Tensorflow

(Code: Subhajit Das)

What is Tensorflow?

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It was created by the Google Brain team and initially released to the public in 2015. TensorFlow can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

In TensorFlow, numerical computation is performed using data flow graphs where:

- Nodes in the graph represent mathematical operations.
- Edges in the graph represent the multidimensional data arrays (called tensors) communicated between them.

TensorFlow makes it easy for beginners and experts to create machine learning models for desktop, mobile, web, and cloud. It also provides robust capabilities to deploy your models on any environment - servers, edge devices, browsers, mobile, microcontrollers, CPUs, GPUs, FPGAs.

Where we can use Tensarflow?

TensorFlow is versatile and can be used in a variety of applications:

1. **Image Recognition**: TensorFlow is widely used in image recognition applications. It involves pixel and pattern matching to identify the image and its parts. This finds application in many domains including healthcare systems, banking systems, and educational institutions.

2. **Voice Recognition**: TensorFlow has significant use in voice recognition systems like Telecom, Mobile companies, security systems, search engines, etc. It uses the voice recognition systems for giving commands, performing operations and giving inputs without using keyboards, mouse.

3. **Video Detection**: With increased technology, companies and businesses look forward to more secure and optimized systems. Hence, motion detection is used widely at airport security checks, gaming controls, and movement detection.

4. **Natural Language Processing**: TensorFlow can be used in text-based applications, and it plays a significant role in natural language processing.

5. **Robotics**: TensorFlow enables us to quickly and easily build powerful AI models with high accuracy and performance, which can be used in robotics.

6. **Handwritten Digit Classification, Word Embeddings, Recurrent Neural Networks, Sequence-to-Sequence Models for Machine Translation**: TensorFlow can train deep neural networks for these applications.

Some key points of Tensorflow:

1. **End-to-End Platform**: TensorFlow is an end-to-end platform for machine learning.
2. **Multidimensional Array-Based Numeric Computation**: TensorFlow supports multidimensional-array based numeric computation, similar to NumPy.
3. **GPU and Distributed Processing**: TensorFlow can utilize GPU and distributed processing for faster computations.
4. **Automatic Differentiation**: TensorFlow implements automatic differentiation, which is essential for gradient-based optimization algorithms.
5. **Model Construction, Training, and Export**: TensorFlow provides functionalities for model construction, training, and export.
6. **Tensor Operations**: TensorFlow implements standard mathematical operations on tensors, as well as many operations specialized for machine learning.
7. **Variables**: To store model weights or other mutable state in TensorFlow, a `tf.Variable` is used.
8. **Scalability and Flexibility**: TensorFlow is designed to be scalable and flexible, allowing it to run on a wide range of devices and in a distributed setting.
9. **Abstraction Levels**: TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs.
10. **Eager Execution**: If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging.

Relation between Tensorflow and Keras

Keras and TensorFlow are both popular libraries used for machine learning and neural networks.

- **TensorFlow** is an open-source platform for machine learning and a symbolic math library that is used for machine learning applications. It offers both high and low-level APIs and is used for large datasets and high-performance models.

- **Keras** is a high-level neural network API that runs on top of TensorFlow (or another open-source library backend). It is designed to be user-friendly and is often used for small datasets. Keras provides various pre-trained models which help the user in further improving the models they are designing[1].

In mid-2017, Keras was adopted and integrated into TensorFlow, making it accessible to TensorFlow users through the `tf.keras` module. This means that when you use Keras, you're really using the TensorFlow library. The `tf.keras` module in TensorFlow 2.0 is the recommended high-level API for this version of TensorFlow.

In summary, while TensorFlow provides the foundational framework for building and training machine learning models, Keras provides a higher-level, more intuitive set of abstractions that make it easier to build and train models. They work together to provide an easy-to-use and powerful toolset for machine learning.

What is TensorBoard and TensorFlow Serving?

TensorBoard and TensorFlow Serving are two powerful tools in the TensorFlow ecosystem that play crucial roles in the machine learning workflow.

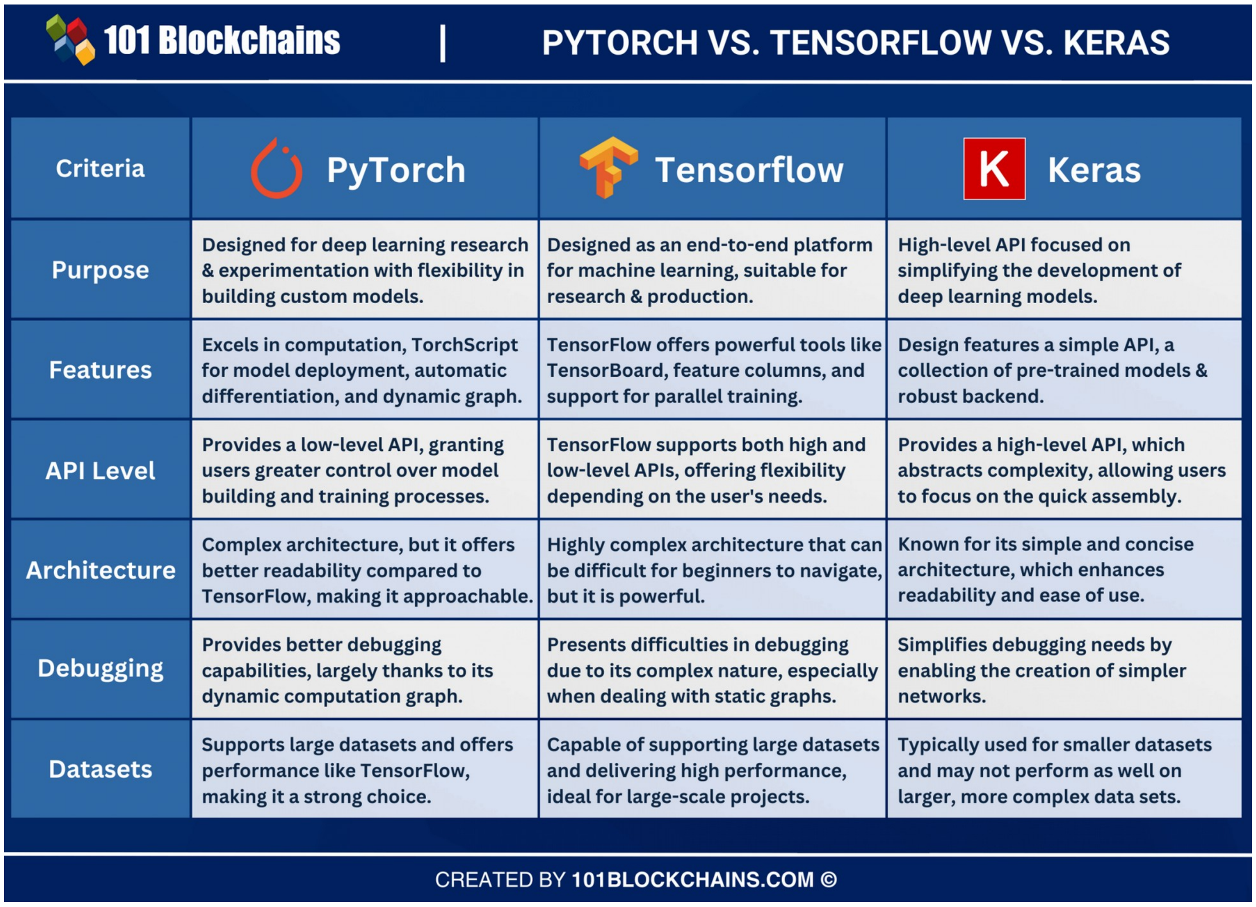
TensorBoard TensorBoard is a visualization tool for machine learning experiments. It helps you:

- **Track and visualize metrics**: Monitor training progress, loss, accuracy, and other metrics over time.
- **Visualize the model graph**: Inspect the architecture of your neural network, including layers, connections, and data flow.
- **Analyze model performance**: Explore histograms of weights, biases, and activations to understand model behavior.
- **Project embeddings**: Visualize high-dimensional data in 2D or 3D space.
- **Profile TensorFlow programs**: Identify performance bottlenecks and optimize your models.

TensorFlow Serving TensorFlow Serving is a flexible, high-performance platform for deploying machine learning models in production. It allows you to:

- **Serve multiple models**: Deploy and manage multiple models simultaneously.
- **Version models**: Easily switch between different versions of a model.
- **Handle real-time requests**: Process incoming requests efficiently and return predictions.
- **Scale horizontally**: Distribute the load across multiple servers for high throughput.

In essence, TensorBoard helps you understand your model's performance and behavior, while TensorFlow Serving helps you deploy and serve your models in production.



101 Blockchains | PYTORCH VS. TENSORFLOW VS. KERAS

Criteria	PyTorch	Tensorflow	Keras
Purpose	Designed for deep learning research & experimentation with flexibility in building custom models.	Designed as an end-to-end platform for machine learning, suitable for research & production.	High-level API focused on simplifying the development of deep learning models.
Features	Excels in computation, TorchScript for model deployment, automatic differentiation, and dynamic graph.	TensorFlow offers powerful tools like TensorBoard, feature columns, and support for parallel training.	Design features a simple API, a collection of pre-trained models & robust backend.
API Level	Provides a low-level API, granting users greater control over model building and training processes.	TensorFlow supports both high and low-level APIs, offering flexibility depending on the user's needs.	Provides a high-level API, which abstracts complexity, allowing users to focus on the quick assembly.
Architecture	Complex architecture, but it offers better readability compared to TensorFlow, making it approachable.	Highly complex architecture that can be difficult for beginners to navigate, but it is powerful.	Known for its simple and concise architecture, which enhances readability and ease of use.
Debugging	Provides better debugging capabilities, largely thanks to its dynamic computation graph.	Presents difficulties in debugging due to its complex nature, especially when dealing with static graphs.	Simplifies debugging needs by enabling the creation of simpler networks.
Datasets	Supports large datasets and offers performance like TensorFlow, making it a strong choice.	Capable of supporting large datasets and delivering high performance, ideal for large-scale projects.	Typically used for smaller datasets and may not perform as well on larger, more complex data sets.

CREATED BY 101BLOCKCHAINS.COM ©

Installing TensorFlow

```
# Install TensorFlow 2.0
#!pip install tensorflow
```

Importing TensorFlow

```
import tensorflow as tf
from tensorflow import keras

# Printing the version
print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.15.0
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Fahion Style Analysis

```
fashion_df = keras.datasets.fashion_mnist
fashion_df
```

```
<module 'keras.api._v2.keras.datasets.fashion_mnist' from
'/usr/local/lib/python3.10/dist-packages/keras/api/_v2/keras/datasets/
fashion_mnist/__init__.py'>
```

Train-Test Spliting

```
(train_images, train_labels), (test_images, test_labels) =
fashion_df.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Loading the dataset returns four NumPy arrays:

- The `train_images` and `train_labels` arrays are the *training set*—the data the model uses to learn.
- The model is tested against the *test set*, the `test_images`, and `test_labels` arrays.

