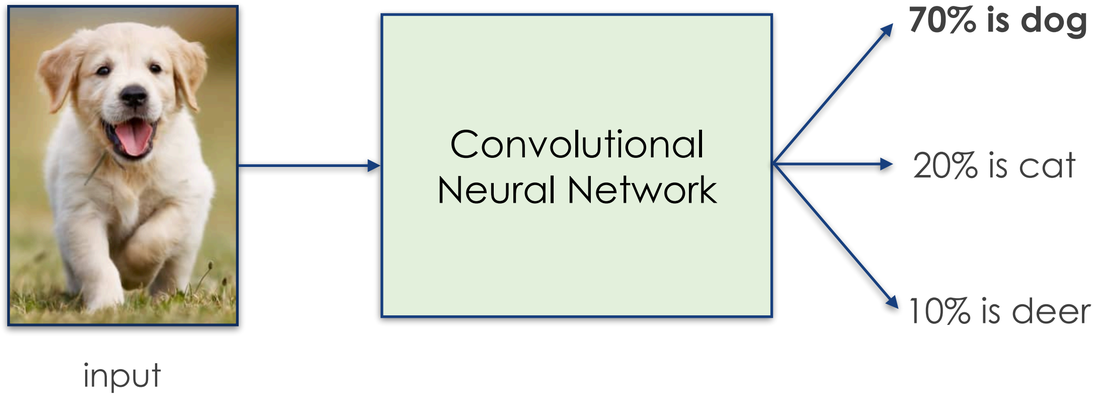
Python Programming and Machine Learning

Convolutional Neural Network

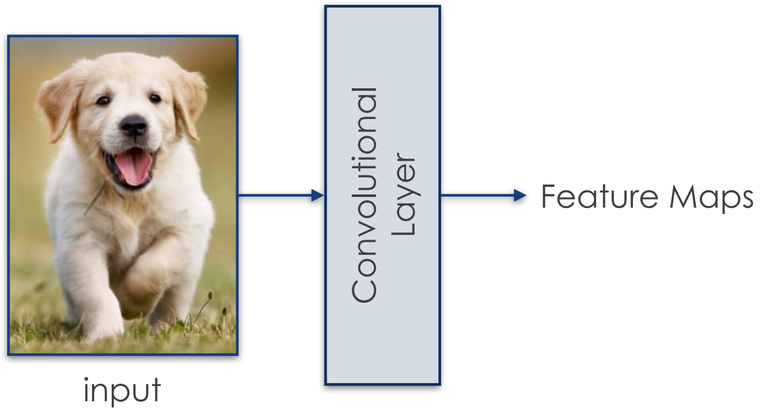
Overview

- A Convolutional Neural Network (CNN) is generally used to analyze visual images
- Given an image, a trained CNN can predict the animal in the image



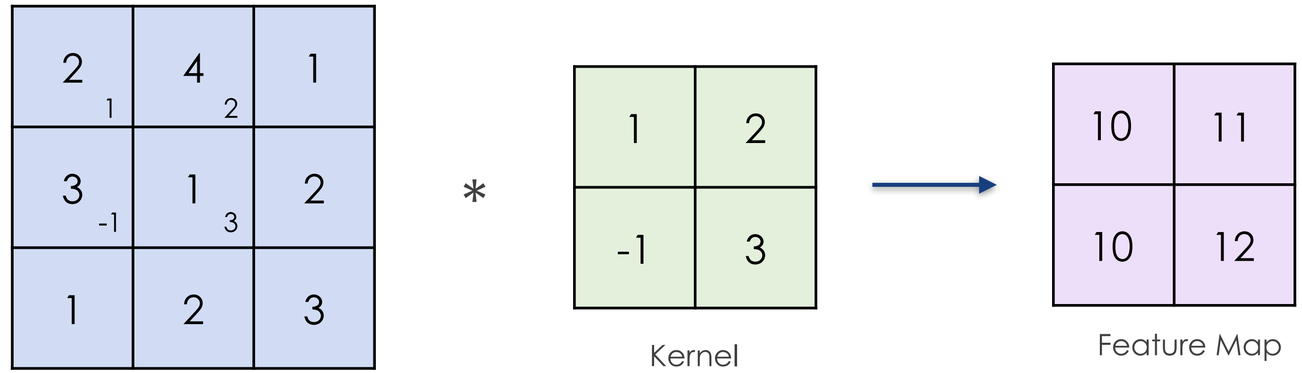
Convolutional Layer

- The Convolutional Layer extracts features from the input to yield Feature Maps
- A Feature Map is a 2D vectors of values computed by a convolutional operation



Convolutional Operation

- A convolutional operation slides a kernel over a 2D input vector
- A **kernel** is simply a matrix of weights
- The kernel is systematically applied to regions of its input to compute aggregated values over each region



