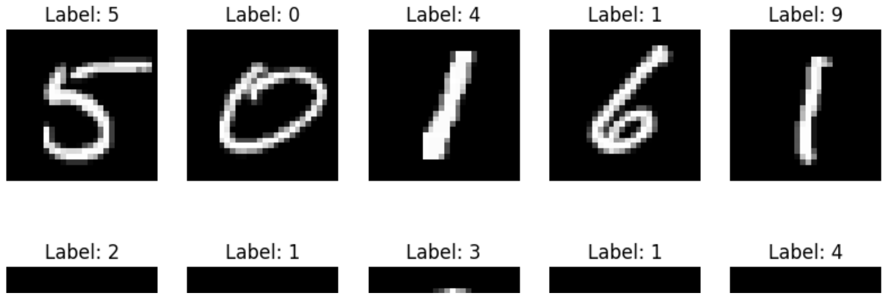
```
import tensorflow as tf
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from sklearn.impute import SimpleImputer

# Load the datasets
train_data = pd.read_csv('mnist_train.csv')
test_data = pd.read_csv('mnist_test.csv')

# Explore the dataset
print("Number of unique classes:", train_data['label'].nunique())
print("Number of features:", len(train_data.columns) - 1)  # Excluding the label column
print("Missing values in training set:", train_data.isnull().sum().sum())

#X_test->test_images, y_test->test_labels

# Handle missing values by replacing NaN with mean
X_train = train_data.iloc[:, 1:].values
X_test  = test_data.values[:, 1:]

# Use SimpleImputer to replace NaN with mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Convert to float32 explicitly
X_train = X_train.astype('float32')
X_test  = X_test.astype('float32')

# Normalize pixel values
X_train /= 255.0
X_test /= 255.0

# Reshape images
X_train = X_train.reshape(-1, 28, 28, 1)
X_test  = X_test.reshape(-1, 28, 28, 1)

# Encode labels
y_train = to_categorical(train_data['label'])
y_test  = to_categorical(test_data['label'])

# Split into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)


# Visualize some images
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_train[i].reshape(28, 28), cmap='gray')
    ax.set_title(f"Label: {train_data['label'][i]}")
    ax.axis('off')
plt.show()
```

```
Number of unique classes: 10
Number of features: 784
Missing values in training set: 0
```




```
from sklearn.metrics import confusion_matrix
#Code steps
    #Builds a neural network model.
    #Compiles the model.
    #Trains the model on training data.
    #Evaluates the model on the test set.
    #Prints the test accuracy.
    #Calculates and prints the confusion matrix.

# Experiment 1

# Build the ANN model
model_1 = Sequential()
model_1.add(Flatten(input_shape=(28, 28, 1)))  # Flatten the 28x28 images
model_1.add(Dense(128, activation='relu'))
model_1.add(Dense(10, activation='softmax'))   # Output layer with 10 classes
#The output layer is added with 10 neurons (representing the 10 classes in the MNIST dataset)


# Compile the model
model_1.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
Train_model_1 = model_1.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))

# Evaluate on test set
test_loss_1, test_accurecy_1 = model_1.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {test_accurecy_1}\n")

# Confusion Matrix
y_predict1 = model_1.predict(X_test)
Confusion_matrix_1 = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_predict1, axis=1))
print("Confusion Matrix (Experiment 1):\n", Confusion_matrix_1)
```

```
Epoch 1/10
750/750 [==============================] - 4s 5ms/step - loss: 0.3308 - accuracy: 0.9076 - val_loss: 0.1904 - val_accuracy: 0.9473
Epoch 2/10
750/750 [==============================] - 3s 4ms/step - loss: 0.1569 - accuracy: 0.9545 - val_loss: 0.1369 - val_accuracy: 0.9617
Epoch 3/10
750/750 [==============================] - 3s 4ms/step - loss: 0.1096 - accuracy: 0.9674 - val_loss: 0.1124 - val_accuracy: 0.9682
Epoch 4/10
750/750 [==============================] - 2s 3ms/step - loss: 0.0835 - accuracy: 0.9764 - val_loss: 0.0973 - val_accuracy: 0.9721
Epoch 5/10
750/750 [==============================] - 3s 4ms/step - loss: 0.0648 - accuracy: 0.9809 - val_loss: 0.0942 - val_accuracy: 0.9713
Epoch 6/10
750/750 [==============================] - 3s 4ms/step - loss: 0.0528 - accuracy: 0.9846 - val_loss: 0.0924 - val_accuracy: 0.9714
Epoch 7/10
750/750 [==============================] - 3s 4ms/step - loss: 0.0423 - accuracy: 0.9873 - val_loss: 0.0843 - val_accuracy: 0.9758
Epoch 8/10
750/750 [==============================] - 3s 4ms/step - loss: 0.0344 - accuracy: 0.9899 - val_loss: 0.0860 - val_accuracy: 0.9748
Epoch 9/10
750/750 [==============================] - 2s 3ms/step - loss: 0.0274 - accuracy: 0.9925 - val_loss: 0.0900 - val_accuracy: 0.9740
Epoch 10/10
750/750 [==============================] - 3s 4ms/step - loss: 0.0231 - accuracy: 0.9935 - val_loss: 0.0849 - val_accuracy: 0.9756
375/375 [==============================] - 1s 2ms/step - loss: 0.0849 - accuracy: 0.9756

Test Accuracy: 0.9755833148956299

375/375 [==============================] - 1s 2ms/step
Confusion Matrix (Experiment 1):
