Project Overview

Google AI-ML Course Internship Completion Report

1. Introduction

- **Internship Name**: Google AI-ML Course
- **Duration**: [Start Date] to [End Date]
- **Platform**: Google Developers
- **Objective**: The goal of this course was to gain proficiency in AI and ML concepts through practical coding exercises, quizzes, and interactive labs using TensorFlow and other tools. This report outlines the key units completed, skills acquired, and badges earned.

2. Course Overview

The Google AI-ML course was designed to provide comprehensive knowledge of neural networks, object detection, image classification, and product image search using TensorFlow. The course was divided into multiple units, with each unit involving hands-on labs and quizzes. Successful completion of each unit was rewarded with a badge.

Units Completed:

- **UNIT-1**: Program neural networks with TensorFlow
- **UNIT-2**: Get started with object detection
- **UNIT-3**: Go further with object detection
- **UNIT-4**: Get started with product image search
- **UNIT-5**: Go further with product image search
- **UNIT-6**: Go further with image classification

3. Steps to Access and Complete the Units

Unit 1: Program neural networks with TensorFlow

- **Link**: Program neural networks with TensorFlow
- Completed all stages, followed by a quiz.

- **Badge Earned**: **Program neural networks with TensorFlow**

Unit 2: Get started with object detection

- **Link**: Get started with object detection
- Completed all stages, followed by a quiz.
- **Badge Earned**: **Get started with object detection**

Unit 3: Go further with object detection

- **Link**: Go further with object detection
- Completed all stages, followed by a quiz.
- **Badge Earned**: **Go further with object detection**

Unit 4: Get started with product image search

- **Link**: Get started with product image search
- Completed all stages, followed by a quiz.
- **Badge Earned**: **Get started with product image search**

Unit 5: Go further with product image search

- **Link**: Go further with product image search
- Completed all stages, followed by a quiz.
- **Badge Earned**: **Go further with product image search**

Unit 6: Go further with image classification

- **Link**: Go further with image classification
- Completed all stages, followed by a quiz.
- **Badge Earned**: **Go further with image classification**

4. Google Developer Profile Creation

Once all the units were completed, a custom Google Developer Profile URL was created to showcase the badges earned.

Steps to Create Profile:

1. Enter your name and college name (e.g., "Ravi Kumar from Ram College") and save your profile.
2. A custom web address was created to display the earned badges, which can be used for future reference or for internship evaluations.

Example URL: https://g.dev/ravikumar_ramcollege
(Note: The above URL is an example and does not exist.)

5. Badges Earned

Throughout the course, the following badges were successfully earned:

1. **Program neural networks with TensorFlow**
2. **Get started with object detection**
3. **Go further with object detection**
4. **Get started with product image search**
5. **Go further with product image search**
6. **Go further with image classification**

These badges are now displayed on my Google Developer Profile, demonstrating my competencies in AI-ML concepts and practical application using TensorFlow.

