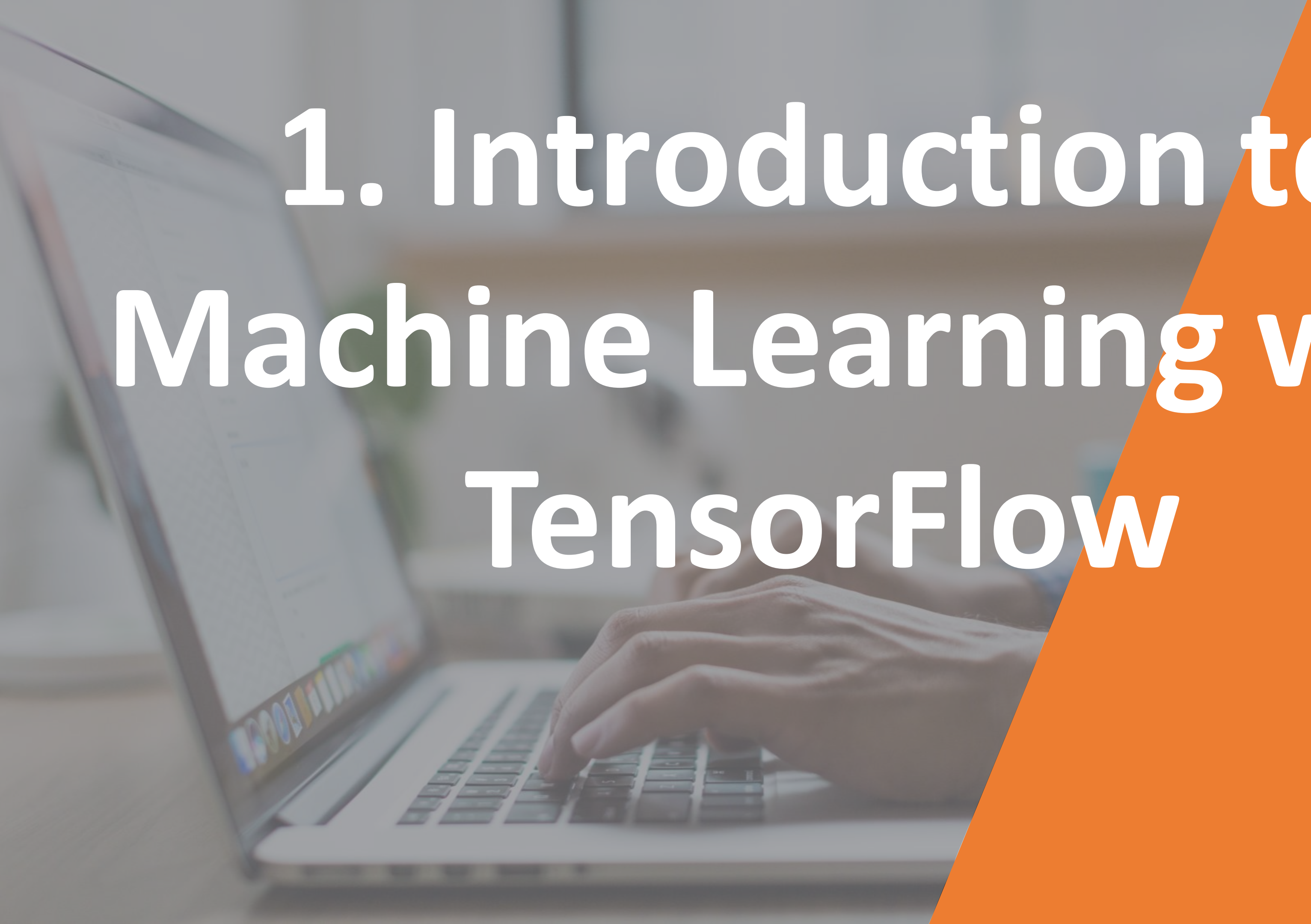
Working with TensorFlow

TriveraTech
TECHNOLOGY TRAINING

Table of Contents



1. Introduction to Machine Learning with TensorFlow

Introduction

- Machine learning (ML) has permeated various aspects of daily life that are unknown to many.
- From the recommendations of your daily social feeds to the results of your online searches, they are all powered by machine learning algorithms.
- These algorithms began in research environments solving niche problems, but as their accessibility broadened, so too have their applications for broader use cases.

Implementing Artificial Neural Networks in TensorFlow

- The advanced flexibility that TensorFlow offers lends itself well to creating artificial neural networks (ANNs).
- ANNs are algorithms that are inspired by the connectivity of neurons in the brain and are intended to replicate the process in which humans learn.
- They consist of layers through which information propagates from the input to the output.

Implementing Artificial Neural Networks in TensorFlow

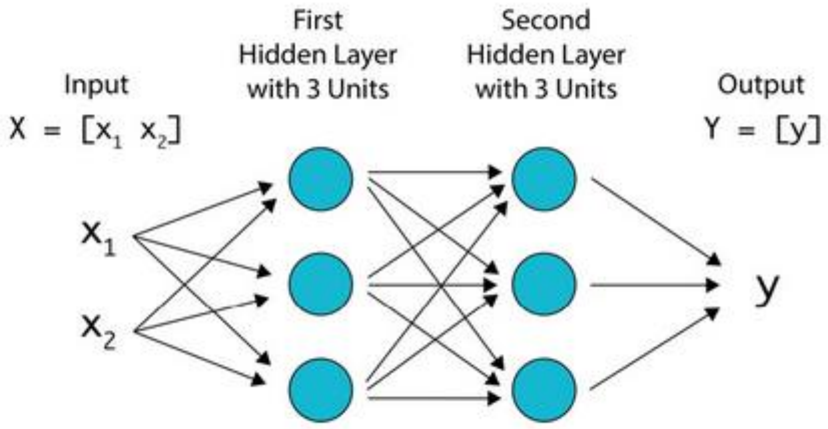


Figure 1.1: A visual representation of an ANN with two hidden layers

Advantages of TensorFlow

The following are a few of the main advantages of using TensorFlow that many practitioners consider when deciding whether to pursue the library for machine learning purposes:

- Library Management
- Pipelining
- Community Support
- Open Source
- Works with Multiple Languages

Disadvantages of TensorFlow

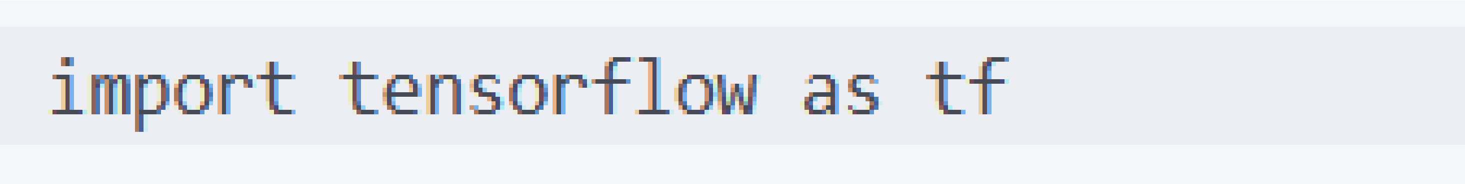
The following are a few of the disadvantages of using TensorFlow:

- Computational Speed
- Steep Learning Curve

The TensorFlow Library in Python

- TensorFlow can be used in Python by importing certain libraries.

- You can import libraries in Python using the import statement:



```
import tensorflow as tf
```

"Complete Exercise"

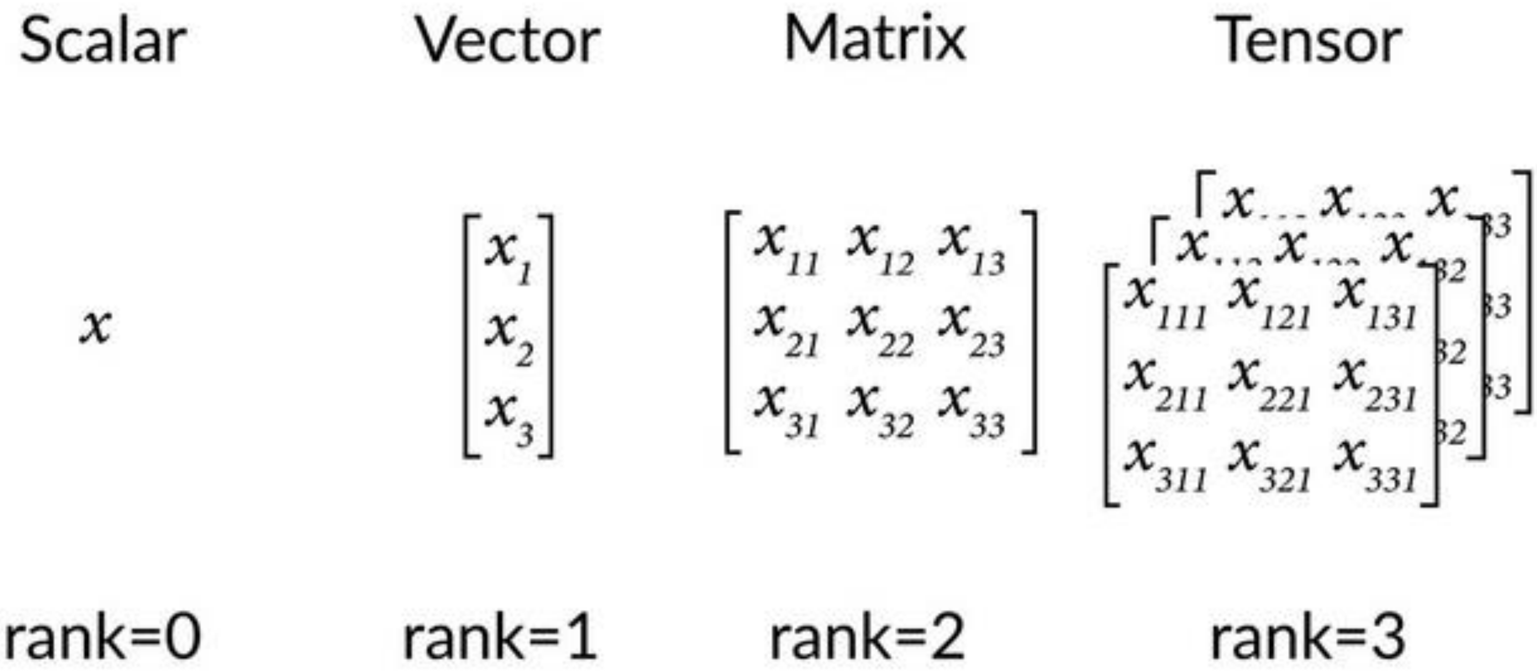
“Verifying Your Version of TensorFlow”

Introduction to Tensors

- Tensors can be thought of as the core components of ANNs—the input data, output predictions, and weights that are learned throughout the training process are all tensors.

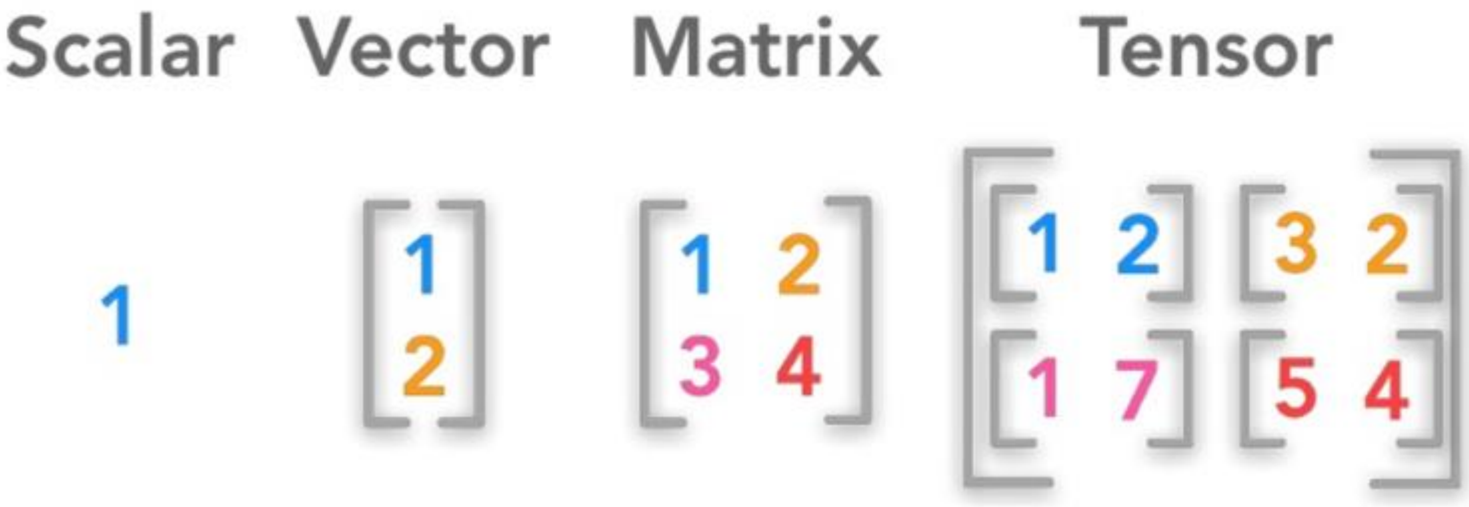
- Information propagates through a series of linear and nonlinear transformations to turn the input data into predictions.

Scalars, Vectors, Matrices, & Tensors



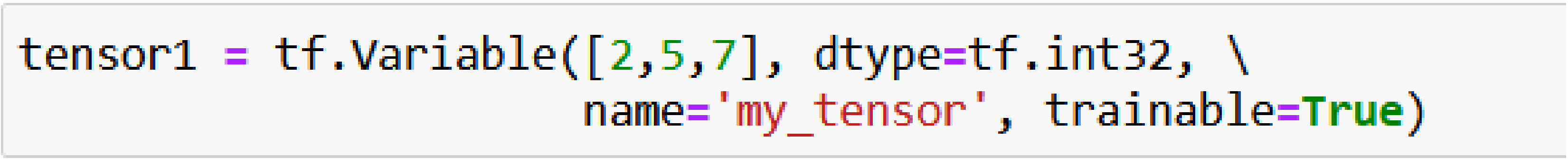
Scalar	Vector	Matrix	Tensor
x	$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$	$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$	$\begin{bmatrix} x_{111} & x_{121} & x_{131} \\ x_{211} & x_{221} & x_{231} \\ x_{311} & x_{321} & x_{331} \end{bmatrix}$
rank=0	rank=1	rank=2	rank=3

Scalars, Vectors, Matrices, & Tensors



Scalars, Vectors, Matrices, & Tensors

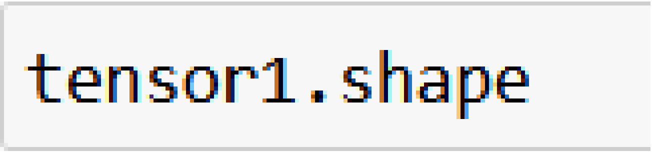
- he following command demonstrates the use of the Variable class where a list of the intended values for the tensor as well as any other attributes that are required to be explicitly defined are passed:



```
tensor1 = tf.Variable([2,5,7], dtype=tf.int32, \
                      name='my_tensor', trainable=True)
```

Scalars, Vectors, Matrices, & Tensors

- The shape attribute of the Variable object can be called as follows:

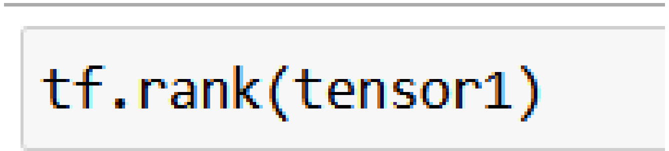


```
tensor1.shape
```

Scalars, Vectors, Matrices, & Tensors

- The rank of a tensor can be determined in TensorFlow using the rank function.

- It can be used by passing the tensor as the single argument to the function and the result will be an integer value:



```
tf.rank(tensor1)
```

"Complete Exercise"

“Creating Scalars, Vectors, Matrices, and Tensors in TensorFlow”

Tensor Addition

- Tensors can be added together to create new tensors. You will use the example of matrices in this lesson, but the concept can be extended to tensors with any rank.

- Matrices may be added to scalars, vectors, and other matrices under certain conditions in a process known as broadcasting. Broadcasting refers to the process of array arithmetic on tensors of different shapes.

Tensor Addition

- The same matrix addition principles apply to scalars, vectors, and tensors.
- An example is shown in the following figure:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} + \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix} = \begin{bmatrix} x_{11}+y_{11} & x_{12}+y_{12} & x_{13}+y_{13} \\ x_{21}+y_{21} & x_{22}+y_{22} & x_{23}+y_{23} \\ x_{31}+y_{31} & x_{32}+y_{32} & x_{33}+y_{33} \end{bmatrix}$$

A visual example of matrix-matrix addition

Tensor Addition

- Scalars can also be added to matrices. Here, each element of the matrix is added to the scalar individually, as shown in Figure.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} + \quad y \quad = \begin{bmatrix} x_{11}+y & x_{12}+y & x_{13}+y \\ x_{21}+y & x_{22}+y & x_{23}+y \\ x_{31}+y & x_{32}+y & x_{33}+y \end{bmatrix}$$

"Complete Exercise"

“Performing Tensor Addition in TensorFlow”

"Complete Activity "

“Performing Tensor Addition in TensorFlow”

Reshaping

- Some operations, such as addition, can only be applied to tensors if they meet certain conditions.

- Reshaping is one method for modifying the shape of tensors so that such operations can be performed.

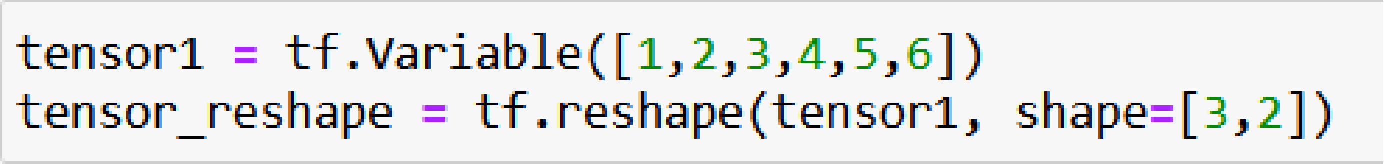
- Reshaping takes the elements of a tensor and rearranges them into a tensor of a different size.

Reshaping

$$\begin{tikzcd}
 & \begin{bmatrix} x_{11} & x_{12} & x_{21} \\ x_{22} & x_{32} & x_{32} \end{bmatrix} \\
\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} \arrow[ru] \arrow[r] \arrow[rd] & \begin{bmatrix} x_{11} & x_{12} & x_{21} & x_{22} & x_{32} & x_{32} \end{bmatrix} \\
 & \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ x_{12} \\ x_{22} \\ x_{32} \end{bmatrix}
\end{tikzcd}$$

Reshaping

- Tensor reshaping can be performed in TensorFlow by using the reshape function and passing in the tensor and the desired shape of the new tensor as the arguments:



```
tensor1 = tf.Variable([1,2,3,4,5,6])
tensor_reshape = tf.reshape(tensor1, shape=[3,2])
```

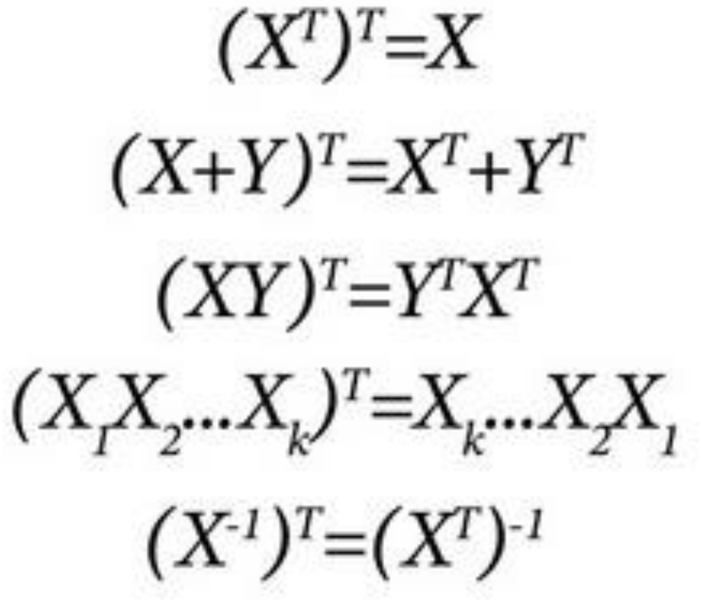
Tensor Transposition

- An example of matrix transposition is shown in Figure.
- Tensors of any rank can be transposed, and often the shape changes as a result:

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix}^T = \begin{bmatrix} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \end{bmatrix}$$

Tensor Transposition

- The following diagram shows the matrix transposition properties of matrices A and B:



$$(X^T)^T = X$$

$$(X+Y)^T = X^T + Y^T$$

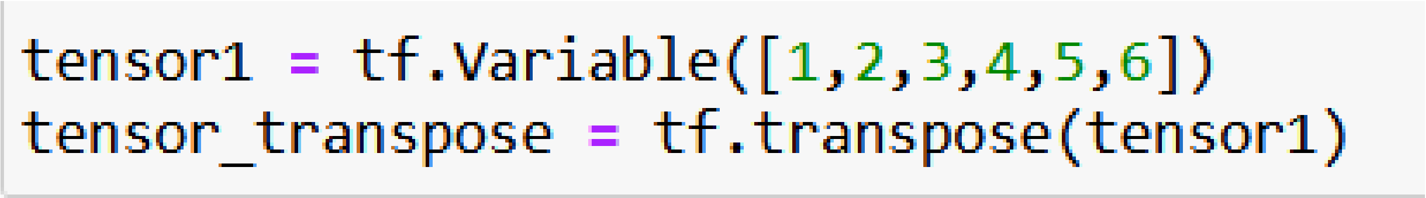
$$(XY)^T = Y^T X^T$$

$$(X_1 X_2 \ldots X_k)^T = X_k \ldots X_2 X_1$$

$$(X^{-1})^T = (X^T)^{-1}$$

Tensor Transposition

- Tensor transposition can be performed in TensorFlow by using its transpose function and passing in the tensor as the only argument:



```
tensor1 = tf.Variable([1,2,3,4,5,6])
tensor_transpose = tf.transpose(tensor1)
```

"Complete Exercise"

“Performing Tensor Reshaping and Transposition in TensorFlow”

"Complete Activity "

"Performing Tensor Reshaping and Transposition in TensorFlow"

Tensor Multiplication

- Tensor multiplication is another fundamental operation that is used frequently in the process of building and training ANNs since information propagates through the network from the inputs to the result via a series of additions and multiplications.

- While the rules for addition are simple and intuitive, the rules for tensors are more complex.

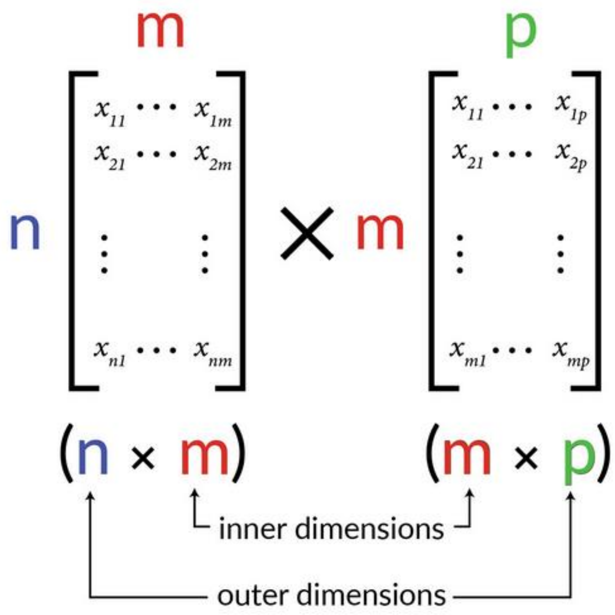
Tensor Multiplication

- Given a matrix, X = [xij]m x n, and another matrix, Y = [yij]n x p, the product of the two matrices is Z = XY = [zij]m x p, and each element, zij, is defined element-wise as

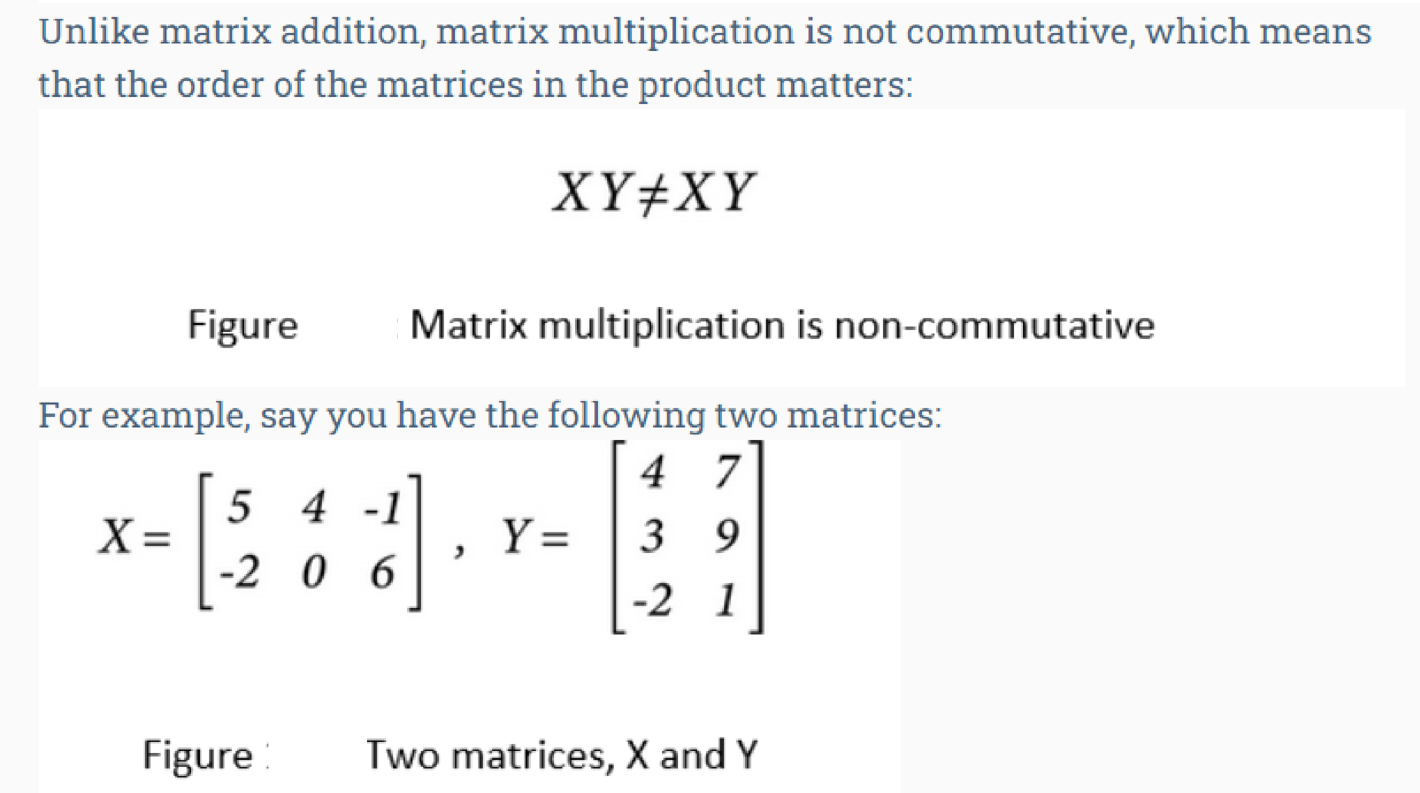
$$Z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj}$$

Tensor Multiplication

- The concept of inner and outer dimensions of matrix multiplication is shown in the following diagram, where X represents the first matrix and Y represents the second matrix:



Tensor Multiplication



Unlike matrix addition, matrix multiplication is not commutative, which means that the order of the matrices in the product matters:

$$XY \neq XY$$

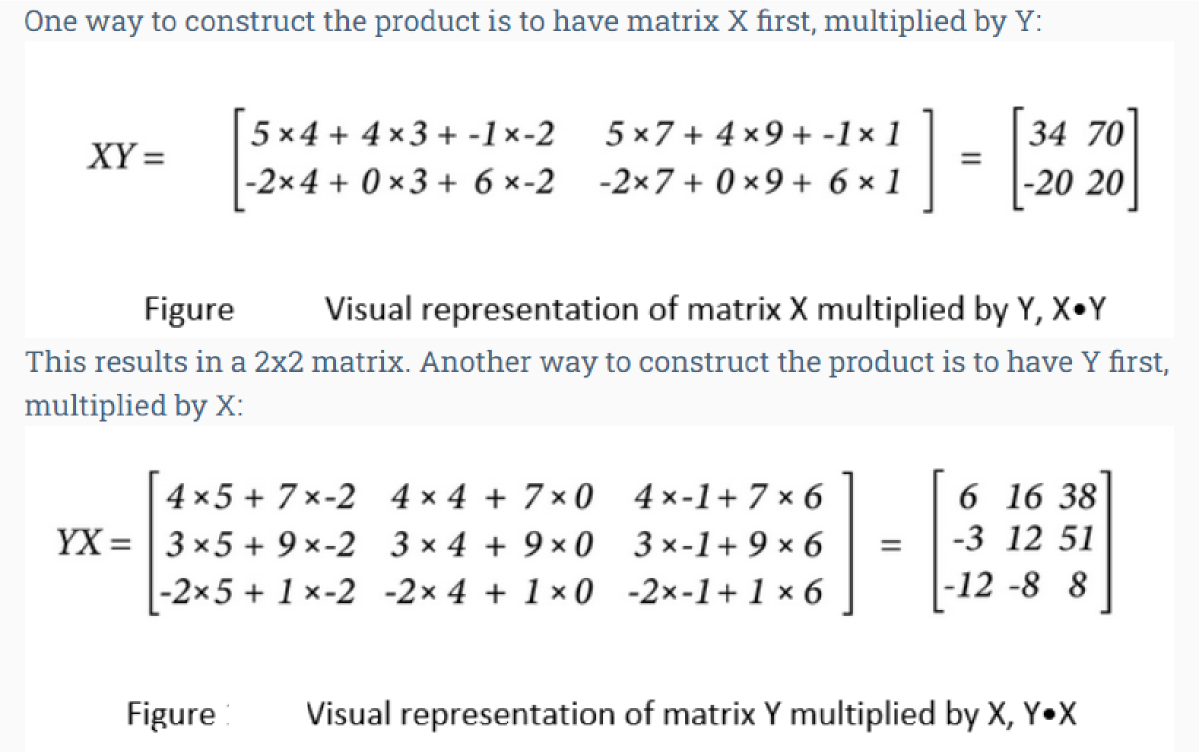
Figure : Matrix multiplication is non-commutative

For example, say you have the following two matrices:

$$X = \begin{bmatrix} 5 & 4 & -1 \\ -2 & 0 & 6 \end{bmatrix}, \quad Y = \begin{bmatrix} 4 & 7 \\ 3 & 9 \\ -2 & 1 \end{bmatrix}$$

Figure : Two matrices, X and Y

Tensor Multiplication



One way to construct the product is to have matrix X first, multiplied by Y:

$$XY = \begin{bmatrix} 5\times4 + 4\times3 + -1\times-2 & 5\times7 + 4\times9 + -1\times1 \\ -2\times4 + 0\times3 + 6\times-2 & -2\times7 + 0\times9 + 6\times1 \end{bmatrix} = \begin{bmatrix} 34 & 70 \\ -20 & 20 \end{bmatrix}$$

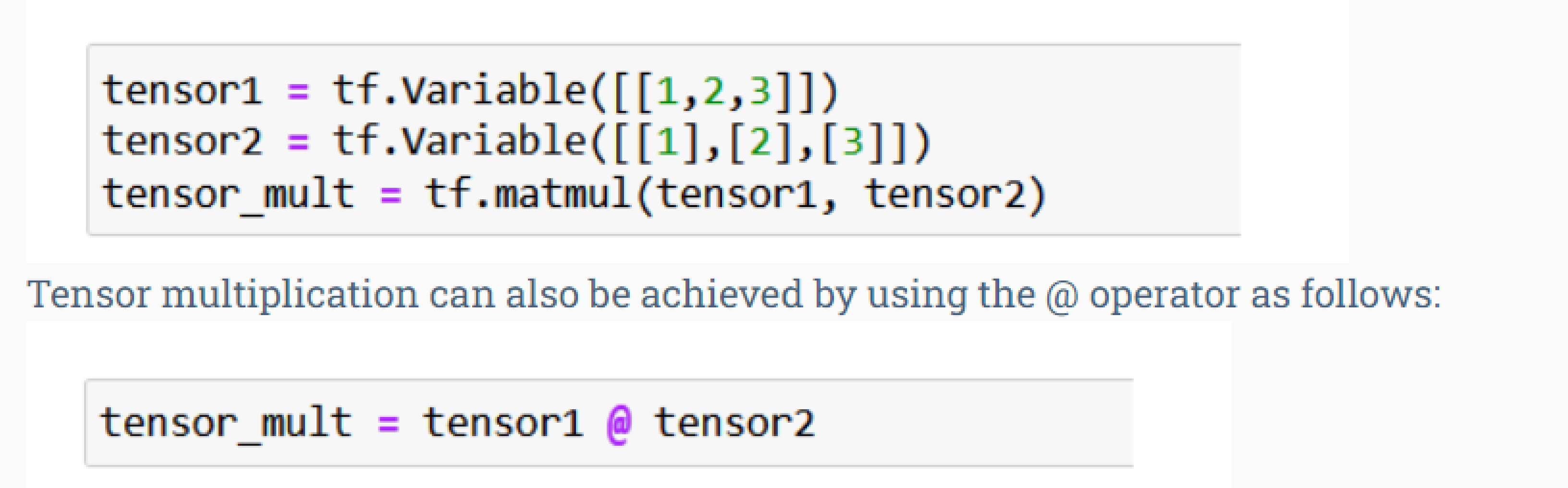
Figure Visual representation of matrix X multiplied by Y, X•Y

This results in a 2x2 matrix. Another way to construct the product is to have Y first, multiplied by X:

$$YX = \begin{bmatrix} 4\times5 + 7\times-2 & 4\times4 + 7\times0 & 4\times-1 + 7\times6 \\ 3\times5 + 9\times-2 & 3\times4 + 9\times0 & 3\times-1 + 9\times6 \\ -2\times5 + 1\times-2 & -2\times4 + 1\times0 & -2\times-1 + 1\times6 \end{bmatrix} = \begin{bmatrix} 6 & 16 & 38 \\ -3 & 12 & 51 \\ -12 & -8 & 8 \end{bmatrix}$$

Figure Visual representation of matrix Y multiplied by X, Y•X

Tensor Multiplication



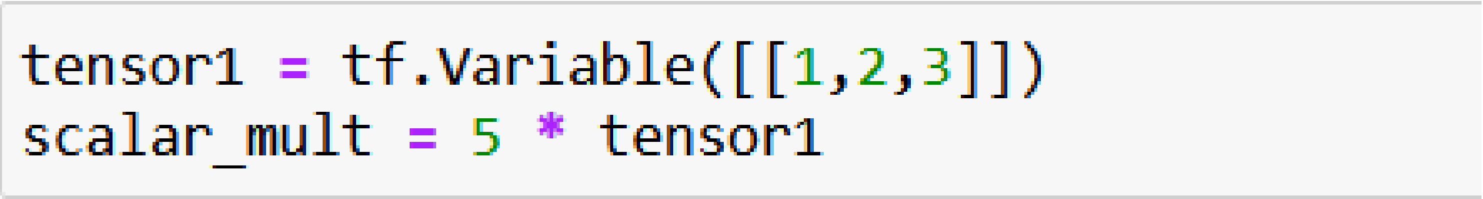
```
tensor1 = tf.Variable([[1,2,3]])
tensor2 = tf.Variable([[1],[2],[3]])
tensor_mult = tf.matmul(tensor1, tensor2)
```

Tensor multiplication can also be achieved by using the @ operator as follows:

```
tensor_mult = tensor1 @ tensor2
```