2 * 1 + 4 * 2 + 3 * -1 + 1 * 3 = **10**

4 * 1 + 1 * 2 + 1 * -1 + 2 * 3 = **11**

3 * 1 + 1 * 2 + 1 * -1 + 2 * 3 = **10**

1 * 1 + 2 * 2 + 2 * -1 + 3 * 3 = **12**

Padding

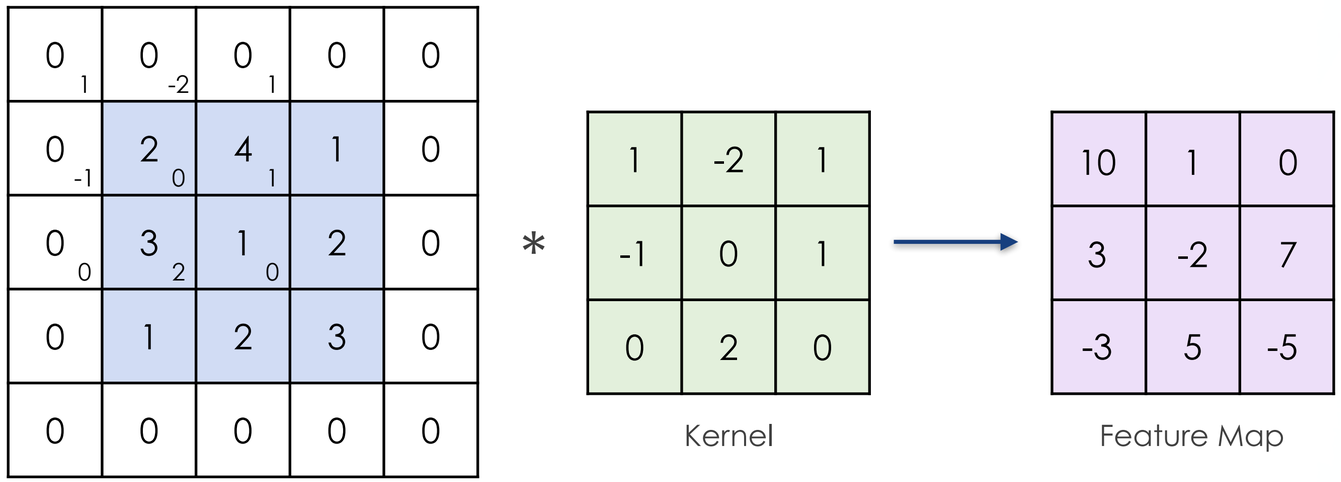
- A convolutional operation has a shrinking effect as the dimensions of the output matrix is usually smaller than the input matrix
- The dimensions of the output matrix can be maintained via padding
- Padding adds borders of 0's to an input matrix before the operation

0	0	0	0	0
0	2	4	1	0
0	3	1	2	0
0	1	2	3	0
0	0	0	0	0

Input (from previous layer)

Padding

- When the input matrix is padded, the dimension of the output matrix can match the input matrix in a convolutional operation



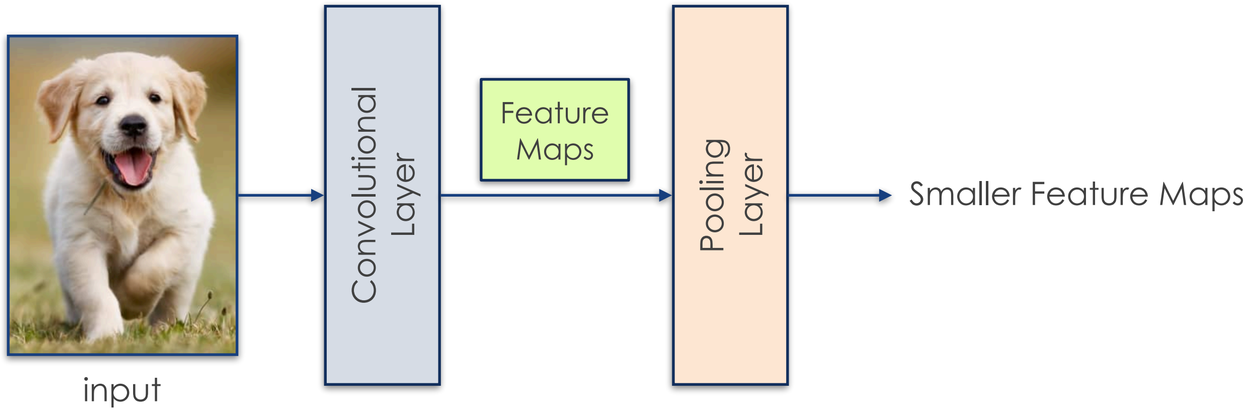
Input (from previous layer)