The images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255. The *labels* are an array of integers, ranging from 0 to 9. These correspond to the *class* of clothing the image represents:

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Analysis of Data

```
# Checking the dtype of train_images
print(type(train_images))
```

```
<class 'numpy.ndarray'>
```

```
# Viewing the shape of train_images
print(train_images.shape) # means there are 60,000 images in the
training set, each of size 28x28 pixels
```

```
(60000, 28, 28)
```

```
# Total size of train_images
print(train_images.size)
```

```
47040000
```

```
# Viewing the shape of train_images
len(train_labels) # means there are 60,000 labels in the training set
```

```
60000
```

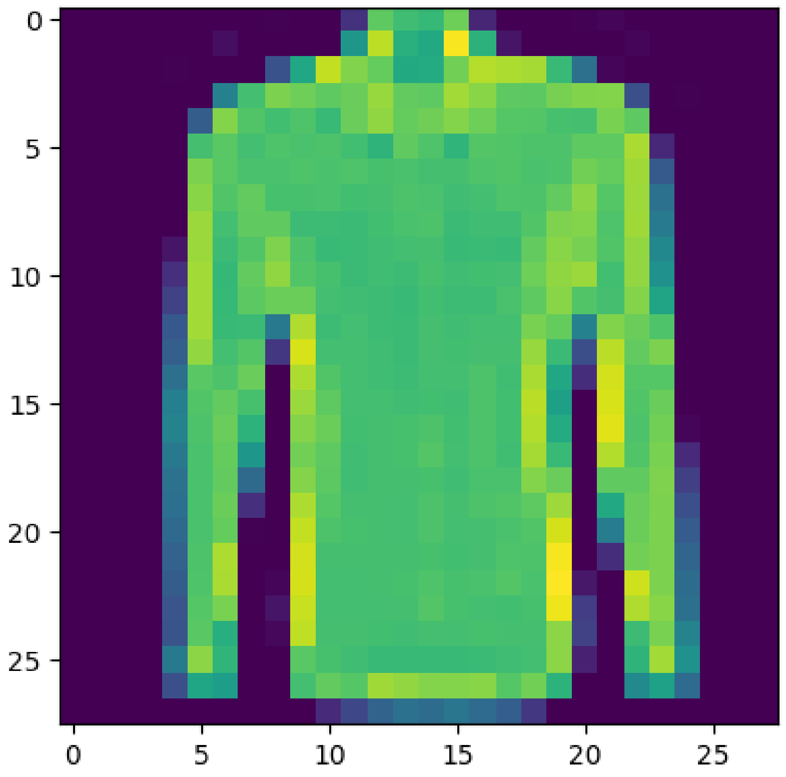
```
# Printing train_labels
train_labels # Each label is an integer between 0 and 9
```

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)

# Viewing the shape of test_images
print(test_images.shape)

(10000, 28, 28)
```

Printting the Images

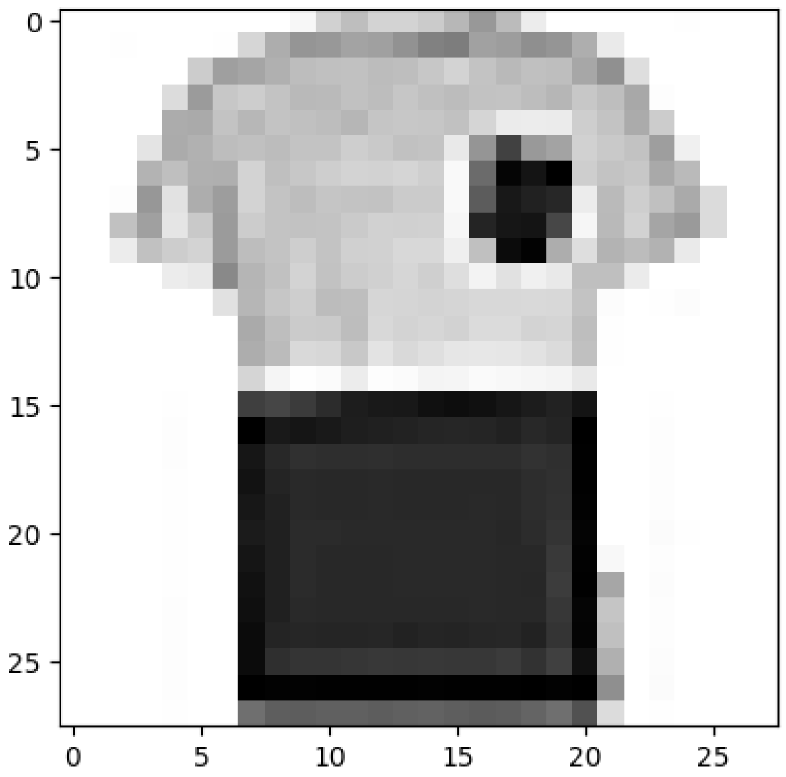
(Here, pixel values fall in the range of 0 to 255)

```
# Printing a image from train_image array
plt.imshow(train_images[40])

<matplotlib.image.AxesImage at 0x79c7436b7310>
```



```
# print the class name of the image
label_index = train_labels[6]
print(class_names[label_index])

Sneaker

# Printing a image from test_image array
plt.imshow(test_images[120],
           cmap = plt.cm.binary) # Viewing in binary form / black and
whhite form
```

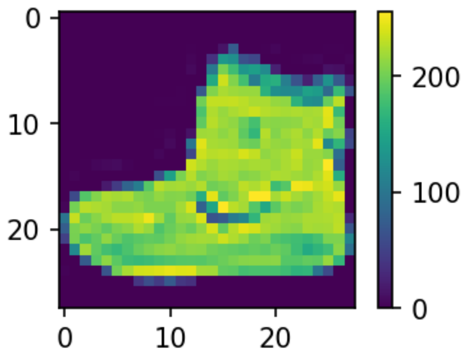
```
<matplotlib.image.AxesImage at 0x79c740187af0>
```

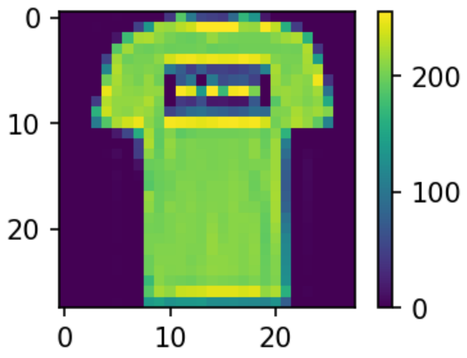


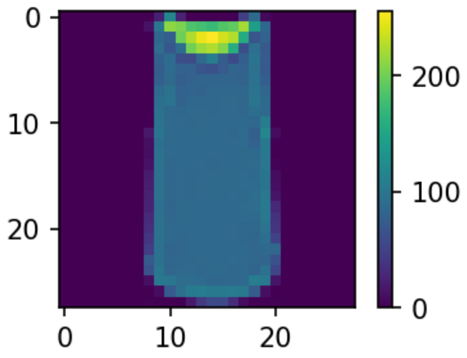
```
# print the class name of the image
label_index = test_labels[120]
print(class_names[label_index])
```

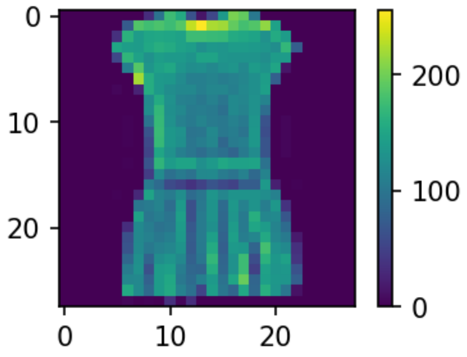
```
T-shirt/top
```

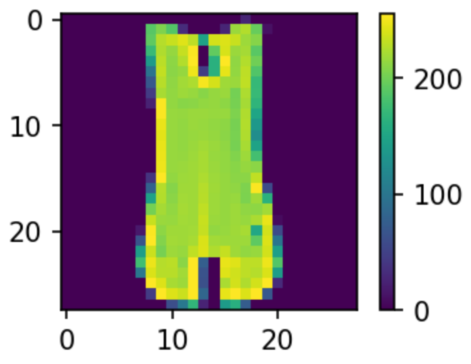
```
# print the first 6 images of training set
for i in range(6):
    plt.figure(figsize=(3, 2), dpi=150)
    plt.imshow(train_images[i])
    plt.colorbar()
    plt.show()
```

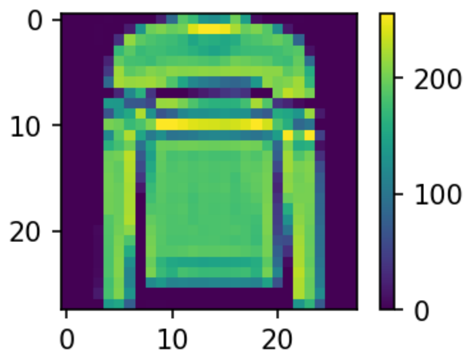

0
10
20
0
10
20
200
100
0


0
10
20
0
10
20
200
100
0


0
10
20
0
10
20
200
100
0


0
10
20
0
10
20
200
100
0





Preprocess the data (Scaling)

Scaling pixel values to a range of 0 to 1 is a common preprocessing step before feeding them into a neural network model. This process is known as normalization[1]. Here are the reasons why we do this:

1. **Improving Convergence**: Neural networks process inputs using small weight values, and inputs with large integer values can disrupt or slow down the learning process[3]. Normalizing the pixel values helps the model to converge faster during the training process.

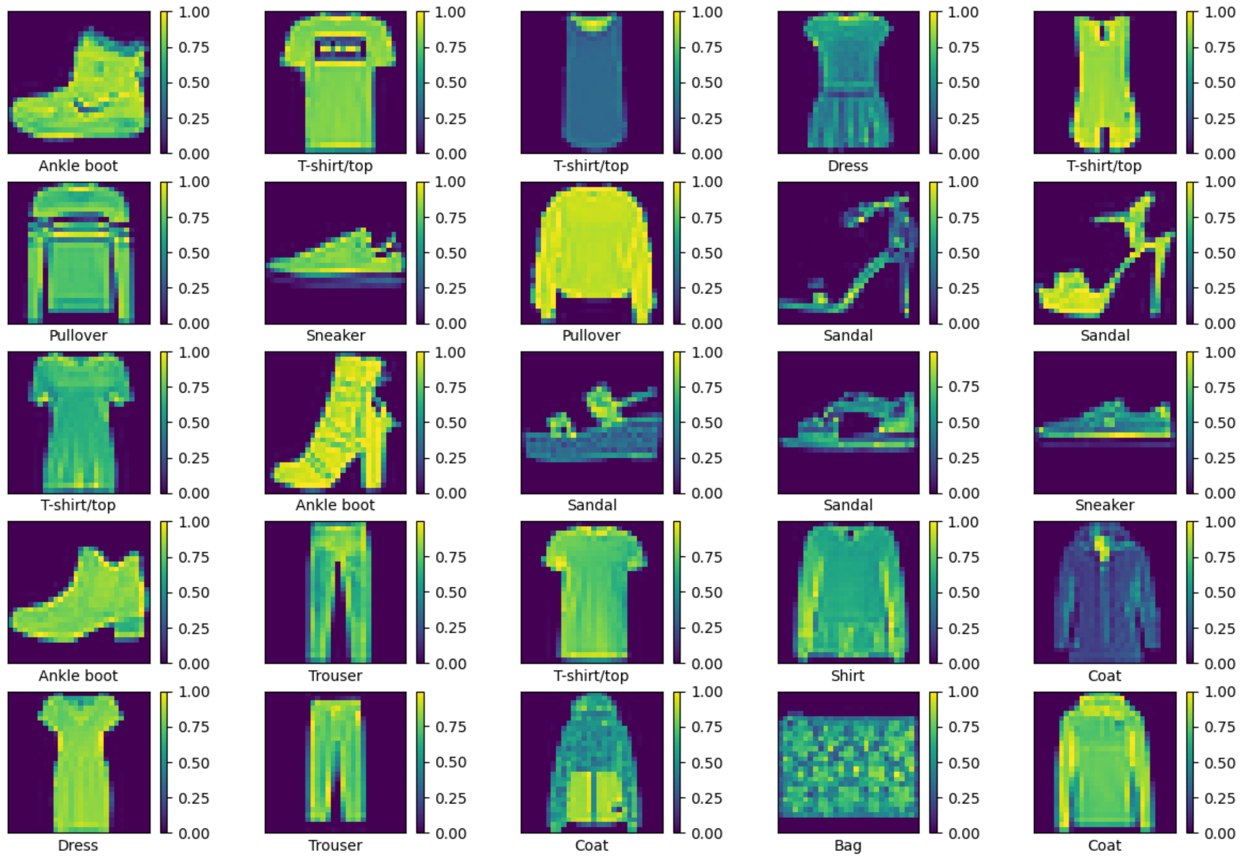
2. **Consistent Data Range**: Images can have pixel values that range from 0 to 255. By scaling these values, we ensure that the input features (pixel intensities) are within a similar range (0 to 1), which allows the model to learn more effectively.

3. **Zero-Centered Data**: After normalization, the data will be centered around 0, which aids in optimization during the learning process[2].

In the case of images, each pixel is represented by a value between 0 and 255 (for an 8-bit grayscale image). When you divide each pixel value by 255, the data is scaled to a range of 0 to 1. This scaled data is what you feed into your neural network model.

Remember, preprocessing steps like this are not exclusive to image data and can be beneficial for many types of data when training a neural network.

```
# Scale these values to a range of 0 to 1 before feeding them to the
neural network model. To do so, divide the values by 255
train_images = train_images / 255.0

test_images = test_images / 255.0

# Display the first 25 images from the training set and display the
class name below each image
plt.figure(figsize=(15,10))

for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.imshow(train_images[i])
    plt.xticks([]) # To remove the measurement values of x-axis
    plt.yticks([]) # To remove the measurement values of y-axis
    plt.colorbar()
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```



Build the model

Set up the layers

The basic building block of a neural network is the layer. Layers extract representations from the data fed into them. **tf.keras.layers.Dense**, have parameters that are learned during training.

```
model = tf.keras.Sequential([ # To initialize a linear stack of
layers, which you can then add to with the .add() method, or in this
case, by passing a list of layers to the constructor.
    tf.keras.layers.Flatten(input_shape=(28, 28)), # This layer
transforms the format of the images from a 2-D array (28*28 pixels) to
a 1-D array (of 28*28=784 pixels). It has no parameters to learn
    tf.keras.layers.Dense(128, activation='relu'), # After the pixels
are flattened. These are densely connected, or fully connected, neural
layers. The Dense layer has 128 nodes and uses the ReLU activation.
    tf.keras.layers.Dense(10) # Logits array with length of 10. Each
node contains a score that indicates the current image belongs to one
of the 10 classes (for working with 10-class classification problem).
])
```

Compile the model

Before the model is ready for training, it needs a few more settings. These are added during the model's compile step:

Optimizer —This is how the model is updated based on the data it sees and its loss function. The optimizer is the algorithm that adjusts the weights and biases during training.

