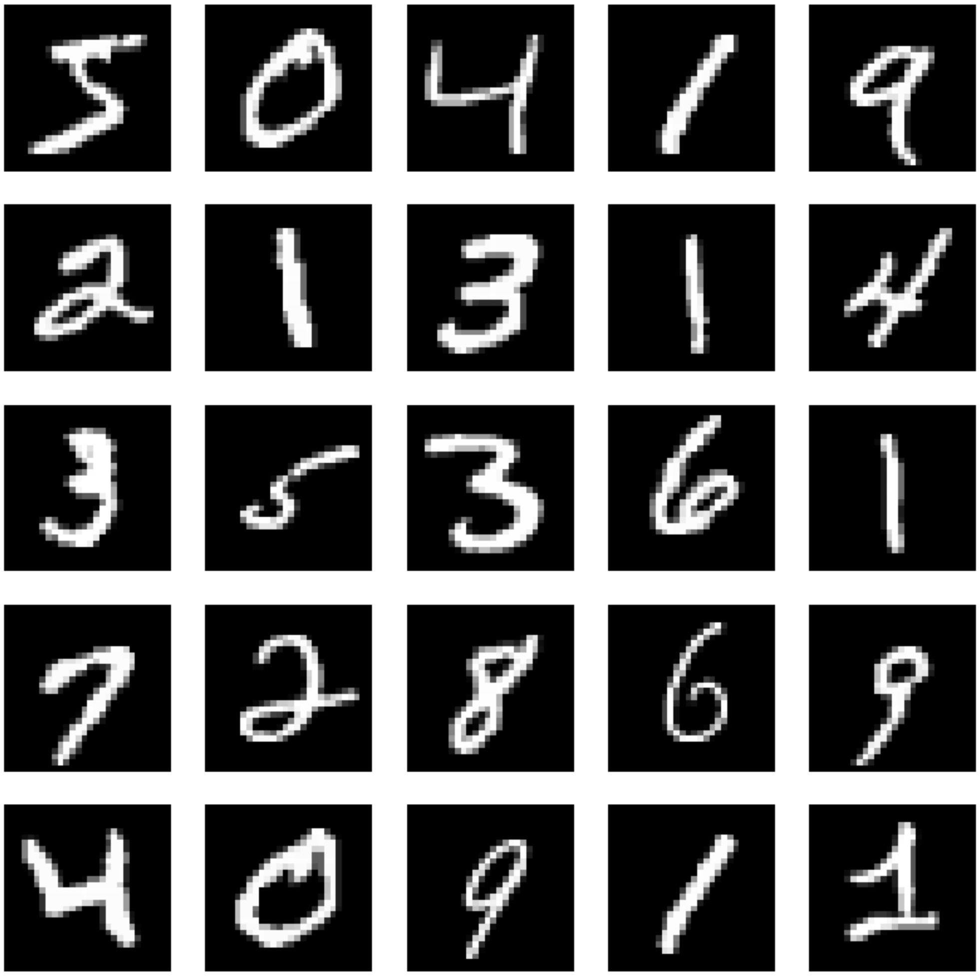
```
import joblib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.losses import SparseCategoricalCrossentropy  # Modified import

# (a) Data Exploration and Preparation

# Load the dataset
train_data = pd.read_csv('/content/drive/MyDrive/mnist_train.csv')

# Data exploration
num_classes = train_data['label'].nunique()
num_features = len(train_data.columns) - 1  # Subtract 1 for the label column
print("Data Exploration:")
print("The number of the unique classes : ",num_classes)
print("The number of unique features : ", num_features)
# Check for missing values
missing_values = train_data.isnull().sum()
print("The number of missing values: ")
print(missing_values)
# split the target and features Normalize pixel values

features=train_data.drop(['label'],axis=1)
target=train_data[['label']]
normalized_features= features / 255.0

# Resize images to 28x28
image_size = (28, 28)
X = normalized_features.values.reshape(-1, *image_size, 1)

# Visualize resized 25 images
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(X[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()

print("KNN USING GRID SEARCH TO GET Optimal Hyper Parameters")
# Split the training data into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(normalized_features, target, test_size=0.2, random_state=42)

# (b) Experiments and Results

# Initial Experiment: K-NN Algorithm with Grid Search

# Prepare K-NN model
knn_model = KNeighborsClassifier()

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

# Perform grid search
search_knn = GridSearchCV(knn_model, param_grid=param_knn, cv=3)
search_knn.fit(X_train, y_train)

# Get the best K-NN model
best_knn_model = search_knn.best_estimator_

# Print the best hyperparameters for K-NN
print("Best Hyperparameters for K-NN:", search_knn.best_params_)

# Predictions on validation set using K-NN
y_pred_knn = best_knn_model.predict(X_test)

# Evaluate K-NN model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("K-NN Validation Accuracy:", accuracy_knn)

print("--------------------------------------------------------------------------------")

# Subsequent Experiment: ANN with variations in architecture
print("The ANN MODEL EXPERIMENT 1")
# Reshape data for ANN
X_train_reshaped = X_train.values.reshape(-1, *image_size, 1)
X_test_reshaped = X_test.values.reshape(-1, *image_size, 1)

# Implement and train first ANN architecture
model_ann_1 = models.Sequential([
    layers.Flatten(input_shape=image_size),
    layers.Dense(64, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

model_ann_1.compile(optimizer='adam', loss=SparseCategoricalCrossentropy(), metrics=['accuracy'])
model_ann_1.fit(X_train_reshaped, y_train, epochs=10, validation_data=(X_test_reshaped, y_test))

# Predictions on validation set for the first ANN
y_pred_ann_1 = np.argmax(model_ann_1.predict(X_test_reshaped), axis=-1)


# Evaluate the first ANN model
accuracy_ann_1 = accuracy_score(y_test, y_pred_ann_1)
print("ANN Architecture 1 Validation Accuracy:", accuracy_ann_1)
```

```
print("--------------------------------------------------------------------------------")
print("THE ANN EXPERIMENT 2:")
# Implement and train second ANN architecture
model_ann_2 = models.Sequential([