Tensor Multiplication

- Scalar multiplication can be achieved in TensorFlow either by using the matmul function or by using the * operator:



```
tensor1 = tf.Variable([[1,2,3]])
scalar_mult = 5 * tensor1
```

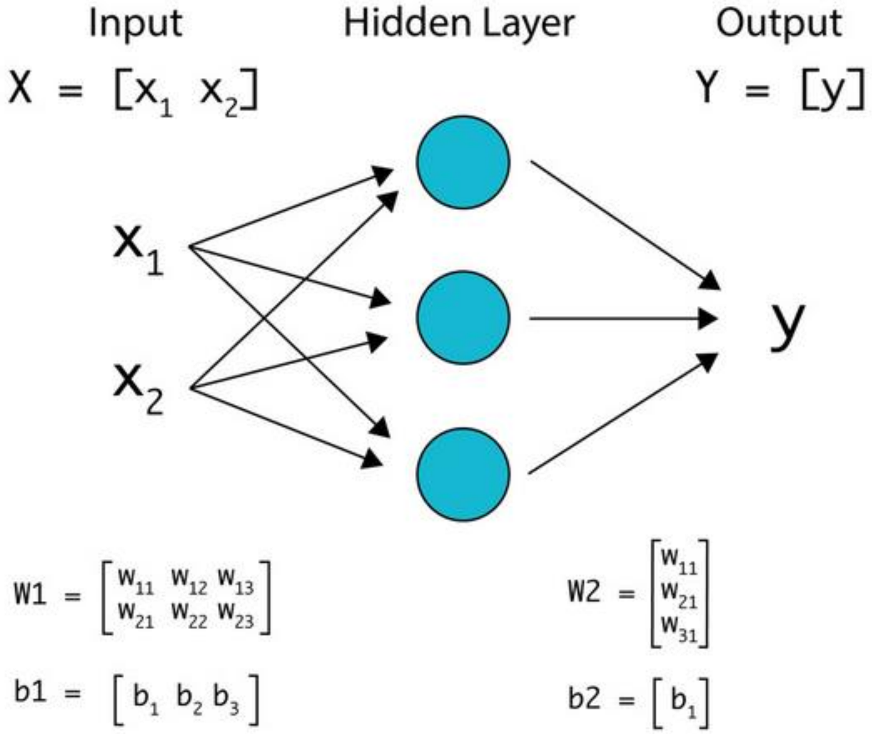
"Complete Exercise"

“Performing Tensor Multiplication in TensorFlow”

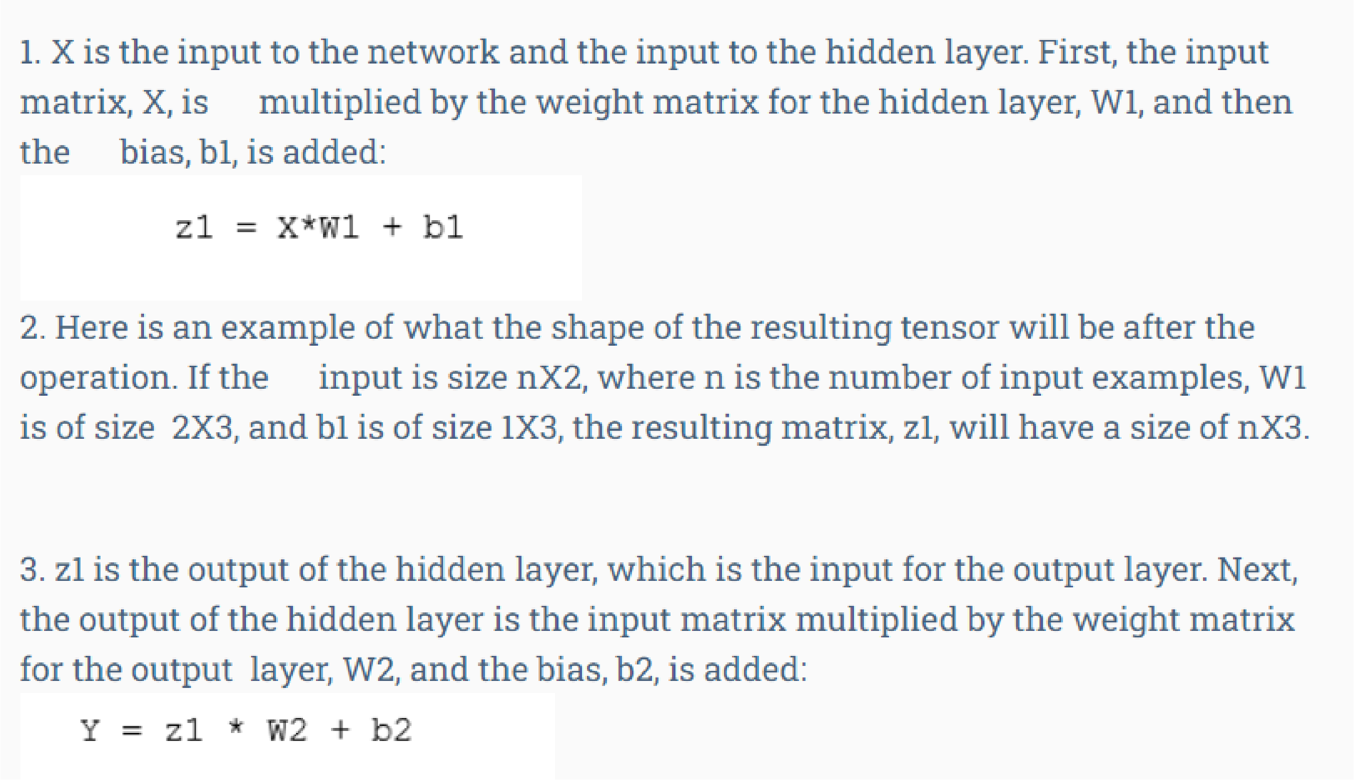
Optimization

- Optimization is the process by which the weights of the layers of an ANN are updated such that the error between the predicted values of the ANN and the true values of the training data is minimized.

Forward Propagation


Input
$X = [x_1 \ x_2]$
Hidden Layer
Output
$Y = [y]$
x_1
x_2
y
$W1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$
$b1 = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$
$W2 = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$
$b2 = \begin{bmatrix} b_1 \end{bmatrix}$

Forward Propagation



1. X is the input to the network and the input to the hidden layer. First, the input matrix, X, is multiplied by the weight matrix for the hidden layer, W1, and then the bias, b1, is added:

```
z1 = X*W1 + b1
```

2. Here is an example of what the shape of the resulting tensor will be after the operation. If the input is size nX2, where n is the number of input examples, W1 is of size 2X3, and b1 is of size 1X3, the resulting matrix, z1, will have a size of nX3.

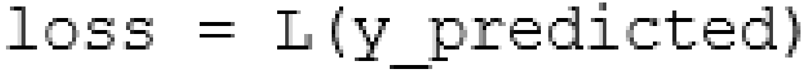
3. z1 is the output of the hidden layer, which is the input for the output layer. Next, the output of the hidden layer is the input matrix multiplied by the weight matrix for the output layer, W2, and the bias, b2, is added:

```
Y = z1 * W2 + b2
```

Backpropagation

- Backpropagation is the process of determining the derivative of the loss with respect to the model parameter.

- The loss is calculated by applying the loss function to the predicted outputs as follows:



```
loss = L(y_predicted)
```

Backpropagation

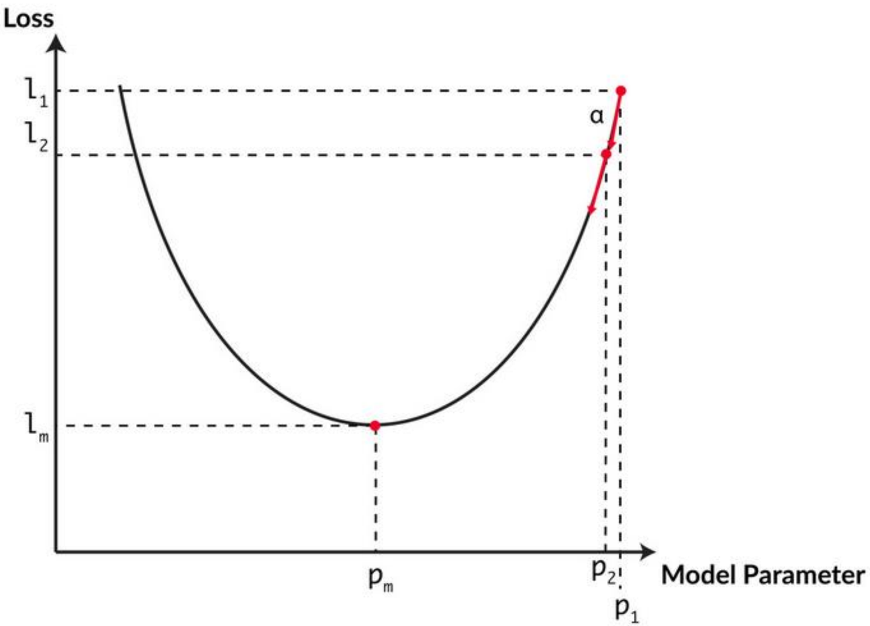
- The chain rule of calculus is a technique used to compute the derivative of a composite function via intermediate functions.

- A generalized version of the function can be written as follows:

```
dz/dx = dz/dy * dy/dx
```

Learning Optimal Parameters

- The following figure shows the loss function of a particular feature:



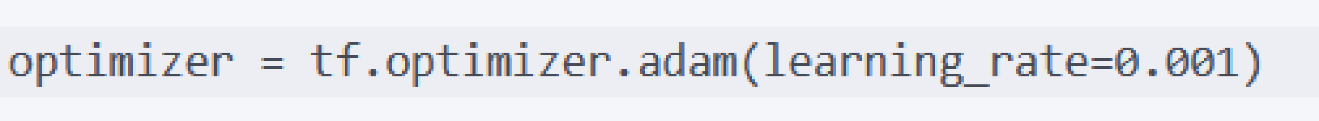
Optimizers in TensorFlow

There are several different optimizers readily available within TensorFlow. Each is based on a different optimization algorithm that aims to reach a global minimum for the loss function. They are all based on the gradient descent algorithm, although they differ slightly in implementation. The available optimizers in TensorFlow include the following:

- Stochastic Gradient Descent (SGD)
- Adam
- Root Mean Squared Propagation (RMSProp)
- Adagrad

Optimizers in TensorFlow

- The optimizers available in TensorFlow are located in the tf.optimizers module; for example, an Adam optimizer with a learning rate equal to 0.001 can be initialized as follows:



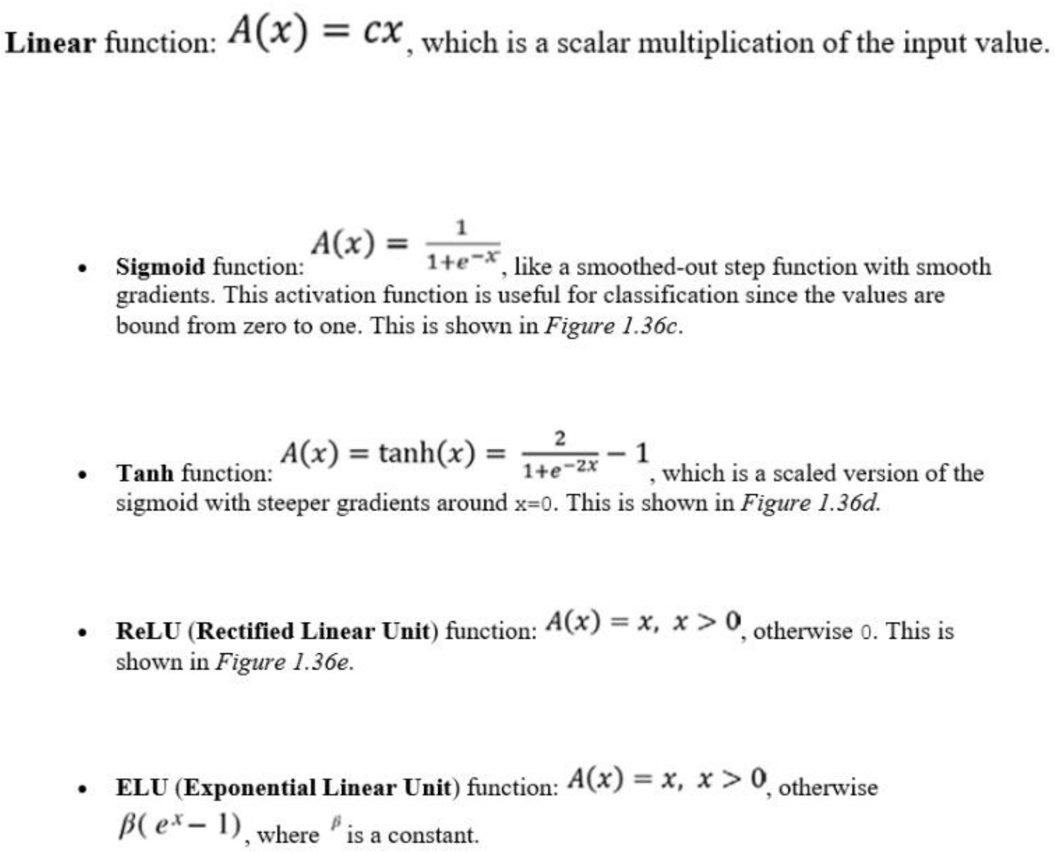
```
optimizer = tf.optimizer.adam(learning_rate=0.001)
```

Activation functions

- Activation functions are mathematical functions that are generally applied to the outputs of ANN layers to limit or bound the values of the layer.

- The reason that values may want to be bounded is that without activation functions, the value and corresponding gradients can either explode or vanish, thereby making the results unusable.

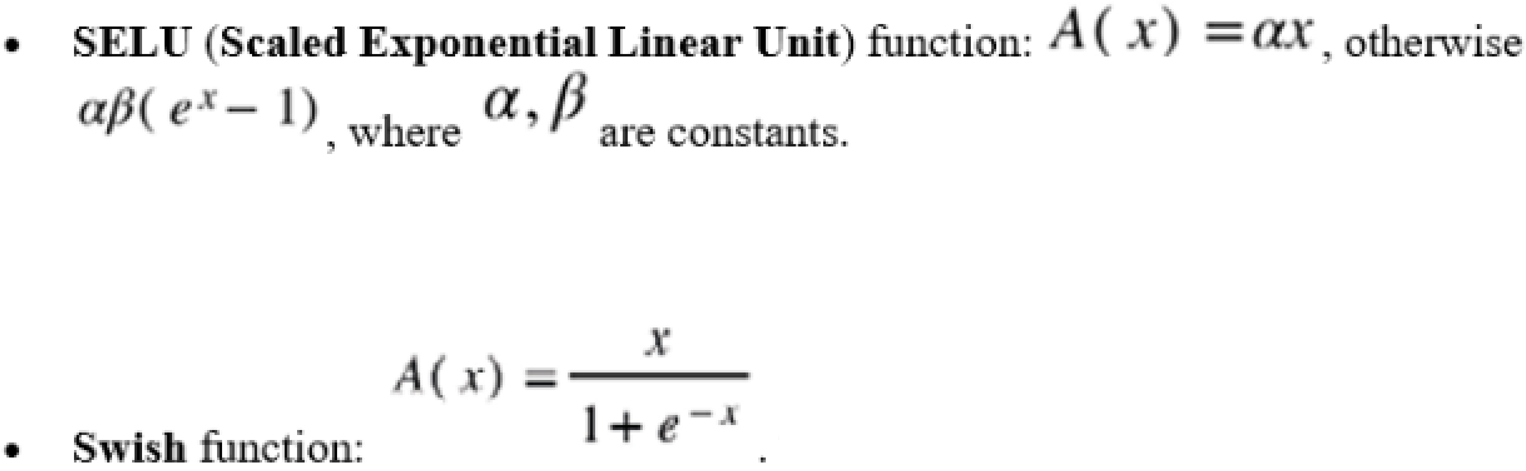
Activation functions



Linear function: $A(x) = cx$, which is a scalar multiplication of the input value.

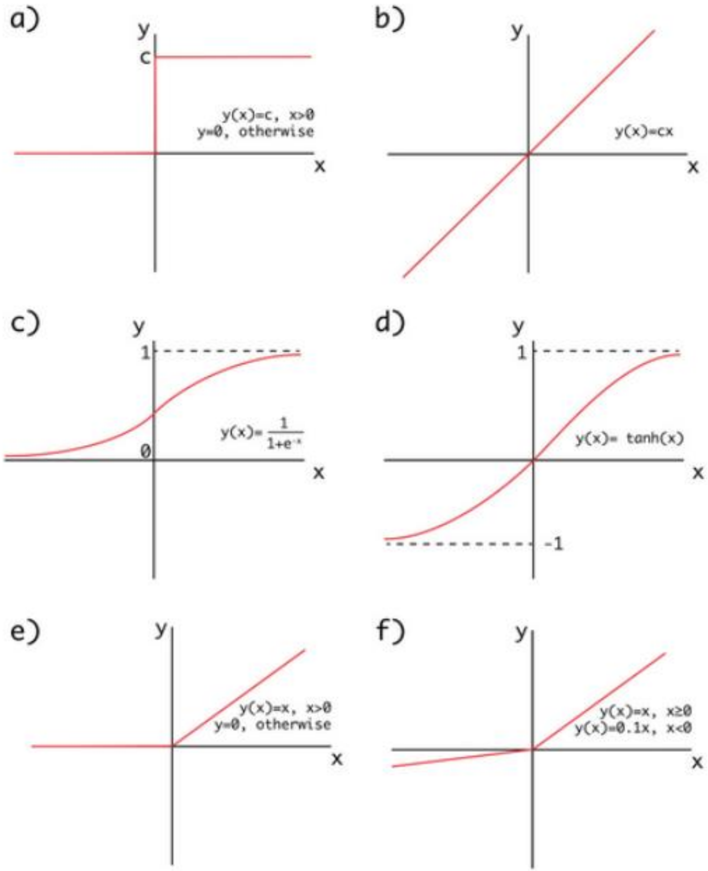
- **Sigmoid** function: $A(x) = \frac{1}{1+e^{-x}}$, like a smoothed-out step function with smooth gradients. This activation function is useful for classification since the values are bound from zero to one. This is shown in *Figure 1.36c*.
- **Tanh** function: $A(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$, which is a scaled version of the sigmoid with steeper gradients around x=0. This is shown in *Figure 1.36d*.
- **ReLU** (**Rectified Linear Unit**) function: $A(x) = x,\ x > 0$, otherwise 0. This is shown in *Figure 1.36e*.
- **ELU** (**Exponential Linear Unit**) function: $A(x) = x,\ x > 0$, otherwise $\beta(\ e^{x} - 1)$, where β is a constant.

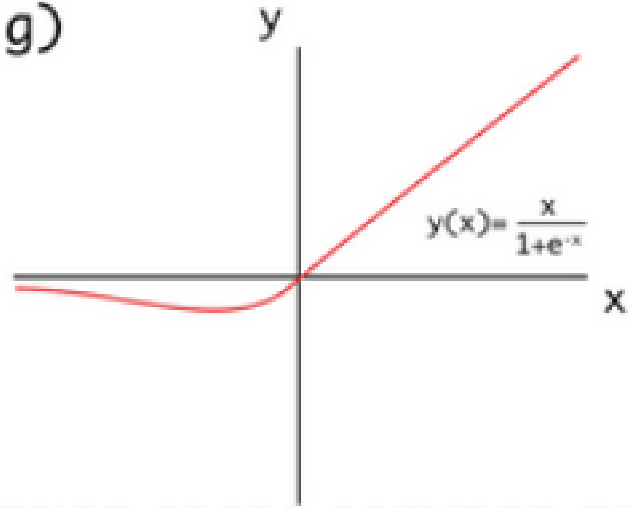
Activation functions



- **SELU (Scaled Exponential Linear Unit)** function: $A(x) = \alpha x$, otherwise $\alpha\beta(e^{x} - 1)$, where α, β are constants.

- **Swish** function: $$A(x) = \frac{x}{1 + e^{-x}}.$$


a)
y
c
y(x)=c, x>0
y=0, otherwise
x
b)
y
y(x)=cx
x
c)
y
1
0
$y(x)=\frac{1}{1+e^{-x}}$
x
d)
y
1
y(x)= tanh(x)
x
-1
e)
y
y(x)=x, x>0
y=0, otherwise
x
f)
y
y(x)=x, x≥0
y(x)=0.1x, x<0
x


g)
y
$y(x)=\frac{x}{1+e^{-x}}$
x

"Complete Activity"

“Verifying Your Version of TensorFlow”

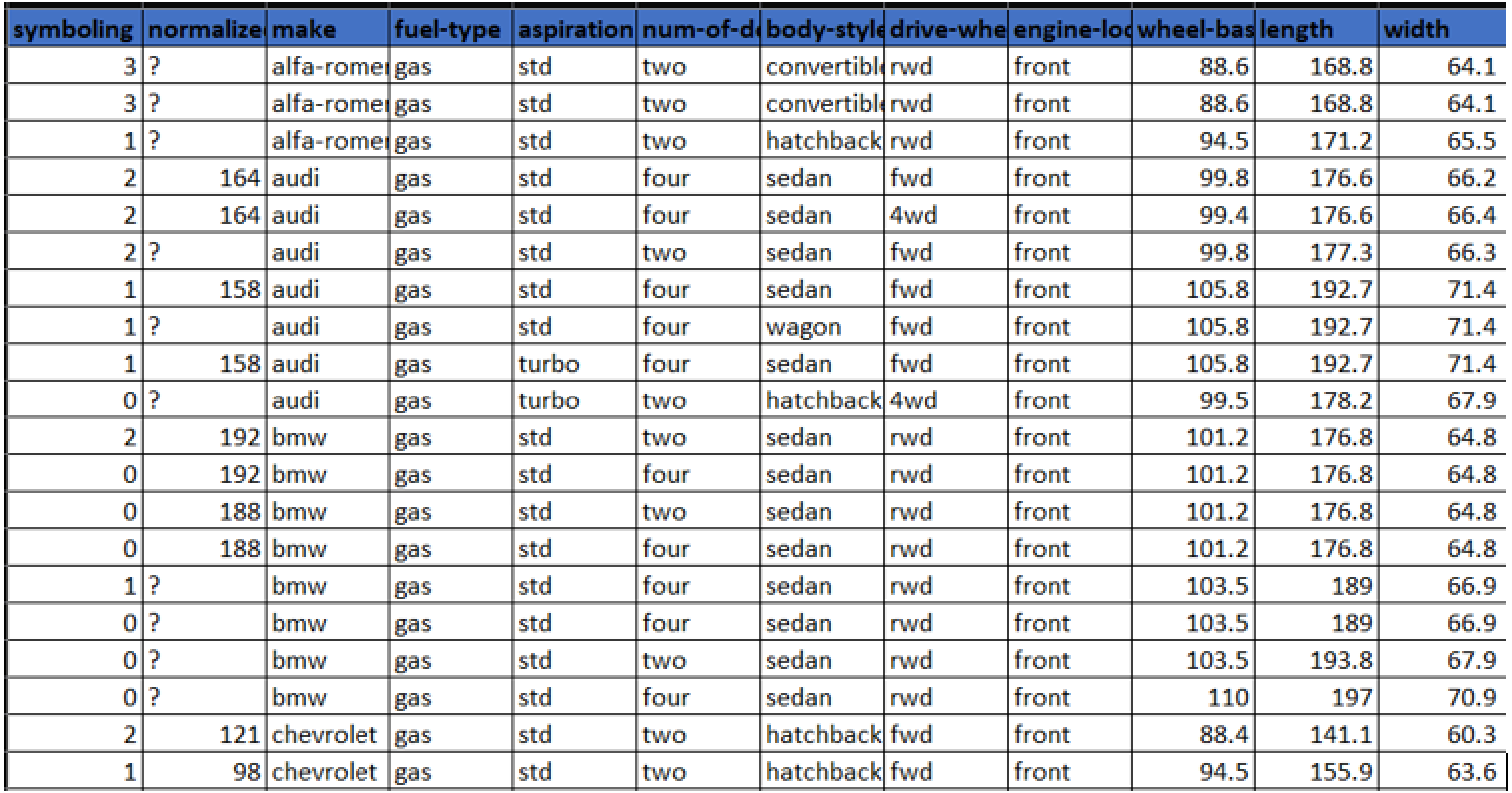
Summary

- In this lesson, you were introduced to the TensorFlow library.
- You learned how to use it in the Python programming language.
- You created the building blocks of ANNs (tensors) with various ranks and shapes, performed linear transformations on tensors using TensorFlow, and implemented addition, reshaping, transposition, and multiplication on tensors—all of which are fundamental for understanding the underlying mathematics of ANNs.

2. Loading and Processing Data

Introduction

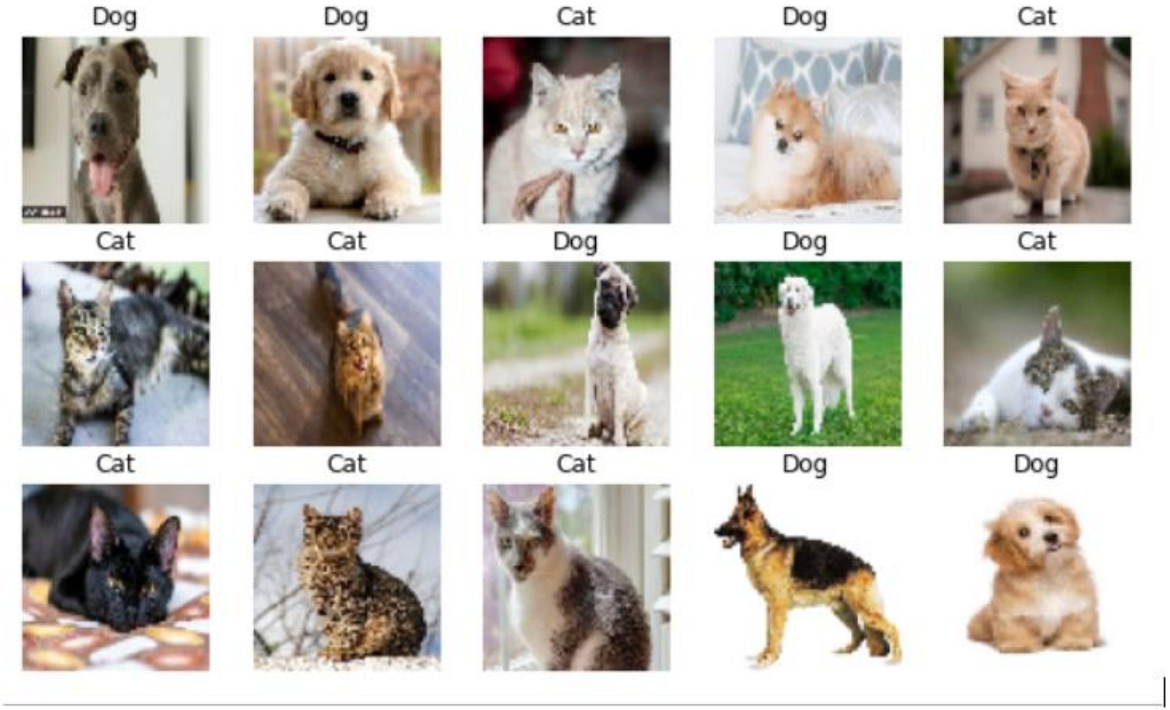
- In the previous lesson, you learned how to create, utilize, and apply linear transformations to tensors using TensorFlow.
- The lesson started with the definition of tensors and how they can be created using the Variable class in the TensorFlow library.
- You then created tensors of various ranks and learned how to apply tensor addition, reshaping, transposition, and multiplication using the library.



symboling	normalize	make	fuel-type	aspiration	num-of-d	body-styl	drive-whe	engine-lo	wheel-bas	length	width
3	?	alfa-rome	gas	std	two	convertibl	rwd	front	88.6	168.8	64.1
3	?	alfa-rome	gas	std	two	convertibl	rwd	front	88.6	168.8	64.1
1	?	alfa-rome	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5
2	164	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2
2	164	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4
2	?	audi	gas	std	two	sedan	fwd	front	99.8	177.3	66.3
1	158	audi	gas	std	four	sedan	fwd	front	105.8	192.7	71.4
1	?	audi	gas	std	four	wagon	fwd	front	105.8	192.7	71.4
1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	192.7	71.4
0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	178.2	67.9
2	192	bmw	gas	std	two	sedan	rwd	front	101.2	176.8	64.8
0	192	bmw	gas	std	four	sedan	rwd	front	101.2	176.8	64.8
0	188	bmw	gas	std	two	sedan	rwd	front	101.2	176.8	64.8
0	188	bmw	gas	std	four	sedan	rwd	front	101.2	176.8	64.8
1	?	bmw	gas	std	four	sedan	rwd	front	103.5	189	66.9
0	?	bmw	gas	std	four	sedan	rwd	front	103.5	189	66.9
0	?	bmw	gas	std	two	sedan	rwd	front	103.5	193.8	67.9
0	?	bmw	gas	std	four	sedan	rwd	front	110	197	70.9
2	121	chevrolet	gas	std	two	hatchback	fwd	front	88.4	141.1	60.3
1	98	chevrolet	gas	std	two	hatchback	fwd	front	94.5	155.9	63.6

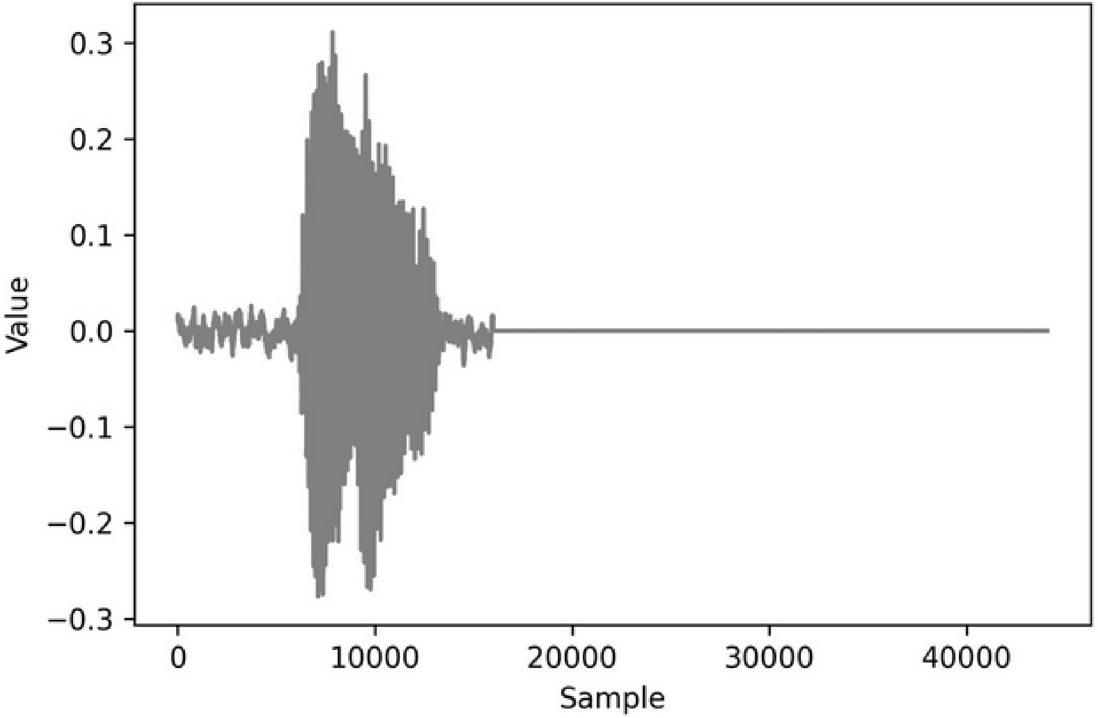
Figure : An example of 10 rows of tabular data that consists of numerical values

Exploring Data Types



A sample of image data that can be used for training machine learning models

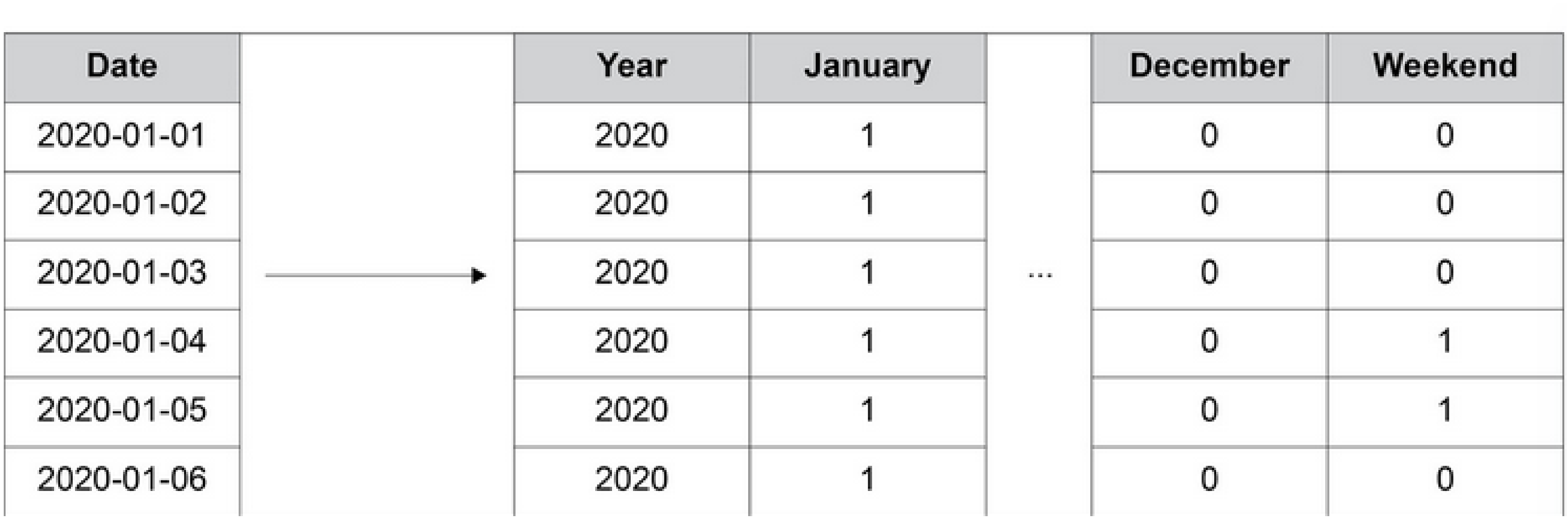
Exploring Data Types



Data Pre-processing

- Data pre-processing refers to the process in which raw data is converted into a form that is appropriate for machine learning models to use as input.
- Each different data type will require different pre-processing steps, with the minimum requirement that the resulting tensor is composed solely of numerical elements, such as integers or decimal numbers.
- Numerical tensors are required since models rely on linear transformations such as addition and multiplication, which can only be performed on numerical tensors.

Data Pre-processing



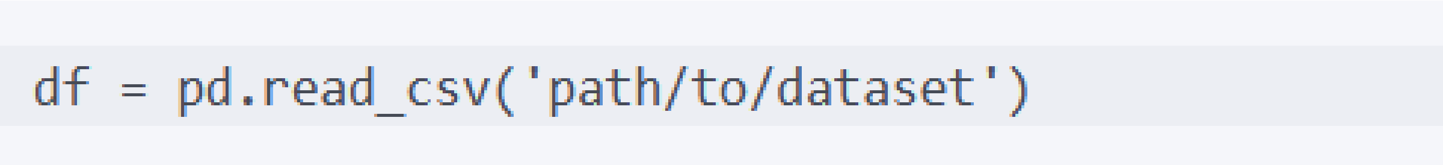
Date
2020-01-01
2020-01-02
2020-01-03
2020-01-04
2020-01-05
2020-01-06

Year	January		December	Weekend
2020	1		0	0
2020	1		0	0
2020	1	…	0	0
2020	1		0	1
2020	1		0	1
2020	1		0	0

Processing Tabular Data

- Tabular data can be loaded into memory by using the pandas “read_csv” function and passing the path into the dataset.

