19) Convolutional Neural Networks: Padding, Strides, Max-Pooling

Vitor Kamada

August 2019

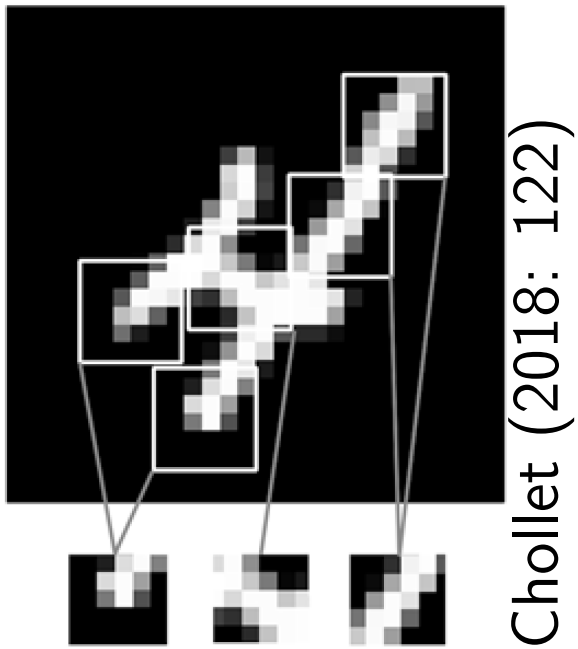
Reference

Chollet (2018): Ch 5.1

https://www.manning.com/books/deep-learning-with-python

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.1-introduction-to-convnets.ipynb

