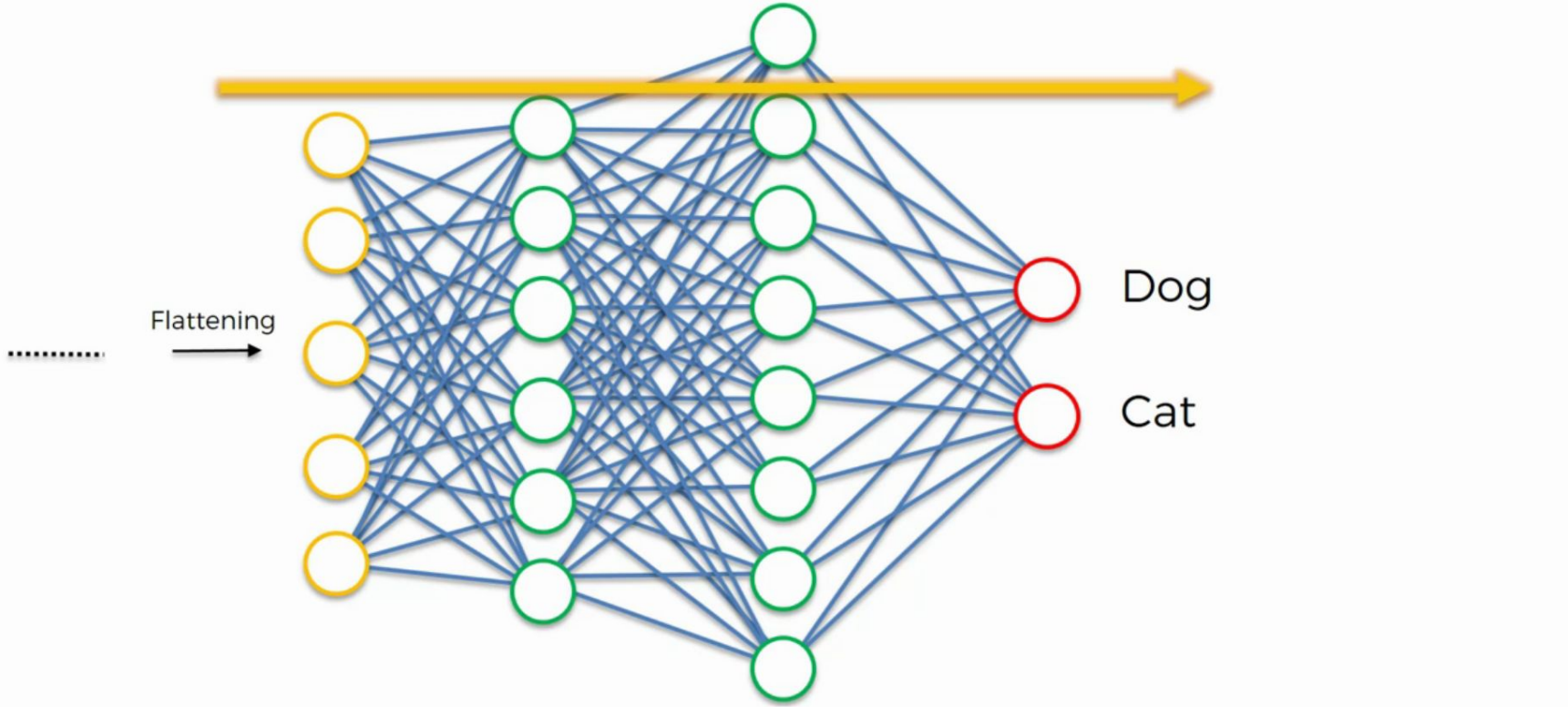

1 1 0
4 2 1
0 2 1
Pooled Feature Map
Flattening
1
1
0
4
2
1
0
2
1

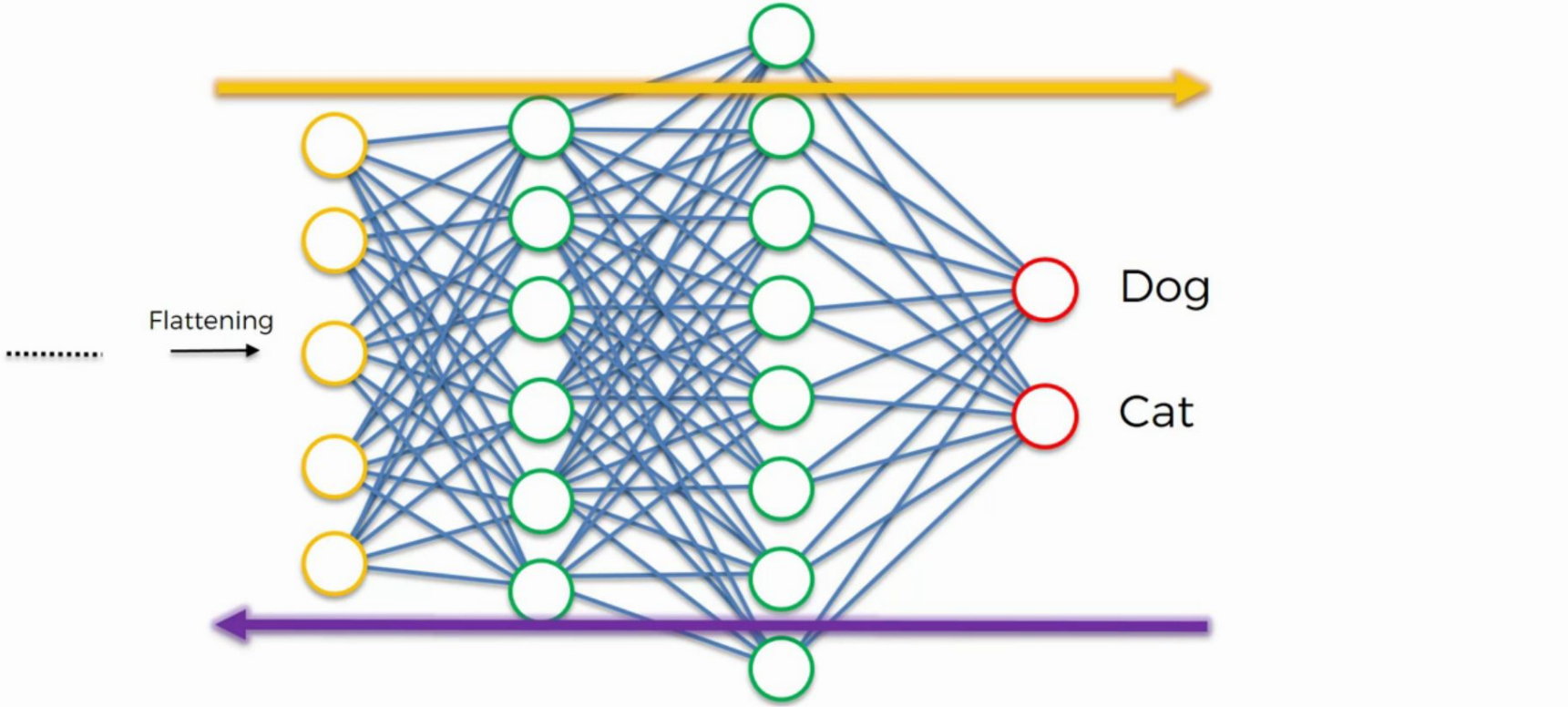

Flattening
Pooling Layer
Input layer of a future ANN

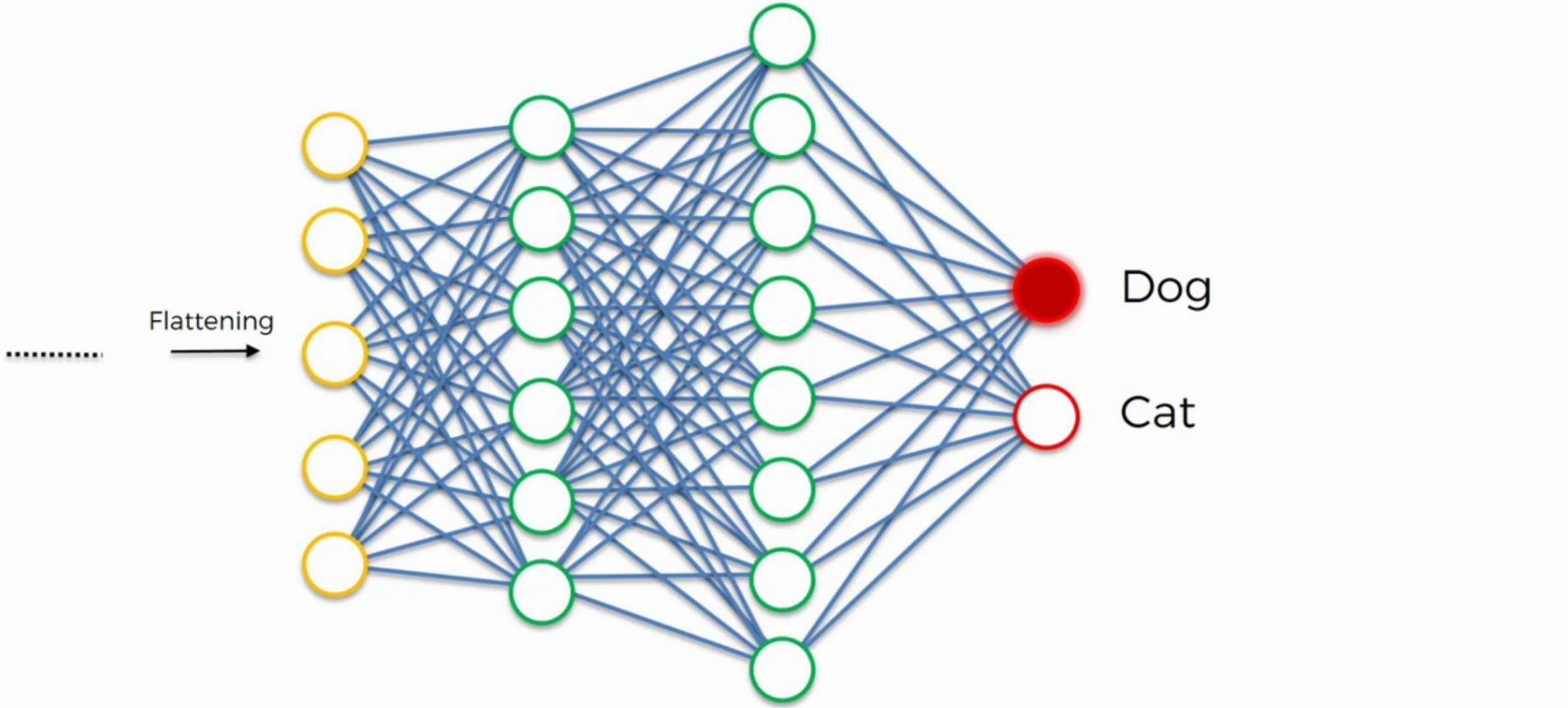

0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolution
Pooling
Flattening
Convolutional Layer
Pooling Layer
Input layer of a future ANN