- The function is well suited and easy to use for loading in tabular data and can be used as follows:

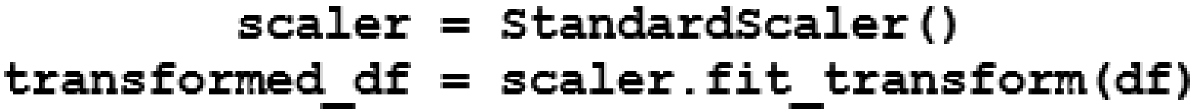


```
df = pd.read_csv('path/to/dataset')
```

Processing Tabular Data

- To use a scaler, it must be initialized and fit to the dataset. By doing this, the dataset can be transformed by the scaler.

- In fact, the fitting and transformation processes can be performed in one step by using the fit_transform method, as follows:



```
          scaler = StandardScaler()
  transformed_df = scaler.fit_transform(df)
```

"Complete Exercise"

“Loading Tabular Data and Rescaling Numerical Fields”

"Complete Activity"

“Loading Tabular Data and Rescaling Numerical Fields with a MinMax Scaler”

"Complete Exercise"

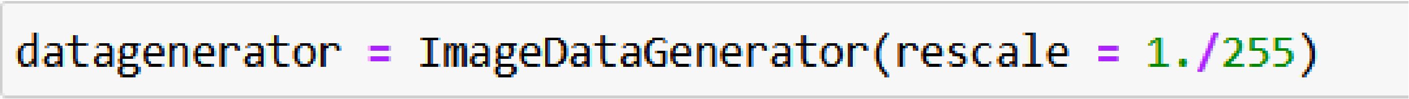
“Pre-processing Non-Numerical Data”

Processing Image Data

- A plethora of images is being generated every day by various organizations that can be used to create predictive models for tasks such as object detection, image classification, and object segmentation.
- When working with image data and some other raw data types, you often need to preprocess the data.
- Creating models from raw data with minimal preprocessing is one of the biggest benefits of using ANNs for modeling since the feature engineering step is minimal.

Processing Image Data

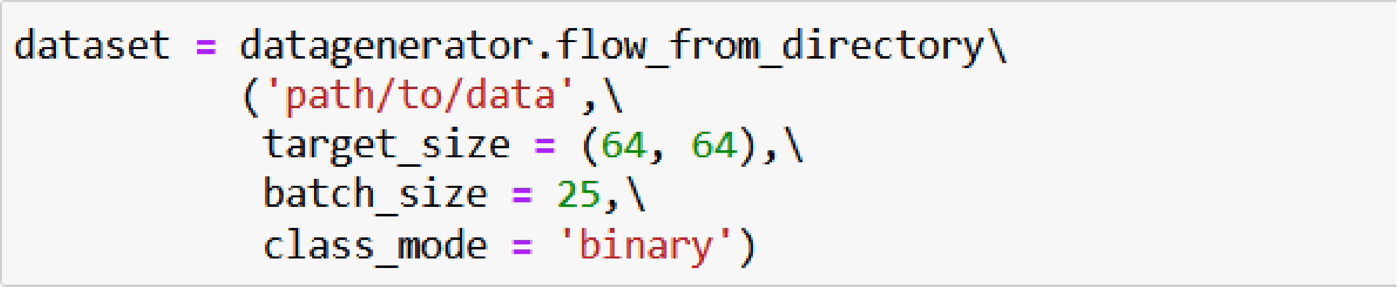
- ImageDataGenerator can be initialized with rescaling, as follows:



```
datagenerator = ImageDataGenerator(rescale = 1./255)
```

Processing Image Data

- Using the flow_from_directorymethod for binary classification with a batch size of 25 and an image size of 64x64 can be done as follows:



```
dataset = datagenerator.flow_from_directory\
          ('path/to/data',\
           target_size = (64, 64),\
           batch_size = 25,\
           class_mode = 'binary')
```

"Complete Exercise"

“Loading Image Data for Batch Processing”

Image Augmentation

- Image augmentation is the process of modifying images to increase the number of training examples available.
- This process can include zooming in on the image, rotating the image, or flipping the image vertically or horizontally.
- This can be performed if the augmentation process does not change the context of the image.
- For example, an image of a banana, when flipped horizontally, is still recognizable as a banana, and new images of bananas are likely to be of either orientation.

Image Augmentation

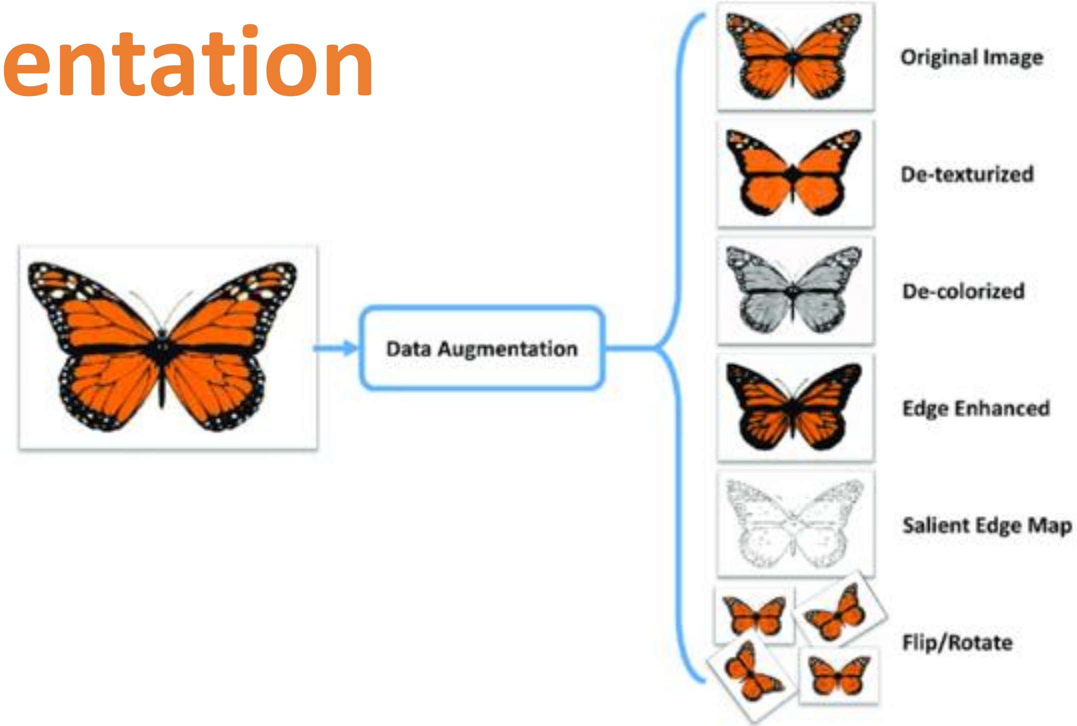
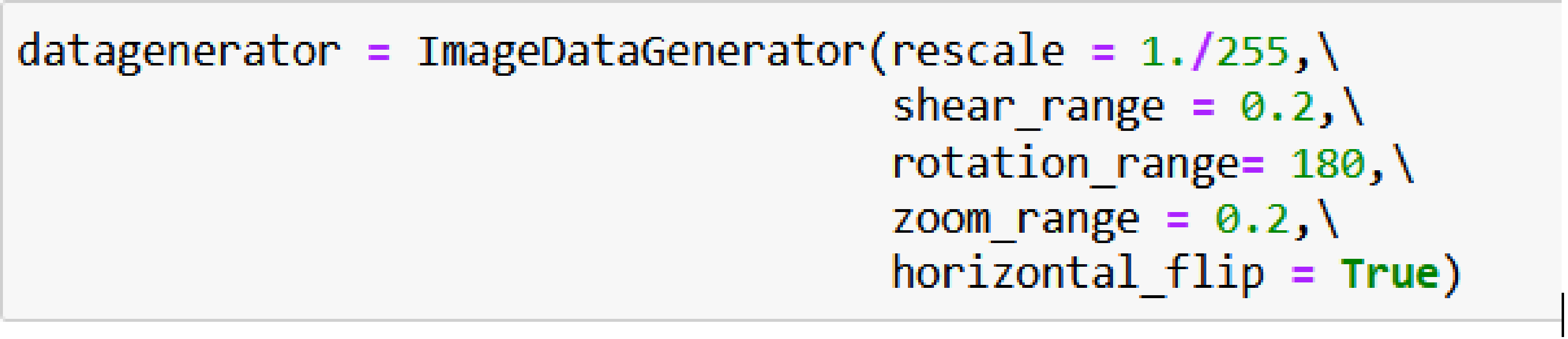


Figure 2.13: An example of image augmentation

Image Augmentation

- Image augmentation can be applied when instantiating the ImageDataGenerator class, as follows:



```
datagenerator = ImageDataGenerator(rescale = 1./255,\
                                   shear_range = 0.2,\
                                   rotation_range= 180,\
                                   zoom_range = 0.2,\
                                   horizontal_flip = True)
```

"Complete Activity"

“Loading Image Data for Batch Processing”

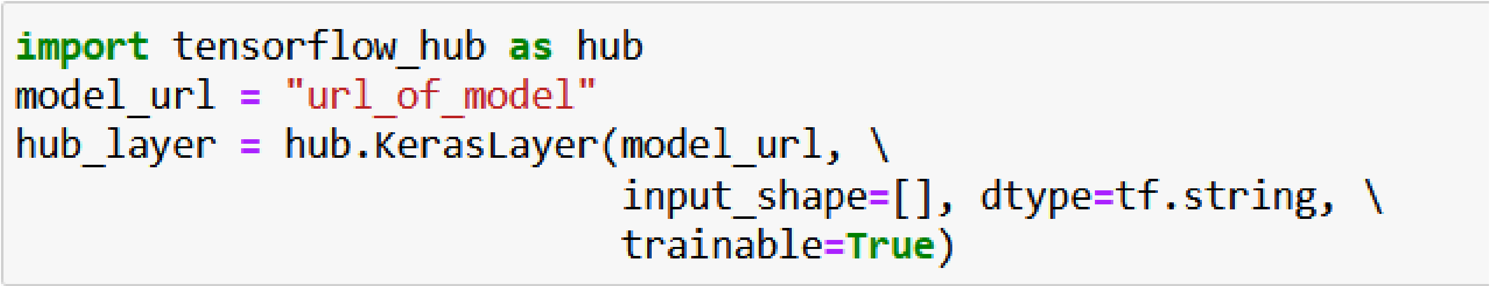
Text Processing

- Text data represents a large class of raw data that is readily available.

- For example, text data can be from web pages such as Wikipedia, transcribed speech, or social media conversations—all of which are increasing at a massive scale and must be processed before they can be used for training machine learning models.

Text Processing

- To load in the pretrained model, you need to import the tensorflow_hub library.
- By doing this, the URL of the model can be loaded, Then, the model can be loaded into the environment by calling the KerasLayer class, which wraps the model so that it can be used like any other TensorFlow model, It can be created as follows:

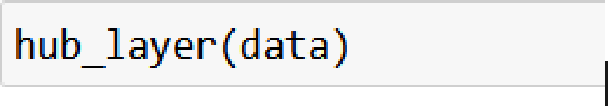


```
import tensorflow_hub as hub
model_url = "url_of_model"
hub_layer = hub.KerasLayer(model_url, \
                           input_shape=[], dtype=tf.string, \
                           trainable=True)
```

Text Processing

- The data type of the input data, indicated by the dtype parameter, should be used as input for the KerasLayer class, as well as a Boolean argument indicating whether the weights are trainable.

- Once the model has been loaded using the tensorflow_hub library, it can be called on text data, as follows:



```
hub_layer(data)
```

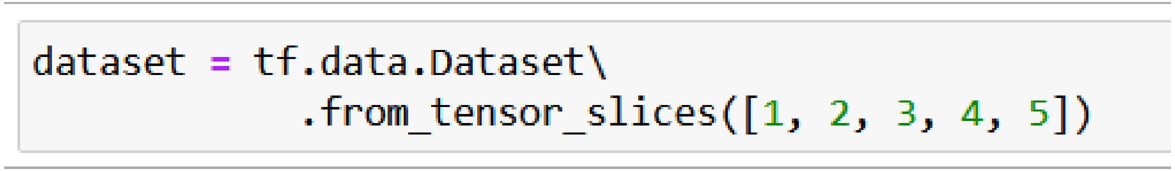
"Complete Exercise"

“Loading Text Data for TensorFlow Models”

Audio Processing

- A generic method for creating a dataset object from raw data is using TensorFlow's from_tensor_slicefunction.

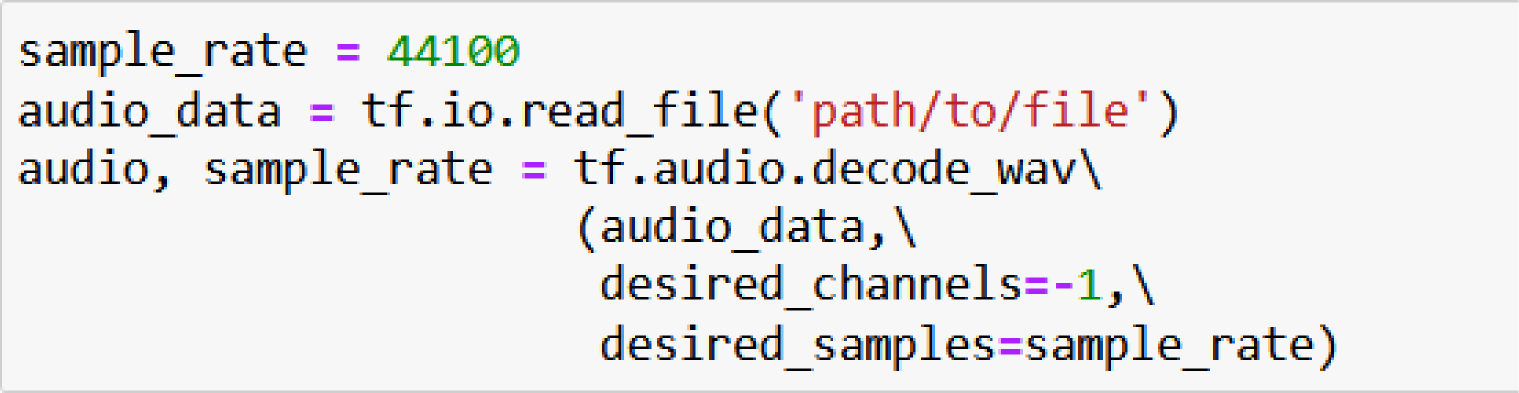
- This function generates a dataset object by slicing a tensor along its first dimension, It can be used as follows:



```
dataset = tf.data.Dataset\
            .from_tensor_slices([1, 2, 3, 4, 5])
```

Audio Processing

- For example, if a value of -1 is passed for the desired channel, then all the audio channels will be decoded. Importing the audio file can be achieved as follows:

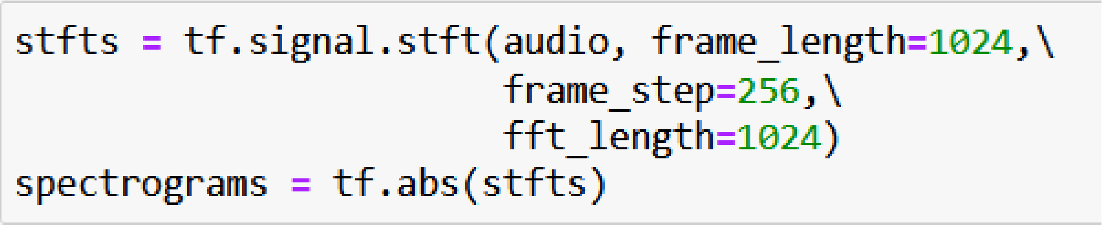


```
sample_rate = 44100
audio_data = tf.io.read_file('path/to/file')
audio, sample_rate = tf.audio.decode_wav\
                     (audio_data,\
                      desired_channels=-1,\
                      desired_samples=sample_rate)
```

Audio Processing

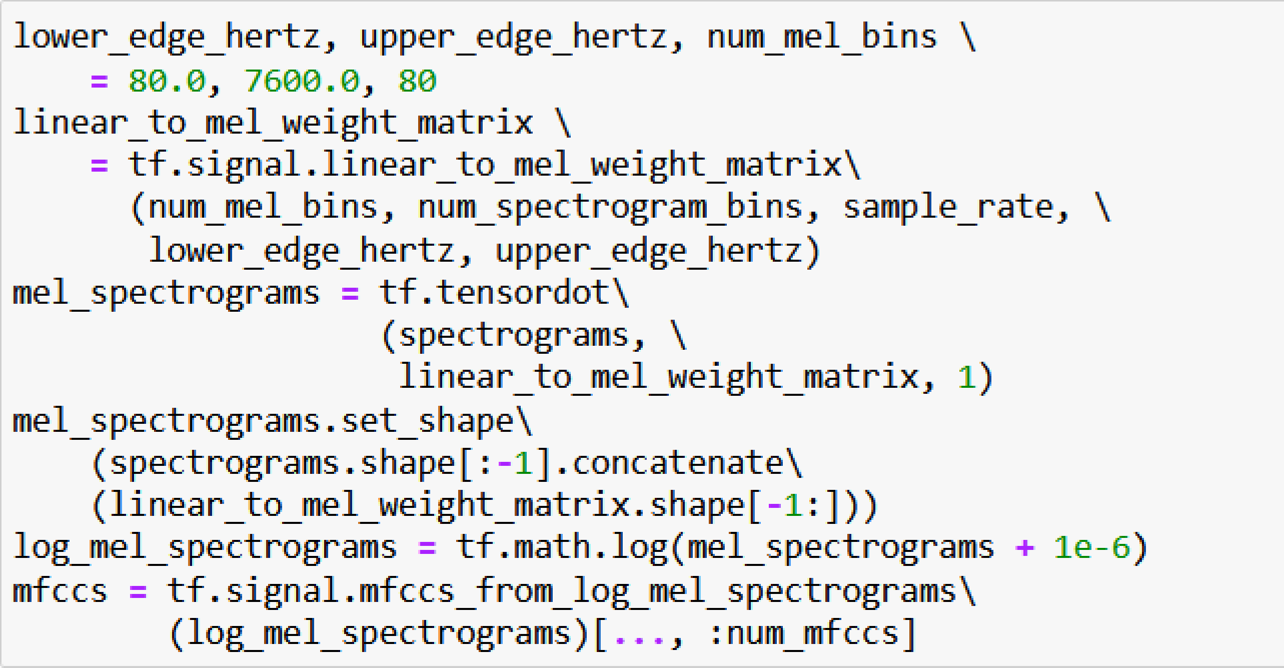
- A spectrogram is the absolute value of the short-time Fourier transform as it is useful for visual interpretation.

- The short-time Fourier transform and spectrogram can be created as follows:



```
stfts = tf.signal.stft(audio, frame_length=1024,\
                              frame_step=256,\
                              fft_length=1024)
spectrograms = tf.abs(stfts)
```

Audio Processing



```
lower_edge_hertz, upper_edge_hertz, num_mel_bins \
    = 80.0, 7600.0, 80
linear_to_mel_weight_matrix \
    = tf.signal.linear_to_mel_weight_matrix\
      (num_mel_bins, num_spectrogram_bins, sample_rate, \
       lower_edge_hertz, upper_edge_hertz)
mel_spectrograms = tf.tensordot\
                   (spectrograms, \
                    linear_to_mel_weight_matrix, 1)
mel_spectrograms.set_shape\
    (spectrograms.shape[:-1].concatenate\
    (linear_to_mel_weight_matrix.shape[-1:]))
log_mel_spectrograms = tf.math.log(mel_spectrograms + 1e-6)
mfccs = tf.signal.mfccs_from_log_mel_spectrograms\
        (log_mel_spectrograms)[..., :num_mfccs]
```

"Complete Exercise"

“Loading Audio Data for TensorFlow Models”

"Complete Activity"

“Loading Audio Data for Batch Processing”

Summary

- In this lesson, you learned how to load different forms of data and perform some pre-processing steps for a variety of data types.
- You began with tabular data in the form of a CSV file, Since the dataset consisted of a single CSV file, you utilized the pandas library to load the file into memory.
- You then proceeded to pre-process the data by scaling the fields and converting all the fields into numerical data types.

3. TensorFlow Development

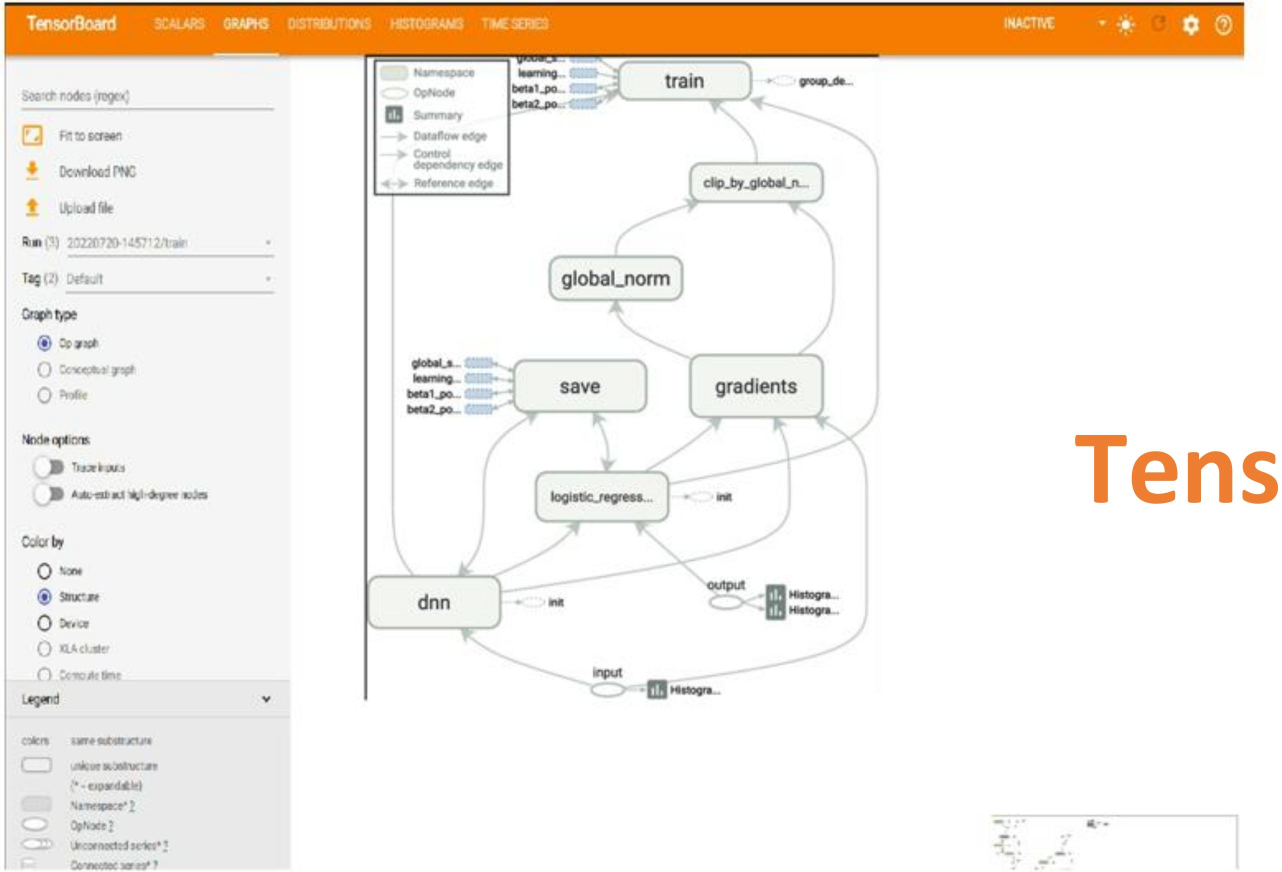
Introduction

- In this lesson, you will learn about TensorFlow resources that will aid you in your model building and help you create performant machine learning algorithms.

- You will explore the practical resources that practitioners can utilize to aid their development workflow, including TensorBoard, TensorFlow Hub, and Google Colab.

TensorBoard

- TensorBoard is a visualization toolkit used to aid in machine learning experimentation.
- The platform has dashboard functionality for visualizing many of the common data types that a data science or machine learning practitioner may need at once, such as scalar values, image batches, and audio files.
- While such visualizations can be created with other plotting libraries, such as matplotlib or ggplot, TensorBoard combines many visualizations in an easy-to-use environment.



Run (3) 20220720-145712/train

Tag (2) Default

Graph type

● Op graph

○ Conceptual graph

○ Profile

Color by

○ None

● Structure

○ Device

○ XLA cluster

○ Compute time

TensorBoard

Figure 3.1: A visual representation of model graphs and functions in TensorBoard

TensorBoard

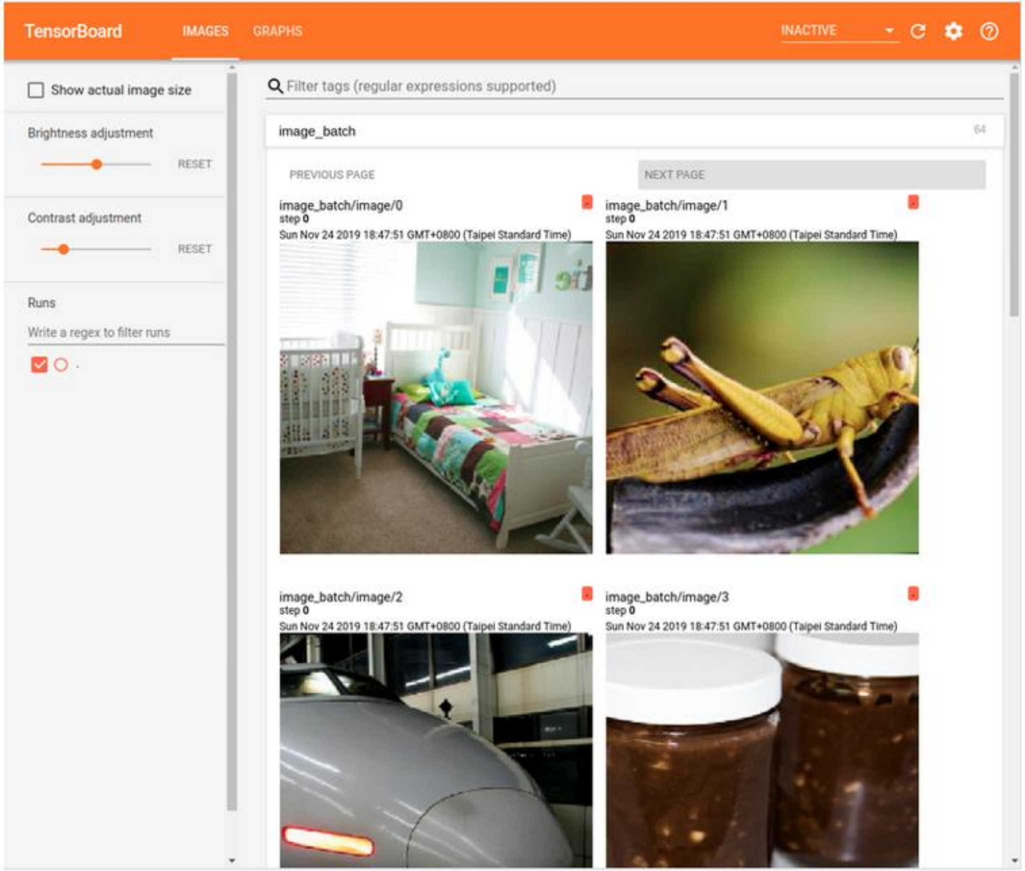
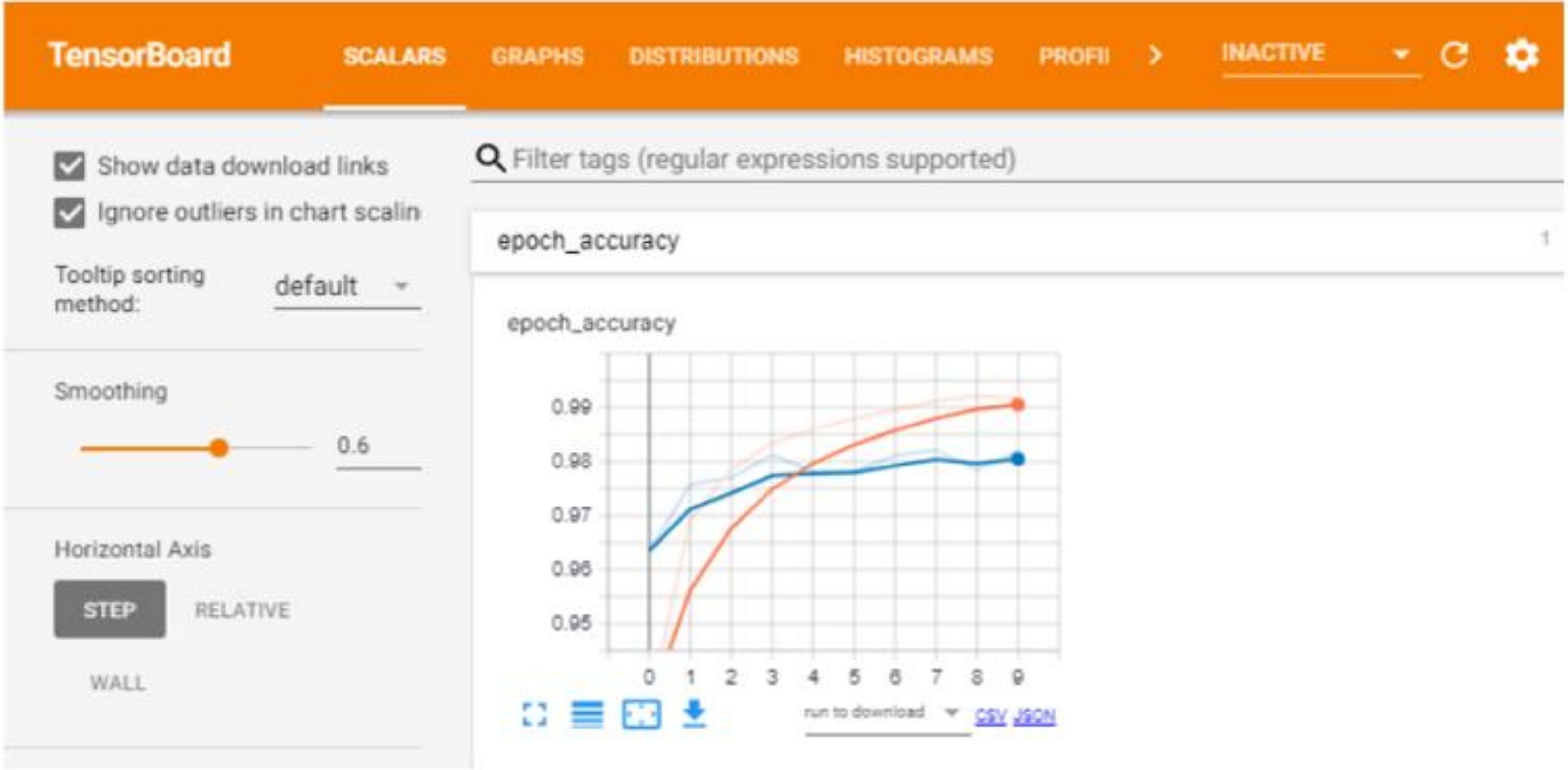

TensorBoard
IMAGES
GRAPHS
INACTIVE
□ Show actual image size
Brightness adjustment
RESET
Contrast adjustment
RESET
Runs
Write a regex to filter runs
☑ ○ .
Filter tags (regular expressions supported)
image_batch
64
PREVIOUS PAGE
NEXT PAGE
image_batch/image/0
step 0
Sun Nov 24 2019 18:47:51 GMT+0800 (Taipei Standard Time)
image_batch/image/1
step 0
Sun Nov 24 2019 18:47:51 GMT+0800 (Taipei Standard Time)
image_batch/image/2
step 0
Sun Nov 24 2019 18:47:51 GMT+0800 (Taipei Standard Time)
image_batch/image/3
step 0
Sun Nov 24 2019 18:47:51 GMT+0800 (Taipei Standard Time)

Figure 3.2: Viewing images in TensorBoard



☑ Show data download links

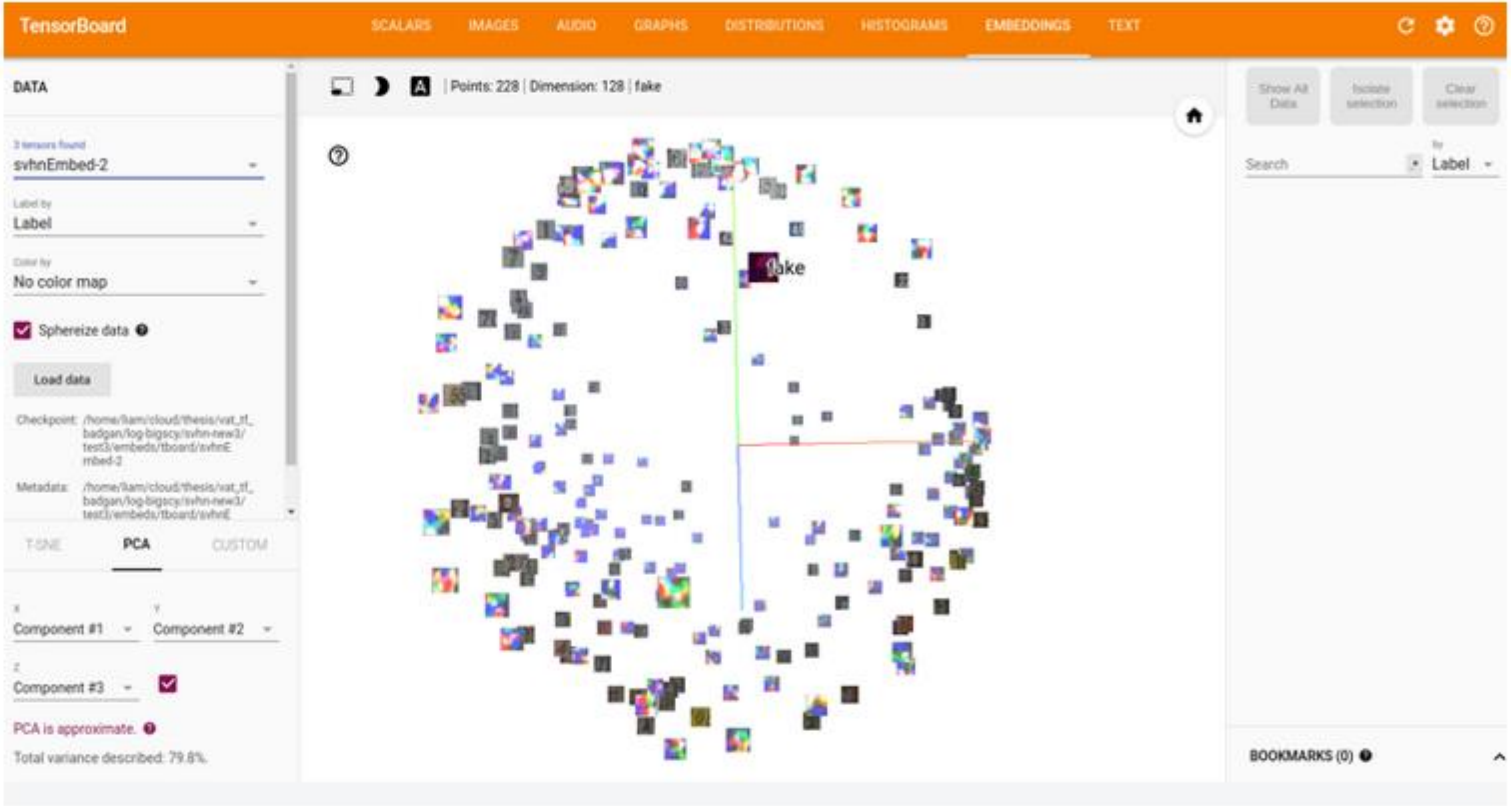
☑ Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis STEP

Figure 3.3: Plotting model evaluation metrics in TensorBoard



3 tensors found: svhnEmbed-2

Label by: Label

Color by: No color map

☑ Sphereize data

Checkpoint: /home/liam/cloud/thesis/vat_tf_badgan/log-bigscy/svhn-new3/test3/embeds/tboard/svhnEmbed-2

Metadata: /home/liam/cloud/thesis/vat_tf_badgan/log-bigscy/svhn-new3/test3/embeds/tboard/svhnE

X: Component #1　Y: Component #2

Z: Component #3 ☑

PCA is approximate.

