Machine Learning

project

Name	ID
Basmala Magdy Mohamed	20201045
Esraa Osama Mohamed	20201014
Eman Fathy Abo Alhassan	20200105
Fatma Mahmoud Ramadan	20201134
Sara Ahmed Sayed	20200214

Data Exploration and preparation

```
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# libraries for ANN
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.metrics import confusion_matrix, classification_report
from random import choice, uniform
from tensorflow.keras.models import load_model
import joblib
```

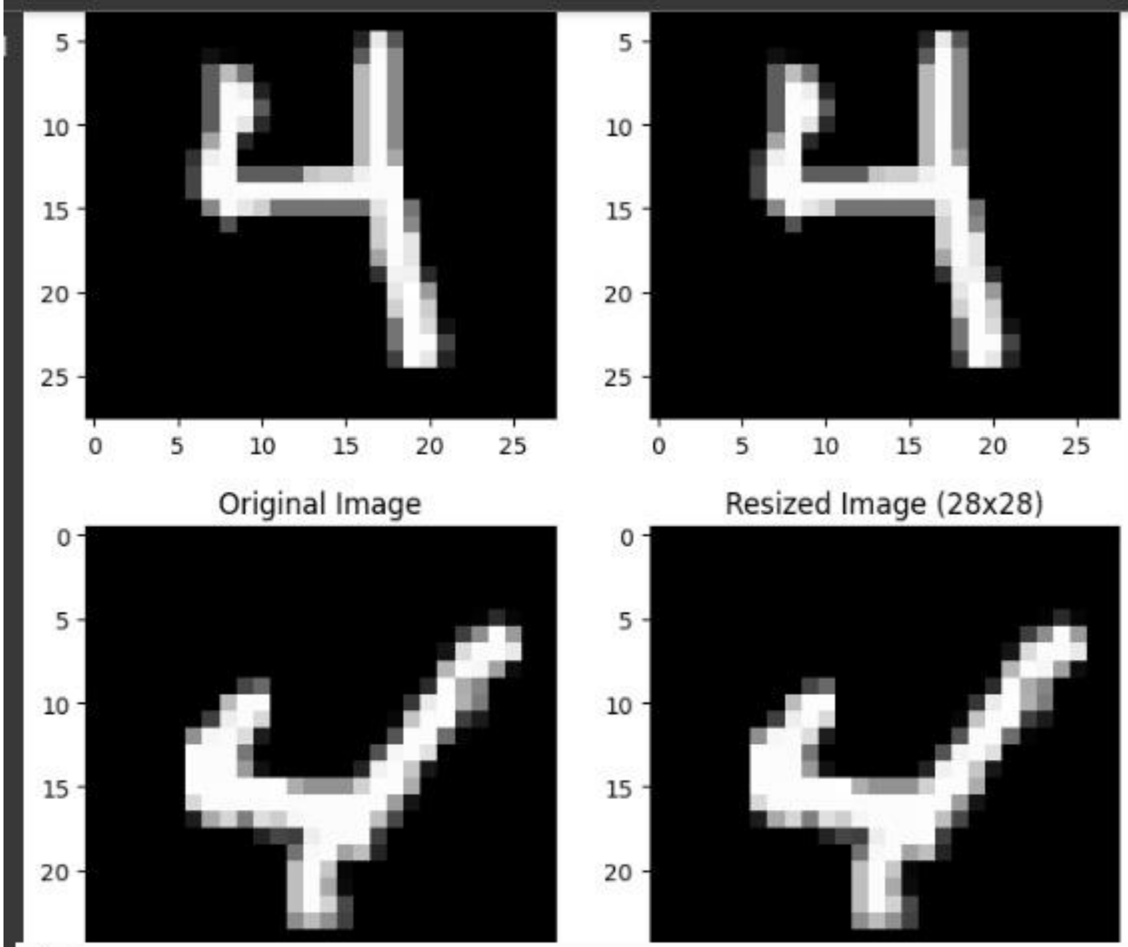
```
##1) Load the dataset and perform initial data exploration.

data =pd.read_csv("mnist_train.csv")
# Display the first 5 rows of the dataset
print(" Data Head: \n",data.head())

# Get an overview of the dataset including column names, data types, and non-null counts
print("Data Information: \n",data.info())

##2) Identify the number of unique classes
unique_classes = data.iloc[:, 0].nunique()
print("Number of Unique Classes:", unique_classes)

##3) Identify the number of features
print("Number of Features:", data.shape[1] - 1)

##4) handle missing values
print("Missing Values:\n", data.isnull().sum())



# Drop rows with any missing values
data.dropna(axis=0, inplace=True)

# Drop columns with any missing values
data.dropna(axis=1, inplace=True)

print("After Handling Missing Values:\n", data.isnull().sum())

##5) Normalize each image by dividing each pixel by 255
data.iloc[:, 1:] = data.iloc[:, 1:] / 255.0
```

```
 Data Head:
    label  1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  ...  28x19  28x20  \
0      5    0    0    0    0    0    0    0    0    0  ...      0      0
1      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
3      1    0    0    0    0    0    0    0    0    0  ...      0      0
4      9    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]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 785 entries, label to 28x28
dtypes: int64(785)
memory usage: 359.3 MB
Data Information:
 None
```