Loss function —The loss function measures how well the model did on each example, and then the optimizer uses this to update the model. You want to minimize this function to "steer" the model in the right direction.

Metrics —Used to monitor the training and testing steps. The following example uses accuracy, the fraction of the images that are correctly classified.

```
model.compile(optimizer = 'adam', # adapts the learning rate based on
how training is going and generally works well with little tuning
              loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), #
Sparse is appropriate for multi-class classification problems where
each example belongs to exactly one class
              # from_logits=True argument means that the function
should interpret the model’s raw output values (the “logits”) as
predictions, rather than probabilities
              metrics = ['accuracy']) # the list of metrics to be
evaluated during training and testing. In this case, we’re asking
Keras to report the ‘accuracy’ of the predictions
```

Train the model

Training the neural network model requires the following steps:

1. Feed the training data to the model. In this example, the training data is in the train_images and train_labels arrays.

2. The model learns to associate images and labels.
3. You ask the model to make predictions about a test set—in this example, the test_images array.
4. Verify that the predictions match the labels from the test_labels array.

```
model.fit(train_images, train_labels, epochs=10) # Trains the model
for 10 epochs on the training data. Each epoch is an iteration over
the entire dataset.
# During each epoch, the model learns to adjust its weights to
minimize the loss, in order to improve its predictions.
# As the model trains, the loss and accuracy metrics are displayed.
This model reaches an accuracy of about 0.91 (or 91%) on the training
data

Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss:
0.4917 - accuracy: 0.8264
Epoch 2/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.3738 - accuracy: 0.8646
Epoch 3/10
1875/1875 [==============================] - 13s 7ms/step - loss:
0.3372 - accuracy: 0.8770
Epoch 4/10
1875/1875 [==============================] - 16s 8ms/step - loss:
0.3137 - accuracy: 0.8847
Epoch 5/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.2952 - accuracy: 0.8910
Epoch 6/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.2799 - accuracy: 0.8965
Epoch 7/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.2687 - accuracy: 0.8995
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss:
0.2587 - accuracy: 0.9035
Epoch 9/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.2473 - accuracy: 0.9075
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss:
0.2382 - accuracy: 0.9105

<keras.src.callbacks.History at 0x79c7219876a0>
```

Evaluate accuracy

```
# Accuracy of test dataset
test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2) #  Evaluate method returns the loss value and metrics
values for the model in test mode.
# verbose parameter is for logging. When verbose=2, it will print one
line per epoch

print('\nTest accuracy:', test_acc)

''' It is a little less than the accuracy on the training dataset.
This gap between training accuracy and test accuracy represents
overfitting. Overfitting happens when a machine learning model
 performs worse on new, previously unseen inputs than it does on the
training data. An overfitted model "memorizes" the noise and details
in the training dataset to a point where it negatively
  impacts the performance of the model on the new data. '''

313/313 - 1s - loss: 0.3443 - accuracy: 0.8831 - 642ms/epoch -
2ms/step

Test accuracy: 0.8830999732017517

{"type":"string"}
```

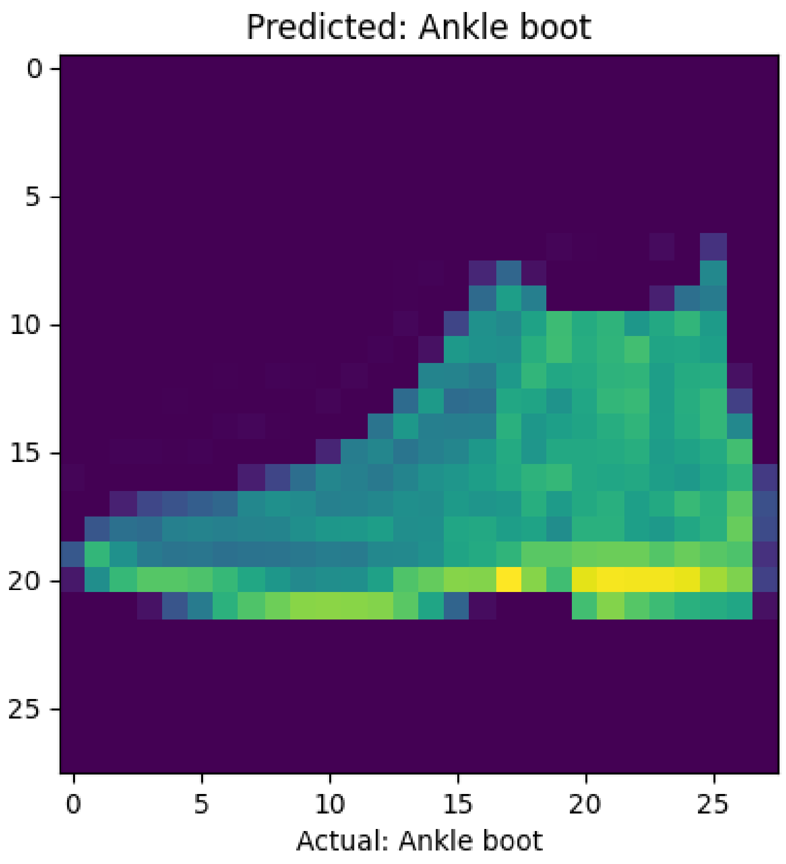
Make predictions

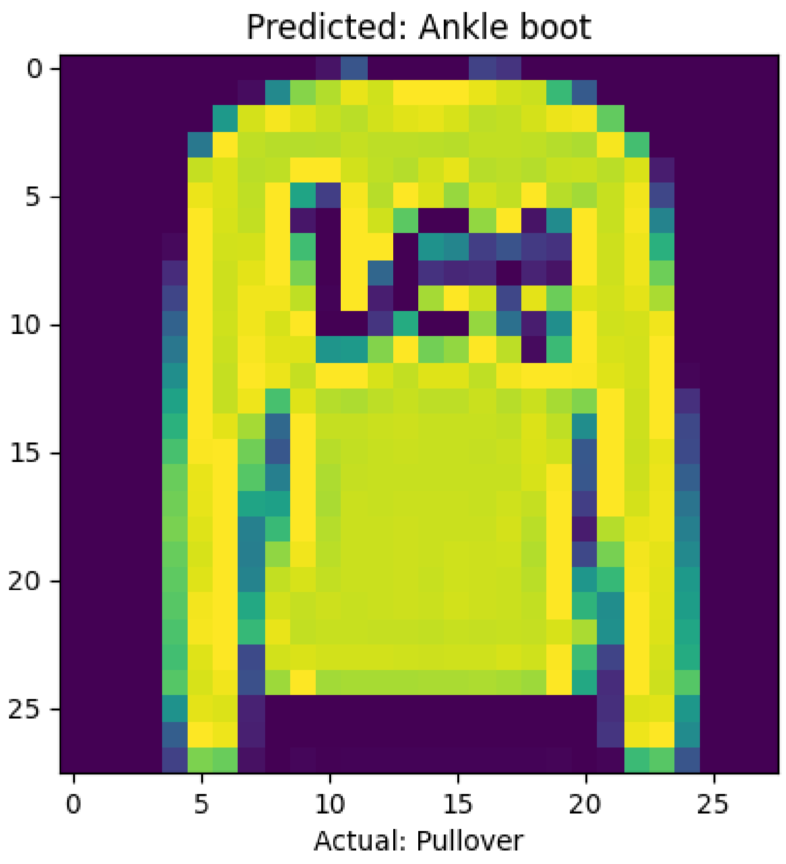
```
# predicted the label for each image in the testing set

prediction = model.predict(test_images)
print(prediction)

313/313 [==============================] - 1s 2ms/step
[[-11.152276   -23.289774   -14.409317   ...   1.9963496  -11.20776
    7.3895383 ]
 [ -0.91507804 -27.583532    10.352686   ... -24.41974    -12.486145
  -13.728784  ]
 [ -4.683188    16.804407   -11.120712   ... -27.724339    -7.4156165
  -26.651854  ]
 ...
 [ -5.7996583  -17.771835    -5.128388   ... -11.1371765    9.021154
  -19.13997   ]
 [ -7.5011067   11.418559   -11.136726   ... -12.318527    -7.940927
  -16.01125   ]
 [ -3.9634926  -11.601119    -6.5488386  ...  -1.28164     -4.3719187
   -5.8676186 ]]

# Printing the first prediction of dataset
# A prediction is an array of 10 numbers. They represent the model's
"confidence" that the image corresponds to each of the 10 different
articles of clothing
```

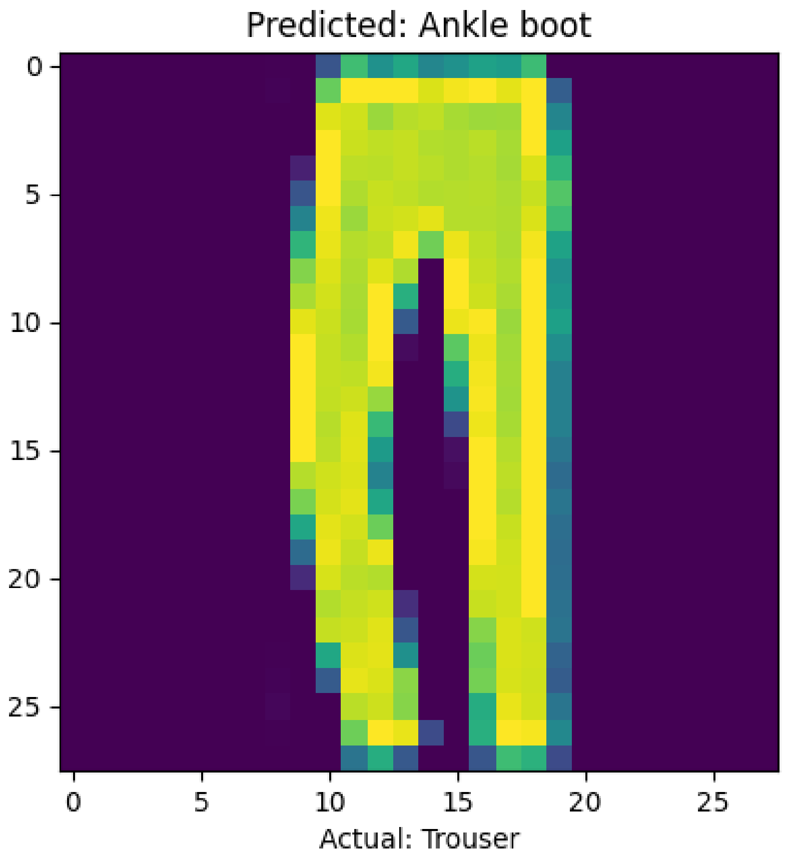
```
prediction = model.predict(test_images)
prediction[0]

313/313 [==============================] - 1s 2ms/step

array([-11.152276 , -23.289774 , -14.409317 , -18.767542 , -
18.234251 ,
        -1.3356622, -16.094244 ,   1.9963496, -11.20776  ,
7.3895383],
      dtype=float32)

# Getting the highest value of prediction number

prediction = model.predict(test_images)
print(np.argmax(prediction[0]))

313/313 [==============================] - 1s 3ms/step
9

# Viewing the label

prediction = model.predict(test_images)
print(class_names[np.argmax(prediction[0])])

313/313 [==============================] - 1s 2ms/step
Ankle boot
```

