Project Title: Image Classification using TensorFlow

Project Description: In this project, you will build a simple image classification model using TensorFlow. The model will be trained to classify images of handwritten digits from the MNIST dataset.

Project Steps:

1. Install TensorFlow and import the necessary libraries.
2. Load the MNIST dataset using TensorFlow.
3. Pre-process the data by scaling the pixel values and splitting the dataset into training and testing sets.
4. Define the architecture of the model using TensorFlow's Sequential API.
5. Compile the model by specifying the loss function, optimizer, and metrics.
6. Train the model using the training set and evaluate its performance on the testing set.
7. Test the model on some sample images and check if it can correctly classify them.

In this code, we first load the MNIST dataset and pre-process it by scaling the pixel values. Then we define a simple neural network model using TensorFlow's Sequential API. We compile the model using an appropriate loss function and optimizer, and train it on the training set. We evaluate the model on the testing set and test it on some sample images. The final prediction is obtained using the argmax() function to get the index of the predicted class.

Code Explanation:

**import tensorflow as tf**

**from tensorflow.keras.datasets import mnist**

*We first import the TensorFlow library and the MNIST dataset from the Keras library within TensorFlow.*

**(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()**

*We then load the MNIST dataset into four separate arrays: x\_train, y\_train, x\_test, and y\_test. The x arrays contain the pixel values of the images, and the y arrays contain the corresponding labels (i.e., the digits the images represent).*

**x\_train = x\_train / 255.0**

**x\_test = x\_test / 255.0**

*Next, we preprocess the data by scaling the pixel values to be between 0 and 1. This is done to make the training process faster and more effective.*

**model = tf.keras.models.Sequential([**

**tf.keras.layers.Flatten(input\_shape=(28, 28)),**

**tf.keras.layers.Dense(128, activation='relu'),**

**tf.keras.layers.Dropout(0.2),**

**tf.keras.layers.Dense(10)**

**])**

*We then define the model architecture using the Sequential API within Keras. In this case, we have a simple model with three layers: a Flatten layer that takes in the 28x28 input image and flattens it into a 784-dimensional array, a Dense layer with 128 neurons and a ReLU activation function, and a Dropout layer that randomly drops out 20% of the neurons to prevent overfitting. Finally, we have another Dense layer with 10 neurons, one for each possible digit class (0-9).*

**loss\_fn = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)**

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

*We then compile the model by specifying the loss function, optimizer, and metrics. In this case, we use the SparseCategoricalCrossentropy loss function, which is appropriate for multi-class classification problems like this one. We use the Adam optimizer, which is a popular choice for deep learning models. We also specify that we want to track the accuracy metric during training.*

**model.fit(x\_train, y\_train, epochs=10, validation\_data=(x\_test, y\_test))**

*We then train the model on the training set using the fit() function. We train for 10 epochs (i.e., 10 passes through the entire training set) and use the validation set to monitor the model's performance during training.*

**model.evaluate(x\_test, y\_test)**

*We evaluate the model's performance on the testing set using the evaluate() function. This gives us the final accuracy score and loss value for the model.*

**import numpy as np**

**from PIL import Image**

**img = Image.open("sample\_image.png").convert("L")**

**img = np.asarray(img.resize((28, 28)))**

**img = img / 255.0**

**pred = model.predict(np.array([img]))**

**class\_idx = np.argmax(pred)**

**print(“Predicted Number is : , class\_idx)**

*Finally, we test the model on a sample image. We first load the image using the PIL library and preprocess it by converting it to grayscale, resizing it to 28x28, and scaling the pixel values. We then use the predict() function to make a prediction on the preprocessed image. The output is a probability distribution over the 10 possible digit classes. We use the argmax() function to get the index of the predicted class with the highest probability.*