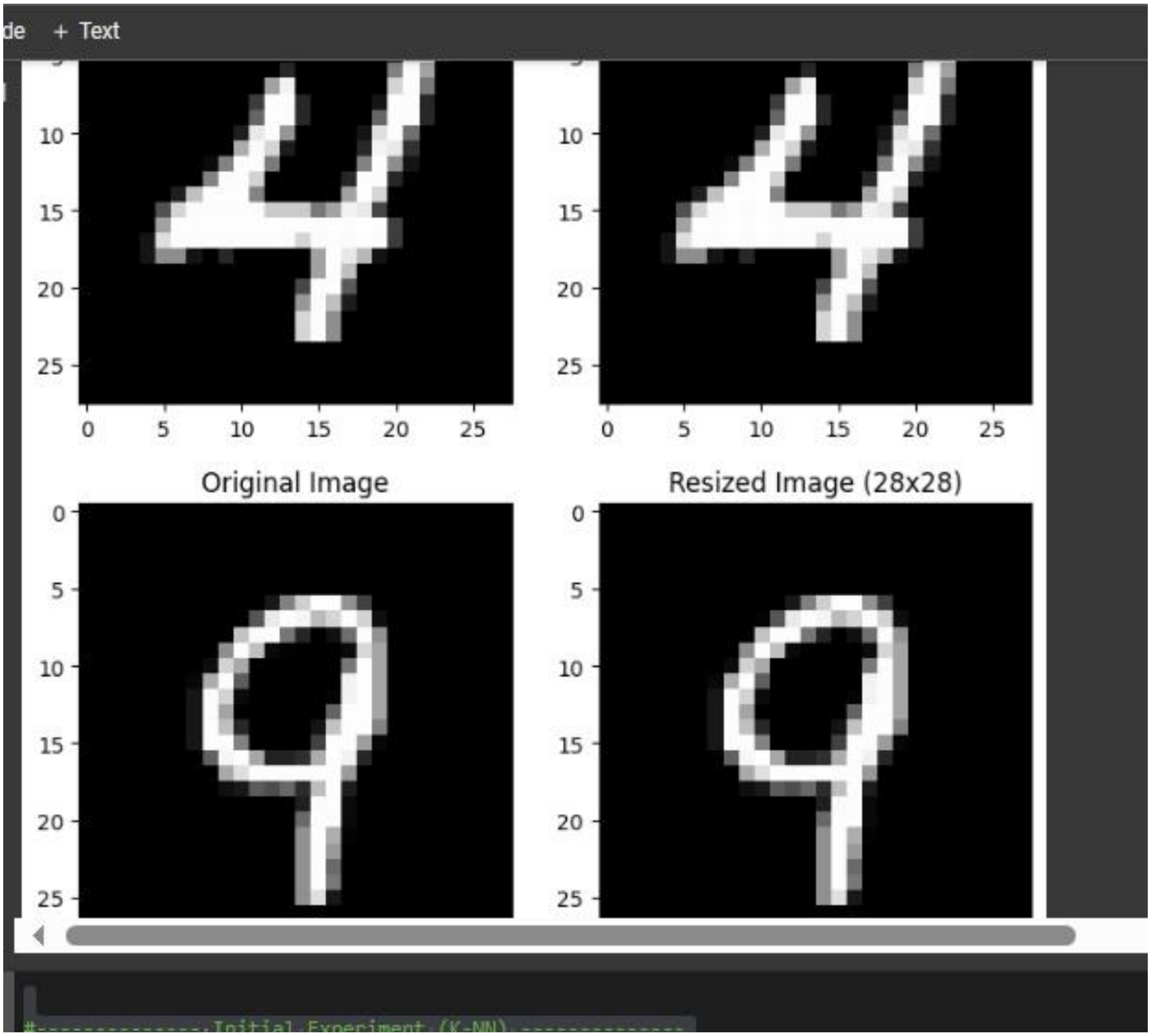
```
[5 rows x 785 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 785 entries, label to 28x28
dtypes: int64(785)
memory usage: 359.3 MB
Data Information:
 None
Number of Unique Classes: 10
Number of Features: 784
Missing Values:
 label    0
1x1      0
1x2      0
1x3      0
1x4      0
        ..
28x24    0
28x25    0
28x26    0
28x27    0
28x28    0
Length: 785, dtype: int64
```