Global vs Local Patterns

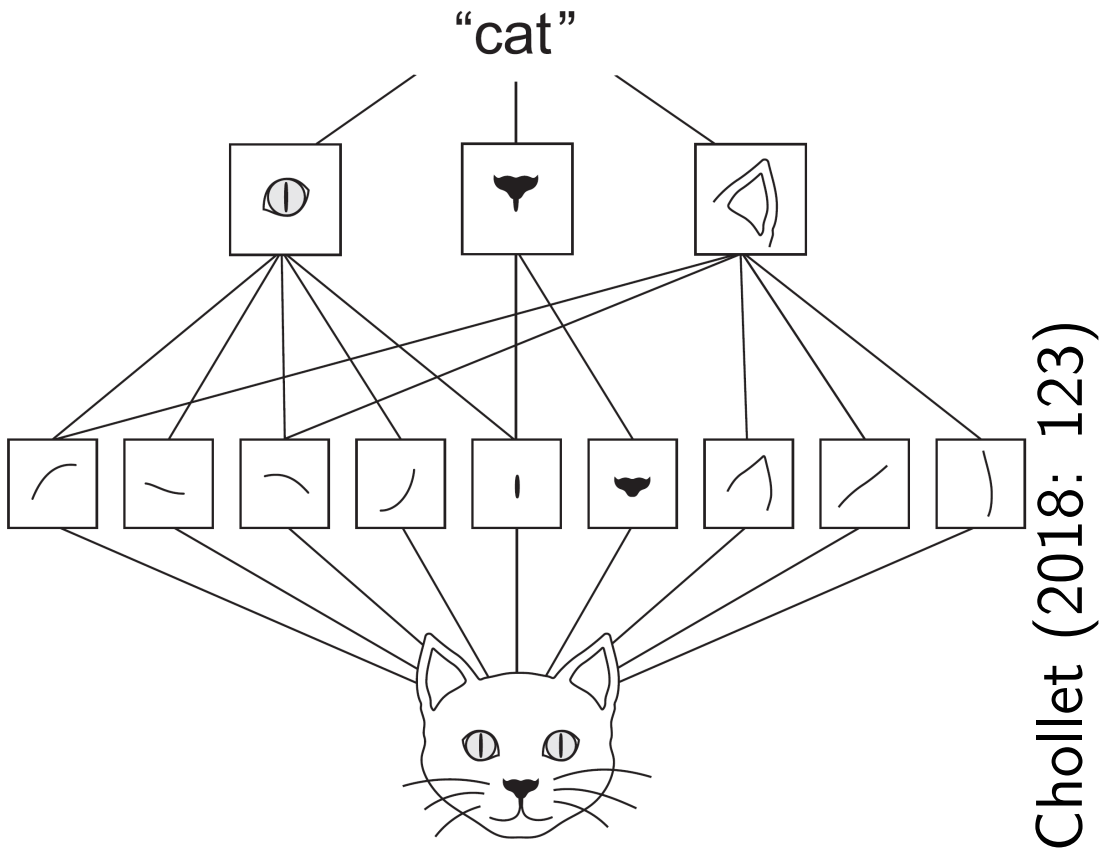


3D tensors (feature maps) = (28, 28, 1)

Spatial Axes (Height and Width) = 28x28 pixels

Depth Axis (Channels) =1 (levels of gray)

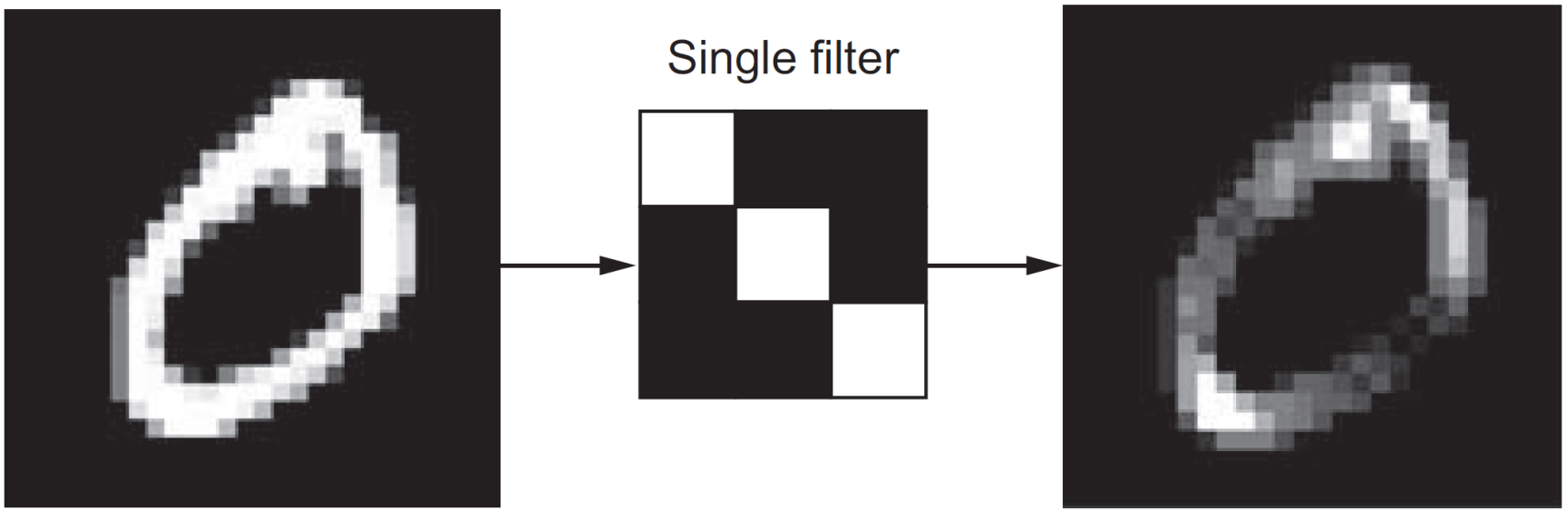
Spatial Hierarchies of Patterns



Depth is a parameter of the layer, and no longer stand for specific colors

Filters encode concepts: eyes, face

Feature Map of Size (28, 28, 1)