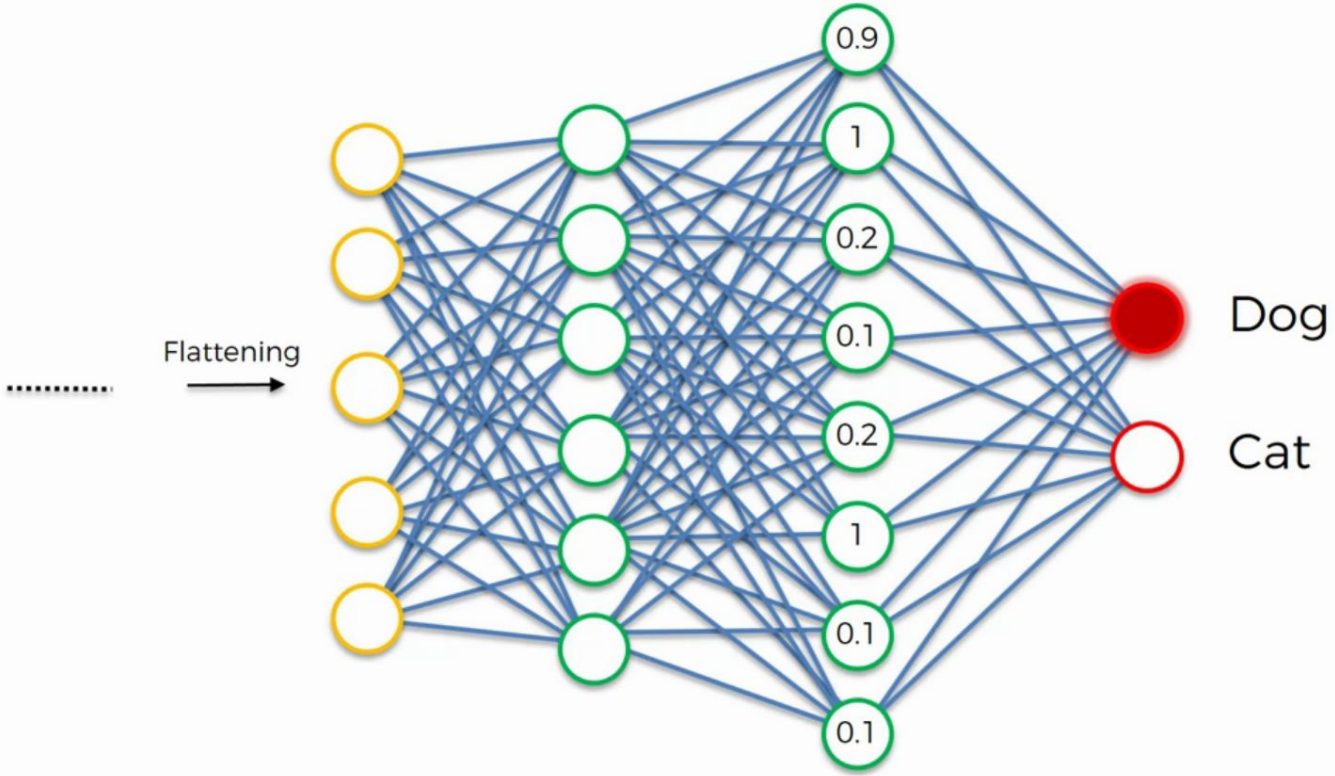
Full Connection

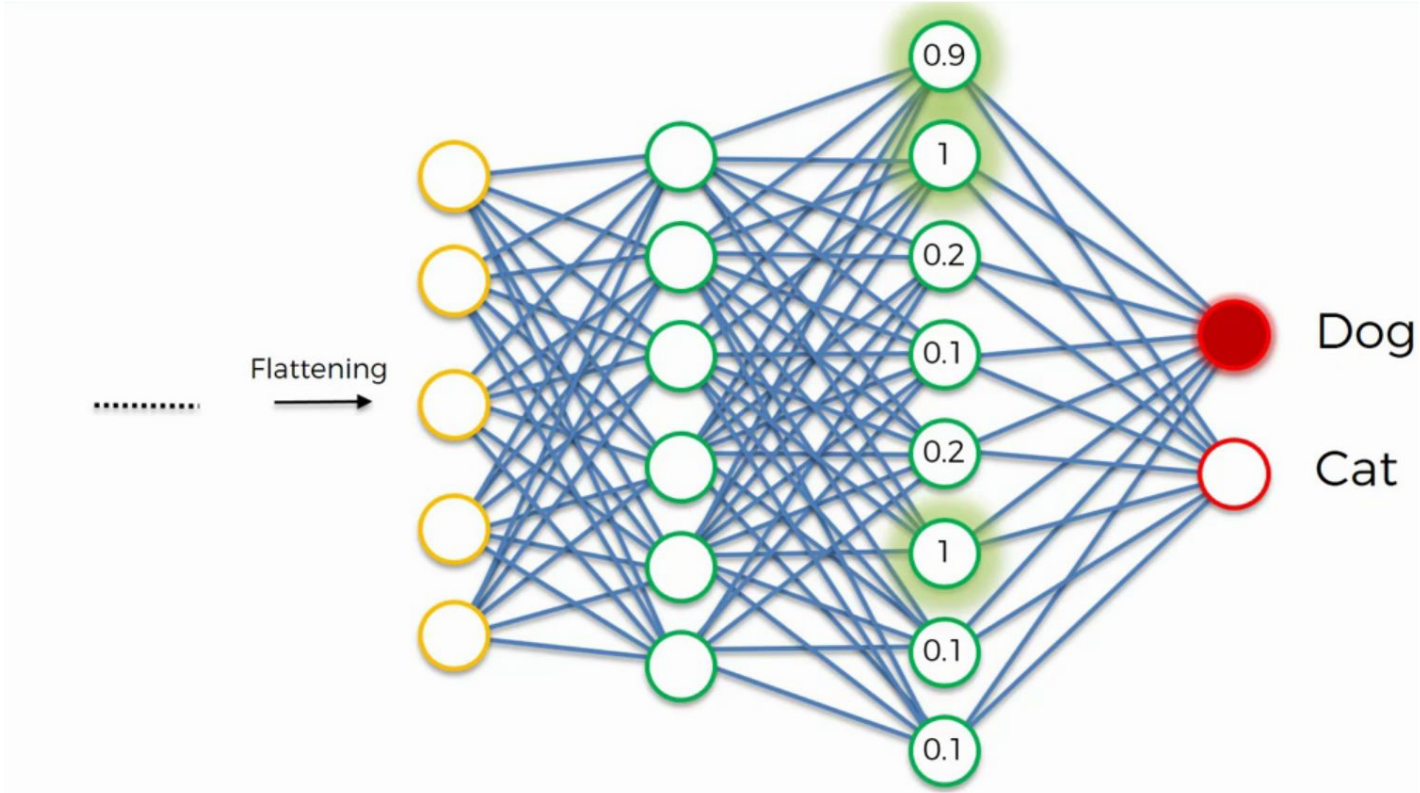

Flattening
Dog
Cat

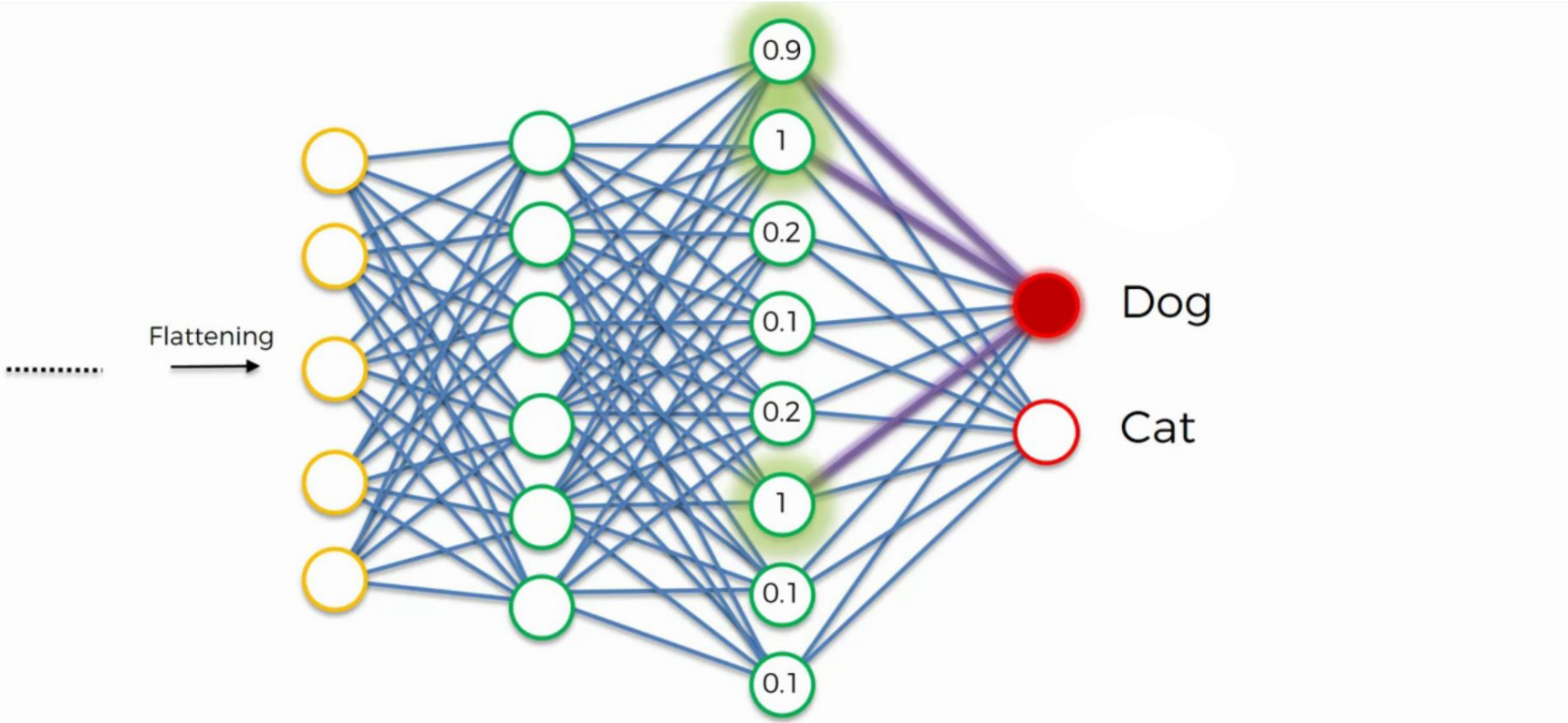

Flattening
Dog
Cat

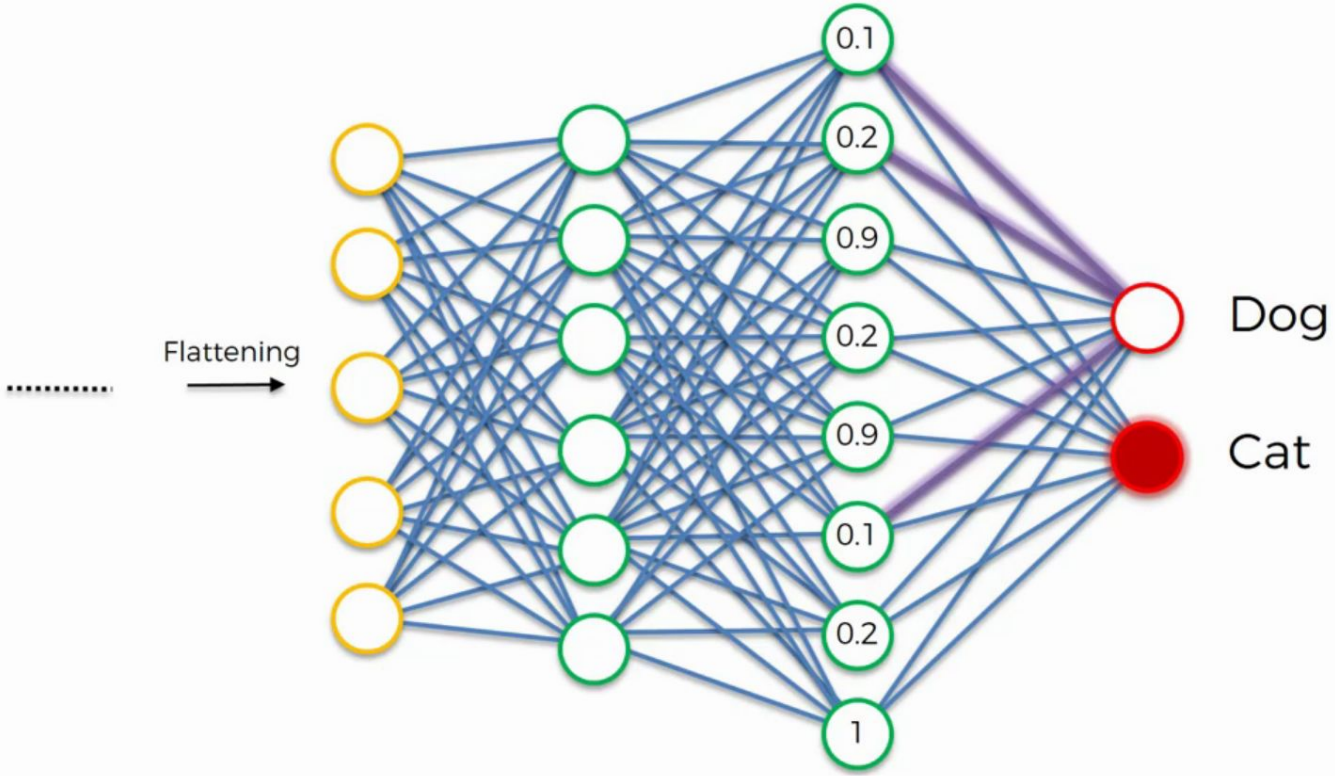

Flattening
Dog
Cat

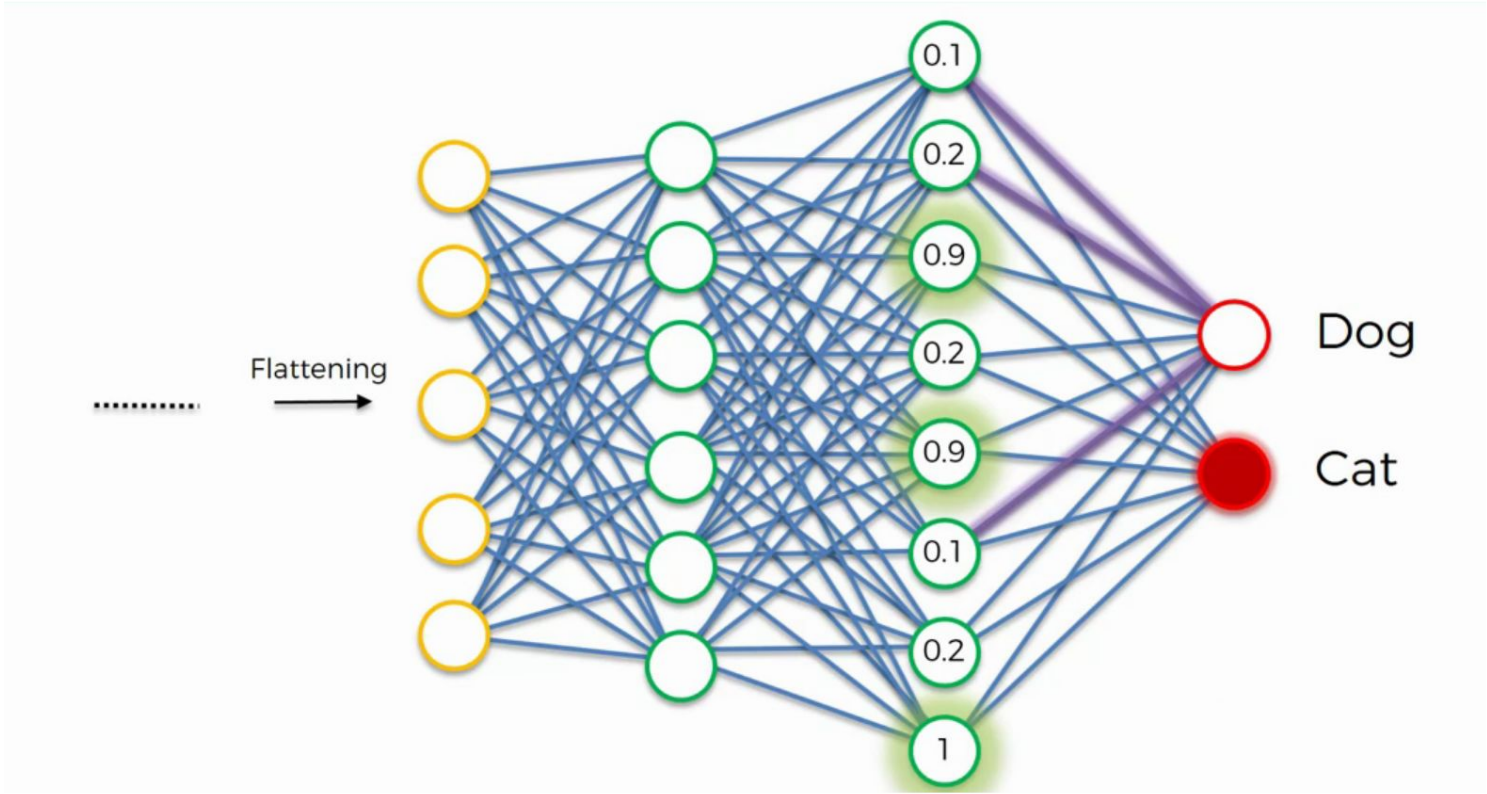

Flattening
Dog
Cat

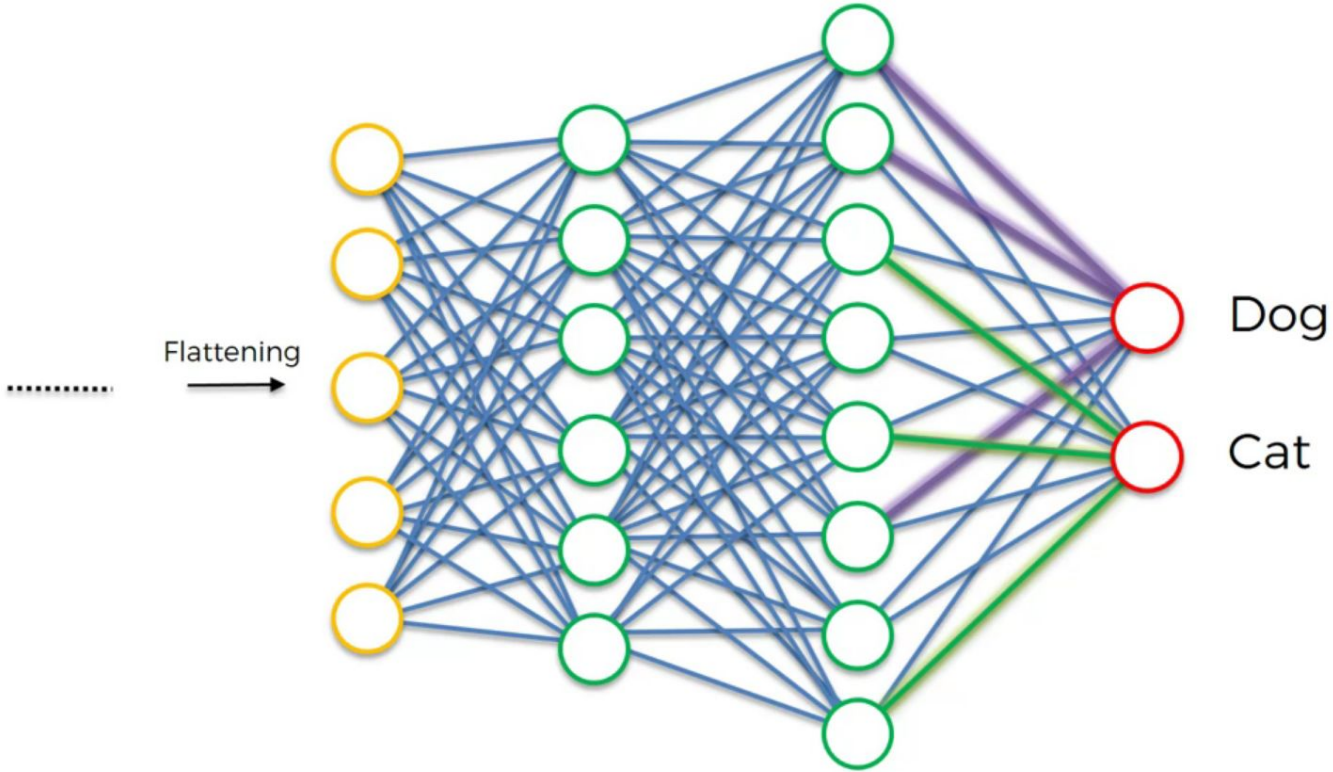

Flattening
0.9
1
0.2
0.1
0.2
1
0.1
0.1
Dog
Cat


Flattening
0.9
1
0.2
0.1
0.2
1
0.1
0.1
Dog
Cat

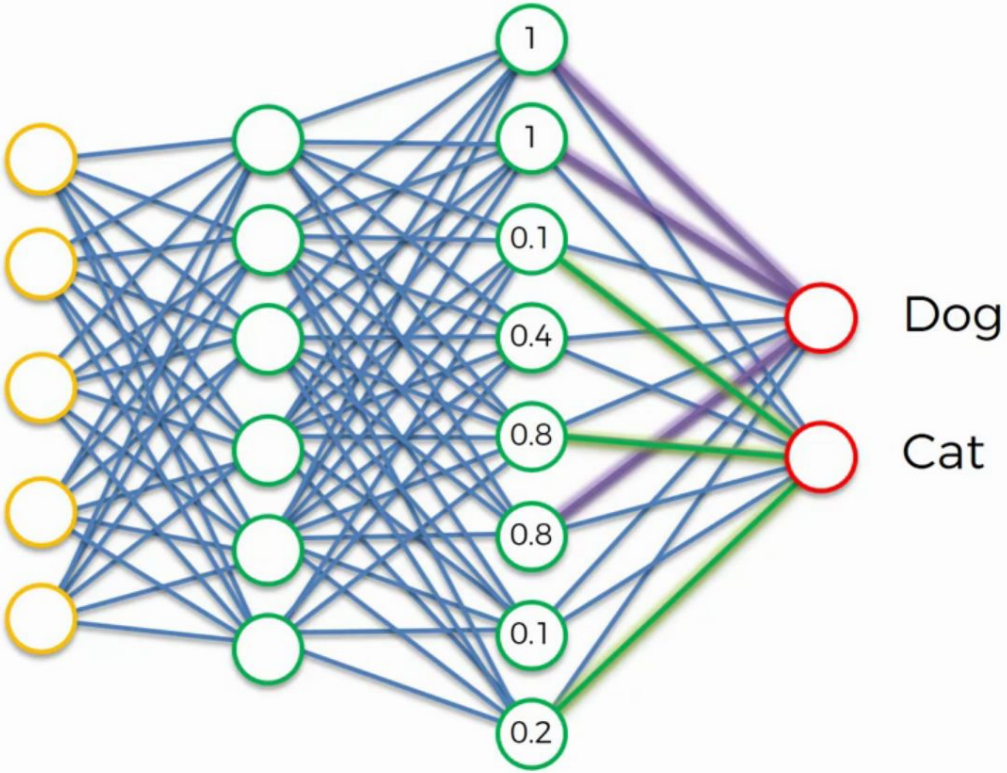

Flattening
0.9
1
0.2
0.1
0.2
1
0.1
0.1
Dog
Cat


Flattening
0.1
0.2
0.9
0.2
0.9
0.1
0.2
1
Dog
Cat

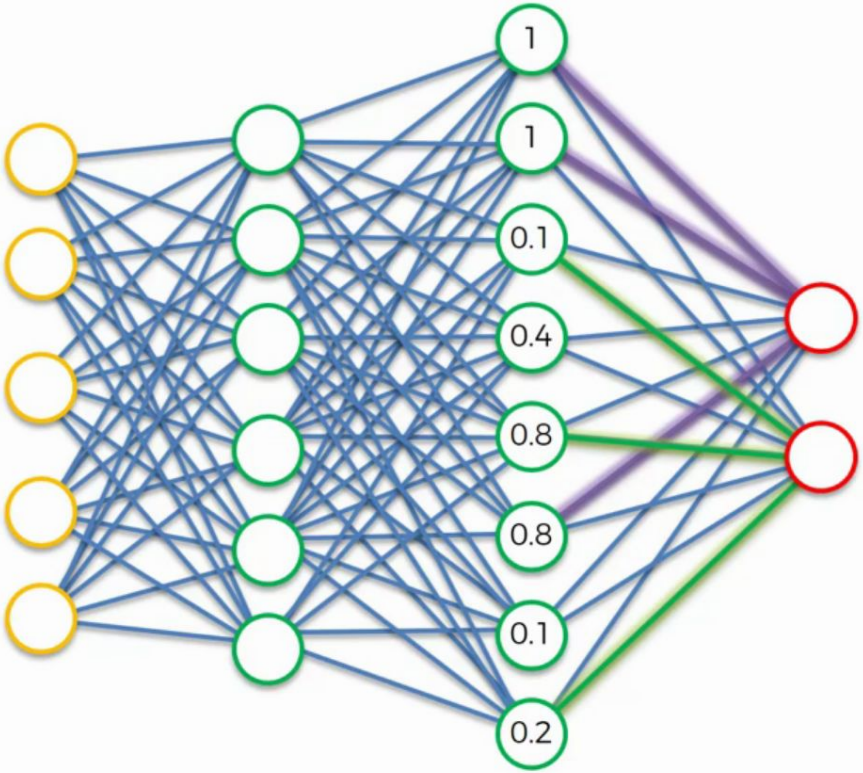

Flattening
0.1
0.2
0.9
0.2
0.9
0.1
0.2
1
Dog
Cat


Flattening
Dog
Cat

Flattening



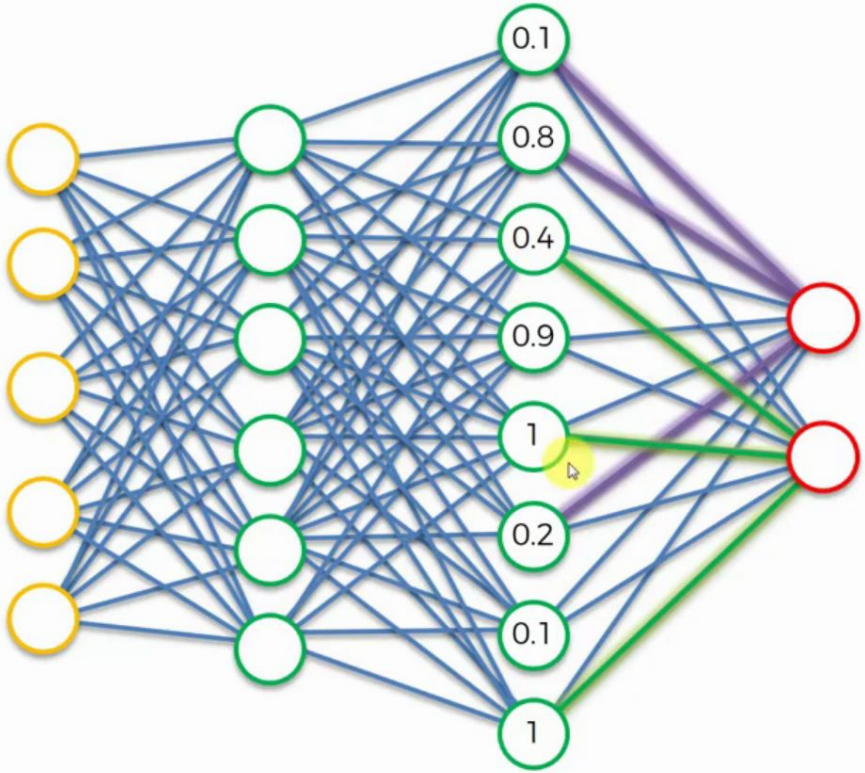
Flattening



Dog 0.95

Cat 0.05

Flattening



Dog 0.21

Cat 0.79

Examples from the test set
(with the network's guesses)

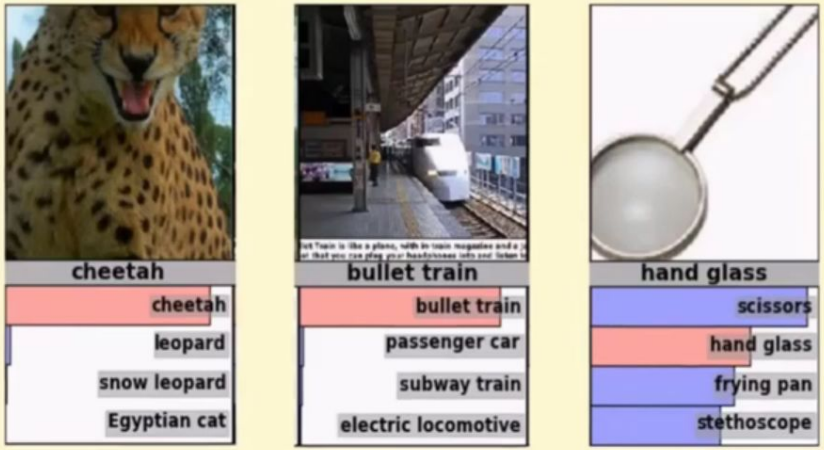
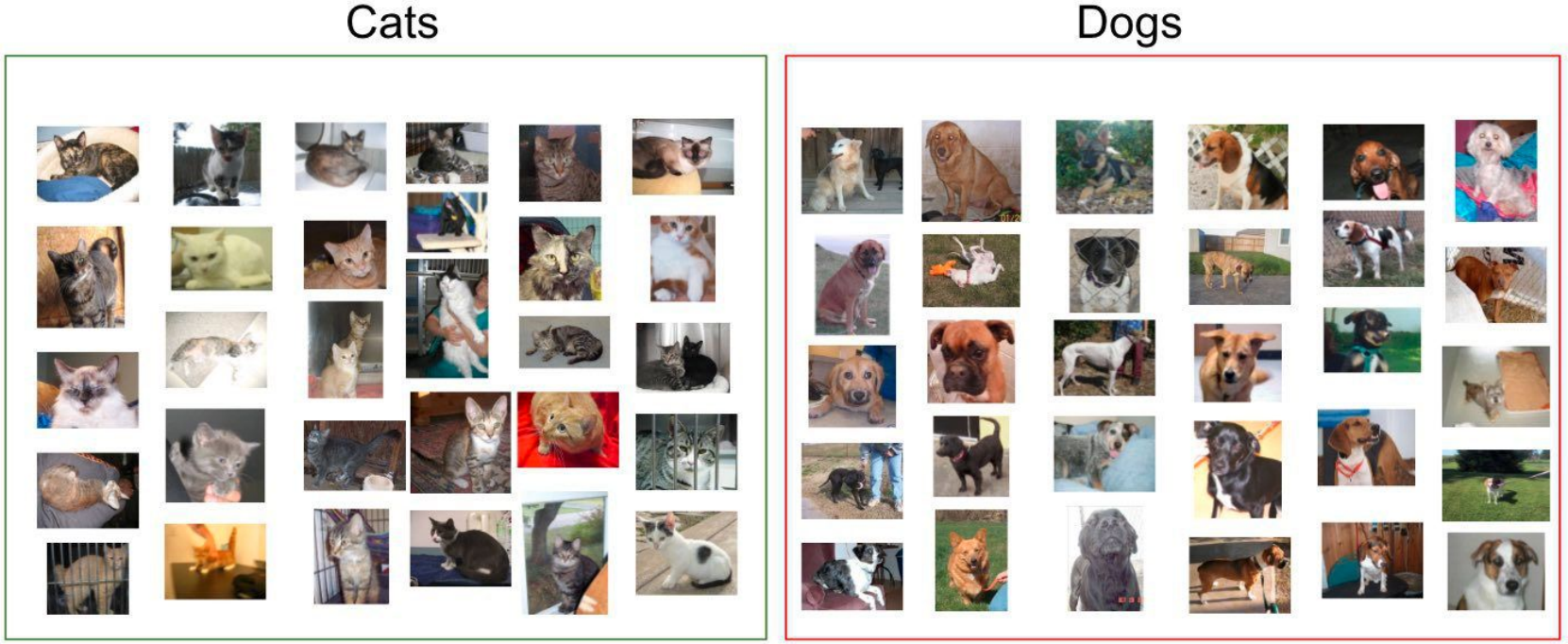


Image Source: a talk by Geoffrey Hinton

Training a ConvNet on a Small Dataset

Dogs vs. Cats (Kaggle - 2013)



Sample of cats & dogs images from Kaggle Dataset

www.kaggle.com/c/dogs-vs-cats/data

Dogs vs. Cats (Kaggle - 2013)

We'll use:

- 4000 pictures of cats (2000) and dogs (2000).
- Train on 2000 images.
- Validate on 1000 images.
- Test on 1000 images.

Naive training → 71% accuracy.

Data augmentation → 82% accuracy.

Strategies

1. Train a small model from scratch.
2. Perform feature extraction with a pre-trained network.
3. Fine-tune the pre-trained model.

Deep Learning for Small-Data Problems

- DL finds interesting features automatically.
- So lots (relative) of training examples are required.
- Lots of samples is relative to the size and depth of the network.
- For a complex problem:
  - Few tens might be insufficient.
  - Few hundred can potentially suffice if:
    - The model is small.
    - The model is well regularized.
    - The task is simple.

Convnets:

- Learn local, translation-invariant features.
- Are highly efficient in perceptual problems.
- Training from scratch on a very small image dataset will still yield reasonable results despite a relative lack of data.
- Deep-learning models are by nature highly repurposable.

Pretrained models (usually from Image-Net) are now publicly available and can bootstrap powerful vision models out of very little data!

Dogs vs. Cats (Kaggle Competition - 2013)

www.kaggle.com/c/dogs-vs-cats/data

Organize the dataset

```
#%%=============================================================================
# 5.2-using-convnets-with-small-datasets
# ============================================================================

#%% Import modules
import keras
keras.__version__

import os, shutil

#%% Organize the dataset
# The path to the directory where the original
# dataset was uncompressed
original_dataset_dir = '/home/tvieira/Dropbox/db/dl/cats_and_dogs'

# The directory where we will
# store our smaller dataset
base_dir = './cats_and_dogs_small'
os.mkdir(base_dir)
```

```
# Directories for our training,
# validation and test splits
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir,
'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)

# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)
```

```
# Directory with our validation cat pictures
validation_cats_dir =
os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)

# Directory with our validation dog pictures
validation_dogs_dir =
os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)

# Directory with our test cat pictures
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)

# Directory with our test dog pictures
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)
```

```
#%% Copy first 1000 cat images to train_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