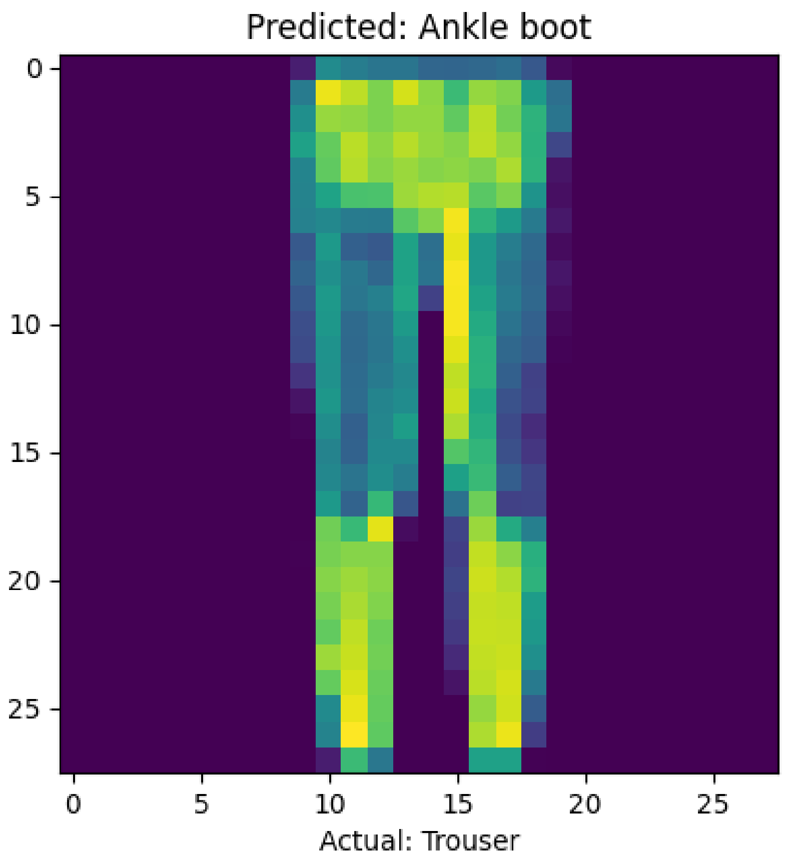
Verify predictions

```
# With the model trained, we can use it to make predictions about some
images

for i in range(4):
  plt.imshow(test_images[i])
  plt.xlabel("Actual: " + class_names[test_labels[i]])
  plt.title("Predicted: " + class_names[np.argmax(prediction[0])])
  plt.show()
```

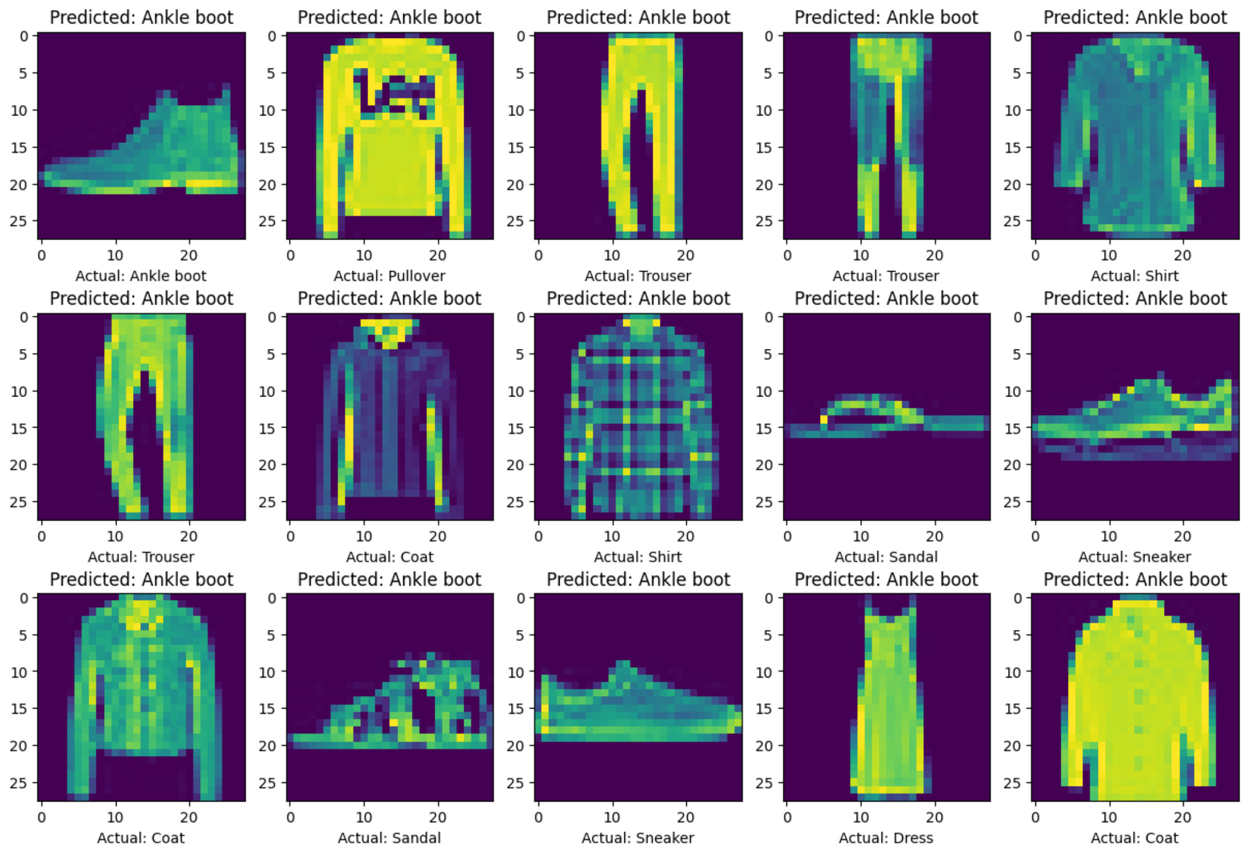

Predicted: Ankle boot
0
5
10
15
20
25
0
5
10
15
20
25
Actual: Ankle boot


Predicted: Ankle boot
0
5
10
15
20
25
0
5
10
15
20
25
Actual: Pullover


Predicted: Ankle boot
0
5
10
15
20
25
0
5
10
15
20
25
Actual: Trouser



```
plt.figure(figsize=(15,10))

for i in range(15):
    plt.subplot(3, 5, i+1)
    plt.imshow(test_images[i])
    plt.xlabel("Actual: " + class_names[test_labels[i]])
    plt.title("Predicted: " + class_names[np.argmax(prediction[0])])

plt.show()
```



Text Classification with Movie Reviews

This notebook classifies movie reviews as positive or negative using the text of the review. This is an example of binary—or two-class—classification.

```
movie_df = keras.datasets.imdb

import tensorflow_datasets as tfds
```

Train-Test Spliting

```
train_data, test_data = tfds.load(name="imdb_reviews", # Name of the
dataset to load
                                  split=["train", "test"], # IMDB
dataset is divided into “train” and “test” splits for training and
evaluating machine learning models
                                  batch_size=-1, # Loads the full
dataset in a single batch. This means that the entire dataset is
loaded into memory
                                  # Be careful with this setting if
your dataset is large, as it could use up all your available memory
                                  as_supervised=True) # Returns the
```

```
dataset in a 2-tuple structure (input, label) according to the dataset
builder. If it's False, it will return a dict

Downloading and preparing dataset 80.23 MiB (download: 80.23 MiB,
generated: Unknown size, total: 80.23 MiB) to
/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0...

{"model_id":"a879b074681e4613afd59620bc517444","version_major":2,"vers
ion_minor":0}

{"model_id":"3ae95d98822a4a6db12d0489dbb32b31","version_major":2,"vers
ion_minor":0}

{"model_id":"a7e4a180bdfa4984b100f437b8e45f8e","version_major":2,"vers
ion_minor":0}

{"model_id":"1581c0d4fdcf4d35b230cdc424ea3d92","version_major":2,"vers
ion_minor":0}

{"model_id":"24db01bd7e514caa98e140c59b58d5d1","version_major":2,"vers
ion_minor":0}

{"model_id":"91118b2d3e4149a396bf81a456a5d482","version_major":2,"vers
ion_minor":0}

{"model_id":"e530f0521c0743cf9058dc3724c418dc","version_major":2,"vers
ion_minor":0}

{"model_id":"54ab5eae1ba541d9b48acdf8fdb78d9d","version_major":2,"vers
ion_minor":0}

{"model_id":"a3b12156ea2e47de91e1aed6e3cf44b4","version_major":2,"vers
ion_minor":0}

Dataset imdb_reviews downloaded and prepared to
/root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0. Subsequent
calls will reuse this data.

print(type(train_data))

<class 'tuple'>

# Convert dataset into a generator of NumPy arrays

train_examples, train_labels = tfds.as_numpy(train_data)
test_examples, test_labels = tfds.as_numpy(test_data)
```