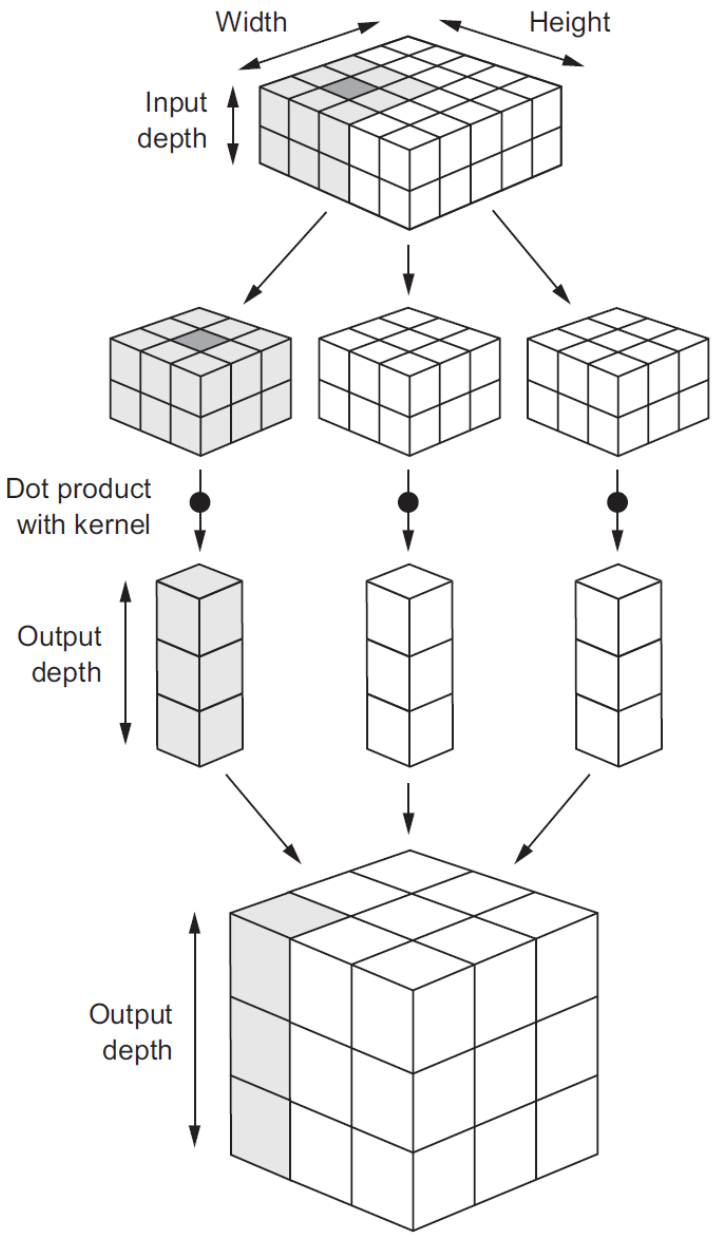
Chollet (2018: 124)

Size of the patches: 3×3

Depth of the output feature map: 32

Outputs a Feature Map of Size (26, 26, 32)

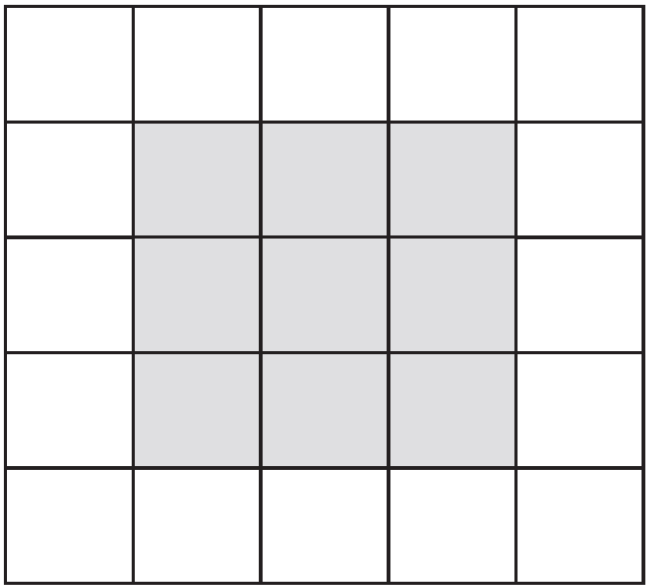
How Convolution Works?