range(1000)]
for fname in fnames:
     src = os.path.join(original_dataset_dir,
fname)
     dst = os.path.join(train_cats_dir, fname)
     shutil.copyfile(src, dst)

# Copy next 500 cat images to validation_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
range(1000, 1500)]
for fname in fnames:
     src = os.path.join(original_dataset_dir,
fname)
     dst = os.path.join(validation_cats_dir,
fname)
     shutil.copyfile(src, dst)

# Copy next 500 cat images to test_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in
range(1500, 2000)]
for fname in fnames:
     src = os.path.join(original_dataset_dir,
fname)
     dst = os.path.join(test_cats_dir, fname)
     shutil.copyfile(src, dst)
```

```
# Copy first 1000 dog images to train_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
        src = os.path.join(original_dataset_dir, fname)
        dst = os.path.join(train_dogs_dir, fname)
        shutil.copyfile(src, dst)

# Copy next 500 dog images to validation_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
        src = os.path.join(original_dataset_dir, fname)
        dst = os.path.join(validation_dogs_dir, fname)
        shutil.copyfile(src, dst)

# Copy next 500 dog images to test_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
        src = os.path.join(original_dataset_dir, fname)
        dst = os.path.join(test_dogs_dir, fname)
        shutil.copyfile(src, dst)
```

```
#%% Sanity check
print('total training cat images:', len(os.listdir(train_cats_dir)))
#total training cat images: 1000
print('total training dog images:', len(os.listdir(train_dogs_dir)))
#total training dog images: 1000
print('total validation cat images:', len(os.listdir(validation_cats_dir)))
#total validation cat images: 500
print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
#total validation dog images: 500
print('total test cat images:', len(os.listdir(test_cats_dir)))
#total test cat images: 500
print('total test dog images:', len(os.listdir(test_dogs_dir)))
#total test dog images: 500
```

- 2000 training images
- 1000 validation images
- 1000 test images
- Balanced dataset → Classification accuracy is an appropriate performance metrics.

Building the Network

```
#%% Listing 5.5 Instantiating a small convnet for dogs vs. cats 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'))
model.summary()
```

```
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
_________________________________________________________________
dense_3 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

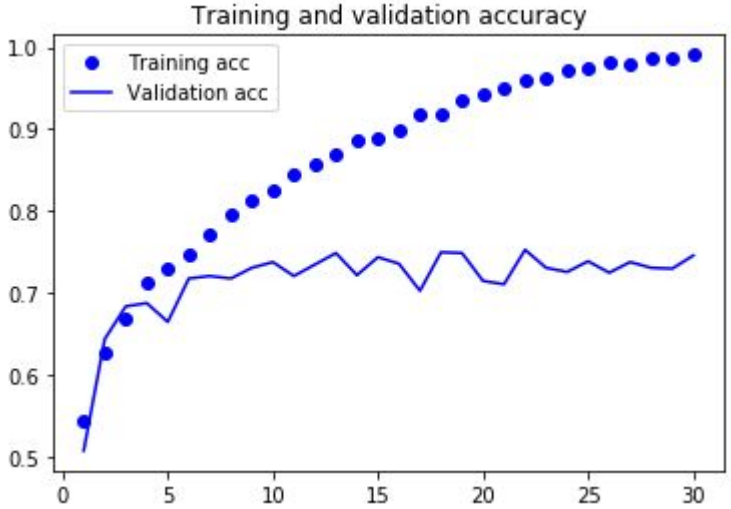
```
#%% Listing 5.6 Configuring the model for training
from keras import optimizers
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

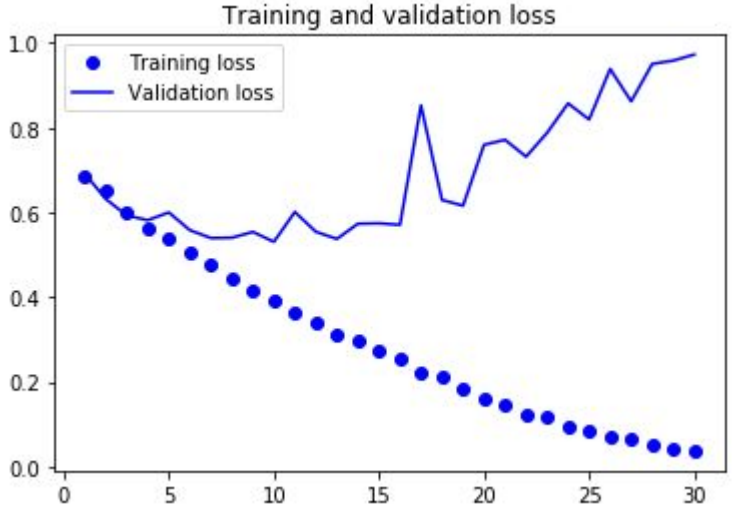
Data Pre-Processing

```
#%% Listing 5.7 Using ImageDataGenerator to read images from directories
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    # This is the target directory
    train_dir,
    # All images will be resized to 150x150
    target_size=(150, 150),
    batch_size=20,
    # Since we use binary_crossentropy loss, we need binary labels
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

```
#%% Check batch dimensions
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break

#%% Listing 5.8 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)

#%% Listing 5.9 Saving the model
#model.save('cats_and_dogs_small_1.h5')
```

```
#%% Listing 5.10 Displaying curves of loss and accuracy during training
import matplotlib.pyplot as plt
from keras.models import load_model
model = load_model('cats_and_dogs_small_1.h5')

acc = history['acc']
val_acc = history['val_acc']
loss = history['loss']
val_loss = history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

