Chapter 4:

Deep learning in practice

Outlines

1. Convolutional networks
2. Recurrent neural networks
3. Commonly used deep learning architectures
4. How to build working deep learning model from scratch
5. Project and lab

Convolutional networks

1. The convolution operation:
   - ConvNet looks like a stack of Conv2D and MaxPooling2D layers.
   - Instantiating a small convent:

```
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'))
```

Convolutional networks

1. The convolution operation:
   - A convnet takes as input tensors of shape (image_height, image_width, image_channels) (not including the batch dimension).
   - In this case, we'll configure the convnet to process inputs of size (28, 28, 1), which is the format of MNIST images.
   - We'll do this by passing the argument input_shape=(28, 28, 1) to the first layer.

```
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)))
```

Convolutional networks

1. The convolution operation:
   - The architecture of the convnet:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

Convolutional networks

1. The convolution operation:

```
# Instantiating a small convnet
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.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

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

```
MaxPooling2D(Tuple[int, int])
```

Convolutional networks

1. The convolution operation:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

Convolutional networks

1. The convolution operation:
   - Training the convnet on MNIST images:

```
from tensorflow.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)

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

Convolutional networks

1. The convolution operation:
   - Training the convnet on MNIST images:

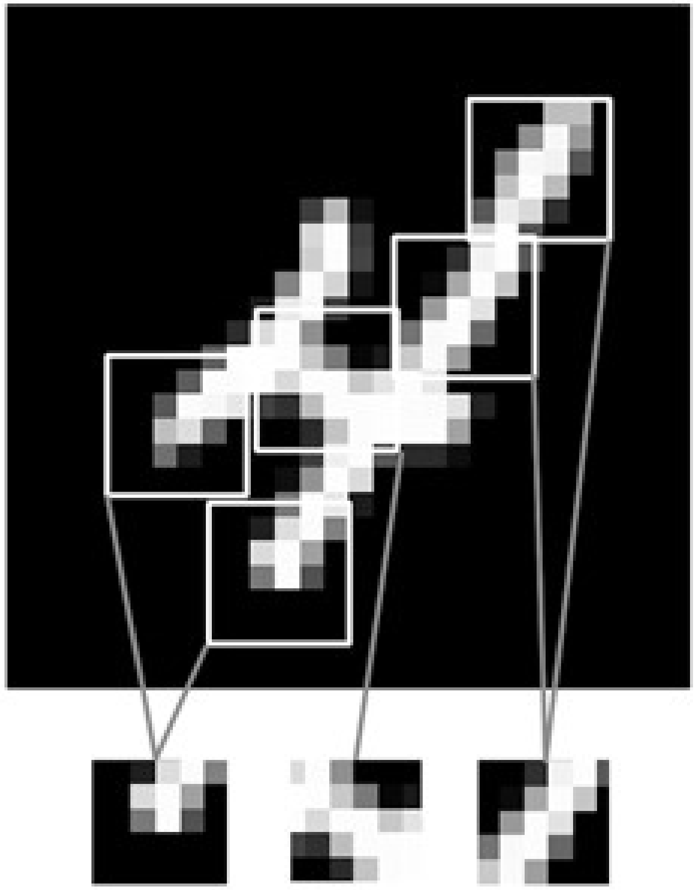
```
938/938 [==============================] - 15s 7ms/step - loss: 0.1647 - accuracy: 0.9488
Epoch 2/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0463 - accuracy: 0.9859
Epoch 3/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0308 - accuracy: 0.9906
Epoch 4/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0250 - accuracy: 0.9922
Epoch 5/5
938/938 [==============================] - 7s 7ms/step - loss: 0.0195 - accuracy: 0.9940
313/313 [==============================] - 1s 3ms/step - loss: 0.0299 - accuracy: 0.9913
0.9912999868392944
```

Convolutional networks

1. The convolution operation:
   - The fundamental difference between a densely connected layer and a convolution layer is this:
     - ✓ Dense layers learn global patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels).
     - ✓ Whereas convolution layers learn local patterns.
       - In the case of images, patterns are found in small 2D windows of the inputs.
       - For example: 3×3.