Unit 1 - Program Neural Networks with TensorFlow

Lesson: The Hello, World of Machine Learning

Overview

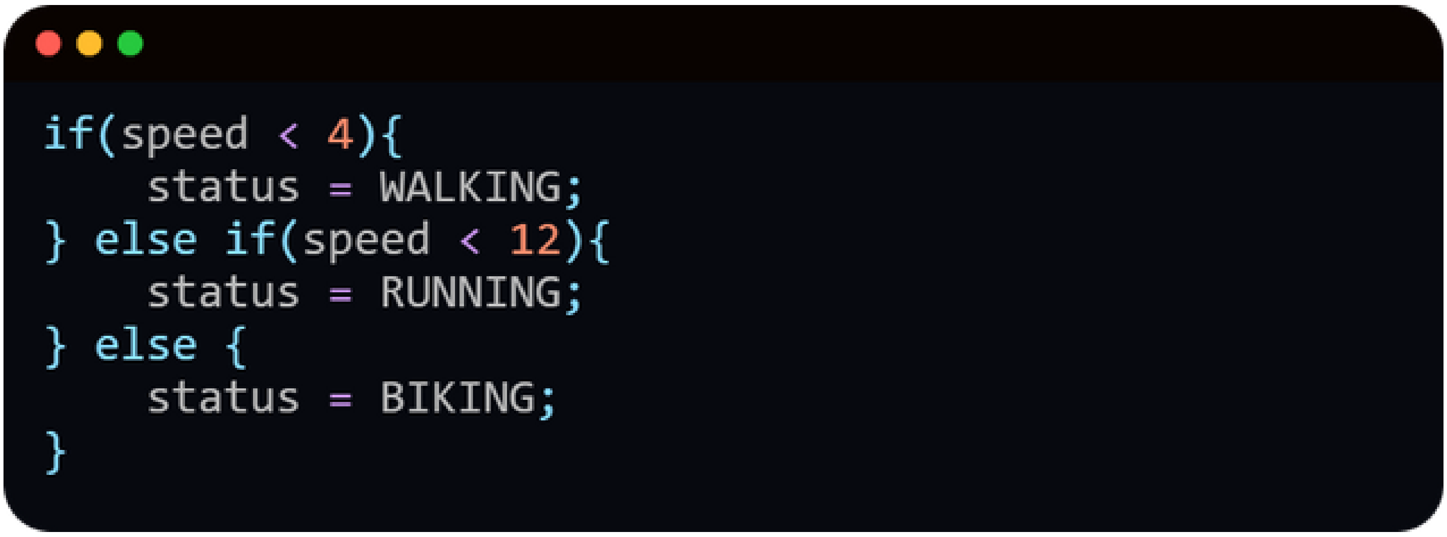
In this lesson, learners are introduced to the basic principles of machine learning (ML), particularly how ML can infer rules from data rather than relying on explicitly programmed logic. This is achieved through creating a simple model that recognizes activities such as walking, running, and biking, based on speed data.

What You'll Learn:

- The basics of machine learning.
- How ML models are trained to infer patterns from data.
- The difference between traditional programming rules and ML-based inference.

Example: Traditional Conditional Programming vs. Machine Learning

Traditional programming relies on explicitly written rules. For example:



```
if(speed < 4){
    status = WALKING;
} else if(speed < 12){
    status = RUNNING;
} else {
    status = BIKING;
}
```

However, when faced with a more complex activity such as **golfing**, writing conditions to infer this activity is much harder, if not impossible.

Solution: Machine Learning

Instead of writing a complex set of conditional rules, machine learning enables you to build a model that can "learn" from a dataset. You train the system to infer rules and relationships between the input data (speed) and the output activity (walking, running, biking, or golfing).

Requirements:

Before starting this codelab, you should have:

- Basic programming knowledge, particularly in **Python**.

- Familiarity with libraries such as **NumPy** and **TensorFlow**.

Tools:

- **Google Colaboratory (Colab)**: A browser-based IDE that already has TensorFlow and other required dependencies installed. This makes it easy to start building and running ML models without any local installation.
- **IDE (Optional)**: If you prefer using your own environment, ensure that **Python**, **TensorFlow**, and **NumPy** are installed.

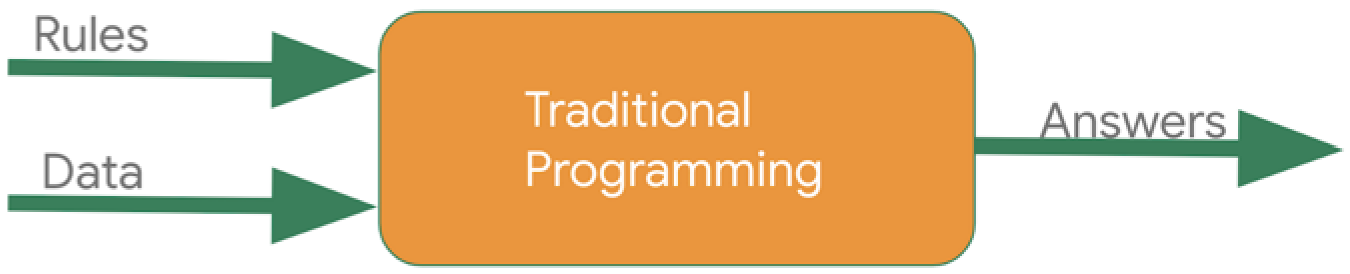
Next Steps:

- Start coding your first machine learning model by following the Colab notebook provided. This lesson will guide you through creating a simple neural network that recognizes patterns in the data.

