```
# Mahmoud Adel Mamdouh 20200500
# Mohamed Tarek Fathi  20200794
# Dina Othman Emam     20200173
# Habiba Ayman Eltahry 20200140
```

```
import pandas as pd
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
import tensorflow as tf
from keras.models import load_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from keras.utils import to_categorical
```

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

# Explore the first few rows
print(test_data.head())
```

```
   label  1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  ...  28x19  28x20  \
0      7    0    0    0    0    0    0    0    0    0  ...      0      0
1      2    0    0    0    0    0    0    0    0    0  ...      0      0
2      1    0    0    0    0    0    0    0    0    0  ...      0      0
3      0    0    0    0    0    0    0    0    0    0  ...      0      0
4      4    0    0    0    0    0    0    0    0    0  ...      0      0

   28x21  28x22  28x23  28x24  28x25  28x26  28x27  28x28
0      0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      0

[5 rows x 785 columns]
```

```
unique_classes = train_data['label'].nunique()
print("Number of unique classes:", unique_classes)

# Check the number of features
num_features = len(train_data.columns) - 1  # subtract 1 for the label column
print("Number of features:", num_features)
```

```
Number of unique classes: 10
Number of features: 784
```

```
print("Missing values in training data:\n", train_data.isnull().sum())
print(" ")
print("Missing values in test data:\n", test_data.isnull().sum())
print(" ")
train_data = train_data.dropna()
# test_data = test_data.dropna()

print("Missing values in training data after drop na :\n", train_data.isnull().sum())
# print("Missing values in test data after drop na :\n", test_data.isnull().sum())
print(" ")
```

```
Missing values in training data:
 label    0
1x1      0
1x2      0
1x3      0
1x4      0
        ..
28x24    0
```

```
28x25    0
28x26    0
28x27    0
28x28    0
Length: 785, dtype: int64

Missing values in test data:
 label    0
1x1      0
1x2      0
1x3      0
1x4      0
        ..
28x24    0
28x25    0
28x26    0
28x27    0
28x28    0
Length: 785, dtype: int64

Missing values in training data after drop na :
 label    0
1x1      0
1x2      0
1x3      0
1x4      0
        ..
28x24    0
28x25    0
28x26    0
28x27    0
28x28    0
Length: 785, dtype: int64
```

```
train_data.iloc[:, 1:] /= 255.0
test_data.iloc[:, 1:] /= 255.0
train_data
```

```
<ipython-input-51-3485f1e34fee>:1: DeprecationWarning: In a future version, `df.iloc[:,
  train_data.iloc[:, 1:] /= 255.0
<ipython-input-51-3485f1e34fee>:2: DeprecationWarning: In a future version, `df.iloc[:,
  test_data.iloc[:, 1:] /= 255.0
```

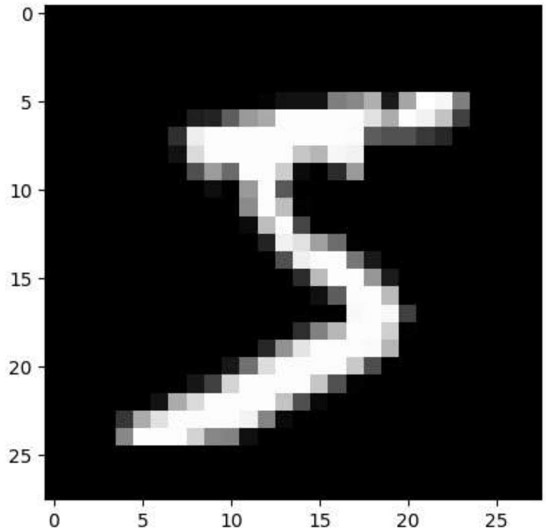
	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x2
0	5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.
1	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.
2	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.
3	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.
4	9	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.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
59995	8	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.
59996	3	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.
59997	5	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.
59998	6	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.
59999	8	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.

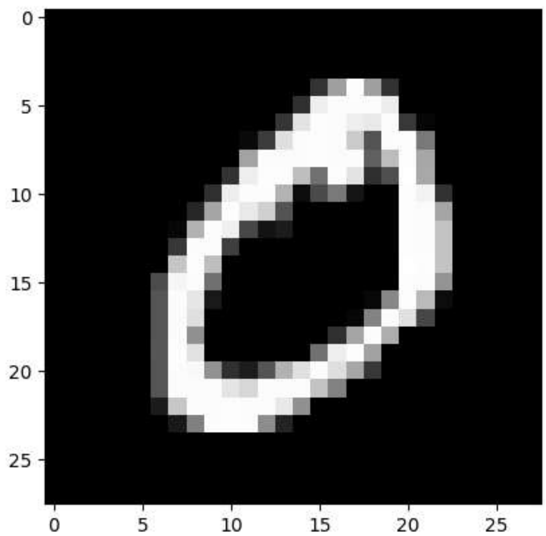
60000 rows × 785 columns

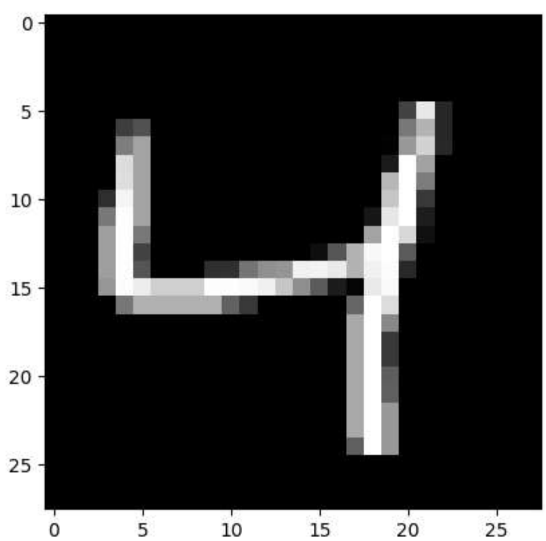
```
# Get rows with unique values in the specified column
unique_records_df = train_data.drop_duplicates(subset=['label'])
```

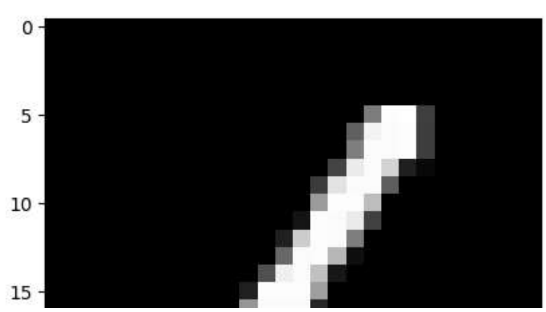
```
def resize_images(data):
    return data.iloc[:, 1:].values.reshape(-1, 28, 28)