What is train_data, train_examples and train_labels?

```
# Checking the datatype of train_data
print(type(train_data))
```

```
<class 'tuple'>

# Viewing the train_data
print(train_data)

(<tf.Tensor: shape=(25000,), dtype=string, numpy=
array([b"This was an absolutely terrible movie. Don't be lured in by
Christopher Walken or Michael Ironside. Both are great actors, but
this must simply be their worst role in history. Even their great
acting could not redeem this movie's ridiculous storyline. This movie
is an early nineties US propaganda piece. The most pathetic scenes
were those when the Columbian rebels were making their cases for
revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love
affair with Walken was nothing but a pathetic emotional plug in a
movie that was devoid of any real meaning. I am disappointed that
there are movies like this, ruining actor's like Christopher Walken's
good name. I could barely sit through it.",
       b'I have been known to fall asleep during films, but this is
usually due to a combination of things including, really tired, being
warm and comfortable on the sette and having just eaten a lot. However
on this occasion I fell asleep because the film was rubbish. The plot
development was constant. Constantly slow and boring. Things seemed to
happen, but with no explanation of what was causing them or why. I
admit, I may have missed part of the film, but i watched the majority
of it and everything just seemed to happen of its own accord without
any real concern for anything else. I cant recommend this film at
all.',
       b'Mann photographs the Alberta Rocky Mountains in a superb
fashion, and Jimmy Stewart and Walter Brennan give enjoyable
performances as they always seem to do. <br /><br />But come on
Hollywood - a Mountie telling the people of Dawson City, Yukon to
elect themselves a marshal (yes a marshal!) and to enforce the law
themselves, then gunfighters battling it out on the streets for
control of the town? <br /><br />Nothing even remotely resembling that
happened on the Canadian side of the border during the Klondike gold
rush. Mr. Mann and company appear to have mistaken Dawson City for
Deadwood, the Canadian North for the American Wild West.<br /><br
/>Canadian viewers be prepared for a Reefer Madness type of enjoyable
howl with this ludicrous plot, or, to shake your head in disgust.',
       ...,
       b'Okay. So I just got back. Before I start my review, let me
tell you one thing: I wanted to like this movie. I know I\'ve been
negative in the past, but I was hoping to be surprised and actually
come out liking the film. I didn\'t.<br /><br />It\'s not just the
fact that every horror clich\xc3\xa9 imaginable is in this. And it\'s
not just the fact that they make every little thing into a jump scare
(walking into a baseball bat left on the floor? Are you kidding me?).
It just wasn\'t scary. One thing I was surprised about: there was more
blood than I thought there was going to be.. which isn\'t saying
much.<br /><br />The film starts off with Donna being dropped off by
```

```
Lisa\'s mom at her house. She comes in.. goes upstairs. Camera pans to
her father dead on the couch. Spooky. She goes upstairs, where the
aforementioned baseball bat scene happens. Finds her brother on his
bed, apparently dead (how could she tell? He didn\'t have a spot of
blood on him). Killer comes in, Donna hides under bed, mom dies. She
runs outside screaming for help. Killer behind her: "I did it for us."
Cut to therapy session. This confused a lot of people- everyone was
asking whether or not her family actually died or if she imagined it-
and she mentions how the nightmares have started coming back. Filler
dialogue ensues.<br /><br />THey cut to the chase pretty quick. Few
scenes at the salon, they go to the hotel. Of course the killer is
already there (for some reason, he escaped 3 days ago but the
police/family weren\'t informed until he\'s already there). More
filler ensues.<br /><br />I\'m not going to go on about what happens
in the film, because I don\'t want to spoil it too much. If you want
to know who dies, Horror_Fan made a post about it already. But on the
subjects of deaths: they weren\'t that exciting. People in the theatre
actually laughed out loud (an experience I\'ve never had before in a
horror movie, not even in When A Stranger Calls) during several of
them. One in particular: the bus boy guy who gives the most
hilarious \'scared\' face I\'ve ever seen. The only death involving
any blood was Lisa\'s, and that was pretty scarce. Her throat is
slashed, blood (if you can even call it that- it was practically
black) splatters on the curtain-thing. The only other blood was on
Claire when we see her body. Apparently, Fenton decided to stab her a
few times after he choked her to death. Um, okay? The movie was one of
the most clich\xc3\xa9d I\'ve ever seen. Let\'s see here.. obligatory
close-mirror-curtain-BOOM! scene. Check. Twice, actually (you could
tell they were struggling). Mandatory backing-up-into-killer. Check.
There\'s also the backing-up-into-lamp scene, but you\'ve all seen
that. Oh, you say you want a birds-flying-away scare? Well, you got
it! (Yes, they managed to incorporate one of those in here). And, of
course, the we-have-security-on-all-exits-but-he-still-escaped scene.
Shall I go on? I could.<br /><br />For anyone saying the characters
weren\'t stupid, are you kidding me? "Oh, even though the massive
alarm is ringing, literally saying PLEASE VACATE THE BUILDING, and 3
of my friends are missing, I\'m going to go upstairs to get my wrap."
These characters were some of the most flawed and stupid characters
ever. The only likable character - Lisa - made one of the most stupid
moves in the movie. "Oh, I just realized the psycho-teacher is here! I
must leave my strong boyfriend behind to run off by myself to warn
her! Oh, shoot, the elevator is being to slow? Guess I\'ll take the
stairs and run off into the construction site!" Ugh. By the end of the
film, they all deserved to die. The only death anyone felt any remorse
for was Donna\'s boyfriend (I can\'t even remember his name- is that
bad?), and by that time, the audience was completely drained out of
this scareless, clich\xc3\xa9d film.<br /><br />There were SOME
positives- the acting was decent for the most part, and it was well-
shot. But that\'s about it.<br /><br />I\'d give it a 1/5, and that\'s
```

```
being generous. Just for the laughs (and believe me, the audience had
a few), and Brittany Snow.<br /><br />Oh, and the reaction was bad.
Very bad. People were boo-ing after the movie ended and buzz
afterwards was very negative. Expect bad legs for this one.',
       b'When I saw this trailer on TV I was surprised. In May of 2008
I was at Six Flags in New Jersey and this was showing at a 4-D
attraction (you know, the attraction that the seats move). I take it
that the version I saw was a shortened version (15 min.) and also re-
created to add the motion effects. It was a cute movie... but that was
it. It was educational and told about the first mission but the ending
of a CGI spacewalk seemed a bit...well...trite. I was not a big fan of
the movie but i would recommend this movie for any parent wanting to
inform their children in a fun way about the first moonwalk. I will
say, the character actors were well selected and the characters
themselves were cute. So all-in-all, I would say, if you want to bring
the younger kids... go for it. But if you are wanting to take your
older kids, take them to another movie... they will thank you.',
       b'First of all, Riget is wonderful. Good comedy and mystery
thriller at the same time. Nice combination of strange \'dogma\' style
of telling the story together with good music and great actors. But
unfortunately there\'s no \'the end\'. As for me it\'s unacceptable. I
was thinking... how it will be possible to continue the story without
Helmer and Drusse? ...and I have some idea. I think Lars should make
RIGET III a little bit different. I\'m sure that 3rd part without
Helmer wouldn\'t be the same. So here\'s my suggestion. Mayble little
bit stupid, maybe not. I know that Lars likes to experiment. So why
not to make small experiment with Riget3? I think the only solution
here is to create puppet-driven animation (like for example "team
America" by Trey Parker) or even computer 3d animation. I know it\'s
not the same as real actors, but in principle I believe it could
work... only this way it\'s possible to make actors alive again. For
Riget fans this shouldn\'t be so big difference - if the animation
will be done in good way average \'watcher\' will consider it normal
just after first few shots of the movie. The most important thing now
is the story. It\'s completely understandable that it\'s not possible
to create Riget 3 with the actors nowadays. So why not to play with
animation? And... look for the possibilities that it gives to you!
Even marketing one! Great director finishes his trilogy after 10 years
using puppet animation. Just dreams?<br /><br />I hope to see Riget 3
someday... or even to see just the script. I\'m curious how the story
ends... and as I expect- everybody here do.<br /><br />greets, slaj<br
/><br />ps: I\'m not talking about the "kingdom hospital" by Stephen
King ;-)'],
      dtype=object)>, <tf.Tensor: shape=(25000,), dtype=int64,
numpy=array([0, 0, 0, ..., 0, 0, 1])>)

# Viewing the total number of entries
print("Training data entries: {}, Test data entries:
{}".format(len(train_data), len(test_data)))
```

```
Training data entries: 2, Test data entries: 2
```

```
# Checking the datatype of train_labels
print(type(train_labels))
```

```
<class 'numpy.ndarray'>
```

```
# Viewing the train_labels
print(train_labels)
```

```
[0 0 0 ... 0 0 1]
```

```
# Viewing the total number of entries
print("Training label entries: {}, Test label entries:
{}".format(len(train_labels), len(test_labels)))
```

```
Training label entries: 25000, Test label entries: 25000
```

```
# Checking the datatype of train_examples
print(type(train_examples))
```

```
<class 'numpy.ndarray'>
```

```
# Viewing the train_examples
print(train_examples)
```

```
[b"This was an absolutely terrible movie. Don't be lured in by
Christopher Walken or Michael Ironside. Both are great actors, but
this must simply be their worst role in history. Even their great
acting could not redeem this movie's ridiculous storyline. This movie
is an early nineties US propaganda piece. The most pathetic scenes
were those when the Columbian rebels were making their cases for
revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love
affair with Walken was nothing but a pathetic emotional plug in a
movie that was devoid of any real meaning. I am disappointed that
there are movies like this, ruining actor's like Christopher Walken's
good name. I could barely sit through it."
 b'I have been known to fall asleep during films, but this is usually
due to a combination of things including, really tired, being warm and
comfortable on the sette and having just eaten a lot. However on this
occasion I fell asleep because the film was rubbish. The plot
development was constant. Constantly slow and boring. Things seemed to
happen, but with no explanation of what was causing them or why. I
admit, I may have missed part of the film, but i watched the majority
of it and everything just seemed to happen of its own accord without
any real concern for anything else. I cant recommend this film at
all.'
 b'Mann photographs the Alberta Rocky Mountains in a superb fashion,
and Jimmy Stewart and Walter Brennan give enjoyable performances as
they always seem to do. <br /><br />But come on Hollywood - a Mountie
telling the people of Dawson City, Yukon to elect themselves a marshal
```

```
(yes a marshal!) and to enforce the law themselves, then gunfighters
battling it out on the streets for control of the town? <br /><br
/>Nothing even remotely resembling that happened on the Canadian side
of the border during the Klondike gold rush. Mr. Mann and company
appear to have mistaken Dawson City for Deadwood, the Canadian North
for the American Wild West.<br /><br />Canadian viewers be prepared
for a Reefer Madness type of enjoyable howl with this ludicrous plot,
or, to shake your head in disgust.'
 ...
 b'Okay. So I just got back. Before I start my review, let me tell you
one thing: I wanted to like this movie. I know I\'ve been negative in
the past, but I was hoping to be surprised and actually come out
liking the film. I didn\'t.<br /><br />It\'s not just the fact that
every horror clich\xc3\xa9 imaginable is in this. And it\'s not just
the fact that they make every little thing into a jump scare (walking
into a baseball bat left on the floor? Are you kidding me?). It just
wasn\'t scary. One thing I was surprised about: there was more blood
than I thought there was going to be.. which isn\'t saying much.<br
/><br />The film starts off with Donna being dropped off by Lisa\'s
mom at her house. She comes in.. goes upstairs. Camera pans to her
father dead on the couch. Spooky. She goes upstairs, where the
aforementioned baseball bat scene happens. Finds her brother on his
bed, apparently dead (how could she tell? He didn\'t have a spot of
blood on him). Killer comes in, Donna hides under bed, mom dies. She
runs outside screaming for help. Killer behind her: "I did it for us."
Cut to therapy session. This confused a lot of people- everyone was
asking whether or not her family actually died or if she imagined it-
and she mentions how the nightmares have started coming back. Filler
dialogue ensues.<br /><br />THey cut to the chase pretty quick. Few
scenes at the salon, they go to the hotel. Of course the killer is
already there (for some reason, he escaped 3 days ago but the
police/family weren\'t informed until he\'s already there). More
filler ensues.<br /><br />I\'m not going to go on about what happens
in the film, because I don\'t want to spoil it too much. If you want
to know who dies, Horror_Fan made a post about it already. But on the
subjects of deaths: they weren\'t that exciting. People in the theatre
actually laughed out loud (an experience I\'ve never had before in a
horror movie, not even in When A Stranger Calls) during several of
them. One in particular: the bus boy guy who gives the most
hilarious \'scared\' face I\'ve ever seen. The only death involving
any blood was Lisa\'s, and that was pretty scarce. Her throat is
slashed, blood (if you can even call it that- it was practically
black) splatters on the curtain-thing. The only other blood was on
Claire when we see her body. Apparently, Fenton decided to stab her a
few times after he choked her to death. Um, okay? The movie was one of
the most clich\xc3\xa9d I\'ve ever seen. Let\'s see here.. obligatory
close-mirror-curtain-BOOM! scene. Check. Twice, actually (you could
tell they were struggling). Mandatory backing-up-into-killer. Check.
There\'s also the backing-up-into-lamp scene, but you\'ve all seen
```

```
that. Oh, you say you want a birds-flying-away scare? Well, you got
it! (Yes, they managed to incorporate one of those in here). And, of
course, the we-have-security-on-all-exits-but-he-still-escaped scene.
Shall I go on? I could.<br /><br />For anyone saying the characters
weren\'t stupid, are you kidding me? "Oh, even though the massive
alarm is ringing, literally saying PLEASE VACATE THE BUILDING, and 3
of my friends are missing, I\'m going to go upstairs to get my wrap."
These characters were some of the most flawed and stupid characters
ever. The only likable character - Lisa - made one of the most stupid
moves in the movie. "Oh, I just realized the psycho-teacher is here! I
must leave my strong boyfriend behind to run off by myself to warn
her! Oh, shoot, the elevator is being to slow? Guess I\'ll take the
stairs and run off into the construction site!" Ugh. By the end of the
film, they all deserved to die. The only death anyone felt any remorse
for was Donna\'s boyfriend (I can\'t even remember his name- is that
bad?), and by that time, the audience was completely drained out of
this scareless, clich\xc3\xa9d film.<br /><br />There were SOME
positives- the acting was decent for the most part, and it was well-
shot. But that\'s about it.<br /><br />I\'d give it a 1/5, and that\'s
being generous. Just for the laughs (and believe me, the audience had
a few), and Brittany Snow.<br /><br />Oh, and the reaction was bad.
Very bad. People were boo-ing after the movie ended and buzz
afterwards was very negative. Expect bad legs for this one.'
 b'When I saw this trailer on TV I was surprised. In May of 2008 I was
at Six Flags in New Jersey and this was showing at a 4-D attraction
(you know, the attraction that the seats move). I take it that the
version I saw was a shortened version (15 min.) and also re-created to
add the motion effects. It was a cute movie... but that was it. It was
educational and told about the first mission but the ending of a CGI
spacewalk seemed a bit...well...trite. I was not a big fan of the
movie but i would recommend this movie for any parent wanting to
inform their children in a fun way about the first moonwalk. I will
say, the character actors were well selected and the characters
themselves were cute. So all-in-all, I would say, if you want to bring
the younger kids... go for it. But if you are wanting to take your
older kids, take them to another movie... they will thank you.'
 b'First of all, Riget is wonderful. Good comedy and mystery thriller
at the same time. Nice combination of strange \'dogma\' style of
telling the story together with good music and great actors. But
unfortunately there\'s no \'the end\'. As for me it\'s unacceptable. I
was thinking... how it will be possible to continue the story without
Helmer and Drusse? ...and I have some idea. I think Lars should make
RIGET III a little bit different. I\'m sure that 3rd part without
Helmer wouldn\'t be the same. So here\'s my suggestion. Mayble little
bit stupid, maybe not. I know that Lars likes to experiment. So why
not to make small experiment with Riget3? I think the only solution
here is to create puppet-driven animation (like for example "team
America" by Trey Parker) or even computer 3d animation. I know it\'s
not the same as real actors, but in principle I believe it could
```

```
work... only this way it\'s possible to make actors alive again. For
Riget fans this shouldn\'t be so big difference - if the animation
will be done in good way average \'watcher\' will consider it normal
just after first few shots of the movie. The most important thing now
is the story. It\'s completely understandable that it\'s not possible
to create Riget 3 with the actors nowadays. So why not to play with
animation? And... look for the possibilities that it gives to you!
Even marketing one! Great director finishes his trilogy after 10 years
using puppet animation. Just dreams?<br /><br />I hope to see Riget 3
someday... or even to see just the script. I\'m curious how the story
ends... and as I expect- everybody here do.<br /><br />greets, slaj<br
/><br />ps: I\'m not talking about the "kingdom hospital" by Stephen
King ;-)']

# Viewing the total number of entries
print("Training example entries: {}, Test example entries:
{}".format(len(train_examples), len(test_examples)))

Training example entries: 25000, Test example entries: 25000
```

