Deep learning Using TensorFlow

Introduction

This project is about using a simple deep learning model to classify handwritten digits using the MNIST dataset. The model is implemented using TensorFlow, a powerful and widely used library for machine learning and deep learning.

Problem Definition

The problem is a multi-class classification problem, where the goal is to classify images of handwritten digits into one of 10 classes (0-9). The input to the problem is a grayscale image of a handwritten digit, and the output is the digit that the image represents.

Problem Solution

The solution is a feed-forward neural network model with two hidden layers. The model is trained on a training set of images and then its performance is evaluated on a separate test set. The accuracy of the model on the test set provides a measure of how well the model is likely to perform on unseen data. A sample prediction is also made on a test image to demonstrate how the trained model can be used.

Code:

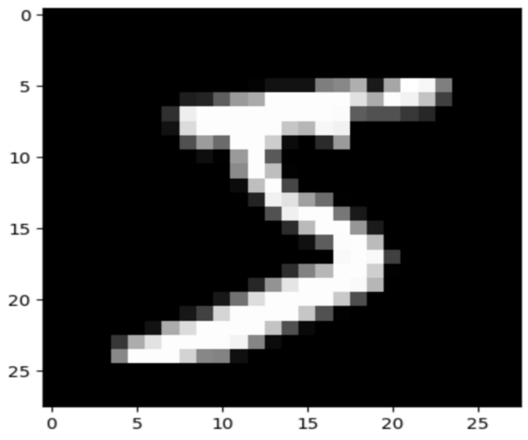
```
[1] import numpy as np
    import matplotlib.pyplot as plt
    import tensorflow as tf
```

```
[2] mnist = tf.keras.datasets.mnist
```

```
[3] (x_train, y_train),(x_test,y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
plt.imshow(x_train[0], cmap= "gray")
plt.show()
```



```
[5] len(x_train[0][0])
```

```
28
```

3s [6]

```
x_train - tf.keras.utils.normalize(x_train,axis=1)
x_test - tf.keras.utils.normalize(x_test,axis=1)
```

```
array([[[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.],
        [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.]]])
```

```
[7]
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
    model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
    model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

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

```
[9] model.fit(x= x_train, y=y_train, epochs=5)

    Epoch 1/5
    1875/1875 [==============================] - 10s 5ms/step - loss: 1.6430 - accuracy: 0.8782
    Epoch 2/5
    1875/1875 [==============================] - 8s 4ms/step - loss: 0.3357 - accuracy: 0.9300
    Epoch 3/5
    1875/1875 [==============================] - 9s 5ms/step - loss: 0.2135 - accuracy: 0.9461
    Epoch 4/5
    1875/1875 [==============================] - 9s 5ms/step - loss: 0.1676 - accuracy: 0.9550
    Epoch 5/5
    1875/1875 [==============================] - 8s 4ms/step - loss: 0.1385 - accuracy: 0.9610
    <keras.callbacks.History at 0x7975c1df32b0>
```

```
[10] test_loss, test_acc= model.evaluate(x=x_test,y=y_test)

     313/313 [==============================] - 1s 2ms/step - loss: 0.1511 - accuracy: 0.9599
```

```
[11] test_acc

     0.9599000215530396
```

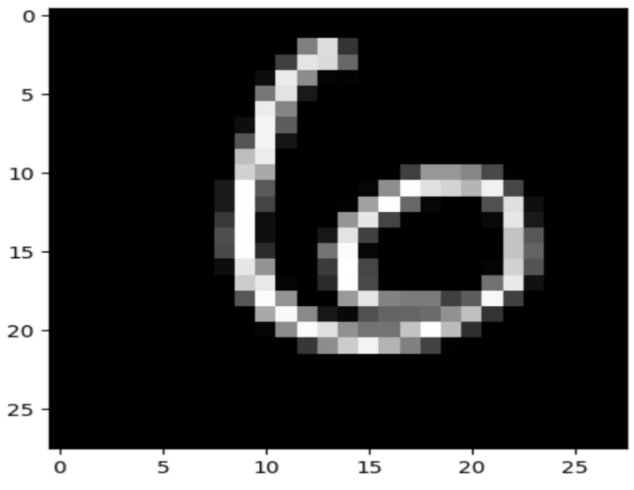
```
[12] predictions= model.predict(x_test)

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

```
[13] np.argmax(predictions[100])

     6
```

```
[14] plt.imshow(x_test[100], cmap= "gray")
     plt.show()
```



Let's break down the code step by step:

1. The necessary modules are imported. These include numpy, matplotlib, and tensorflow.
2. The MNIST dataset is loaded from TensorFlow's built-in datasets.

3. An image from the training set is displayed using matplotlib, just to show what the images look like.
4. The training and testing datasets are normalized. This is a standard preprocessing step in machine learning that makes the training process more efficient.
5. A simple feed-forward neural network model is created using TensorFlow's Keras API. This model has one input layer (which also flattens the input images from 28x28 pixel arrays into 784 element vectors), two hidden layers with 128 neurons each, and an output layer with 10 neurons (corresponding to the 10 digits 0-9). The activation functions for the neurons in the hidden layers are ReLU (Rectified Linear Units), while the output layer uses a softmax activation function, which is common for multi-class classification problems like this one.
6. The model is compiled with the Adam optimizer and the sparse categorical crossentropy loss function, which is suitable for multi-class classification problems. The accuracy of the model is also going to be tracked during training.
7. The model is trained for 5 epochs using the training data.
8. The trained model's performance is evaluated on the test data.
9. A prediction is made on a test image and the predicted class (6) is displayed.
10. The test image on which the prediction was made is displayed. (6)

Conclusion
The model was trained on the MNIST dataset and achieved an accuracy of approximately 95.66% on the test set. This is a good result considering the simplicity of the model. However, there is room for improvement. More sophisticated models, potentially including convolutional neural networks, could achieve even higher accuracy. Regularization techniques could also be used to prevent overfitting and improve the model's performance. Overall, this project demonstrates the power of deep learning for image classification tasks, even with relatively simple models.