resized_unique_train_images = resize_images(unique_records_df)
```

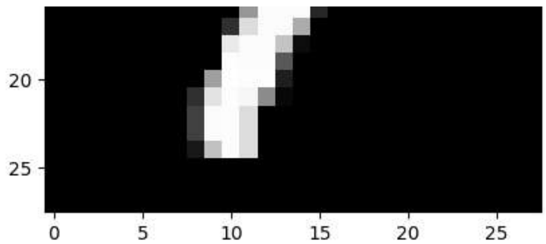
```
for image in resized_unique_train_images:
  plt.imshow(image, cmap='gray')
  plt.show()
```

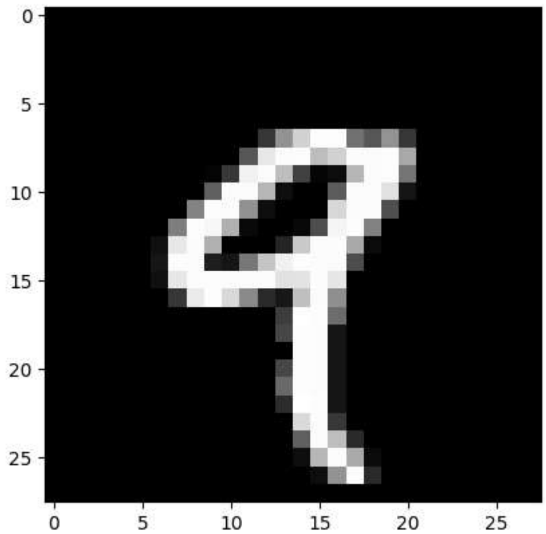

0
5
10
15
20
25
0
5
10
15
20
25

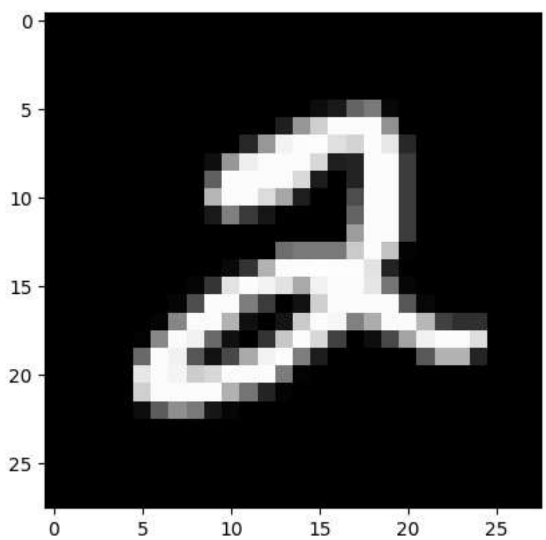

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15