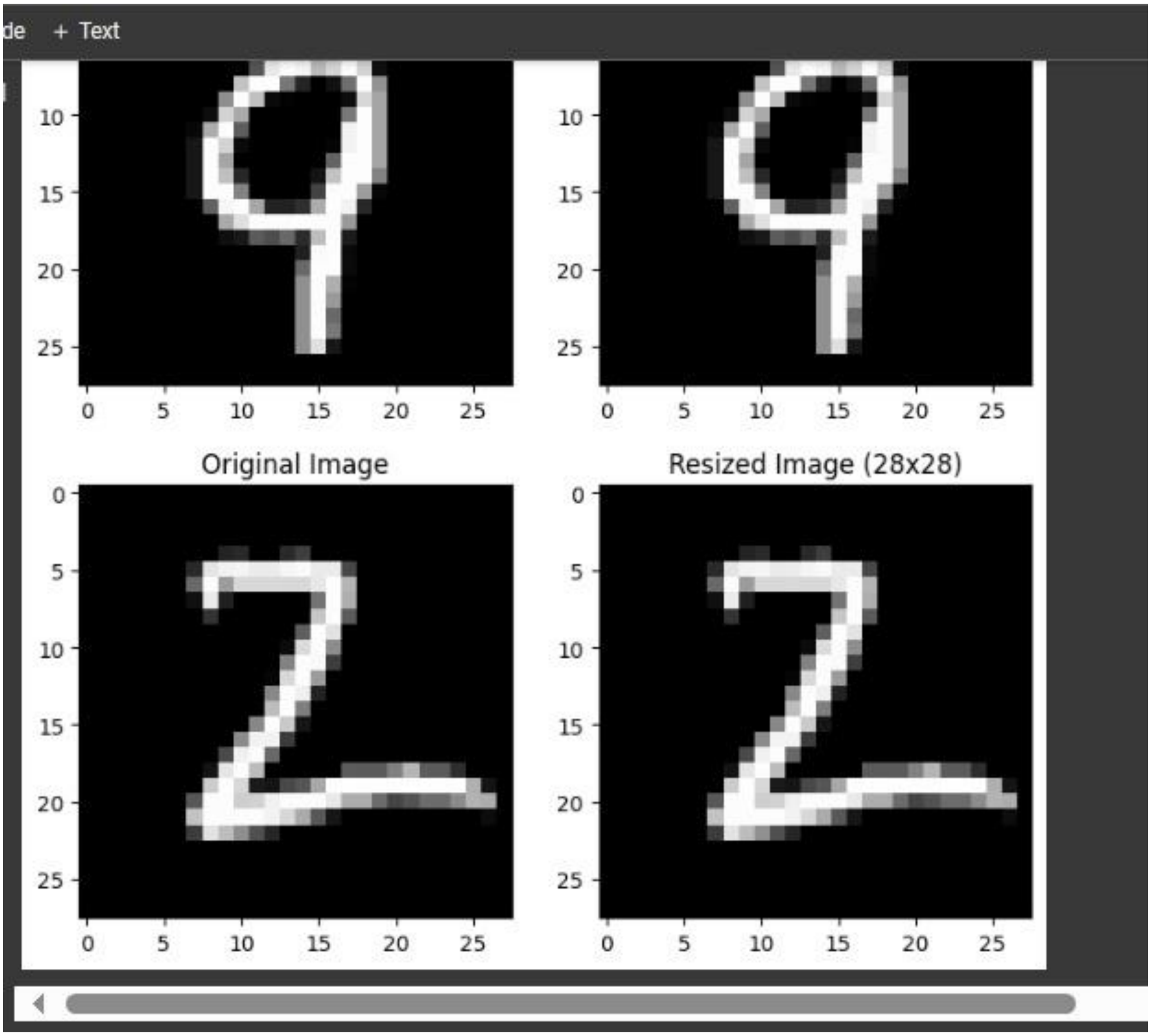
```
Length: 785, dtype: int64
After Handling Missing Values:
 label    0
1x1      0
1x2      0
1x3      0
1x4      0
        ..
28x24    0
28x25    0
28x26    0
28x27    0
28x28    0
Length: 785, dtype: int64
<ipython-input-1-58691746a43e>:58: DeprecationWarning: In a future version, `df.il
  data.iloc[:, 1:] = data.iloc[:, 1:] / 255.0
Training set shape: (48000, 784) (48000,)
Validation set shape: (12000, 784) (12000,)
```