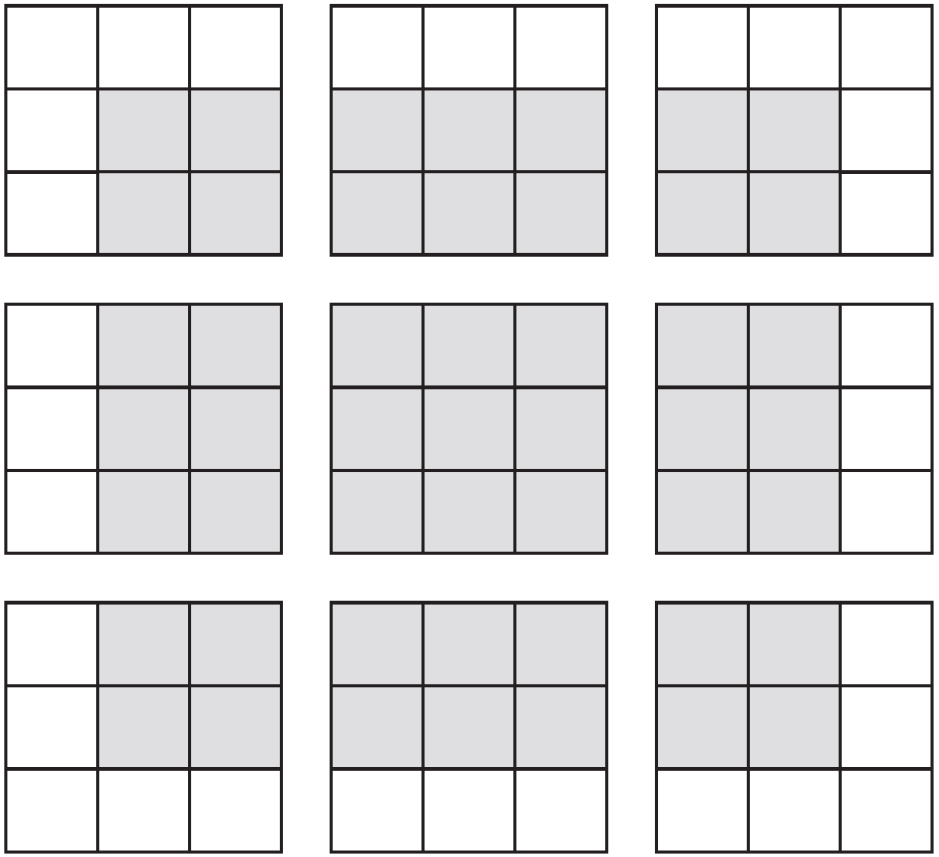


Chollet (2018: 125)

