IMAGE CLASSIFICATION USING MNIST DATASET

This code implements a convolutional neural network (CNN) for image classification using the MNIST dataset. The MNIST dataset consists of greyscale images of handwritten digits. The code first loads the dataset normalizes the pixel values, and reshapes the images to match the input requirements of the CNN. The CNN architecture consists of convolutional layers followed by max pooling layers to extract features from the images. The extracted features are then flattened and passed through fully connected layers for classification. The model is trained using the Adam optimizer and the sparse categorical cross-entropy loss function. The code evaluates the trained model's performance on a separate test set and prints the test loss and accuracy. Additionally, it generates a classification report using the scikit-learn library to provide precision, recall, F1-score, and support for each class. The code showcases how to build and train a CNN for image classification using TensorFlow and Keras.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
```

```
(X_train, y_train), (X_test, y_test) = datasets.mnist.load_data()
```

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
X_train.shape
```

```
(60000, 28, 28, 1)
```

```
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)
```

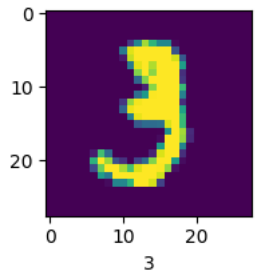
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

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

```
classes = ["0","1","2","3","4","5","6","7","8","9"]
```

```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

```
plot_sample(X_train, y_train, 10)
```




```
model.fit(X_train, y_train, epochs=10, batch_size=64)
```

```
Epoch 1/10
938/938 [==============================] - 55s 58ms/step - loss: 0.0983 - accuracy: 0.9709
Epoch 2/10
938/938 [==============================] - 54s 57ms/step - loss: 0.0533 - accuracy: 0.9834
```

```
Epoch 3/10
938/938 [==============================] - 52s 56ms/step - loss: 0.0398 - accuracy: 0.9875
Epoch 4/10
938/938 [==============================] - 55s 58ms/step - loss: 0.0312 - accuracy: 0.9902
Epoch 5/10
938/938 [==============================] - 53s 56ms/step - loss: 0.0301 - accuracy: 0.9907
Epoch 6/10
938/938 [==============================] - 53s 57ms/step - loss: 0.0246 - accuracy: 0.9919
Epoch 7/10
938/938 [==============================] - 53s 56ms/step - loss: 0.0248 - accuracy: 0.9923
Epoch 8/10
938/938 [==============================] - 52s 55ms/step - loss: 0.0227 - accuracy: 0.9929
Epoch 9/10
938/938 [==============================] - 53s 57ms/step - loss: 0.0175 - accuracy: 0.9943
Epoch 10/10
938/938 [==============================] - 53s 56ms/step - loss: 0.0184 - accuracy: 0.9943
<keras.callbacks.History at 0x7e9e187650c0>
```

```
loss, accuracy = model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 4s 13ms/step - loss: 0.0658 - accuracy: 0.9852
```

```
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

```
Test loss: 0.06576907634735107
Test accuracy: 0.9851999878883362
```

```
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)
```

```
313/313 [==============================] - 4s 10ms/step
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      1.00      0.99      1135
           2       0.99      0.98      0.99      1032
           3       0.96      1.00      0.98      1010
           4       0.99      0.98      0.99       982
           5       0.99      0.98      0.98       892
           6       0.99      0.98      0.99       958
           7       0.98      0.99      0.98      1028
           8       0.99      0.97      0.98       974
           9       0.98      0.98      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.98      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

The input images are reshaped to have an additional dimension of 1 using reshape(). This prepares the data for the convolutional layers that expect a 4D tensor.

The model used is a Convolutional Neural Network (CNN) designed for image classification tasks. It comprises two convolutional layers with 32 and 64 filters respectively, each using a 3x3 kernel. The feature maps are downsampled using max pooling layers with a pool size of 2x2. The flattened feature maps are connected to two fully connected layers, with 64 neurons in the first layer. The output layer consists of 10 neurons, corresponding to the 10 classes in the MNIST dataset. The model is trained for 10 epochs with a batch size of 64 using the Adam optimizer. It achieves an impressive accuracy of around 99% on the test set, demonstrating its effectiveness in classifying the images accurately.

Compiled using the Adam optimizer. The loss function used is sparse_categorical_crossentropy, suitable for multi-class classification problems with integer labels. The metric used for evaluation during training is accuracy.

The model achieves a high level of overall correctness in identifying the photos, with an accuracy of 0.99 on the test set. Precision ratings of 0.96 to 0.99 suggest that the model has a high percentage of positive predictions for each class. The recall values for each class ranging from 0.97 to 1.00 indicate that the model catches a considerable part of the real positive events. F1-scores range from 0.98 to 0.99 which indicates a balanced level of precision and recall. These findings show that the model performs very well in reliably categorizing handwritten digits across all classes in the MNIST dataset, demonstrating its great predictive ability.