```
X_train, X_val, y_train, y_val = train_test_split(
    train_data.iloc[:, 1:], train_data['label'], test_size=0.2, random_state=42
)
```

0 5 10 15 20 25

```
knn = KNeighborsClassifier()

# Define the parameter grid
param_grid = {'n_neighbors': np.arange(2, 9)}
param_grid
# Initialize the grid search
grid_search = GridSearchCV(knn, param_grid, cv=5)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best parameters: ", best_params)

# Make predictions
predictions = grid_search.predict(X_val)

# Print the accuracy and confusion matrix
print("Accuracy: ", accuracy_score(y_val, predictions))
print("Confusion Matrix: \n", confusion_matrix(y_val, predictions))
```

```
Best parameters:  {'n_neighbors': 3}
Accuracy:  0.9726666666666667
Confusion Matrix:
 [[1167    0    1    0    1    1    2    1    1    1]
 [   0 1318    1    0    0    0    0    2    0    1]
 [   7   11 1133    1    3    1    0   15    2    1]
 [   1    0    8 1181    0   14    0    3    6    6]
 [   1    7    1    1 1135    0    1    4    0   26]
 [   7    4    0   13    2 1067    7    0    3    1]
 [   3    3    0    0    1    3 1167    0    0    0]
 [   0   19    2    0    2    0    0 1269    2    5]
 [   3    9    8   17    9   16    2    2 1089    5]
 [   4    4    2    2   18    2    2   13    1 1146]]
```

```
# Define the first model
model1 = Sequential([
   Dense(32, activation='relu', input_shape=(num_features,)),
   Dense(10, activation='softmax'),
])

# Compile the first model
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define the second model
model2 = Sequential([
   Dense(64, activation='relu', input_shape=(num_features,)),
   Dense(10, activation='softmax'),
])

# Compile the second model
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Convert labels to categorical
y_train_cat = to_categorical(y_train)
y_val_cat = to_categorical(y_val)

# Fit the first model
history1 = model1.fit(X_train, y_train_cat, epochs=10, batch_size=32, validation_data=(X_val, y_val_cat))

# Fit the second model
history2 = model2.fit(X_train, y_train_cat, epochs=10, batch_size=64, validation_data=(X_val, y_val_cat))

# Evaluate the first model
```

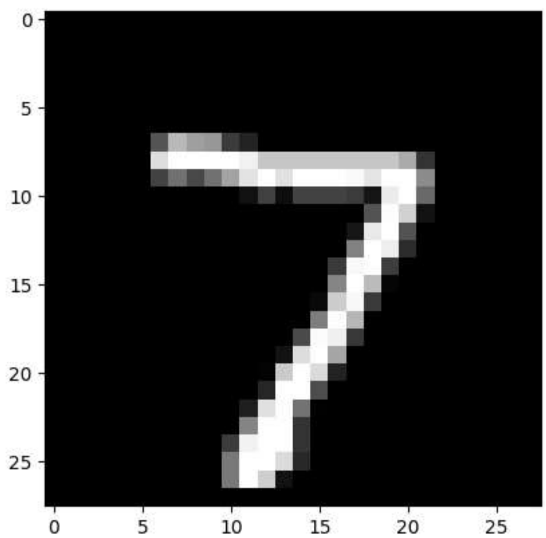
```
loss1, accuracy1 = model1.evaluate(X_val, y_val_cat)

# Evaluate the second model
loss2, accuracy2 = model2.evaluate(X_val, y_val_cat)

print("Model 1 Accuracy: ", accuracy1)
print("Model 2 Accuracy: ", accuracy2)
```