de   + Text




```
#--------------Initial Experiment (K-NN)--------------
```


+ Text
Original Image
Resized Image (28x28)



+ Text
Original Image
Resized Image (28x28)

KNN with with grid search

```
#------------- Initial Experiment (K-NN) -------------
knn_model = KNeighborsClassifier()
param_grid = {
    'n_neighbors': [3, 5],
    'weights': ['distance'],
}

# # Perform grid search using cross-validation
grid_search = GridSearchCV(knn_model, param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

print("Best Hyperparameters:", grid_search.best_params_)

best_knn_model = grid_search.best_estimator_
best_knn_model.fit(X_train, y_train)

knn_predictions = best_knn_model.predict(X_val)

accuracy = accuracy_score(y_val, knn_predictions)
print("K-NN Accuracy:", accuracy)

# best_knn_model.save('best_knn_model.h5')
```

```
Best Hyperparameters: {'n_neighbors': 3, 'weights': 'distance'}
K-NN Accuracy: 0.9735833333333334
Epoch 1/5
```

ANN ,Compression

```
# Compile the model
ANN_model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Train the model
ANN_model2.fit(X_train.values.reshape(-1, 28, 28), y_train.values, epochs=5, validation_data=(X_val.values.reshape(-1, 28, 28), y_val.val

# Evaluate the model on the validation set
ANN_model2_accuracy = ANN_model2.evaluate(X_val.values.reshape(-1, 28, 28), y_val.values)[1]
print("ANN Model 2 Accuracy:", ANN_model2_accuracy)


best_ANN_model = None
best_ANN_accuracy = 0

if ANN_model1_accuracy > ANN_model2_accuracy:
    best_ann_accuracy = ANN_model1_accuracy
    best_ann_model = ANN_model1
else:
    best_ann_accuracy = ANN_model2_accuracy
    best_ann_model = ANN_model2

print("Best ANN Model Accuracy:", best_ann_accuracy)

best_ann_model.save('best_ann_model.h5')

if accuracy >= best_ann_accuracy:
    best_model = best_knn_model
    model_type = "K-NN"
else:
    best_model = best_ann_model
    model_type = "ANN"
print(f"\nThe best model is {model_type} with an accuracy of {max(accuracy, best_ann_accuracy)}")
```

```
[ ] #--------------- Subsequent Experiment (ANN) -------------
ANN_model1 = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(25, activation='relu'),
    layers.Dense(15, activation='relu'),
    layers.Dense(10, activation='linear')
])

# Compile the model
ANN_model1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Train the model
ANN_model1.fit(X_train.values.reshape(-1, 28, 28), y_train.values, epochs=5, validation_data=(X_val.values.reshape(-1, 28, 28), y_val.val

# Evaluate the model on the validation set
ANN_model1_accuracy = ANN_model1.evaluate(X_val.values.reshape(-1, 28, 28), y_val.values)[1]
print("ANN Model 1 Accuracy:", ANN_model1_accuracy)

ANN_model2 = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile the model
ANN_model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Train the model
ANN_model2.fit(X_train.values.reshape(-1, 28, 28), y_train.values, epochs=5, validation_data=(X_val.values.reshape(-1, 28, 28), y_val.val

# Evaluate the model on the validation set
ANN_model2_accuracy = ANN_model2.evaluate(X_val.values.reshape(-1, 28, 28), y_val.values)[1]
print("ANN Model 2 Accuracy:", ANN_model2_accuracy)


best_ANN_model = None
best_ANN_accuracy = 0
```

```
ANN Model 2 Accuracy: 0.9764999747276306
Best ANN Model Accuracy: 0.9764999747276306

The best model is ANN with an accuracy of 0.9764999747276306
```