Build the model

```
''' We have to represent the text is to convert sentences into
embeddings vectors. We can use a pre-trained text embedding as the
first layer '''
import tensorflow_hub as hub

model = "https://tfhub.dev/google/nnlm-en-dim50/2" # Setting the URL
for the pre-trained model on TensorFlow Hub

# Creating a Keras layer using the model from TensorFlow Hub
hub_layer = hub.KerasLayer(model,
                           input_shape=[],
                           dtype=tf.string, # input_shape=[] and
dtype=tf.string parameters indicate that this layer will accept string
inputs
                           trainable=True) # weights of this layer
will be updated during training
hub_layer(train_examples[:3])

<tf.Tensor: shape=(3, 50), dtype=float32, numpy=
array([[ 0.5423195 , -0.0119017 ,  0.06337538,  0.06862972, -
0.16776837,
        -0.10581174,  0.16865303, -0.04998824, -0.31148055,
0.07910346,
         0.15442263,  0.01488662,  0.03930153,  0.19772711, -
0.12215476,
        -0.04120981, -0.2704109 , -0.21922152,  0.26517662, -
0.80739075,
         0.25833532, -0.3100421 ,  0.28683215,  0.1943387 , -
```

```
0.29036492,
          0.03862849, -0.7844411 , -0.0479324 ,  0.4110299 , -
0.36388892,
         -0.58034706,  0.30269456,  0.3630897 , -0.15227164, -
0.44391504,
          0.19462997,  0.19528408,  0.05666234,  0.2890704 , -
0.28468323,
         -0.00531206,  0.0571938 , -0.3201318 , -0.04418665, -
0.08550783,
         -0.55847436, -0.23336391, -0.20782952, -0.03543064, -
0.17533456],
        [ 0.56338924, -0.12339553, -0.10862679,  0.7753425 , -
0.07667089,
         -0.15752277,  0.01872335, -0.08169781, -0.3521876 ,  0.4637341
,
         -0.08492756,  0.07166859, -0.00670817,  0.12686075, -
0.19326553,
         -0.52626437, -0.3295823 ,  0.14394785,  0.09043556, -0.5417555
,
          0.02468163, -0.15456742,  0.68333143,  0.09068331, -
0.45327246,
          0.23180096, -0.8615696 ,  0.34480393,  0.12838456, -
0.58759046,
         -0.4071231 ,  0.23061076,  0.48426893, -0.27128142, -0.5380916
,
          0.47016326,  0.22572741, -0.00830663,  0.2846242 , -0.304985
,
          0.04400365,  0.25025874,  0.14867121,  0.40717036, -
0.15422426,
         -0.06878027, -0.40825695, -0.3149215 ,  0.09283665, -
0.20183425],
        [ 0.7456154 ,  0.21256861,  0.14400336,  0.5233862 ,
0.11032254,
          0.00902788, -0.3667802 , -0.08938274, -0.24165542,
0.33384594,
         -0.11194605, -0.01460047, -0.0071645 ,  0.19562712,
0.00685216,
         -0.24886718, -0.42796347,  0.18620004, -0.05241098, -
0.66462487,
          0.13449019, -0.22205497,  0.08633006,  0.43685386,  0.2972681
,
          0.36140734, -0.7196889 ,  0.05291241, -0.14316116, -0.1573394
,
         -0.15056328, -0.05988009, -0.08178931, -0.15569411, -
0.09303783,
         -0.18971172,  0.07620788, -0.02541647, -0.27134508, -0.3392682
,
         -0.10296468, -0.27275252, -0.34078008,  0.20083304, -
0.26644835,
        0.00655449, -0.05141488, -0.04261917, -0.45413622,
0.20023568]],
      dtype=float32)>

model = tf.keras.Sequential() # Sequential models are a stack of
layers where the output of one layer is the input to the next
model.add(hub_layer) # adds the hub_layer you defined earlier to the
model, this layer will transform input text into embeddings
model.add(tf.keras.layers.Dense(16, activation='relu')) # adds a
densely connected layer with 16 units to the model, the relu
activation function is used
model.add(tf.keras.layers.Dense(1)) # adds another dense layer with a
single unit. By default, this layer will use a linear activation
function

model.summary() # summary of the model, including the number of
parameters and the output shapes of each layer

''' Hidden Units: The above model has two intermediate or "hidden"
layers, between the input and output. The number of outputs (units,
nodes, or neurons) is the dimension of the representational
space for the layer. In other words, the amount of freedom the network
is allowed when learning an internal representation.
If a model has more hidden units (a higher-dimensional representation
space), and/or more layers, then the network can learn more complex
representations.'''

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 keras_layer (KerasLayer)    (None, 50)                48190600  
                                                                 
 dense_2 (Dense)             (None, 16)                816       
                                                                 
 dense_3 (Dense)             (None, 1)                 17        
                                                                 
=================================================================
Total params: 48191433 (183.84 MB)
Trainable params: 48191433 (183.84 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

{"type":"string"}
```

Loss function and optimizer (Model Compile)

Model compilation is an activity performed after writing the statements in a model and before training starts. It checks for format errors, and defines the loss function, the optimizer or learning rate, and the metrics. A compiled model is needed for training but not necessary for predicting.

Model needs a loss function and an optimizer for training. Since this is a binary classification problem and the model outputs a probability (a single-unit layer with a sigmoid activation), we'll use the binary_crossentropy loss function.

This isn't the only choice for a loss function, you could, for instance, choose mean_squared_error. But, generally, binary_crossentropy is better for dealing with probabilities —it measures the "distance" between probability distributions, or in our case, between the ground-truth distribution and the predictions.

```
model.compile(optimizer='adam', # optimizer is the algorithm that adjusts the weights and biases during training to minimize the loss function
              loss=tf.losses.BinaryCrossentropy(from_logits=True), # Sets the loss function for your model to binary cross-entropy. This argument means that the function expects raw, unscaled values
              # The loss function measures how well the model did on each example, and tries to steer it towards the correct answer
              metrics=[tf.metrics.BinaryAccuracy(threshold=0.0, name='accuracy')]) # Sets the metric for your model to binary accuracy with a threshold of 0.0

results = model.evaluate(test_examples, test_labels) # Only to see the accuracy. No needed right now

print(results)
```

```
782/782 [==============================] - 4s 5ms/step - loss: 0.6851 - accuracy: 0.5434
[0.685131847858429, 0.543399998960495]
```

Create a validation set

Create a validation set by setting apart 10,000 examples from the original training data

Why not use the testing set now?

Our goal is to develop and tune our model using only the training data, then use the test data just once to evaluate our accuracy

```
x_val = train_examples[:10000] # selects the first 10,000 examples from your training data
partial_x_train = train_examples[10000:] # selects all examples after the first 10,000 from your training data

y_val = train_labels[:10000]# selects the first 10,000 labels from your training labels
partial_y_train = train_labels[10000:] # selects all labels after the first 10,000 from your training labels
```

Train the model

```
text_classify = model.fit(partial_x_train, # model will learn from
this data
                 partial_y_train, # model will learn from this data
                 epochs=40, # number of times the learning algorithm
will work through the entire training dataset
                 batch_size=512, # number of samples per gradient
update
                 validation_data=(x_val, y_val), # after each epoch,
the model will evaluate its performance on this validation data
                 verbose=1) # model will output detailed information
about the training process, including the number of the current epoch
and the loss and accuracy on the training and validation data
#  The number of batches (30 in this case) depends on the size of your
training data and the batch size you have set
# So, otal number of training data is 512 * 30 = 15360. 15,360
training examples in dataset

Epoch 1/40
30/30 [==============================] - 46s 2s/step - loss: 0.6337 -
accuracy: 0.6879 - val_loss: 0.5768 - val_accuracy: 0.7568
Epoch 2/40
30/30 [==============================] - 45s 2s/step - loss: 0.5069 -
accuracy: 0.8019 - val_loss: 0.4633 - val_accuracy: 0.8113
Epoch 3/40
30/30 [==============================] - 46s 2s/step - loss: 0.3758 -
accuracy: 0.8658 - val_loss: 0.3768 - val_accuracy: 0.8461
Epoch 4/40
30/30 [==============================] - 45s 1s/step - loss: 0.2732 -
accuracy: 0.9071 - val_loss: 0.3374 - val_accuracy: 0.8590
Epoch 5/40
30/30 [==============================] - 45s 2s/step - loss: 0.2006 -
accuracy: 0.9365 - val_loss: 0.3084 - val_accuracy: 0.8736
Epoch 6/40
30/30 [==============================] - 49s 2s/step - loss: 0.1466 -
accuracy: 0.9589 - val_loss: 0.3004 - val_accuracy: 0.8761
Epoch 7/40
30/30 [==============================] - 66s 2s/step - loss: 0.1060 -
accuracy: 0.9743 - val_loss: 0.3018 - val_accuracy: 0.8763
Epoch 8/40
30/30 [==============================] - 51s 2s/step - loss: 0.0757 -
accuracy: 0.9850 - val_loss: 0.3073 - val_accuracy: 0.8760
Epoch 9/40
30/30 [==============================] - 49s 2s/step - loss: 0.0538 -
accuracy: 0.9923 - val_loss: 0.3171 - val_accuracy: 0.8742
Epoch 10/40
30/30 [==============================] - 55s 2s/step - loss: 0.0383 -
accuracy: 0.9966 - val_loss: 0.3288 - val_accuracy: 0.8746
Epoch 11/40
30/30 [==============================] - 51s 2s/step - loss: 0.0277 -
accuracy: 0.9986 - val_loss: 0.3420 - val_accuracy: 0.8713
```

```
Epoch 12/40
30/30 [==============================] - 46s 2s/step - loss: 0.0205 -
accuracy: 0.9991 - val_loss: 0.3533 - val_accuracy: 0.8715
Epoch 13/40
30/30 [==============================] - 47s 2s/step - loss: 0.0157 -
accuracy: 0.9994 - val_loss: 0.3651 - val_accuracy: 0.8716
Epoch 14/40
30/30 [==============================] - 47s 2s/step - loss: 0.0121 -
accuracy: 0.9997 - val_loss: 0.3765 - val_accuracy: 0.8703
Epoch 15/40
30/30 [==============================] - 47s 2s/step - loss: 0.0095 -
accuracy: 0.9999 - val_loss: 0.3873 - val_accuracy: 0.8693
Epoch 16/40
30/30 [==============================] - 46s 2s/step - loss: 0.0076 -
accuracy: 0.9999 - val_loss: 0.3976 - val_accuracy: 0.8688
Epoch 17/40
30/30 [==============================] - 45s 2s/step - loss: 0.0062 -
accuracy: 1.0000 - val_loss: 0.4067 - val_accuracy: 0.8687
Epoch 18/40
30/30 [==============================] - 45s 2s/step - loss: 0.0052 -
accuracy: 1.0000 - val_loss: 0.4153 - val_accuracy: 0.8688
Epoch 19/40
30/30 [==============================] - 44s 1s/step - loss: 0.0044 -
accuracy: 1.0000 - val_loss: 0.4238 - val_accuracy: 0.8687
Epoch 20/40
30/30 [==============================] - 60s 2s/step - loss: 0.0038 -
accuracy: 1.0000 - val_loss: 0.4313 - val_accuracy: 0.8691
Epoch 21/40
30/30 [==============================] - 46s 2s/step - loss: 0.0033 -
accuracy: 1.0000 - val_loss: 0.4388 - val_accuracy: 0.8689
Epoch 22/40
30/30 [==============================] - 45s 2s/step - loss: 0.0029 -
accuracy: 1.0000 - val_loss: 0.4463 - val_accuracy: 0.8689
Epoch 23/40
30/30 [==============================] - 46s 2s/step - loss: 0.0026 -
accuracy: 1.0000 - val_loss: 0.4526 - val_accuracy: 0.8683
Epoch 24/40
30/30 [==============================] - 47s 2s/step - loss: 0.0023 -
accuracy: 1.0000 - val_loss: 0.4594 - val_accuracy: 0.8686
Epoch 25/40
30/30 [==============================] - 46s 2s/step - loss: 0.0020 -
accuracy: 1.0000 - val_loss: 0.4659 - val_accuracy: 0.8682
Epoch 26/40
30/30 [==============================] - 45s 2s/step - loss: 0.0018 -
accuracy: 1.0000 - val_loss: 0.4714 - val_accuracy: 0.8681
Epoch 27/40
30/30 [==============================] - 45s 1s/step - loss: 0.0017 -
accuracy: 1.0000 - val_loss: 0.4774 - val_accuracy: 0.8680
Epoch 28/40
```

```
30/30 [==============================] - 45s 2s/step - loss: 0.0015 -
accuracy: 1.0000 - val_loss: 0.4822 - val_accuracy: 0.8674
Epoch 29/40
30/30 [==============================] - 45s 2s/step - loss: 0.0014 -
accuracy: 1.0000 - val_loss: 0.4875 - val_accuracy: 0.8680
Epoch 30/40
30/30 [==============================] - 46s 2s/step - loss: 0.0013 -
accuracy: 1.0000 - val_loss: 0.4923 - val_accuracy: 0.8675
Epoch 31/40
30/30 [==============================] - 46s 2s/step - loss: 0.0012 -
accuracy: 1.0000 - val_loss: 0.4964 - val_accuracy: 0.8673
Epoch 32/40
30/30 [==============================] - 46s 2s/step - loss: 0.0011 -
accuracy: 1.0000 - val_loss: 0.5011 - val_accuracy: 0.8674
Epoch 33/40
30/30 [==============================] - 61s 2s/step - loss: 0.0010 -
accuracy: 1.0000 - val_loss: 0.5059 - val_accuracy: 0.8670
Epoch 34/40
30/30 [==============================] - 45s 2s/step - loss: 9.2657e-
04 - accuracy: 1.0000 - val_loss: 0.5102 - val_accuracy: 0.8669
Epoch 35/40
30/30 [==============================] - 46s 2s/step - loss: 8.6197e-
04 - accuracy: 1.0000 - val_loss: 0.5141 - val_accuracy: 0.8671
Epoch 36/40
30/30 [==============================] - 45s 2s/step - loss: 8.0488e-
04 - accuracy: 1.0000 - val_loss: 0.5181 - val_accuracy: 0.8666
Epoch 37/40
30/30 [==============================] - 46s 2s/step - loss: 7.5150e-
04 - accuracy: 1.0000 - val_loss: 0.5221 - val_accuracy: 0.8665
Epoch 38/40
30/30 [==============================] - 46s 2s/step - loss: 7.0316e-
04 - accuracy: 1.0000 - val_loss: 0.5262 - val_accuracy: 0.8666
Epoch 39/40
30/30 [==============================] - 45s 2s/step - loss: 6.5933e-
04 - accuracy: 1.0000 - val_loss: 0.5301 - val_accuracy: 0.8662
Epoch 40/40
30/30 [==============================] - 45s 2s/step - loss: 6.2038e-
04 - accuracy: 1.0000 - val_loss: 0.5339 - val_accuracy: 0.8663
```

```
print(type(text_classify))
```

```
<class 'keras.src.callbacks.History'>
```