Total variance described: 79.8%.

| Points: 228 | Dimension: 128 | fake

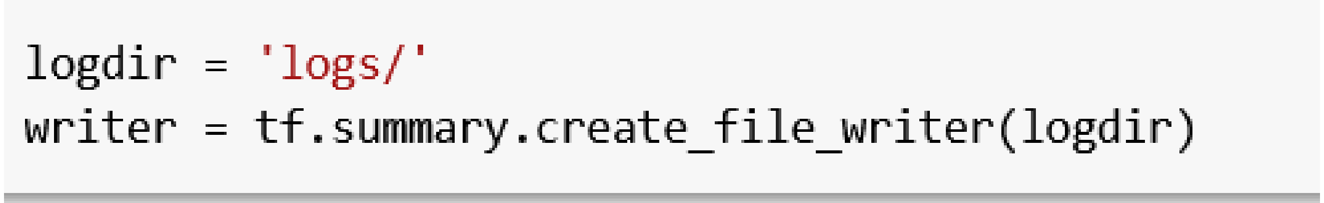
By: Label

Figure 3.4: Visualizing embedding vectors in TensorBoard

TensorBoard

- The file writer is typically created at the beginning of a Jupyter notebook or equivalent development environment before any logs are written.

- This can be done as follows:

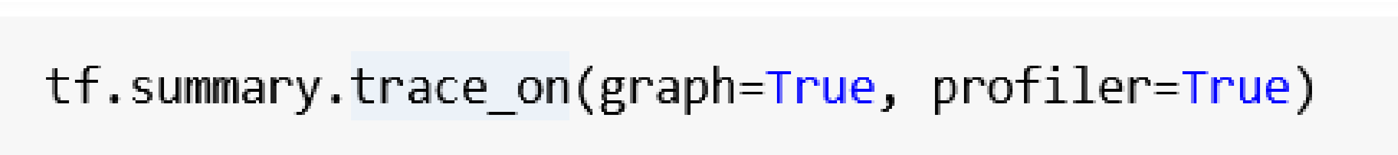


```
logdir = 'logs/'
writer = tf.summary.create_file_writer(logdir)
```

TensorBoard

- The file writer object writes a file to the log directory when the logs are exported.

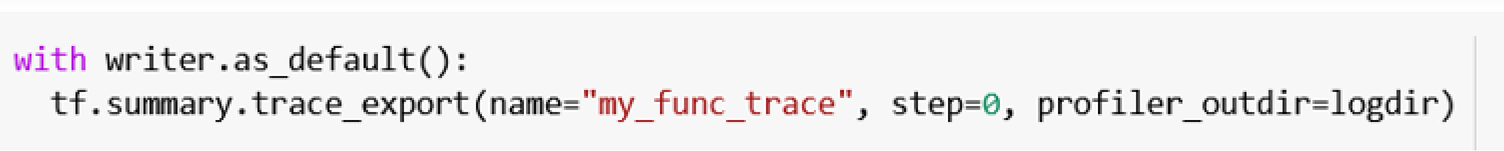
- To begin tracing, the following code must be executed:



```
tf.summary.trace_on(graph=True, profiler=True)
```

TensorBoard

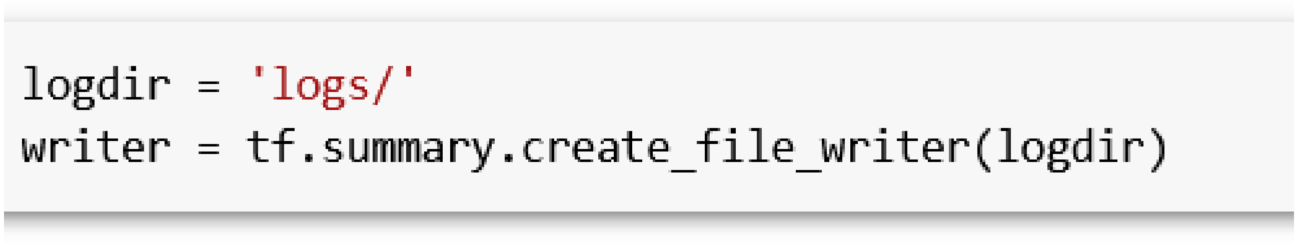
- Once the tracing of the computational graph is complete, the logs can be written to the log directory using the file writer object, as follows:



```
with writer.as_default():
  tf.summary.trace_export(name="my_func_trace", step=0, profiler_outdir=logdir)
```

TensorBoard

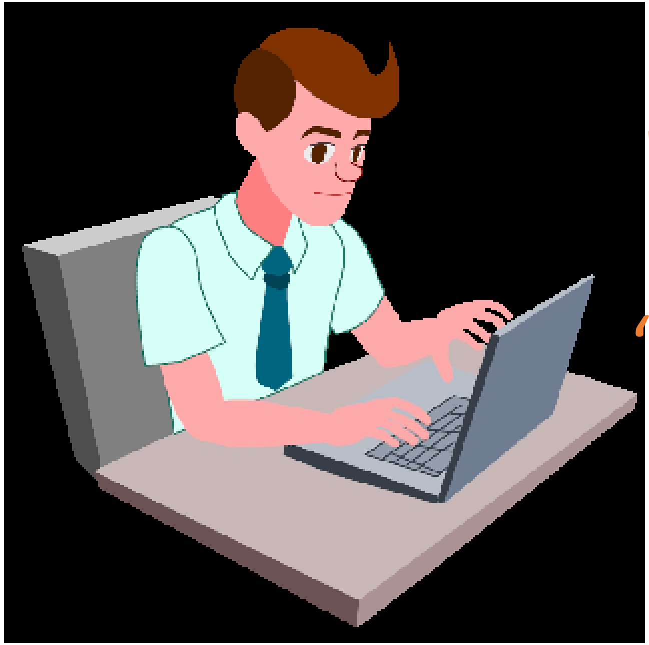
- After logs have been written to a directory, TensorBoard can be launched through the command line using the following command, thereby passing in the directory for the logs as the logdir parameter:



```
logdir = 'logs/'
writer = tf.summary.create_file_writer(logdir)
```

"Complete Exercise"

“Using TensorBoard to Visualize Matrix Multiplication”



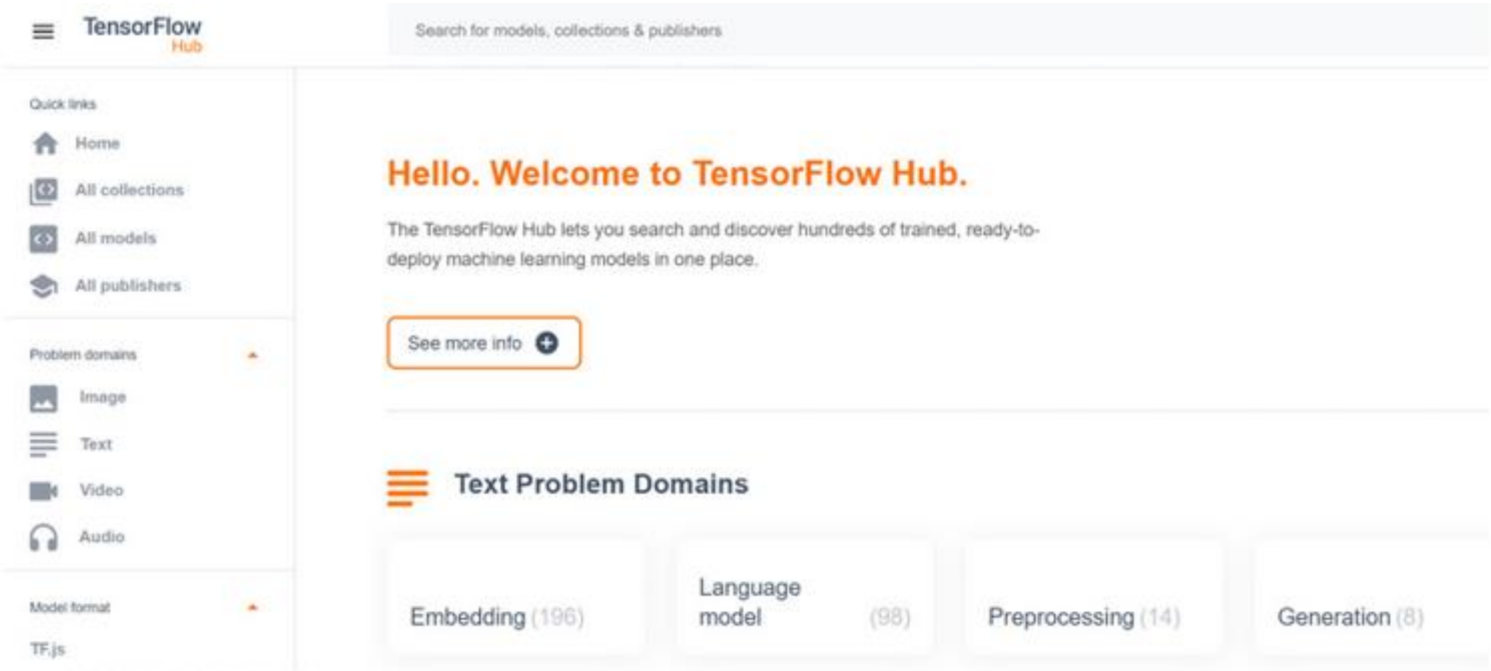
"Complete Activity"

"Using TensorBoard to Visualize Tensor Transformations"

"Complete Exercise"

"Using TensorBoard to Visualize Image Batches"

TensorFlow Hub



Hello. Welcome to TensorFlow Hub.

The TensorFlow Hub lets you search and discover hundreds of trained, ready-to-deploy machine learning models in one place.

Figure 3.8: TensorFlow Hub home page

TensorFlow Hub

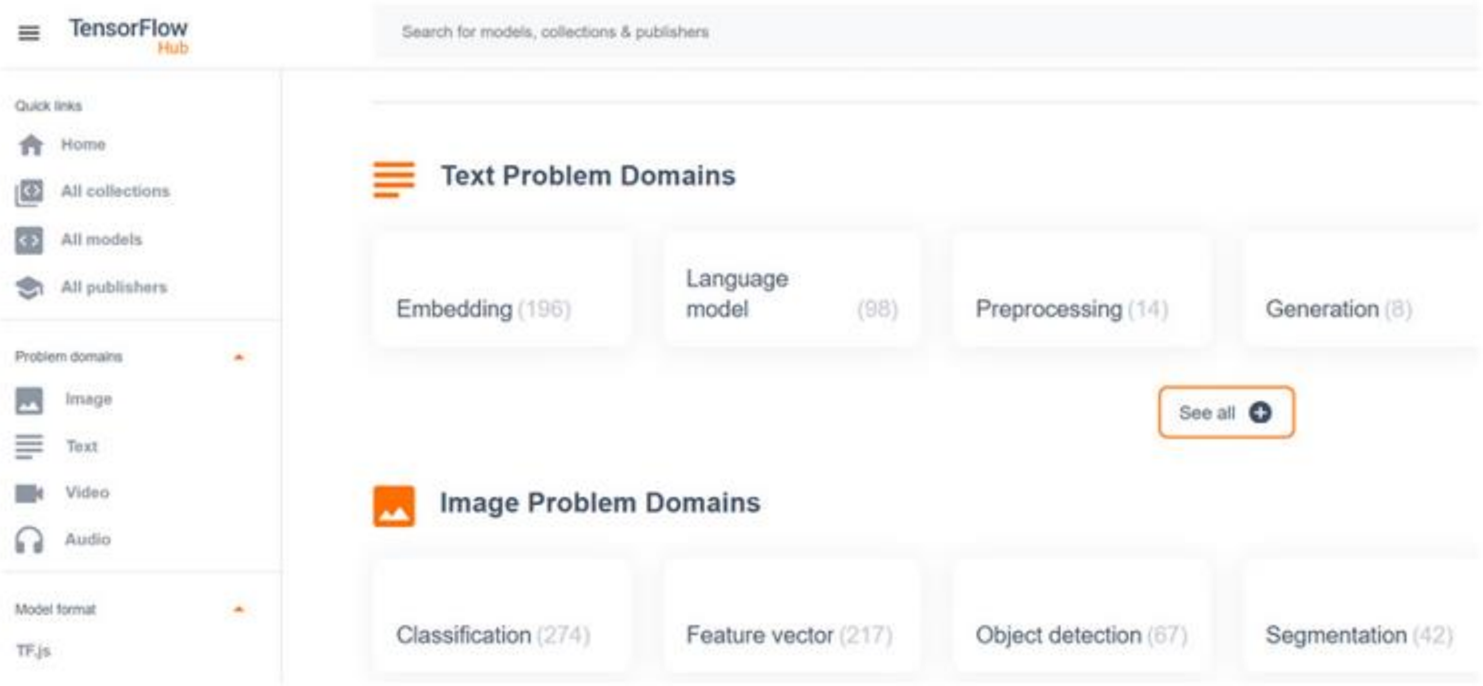


Figure 3.9: The model domains available on TensorFlow Hub

TensorFlow Hub

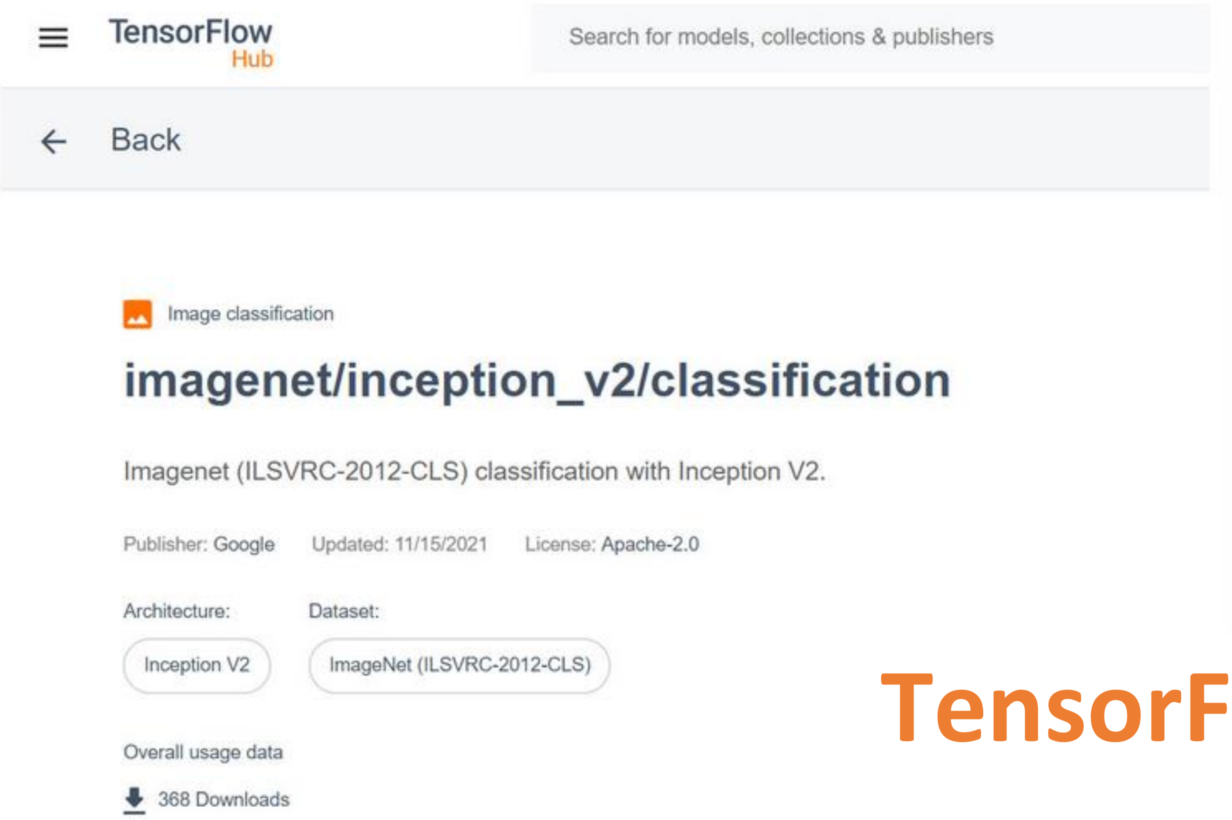


Image classification

imagenet/inception_v2/classification

Imagenet (ILSVRC-2012-CLS) classification with Inception V2.

Publisher: Google Updated: 11/15/2021 License: Apache-2.0

Architecture: Inception V2

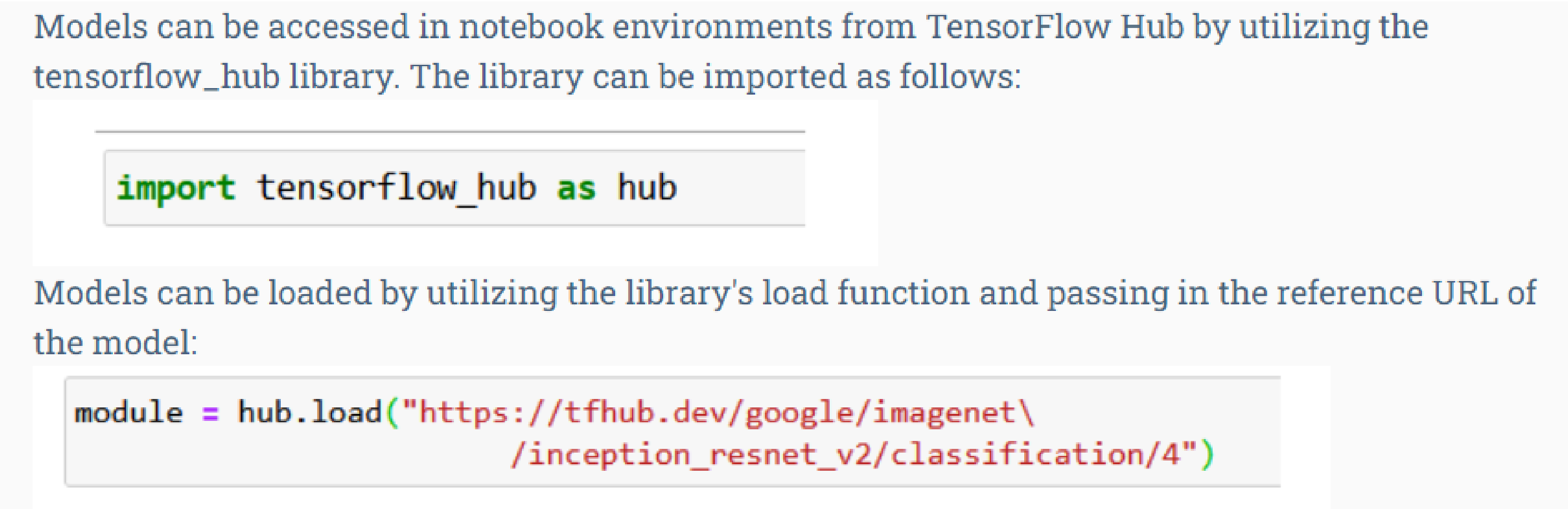
Dataset: ImageNet (ILSVRC-2012-CLS)

Overall usage data

368 Downloads

Figure 3.10: The page of a TensorFlow Hub model

TensorFlow Hub



Models can be accessed in notebook environments from TensorFlow Hub by utilizing the tensorflow_hub library. The library can be imported as follows:

```
import tensorflow_hub as hub
```

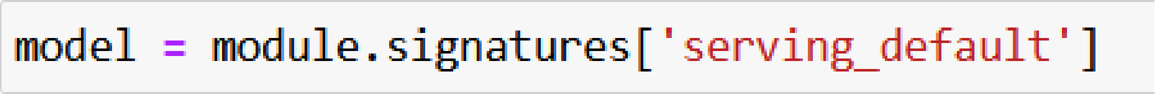
Models can be loaded by utilizing the library's load function and passing in the reference URL of the model:

```
module = hub.load("https://tfhub.dev/google/imagenet\
                                /inception_resnet_v2/classification/4")
```

TensorFlow Hub

- Assets of the model's module, such as its architecture, can be viewed by accessing the signatures attribute.

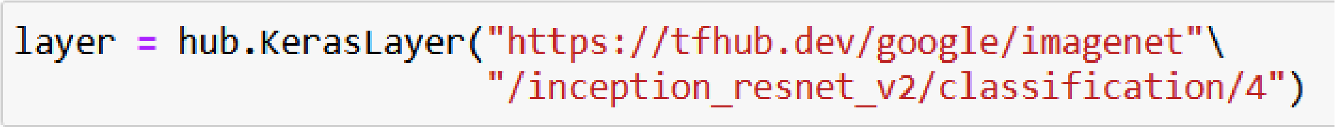
- Each model may have different keys within the signatures attribute; however, much of the pertinent information will be contained within the “serving_default” key:



```
model = module.signatures['serving_default']
```

TensorFlow Hub

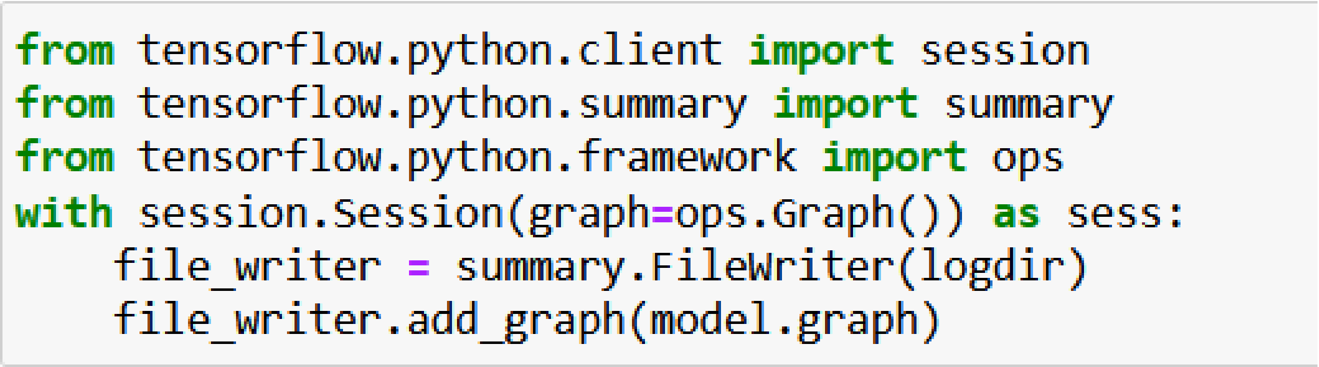
- The model can also be used directly in training by treating the whole model like a single Keras layer using the KerasLayer method:



```
layer = hub.KerasLayer("https://tfhub.dev/google/imagenet"\
                       "/inception_resnet_v2/classification/4")
```

TensorFlow Hub

- Viewing a model in TensorFlow Hub can be done by writing the model graph to the logs using a file writer as follows:



```
from tensorflow.python.client import session
from tensorflow.python.summary import summary
from tensorflow.python.framework import ops
with session.Session(graph=ops.Graph()) as sess:
    file_writer = summary.FileWriter(logdir)
    file_writer.add_graph(model.graph)
```

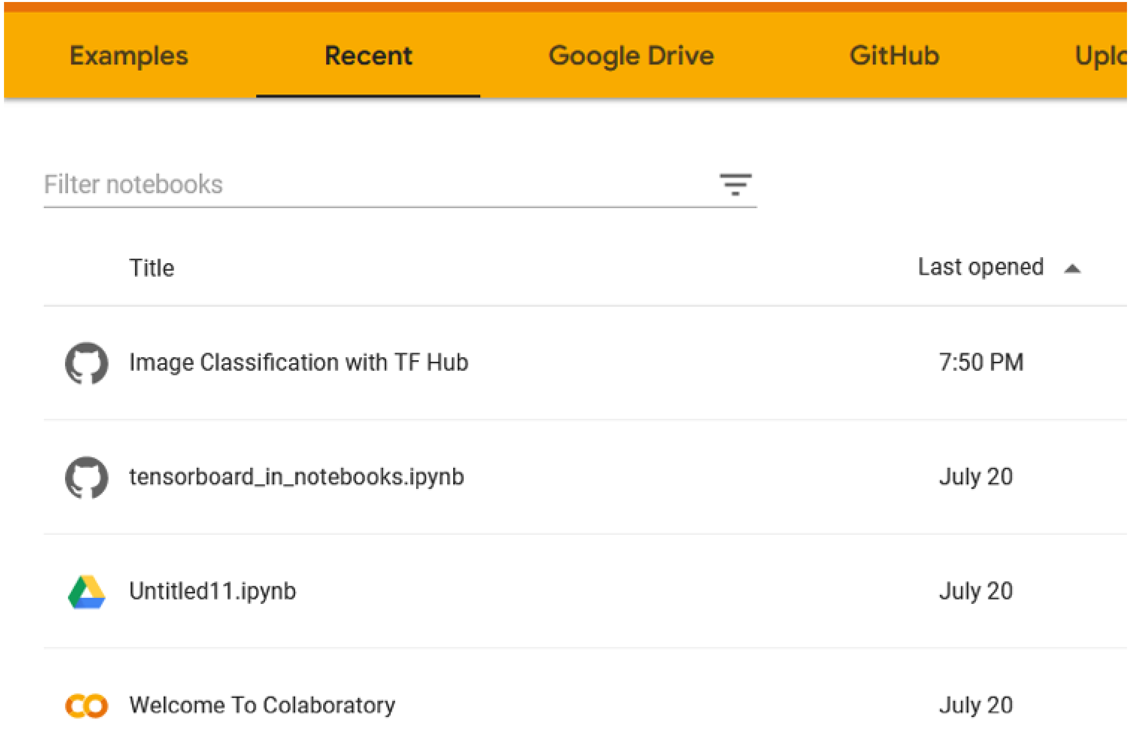
"Complete Exercise"

“Downloading a Model from TensorFlow Hub”

Google Colab

- Google Colab enables users to execute code on Google servers and is designed specifically for data science practitioners to develop code for machine learning in a collaborative environment.
- The platform is available at https://colab.research.google.com/ and offers an opportunity to develop in the Python programming language directly within a web browser with no code executing on your local machine.

Development on Google Colab



Title	Last opened
Image Classification with TF Hub	7:50 PM
tensorboard_in_notebooks.ipynb	July 20
Untitled11.ipynb	July 20
Welcome To Colaboratory	July 20

Figure 3.12: The home page of Google Colab

Development on Google Colab

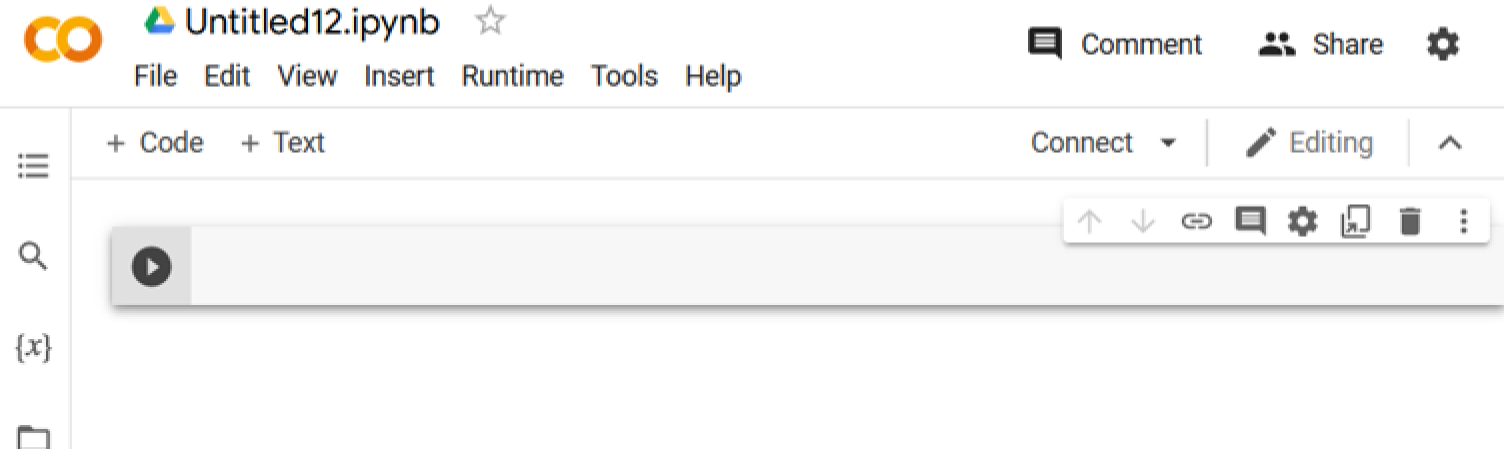


Figure 3.13: A blank notebook in Google Colab

"Complete Exercise"

“Using Google Colab to Visualize Data”

"Complete Activity"

“Performing Word Embedding from a Pre-Trained Model from TensorFlow Hub”

Summary

- In this lesson, you used a variety of TensorFlow resources, including TensorBoard, TensorFlow Hub, and Google Colab.
- TensorBoard offers users a method to visualize computational model graphs, metrics, and any experimentation results.
- TensorFlow Hub allows users to accelerate their machine learning development using pre-trained models built by experts in the field.

4. Regression and Classification Models

Introduction

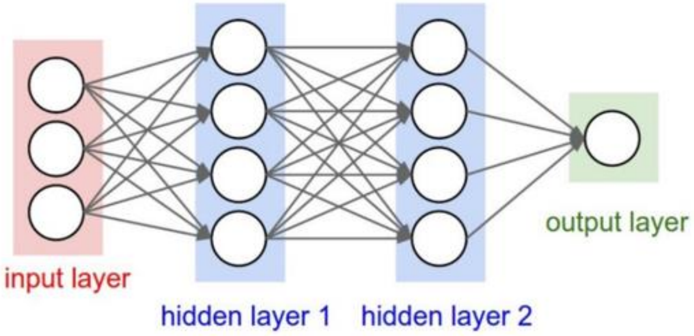
- In this lesson, you will explore how to create ANNs using TensorFlow.
- You will build ANNs with different architectures to solve regression and classification tasks. Regression tasks aim to predict continuous variables from the input training data, while classification tasks aim to classify the input data into two or more classes.
- For example, a model to predict whether or not it will rain on a given day is a classification task since the result of the model will be of two classes—rain or no rain.

Sequential Models

- A sequential model is used to build regression and classification models. In sequential models, information propagates through the network from the input layer at the beginning to the output layer at the end.

- Layers are stacked in the model sequentially, with each layer having an input and an output.

Sequential Models



Sequential Model

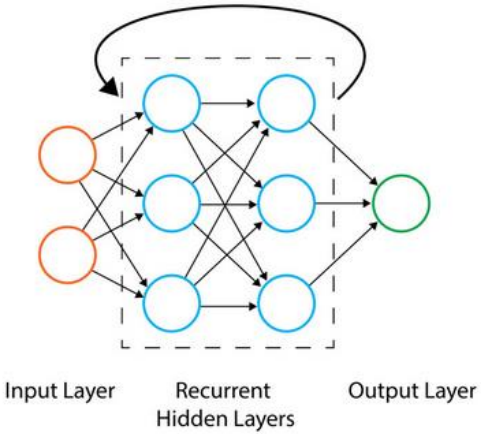
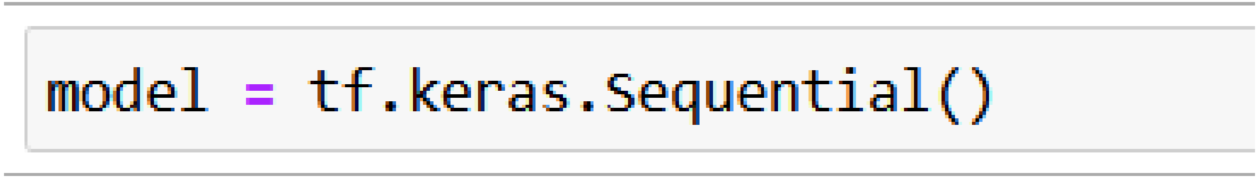


Figure 4.1: Recurrent Model

Sequential Models

- A sequential model can be initialized as follows:



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

Keras Layers

- Keras layers are included in the TensorFlow package, Keras layers are a collection of commonly used layers that can be added easily to your sequential models.

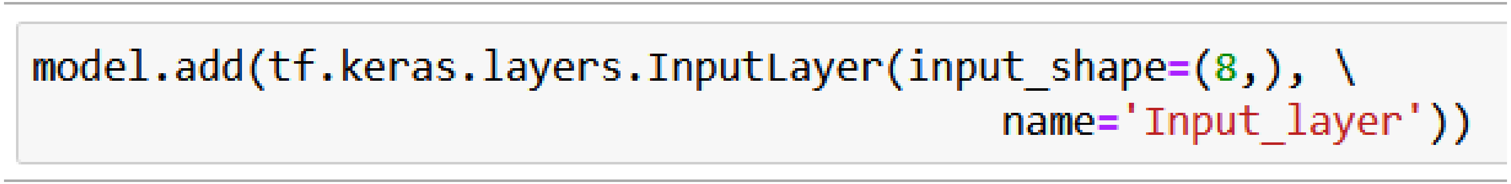
- To add layers to a model of the Sequential class, you can use the model's add method.

- One optional layer that can be added to the beginning of a sequential model is an input layer as an entry point to the network.

Keras Layers

- Input layers can be added to a model as follows.

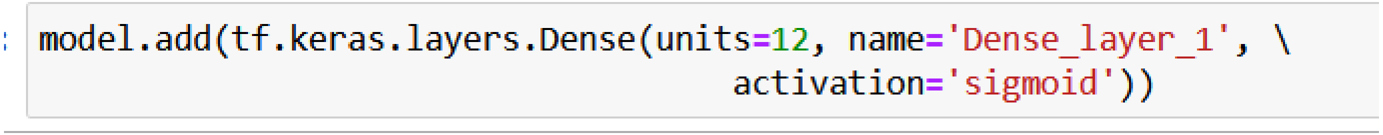
- The following code snippet is used to add a layer, expecting inputs to have eight features:



```
model.add(tf.keras.layers.InputLayer(input_shape=(8,), \
                                         name='Input_layer'))
```

Keras Layers

- The following is an example of adding a dense layer to a model with 12 units, adding a sigmoid activation function at the output of the layer, and naming the layer Dense_layer_1:



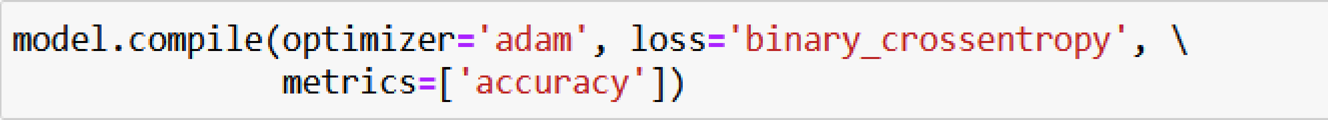
```
model.add(tf.keras.layers.Dense(units=12, name='Dense_layer_1', \
                                    activation='sigmoid'))
```

"Complete Exercise"

“Creating an ANN with TensorFlow”

Model Fitting

- Once a model has been initialized and layers have been added to the ANN, the model must be configured with an optimizer, losses, and any evaluation metrics through the compilation process.
- A model can be compiled using the model's compile method, as follows:



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

The Loss Function

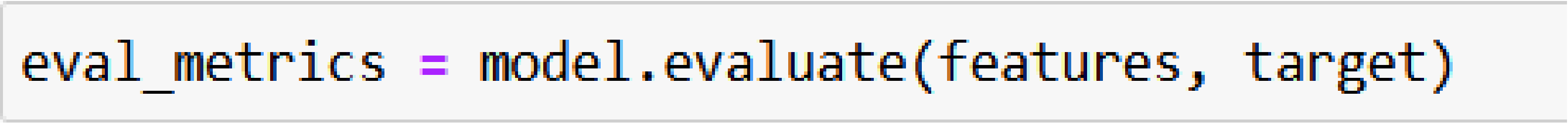
- The loss function is the measure of error between the predicted results and the true results.
- You use the loss function during the training process to determine whether varying any of the weights and biases will create a better model by minimizing the loss function's value through the optimization process.
- There are many different types of loss functions that can be used, and the specific one will depend on the problem and goal.