Kernel

Feature Map

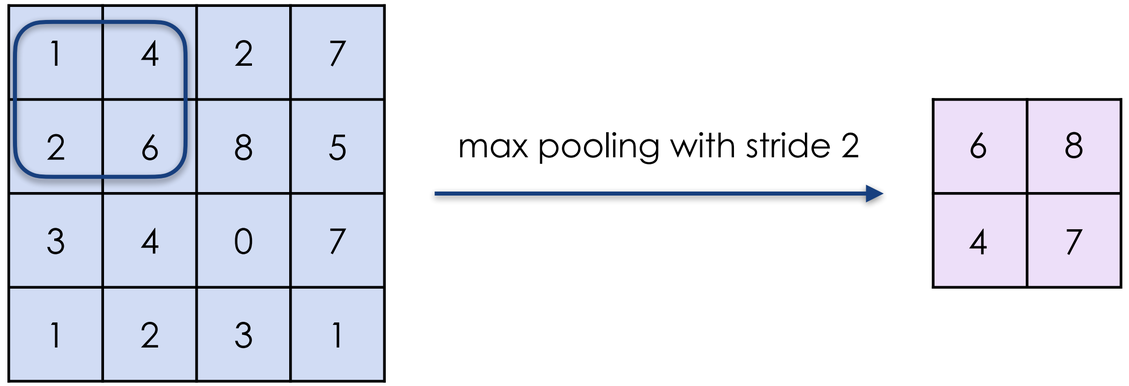
Pooling Layer

- Pooling summarizes regions of a Feature Map, hence reducing the overall dimensions of data that a network has to work with
- Hence, it reduces resource-usage (e.g. memory and GPU cycles) and allows our network to train faster



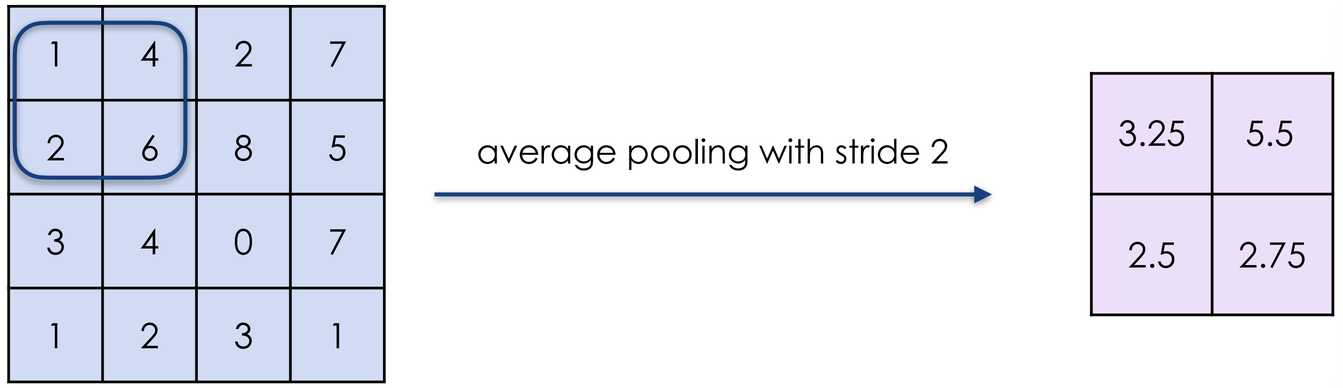
Max Pooling

- Max Pooling extracts the maximum value of a region in a Feature Map



Average Pooling

- Average Pooling computes the average value of a region in a Feature Map



Stride

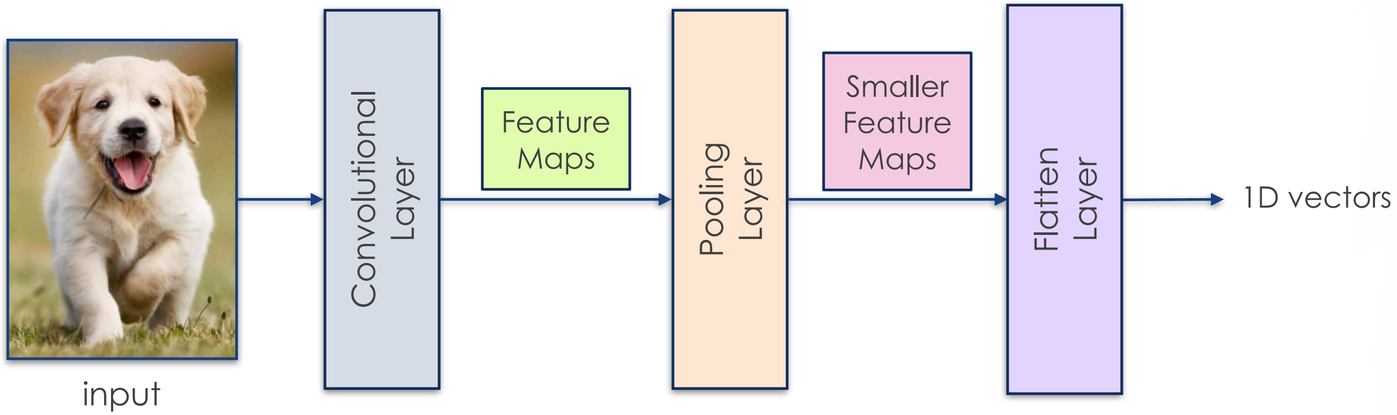
- Stride is the number of steps to shift over a feature map during convolutional or pooling operations
- A larger stride reduces dimensions better but at a risk of information loss

