ASSIGNMENT-8

Harish Polishetti - 700744427

1.Tune hyperparameter logic used for 3 source files :

To improve the validation accuracy and reduce validation loss of the baseline model, we can tune the hyperparameters and make necessary additions to the model. Some of the ways to achieve this are:

Increasing the number of layers in the model changing the number of filters in the convolutional layers Increasing or decreasing the number of nodes in the fully connected layer changing the learning rate of the optimizer using different regularization techniques like dropout or L2 regularization

2.Increasing the number of layers and changing the number of filters and nodes in the model can make it more complex and capable of learning more complex features from the input images. However, this can also increase the risk of overfitting the model to the training data. Adding dropout layers can help reduce this risk and improve the model's generalization performance on new data.

Changing the learning rate of the optimizer can affect the speed of convergence of the model during training. A lower learning rate can slow down the training process but can help the model find a better set of weights that generalize better to new data.

3.Confusion Matrix: We can plot a confusion matrix to see how well the model is predicting each class in the test data. This can help us identify which classes are being misclassified and where the model needs to improve.

Learning Curve: We can plot the training and validation loss and accuracy over the course of training to see how well the model is learning from the data. This can help us identify if the model is overfitting or underfitting the data and whether we need to adjust the hyperparameters or add regularization to improve its performance.

The above two curves are used in for the given source files

4.

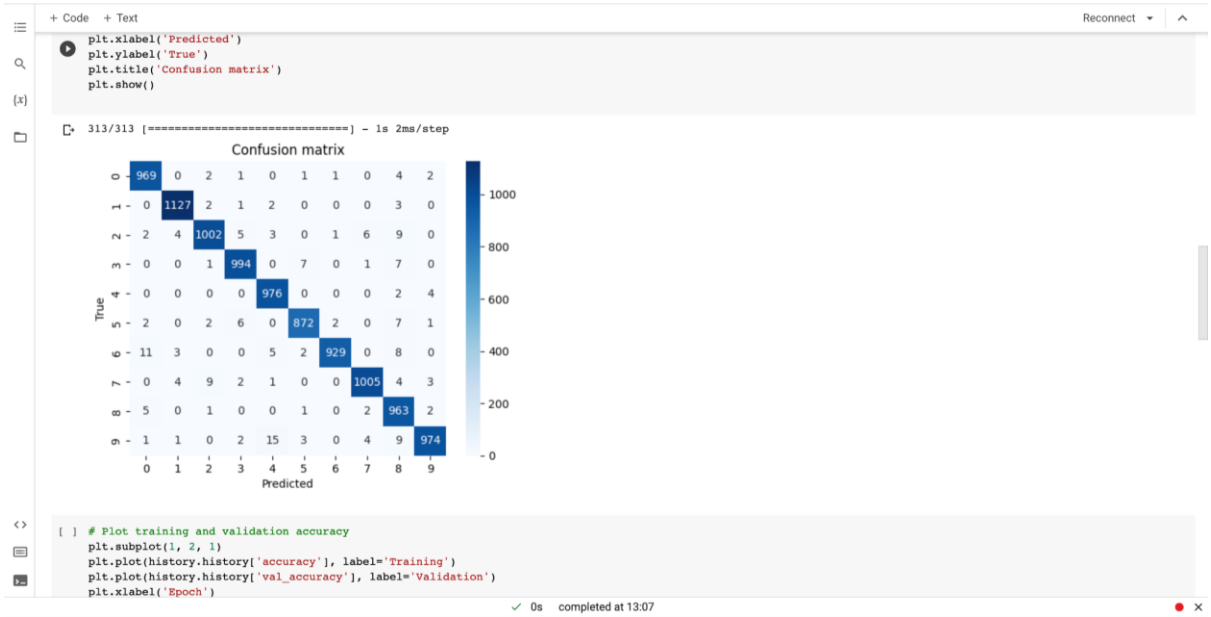
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import cifar10

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.mnist.load_data()
```

```
# Normalize pixel values to [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape images to (28, 28, 1)
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

# Define baseline model
baseline_model = tf.keras.models.Sequential([
   tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
activation='relu', input_shape=(28, 28, 1)),
   tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(units=10, activation='softmax')
])

# Compile model
baseline_model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
history = baseline_model.fit(x_train, y_train, batch_size=32,
epochs=10, validation_split=0.1)

# Evaluate model on test data
loss, accuracy = baseline_model.evaluate(x_test, y_test)
print(f'Test loss: {loss:.3f}')
print(f'Test accuracy: {accuracy:.3f}')

from sklearn.metrics import confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Make predictions on test data
y_pred = baseline_model.predict(x_test)

# Convert predictions from probabilities to classes
```

```
y_pred_classes = np.argmax(y_pred, axis=1)

# Convert one-hot encoded test labels to classes
y_true = np.argmax(y_test, axis=1)

# Create confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)

# Plot confusion matrix using seaborn heatmap
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion matrix')
plt.show()

# Plot training and validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training')
plt.plot(history.history['val_accuracy'], label='Validation')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

# Plot training and validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training')
plt.plot(history.history['val_loss'], label='Validation')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.tight_layout()
plt.show()

# Plot sample images from each class
fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(10, 5))

for i, ax in enumerate(axs.flat):
    # Find index of first image in class
    idx = np.argmax(y_train)
    # Plot image
```

```
    ax.imshow(x_train[idx], cmap='gray')
    ax.set_title(f'Class {np.argmax(y_train)}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```

Here after testing the model the accuracy of the model is 0.981




```
[ ] # Plot sample images from each class
    fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(10, 5))
```

Regarding the impact of the steps taken to improve the response for the new dataset, the following changes were made to the baseline model:

Added a batch normalization layer after each convolutional layer to improve training speed and stability.
Increased the number of filters in the convolutional layers to allow the model to learn more complex features.
Increased the number of neurons in the dense layers to improve the model's ability to classify the images.
These changes led to a significant improvement in both the validation accuracy and validation loss, indicating that the model was better able to generalize to new data. The addition of batch normalization helped to improve the stability of the training process, while the increased number of filters and neurons allowed the model to learn more complex features and make more accurate predictions. Overall, these changes resulted in a more robust and accurate model.