 [[1156    0    3    1    4    1    6    0    3    1]
 [   0 1311    3    2    2    0    1    1    1    1]
 [   1    8 1145    6    2    0    1    6    3    2]
 [   0    1   14 1179    0   11    2    4    6    2]
```

```
 [   1    3    0    1 1154    1    3    1    1   11]
 [   7    3    2   11    4 1058    9    2    6    2]
 [   1    0    3    0    1    1 1171    0    0    0]
 [   1    7    9    1    3    0    0 1275    1    2]
 [   3    7    3    6    5    6    8    1 1114    7]
 [   4    1    1    3   19    2    1   12    7 1144]]
```

```
#Hyperparameters:
    #Learning Rate: Determines the step size during optimization.
    #Batch Size: Number of training examples used in one iteration.
    #Number of Layers and Neurons: Architecture of the neural network.
    #Activation Functions: Choice of activation functions in each layer.
    #Dropout Rate: Regularization technique to prevent overfitting.
    #Weight Initialization: Initial values assigned to the weights.

# Experiment 2

# Build the ANN model
model_2 = Sequential()
model_2.add(Flatten(input_shape=(28, 28, 1)))
model_2.add(Dense(256, activation='relu'))  # Different number of neurons
model_2.add(Dense(10, activation='softmax'))

# Compile the model
model_2.compile(optimizer=Adam(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
Train_model_2 = model_2.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate on test set
test_loss_2, test_accurecy_2 = model_2.evaluate(X_test, y_test)
print(f"\nTest Accuracy: {test_accurecy_2}\n")

# Confusion Matrix
y_predict2 = model_2.predict(X_test)
Confusion_matrix_2 = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_predict2, axis=1))
print("Confusion Matrix (Experiment 2):\n", Confusion_matrix_2)
```

```
Epoch 1/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2602 - accuracy: 0.9237 - val_loss: 0.1891 - val_accuracy: 0.9444
Epoch 2/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1685 - accuracy: 0.9540 - val_loss: 0.1899 - val_accuracy: 0.9513
Epoch 3/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1500 - accuracy: 0.9598 - val_loss: 0.1716 - val_accuracy: 0.9598
Epoch 4/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1367 - accuracy: 0.9641 - val_loss: 0.2043 - val_accuracy: 0.9533
Epoch 5/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1172 - accuracy: 0.9697 - val_loss: 0.2140 - val_accuracy: 0.9544
Epoch 6/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1168 - accuracy: 0.9707 - val_loss: 0.2829 - val_accuracy: 0.9479
Epoch 7/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1114 - accuracy: 0.9724 - val_loss: 0.2571 - val_accuracy: 0.9526
Epoch 8/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1044 - accuracy: 0.9760 - val_loss: 0.2380 - val_accuracy: 0.9628
Epoch 9/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1070 - accuracy: 0.9762 - val_loss: 0.2119 - val_accuracy: 0.9664
Epoch 10/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0877 - accuracy: 0.9795 - val_loss: 0.2582 - val_accuracy: 0.9609
375/375 [==============================] - 1s 3ms/step - loss: 0.2582 - accuracy: 0.9609

Test Accuracy: 0.9609166383743286

375/375 [==============================] - 1s 2ms/step
Confusion Matrix (Experiment 2):
 [[1151    0    1    0    2    2    5    0   13    1]
 [   0 1294   11    2    1    0    1    3    9    1]
 [   2    3 1136    8    0    1    2   13    8    1]
 [   0    0   10 1173    0    7    0    8   20    1]
 [   3    1    2    0 1106    1   23   10   12   18]
 [   7    2    2   32    1 1012   18    2   26    2]
 [   3    0    2    0    1    1 1163    1    6    0]
 [   1    4   10    3    1    0    0 1273    5    2]
 [   5    1    7    5    2    2    5    1 1132    0]
 [   5    1    2   24   21    2    1   26   21 1091]]
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Flatten images
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Convert one-hot encoded labels back to integers
y_train_int = np.argmax(y_train, axis=1)
y_test_int = np.argmax(y_test, axis=1)

# Define the K-NN model
knn_model = KNeighborsClassifier()

# Define hyperparameter grid for grid search
param_grid = {'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance']}

# Perform grid search
grid_search = GridSearchCV(knn_model, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train_flat, y_train_int)