    layers.Flatten(input_shape=image_size),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

model_ann_2.compile(optimizer='adam', loss=SparseCategoricalCrossentropy(), metrics=['accuracy'])
model_ann_2.fit(X_train_reshaped, y_train, epochs=10, validation_data=(X_test_reshaped, y_test))

# Predictions on validation set for the second ANN
y_pred_ann_2 = np.argmax(model_ann_2.predict(X_test_reshaped), axis=-1)

# Evaluate the second ANN model
accuracy_ann_2 = accuracy_score(y_test, y_pred_ann_2)
print("ANN Architecture 2 Validation Accuracy:", accuracy_ann_2)

# Compare outcomes and select the best model
best_model = None
best_accu=0
y_pred_best=None
if accuracy_knn >= accuracy_ann_1 and accuracy_knn >= accuracy_ann_2:
    best_model = best_knn_model
    best_model_type = "K-NN"
    best_accu=accuracy_knn
    y_pred_best=y_pred_knn
elif accuracy_ann_1 >= accuracy_ann_2:
    best_model = model_ann_1
    best_model_type = "ANN Architecture 1"
    best_accu=accuracy_ann_1
    y_pred_best=y_pred_ann_1
else:
    best_model = model_ann_2
    best_model_type = "ANN Architecture 2"
    best_accu=accuracy_ann_2
    y_pred_best=y_pred_ann_2

print(f"The best model is {best_model_type} with Validation Accuracy: {best_accu}")

print("--------------------------------------------------------------------------------")
conf_matrix = confusion_matrix(y_test, y_pred_best)
print("Confusion Matrix of the Best Model:")
print(conf_matrix)

# Save the best model to a file
if best_model_type == "K-NN":
  #this saving way because the knn don't have save function to save the model in joblib file
    joblib.dump(best_model, 'best_knn_model.pkl')
else:
  #this saving way because the ANN has function to save the joblib file
    best_model.save('best_ann_model.h5')

print(f"The model {best_model_type} is saved in a file")
print("--------------------------------------------------------------------------------")


# Reload the best model from the file
if best_model_type == "K-NN":
    loaded_model = joblib.load('best_knn_model.pkl')
    print("The file best_knn_model.pkl is loaded from the file")
else:
    loaded_model = models.load_model('best_ann_model.h5')
    print("The file best_ann_model.h5 is loaded from the file")


# Load test data
test_data = pd.read_csv('/content/drive/MyDrive/mnist_test.csv')

# split the test data and Normalize pixel values for test data features

test_features=test_data.drop(['label'],axis=1)
test_target=test_data[['label']]

#normalize the test data
normalized_features_test= test_features / 255.0

# Resize images to 28x28 for test data features to match the ANN model if wanted
reshaped_features_test= normalized_features_test.values.reshape(-1, *image_size, 1)

accuracy_test=0

# Use the best model on the testing data
if best_model_type == "K-NN":
    y_test_pred = loaded_model.predict(normalized_features_test)
    accuracy_test = accuracy_score(test_target, y_test_pred)
else:
    y_test_pred = np.argmax(loaded_model.predict(reshaped_features_test), axis=-1)
    accuracy_test = accuracy_score(test_target, y_test_pred)

print("Best Model Testing Accuracy:", accuracy_test)
```

```
Data Exploration:
The number of the unique classes :  10
The number of unique features :  784
The number of missing values:
label    0
1x1      0
1x2      0
1x3      0
1x4      0
        ..