2. What is ML?

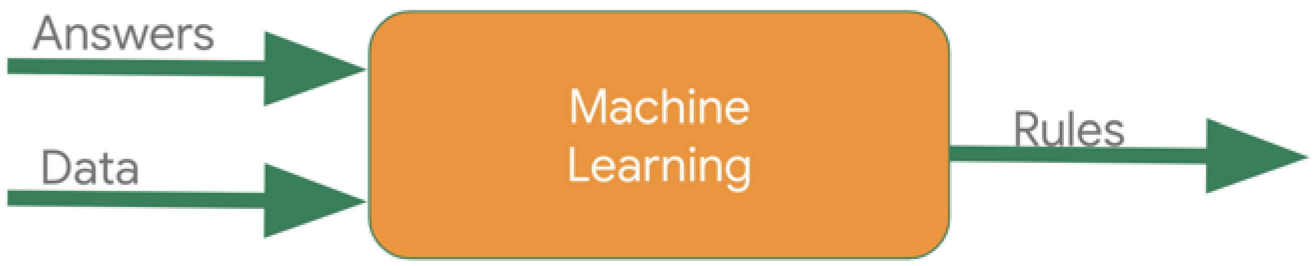
In traditional app development, you express rules in a programming language, which act on data to provide answers. For example, in activity detection, the rules (code defining activity types) process the data (movement speed) to determine the activity status (walking, running, biking, etc.).

(Traditional Programming Diagram)



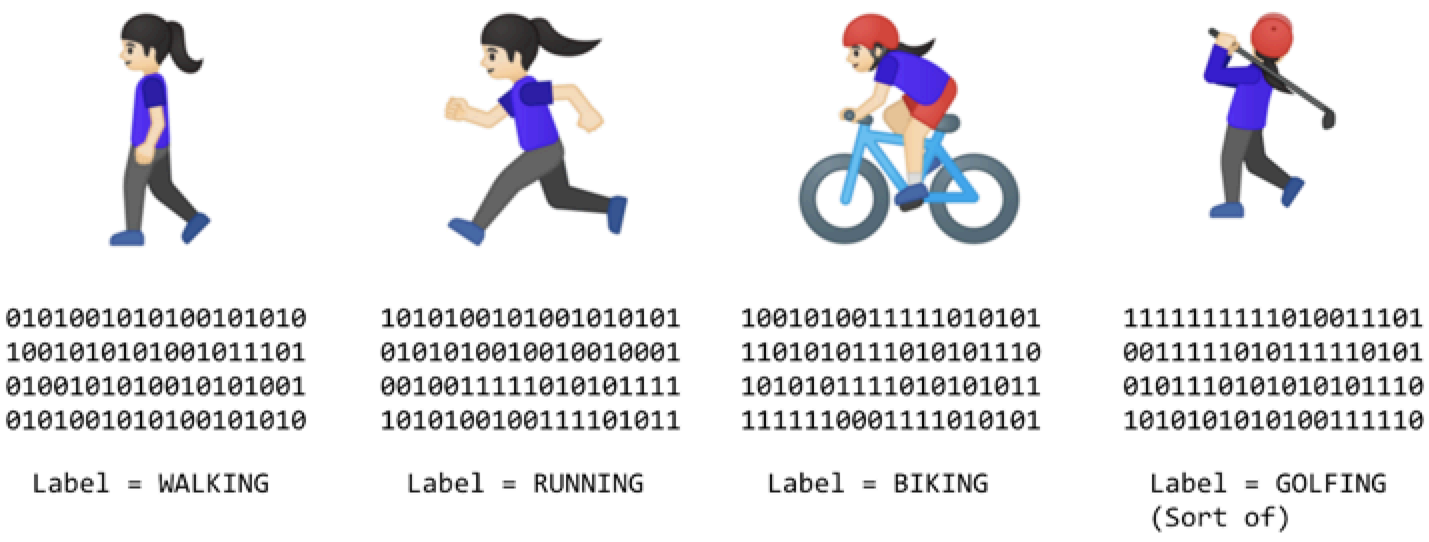
In machine learning (ML), the approach is different. Instead of defining explicit rules, you provide data and labels (answers), allowing the machine to infer the rules that establish the relationship between data and answers.