Model Evaluation

- Once models are trained, they can be evaluated by utilizing the model's evaluate method.
- The evaluate method assesses the performance of the model according to the loss function used to train the model and any metrics that were passed to the model.
- The method is best used when determining how the model will perform on new, unseen data by passing in a feature and target dataset that has not been used in the training process or out-of-sample dataset.

Model Evaluation

- The method can be called as follows:



```
eval_metrics = model.evaluate(features, target)
```

"Complete Exercise"

“Creating a Linear Regression Model as an ANN with TensorFlow”

"Complete Exercise"

“Creating a Multi-Layer ANN with TensorFlow”

"Complete Activity"

“Creating a Multi-Layer ANN with TensorFlow”

Classification Models

- The goal of classification models is to classify data into distinct classes.

- For example, a spam filter is a classification model that aims to classify emails into "spam" (referring to unsolicited and unwanted email) or "ham" (a legitimate email).

- Spam filters are an example of a binary classifier since there are two classes.

Classification Models

sigmoid function: $\sigma(t) = \frac{1}{1+e^{-t}}$

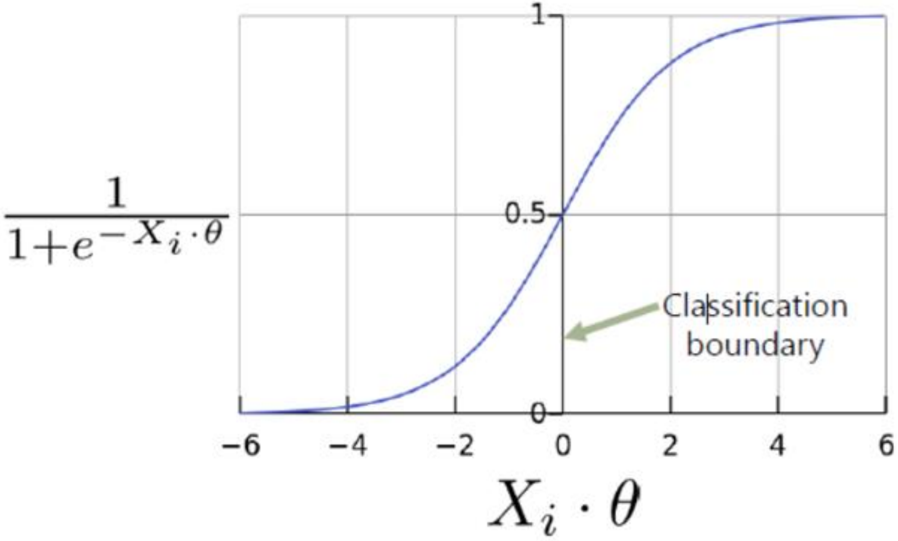


Figure 4.12: A visual representation of the sigmoid function

"Complete Exercise"

"Creating a Logistic Regression Model as an ANN with TensorFlow"

"Complete Activity"

“Creating a Multi-Layer Classification ANN with TensorFlow”

Summary

- In this lesson, you began your journey into creating ANNs in TensorFlow.
- You saw how simple it is to create regression and classification models by utilizing Keras layers.
- Keras layers are distinct classes that exist in a separate library that uses TensorFlow in the backend.
- Due to their popularity and ease of use, they are now included in TensorFlow and can be called in the same way as any other TensorFlow class.

5. Classification Models

Introduction

- In this lesson, you will look at another type of supervised learning problem called classification, where the target variable is discrete — meaning it can only take a finite number of values.
- In industry, you will most likely encounter such projects where variables are aggregated into groups such as product tiers, or classes of users, customers, or salary ranges.
- The objective of a classifier is to learn the patterns from the data and predict the right class for observation.

Binary Classification

- Machine learning algorithms such as the random forest classifier, support vector classifier, or logistic regression work well for classification.
- Neural networks can also achieve good results for binary classification.
- It is extremely easy to turn a regression model such as those in the previous lesson into a binary classifier.
- There are only two key changes required: the activation function for the last layer and the loss function.

Logistic Regression

- Logistic regression is one of the most popular algorithms for dealing with binary classification.
- As its name implies, it is an extension of the linear regression algorithm.
- A linear regression model predicts an output that can take an infinite number of values within a range.
- For logistic regression, you want your model to predict values between 0 and 1, The value 0 usually corresponds to false (or no) while the value 1 refers to true (or yes).

Logistic Regression

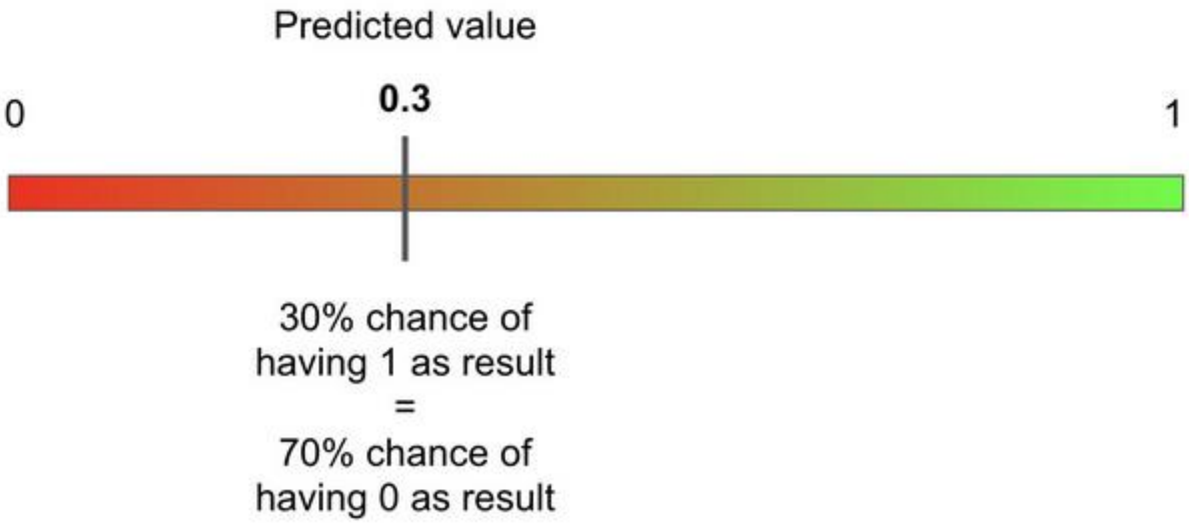

Predicted value
0
0.3
1
30% chance of having 1 as result
=
70% chance of having 0 as result

Figure 5.1: Output of logistic regression

Logistic Regression

- Luckily, such a mathematical function does exist, and it is called the sigmoid function.
- The formula for this function is as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

Figure 5.2: Formula of the sigmoid function

e^x corresponds to the exponential function applied to x.

Logistic Regression

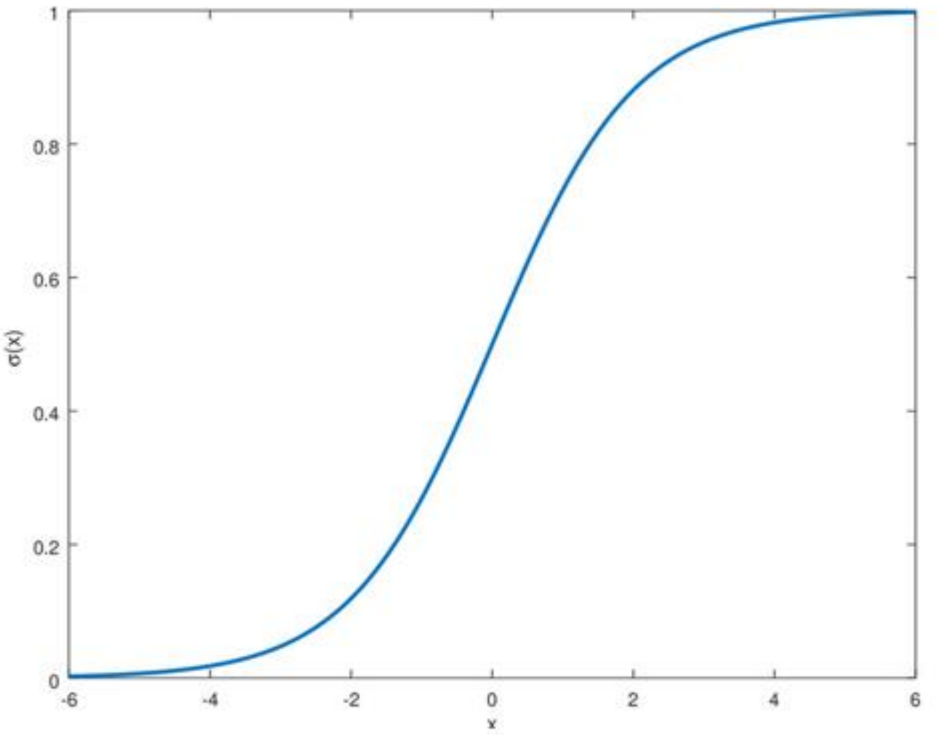
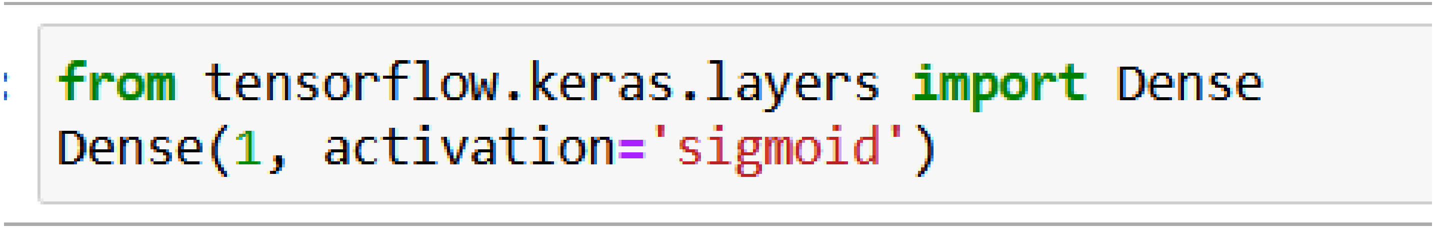


Figure 5.3: Curve of the sigmoid function

Logistic Regression

- To do so, you just need to specify sigmoid as the activation function of the last fully connected layer of a perceptron model.

- In TensorFlow, you specify the activation parameter as:



```
from tensorflow.keras.layers import Dense
Dense(1, activation='sigmoid')
```

Binary Classification Architecture

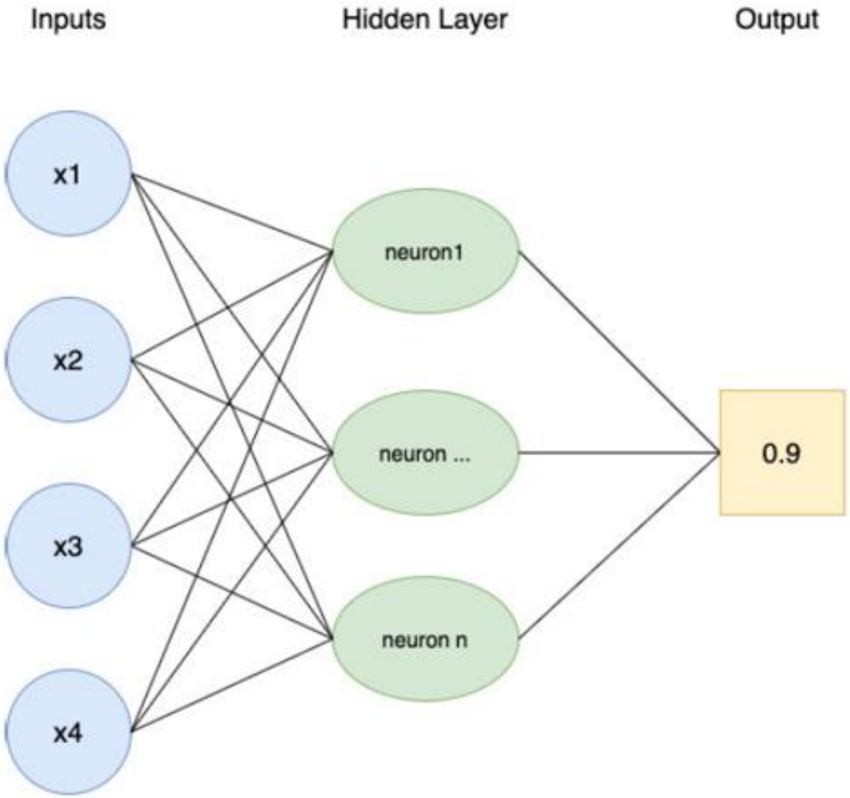


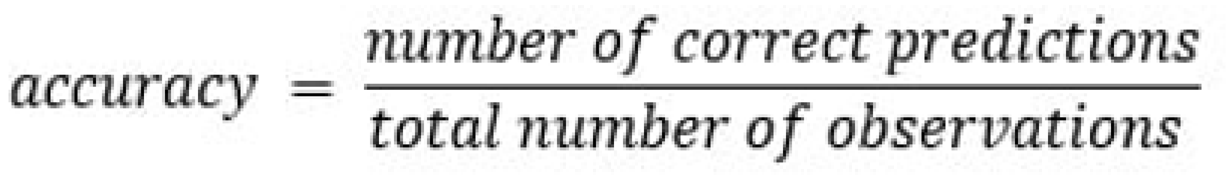
Figure 5.5: Architecture of the binary classifier

"Complete Exercise"

“Building a Logistic Regression Model”

Accuracy and Null Accuracy

- One of the most widely used metrics for classification problems is accuracy.
- Its formula is quite simple:



$$accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ observations}$$

Formula of the accuracy metric

Accuracy and Null Accuracy

- For a binary classifier, the number of correct predictions is the number of observations with a value of 0 or 1 as the correctly predicted value:

$$accuracy = \frac{number\ of\ correct\ predictions\ for\ value\ 0\ +\ number\ of\ correct\ predictions\ for\ value\ 1}{total\ number\ of\ observations}$$

Formula of the accuracy metric for a binary classifier

Accuracy and Null Accuracy

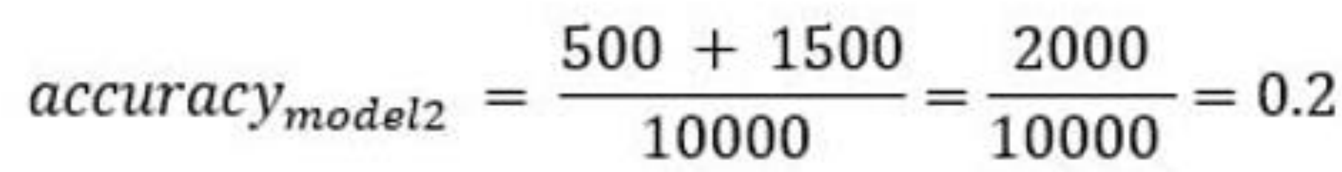
- You are assessing the performance of two different binary classifiers predicting the outcome on 10,000 observations on the test set.
- The first model correctly predicted 5,000 instances of value 0 and 3,000 instances of value 1.
- Its accuracy score will be as follows:

$$accuracy_{model1} = \frac{5000 + 3000}{10000} = \frac{8000}{10000} = 0.8$$

Formula for the accuracy of model1

Accuracy and Null Accuracy

- The second model correctly predicted the value 0 for 500 cases and the value 1 for 1,500 observations. Its accuracy score will be as follows:

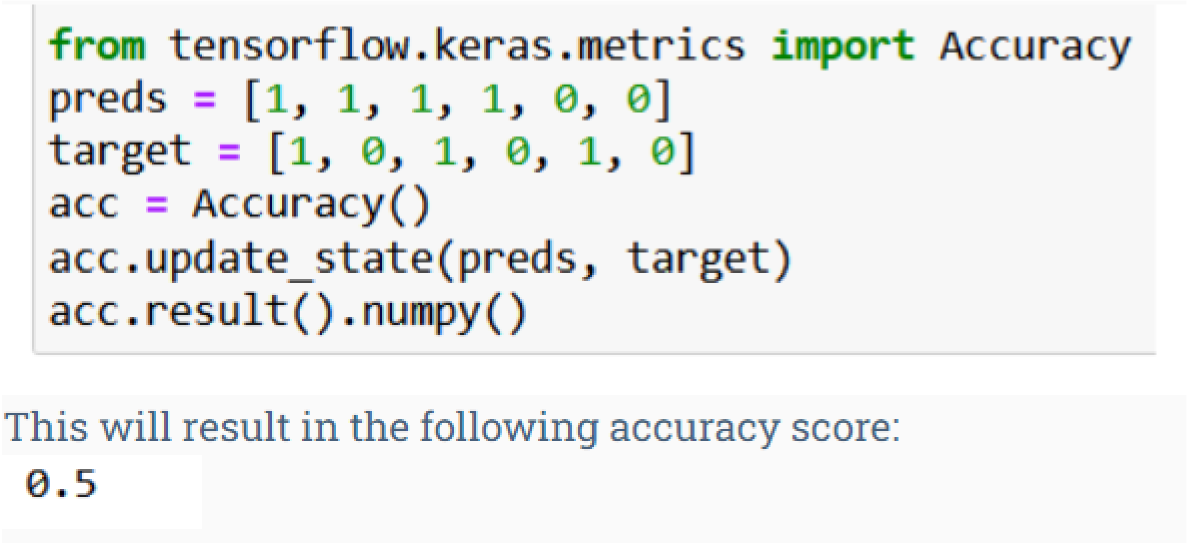


$$accuracy_{model2} = \frac{500 + 1500}{10000} = \frac{2000}{10000} = 0.2$$

Formula for the accuracy of model2

Accuracy and Null Accuracy

- You can use the numpy() method to convert it into a NumPy array.
- Here is an example of how to calculate the accuracy score:

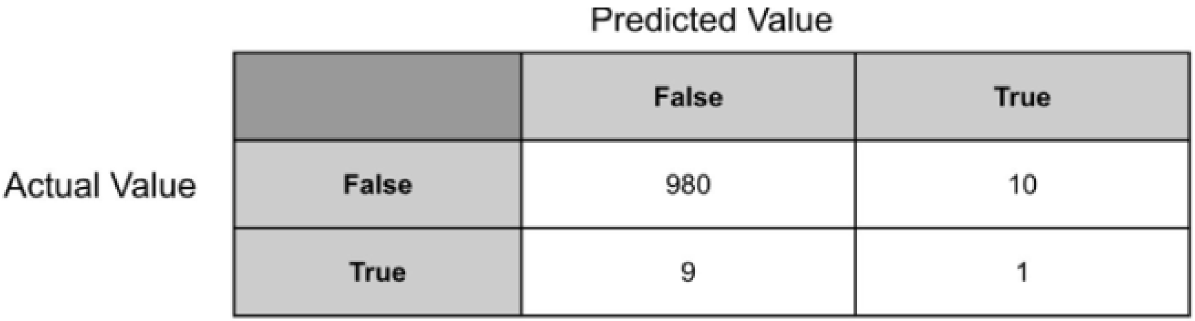


```
from tensorflow.keras.metrics import Accuracy
preds = [1, 1, 1, 1, 0, 0]
target = [1, 0, 1, 0, 1, 0]
acc = Accuracy()
acc.update_state(preds, target)
acc.result().numpy()
```

This will result in the following accuracy score:

```
0.5
```

Precision, Recall, and the F1 Score



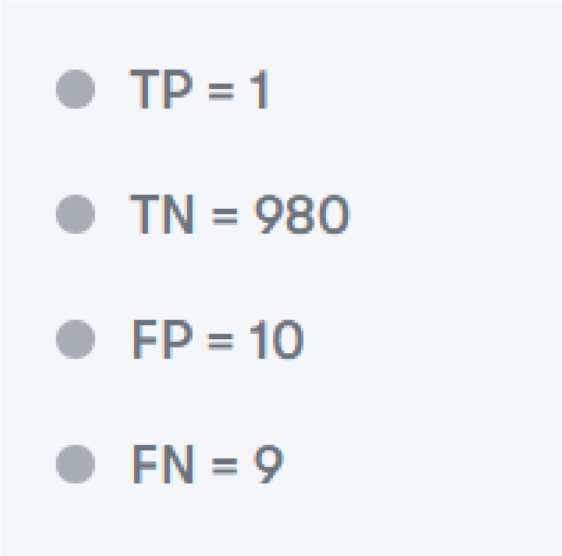
		Predicted Value	
		False	True
Actual Value	False	980	10
	True	9	1

Figure 5.15: Example of model predictions versus actual values

This model achieves an accuracy score of 0.981 $\left(\frac{980 + 1}{1000}\right)$, which is quite high. But if this model is used to predict whether a person has a disease, it will only predict correctly in a single case. It incorrectly predicted in nine cases that these people are not sick while they actually have the given disease. At the same time, it incorrectly predicted sickness for 10 people who were actually healthy. This model's performance, then, is clearly unsatisfactory. Unfortunately, the accuracy score is simply an overall score, and it doesn't tell you where the model is performing badly.

Precision, Recall, and the F1 Score

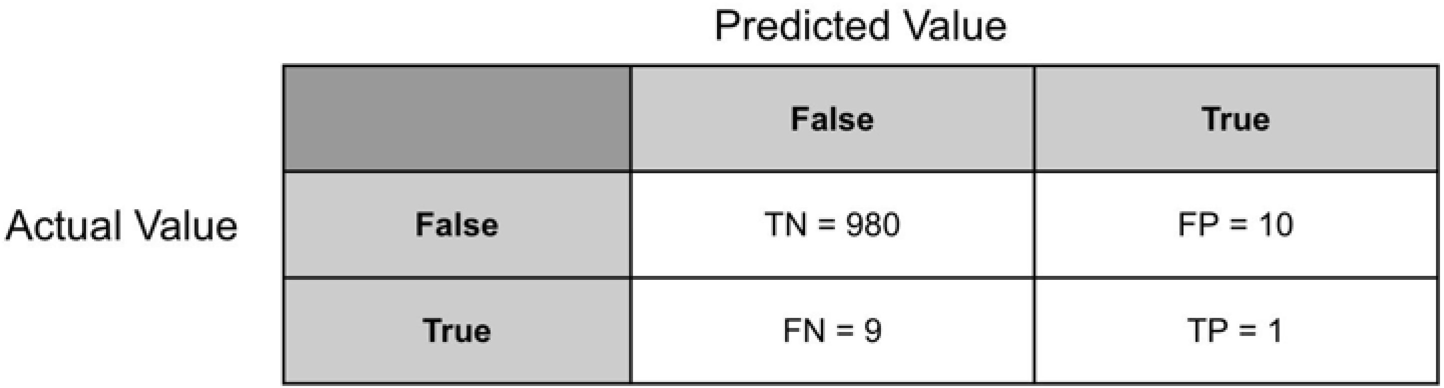
- Taking the same example as Figure 5.14, you will get the following:



- TP = 1
- TN = 980
- FP = 10
- FN = 9

Precision, Recall, and the F1 Score

- This is seen in the following table:



		Predicted Value	
		False	**True**
Actual Value	**False**	TN = 980	FP = 10
	True	FN = 9	TP = 1

Figure 5.16: Example of TP, TN, FP, and FN

Precision, Recall, and the F1 Score

- The precision score is a metric that assesses whether a model has predicted a lot of FPs. Its formula is as follows:

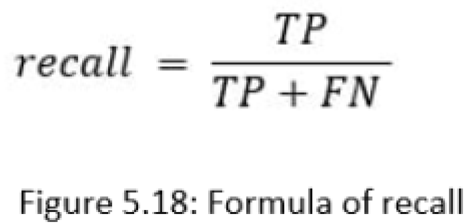
$$precision = \frac{TP}{TP + FP}$$

Figure 5.17: Formula of precision

In the preceding example, the precision score will be $\frac{1}{1+10} = 0.09$. You can see this model is making a lot of mistakes and has predicted a lot of FPs compared to the actual TP.

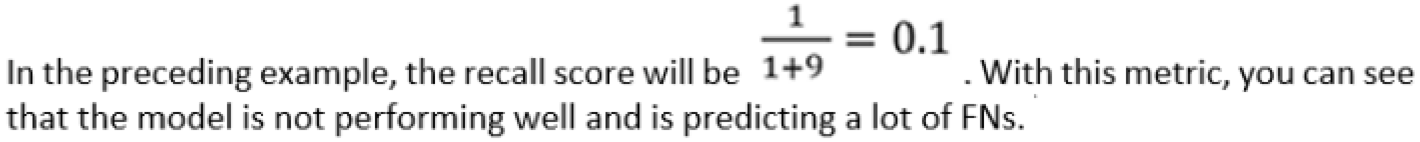
Precision, Recall, and the F1 Score

- Recall is used to assess the number of FNs compared to TPs. Its formula is as follows:



$$recall = \frac{TP}{TP + FN}$$

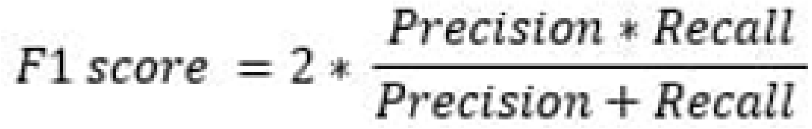
Figure 5.18: Formula of recall



In the preceding example, the recall score will be $\frac{1}{1+9} = 0.1$. With this metric, you can see that the model is not performing well and is predicting a lot of FNs.

Precision, Recall, and the F1 Score

- Finally, the F1 score is a metric that combines both precision and recall (it is the harmonic mean of precision and recall).
- Its formula is as follows:



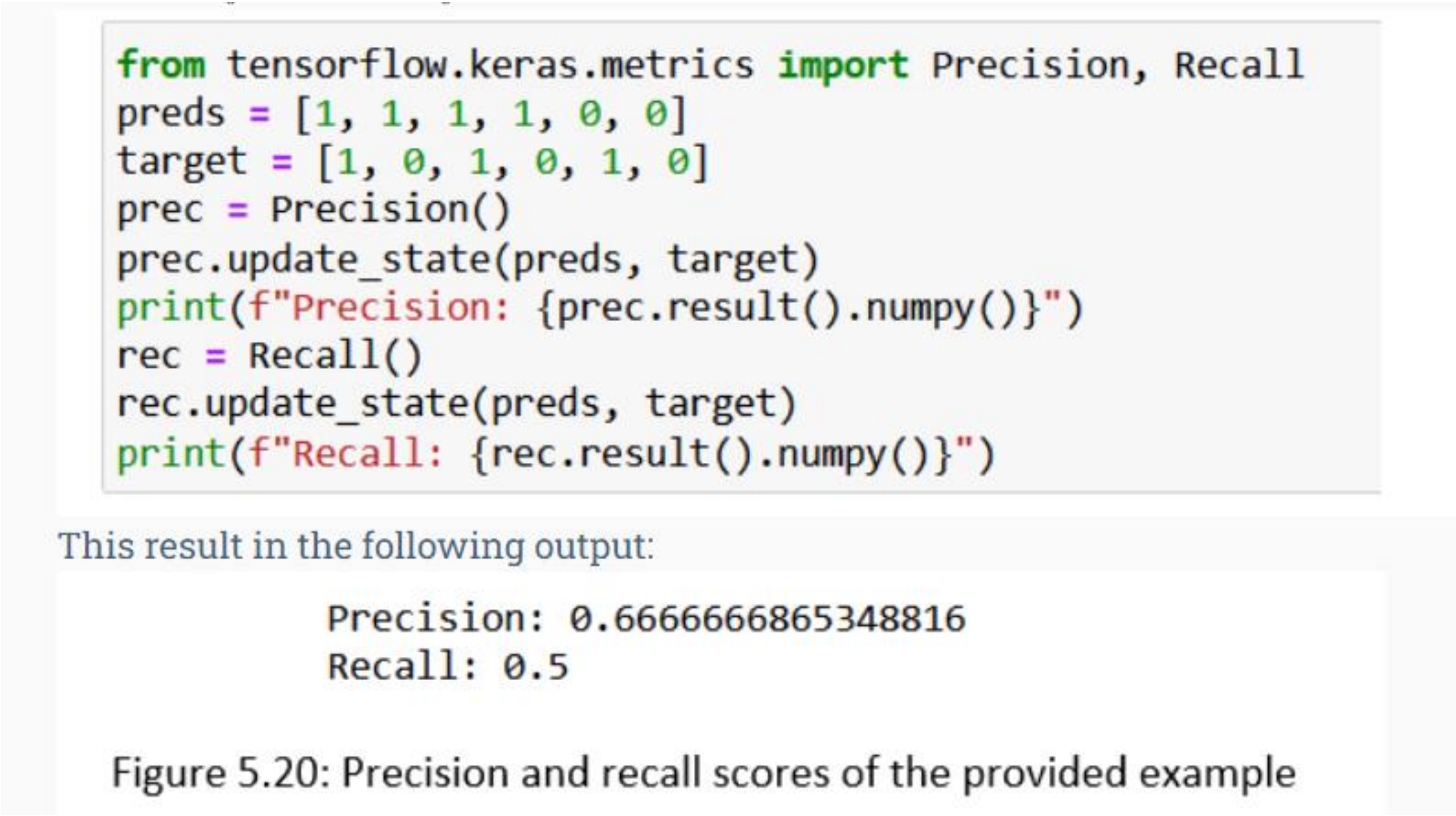
$$F1\ score\ = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Figure 5.19: Formula for the F1 score

Precision, Recall, and the F1 Score

- Taking the same example as the preceding, the F1 score will be

$$2 \times \frac{0.09 \times 0.1}{0.09+0.1} = 2 \times \frac{0.009}{0.19} = 0.095$$



```
from tensorflow.keras.metrics import Precision, Recall
preds = [1, 1, 1, 1, 0, 0]
target = [1, 0, 1, 0, 1, 0]
prec = Precision()
prec.update_state(preds, target)
print(f"Precision: {prec.result().numpy()}")
rec = Recall()
rec.update_state(preds, target)
print(f"Recall: {rec.result().numpy()}")
```

This result in the following output:

```
Precision: 0.6666666865348816
Recall: 0.5
```

Figure 5.20: Precision and recall scores of the provided example

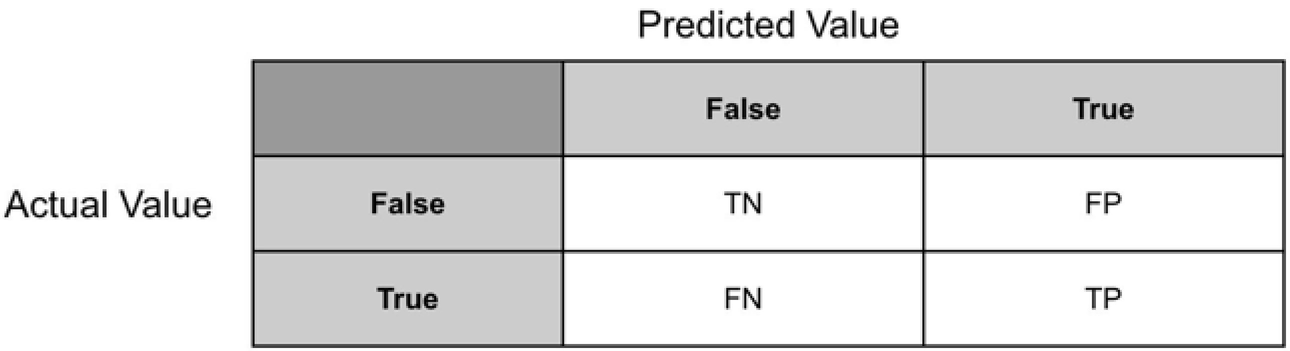
Confusion Matrices

- A confusion matrix is not a performance metric per se, but more a graphical tool used to visualize the predictions of a model against the actual values.

- You have actually already seen an example of this in the previous section.

Confusion Matrices

- For a binary classification, the confusion matrix will look like the following:

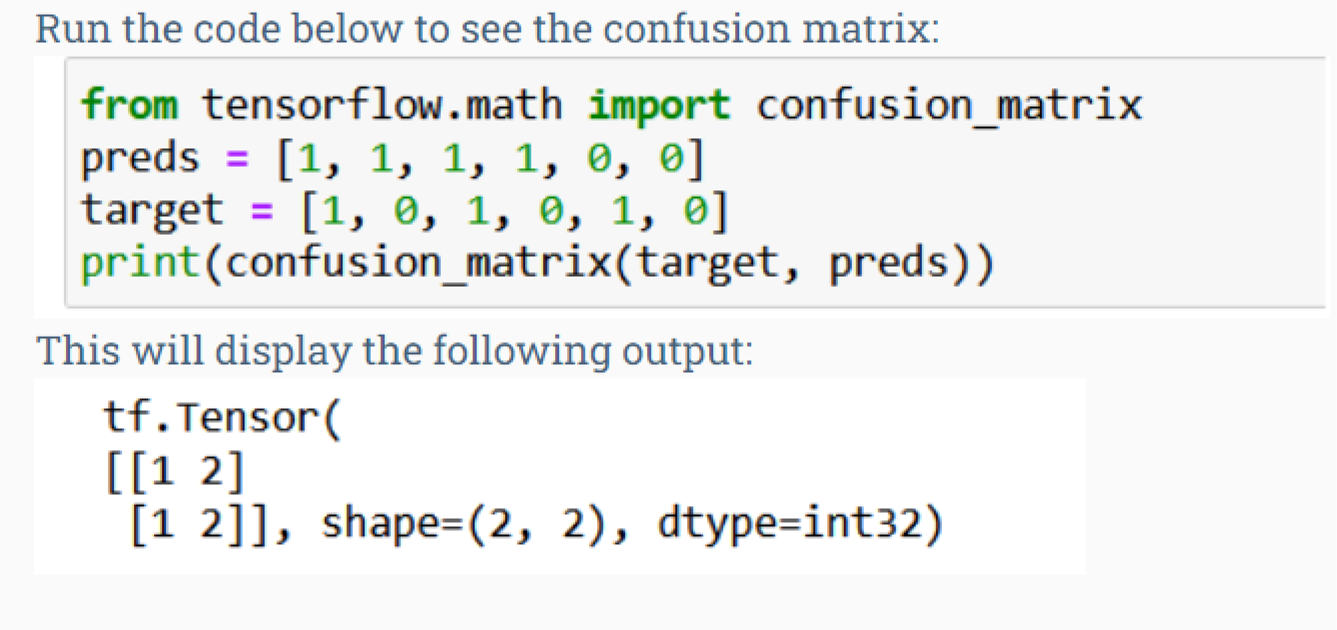


Predicted Value

		False	True
Actual Value	False	TN	FP
	True	FN	TP

Figure 5.21: Confusion matrix for a binary classification

Confusion Matrices



Run the code below to see the confusion matrix:

```
from tensorflow.math import confusion_matrix
preds = [1, 1, 1, 1, 0, 0]
target = [1, 0, 1, 0, 1, 0]
print(confusion_matrix(target, preds))
```

This will display the following output:

```
tf.Tensor(
[[1 2]
 [1 2]], shape=(2, 2), dtype=int32)
```

"Complete Exercise"

“Classification Evaluation Metrics”

Multi-Class Classification

- With binary classification, you were limited to dealing with target variables that can only take two possible values: 0 and 1 (false or true).

- Multi-class classification can be seen as an extension of this and allows the target variable to have more than two values (or you can say binary classification is just a subset of multi-class classification)..