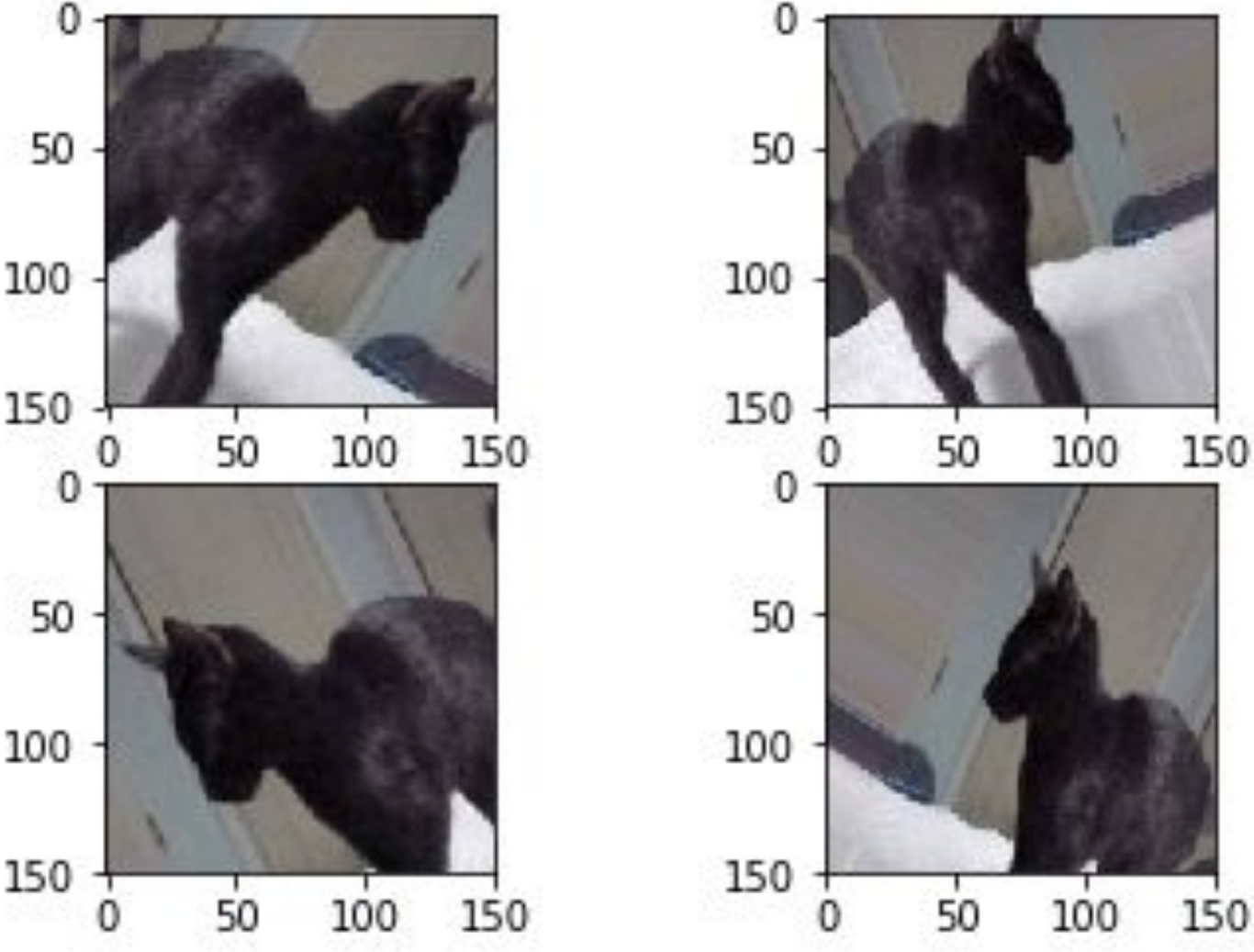

Training and validation accuracy
Training acc
Validation acc
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30


Training and validation loss
Training loss
Validation loss
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

Using data augmentation

```
#%% Listing 5.11 Setting up a data augmentation configuration via
ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator
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')
```

```
#%% Listing 5.12 Displaying some randomly augmented training images
# This is module with image preprocessing utilities
from keras.preprocessing import image
fnames = [os.path.join(train_cats_dir, fname) for fname in
os.listdir(train_cats_dir)]
# We pick one image to "augment"
img_path = fnames[2]
# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))
# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)
# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)
# The .flow() command below generates batches of randomly transformed images.
# It will loop indefinitely, so we need to `break` the loop at some point!
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.subplot('22' + str(i + 1))
    plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
    break
plt.show()
```


0
50
100
150
0
50
100
150
0
50
100
150
0
50
100
150
0
50
100
150
0
50
100
150
0
50
100
150
0
50
100
150

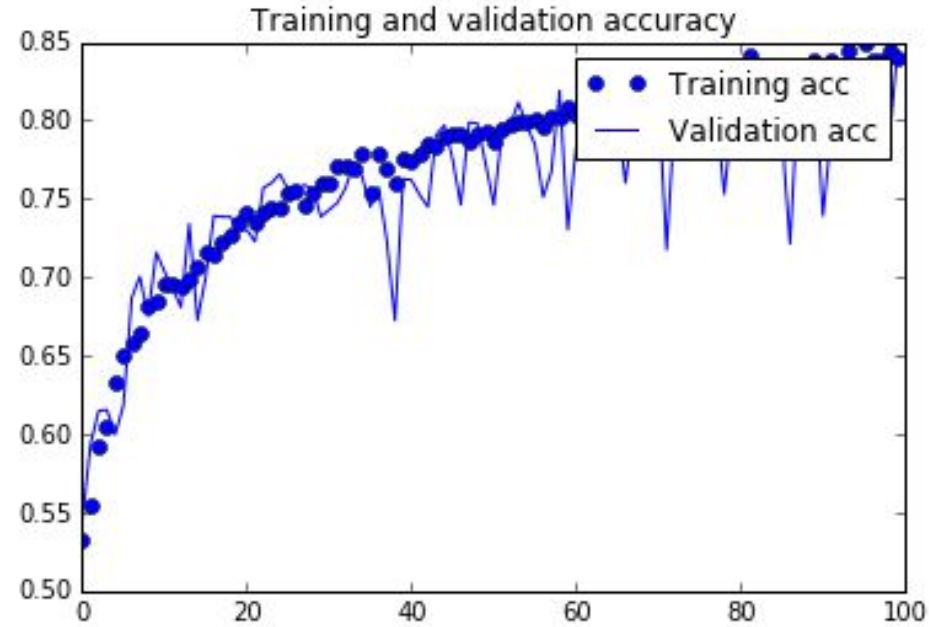
```
#%% Listing 5.13 Defining a new convnet that includes dropout
from keras import optimizers
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 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.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

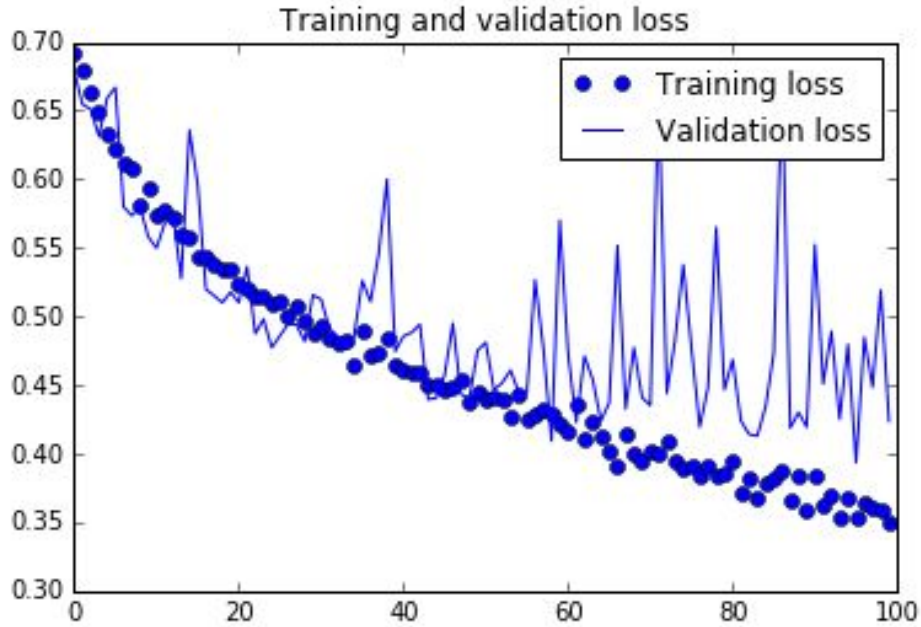
```
#%% Listing 5.14 Training the convnet using data-augmentation generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)
```

```
# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    # This is the target directory
    train_dir,
    # All images will be resized to 150x150
    target_size=(150, 150),
    batch_size=32,
    # Since we use binary_crossentropy loss, we need binary labels
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')
```

```
#%% Fit the model
#history = model.fit_generator(
#    train_generator,
#    steps_per_epoch=100,
#    epochs=100,
#    validation_data=validation_generator,
#    validation_steps=50)

#%% Listing 5.15 Saving the model
import json
#model.save('cats_and_dogs_small_2.h5')
model = load_model('cats_and_dogs_small_2.h5')
# Write history to file
#json.dump(history.history, open('cats_and_dogs_small_2_history.json','wb'))
# Read history
history = json.load(open('cats_and_dogs_small_2_history.json'))
```

```
#%% Let's plot our results again
acc = history['acc']
val_acc = history['val_acc']
loss = history['loss']
val_loss = history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
20
40
60
80
100


Training and validation loss
Training loss
Validation loss
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
20
40
60
80
100

Using a pre-trained convnet

airplane

automobile

bird

cat

deer

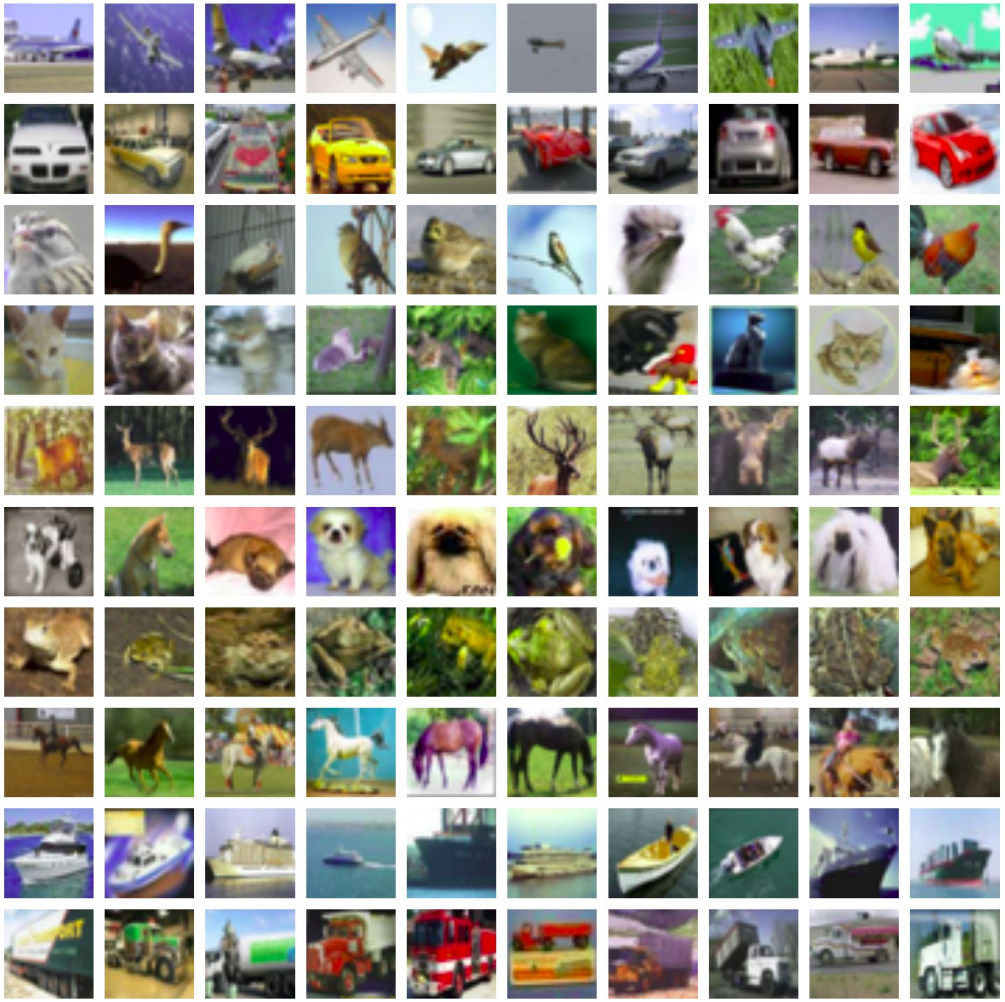
dog

frog

horse

ship

truck



- ~1.4 million labeled images.
- ~1,000 different classes

Using a pre-trained convnet

- A pretrained network is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task.
- If this original dataset is large enough and general enough, then the spatial hierarchy of features learned by the pretrained network can effectively act as a generic model of the visual world.
    - hence its features can prove useful for many different computer-vision problems.
    - even though these new problems may involve completely different classes than those of the original task.
- For instance, you might train a network on ImageNet (where classes are mostly animals and everyday objects) and then repurpose this trained network for something as remote as identifying furniture items in images.

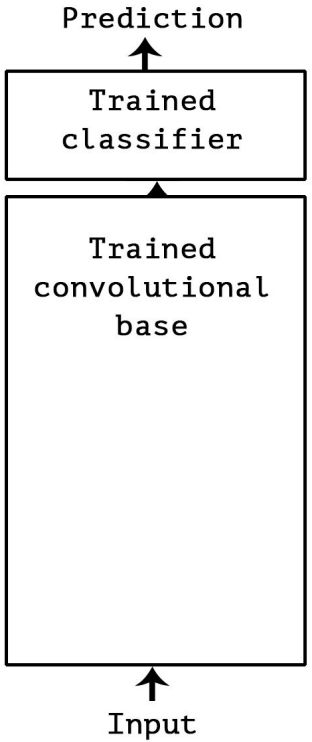
Using a pre-trained convnet

- Deep Learning is portable!
- Let's use the VGG16:
  - Karen Simonyan and Andrew Zisserman, "*Very Deep Convolutional Networks for Large-Scale Image Recognition*," arXiv (**2014**), https://arxiv.org/abs/1409.1556.
  - ImageNet Challenge 2014 - 1st and the 2nd places in the localisation and classification tracks respectively.
  - 6th International Conference on Learning Representations
  - (ICLR 2018) International Conference on Learning Representations.

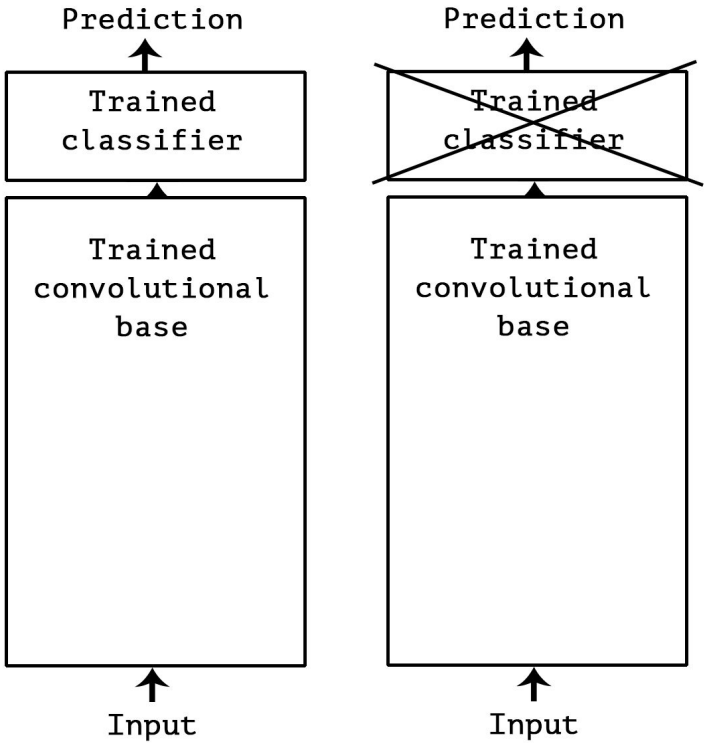
Pre-Trained ConvNets in Keras

- Xception
- Inception V3
- ResNet50
- VGG16
- VGG19
- MobileNet

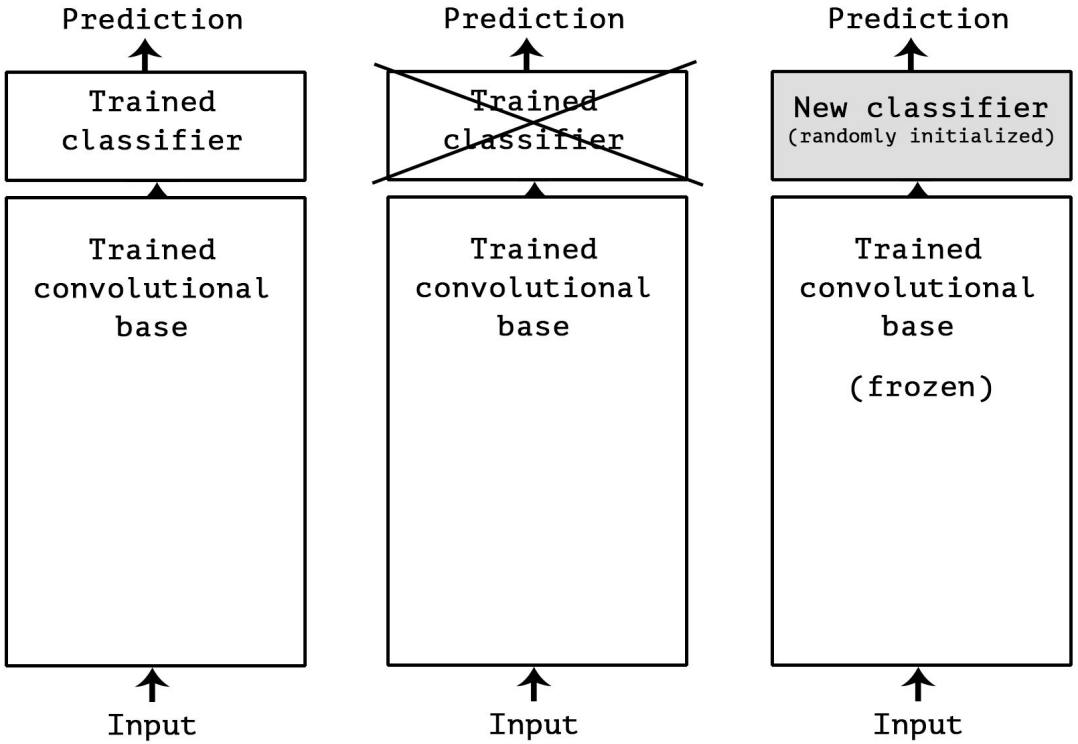
How does it work?