1	4	2	7
2	6	8	5
3	4	0	7
1	2	3	1

Kernel size of 2x2 with
a stride of 1

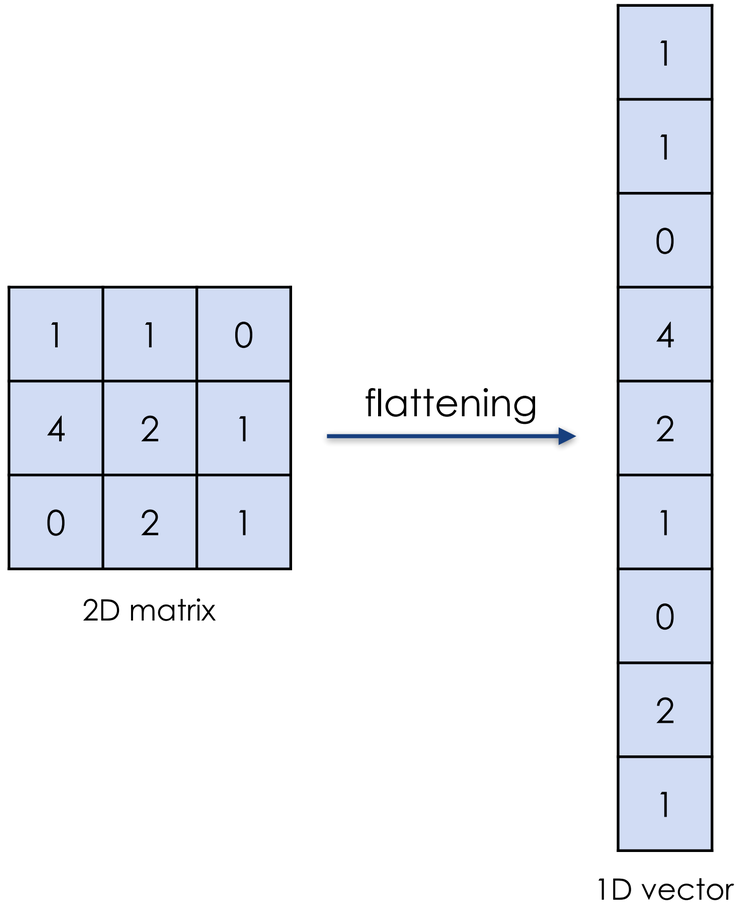
Flatten Layer

- The Flatten Layer flattens 2D feature maps into 1D vectors



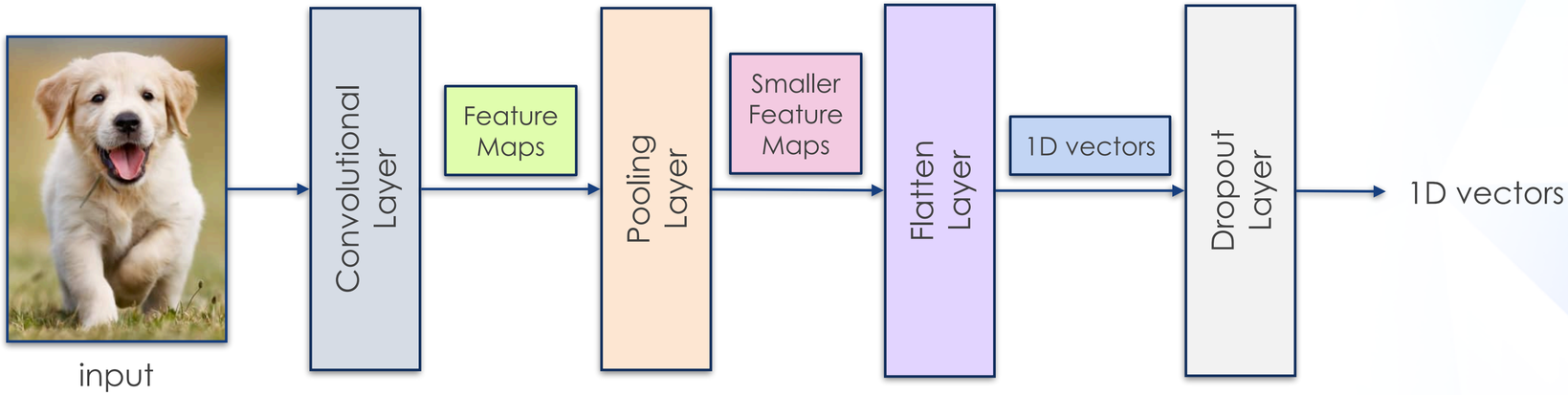
Flatten Operation

- The Flatten operation transforms a 2D matrix into a 1D vector



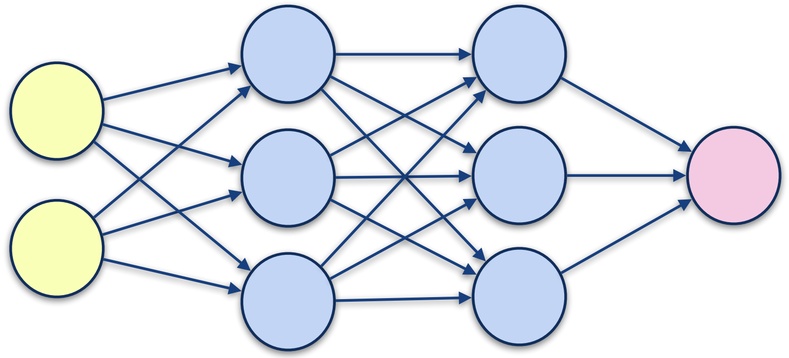
Dropout Layer

- A Dropout Layer drops a fraction of neurons from the computation at random to regularize the network