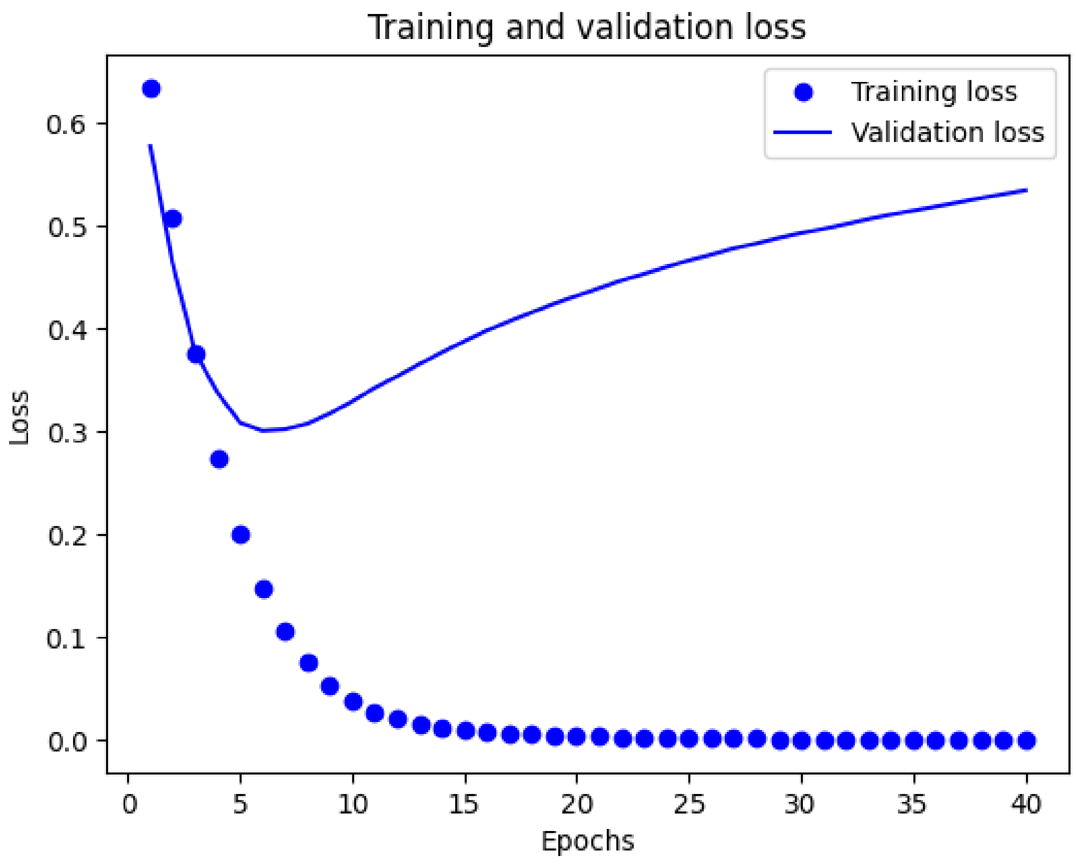
Evaluate the model

```
results = model.evaluate(test_examples, test_labels)

print(results)
```

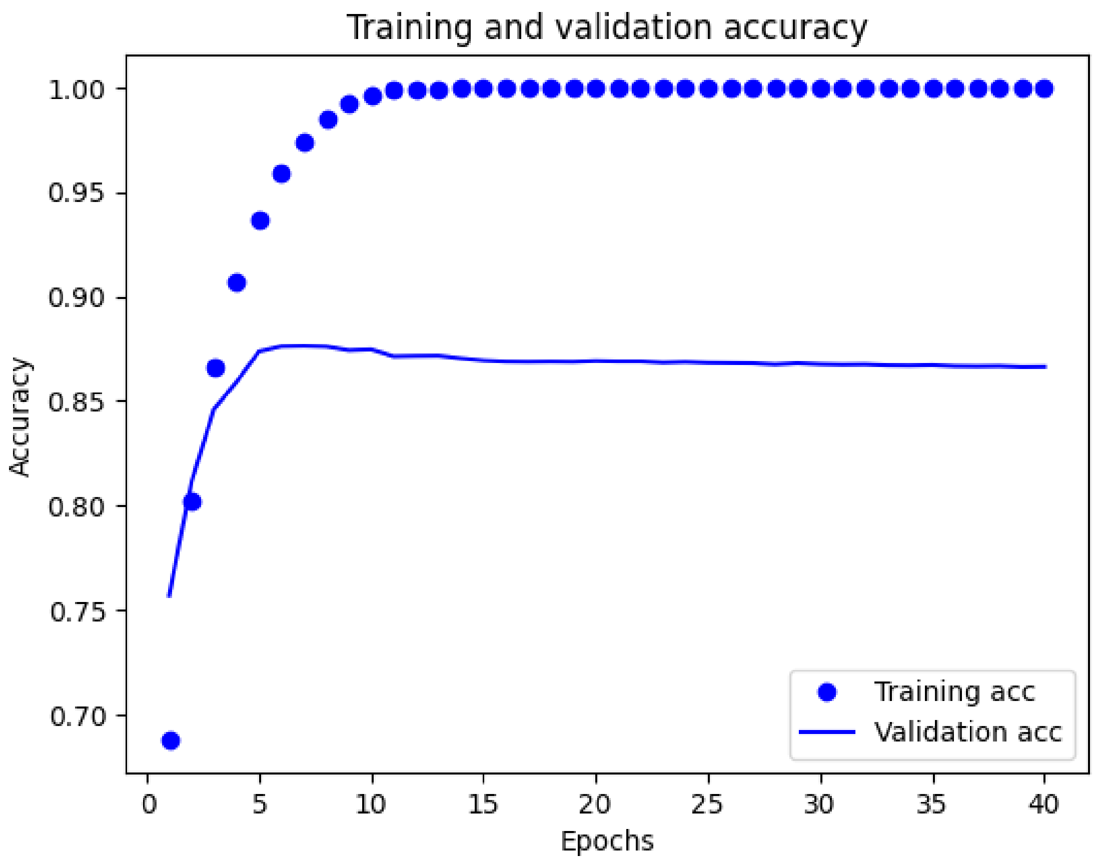
```
782/782 [==============================] - 143s 183ms/step - loss:
0.6003 - accuracy: 0.8471
[0.6002957820892334, 0.8470799922943115]
```

Create a graph of accuracy and loss over time

```
''' When you train a model in Keras using the fit() function, it
returns a History object. This object has a member history, which is a
dictionary. The keys of this dictionary could be the
 loss and accuracy of the model on the training and validation sets,
depending on your model and the parameters you passed to the fit()
function '''
text_dict = text_classify.history # .history is a dictionary
containing data about everything that happened during training

text_dict.keys()

acc = text_dict['accuracy'] # extracting the training accuracy for
each epoch from the dictionary
val_acc = text_dict['val_accuracy'] # extracting the validation
accuracy for each epoch from the dictionary
loss = text_dict['loss'] # extracting the training loss for each epoch
from the dictionary
val_loss = text_dict['val_loss'] # extracting the validation loss for
each epoch from the dictionary

epochs = range(1, len(acc) + 1) # creating a range of numbers from 1
to the number of epochs (which is the length of the acc list), and
storing it in the variable epochs

# Training and validation loss

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

plt.show()
```

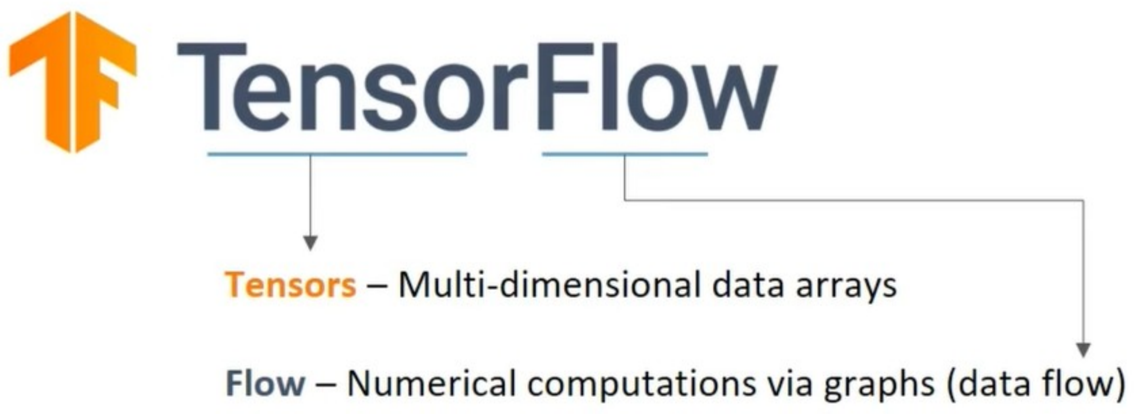


```python
# Training and validation accuracy

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

plt.show()
```



```
''' In this model we can ovserve overfitting: the model performs
better on the training data. After this point, the model over-
optimizes and learns representations specific to the training data
 that do not generalize to test data

For this particular case, we could prevent overfitting by simply
stopping the training after 20 or so epochs'''

{"type":"string"}
```

Some imp concepts of TensorFlow



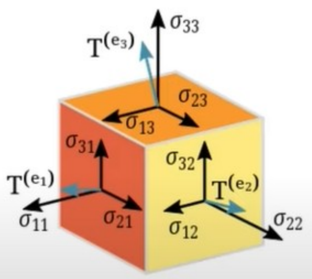
Tensors – Multi-dimensional data arrays

Flow – Numerical computations via graphs (data flow)

A flow of tensors which is fundamental to the working of a Neural Network.

What are Tensors?

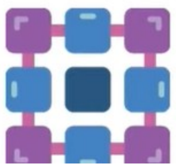
Think of it this way:



Tensors are objects that describe a relationship between algebraic objects and vector space associated to them.

What are Tensors?

Think of it this way:



For **scalars**, you will need only one number for 'n' dimensions

For **Tensors**, you will need n^r where 'r' is the rank of a tensor.