How does it work?



How does it work?



Should we reuse the classifier?

- Representations learned by the convolutional base are likely to be more generic and therefore more reusable: the feature maps of a convnet are presence maps of generic concepts over a picture, which is likely to be useful regardless of the computer-vision problem at hand.
  - object location is still described by convolutional feature maps.
- Representations learned by the classifier will necessarily be specific to the set of classes on which the model was trained—they will only contain information about the presence probability of this or that class in the entire picture.
  - No information about where objects are located in the input image.

level of generality
(reusability)

Layer depth

Hierarchy

- First layers:
  - Local, highly generic feature maps (such as visual edges, colors, and textures)
- Higher layers extract more-abstract concepts.
  - “cat ear” or “dog eye”.
- If your new dataset differs a lot from the dataset on which the original model was trained, you may be better off using only the first few layers of the model to do feature extraction, rather than using the entire convolutional base.

```
#%% Import modules
import matplotlib.pyplot as plt
import keras
import json
from keras.models import load_model
Keras.__version__

#%% Listing 5.16 Instantiating the VGG16 convolutional base
from keras.applications import VGG16
conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))

#%%
conv_base.summary()
```

FEATURE EXTRACTION
WITHOUT DATA AUGMENTATION

```
#%% Listing 5.17 Extracting features using the pretrained convolutional base
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

base_dir = './cats_and_dogs_small'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20
```

```
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
    directory,
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
    features_batch = conv_base.predict(inputs_batch)
    features[i * batch_size : (i + 1) * batch_size] = features_batch
    labels[i * batch_size : (i + 1) * batch_size] = labels_batch
    print('i = ' + str(i) + '; batch_size = ' + str(batch_size) + ';')
    i += 1
    if i * batch_size >= sample_count:
        # Note that since generators yield data indefinitely in a loop,
        # we must `break` after every image has been seen once.
        break
    return features, labels
```

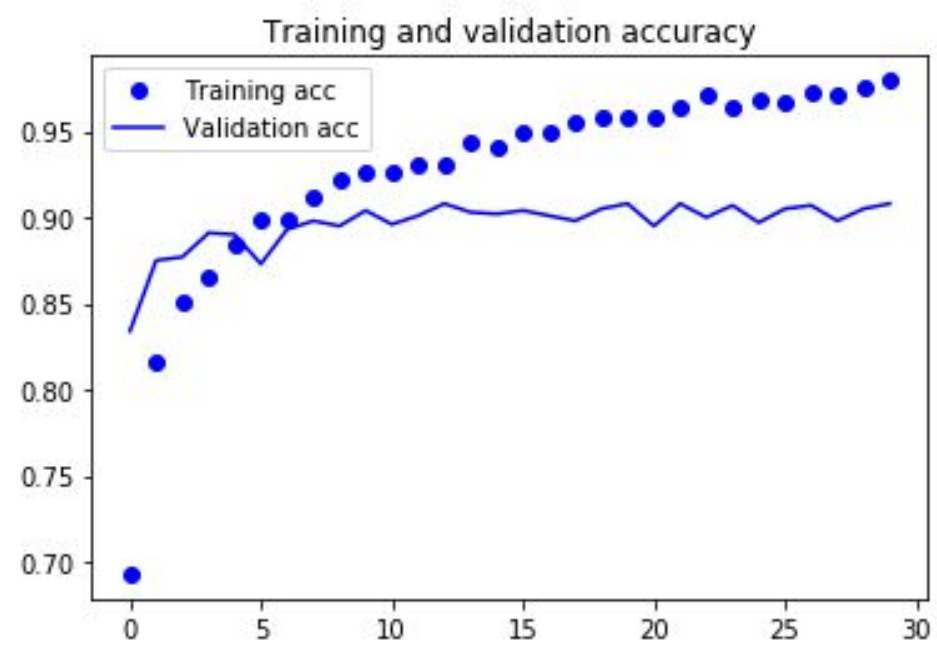
```
#%%
#train_features, train_labels = extract_features(train_dir, 2000)
#validation_features, validation_labels = extract_features(validation_dir, 1000)
#test_features, test_labels = extract_features(test_dir, 1000)

#### Save data - human readable
###train_features.tofile('train_features.txt', sep = ',')
###train_labels.tofile('train_labels.csv', sep = ',')
###validation_features.tofile('validation_features.txt', sep = ',')
###validation_labels.tofile('validation_labels.csv', sep = ',')
###test_features.tofile('test_features.txt', sep = ',')
###test_labels.tofile('test_labels.csv', sep = ',')

#%% Save data to binary files
#np.save('train_features.npy', train_features)
#np.save('train_labels.npy', train_labels)
#np.save('validation_features.npy', validation_features)
#np.save('validation_labels.npy', validation_labels)
#np.save('test_features.npy', test_features)
#np.save('test_labels.npy', test_labels)
```

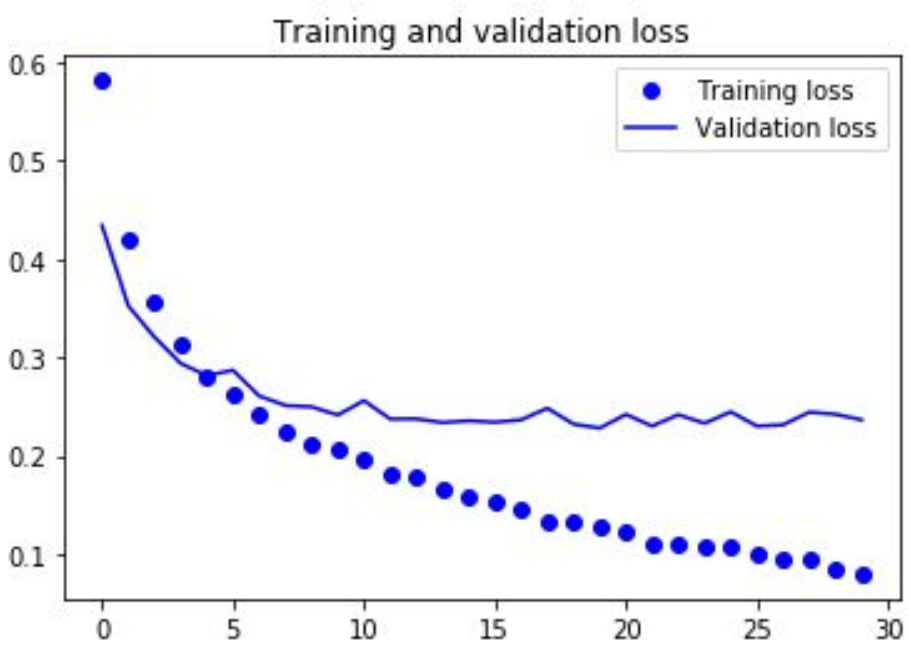
```
#%% Load data from binary files
train_features = np.load('train_features.npy')
train_labels = np.load('train_labels.npy')
validation_features = np.load('validation_features.npy')
validation_labels = np.load('validation_labels.npy')
test_features = np.load('test_features.npy')
test_labels = np.load('test_labels.npy')

#%% Reshape variables
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

```
#%% Listing 5.18 Defining and training the densely connected classifier
from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr = 2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

#history = model.fit(train_features, train_labels,
#                    epochs=30,
#                    batch_size=20,
#                    validation_data=(validation_features, validation_labels))
```

```
#%% Save the results
#model.save('cats_and_dogs_small_2.h5')
#np.save('cats_and_dogs_small_3.npy', history.history)

#%% Read back the results
model2 = load_model('cats_and_dogs_small_2.h5')
history = np.load('cats_and_dogs_small_3.npy')
```

```
#%% Listing 5.19 Plotting the results
acc = history.item().get('acc')
val_acc = history.item().get('val_acc')
loss = history.item().get('loss')
val_loss = history.item().get('val_loss')

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





- Validation accuracy of about 90%. This is much better than our small convnet trained from scratch.

- But the plots also indicate overfitting almost from the start.

FEATURE EXTRACTION
WITH DATA AUGMENTATION

```
#%% Listing 5.20 Adding a densely connected classifier on top of the convolutional
base
from keras import models
from keras import layers
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
conv_base.summary()
#conv_base.trainable = True

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'))
model.summary()
```

```
_________________________________________________________________
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
_________________________________________________________________
```

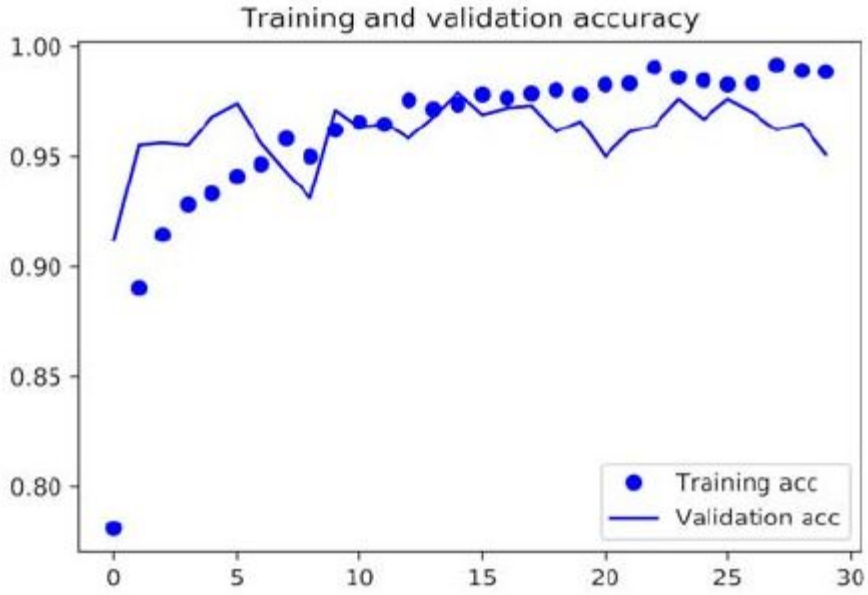
- CV base of VGG16 has 14,714,688 parameters!!
- The classifier on top has ~2 million parameters.

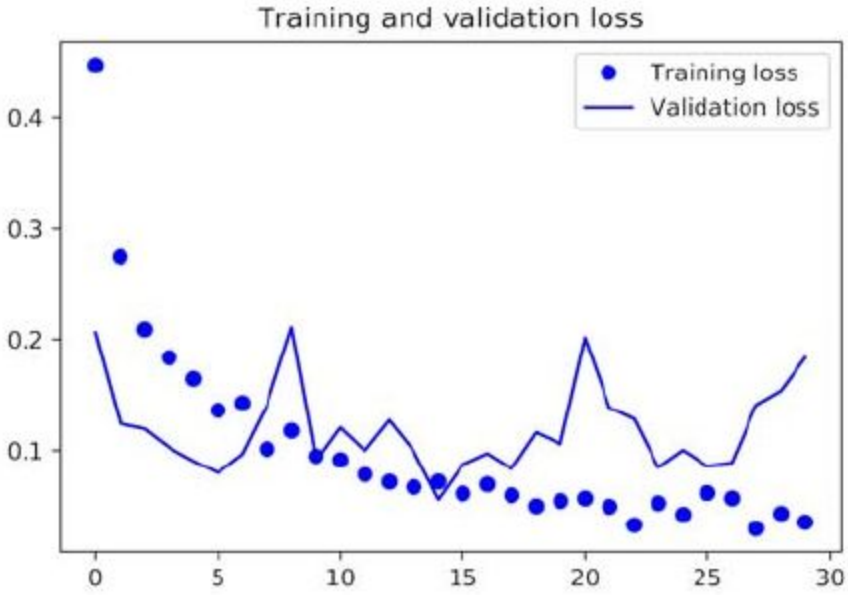
```
#%%
print('This is the number of trainable weights '
     'before freezing the conv base:', len(conv_base.trainable_weights))
conv_base.trainable = False
print('This is the number of trainable weights '
     'after freezing the conv base:', len(conv_base.trainable_weights))
model.summary()
```

```
#%% Listing 5.21 Training the model end to end with a frozen
convolutional base
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
train_datagen = ImageDataGenerator(
        rescale=1./255,
        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')

base_dir = './cats_and_dogs_small'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

```
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')

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['acc'])
```

```
#%%
history = model.fit_generator(
        train_generator,
        steps_per_epoch=100,
        epochs=30,