(ML Process Diagram)



For instance, in activity detection with ML, you gather and label data to indicate what walking or running looks like. The machine learns the patterns from this data to infer the rules that define each activity.

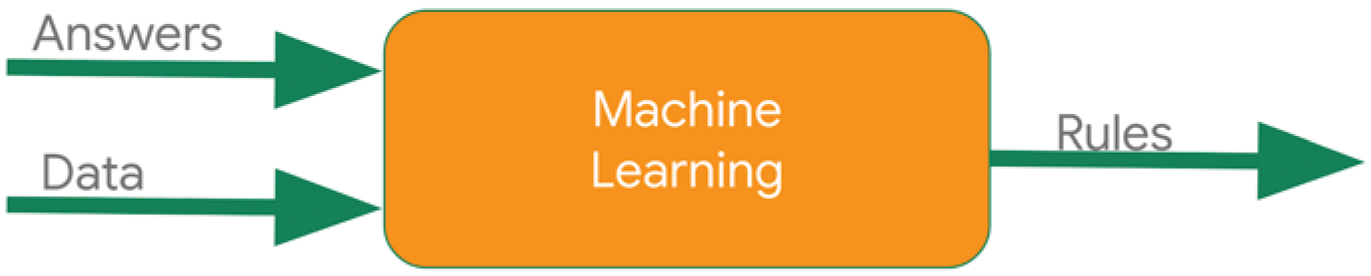
(ML Activity Detection Diagram)



This approach not only offers an alternative to traditional programming but also opens new possibilities for scenarios that might be complex to handle with predefined rules.

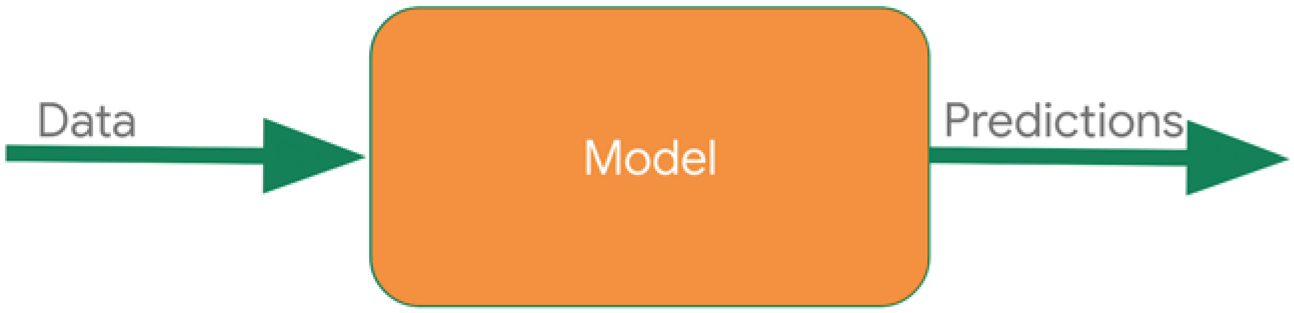
In traditional programming, your code compiles into a binary program. In ML, the result from data and labels is a model.

(Traditional Programming vs. ML Model Diagram)



The model is used at runtime by passing data through it, allowing the model to make predictions based on the rules it inferred during training.

(Model Prediction Diagram)