Convolutional networks

1. The convolution operation:
   - Images can be broken into local patterns such as edges, textures, and so on

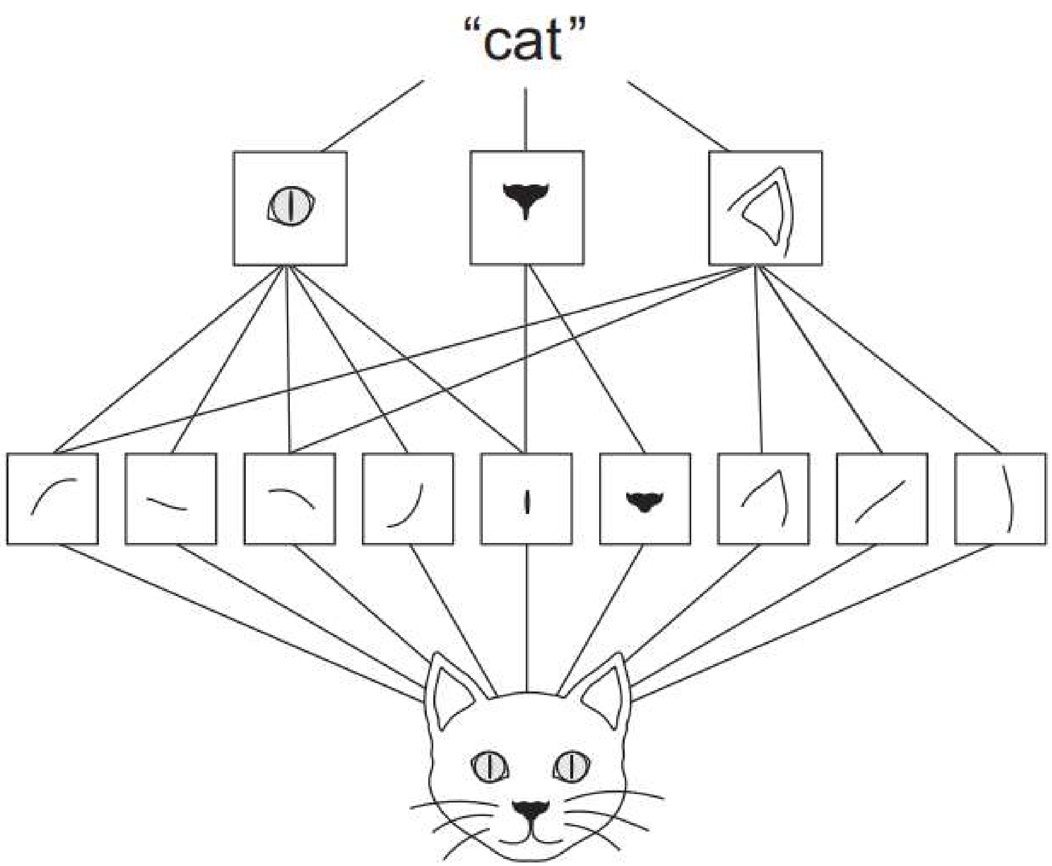


Convolutional networks

1. The convolution operation:
   - Two properties of convnets:
     - ✓ The patterns they learn are translation-invariant:
       - After learning a certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere.
     - ✓ They can learn spatial hierarchies of patterns:
       - A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on.

Convolutional networks

1. The convolution operation:
   - The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as “cat.”.



Convolutional networks

1. The convolution operation:
   - Convolutions operate over 3D tensors, called feature maps, with two spatial axes (height and width) as well as a depth axis (also called the channels axis).
   - The dimension of the depth axis is 3 for an RGB image because the image has three color channels: red, green, and blue.
   - The black-and-white picture's depth is 1.
     - ✓ The MNIST is a set of black-and-white pictures, so we set to 1 when reshape the train data.