Dropout Layer

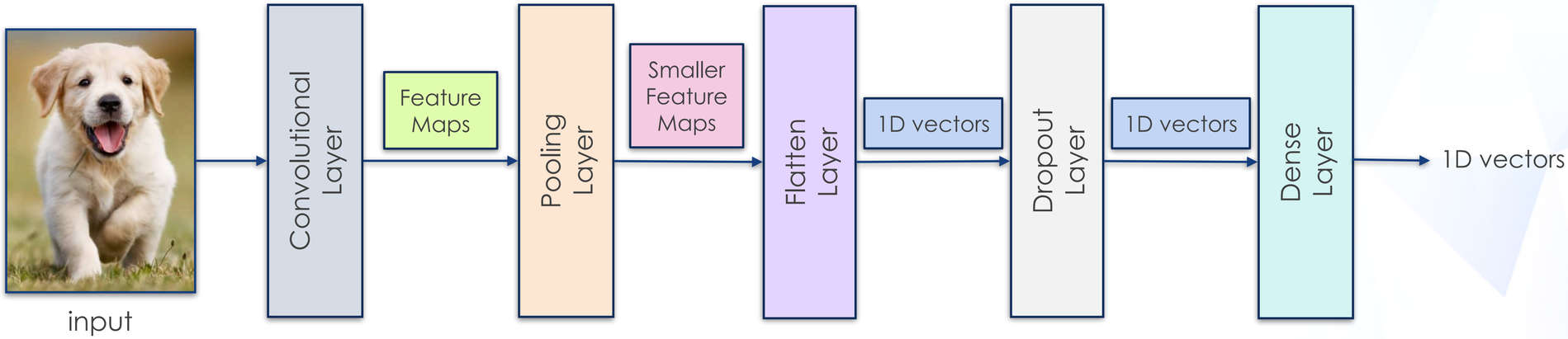
- Dropout prevents a neural network from over-fitting due to a deterministic data path from and to each neuron
- Dropout randomly deactivates some neurons at each iteration in the training phase



input layer hidden layers output layer

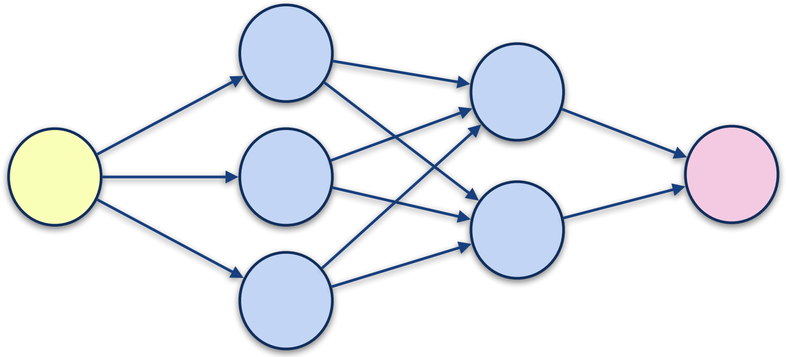
Dense Layer

- A Dense Layer has all its neurons connected to all neurons in the previous layer