The Softmax Function

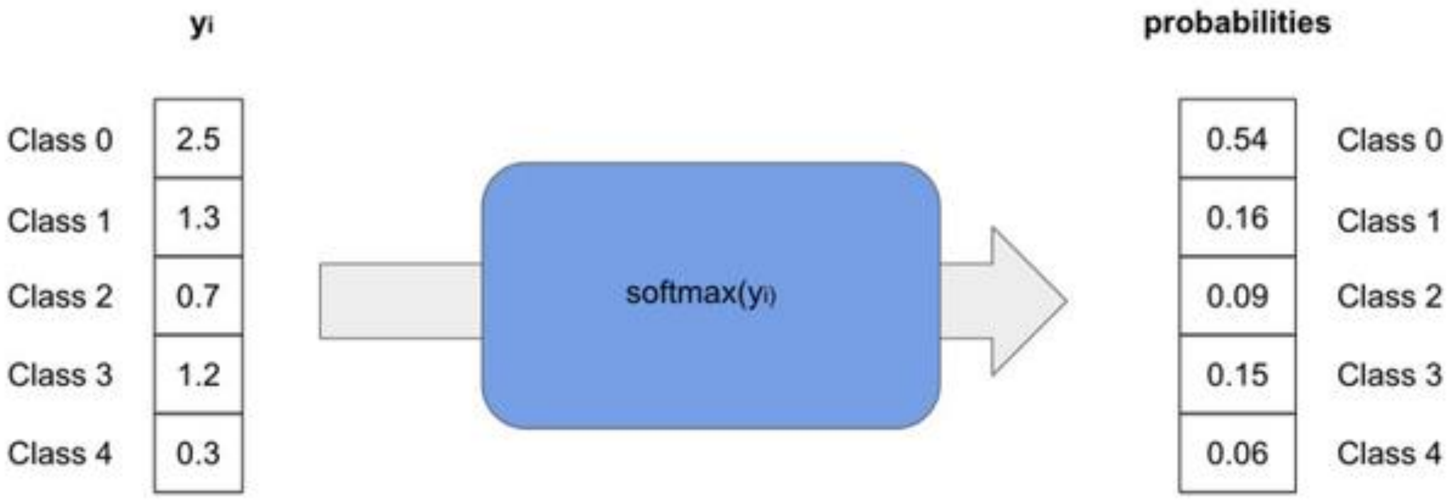
$$softmax(y_i) = \frac{e^{y_i}}{\Sigma_j e^{y_j}}$$

Figure 5.24: Formula of softmax function

y_i corresponds to the predicted value for class i.

y_j corresponds to the predicted value for class j.

The Softmax Function



	y_i
Class 0	2.5
Class 1	1.3
Class 2	0.7
Class 3	1.2
Class 4	0.3

probabilities	
0.54	Class 0
0.16	Class 1
0.09	Class 2
0.15	Class 3
0.06	Class 4

Categorical Cross-Entropy

$$L(y,\hat{y}) =- \frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{M}(y_{ij} * log(\hat{y}_{ij}))$$

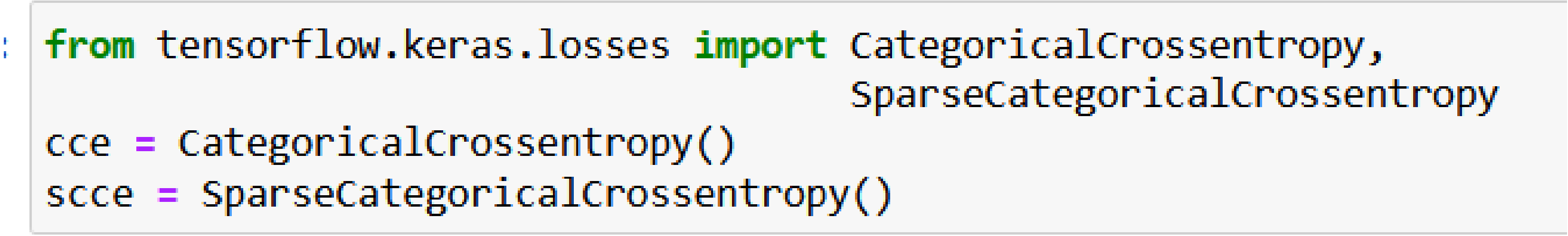
Figure 5.25: Formula of categorical cross-entropy

y_{ij} represents the probability of the actual value for the observation i to be of class j.

$\widehat{y_\iota}$ represents the predicted probability for the observation i to be of class j.

Categorical Cross-Entropy

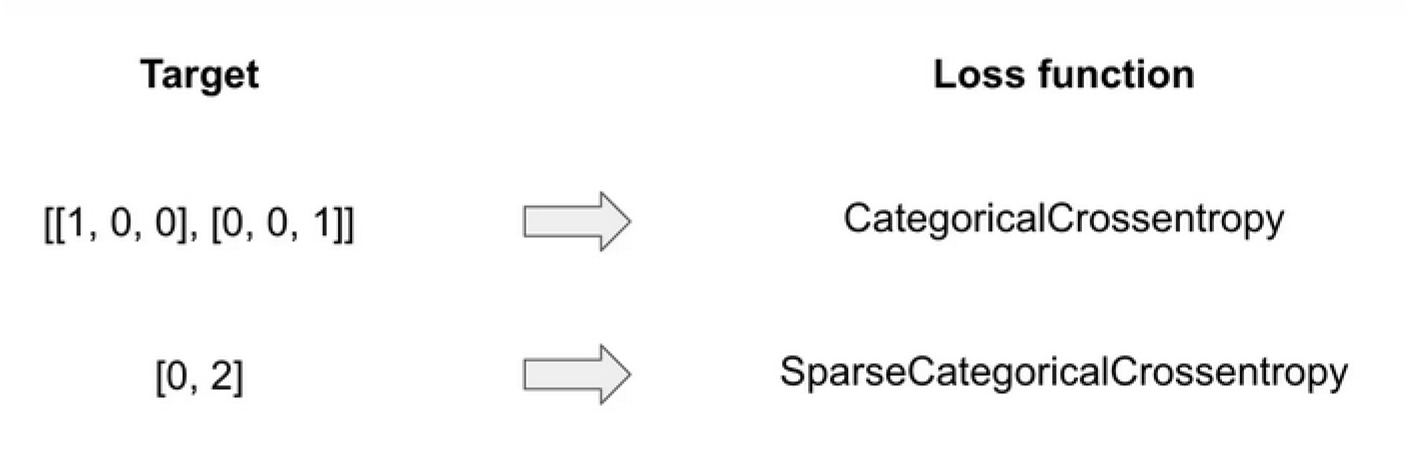
- TensorFlow provides two different classes for this loss function: CategoricalCrossentropy() and SparseCategoricalCrossentropy():



```
from tensorflow.keras.losses import CategoricalCrossentropy,
                                    SparseCategoricalCrossentropy
cce = CategoricalCrossentropy()
scce = SparseCategoricalCrossentropy()
```

Categorical Cross-Entropy

- On the other hand, if the response variable is stored as integers for representing the actual classes, you will have to use SparseCategoricalCrossentropy():

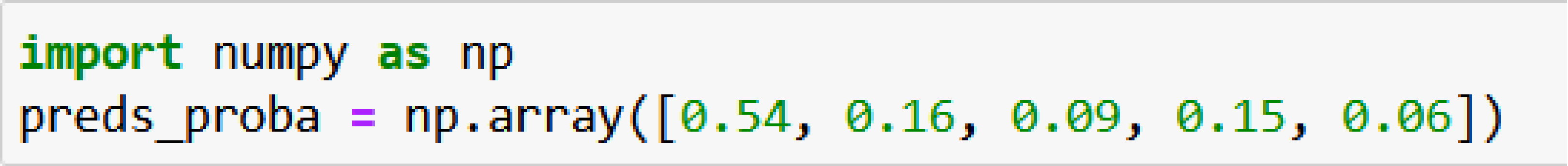


Target	Loss function
[[1, 0, 0], [0, 0, 1]]	CategoricalCrossentropy
[0, 2]	SparseCategoricalCrossentropy

Figure 5.26: Loss function to be used depending on the format of the target variable

Categorical Cross-Entropy

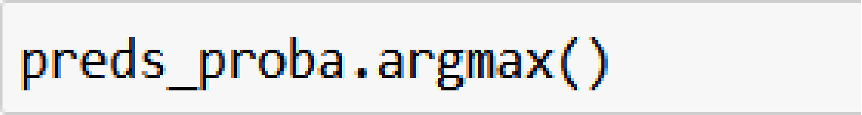
- The output of a multi-class model will be a vector containing probabilities for each class of the target variable, such as the following:



```
import numpy as np
preds_proba = np.array([0.54, 0.16, 0.09, 0.15, 0.06])
```

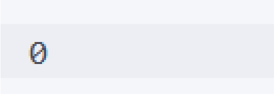
Categorical Cross-Entropy

- In order to get the final prediction (that is, the class with the highest probability), you need to use the argmax() function, which will look at all the values from a vector, find the maximum one, and return the index associated with it:



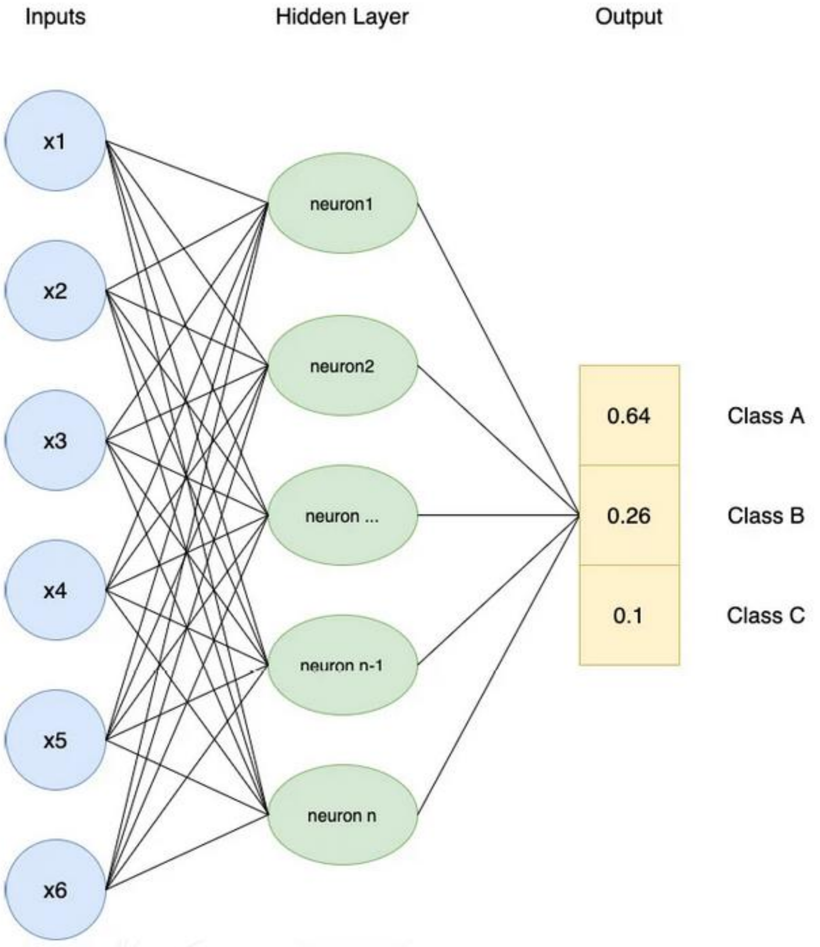
```
preds_proba.argmax()
```

- This will display the following output:



```
0
```

Multi-Class Classification Architecture



"Complete Exercise"

"Building a Multi-Class Model"

"Complete Activity"

“Building a Multi-Class with TensorFlow”

"Complete Activity"

“Building a Handwritten Digit Classification model with TensorFlow”

Multi-Label Classification

- Multi-label classification is another type of classification where you predict not only one target variable as in binary or multi-class classification, but several response variables at the same time.

- For instance, you can predict multiple outputs for the different objects present in an image (for instance, a model will predict whether there is a cat, a man, and a car in a given picture) or you can predict multiple topics for an article (such as whether the article is about the economy, international news, and manufacturing).

Multi-Label Classification

- The sigmoid function will predict the probability of occurrence for each target variable:

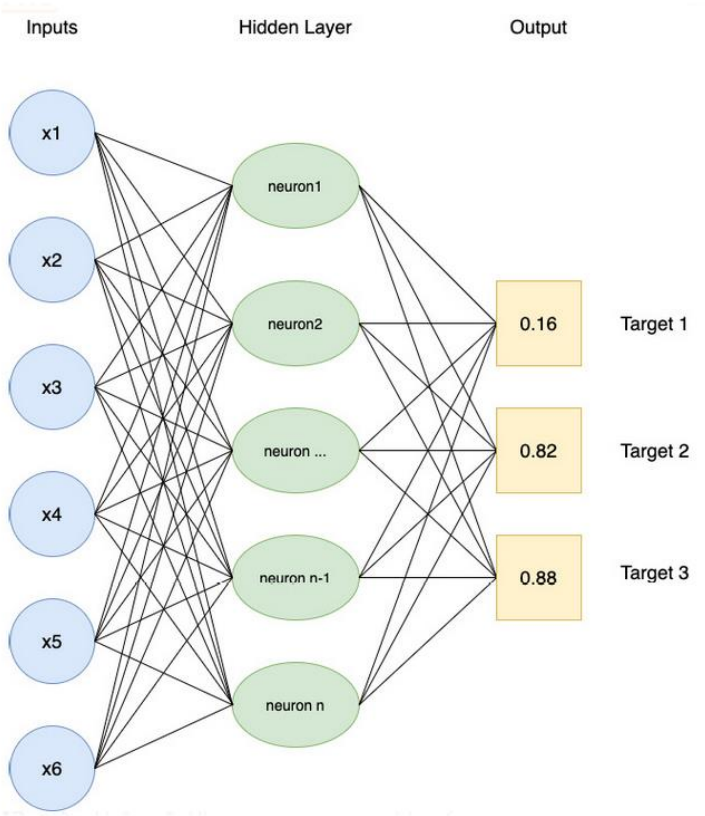
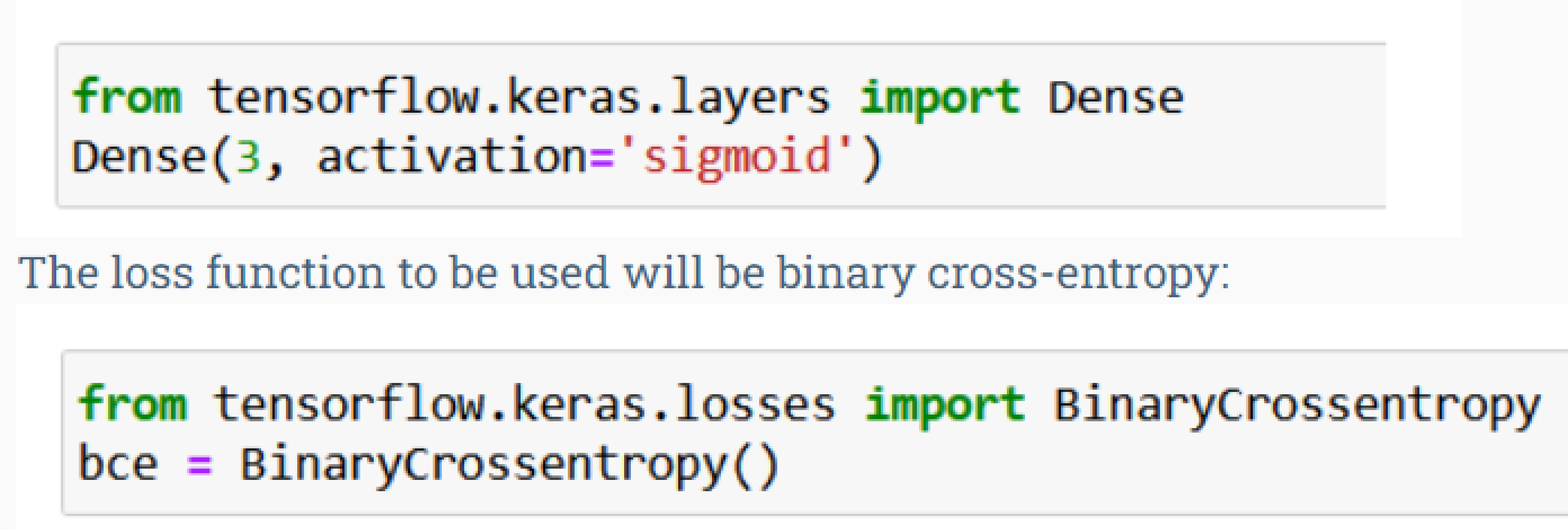


Figure 5.40: Architecture of the multi-label classifier

Multi-Label Classification



```
from tensorflow.keras.layers import Dense
Dense(3, activation='sigmoid')
```

The loss function to be used will be binary cross-entropy:

```
from tensorflow.keras.losses import BinaryCrossentropy
bce = BinaryCrossentropy()
```

Summary

- You started your journey in this lesson with an introduction to classification models and their differences compared with regression models.

- You learned that the target variable for classifiers can only contain a limited number of possible values.

- You then explored binary classification, wherein the response variable can only be from two possible values: 0 or 1.

6. Regularization and Hyperparameter Tuning

Introduction

- When evaluating a model, you will face three different situations: model overfitting, model underfitting, and model performing.
- The last one is the ideal scenario, in which a model is accurately predicting the right outcome and is generalizing to unseen data well.
- If a model is underfitting, it means it is neither achieving satisfactory performance nor accurately predicting the target variable.

Regularization Techniques

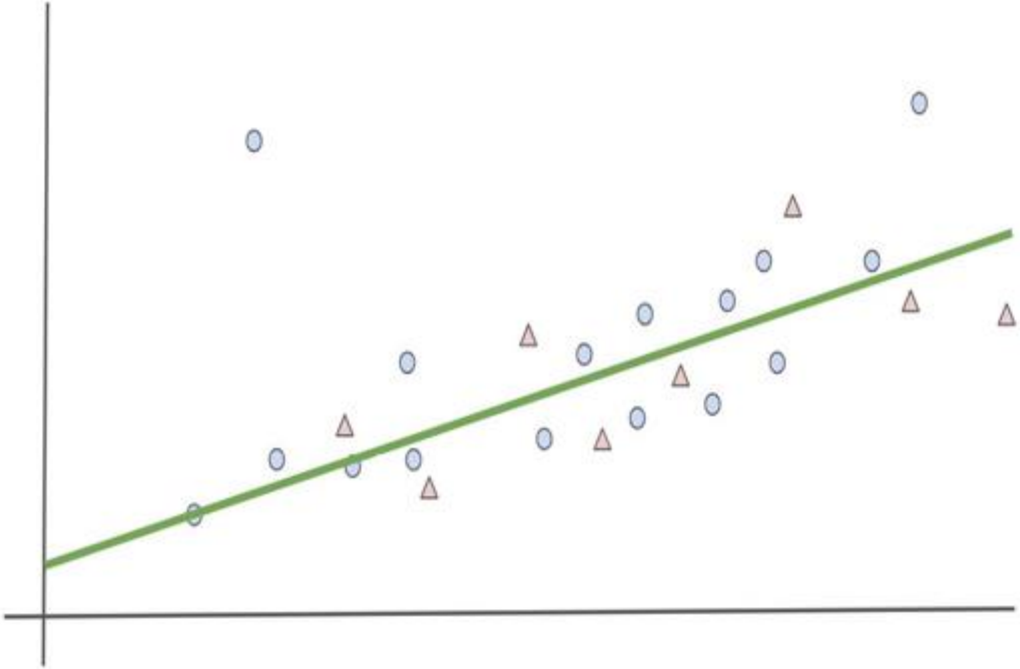
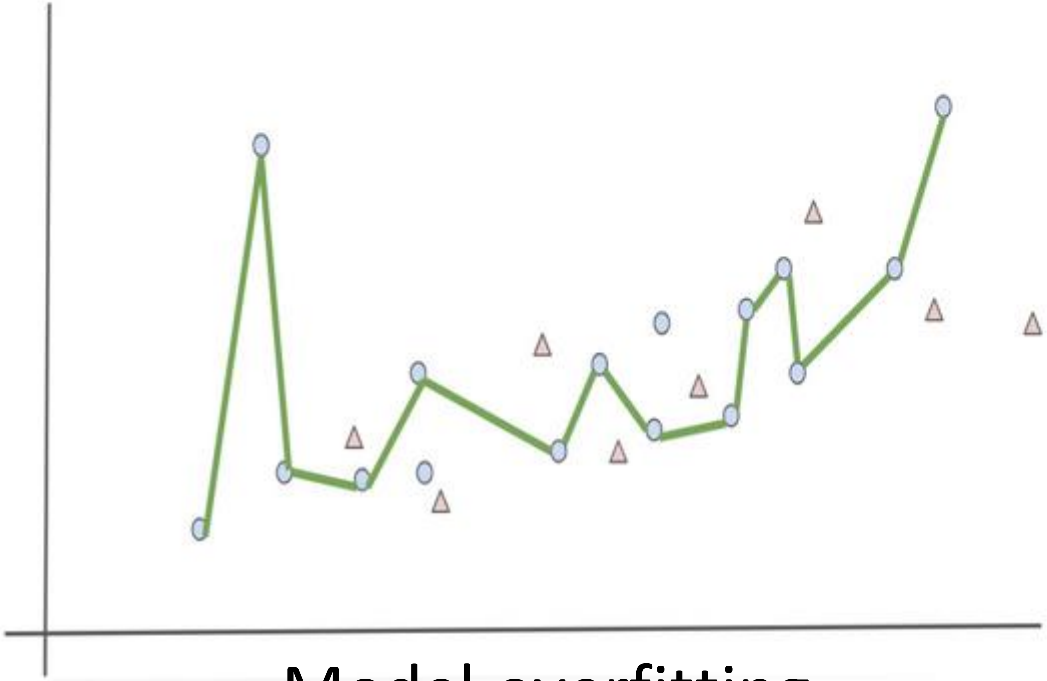


Figure 6.1: Model not overfitting or underfitting

Regularization Techniques



Model overfitting

L1 Regularization

- For deep learning models, overfitting happens when some of the features have higher weights than they should.

- The model puts too much emphasis on these features as it believes they are extremely important for predicting the training set.

- Unfortunately, these features are less relevant for the test set or any new unseen data.

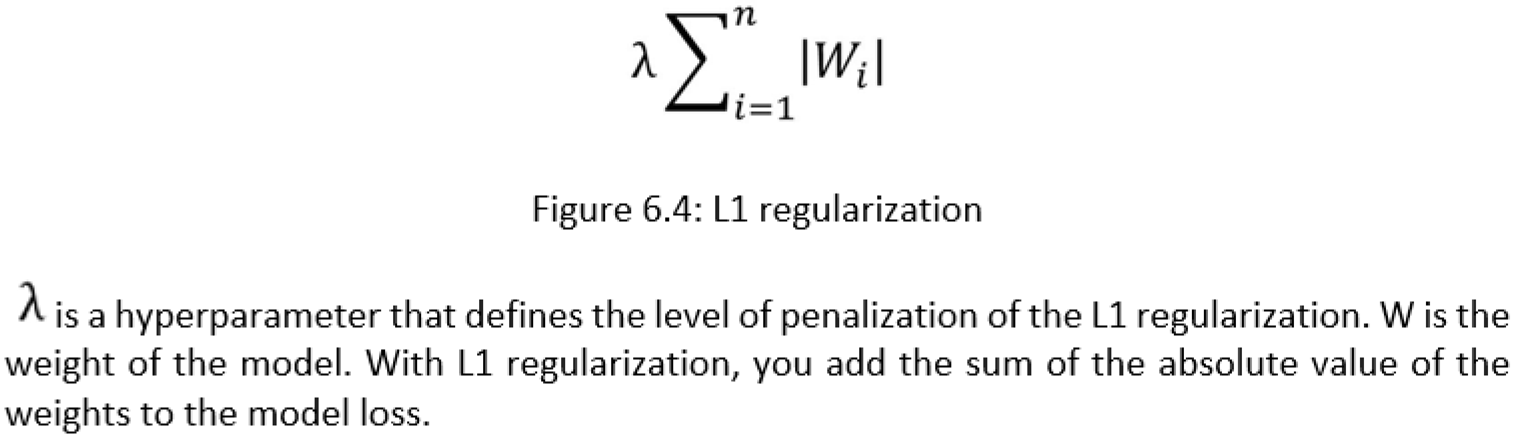
L1 Regularization

- There are multiple ways to perform regularization. One of them is to add a regularization component to the cost function:

$$cost\ function = loss + regularization$$

L1 Regularization

- One very popular regularization component is L1. Its formula is as follows:



$$\lambda \sum_{i=1}^{n} |W_i|$$

Figure 6.4: L1 regularization

λ is a hyperparameter that defines the level of penalization of the L1 regularization. W is the weight of the model. With L1 regularization, you add the sum of the absolute value of the weights to the model loss.

L1 Regularization

- In TensorFlow, you can define L1 regularization with the following code snippet:

```
from tensorflow.keras.regularizers import l1

l1_reg = l1(l=0.01)
```

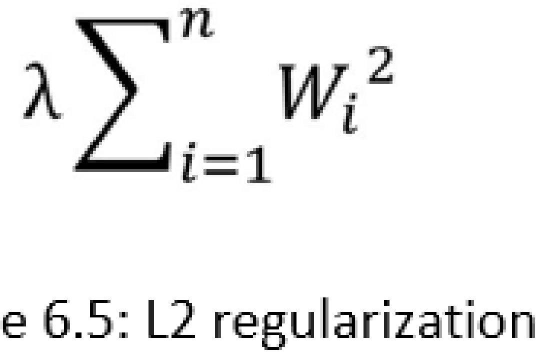
L1 Regularization

The l parameter corresponds to the λ hyperparameter. The instantiated L1 regularization can then be added to any layer from TensorFlow Keras:

```
from tensorflow.keras.layers import Dense

Dense(10, kernel_regularizer=l1_reg)
```

L2 Regularization

- L2 regularization is similar to L1 in that it adds a regularization component to the cost function, but its formula is different:

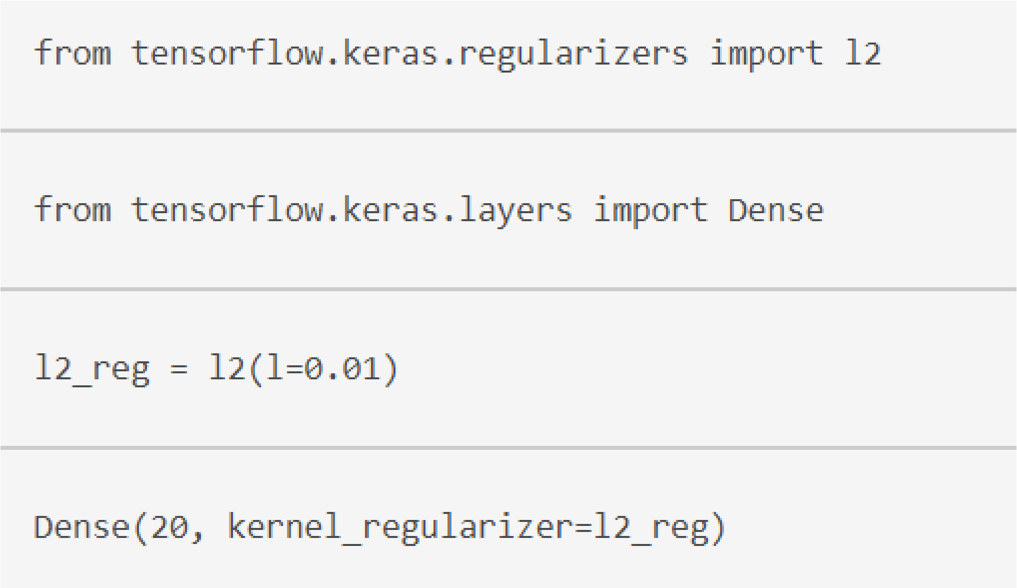


$$\lambda \sum_{i=1}^{n} W_i^{\,2}$$

Figure 6.5: L2 regularization

L2 Regularization

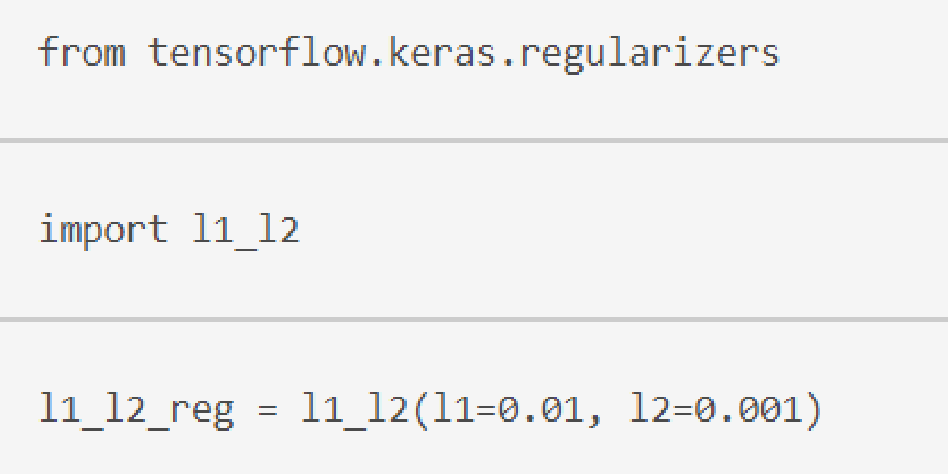
- In TensorFlow, you can define L2 regularization as follows:



```
from tensorflow.keras.regularizers import l2

from tensorflow.keras.layers import Dense

l2_reg = l2(l=0.01)

Dense(20, kernel_regularizer=l2_reg)
```

L2 Regularization

- TensorFlow provides another regularizer class that combines both L1 and L2 regularizers.
- You can instantiate it with the following code snippet:



```
from tensorflow.keras.regularizers

import l1_l2

l1_l2_reg = l1_l2(l1=0.01, l2=0.001)
```

"Complete Exercise"

“Predicting the radiator position of a space shuttle using L2 Regularizer”

Dropout Regularization

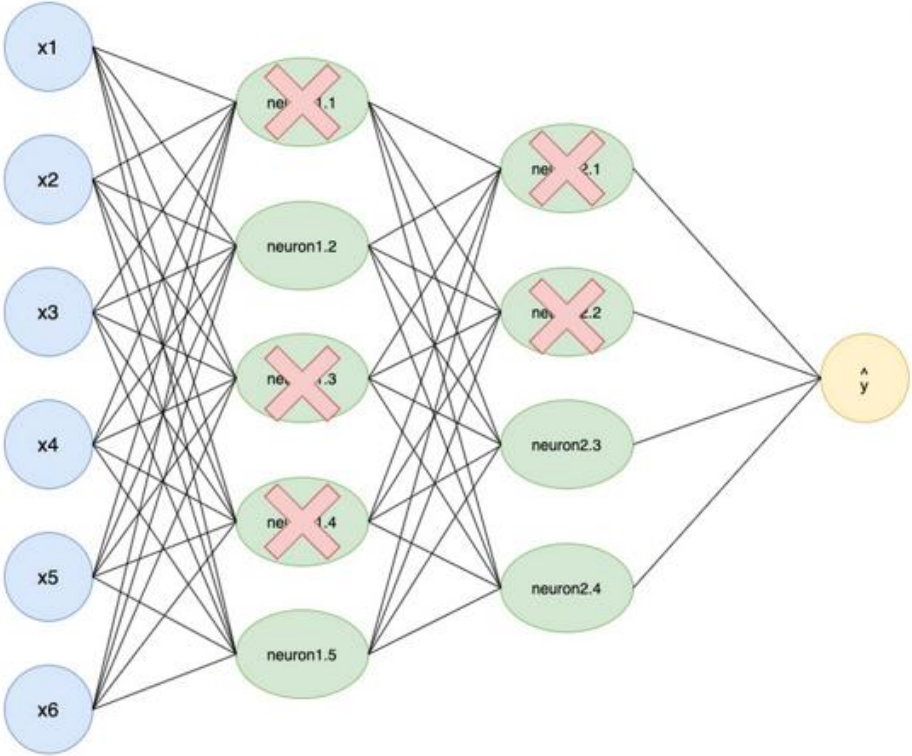


Figure 6.11: Dropout of Neural networks

Dropout Regularization

- The following code snippet shows you how to create a dropout layer of 50% in TensorFlow:

```
from tensorflow.keras.layers import Dropout

do = Dropout(0.5)
```

"Complete Exercise"

“Predicting the radiator position of a space shuttle using Dropout”

Early Stopping

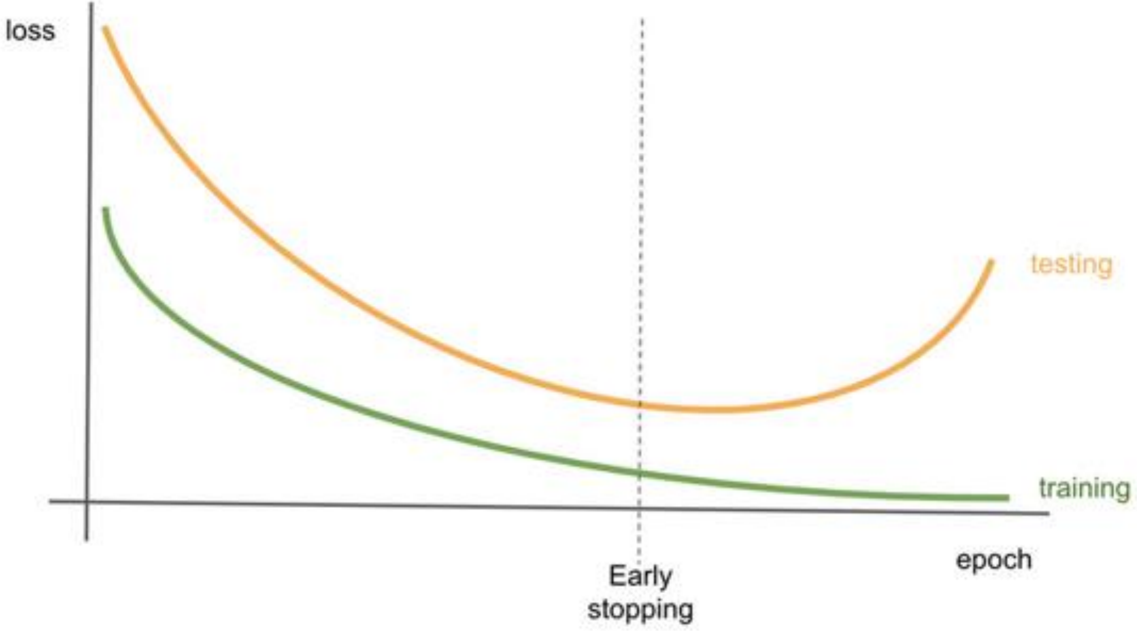


Figure 6.15: Early stopping to prevent overfitting

Early Stopping

- In TensorFlow, you can achieve this by setting up callbacks that analyze the performance of the models at each epoch and compare its score between the training and test sets.

- To define an early stopping callback, you will do the following:

```
from tensorflow.keras.callbacks import EarlyStopping
EarlyStopping(monitor='val_accuracy',patience=5)
```

"Complete Activity"

“Predicting Motor Failure with L1 and L2 Regularizers”

Hyperparameter Tuning

- With neural networks, data scientists have access to different hyperparameters they can tune to improve the performance of a model.

- For example, you can try different learning rates and see whether one leads to better results, you can try different numbers of units for each hidden layer of a network, or you can test to see whether different ratios of dropout can achieve a better trade-off between overfitting and underfitting.