Confusion Matrix ,Save, Load and Use Best Model

```python
[14] test_data = pd.read_csv("mnist_test.csv")

# Normalize the pixel values
test_data.iloc[:, 1:] = test_data.iloc[:, 1:] / 255.0
# Split the testing data into features (X_test) and labels (y_test)
X_test = test_data.iloc[:, 1:]
y_test = test_data.iloc[:, 0]

if model_type == "K-NN":
    #get confusion matrix for best knn model
    conf_matrix = confusion_matrix(y_val, best_knn_model.predict(X_val))
    #print confusion matrix
    print("Confusion Matrix (K-NN):")
    print(conf_matrix)

    # Save the best K-NN model using joblib
    joblib.dump(best_knn_model, 'best_knn_model.joblib')

    # Reload the best K-NN model from the saved file
    loaded_knn_model = joblib.load('best_knn_model.joblib')

    # Evaluate the best K-NN model on the testing set
    y_pred_knn = loaded_knn_model.predict(X_test)
    print("Predictions using the loaded best K-NN model:")
    print(y_pred_knn)
```

```python
else:
    # Load the best ANN model
    loaded_ann_model = load_model('best_ann_model.h5')

    # Make predictions on the entire dataset
    predictions_ann = loaded_ann_model.predict(X_val.values.reshape(-1, 28, 28))

    # Convert probability scores to class predictions
    predictions_ann_classes = tf.argmax(predictions_ann, axis=1).numpy()

    #get confusion matrix
    conf_matrix_ann = confusion_matrix(y_val, predictions_ann_classes)

    # Print the confusion matrix
    print("Confusion Matrix (ANN):")
    print(conf_matrix_ann)

    # Make predictions on the testing set
    predictions_annn = loaded_ann_model.predict(X_test.values.reshape(-1, 28, 28))

    if predictions_annn.shape[1] == 1:
    # If you have a single output neuron with softmax activation
     predictions_annn_classes = tf.argmax(predictions_ann, axis=1).numpy()
    else:
    # If you have multiple output neurons with softmax activation
     predictions_annn_classes = tf.argmax(predictions_ann, axis=-1).numpy()

    print("Predictions using the loaded ANN model:")
    print(predictions_annn_classes)
```

```
Confusion Matrix (ANN):
[[1159    0    2    0    0    2    3    1    6    2]
 [   0 1302    5    3    1    0    2    2    6    1]
 [   2    4 1151    3    0    0    2    8    2    2]
 [   1    0    9 1186    0    8    0    2    8    5]
 [   1    2    2    2 1144    0    9    0    4   12]
 [   5    2    3   21    4 1049    8    2    6    4]
 [   2    2    3    0    0    2 1164    0    4    0]
 [   1    5   10    0    4    1    0 1271    3    4]
 [   2    1    6    8    1    5    3    2 1129    3]
 [   4    1    1    0    7    3    1    9    5 1163]]
313/313 [==============================] - 0s 1ms/step
Predictions using the loaded ANN model:
[7 3 8 ... 9 7 2]
```