Dense Layer

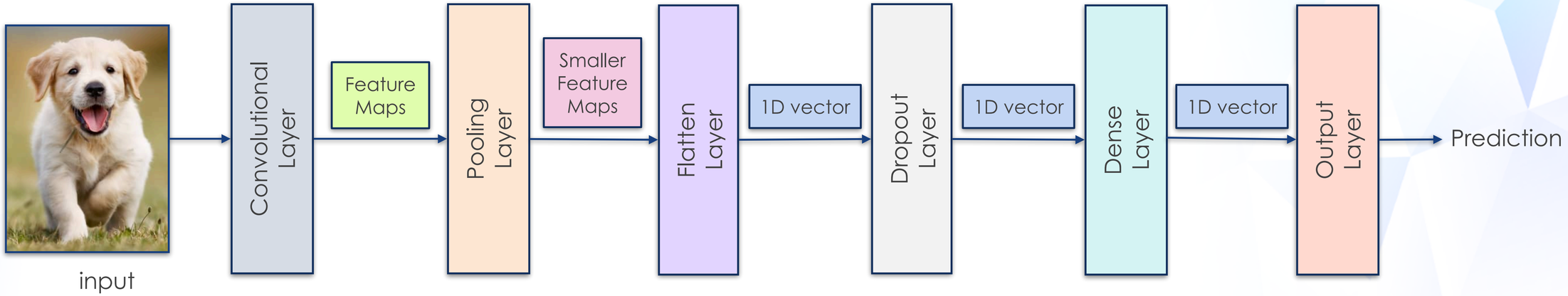
- The two hidden layers are Dense layers as their neurons connect to all neurons in the previous layer
- In CNN, a Dense layer is often placed before the output layer as a way to aggregate all computed values



input layer hidden layers output layer

Output Layer

- An Output Layer outputs a prediction regarding the input

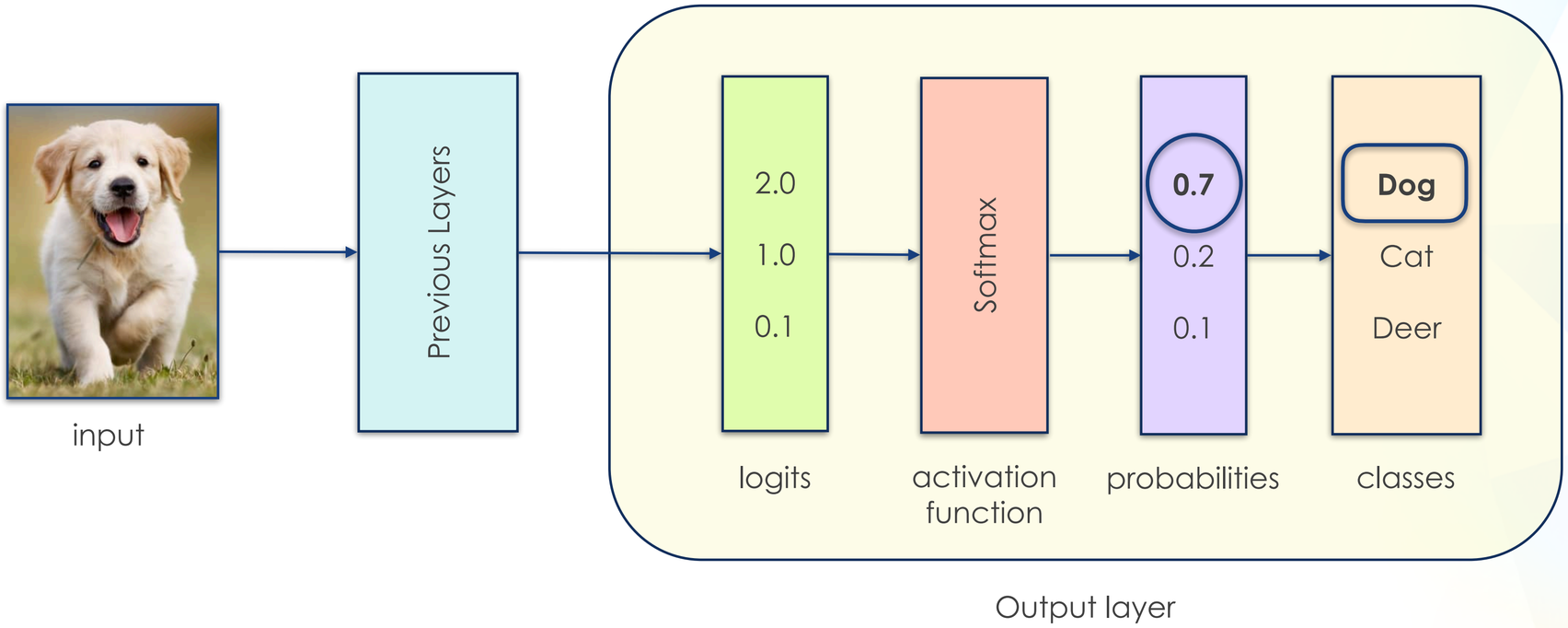


Softmax

- The Softmax activation function takes the computed vector by the previous layers and outputs a vector of probabilities
- The vector of probabilities adds up to 1.0
- Softmax is very often used for multi-class classification problems, where the probabilities denotes the class-predictions of the Neural Network