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
```

Convolutional networks

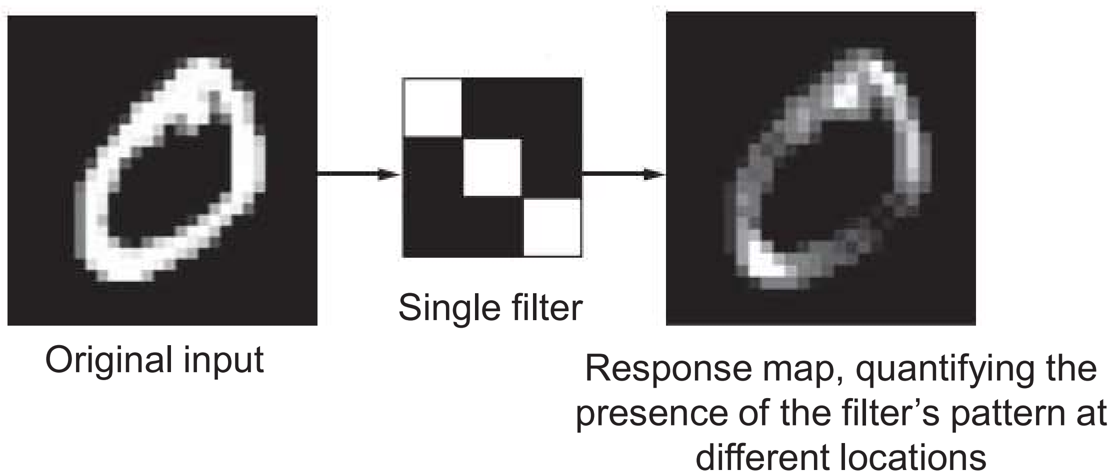
1. The convolution operation:
   - The first convolution layer takes a feature map of size (28, 28, 1) and outputs a feature map of size (26, 26, 32):
     - ✓ Computes 32 filters over its input.

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

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
```

Convolutional networks

1. The convolution operation:
   - The first convolution layer takes a feature map of size (28, 28, 1) and outputs a feature map of size (26, 26, 32):
     - Each of these 32 output channels contains a 26 $\times$ 26 grid of values, which is a response map of the filter over the input. => the response of that filter pattern atdifferent locations in the input => feature map



Convolutional networks

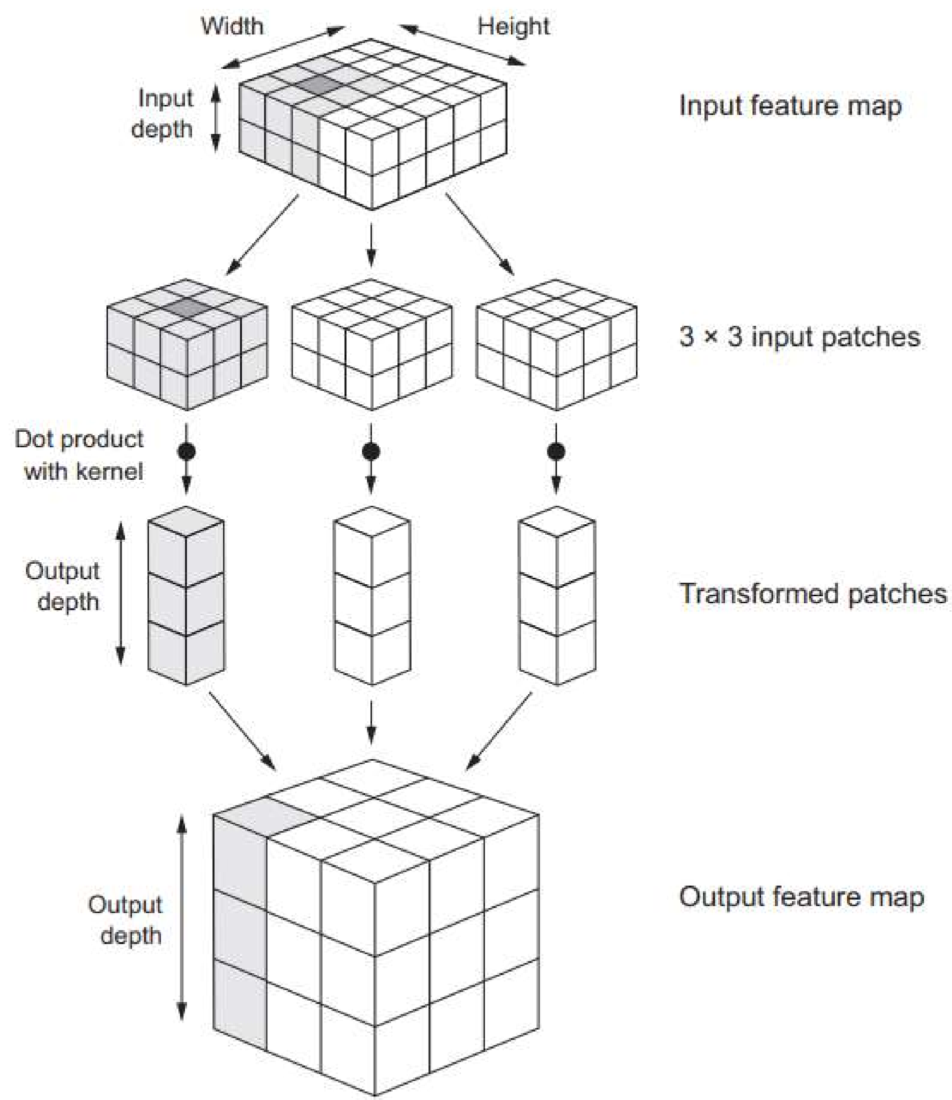
1. The convolution operation:
   - Convolutions are defined by two key parameters:
     - ✓ Size of the patches extracted from the inputs: These are typically 3×3 or 5×5.
     - ✓ Depth of the output feature map: The number of filters computed by the convolution.
       - The above example started with a depth of 32 and ended with a depth of 64