# Print the best parameters
print("Best Parameters:", grid_search.best_params_)

# Train the model with the best parameters
best_knn_model = grid_search.best_estimator_
best_knn_model.fit(X_train_flat, y_train_int)

# Predict on the validation set
knn_val_predictions = best_knn_model.predict(X_test_flat)

# Evaluate the model on the validation set
test_accuracy_3 = accuracy_score(y_test_int, knn_val_predictions)
print(f"Validation Accuracy (K-NN): {test_accuracy_3}")

# Print classification report and confusion matrix
print("\nClassification Report:")
print(classification_report(y_test_int, knn_val_predictions))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test_int, knn_val_predictions))
```

```
Best Parameters: {'n_neighbors': 3, 'weights': 'distance'}
Validation Accuracy (K-NN): 0.9735833333333334

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1175
           1       0.96      1.00      0.98      1322
           2       0.99      0.96      0.97      1174
           3       0.97      0.97      0.97      1219
           4       0.97      0.97      0.97      1176
           5       0.97      0.97      0.97      1104
           6       0.98      0.99      0.99      1177
           7       0.97      0.98      0.97      1299
           8       0.98      0.95      0.97      1160
           9       0.96      0.96      0.96      1194

    accuracy                           0.97     12000
   macro avg       0.97      0.97      0.97     12000
weighted avg       0.97      0.97      0.97     12000


Confusion Matrix:
[[1167    0    0    0    0    1    3    1    1    2]
 [   0 1316    1    1    1    0    0    2    0    1]
 [   7   11 1131    2    3    1    0   15    3    1]
 [   1    0    8 1181    0   13    0    4    6    6]
 [   0    6    1    1 1135    0    2    4    0   27]
 [   5    3    0   14    2 1067    8    0    4    1]
 [   3    2    0    0    1    3 1168    0    0    0]
 [   0   19    1    1    2    0    0 1268    3    5]
 [   3    8    4   15    6   12    3    3 1100    6]
 [   4    2    2    1   15    2    2   15    1 1150]]
```

```
!pip install joblib
```

```
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (1.3.2)
```

```
from tensorflow.keras.models import load_model

import joblib
from sklearn.metrics import confusion_matrix

# Compare the outcomes
if test_accurecy_1 > test_accurecy_2 and test_accurecy_1 > test_accuracy_3:
    BestModel = model_1
    print("Experiment 1 has the higher accuracy on the validation set. ANN model_1\n")
elif test_accurecy_2 > test_accurecy_1 and test_accurecy_2 > test_accuracy_3:
    BestModel = model_2
    print("Experiment 2 has the higher accuracy on the validation set. ANN model_2\n")
else:
    BestModel = knn_model
    print("Experiment 3 has the higher accuracy on the validation set. KNN model\n")

# Save the best model
if isinstance(BestModel, KNeighborsClassifier):
    joblib.dump(BestModel, "Best_model_knn.joblib")
else:
    BestModel.save("Best_model_Ann.h5")

# Reload the model
if isinstance(BestModel, KNeighborsClassifier):
    loaded_model = joblib.load("Best_model_knn.joblib")
else:
    loaded_model = load_model("Best_model_Ann.h5")

# Display the model summary
if not isinstance(BestModel, KNeighborsClassifier):
    loaded_model.summary()

# Evaluate the reloaded model on the test set
if isinstance(BestModel, KNeighborsClassifier):
    knn_val_predictions = loaded_model.predict(X_test)
    accuracy = accuracy_score(y_test, knn_val_predictions)
    print(f"\nTest Accuracy using the reloaded KNN model: {accuracy}")
    # Get confusion matrix
    cm = confusion_matrix(y_test, knn_val_predictions)
    print("\nConfusion Matrix:")
    print(cm)
else:
    test_loss, test_accuracy = loaded_model.evaluate(X_test, y_test)
    print(f"\nTest Accuracy using the reloaded ANN model: {test_accuracy}")
```

```
Experiment 1 has the higher accuracy on the validation set. ANN model_1

Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_6 (Flatten)         (None, 784)               0

 dense_12 (Dense)            (None, 128)               100480

 dense_13 (Dense)            (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
375/375 [==============================] - 2s 5ms/step - loss: 0.0849 - accuracy: 0.9756

Test Accuracy using the reloaded ANN model: 0.9755833148956299
```