3. Create Your First ML Model

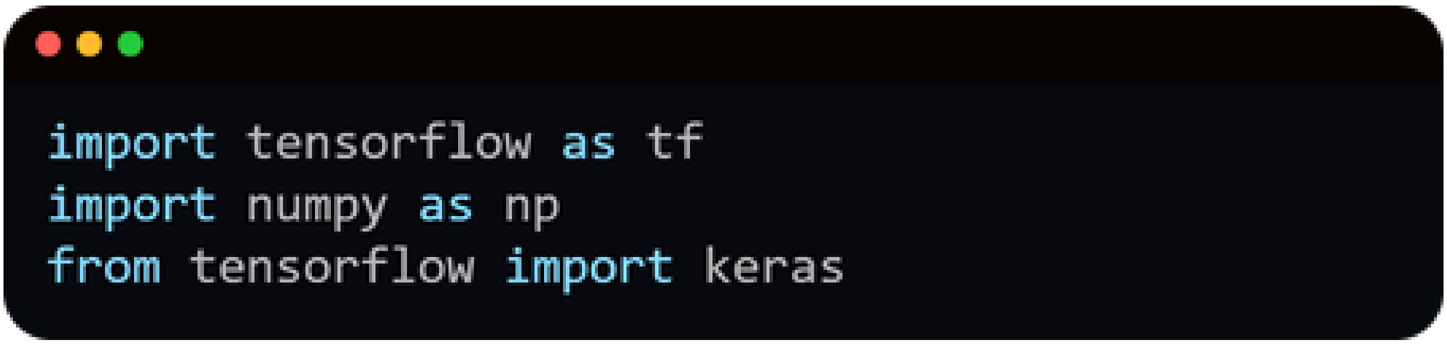
Objective: Train a neural network to identify the relationship between sets of numbers.

Data:

X:	-1	0	1	2	3	4
Y:	2	1	4	7	10	13

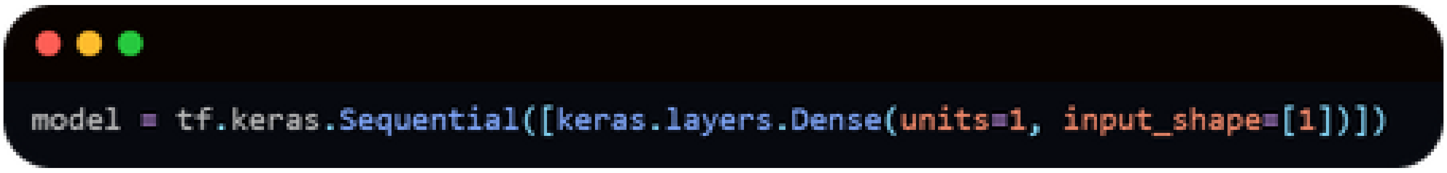
As you look at them, you might notice that the value of X is increasing by 1 as you read left to right and the corresponding value of Y is increasing by 3. You probably think that Y equals 3X plus or minus something. Then, you'd probably look at the 0 on X and see that Y is 1, and you'd come up with the relationship Y=3X+1.

In machine learning, this relationship can be discovered by training a neural network rather than coding explicit rules. Begin by setting up your Python environment with TensorFlow, NumPy, and Keras:



```python
import tensorflow as tf
import numpy as np
from tensorflow import keras
```

Define a simple neural network using Keras. The network should have one layer with one neuron and an input shape of one value:



```python
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
```

Compile the model by choosing an optimizer and a loss function. For this task, use stochastic gradient descent (sgd) as the optimizer and mean squared error as the loss function:

```
model.compile(optimizer='sgd', loss='mean_squared_error')
```

Prepare your data using NumPy arrays:

```
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-2.0, 1.0, 4.0, 7.0, 10.0, 13.0], dtype=float)
```

Feed this data into the model and train it. The neural network will learn the pattern between X and Y values, ideally discovering the formula Y = 3X + 1. This approach enables the model to infer the underlying relationship from the data rather than relying on manually written rules.

4. Train the Neural Network

To train the neural network, use the model.fit function, which adjusts the model's parameters over a specified number of epochs. During this process, the model makes predictions, calculates the error (loss), and updates its parameters to minimize the loss.