validation_data=validation_generator,
        validation_steps=50)
```

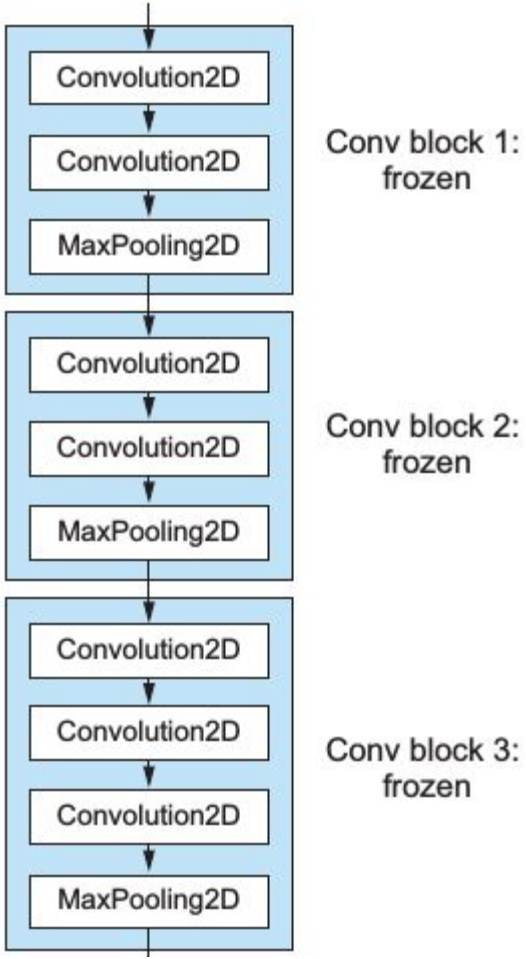

Training and validation accuracy
1.00
0.95
0.90
0.85
0.80
0
5
10
15
20
25
30
Training acc
Validation acc

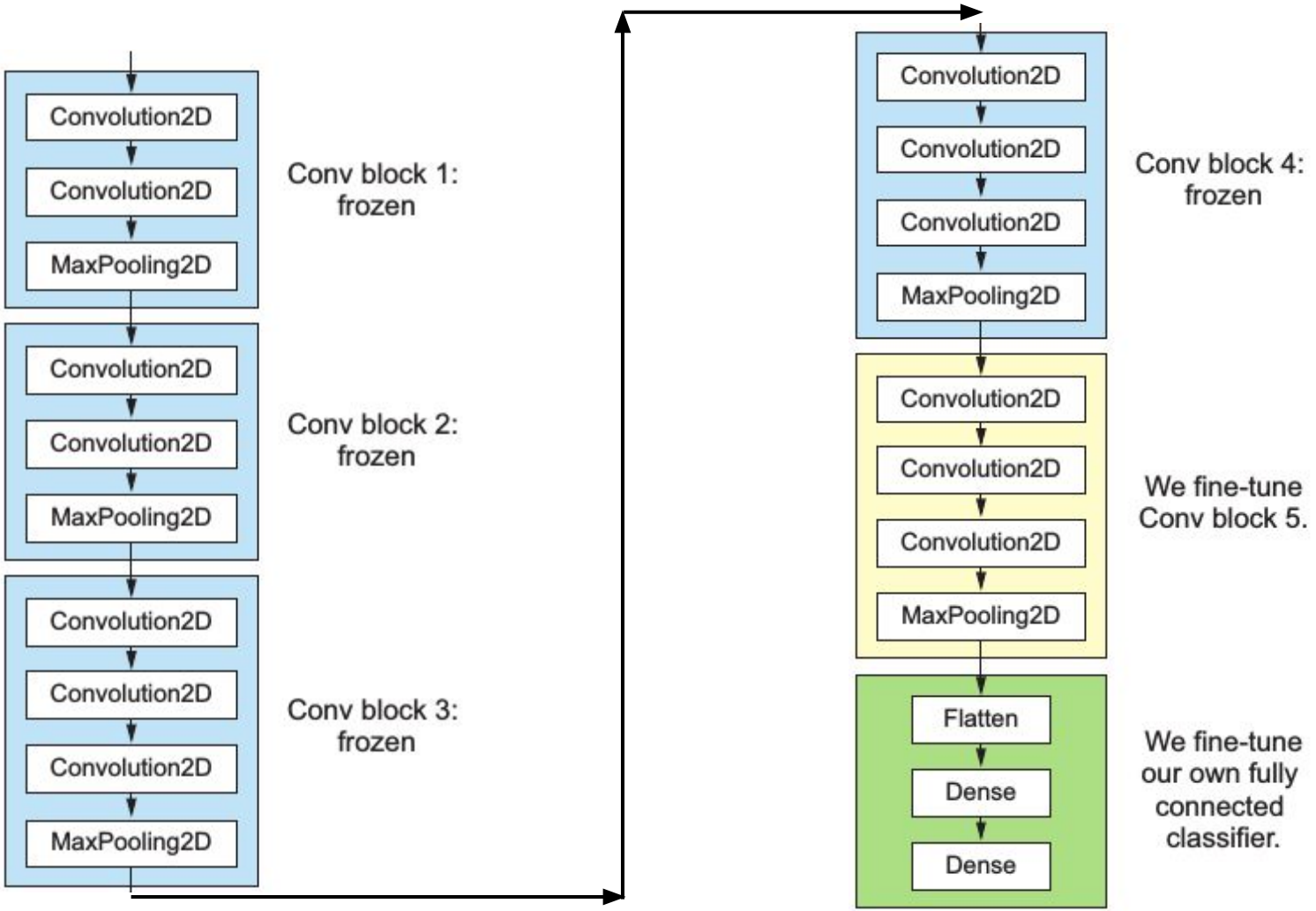

Training and validation loss
0.4
0.3
0.2
0.1
0
5
10
15
20
25
30
Training loss
Validation loss

Fine-tuning

Fine-Tuning

- Another widely used technique for model reuse.
- Unfreeze a few of the top layers of a frozen model base used for feature extraction.
- Jointly training both the newly added part of the model.
- Slightly adjusts the more abstract representations of the model being reused, in order to make them more relevant for the problem at hand.


Convolution2D
Convolution2D
MaxPooling2D
Conv block 1:
frozen
Convolution2D
Convolution2D
MaxPooling2D
Conv block 2:
frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 3:
frozen


Convolution2D
Convolution2D
MaxPooling2D
Conv block 1:
frozen
Convolution2D
Convolution2D
MaxPooling2D
Conv block 2:
frozen
Convolution2D
Convolution2D
MaxPooling2D
Conv block 3:
frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 4:
frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
We fine-tune
Conv block 5.
Flatten
Dense
Dense
We fine-tune
our own fully
connected
classifier.

Important

1. Add your custom network on top of an already-trained base network.
2. Freeze the base network.
3. Train the part you added (classifier).
4. Unfreeze some layers in the base network.
5. Jointly train both these layers and the part you added.

Let’s proceed to step 4!

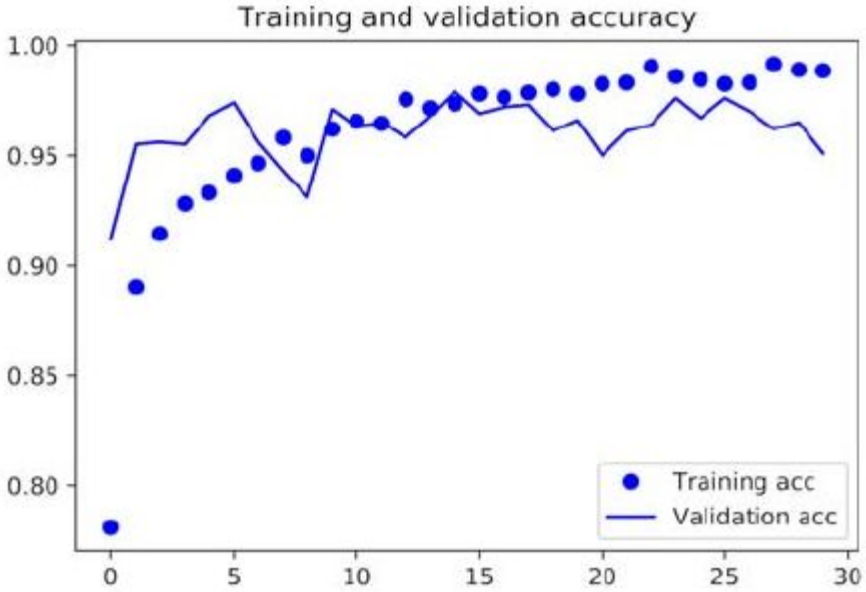
Attention

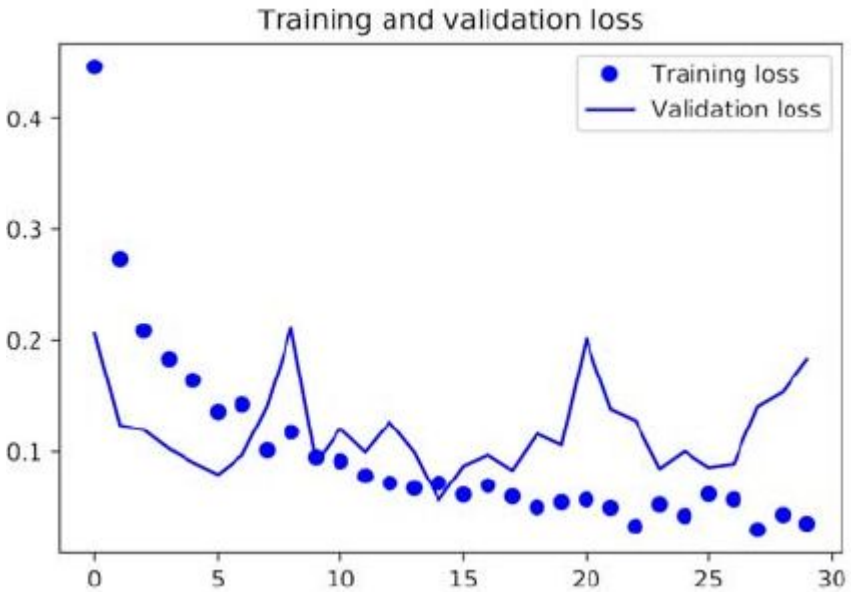
Why not fine-tune more layers? Why not fine-tune the entire convolutional base?

- Earlier layers = more-generic, reusable features.
- higher layers = more-specialized features.
- More specialized features are the ones that need to be repurposed on your new problem.
- There would be fast-decreasing returns in fine-tuning lower layers.
- The more parameters you’re training, the more you’re at risk of overfitting.
- The convolutional base has 15 million parameters, so it would be risky to attempt to train it on your small dataset.

```
#%%Listing 5.22 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
```

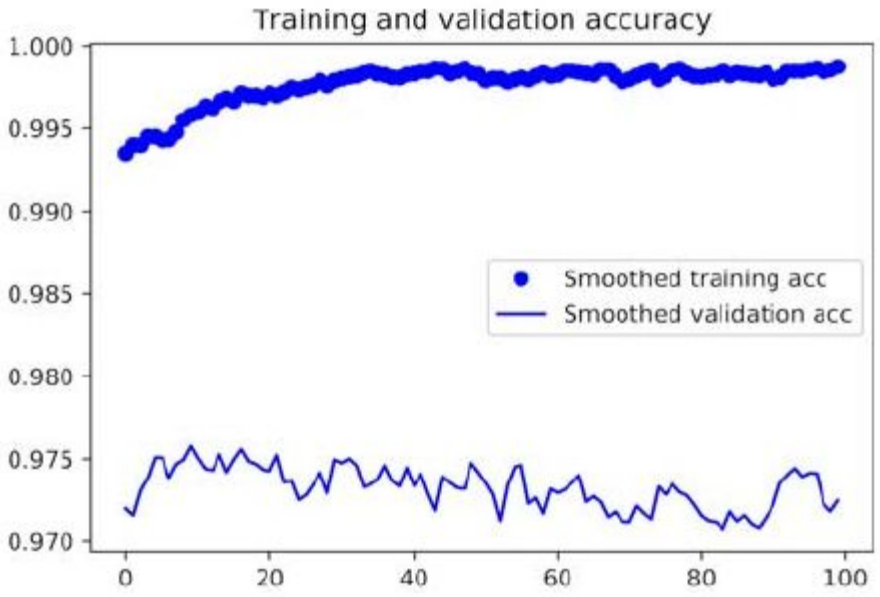
```
#%% Listing 5.23 Fine-tuning the model
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])
history = model.fit_generator(
     train_generator,
     steps_per_epoch=100,
     epochs=100,
     validation_data=validation_generator,
     validation_steps=50)

np.save('5-3-2-history-fine-tunning.npy', history)
model.save('5-3-2-fine-tunning.h5')
```

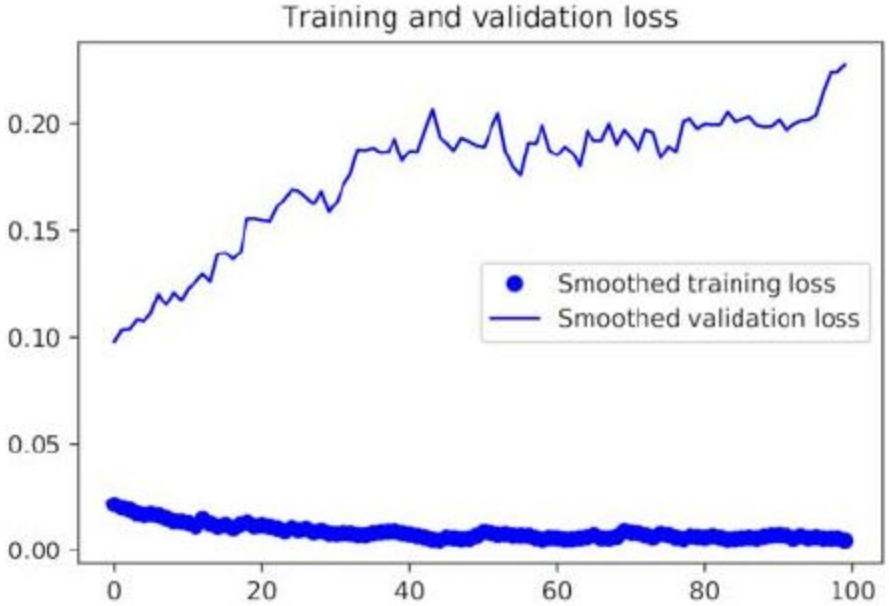

Training and validation accuracy
1.00
0.95
0.90
0.85
0.80
0
5
10
15
20
25
30
Training acc
Validation acc


Training and validation loss
0.4
0.3
0.2
0.1
0
5
10
15
20
25
30
Training loss
Validation loss

```
#%% Listing 5.24 Smoothing the plots
def smooth_curve(points, factor=0.8):
    smoothed_points = []
    for point in points:
    if smoothed_points:
        previous = smoothed_points[-1]
        smoothed_points.append(previous * factor + point * (1 - factor))
    else:
        smoothed_points.append(point)
    return smoothed_points
```

```
plt.plot(epochs,
smooth_curve(acc), 'bo', label='Smoothed training acc')
plt.plot(epochs,
smooth_curve(val_acc), 'b', label='Smoothed validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs,
smooth_curve(loss), 'bo', label='Smoothed training loss')
plt.plot(epochs,
smooth_curve(val_loss), 'b', label='Smoothed validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0
20
40
60
80
100
Smoothed training acc
Smoothed validation acc


Training and validation loss
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
Smoothed training loss
Smoothed validation loss

Evaluate on test data

```
test_generator = test_datagen.flow_from_directory(
test_dir,
target_size=(150, 150),
batch_size=20,
class_mode='binary')
test_loss, test_acc =
model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)
```

97%
Accuracy

Take away

- Convnets → best type of ML models for CV Tasks.
- Possible to train one from scratch even on a very small dataset, with decent results.
- On a small dataset, overfitting will be the main issue. Data augmentation is a powerful way to fight overfitting when you're working with image data.
- It's easy to reuse an existing convnet on a new DB via **feature extraction**.
- This is a valuable technique for working with small image datasets.
- As a complement to feature extraction, you can use **fine-tuning**, which adapts to a new problem some of the representations previously learned by an existing model. This pushes performance a bit further.

Visualizing what ConvNets Learn

Is ConvNet a black box?

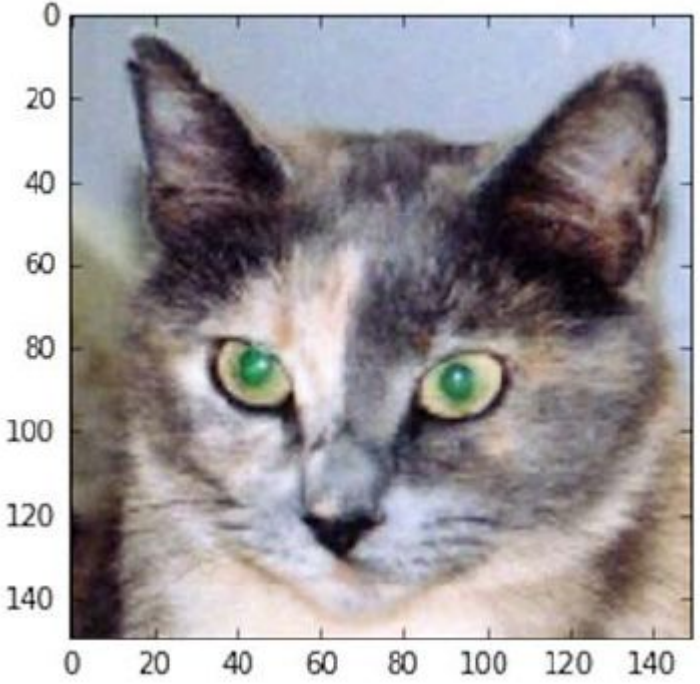
How to retrieve
VISUALIZATION
back from
REPRESENTATIONS?