Keras Tuner

Hyperparameters are the classes used to define a parameter that will be assessed by the tuner. You can use different types of hyperparameters. The main ones are the following:

- hp.Boolean: A choice between True and False
- hp.Int: A choice with a range of integers
- hp.Float: A choice with a range of decimals
- hp.Choice: A choice within a list of possible values

Keras Tuner

- The following code snippet shows you how to define a hyperparameter called learning_rate that can only take one of four values—0.1, 0.01, 0.001, or 0.0001:

```
hp.Choice('learning_rate', values = [0.1, 0.01, 0.001, 0.0001])
```

Keras Tuner

- Once defined with the algorithm of your choice, you can call the search() method to start the hyperparameter tuning process on the training and test sets, as follows:

```
tuner.search(X_train, y_train, validation_data=(X_test,
y_test))
```

Keras Tuner

- Once the search is complete, you can access the best combination with

get_best_hyperparameters()

- and then look specifically at one of the hyperparameters you defined:

```
best_hps = tuner.get_best_hyperparameters()[0]
best_hps.get('learning_rate')
```

Keras Tuner

- Finally, the hypermodel.build() method will instantiate a TensorFlow Keras model with the best hyperparameters found:

```
model = tuner.hypermodel.build(best_hps)
```

Random Search

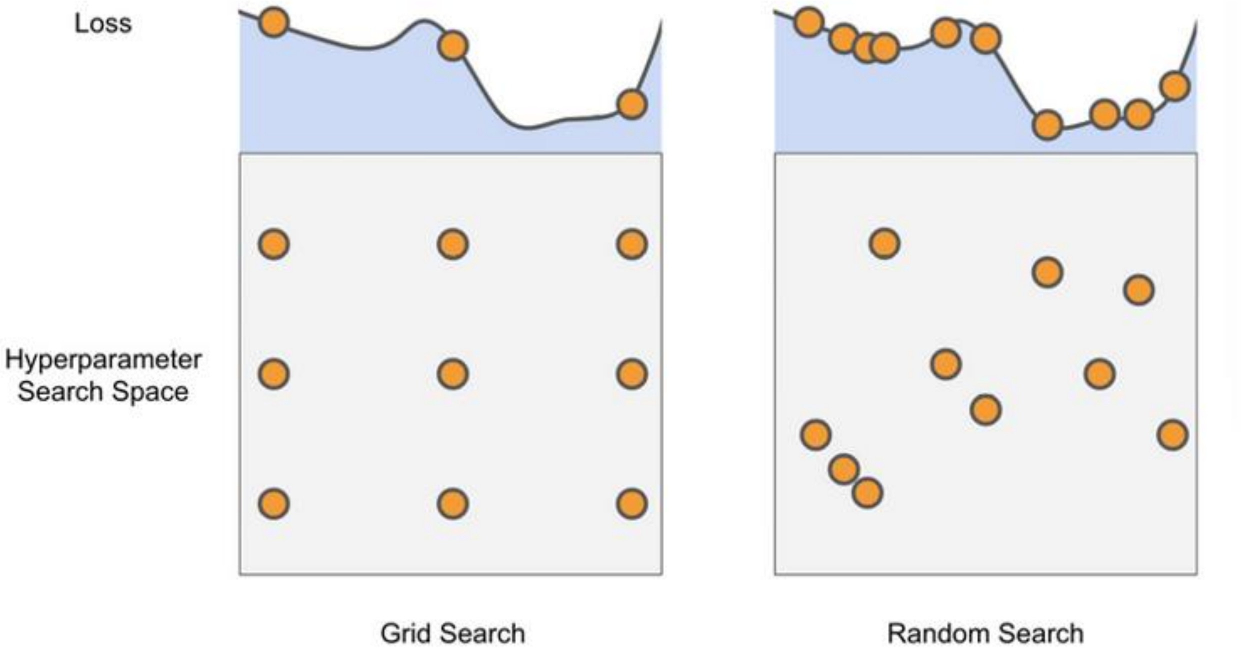
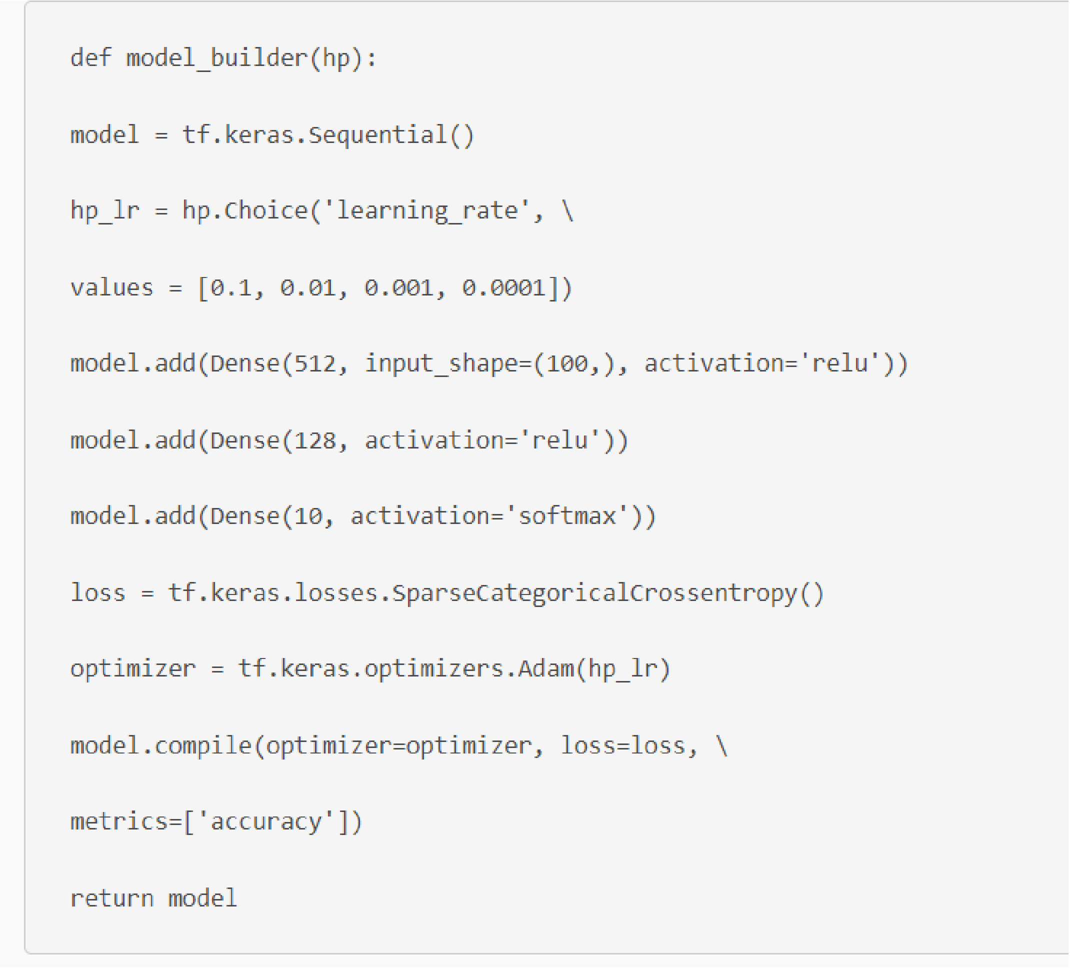


Figure 6.17: Comparison between grid search and random search

Random Search



```
def model_builder(hp):
model = tf.keras.Sequential()
hp_lr = hp.Choice('learning_rate', \
values = [0.1, 0.01, 0.001, 0.0001])
model.add(Dense(512, input_shape=(100,), activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
loss = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam(hp_lr)
model.compile(optimizer=optimizer, loss=loss, \
metrics=['accuracy'])
return model
```

Random Search

- Once the model-building function is defined, you can instantiate a random search tuner like the following:

```
import kerastuner as kt

tuner = kt.RandomSearch(model_builder,
objective='val_accuracy', \

max_trials=10
```

"Complete Exercise"

"Predicting the radiator position of a space shuttle using Random Search from Keras Tuner"

Hyperband

- Hyperband is another tuner available in the Keras Tuner package.
- Like random search, it randomly picks candidates from the search space, but more efficiently.
- To instantiate a Hyperband tuner, execute the following command:

```
tuner = kt.Hyperband(model_builder, objective='val_accuracy', \
                     max_epochs=5)
```

"Complete Exercise"

"Predicting the radiator position of a space shuttle using Hyperband from Keras Tuner"

Bayesian Optimization

- Bayesian optimization is another very popular algorithm used for automatic hyperparameter tuning.
- It uses probabilities to determine the best combination of hyperparameters.
- The following code snippet will show you how to instantiate a Bayesian optimizer in Keras Tuner:

```
tuner = kt.BayesianOptimization(model_builder,
objective='val_accuracy', \
                                        max_trials=10)
```

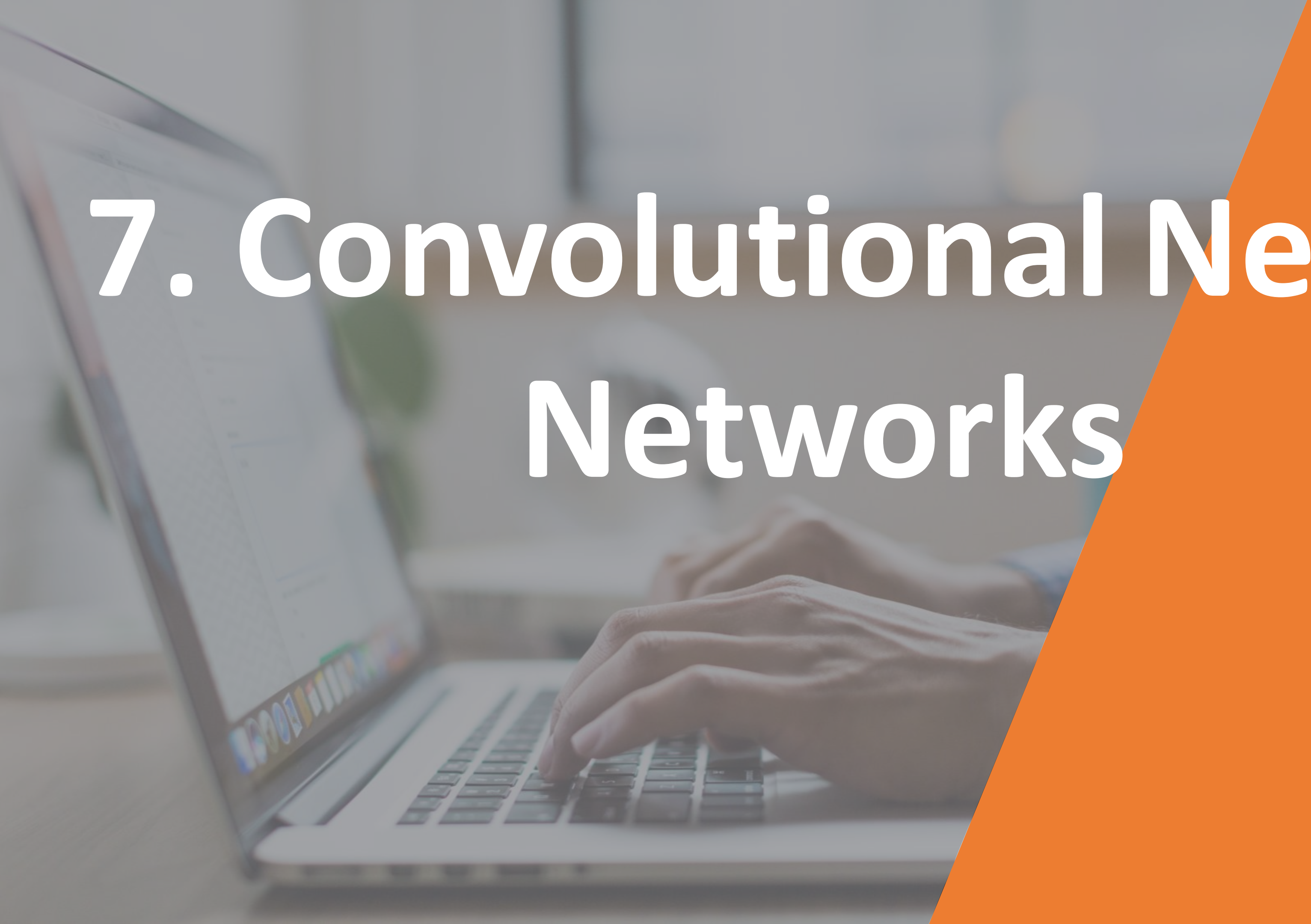
"Complete Activity"

"Predicting Motor Failure with Bayesian Optimization from Keras Tuner"

Summary

- You started your journey in this lesson with an introduction to the different scenarios of training a model.
- A model is overfitting when its performance is much better on the training set than the test set.
- An underfitting model is one that can achieve good results only after training. Finally, a good model achieves good performance on both the training and test sets.
- Then, you encountered several regularization techniques that can help prevent a model from overfitting.

7. Convolutional Neural Networks



Introduction

- In this lesson, you will learn how convolutional neural networks (CNNs) process image data.

- You will also learn how to correctly use a CNN on image data.

- By the end of the lesson, you will be able to create your own CNN for classification and object identification on any image dataset using TensorFlow.

CNNs

- CNNs share many common components with the ANNs you have built so far.
- The key difference is the inclusion of one or more convolutional layers within the network. Convolutional layers apply convolutions of input data with filters, also known as kernels.
- Think of a convolution as an image transformer, You have an input image, which goes through the CNN and gives you an output label.
- Each layer has a unique function or special ability to detect patterns such as curves or edges in an image.

CNNs

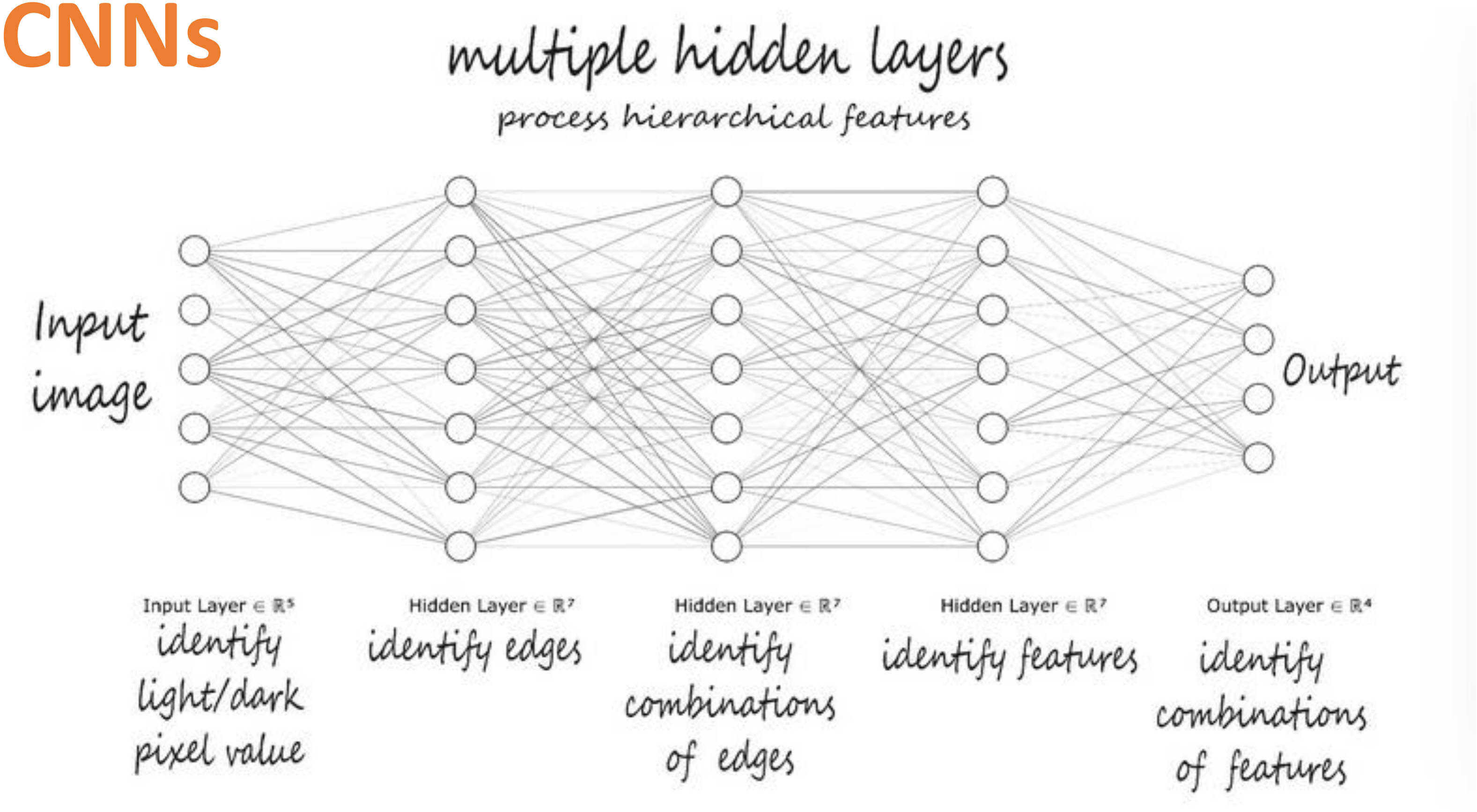
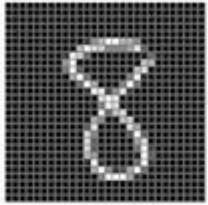
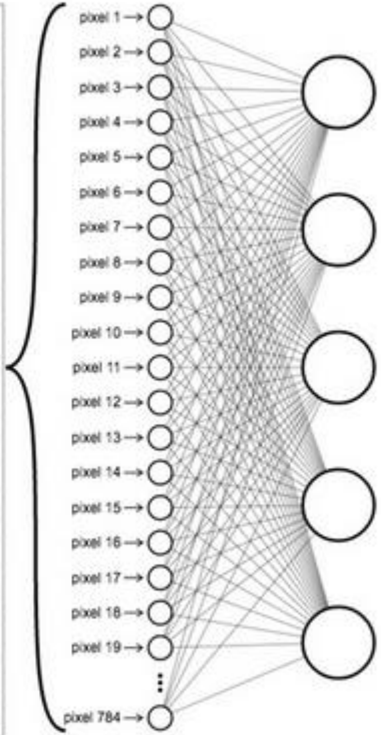

multiple hidden layers
process hierarchical features
Input image
Output
Input Layer $\in \mathbb{R}^5$
identify light/dark pixel value
Hidden Layer $\in \mathbb{R}^7$
identify edges
Hidden Layer $\in \mathbb{R}^7$
identify combinations of edges
Hidden Layer $\in \mathbb{R}^7$
identify features
Output Layer $\in \mathbb{R}^4$
identify combinations of features

Image Representation



28 x 28
784 pixels

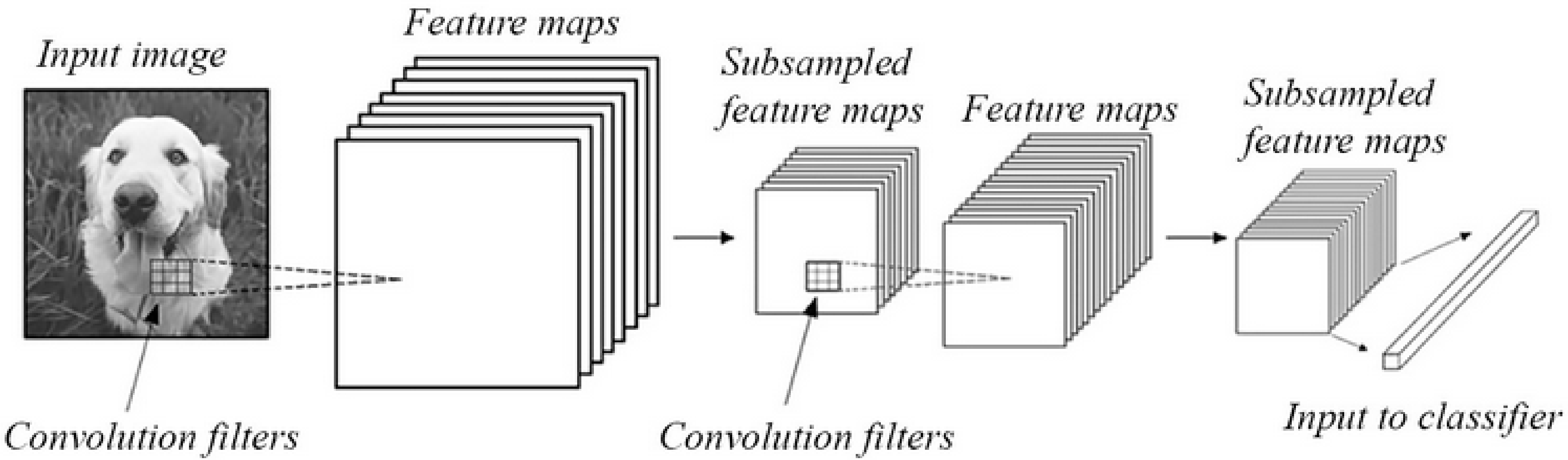
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



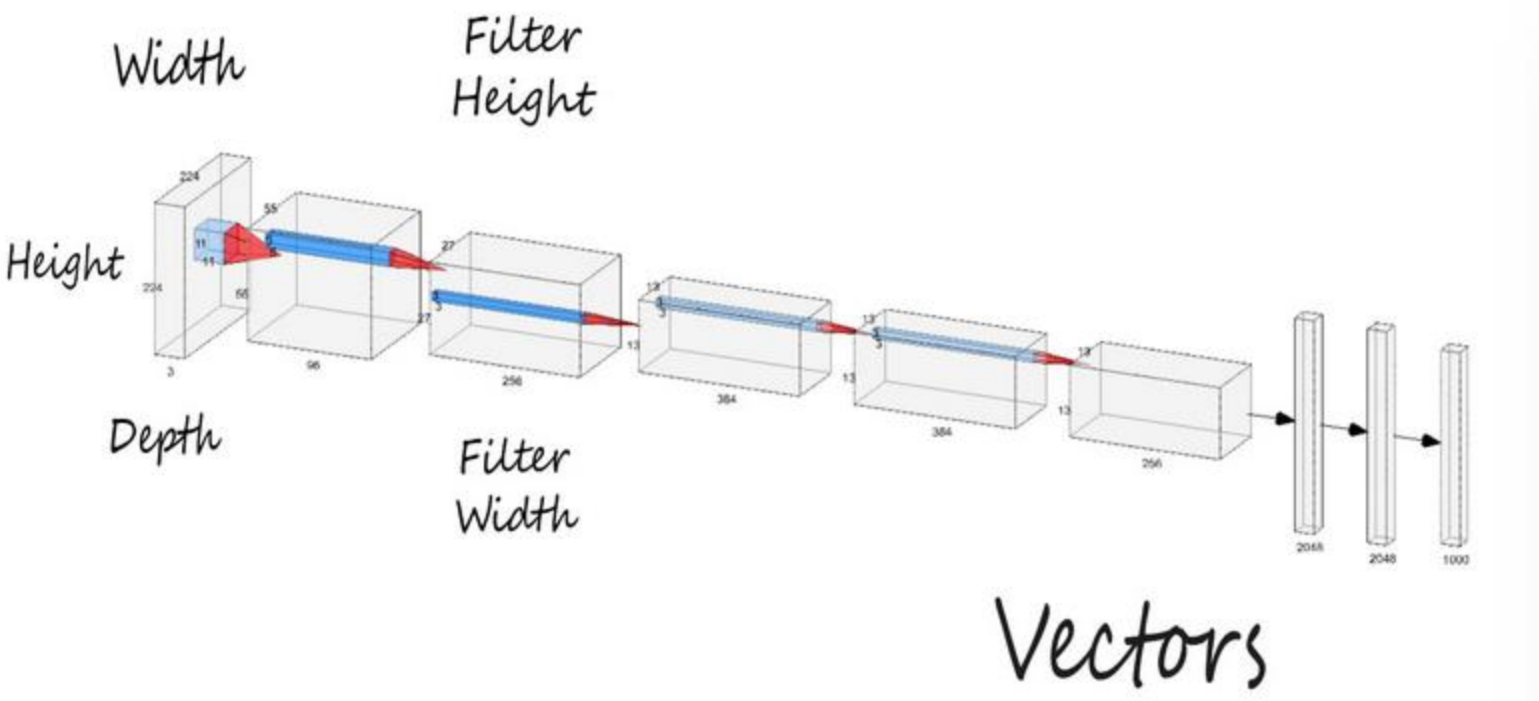
The Convolutional Layer

- Think of a convolution as nothing more than an image transformer with three key elements.
- First, there is an input image, then a filter, and finally, a feature map.
- This section will cover each of these in turn to give you a solid idea of how images are filtered in a convolutional layer.
- The convolution is the process of passing a filter window over the input data, which will result in a map of activations known as a feature map.

The Convolutional Layer



Creating the Model



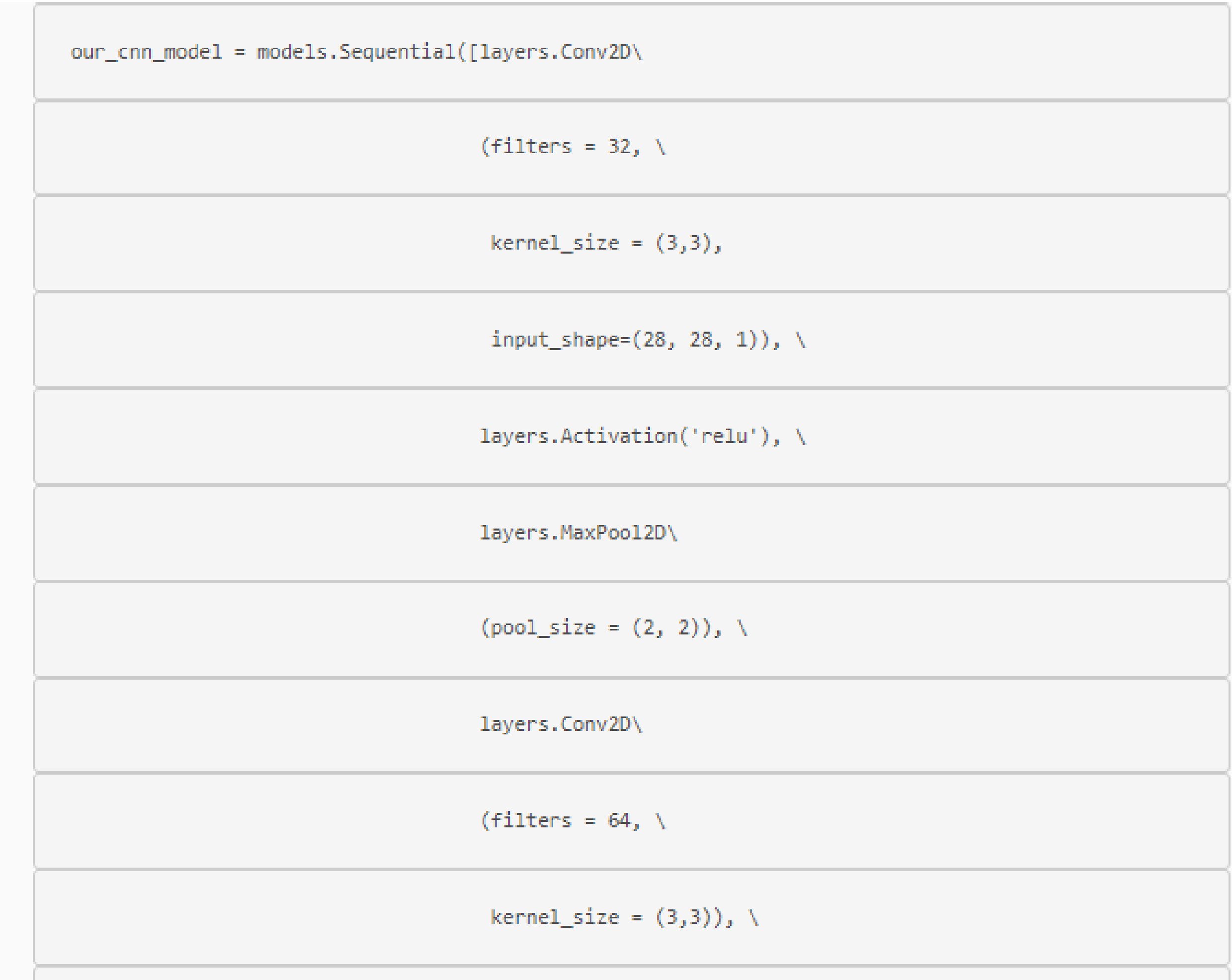
Creating the Model

- A sequential model can be used to build a CNN.

- Different methods can be used to add a layer; here, we will use the framework of sequentially adding layers to the model using the model's add method or passing in a list of all layers when the model is instantiated.

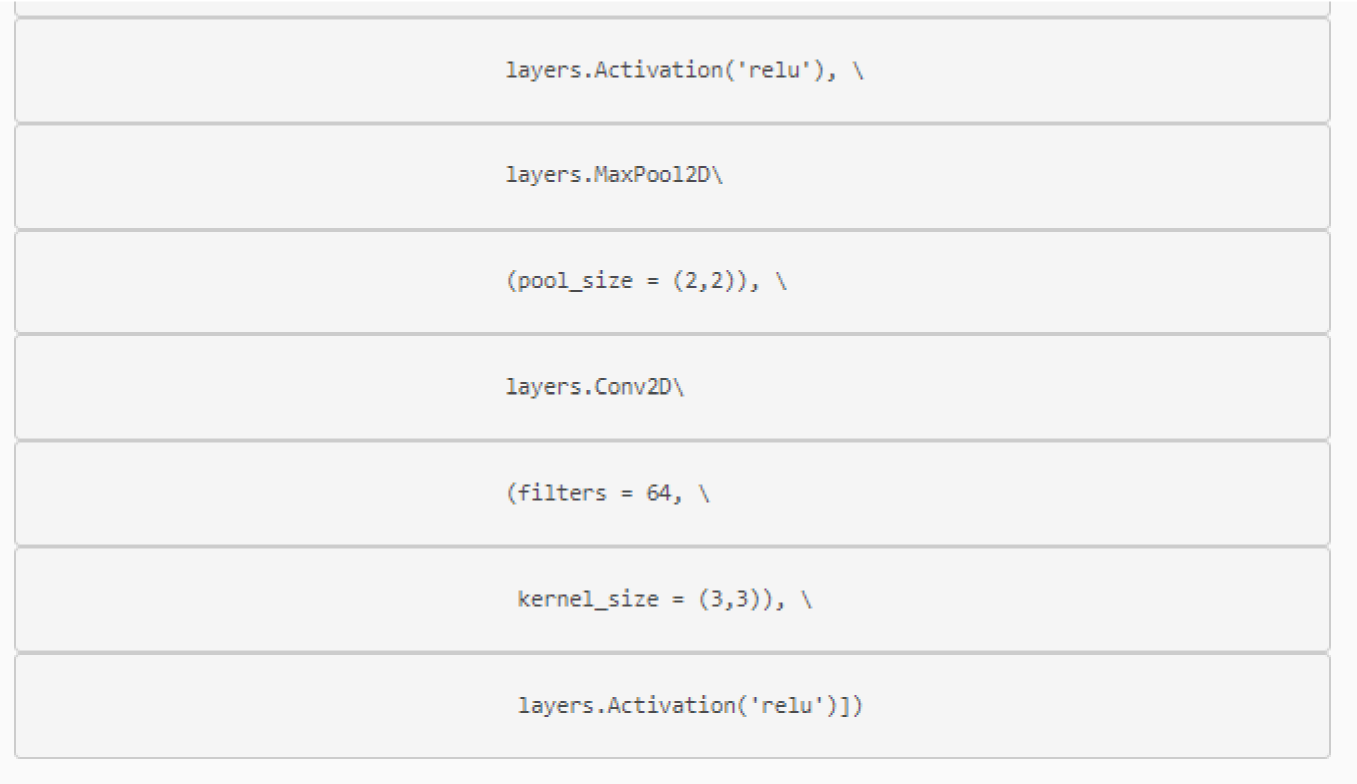
```
model = models.Sequential()
model.add(Dense(32, input_shape=(250,)))
```

Creating the Model



```
our_cnn_model = models.Sequential([layers.Conv2D\
                                   (filters = 32, \
                                    kernel_size = (3,3),
                                    input_shape=(28, 28, 1)), \
                                   layers.Activation('relu'), \
                                   layers.MaxPool2D\
                                   (pool_size = (2, 2)), \
                                   layers.Conv2D\
                                   (filters = 64, \
                                    kernel_size = (3,3)), \
```

Creating the Model



```
layers.Activation('relu'), \
layers.MaxPool2D\
(pool_size = (2,2)), \
layers.Conv2D\
(filters = 64, \
 kernel_size = (3,3)), \
 layers.Activation('relu')])
```

"Complete Exercise"

“Creating the First Layer to Build a CNN”

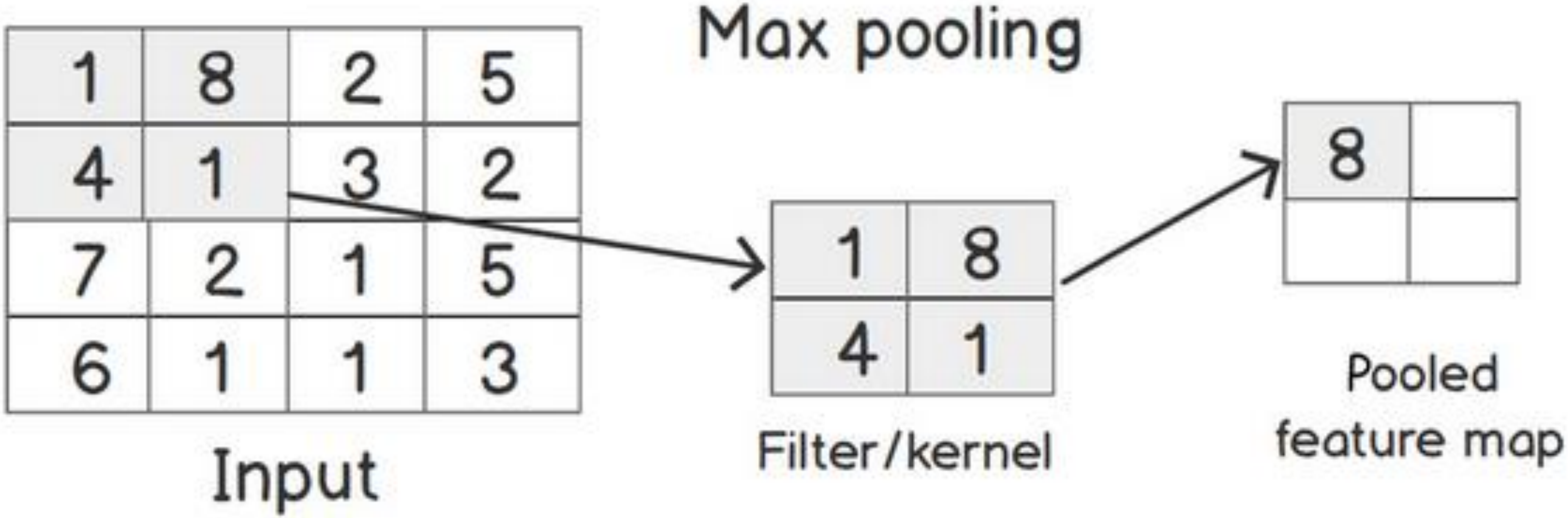
Pooling Layer

- A pooling layer immediately follows a convolution layer and is considered another important part of the CNN structure.