Here's how to train the model:

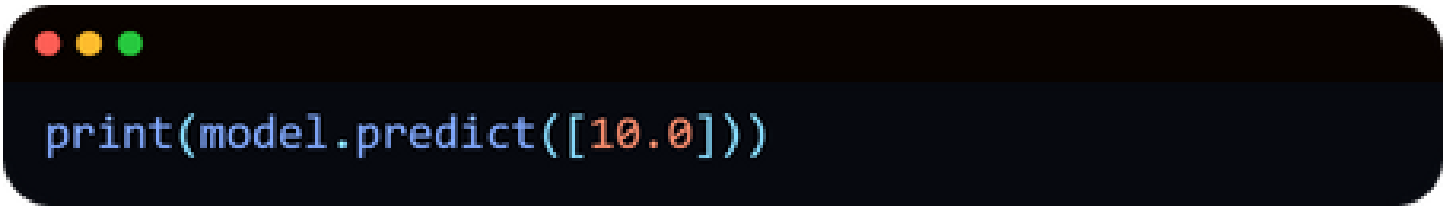
```
model.fit(xs, ys, epochs=500)
```

Initially, the loss value might be high, but it should decrease as training progresses. After a few epochs, the loss will become quite small, indicating that the model is learning the relationship between the X and Y values effectively.

You might not need all 500 epochs; in many cases, fewer epochs will be enough. For example, the model might achieve satisfactory performance after just 50 epochs.

5. Use the model

You have a model that has been trained to learn the relationship between X and Y. You can use the model.predict method to have it figure out the Y for a previously unknown X. For example, if X is 10, what do you think Y will be? Take a guess before you run the following code:



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

You might have thought 31, but it ended up being a little over. Why do you think that is?

Neural networks deal with probabilities, so it calculated that there is a very high probability that the relationship between X and Y is Y=3X+1, but it can't know for sure with only six data points. The result is very close to 31, but not necessarily 31.

As you work with neural networks, you'll see that pattern recurring. You will almost always deal with probabilities, not certainties, and will do a little bit of coding to figure out what the result is based on the probabilities, particularly when it comes to classification.

Lesson: Build a Computer Vision Model with TensorFlow

About This Codelab

In this codelab, you’ll create a computer vision model to recognize clothing items using TensorFlow. This guide will walk you through training a neural network with the Fashion MNIST dataset, including model design, training, and evaluation.

Prerequisites

- Solid knowledge of Python
- Basic programming skills

What You’ll Learn

- Train a neural network to recognize clothing articles
- Experiment with different layers of the network

What You’ll Build

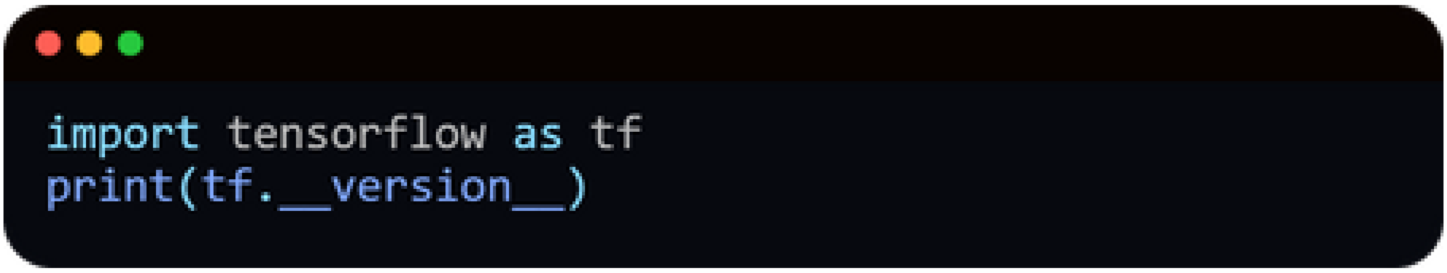
- A neural network that identifies articles of clothing

What You’ll Need

- **Colaboratory**: For a browser-based environment with all required dependencies.
- **Python**: If using your local setup, ensure Python, TensorFlow, and NumPy are installed. Install TensorFlow and NumPy.