28x24    0
28x25    0
28x26    0
28x27    0
28x28    0
Length: 785, dtype: int64
```



```
KNN USING GRID SEARCH TO GET Optimal Hyper Parameters
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
```

```
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWa
  return self._fit(X, y)
Best Hyperparameters for K-NN: {'n_neighbors': 3, 'weights': 'distance'}
K-NN Validation Accuracy: 0.9735833333333334
--------------------------------------------------------------------------------
The ANN MODEL EXPERIMENT 1
Epoch 1/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.3327 - accuracy: 0.9067 - val_
Epoch 2/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1640 - accuracy: 0.9525 - val_
Epoch 3/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1186 - accuracy: 0.9650 - val_
Epoch 4/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0943 - accuracy: 0.9724 - val_
Epoch 5/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0772 - accuracy: 0.9771 - val_
Epoch 6/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0639 - accuracy: 0.9808 - val_
Epoch 7/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0550 - accuracy: 0.9825 - val_
Epoch 8/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0460 - accuracy: 0.9861 - val_
Epoch 9/10
1500/1500 [==============================] - 7s 4ms/step - loss: 0.0392 - accuracy: 0.9883 - val_
Epoch 10/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0347 - accuracy: 0.9896 - val_
375/375 [==============================] - 1s 3ms/step
ANN Architecture 1 Validation Accuracy: 0.9705833333333334
--------------------------------------------------------------------------------
THE ANN EXPERIMENT 2:
Epoch 1/10
1500/1500 [==============================] - 7s 4ms/step - loss: 0.2668 - accuracy: 0.9232 - val_
Epoch 2/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.1152 - accuracy: 0.9653 - val_
Epoch 3/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0793 - accuracy: 0.9755 - val_
Epoch 4/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0610 - accuracy: 0.9813 - val_
Epoch 5/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0446 - accuracy: 0.9854 - val_
Epoch 6/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0367 - accuracy: 0.9880 - val_
Epoch 7/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0315 - accuracy: 0.9900 - val_
Epoch 8/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0246 - accuracy: 0.9916 - val_
Epoch 9/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0229 - accuracy: 0.9923 - val_
Epoch 10/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0186 - accuracy: 0.9939 - val_
375/375 [==============================] - 1s 2ms/step
ANN Architecture 2 Validation Accuracy: 0.9750833333333333
The best model is ANN Architecture 2 with Validation Accuracy: 0.9750833333333333
--------------------------------------------------------------------------------
Confusion Matrix of the Best Model:
[[1157    0    3    1    3    2    3    1    5    0]
 [   0 1309    4    1    3    0    1    0    2    2]
 [   2    3 1150    6    4    0    2    4    2    1]
 [   1    0   14 1171    0   13    0    2    7   11]
 [   1    1    1    1 1145    0    2    1    1   23]
 [   4    1    1   13    2 1071    3    1    4    4]
 [   2    0    0    2    1    6 1164    0    2    0]
 [   0    2   12    2    3    1    0 1247    4   28]
 [   5    3    3    6    2    7    4    1 1118   11]
 [   2    0    2    2    3    4    0    6    6 1169]]
The model ANN Architecture 2 is saved in a file
--------------------------------------------------------------------------------
The file best_ann_model.h5 is loaded from the file
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are sa
  saving_api.save_model(
313/313 [==============================] - 1s 2ms/step
Best Model Testing Accuracy: 0.9769
```