Constant

A constant in TensorFlow is a tensor whose value cannot be changed. It is initialized when it is created and cannot be modified afterwards. Constants are created using the tf.constant function.

```
# Creating and running computational graph from tensors
a = tf.constant([1, 2])
b = tf.constant([3, 4])

c = (a + b)
print(c)
```

```
tf.Tensor([4 6], shape=(2,), dtype=int32)
```

```
import tensorflow as tf

# Create a constant tensor
a = tf.constant([1, 2])
print(a)
```

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

Variables

A TensorFlow variable is a tensor whose value can be changed by running operations on it. Variables are created and tracked via the tf.Variable class. They are used to represent shared, persistent state that your program manipulates.

```
# Create a variable tensor
a = tf.Variable([1, 2])
print(a)
```

```
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2],
dtype=int32)>
```

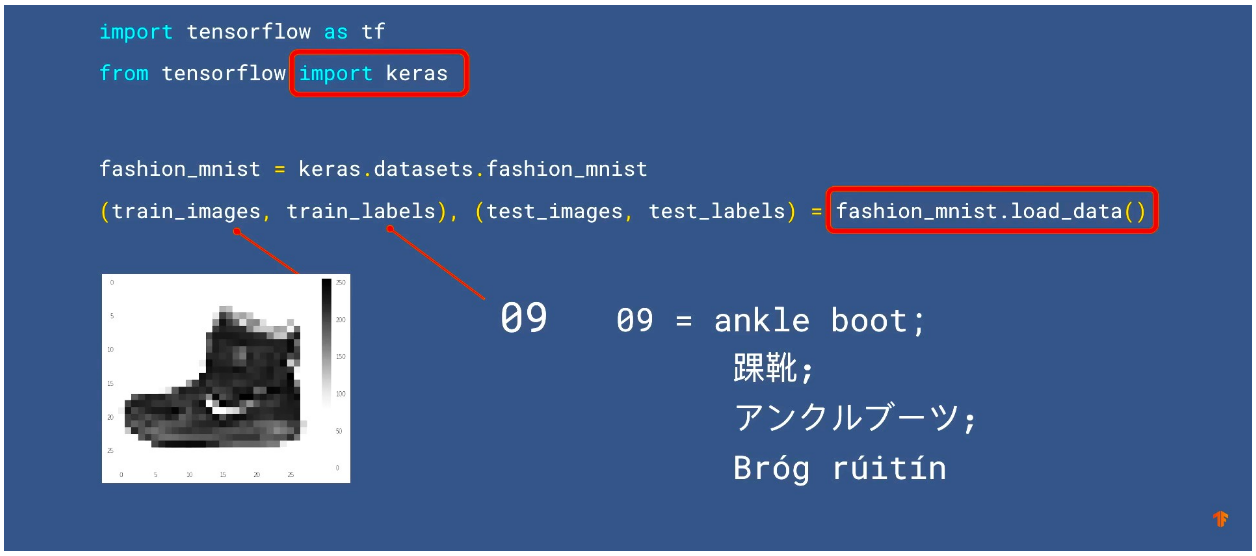
Placeholders

A placeholder in TensorFlow is a variable to which data will be assigned later. It allows us to create operations and build our computation graph without needing the data. Data is fed into the placeholder when the session starts. However, please note that placeholders have been deprecated in TensorFlow 2.0 and are only available in TensorFlow 1.x.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

# Create a placeholder
a = tf.placeholder(tf.float32, shape=(None,))
print(a)
```

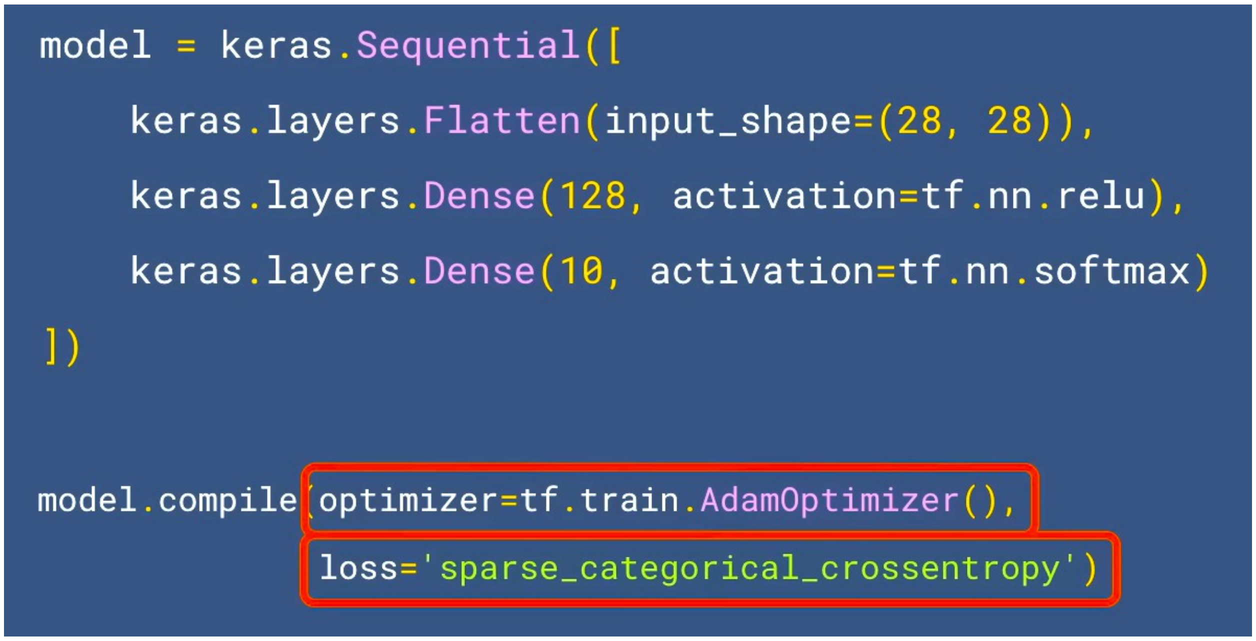
```
Tensor("Placeholder_1:0", shape=(?,), dtype=float32)
```

TensorFlow for ANN

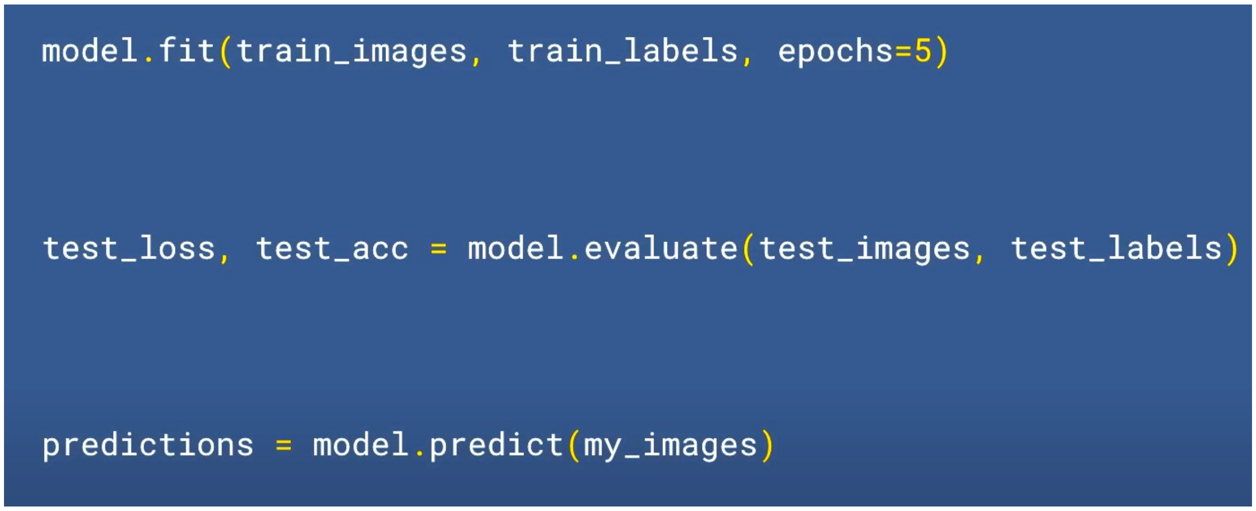


```
import tensorflow as tf
from tensorflow import keras

fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

09   09 = ankle boot;
          踝靴;
          アンクルブーツ;
          Bróg rúitín

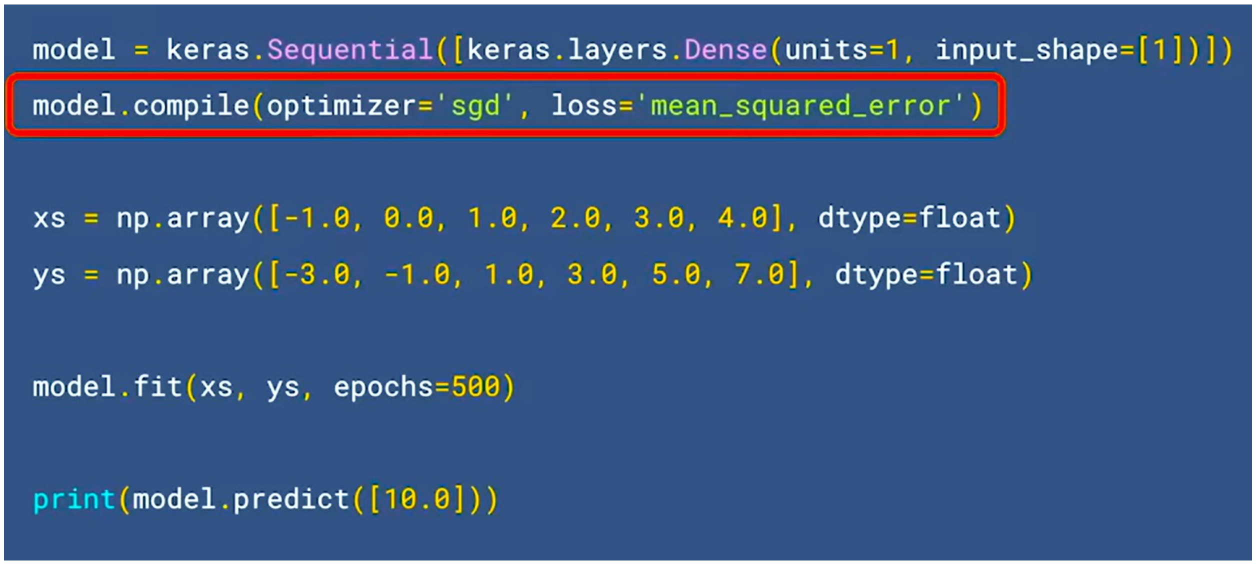


```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy')
```



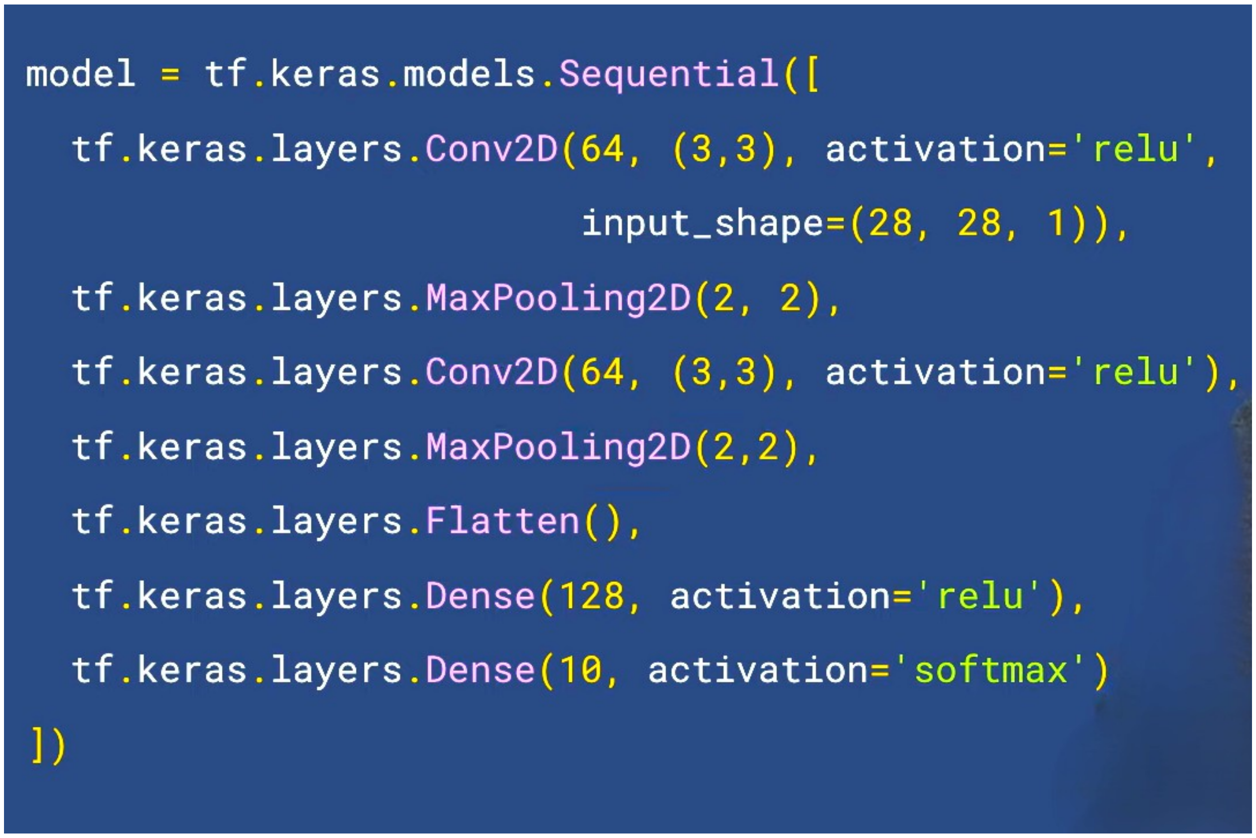
```
model.fit(train_images, train_labels, epochs=5)


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


predictions = model.predict(my_images)
```

Simple Linear-Regression model in ANN



```
model = keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(xs, ys, epochs=500)

print(model.predict([10.0]))
```

TensorFlow for CNN



```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(64, (3,3), activation='relu',
                         input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```