Some strategies

1. Visualizing **intermediate convnet outputs** (intermediate activations) — How successive convnet layers transform their input, and for getting 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** — Understand which parts of an image were identified as belonging to a given class, thus allowing you to localize objects in images.

```
#%% Import modules
from keras.models import load_model
model = load_model('cats_and_dogs_small_2.h5')
model.summary() # As a reminder.

#%% Listing 5.25 Preprocessing a single image
img_path = './cats_and_dogs_small/test/cats/cat.1700.jpg'
from keras.preprocessing import image
#Preprocesses the image into a 4D tensor
import numpy as np
img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.
#Remember that the model was trained on inputs that
#Its shape is (1, 150, 150, 3) were preprocessed this way.
print(img_tensor.shape)
```

```
#%% Listing 5.26 Displaying the test picture
import matplotlib.pyplot as plt
plt.imshow(img_tensor[0])
plt.show()
```



```
#%% Listing 5.27 Instantiating a model from an input tensor and a list of output tensors
#Extracts the outputs of the top eight layers
layer_outputs = [layer.output for layer in model.layers[:8]]
# Creates a model that will return these outputs, given the model input
#When fed an image input, this model returns the values of the layer activations in the
#original model.
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

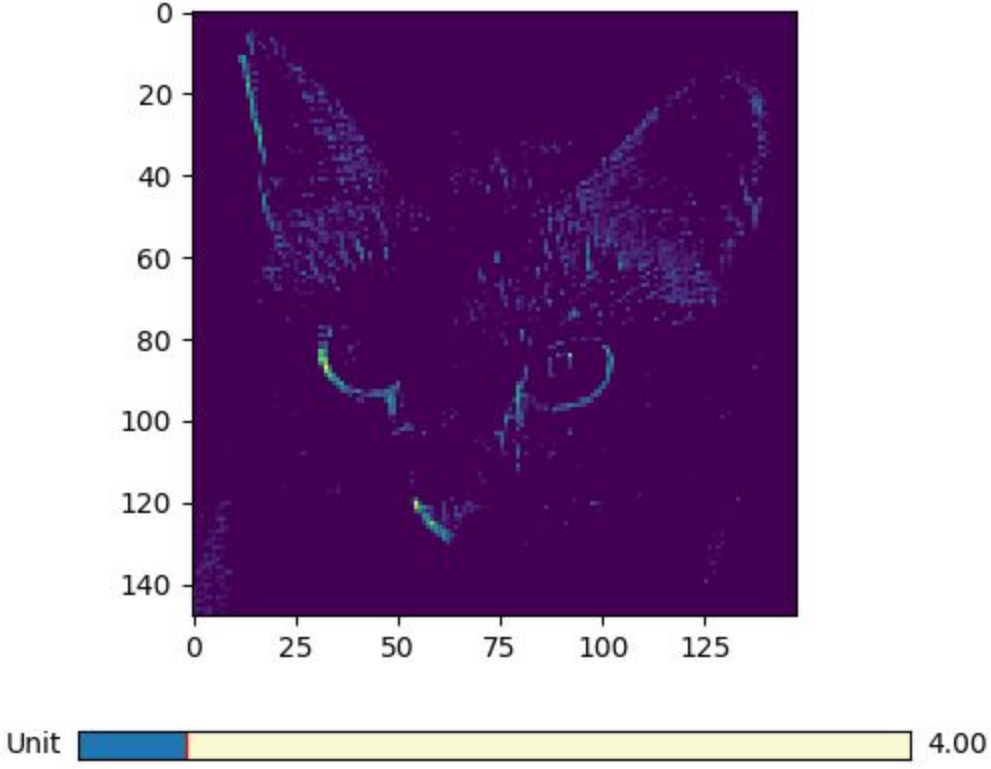
```
#%% Listing 5.29 Visualizing the fourth channel
import matplotlib.pyplot as plt
activations = activation_model.predict(img_tensor)
#print(first_layer_activation.shape)
unit = 4
layer = 0
fig, ax = plt.subplots()
plt.subplots_adjust(left=0.25, bottom=0.25)
first_layer_activation = activations[layer]
l = plt.imshow(first_layer_activation[0, :, :, unit], cmap='jet')
axUnit = plt.axes([0.25, 0.1, 0.65, 0.03], facecolor='lightgoldenrodyellow')
axLayer = plt.axes([0.25, 0.15, 0.65, 0.03], facecolor='lightgoldenrodyellow')
sUnit = Slider(axUnit, 'Unit', 0, 31, valinit=unit, valstep=1)
sLayer = Slider(axLayer, 'Layer', 0, 8, valinit=layer, valstep=1)
```

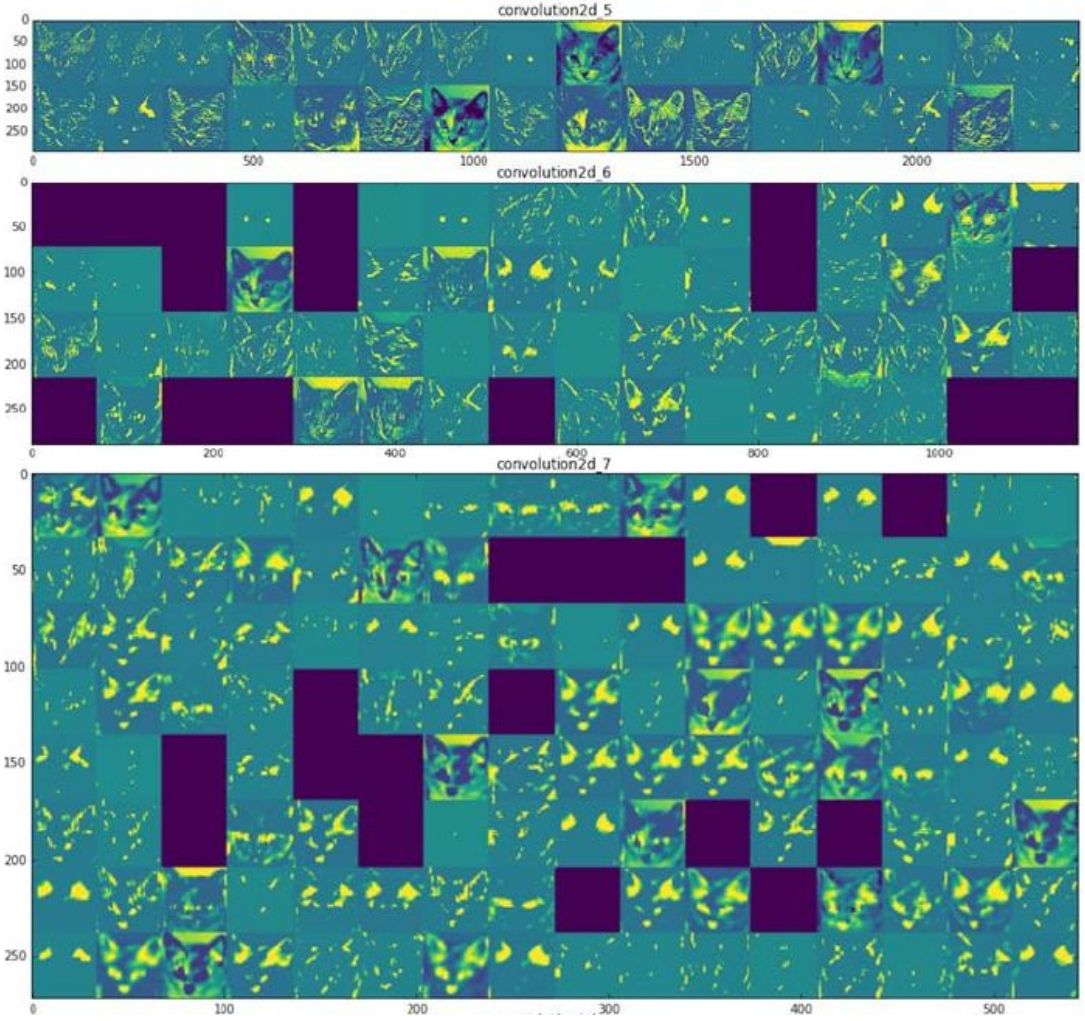
```
def update(val):
        unit = int(sUnit.val)
        layer = int(sLayer.val)
        first_layer_activation = activations[layer]
        l.set_data(first_layer_activation[0, :, :, unit])
        plt.draw()
sUnit.on_changed(update)
sLayer.on_changed(update)
plt.show()
```

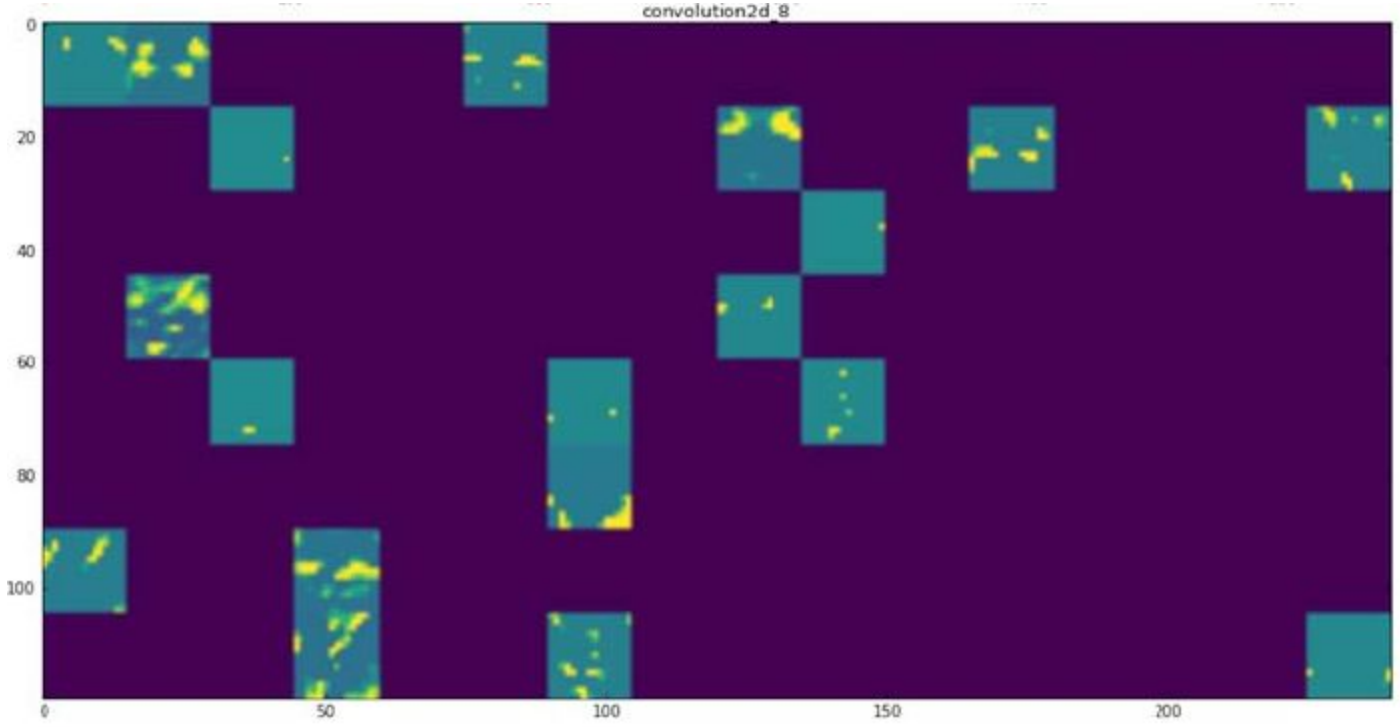
```
#%% Listing 5.31 Visualizing every channel in every intermediate activation
layer_names = []
for layer in model.layers[:8]:
        layer_names.append(layer.name)
images_per_row = 16
```

```
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()
    #Post-processes the feature to make it visually palatable
    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
#Displays the grid
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')
```


0
20
40
60
80
100
120
140
0
25
50
75
100
125
Unit
4.00


convolution2d_5
convolution2d_6
convolution2d_7


convolution2d_8
0
20
40
60
80
100
0
50
100
150
200

Note

- First layer → edge detectors. At that stage, the activations retain almost all of the information present in the initial picture.
- Higher → activations become increasingly abstract and less visually interpretable (“cat ear”, “cat eye”)
- Higher representations → less information about the visual contents of the image (more information related to the class of the image).
- Sparsity increases with the depth of the layer: in the first layer, all filters are activated by the input image; but in the following layers, more and more filter are blank (pattern encoded by the filter isn’t found in the input image).

Features extracted by a layer become increasingly **abstract** with the **depth** of the layer

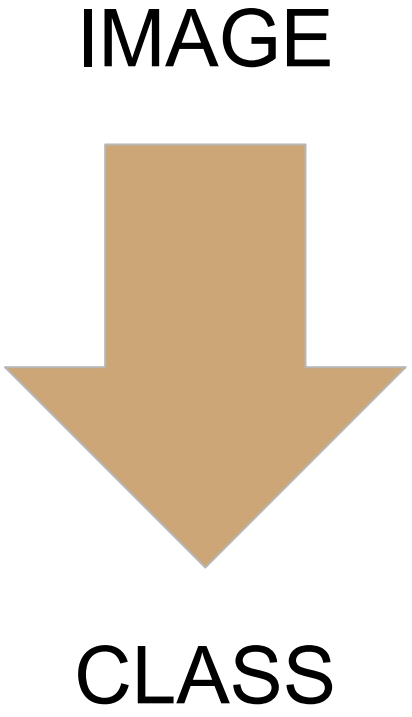
Deep Neural Network: *Information distillation pipeline*

IMAGE

Deep Neural Network: *Information distillation pipeline*

Deep Neural Network: *Information distillation pipeline*

Deep Neural Network: *Information distillation pipeline*

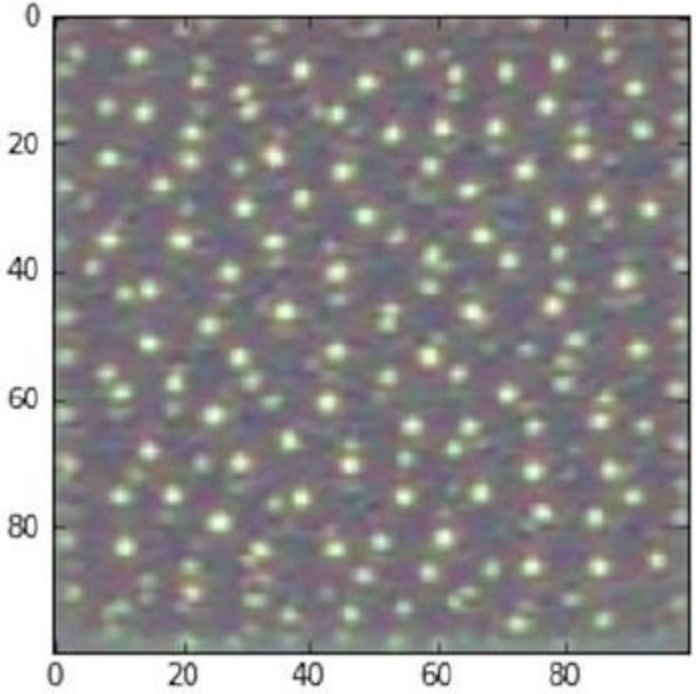


Visualizing convnet filters

```
#%% Listing 5.32 Defining the loss tensor for filter visualization
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])

#%% Listing 5.33 Obtaining the gradient of the loss with regard to the input
grads = K.gradients(loss, model.input)[0]

#%% Listing 5.34 Gradient-normalization trick
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

#%% Listing 5.35 Fetching Numpy output values given Numpy input values
iterate = K.function([model.input], [loss, grads])
import numpy as np
loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])
```

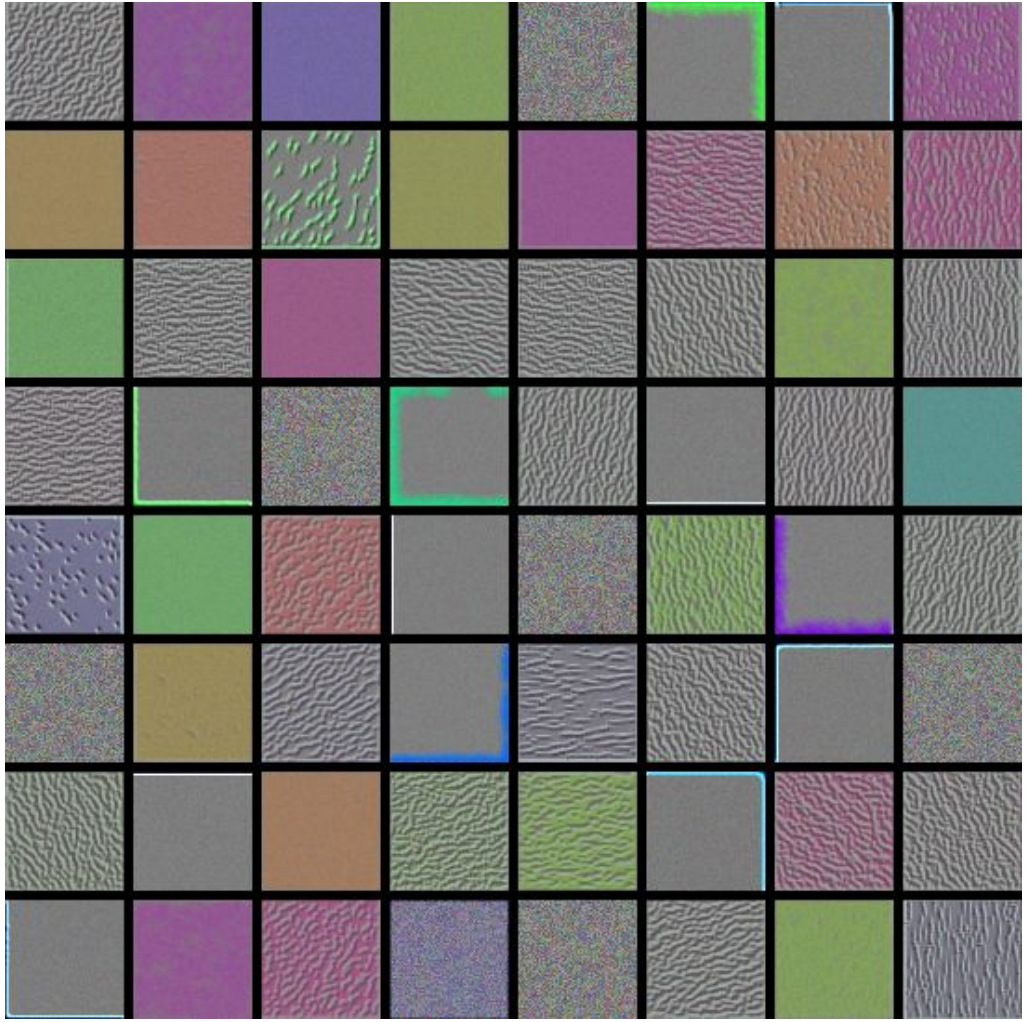
```
#%% Listing 5.36 Loss maximization via stochastic gradient descent
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.
step = 1.
for i in range(40):
    loss_value, grads_value = iterate([input_img_data])
    input_img_data += grads_value * step

#%% Listing 5.37 Utility function to 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)
    #Normalizes the tensor:
    #centers on 0,
    #ensures that std is 0.1
    #Clips to [0, 1]
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

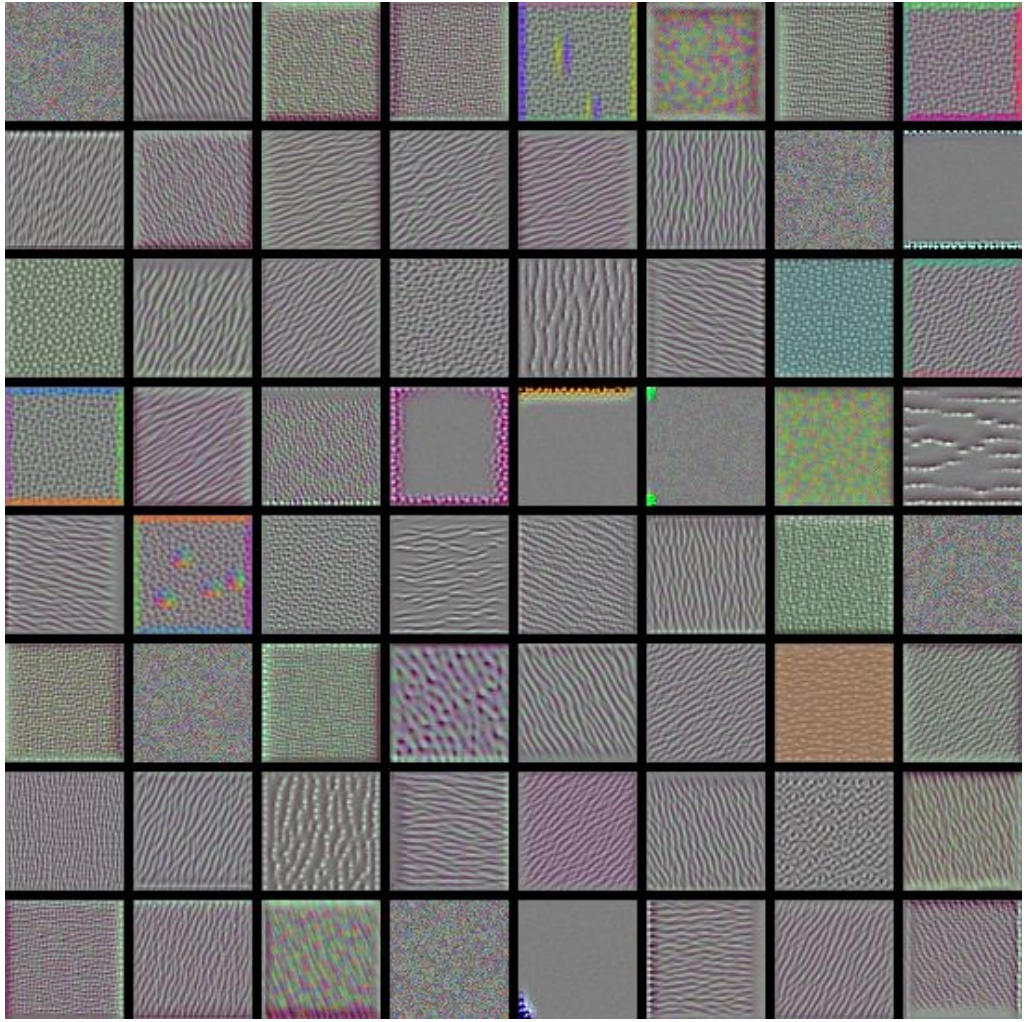
```
#%% Listing 5.38 Function to generate filter visualizations
def generate_pattern(layer_name, filter_index, size=150):
        layer_output = model.get_layer(layer_name).output
        loss = K.mean(layer_output[:, :, :, filter_index])

        grads = K.gradients(loss, model.input)[0]

        grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

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

        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)

plt.imshow(generate_pattern('block3_conv1', 0))
```