Softmax

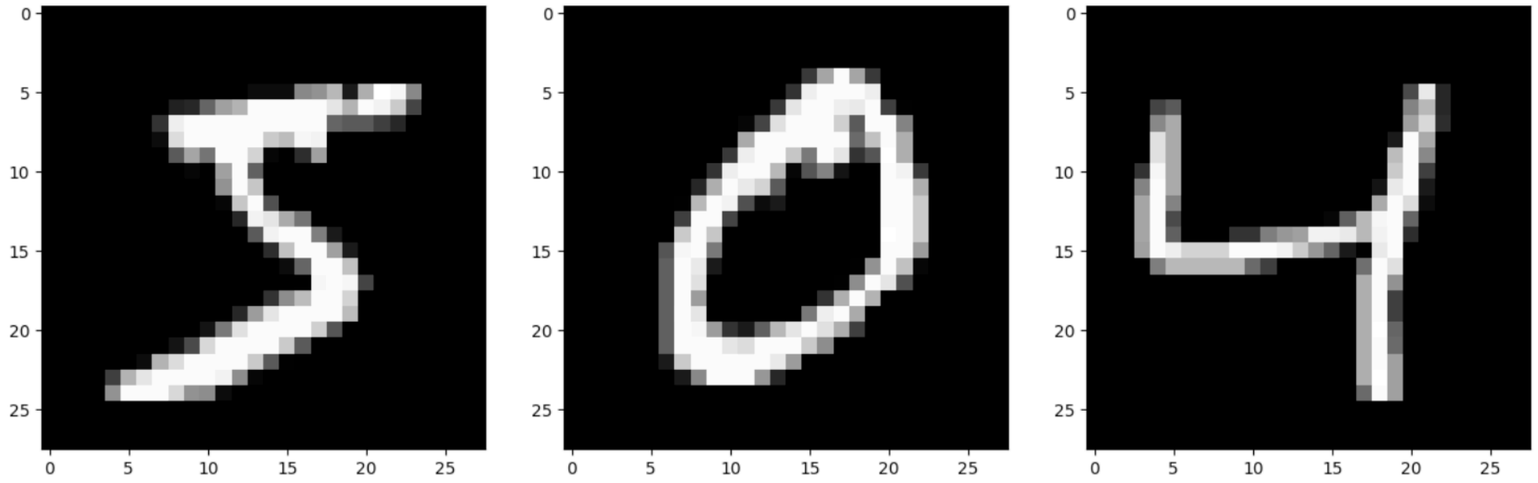
- For classification problems, Softmax is often the final activation function in the Output layer and returns a vector of probabilities that sums to 1.0



CNN example - MNIST

CNN example - MNIST

- MNIST is a dataset of 60,000 hand-written digits of 0 to 9
- Each grayscale digit is an image of 28 x 28 pixels



CNN example - MNIST

- MNIST training data is made up of 60000 rows of 28x28 pixels and has the shape (60000, 28, 28)
- Likewise, MNIST test data has 10000 rows of 28x28 pixels with shape (10000, 28, 28)
- Keras require us to reshape our data to explicitly include the color channel. The color channel is 1 for grayscale images
- Hence, we need to reshape our data to (<rows>, 28, 28, 1)

```
# establish the depth of our images; here we have
# grayscale images, so the depth is 1. for color
# images, the depth is 3.
x_train = np.reshape(_x_train, (_x_train.shape[0], 28, 28, 1))
x_test = np.reshape(_x_test, (_x_test.shape[0], 28, 28, 1))
```

CNN example - MNIST

- Let's create a CNN to train it to recognize the digits in MNIST
- First, we load the training and test samples

```
import numpy as np
import tensorflow as tf

print("Loading MNIST data into memory...")
(_x_train, _y_train), (_x_test, _y_test) = \
    tf.keras.datasets.mnist.load_data()
```

CNN example - MNIST

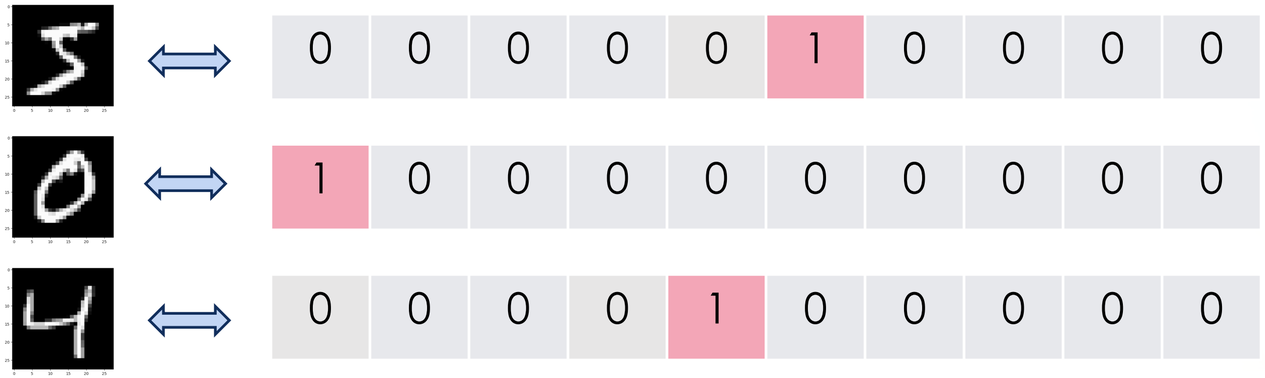
- Grayscale images contains a value of 0 to 255 for each pixel value
- A common practice is to normalize all pixels to 0 and 1
- As NumPy works with vectors of values, so the code below actually divides every single value in ‘x_train’ and ‘x_test’ with 255

```
# normalize values to between 0 and 1
x_train = x_train / 255
x_test = x_test / 255
```