- This section will focus on two types of pooling:

1. Max pooling
2. Average pooling

Max Pooling



Max pooling

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

Input

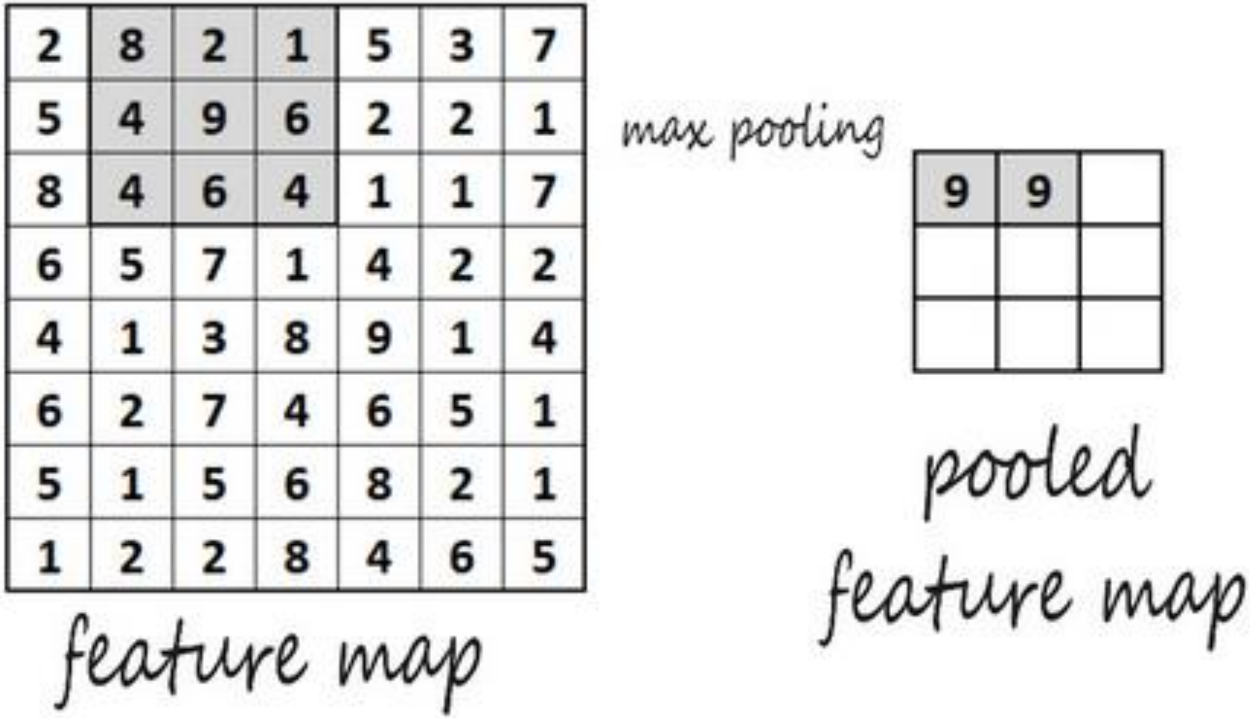
1	8
4	1

Filter/kernel

8	

Pooled feature map

Max Pooling



2	8	2	1	5	3	7
5	4	9	6	2	2	1
8	4	6	4	1	1	7
6	5	7	1	4	2	2
4	1	3	8	9	1	4
6	2	7	4	6	5	1
5	1	5	6	8	2	1
1	2	2	8	4	6	5

9	9	

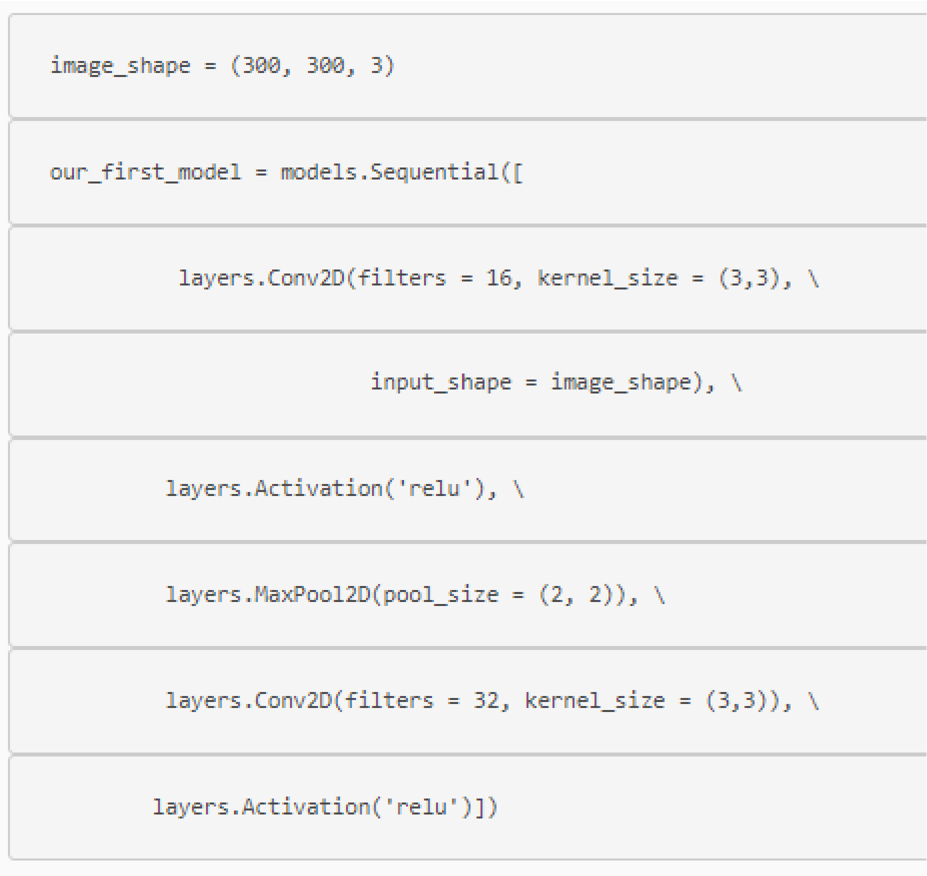
Max Pooling

- The preceding pool size is (2, 2). It specifies factors that you will downscale with.

- Here's a more detailed look at what you could do to implement MaxPool2D:

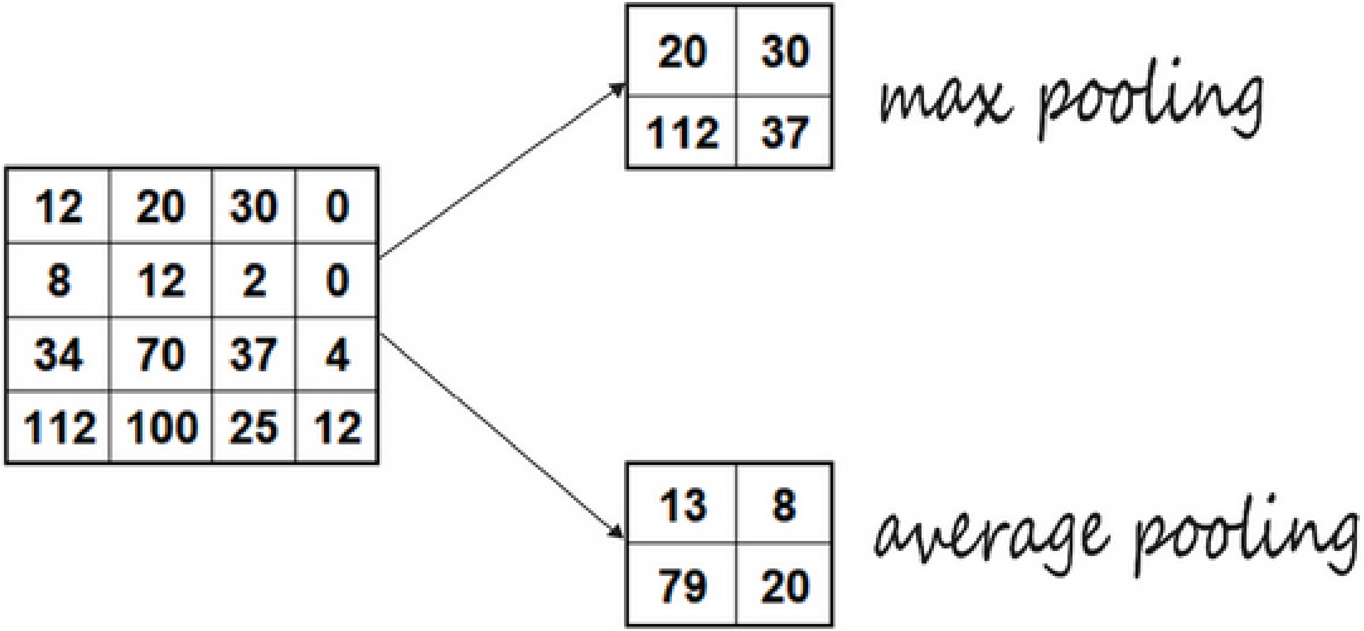
```
layers.MaxPool2D(pool_size=(2, 2), strides=None, \
            padding='valid')
```

Max Pooling



```
image_shape = (300, 300, 3)

our_first_model = models.Sequential([

          layers.Conv2D(filters = 16, kernel_size = (3,3), \

                         input_shape = image_shape), \

         layers.Activation('relu'), \

         layers.MaxPool2D(pool_size = (2, 2)), \

         layers.Conv2D(filters = 32, kernel_size = (3,3)), \

        layers.Activation('relu')])
```

Average Pooling



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

13	8
79	20

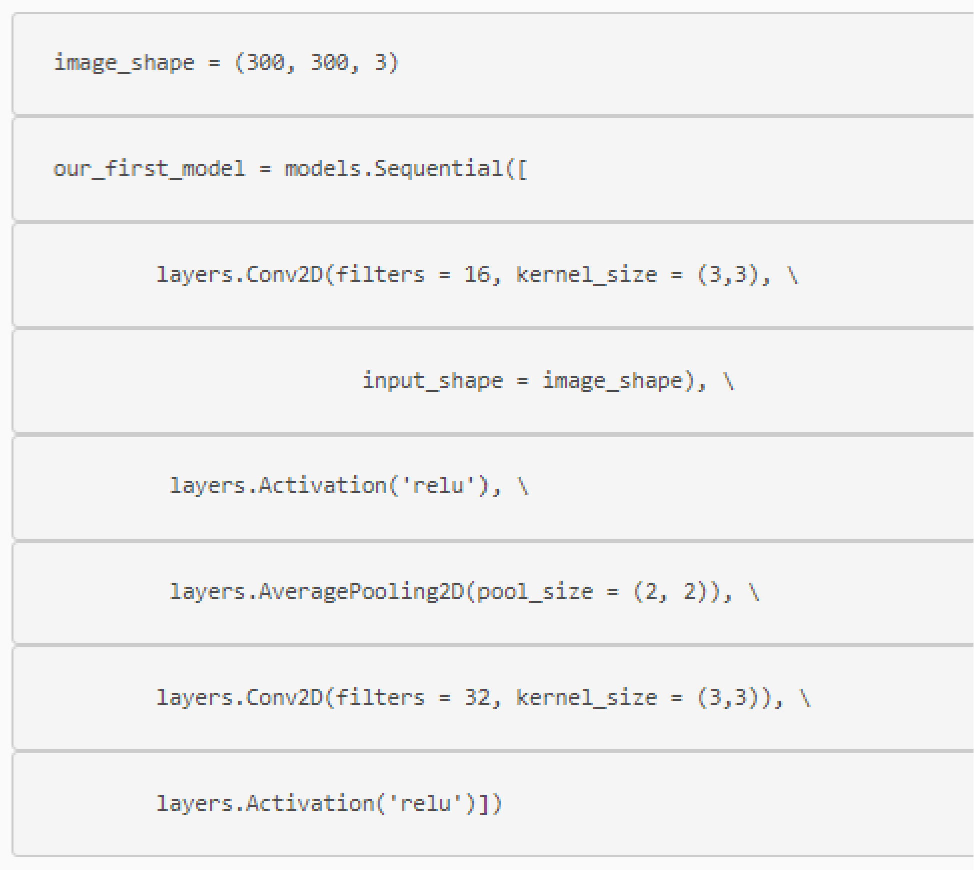
Average Pooling

- For average pooling, you would use AveragePooling2D in place of MaxPool2D.

- To implement the average pooling code, you could use the following:

```
layers.AveragePooling2D(pool_size=(2, 2), strides=None, \
                        padding='valid')
```

Average Pooling

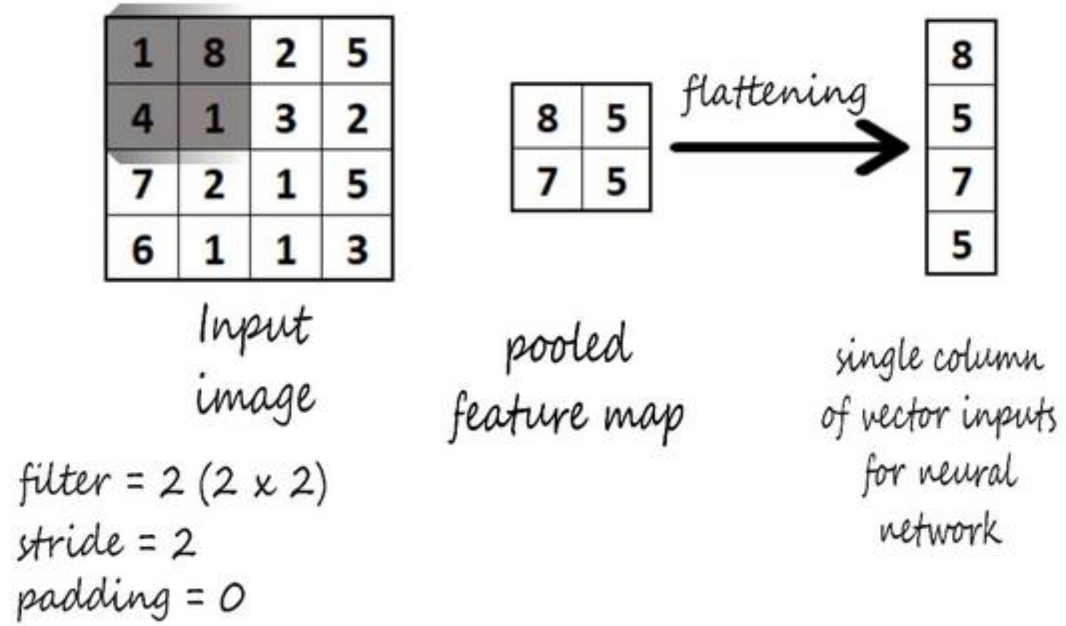


```
image_shape = (300, 300, 3)
our_first_model = models.Sequential([
        layers.Conv2D(filters = 16, kernel_size = (3,3), \
                        input_shape = image_shape), \
         layers.Activation('relu'), \
         layers.AveragePooling2D(pool_size = (2, 2)), \
        layers.Conv2D(filters = 32, kernel_size = (3,3)), \
        layers.Activation('relu')])
```

"Complete Exercise"

“Creating a Pooling Layer for a CNN”

Flattening Layer



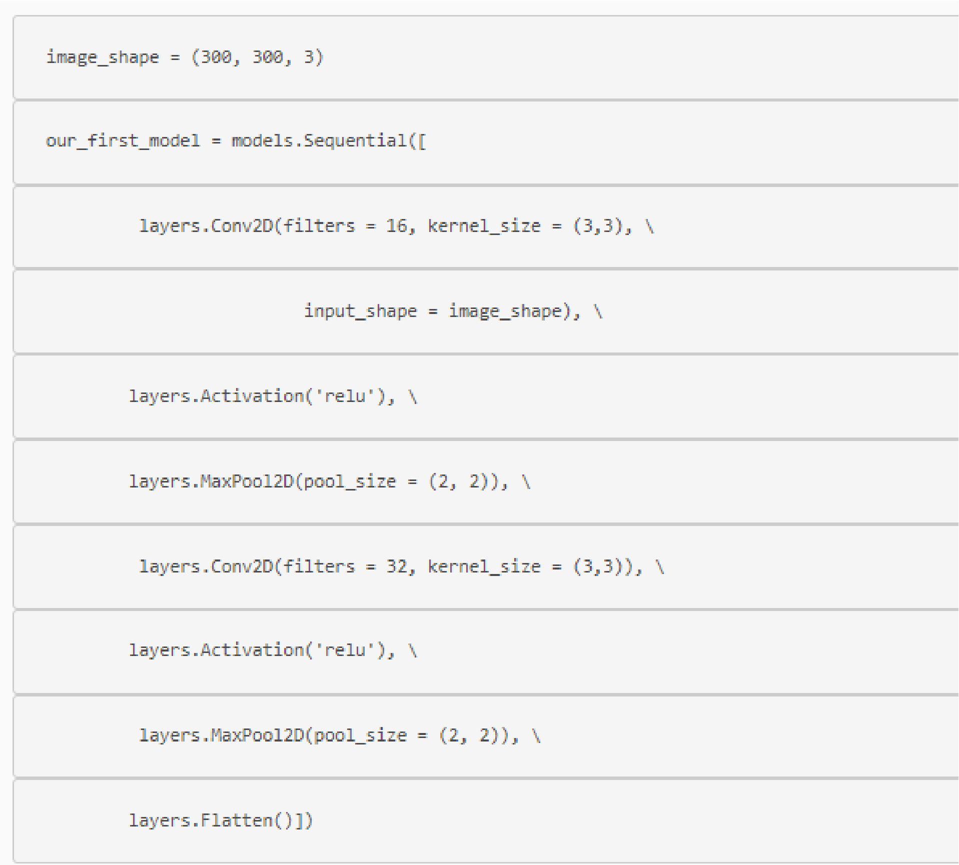
1	8	2	5
4	1	3	2
7	2	1	5
6	1	1	3

8	5
7	5

8
5
7
5

Flattening Layer

- The following is an implemented flattening layer:



```
image_shape = (300, 300, 3)
our_first_model = models.Sequential([
         layers.Conv2D(filters = 16, kernel_size = (3,3), \
                         input_shape = image_shape), \
        layers.Activation('relu'), \
        layers.MaxPool2D(pool_size = (2, 2)), \
         layers.Conv2D(filters = 32, kernel_size = (3,3)), \
        layers.Activation('relu'), \
         layers.MaxPool2D(pool_size = (2, 2)), \
        layers.Flatten()])
```

Image Augmentation

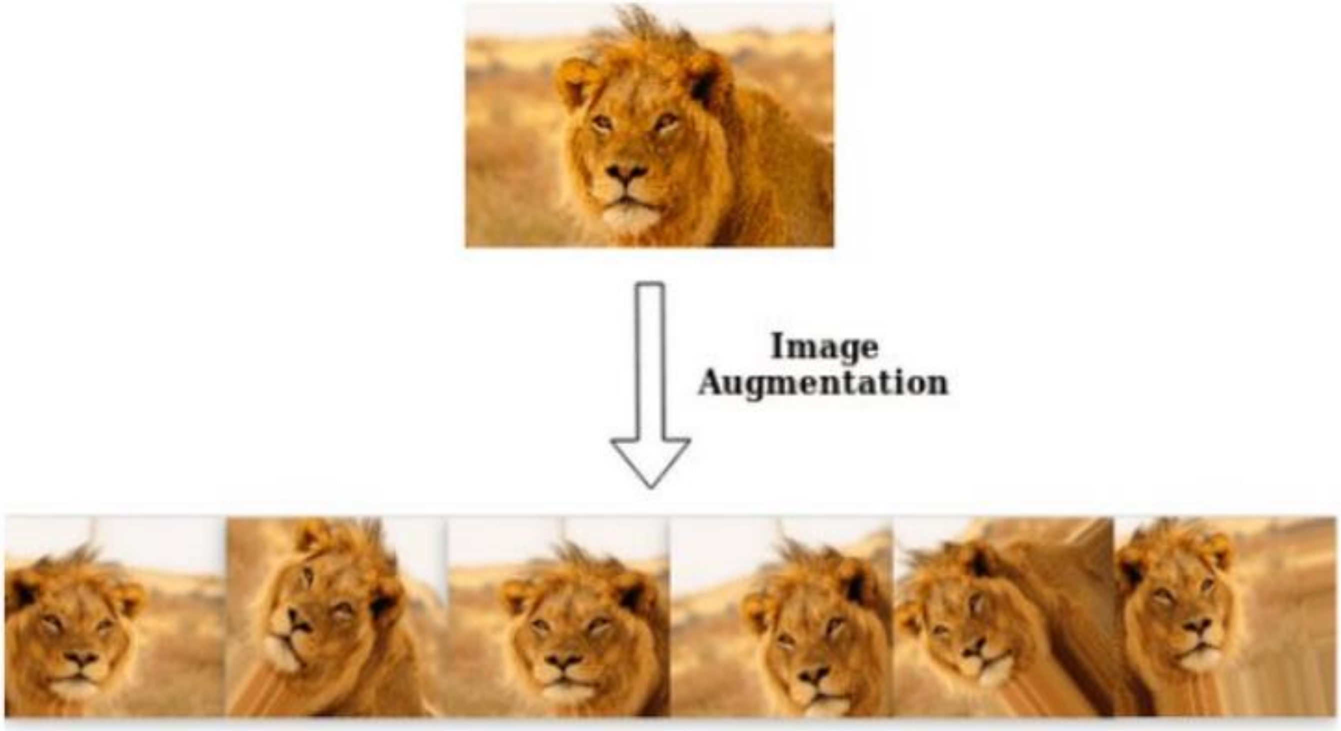


Image Augmentation

- You will use the dataset.map() function to map preprocessing image augmentation functions to your dataset, that is, our_train_dataset:

```
from tensorflow import image as tfimage
from tensorflow.keras.preprocessing import image as kimage
```

Image Augmentation

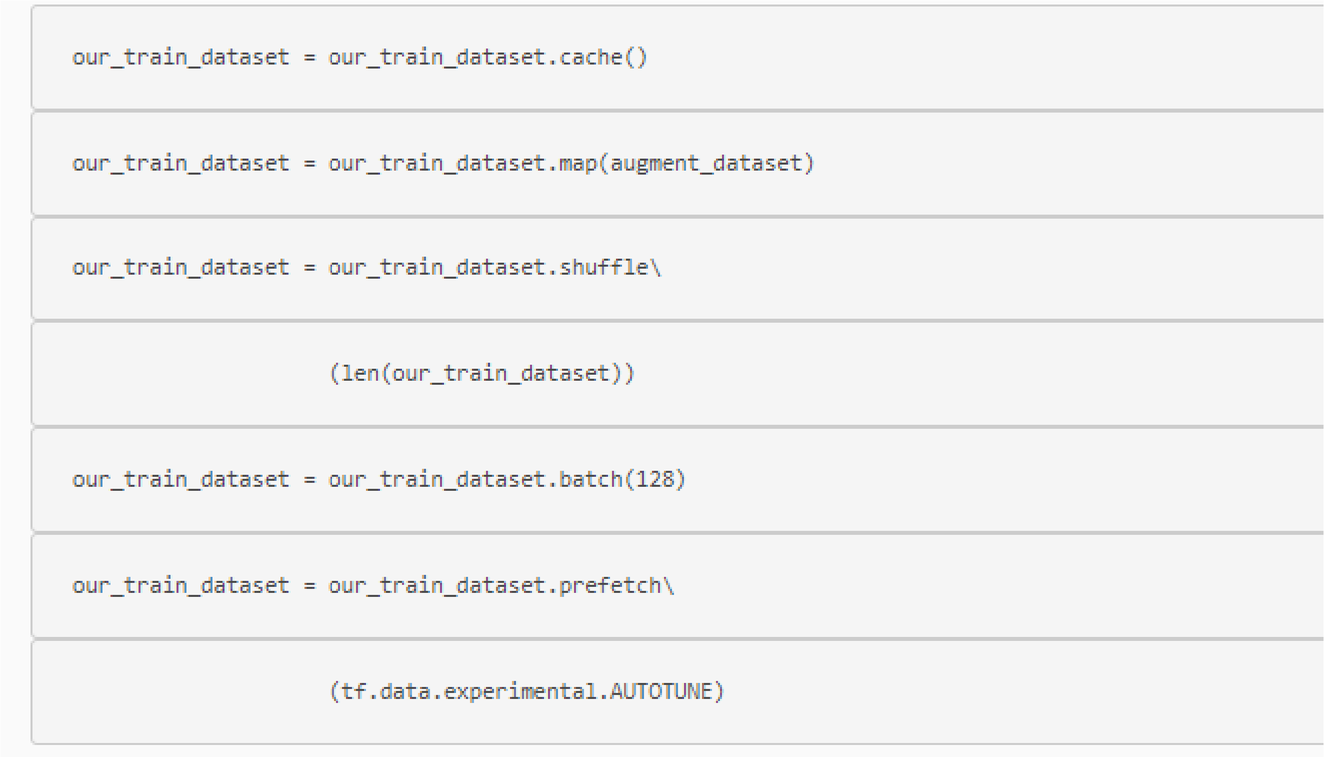
- You will use the tensorflow.image and tensorflow.keras.preprocessing.image packages for this purpose.
- These packages have a lot of image manipulation functions that can be used for image data augmentation:

```
augment_dataset(image, label):
    image = kimage.random_shift(image, wrg = 0.1, hrg = 0.1)
    image = tfimage.random_flip_left_right(image)
    return image, label
```

Image Augmentation

```
augment_dataset(image, label):
    image = kimage.random_shift(image, wrg = 0.1, hrg = 0.1)
    image = tfimage.random_flip_left_right(image)
    return image, label
our_train_dataset =
our_train_dataset.map(augment_dataset)
model.fit(our_train_dataset,\
          epochs=50,\
          validation_data=our_test_dataset)
```

Image Augmentation



```
our_train_dataset = our_train_dataset.cache()

our_train_dataset = our_train_dataset.map(augment_dataset)

our_train_dataset = our_train_dataset.shuffle\

                    (len(our_train_dataset))

our_train_dataset = our_train_dataset.batch(128)

our_train_dataset = our_train_dataset.prefetch\

                    (tf.data.experimental.AUTOTUNE)
```

Image Augmentation

- Here's an example of random_rotation, random_shift, and random_brightnes implementation.
- Use the following code to randomly rotate an image up to an assigned value:

```
image = kimage.random_rotation(image, rg = 135)
```

- In Figure, you can see the outcome of random_rotation

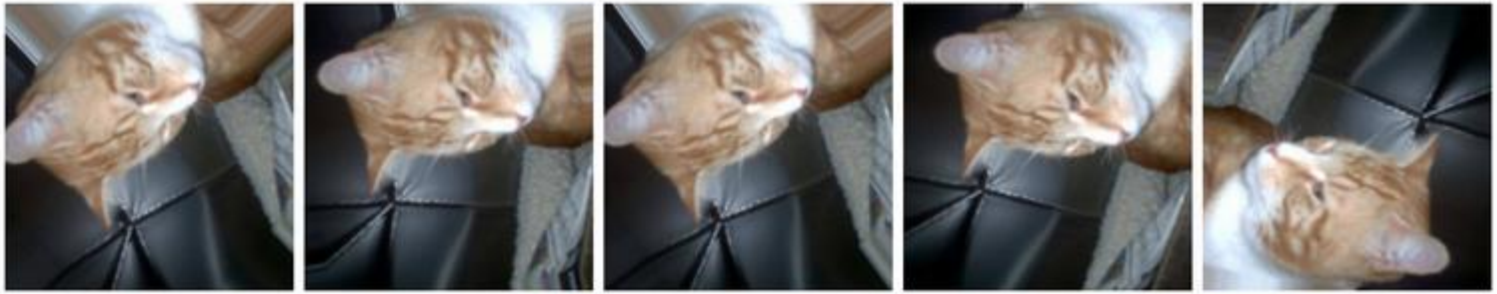


Image Augmentation

- random_shift is used to randomly shift the pixels width-wise. Notice the .15 in the following code, which means the image can be randomly shifted up to 15 pixels:

```
image = kimage.random_shift(image, wrg = 0.15, hrg = 0)
```

- The following figure shows the random adjustment of an image's width by up to 15 pixels:

Image Augmentation

- Again, random_shift is used here, which randomly adjusts the height by 15 pixels:

```
image = kimage.random_shift(image, wrg = 0, hrg = 0.15)
```

- Figure shows the random adjustment of an image’s height by up to 15 pixels:

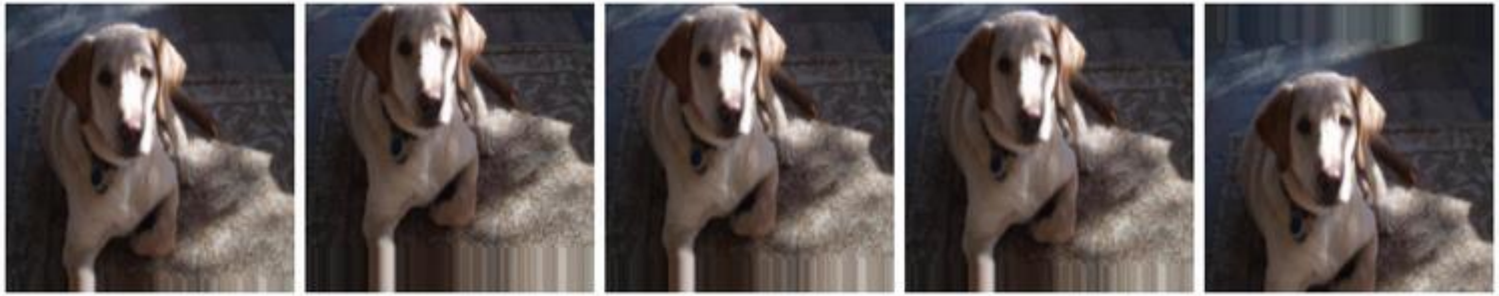
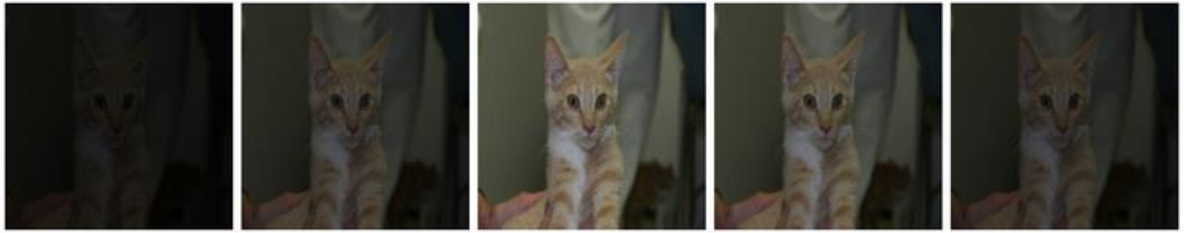


Image Augmentation

- Anything below 1.0 will darken the image, So, in this example, the images are being darkened randomly between 10% and 90%:

```
image = kimage.random_brightness(image,
brightness_range=(0.1,0.9))
```

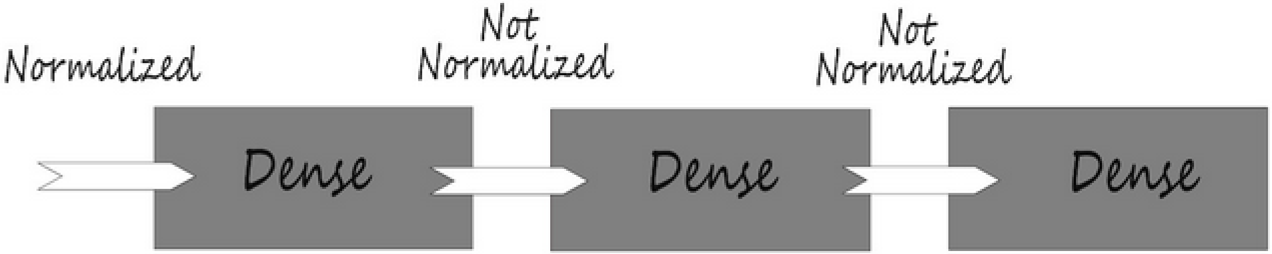
- In the following figure, you've adjusted the brightness with random_brightness:



Batch Normalization

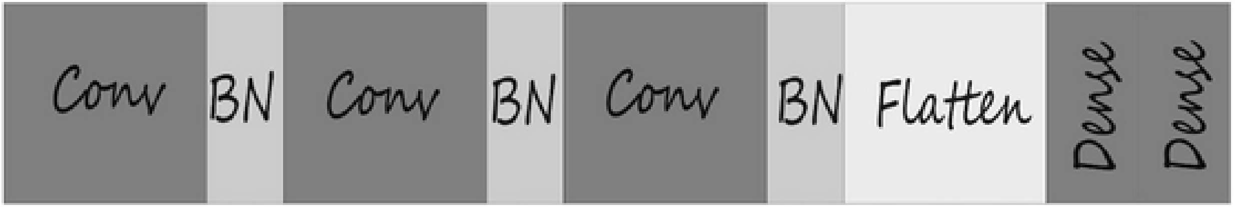
- Batch norm standardizes the inputs for a mini-batch and "normalizes" the input layer.

- It is most commonly used following a convolutional layer, as shown in the following figure:



Batch Normalization

- The following figure shows one common way that batch normalization is implemented.
- In the following example, you can see that you have a batch norm layer following a convolutional layer three times, Then you have a flattening layer, followed by two dense layers:



Batch Normalization

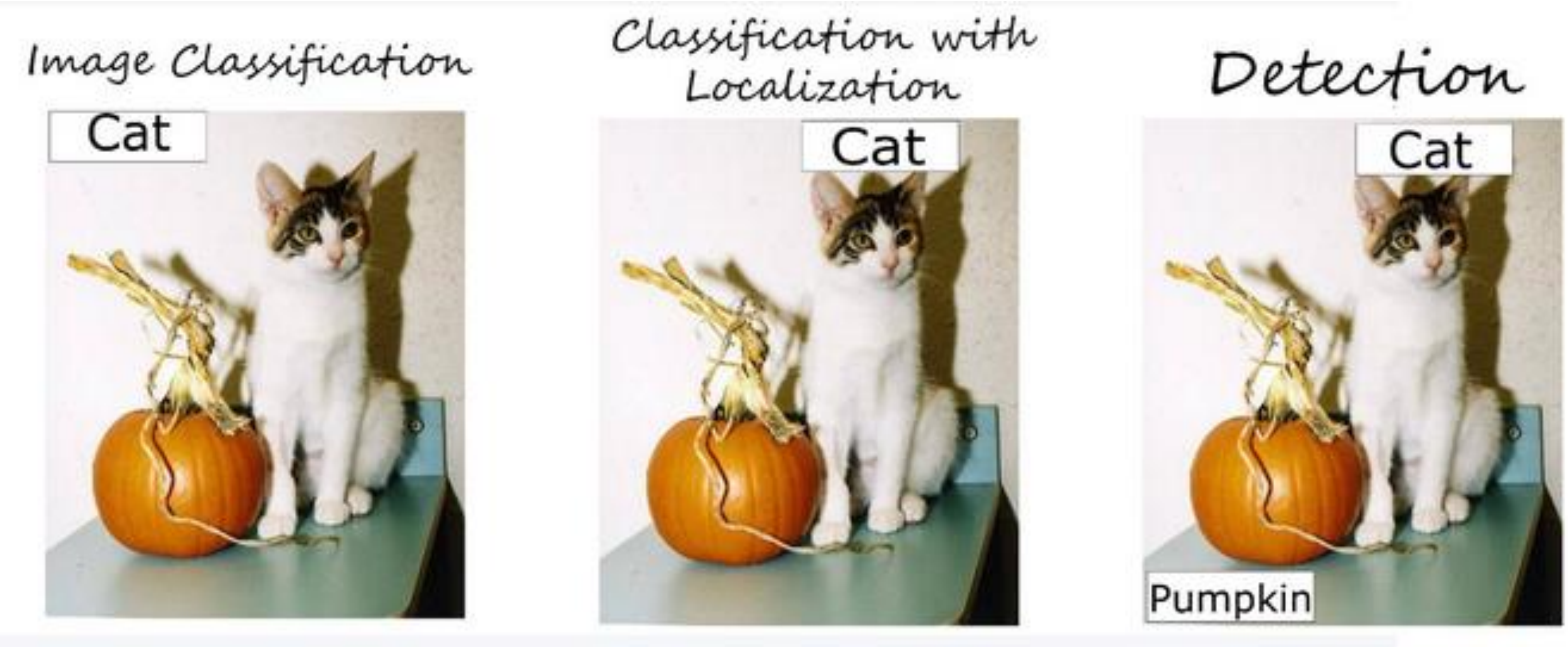
- The following is an example of BatchNormalization implementation.
- You can simply add a batch norm layer, followed by an activation layer:

```
model.add(layers.Conv2D(filters = 64, kernel_size = (3, 3),
use_bias=False))
model.add(layers.BatchNormalization())
model.add(layers.Activation("relu"))
```

Binary Image Classification

- Binary classification is the simplest approach for classification models as it classifies images into just two categories.
- In this lesson, we started with the convolutional operation and discussed how you use it as an image transformer.
- Then, you learned what a pooling layer does and the differences between max and average pooling.

Object Classification



"Complete Exercise"

“Building a CNN”

"Complete Activity"

“Building a CNN with More ANN Layers”

Summary

- This lesson covered CNNs. We reviewed core concepts such as neurons, layers, model architecture, and tensors to understand how to create effective CNNs.
- You learned about the convolution operation and explored kernels and feature maps.
- We analyzed how to assemble a CNN, and then explored the different types of pooling layers and when to apply them.