Convolutional networks

1. The convolution operation:
   - A convolution works by sliding these windows of size 3 × 3 or 5 × 5 over the 3D input feature map and extracting the 3D patch of surrounding features (shape (window_height, window_width, input_depth)).
   - Each such 3D patch is then transformed (via a tensor product with the same learned weight matrix, called the convolution kernel) into a 1D vector of shape (output_depth,).
   - All of these vectors are then spatially reassembled into a 3D output map of shape (height, width, output_depth).
   - Every spatial location in the output feature map corresponds to the same location in the input feature map.

Convolutional networks

1. The convolution operation:



Convolutional networks

2. The pooling operation :
   - The role of max pooling is to aggressively downsample feature maps.
   - Max pooling consists of extracting windows from the input feature maps and outputting the max value of each channel.
   - Max pooling transformed via a hardcoded max tensor operation.
   - A big difference from convolution is that max pooling is usually done with 2 $\times$ 2 windows and stride 2, in order to downsample the feature maps by a factor of 2.

Convolutional networks

2. The pooling operation :

```
# Instantiating a small convnet
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'))
```

```
MaxPooling2D (Tuple[int, int])
```

Convolutional networks

2. The pooling operation :

```
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)))
```

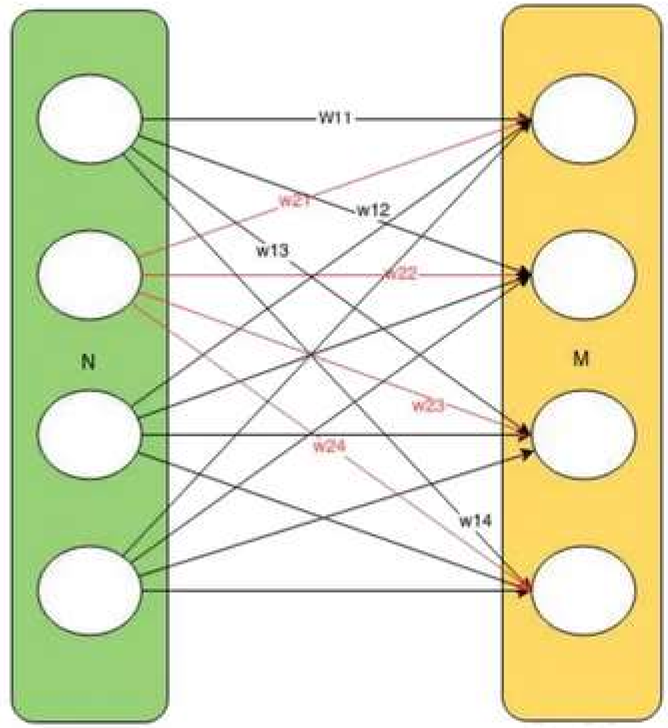
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
```

Convolutional networks

2. The pooling operation :
   - We can use average pooling instead of max pooling:
     - ✓ Each local input patch is transformed by taking the average value of each channel over the patch rather than the max value.