3×3 Patches in a 5×5 Input Feature Map

Border Effect: 5x5→3x3



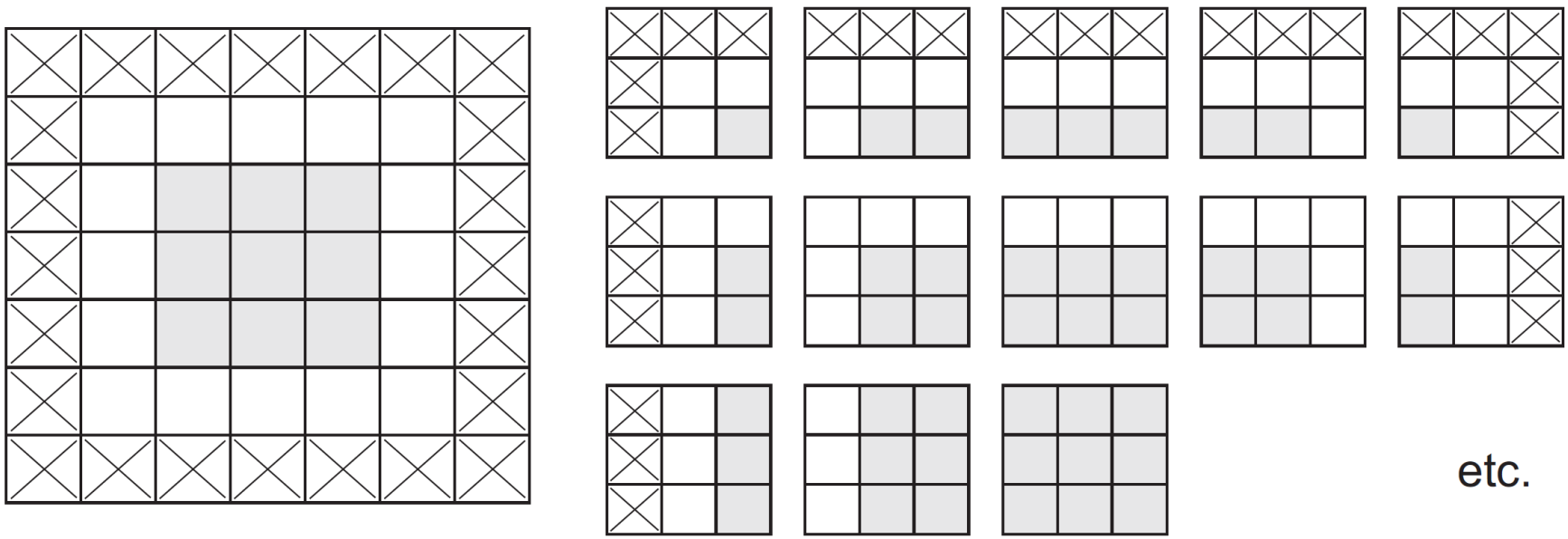


Chollet (2018: 126)

Padding a 5×5 input to extract 25 3×3 patches

In Conv2D layers:

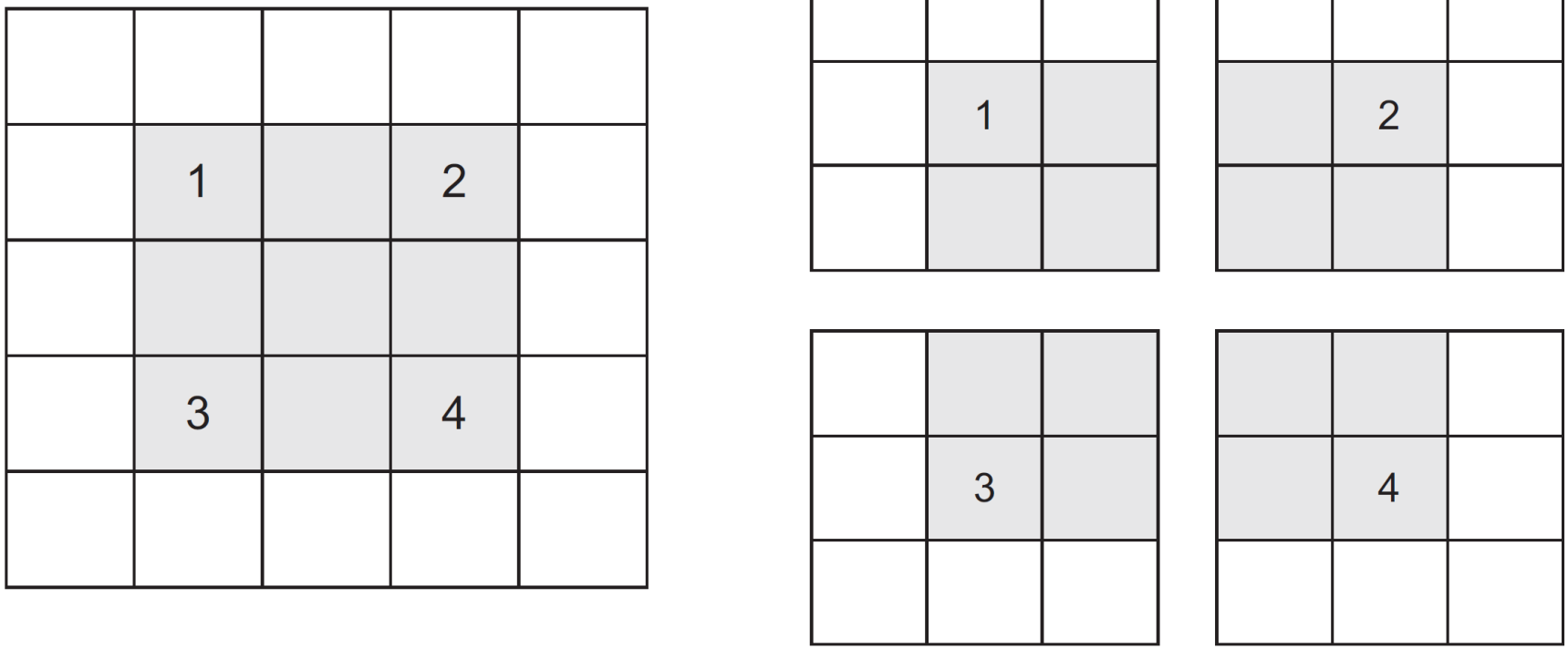
padding = "valid": means no padding



Chollet (2018: 126)

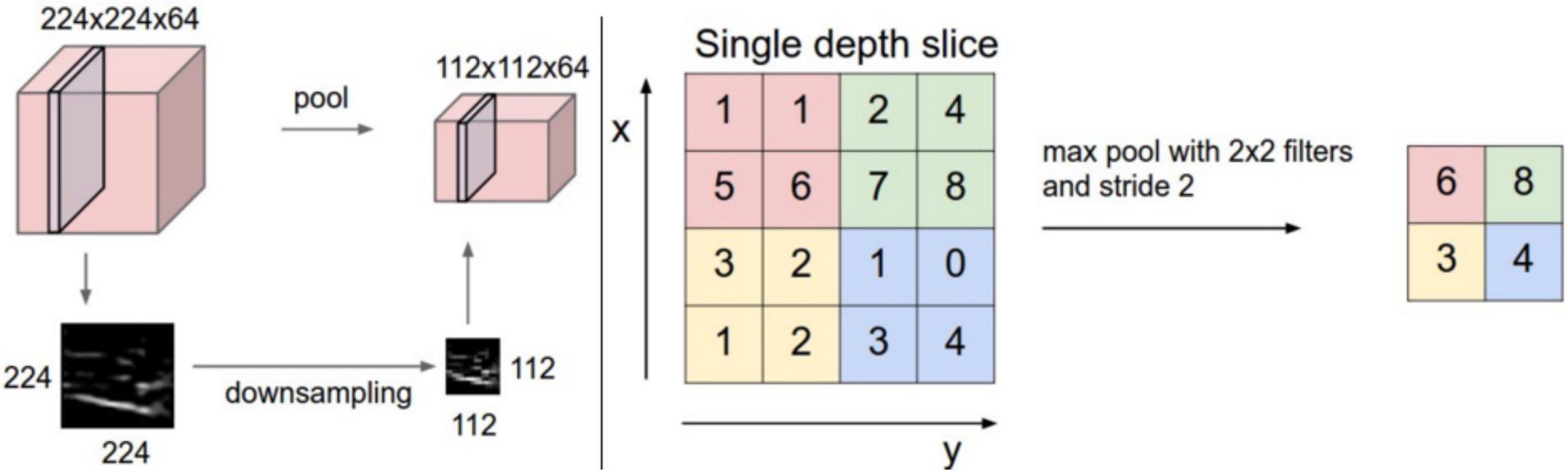
3×3 Convolution Patches with 2×2 Strides

Stride: Distance between two Successive Windows



Chollet (2018: 126)

Max-pooling Operation



Source: Stanford's CS231n github

```
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
```

(output_channels)*(input_channels*window_size + 1)

```
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
```

$$32*(1*3*3+1)=320$$

$$64*(32*3*3+1)=18496$$

MNIST Digits

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

```
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images,
                  test_labels) = mnist.load_data()

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'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
60000/60000 [==============================] - 8s - loss: 0.1766 - acc: 0.9440
Epoch 2/5
60000/60000 [==============================] - 7s - loss: 0.0462 - acc: 0.9855
Epoch 3/5
60000/60000 [==============================] - 7s - loss: 0.0322 - acc: 0.9902
Epoch 4/5
60000/60000 [==============================] - 7s - loss: 0.0241 - acc: 0.9926
Epoch 5/5
60000/60000 [==============================] - 7s - loss: 0.0187 - acc: 0.9943
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
test_acc
```

```
0.99129999
```