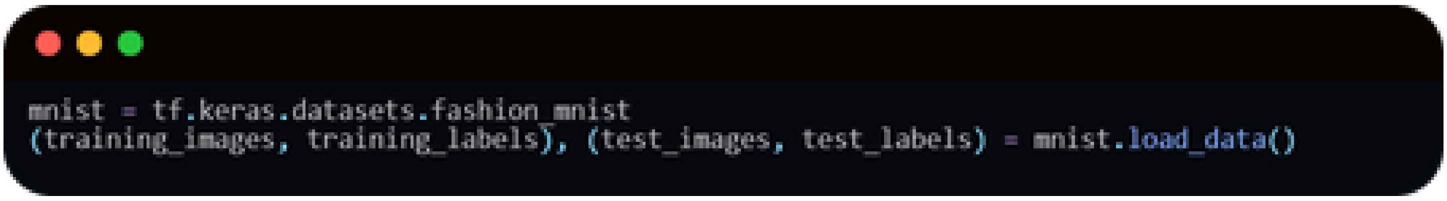
1. Start Coding

Begin by importing TensorFlow and checking its version:



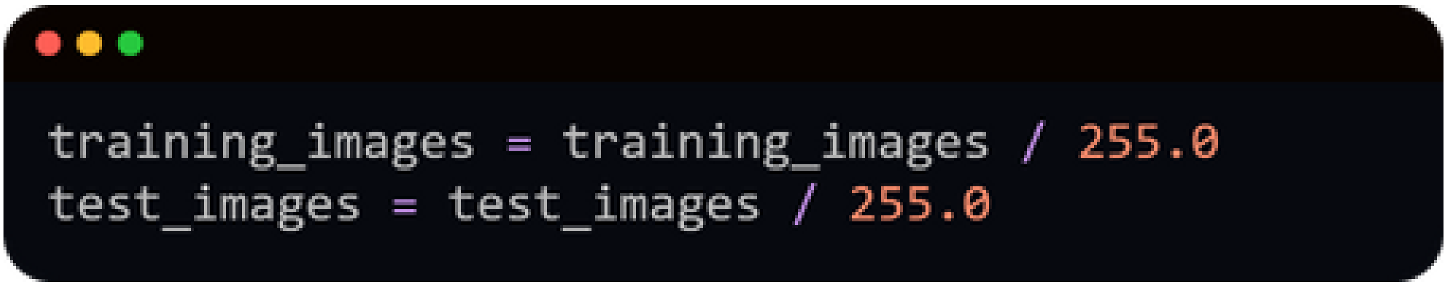
```python
import tensorflow as tf
print(tf.__version__)
```

Load the Fashion MNIST dataset, which contains 70,000 28x28 grayscale images in 10 categories:



```python
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
```

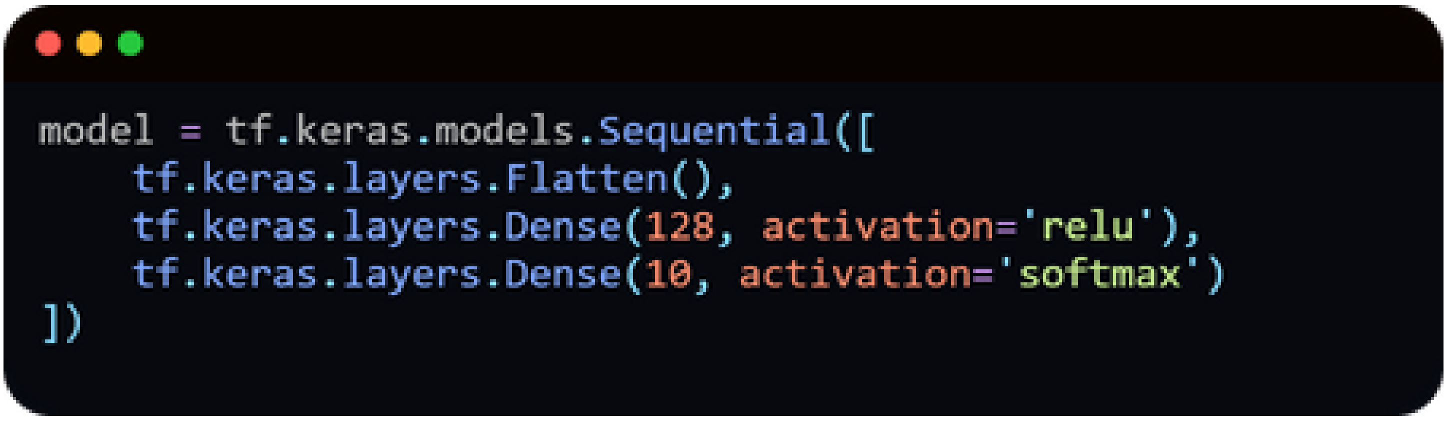
Normalize the image data by scaling pixel values to between 0 and 1:



```python
training_images = training_images / 255.0
test_images = test_images / 255.0
```