CNN example - MNIST

- The original '_y_train' labels looks like this = [5, 0, 4, …] to denote the ground-truth for each 28x28 pixels of training data
- With Softmax as the activation layer to give us a prediction for each digit 0 to 9, we must transform our labels to be one-hot encoded

```
# convert 1 dimensional array to 10-dimensional array
# each row in y_train and y_test is one-hot encoded
y_train = tf.keras.utils.to_categorical(_y_train, 10)
y_test = tf.keras.utils.to_categorical(_y_test, 10)
```



CNN example - MNIST

- To construct a CNN in code, we first create an empty Neural Network

```
model = tf.keras.Sequential()
```

- We will then add more layers to the ‘model’ object as needed

CNN example - MNIST

- For a CNN model, the first layer should be a Convolutional layer
- The ‘filters’ are the number of kernels that we want to run over our pixels
- The ’kernel_size’ specifies the size of our kernel
- The activation function to used can be specified
- The input shape, for one row of sample, must be specified for the first layer (subsequent layers are not required)

```
# add convolutional layers
model.add(tf.keras.layers.Conv2D(filters=32,
                                 kernel_size=(3, 3),
                                 activation='relu',
                                 input_shape=(28, 28, 1)))

model.add(tf.keras.layers.Conv2D(filters=32,
                                 kernel_size=(3, 3),
                                 activation='relu'))
```

CNN example - MNIST

- Other layers such as Max Pooling, Dropout and Dense are added to the model as required
- Before passing 2D data to a Dense layer, a Flatten layer is required to flatten the data to 1D

```
model.add(tf.keras.layers.Conv2D(32, (3, 3),
    activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
```

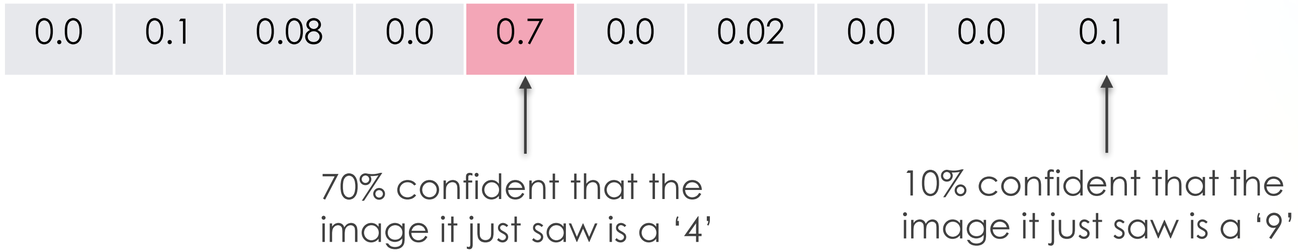
CNN example - MNIST

- To generate a probability distribution, of predictive confidence, across all our classes/labels (i.e. digit 0 to 9), the last layer should be a Dense layer with Softmax as its activation function

```
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

- For our case, the Dense layer should contain 10 units (or neurons), one unit for each digit (i.e. class/label) that our network can assign a probability value to

- The output vector would look like this



CNN example - MNIST

- When using Softmax as the activation function as the last layer, our loss function should be ‘categorical_crossentropy’
- Optimizers are algorithms to change the weights and the learning rate of our neural network; ‘adam’ is a good choice
- When determining the accuracy of a prediction, use ‘accuracy’ as the metrics as we can count the number of times the neural network has given a correct prediction

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

CNN example - MNIST

- To train our CNN model, we call fit(…) with the training dataset

```
# train CNN model
model.fit(x=x_train, y=y_train, batch_size=64, epochs=10)
```

- Batch Size determines the number of samples (or rows) to use for each forward pass
  - The larger the batch size, the more memory is needed
  - A network trains faster with larger batch sizes

- An Epoch means that our neural network has seen the full training data once

CNN example - MNIST

- Finally, we test the accuracy of our CNN model using evaluate(…)
- Note the accuracy value was provided because we specified metrics=[‘accuracy’] when compiling our model

```
# test CNN model
loss, accuracy = model.evaluate(x=x_test, y=y_test)
print("loss =", loss, ', accuracy = ', accuracy)
```

The End