```
Epoch 1/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.3964 - accuracy: 0.8888 - val_loss: 0.2490 - val_accuracy: 0.9305
Epoch 2/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.2148 - accuracy: 0.9391 - val_loss: 0.2045 - val_accuracy: 0.9421
Epoch 3/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.1721 - accuracy: 0.9500 - val_loss: 0.1681 - val_accuracy: 0.9515
Epoch 4/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1440 - accuracy: 0.9580 - val_loss: 0.1551 - val_accuracy: 0.9561
Epoch 5/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1254 - accuracy: 0.9631 - val_loss: 0.1388 - val_accuracy: 0.9594
Epoch 6/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.1106 - accuracy: 0.9673 - val_loss: 0.1344 - val_accuracy: 0.9603
Epoch 7/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0987 - accuracy: 0.9712 - val_loss: 0.1301 - val_accuracy: 0.9616
Epoch 8/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0901 - accuracy: 0.9737 - val_loss: 0.1307 - val_accuracy: 0.9607
Epoch 9/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0821 - accuracy: 0.9755 - val_loss: 0.1308 - val_accuracy: 0.9603
Epoch 10/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0746 - accuracy: 0.9777 - val_loss: 0.1164 - val_accuracy: 0.9653
Epoch 1/10
750/750 [==============================] - 4s 4ms/step - loss: 0.3850 - accuracy: 0.8922 - val_loss: 0.2187 - val_accuracy: 0.9398
Epoch 2/10
750/750 [==============================] - 3s 4ms/step - loss: 0.1939 - accuracy: 0.9447 - val_loss: 0.1689 - val_accuracy: 0.9516
Epoch 3/10
750/750 [==============================] - 4s 5ms/step - loss: 0.1448 - accuracy: 0.9585 - val_loss: 0.1449 - val_accuracy: 0.9567
Epoch 4/10
750/750 [==============================] - 3s 4ms/step - loss: 0.1161 - accuracy: 0.9668 - val_loss: 0.1343 - val_accuracy: 0.9591
Epoch 5/10
750/750 [==============================] - 3s 3ms/step - loss: 0.0969 - accuracy: 0.9716 - val_loss: 0.1159 - val_accuracy: 0.9663
Epoch 6/10
750/750 [==============================] - 2s 3ms/step - loss: 0.0816 - accuracy: 0.9761 - val_loss: 0.1034 - val_accuracy: 0.9679
Epoch 7/10
750/750 [==============================] - 3s 3ms/step - loss: 0.0713 - accuracy: 0.9791 - val_loss: 0.0996 - val_accuracy: 0.9693
Epoch 8/10
750/750 [==============================] - 3s 4ms/step - loss: 0.0616 - accuracy: 0.9816 - val_loss: 0.0987 - val_accuracy: 0.9700
Epoch 9/10
750/750 [==============================] - 3s 4ms/step - loss: 0.0548 - accuracy: 0.9839 - val_loss: 0.0997 - val_accuracy: 0.9712
Epoch 10/10
750/750 [==============================] - 3s 3ms/step - loss: 0.0486 - accuracy: 0.9851 - val_loss: 0.0905 - val_accuracy: 0.9747
375/375 [==============================] - 1s 2ms/step - loss: 0.1164 - accuracy: 0.9653
375/375 [==============================] - 1s 2ms/step - loss: 0.0905 - accuracy: 0.9747
Model 1 Accuracy:  0.9653333425521851
Model 2 Accuracy:  0.9746666550636292
```

```
# Save the best model
if accuracy1 > accuracy2:
   model1.save('best_model.h5')
else:
   model2.save('best_model.h5')

# Reload the best model
best_model = load_model('best_model.h5')


# Get the first record (row) in the dataset
first_record = test_data.iloc[0]

# Extract pixel values (assuming they start from the second column)
image = first_record[1:].values.reshape(28, 28)

# Plot the image
print("first image data in test.")
plt.imshow(image, cmap='gray')
plt.show()
print("")
# Use the best model to make predictions on the test data
y_test_cat = to_categorical(test_data['label'])
predictions = best_model.predict(test_data.iloc[:, 1:])
```

```
predictions = best_model.predict(test_data.iloc[:, 1:])



# Assuming y_test_cat is in one-hot encoded format
y_test_labels = np.argmax(y_test_cat, axis=1)
predictions_labels = np.argmax(predictions, axis=1)

print("")
print("first predictions_labels label: ",predictions_labels[0])
print("")


# Print the confusion matrix
print("Confusion Matrix: \n", confusion_matrix(y_test_labels, predictions_labels))
print("")

# Calculate accuracy using accuracy_score
accuracy = accuracy_score(y_test_labels, predictions_labels)

# Print the confusion matrix and accuracy

print("Accuracy: {:.2%}".format(accuracy))
```

```
first image data in test.
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
```




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

first predictions_labels label:  7

Confusion Matrix: 
 [[ 971    0    0    1    0    2    4    1    1    0]
 [   0 1125    4    0    0    1    4    0    1    0]
 [   4    2 1005    2    2    0    4    5    8    0]
 [   0    0    4  982    1    4    0    6    8    5]
 [   1    0    2    1  961    0    5    2    0   10]
 [   2    2    0    7    2  863    7    2    5    2]
 [   4    2    2    0    5    5  940    0    0    0]
 [   2    6   17    2    1    0    0  987    2   11]
 [   3    1    5    3    5    3    7    3  939    5]
 [   3    3    2    4    8    3    0    2    5  979]]

Accuracy: 97.52%
```