2. Design the Model

Create a neural network with three layers:

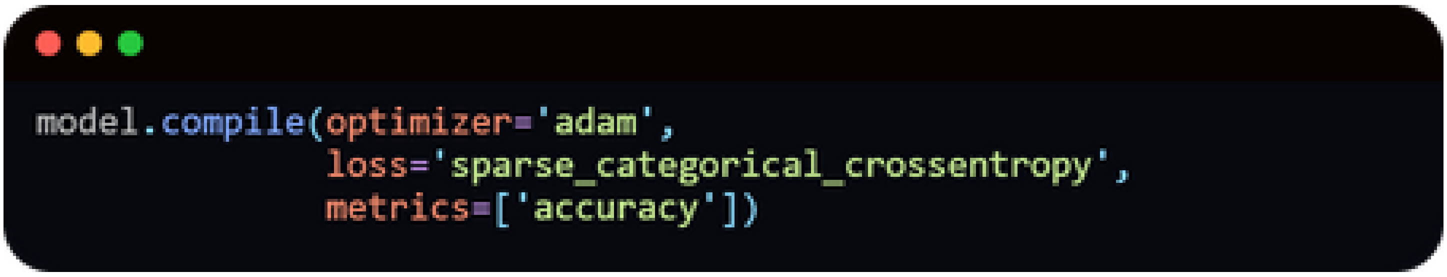


```python
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

- **Flatten**: Converts the 28x28 image into a 1D vector.
- **Dense**: Fully connected layers. The first layer has 128 neurons with ReLU activation, and the second has 10 neurons with Softmax activation to predict 10 classes.

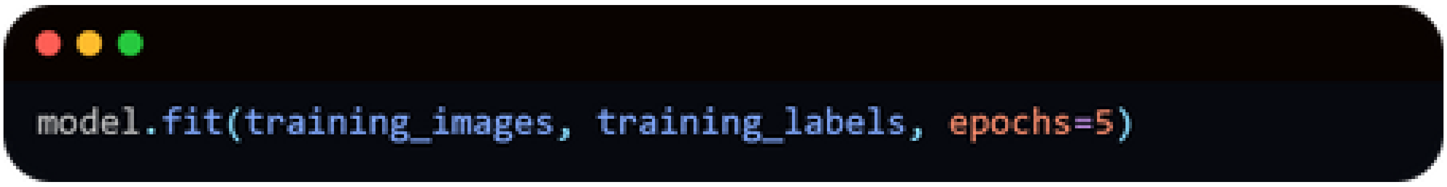
3. Compile and Train the Model

Compile the model using an optimizer and loss function:



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

Train the model with the training data



```python
model.fit(training_images, training_labels, epochs=5)
```

4. Test the Model

Evaluate the model's performance on unseen test data:

```
model.evaluate(test_images, test_labels)
```

5. Exploration Exercises

- **Exercise 1**: Predict and print classifications for test images.

```
classifications = model.predict(test_images)
print(classifications[0])
print(test_labels[0])
```

- **Exercise 2**: Experiment with different numbers of neurons in the Dense layer to see how it affects performance and training time.
- **Exercise 3**: Remove the Flatten() layer to observe the resulting error due to data shape mismatch.
- **Exercise 4**: Adjust the number of output neurons to match the number of classes and observe the impact.
- **Exercise 5**: Add additional layers and evaluate their effect on model performance.
- **Exercise 6**: Try training without normalizing the data and compare the results.

6. Explore Callbacks

Implement callbacks to stop training once a certain accuracy is reached:

```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('accuracy') > 0.95:
            print("\nReached 95% accuracy, stopping training!")
            self.model.stop_training = True

callbacks = myCallback()
model.fit(training_images, training_labels, epochs=5, callbacks=[callbacks])
```

You've successfully built and trained your first computer vision model! To further enhance your models, explore convolutional layers and pooling techniques in the next steps.