   - However, max pooling tends to work better than these alternative solutions.

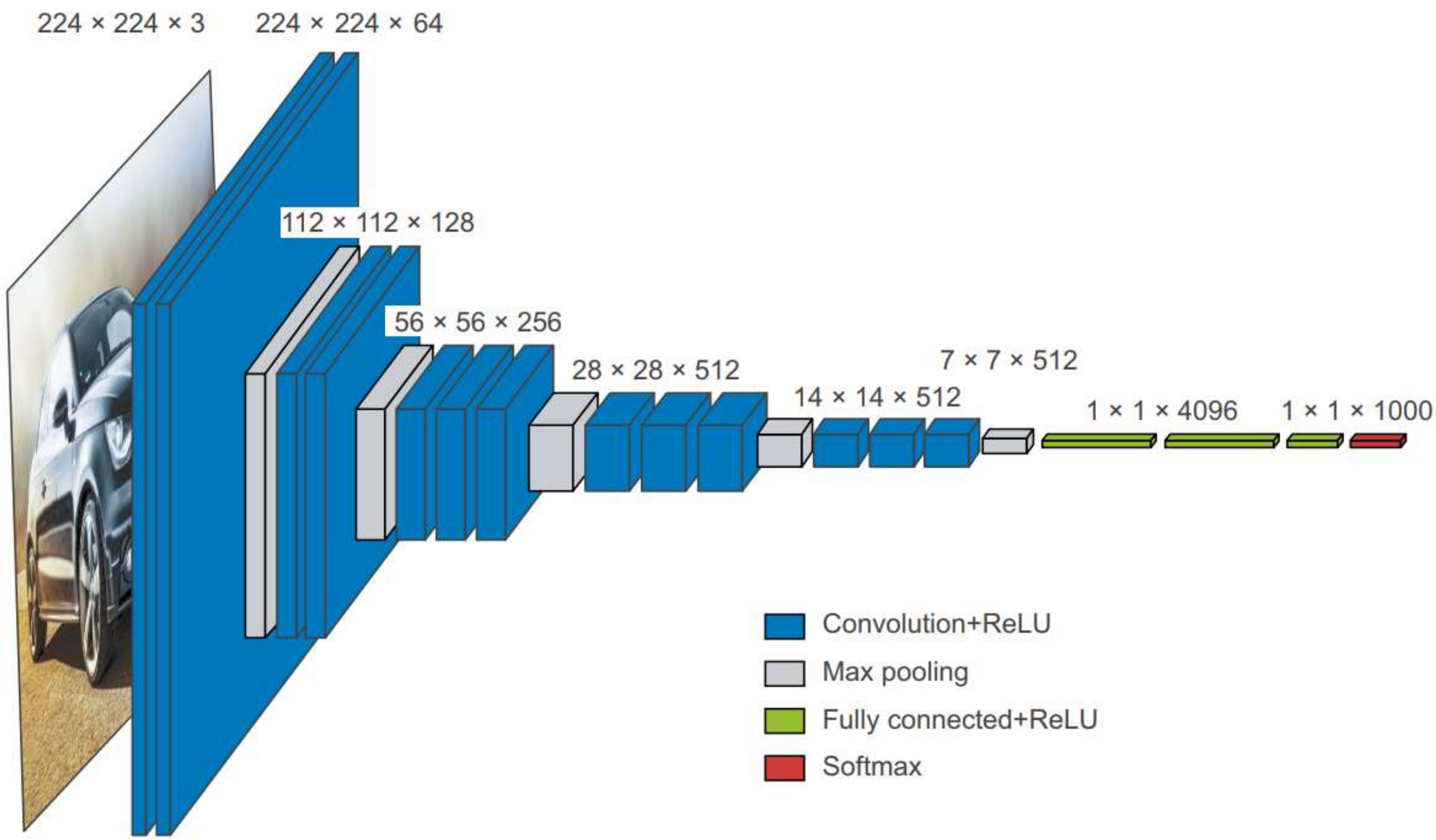
Convolutional networks

3. The fully connected operation:
    - This layer have full connection to all activations in the previous layer
    - Fully connected layers act as classifiers.
    - This fully connected layer is where the features extracted by convolution and pooling create the final result (classification or regression)



Convolutional networks

3. The fully connected operation:



Convolutional networks

4. Regularization, dropout operation:
   - Regularization
     - ✓ In deep learning, the models learned by neural networks are:
       - Given some training data and a network architecture, multiple sets of weight values (multiple models) could explain the data.
       - Simpler models are less likely to overfit than complex ones.

Convolutional networks

4. Regularization, dropout operation:
   - Regularization
     - ✓ To mitigate overfitting, we put constraints on the complexity of a network by forcing its weights to take only small values, which makes the distribution of weight values more regular.
       - ⇒ We called weight regularization, and it's done by adding to the loss function of the network a cost associated with having large weights.
     - ✓ This cost comes in two flavors:
       - L1 regularization: The cost added is proportional to the absolute value of the weight coefficients
       - L2 regularization: The cost added is proportional to the square of the value of the weight coefficients.

Convolutional networks

4. Regularization, dropout operation:
   - Regularization
     - ✓ In Keras, weight regularization is added by passing weight regularizer instances to layers as keyword arguments.