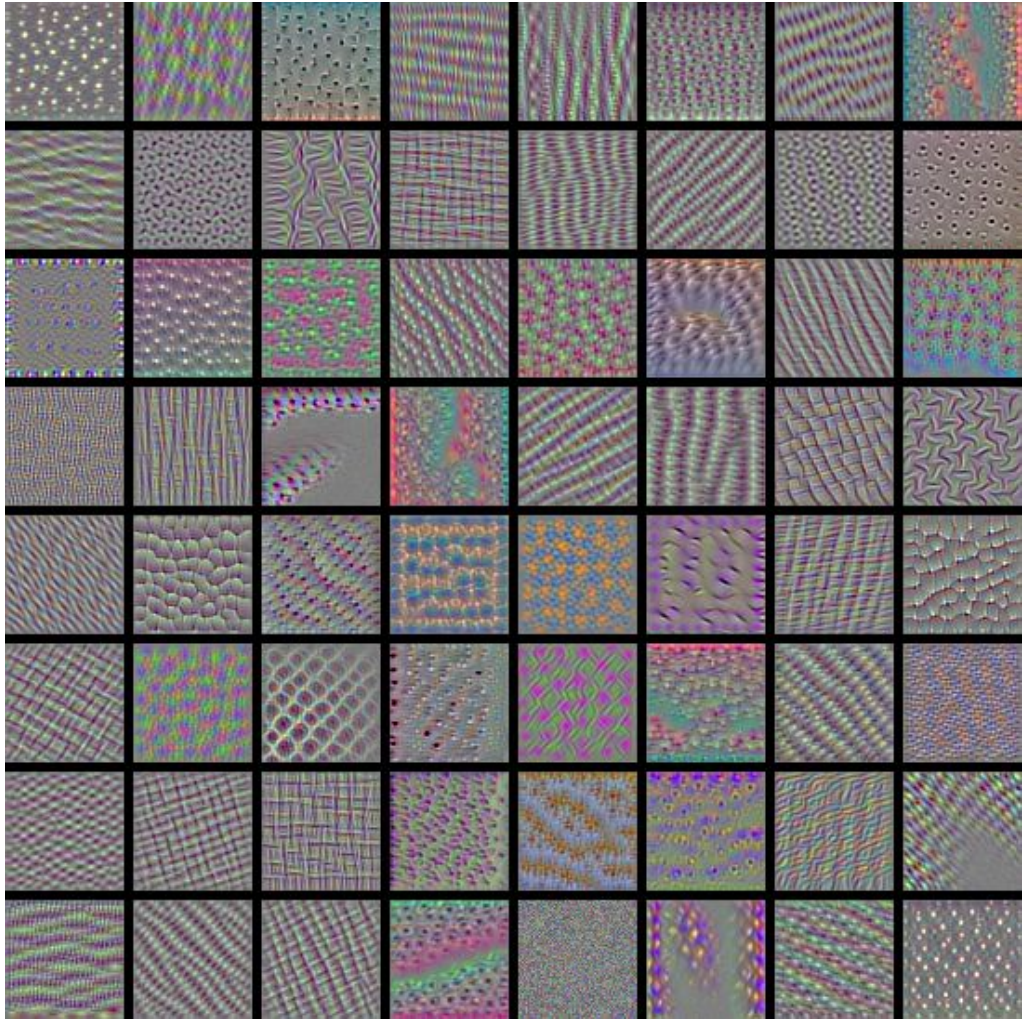

0
20
40
60
80
0
20
40
60
80

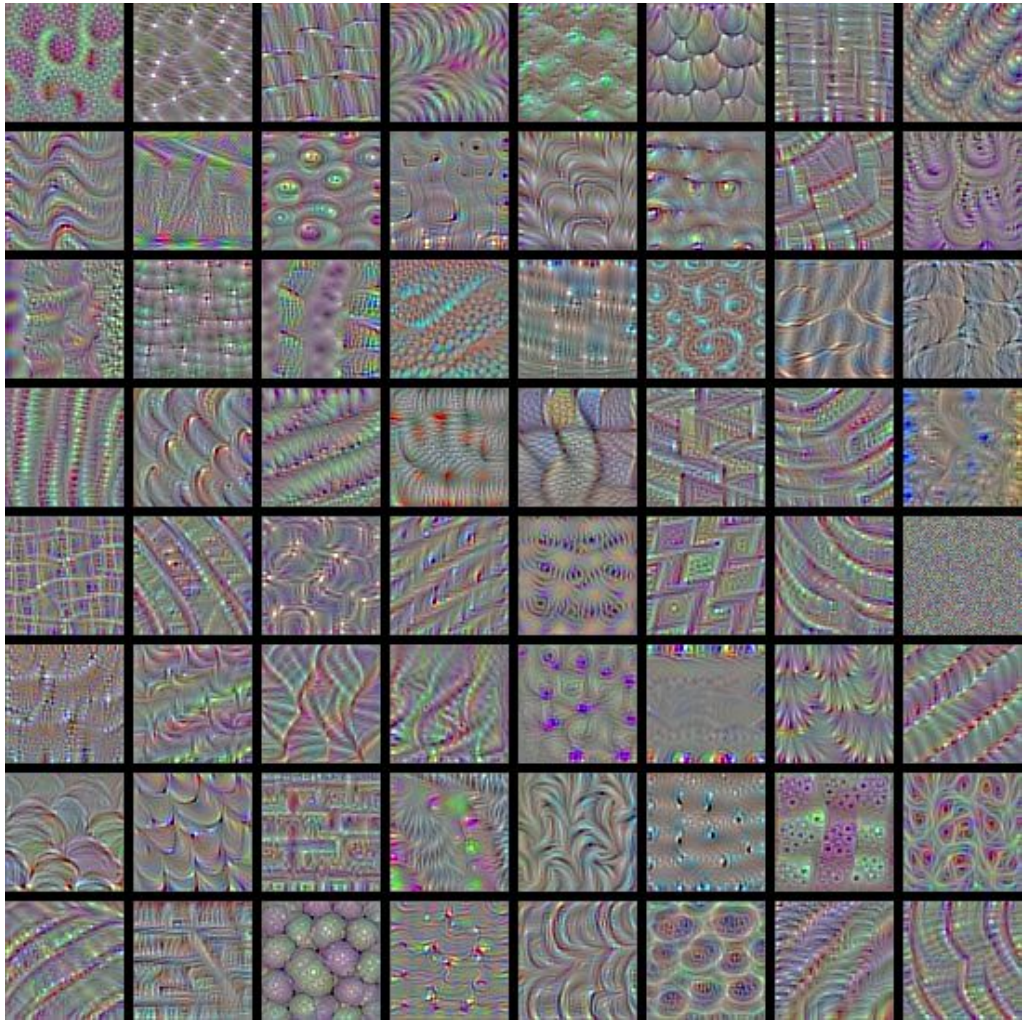
```
#%% Listing 5.39 Generating a grid of all filter response patterns in a layer
layer_name = 'block4_conv1'
size = 64
margin = 5

results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3))
for i in range(8):
    #Iterates over the rows of the results grid
    for j in range(8):
    #Iterates over the columns of the results grid
    filter_img = generate_pattern(layer_name, i + (j * 8), size=size)
    #Generates the
    #pattern for
    #filter i + (j * 8)
    #in layer_name
    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
```

```
plt.figure(figsize=(20,20))
plt.imshow(results.astype(np.uint8))
plt.show()

image.save_img(layer_name + '.png', results.astype(np.uint8))
```









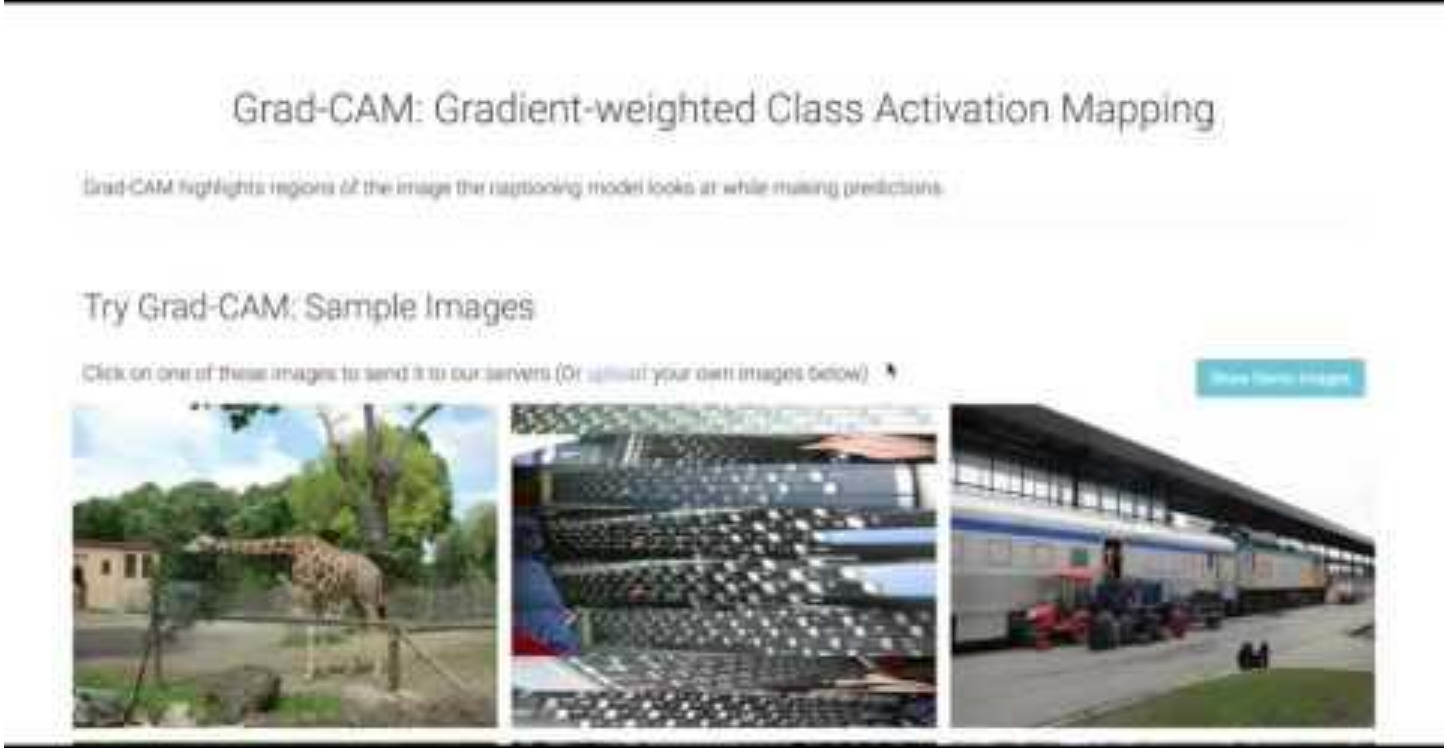
Take away

- The filters from the first layer in the model (block1_conv1) encode simple directional edges and colors (or colored edges, in some cases).
- The filters from block2_conv1 encode simple textures made from combinations of edges and colors.
- The filters in higher layers begin to resemble textures found in natural images:
    - feathers, eyes, leaves, and so on.

Visualizing heatmaps of class activation

Heatmaps of class activation

- *Class Activation Map* (CAM).
- Understanding which parts of a given image led a convnet to its final classification decision.
- Helpful for debugging.
- You can create one CAM for each category.
- *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization* https://arxiv.org/abs/1610.02391



youtu.be/COjUB9Izk6E

```
#%% Listing 5.40 Loading the VGG16 network with pretrained weights
from keras.applications.vgg16 import VGG16
model = VGG16(weights='imagenet')

#%% Listing 5.41 Preprocessing an input image for VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np
img_path = 'creative_commons_elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
preds = model.predict(x)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

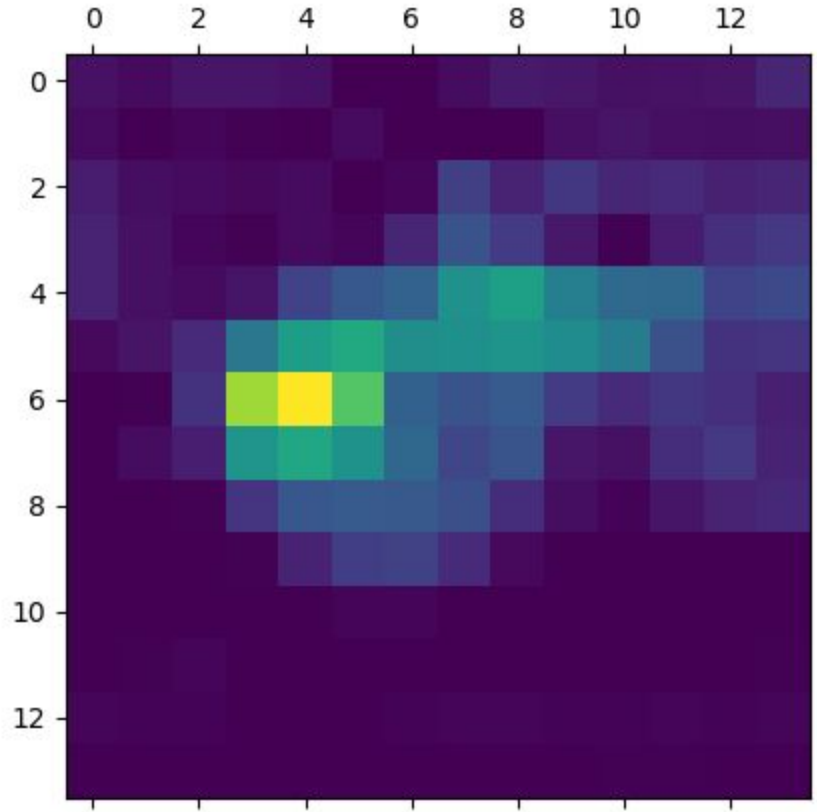
```
Predicted:', [(u'n02504458', u'African_elephant', 0.92546833),
(u'n01871265', u'tusker', 0.070257246),
(u'n02504013', u'Indian_elephant', 0.0042589349)]
```

The top three classes predicted for this image are as follows:

- African elephant (with 92.5% probability).
- Tusker (with 7% probability).
- Indian elephant (with 0.4% probability).

```
#%% Listing 5.42 Setting up the Grad-CAM algorithm
african_elephant_output = model.output[:, 386]
last_conv_layer = model.get_layer('block5_conv3')
grads = K.gradients(african_elephant_output,
last_conv_layer.output)[0]
pooled_grads = K.mean(grads, axis=(0, 1, 2))
iterate = K.function([model.input],
                     [pooled_grads, last_conv_layer.output[0]])
pooled_grads_value, conv_layer_output_value = iterate([x])
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
heatmap = np.mean(conv_layer_output_value, axis=-1)
```

```
#%% Listing 5.43 Heatmap post-processing
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)

#%% Listing 5.44 Superimposing the heatmap with the original
picture
import cv2
#Resizes the heatmap to
#Uses cv2 to load the
#be the same size as the
img = cv2.imread(img_path)
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
heatmap = np.uint8(255 * heatmap)
#Converts the heatmap to RGB
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
superimposed_img = heatmap * 0.4 + img
cv2.imwrite('elephant_cam.jpg', superimposed_img)
```


0
2
4
6
8
10
12
0
2
4
6
8
10
12

The visualization answers:

1. Why did the network think this image contained an African elephant!
2. Where is the African elephant located in the picture!

ConvNets Summary

Summary

1. Convnets are the best tool for attacking visual-classification problems.
2. Convnets work by learning a hierarchy of modular patterns and concepts to represent the visual world.
3. The representations they learn are easy to inspect—convnets are the opposite of black boxes!
4. You’re now capable of training your own convnet from scratch to solve an image-classification problem.

Summary

5. You understand how to use visual data augmentation to fight overfitting.
6. You know how to use a pretrained convnet to do feature extraction and fine-tuning.
7. You can generate visualizations of the filters learned by your convnets, as well as heatmaps of class activity.

References

1) François Chollet, “*Deep Learning with Python*”, 2018.