```
from keras import regularizers
```

```
# The model definition - regularization
model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

- ❑ l2(0.001) means every coefficient in the weight matrix of the layer will add 0.001 * weight_coefficient_value to the total loss of the network

Convolutional networks

4. Regularization, dropout operation:
   - Dropout
     - ✓ Dropout is a widely used regularization technique for neural networks, developed by Geoff Hinton and his team.
     - ✓ Dropout involves randomly setting a fraction of a layer's output features to zero during training to prevent overfitting.
     - ✓ The dropout rate, typically between 0.2 and 0.5, determines the proportion of features dropped.
     - ✓ During testing, no units are dropped; instead, the outputs are scaled down to compensate for the increased number of active units.

Convolutional networks

4. Regularization, dropout operation:
   - Dropout
     - Example:
       - Given a vector [0.2, 0.5, 1.3, 0.8, 1.1] for a given input sample during training.
       - After applying dropout, this vector will have a few zero entries distributed at random:
       - For example, [0, 0.5, 1.3, 0, 1.1].

Convolutional networks

4. Regularization, dropout operation:
   - Dropout
     - ✓ Example 2:
       - Dropout applied to an activation matrix at training time, with rescaling happening during training. At test time, the activation matrix is unchanged.

0.3	0.2	1.5	0.0
0.6	0.1	0.0	0.3
0.2	1.9	0.3	1.2
0.7	0.5	1.0	0.0

50% dropout

0.0	0.2	1.5	0.0
0.6	0.1	0.0	0.3
0.0	1.9	0.3	0.0
0.7	0.0	0.0	0.0

* 2

Convolutional networks

4. Regularization, dropout operation:
   - Dropout
     - ✓ In Keras, we can introduce dropout in a network via the Dropout layer, which is applied to the output of the layer right before it::

```
model.add(layers.Dropout(0.5))
```

Convolutional networks

4. Regularization, dropout operation:
   - Dropout
     - ✓ Example 3::

```
# The model definition - dropout
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```

Convolutional networks

4. Regularization, dropout operation:
   - Dropout
     - ✓ To recap, these are the most common ways to prevent overfitting in neural networks:
       - Get more training data.
       - Reduce the capacity of the network.
       - Add weight regularization.
       - Add dropout.