**Machine Learning Course**

# Project: COMPUTER VISION: MNIST CLASSIFICATION

**Submitted to: Dr-Abeer Korany**

| Name | ID |
|---|---|
| Rana Muhammad Ali | 20200182 |
| Hady Abdallah Hafez | 20200617 |
| Norhan Hassan Ali | 20201202 |
| Mohammed Essam Eldin Mohammed | 20200464 |

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from skimage.transform import resize
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
```

```
#Load the "mnist_train.csv" dataset and perform initial data exploration..
data = pd.read_csv("mnist_train.csv")

print(data)
print('----------------------')
print(data.dtypes)
print('----------------------')
print("data set shape:", data.shape)
print('----------------------')
```

```
       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
...      ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...    ...    ...
59995      8    0    0    0    0    0    0    0    0    0  ...      0      0
59996      3    0    0    0    0    0    0    0    0    0  ...      0      0
59997      5    0    0    0    0    0    0    0    0    0  ...      0      0
59998      6    0    0    0    0    0    0    0    0    0  ...      0      0
59999      8    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
...      ...    ...    ...    ...    ...    ...    ...    ...
59995      0      0      0      0      0      0      0      0
59996      0      0      0      0      0      0      0      0
59997      0      0      0      0      0      0      0      0
59998      0      0      0      0      0      0      0      0
59999      0      0      0      0      0      0      0      0

[60000 rows x 785 columns]
----------------------
label    int64
1x1      int64
1x2      int64
1x3      int64
1x4      int64
         ...
28x24    int64
28x25    int64
28x26    int64
28x27    int64
28x28    int64
Length: 785, dtype: object
----------------------
data set shape: (60000, 785)
----------------------
```

```
#Separate the features and target
x = data.drop(columns=['label'])
y = data['label']

#Identify the number of unique classes.
num_classes = y.nunique()
print("Number of unique classes:",num_classes)
print('----------------------')
```

```
Number of unique classes: 10
----------------------
```

```
#Identify the number of features.
num_features = len(x.columns)
print("Number of features (pixels):", num_features)
print('----------------------')
```

```
Number of features (pixels): 784
----------------------
```

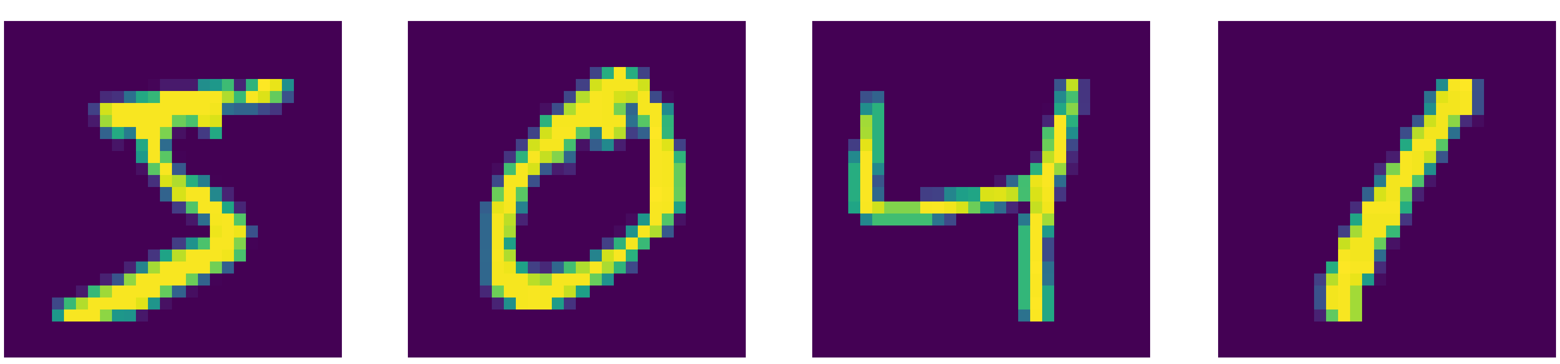
```
# Check whether there are missing values
missing_values = data.isnull().sum()
print('Missing values:')
print(missing_values)
print('----------------------')
```

```
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
----------------------
```

```
#Normalize each image by dividing each pixel by 255.
x=x/255
```

```
'''
Resize images to dimensions of 28 by 28.
After resizing, visualize some images to verify the correctness of the reshaping process.
'''
resized_image=[]
for i in range(len(x)):
    img = np.array(x.loc[i]).reshape(28, 28)
    resized_img = resize(img, (28, 28))
    resized_image.append(resized_img.flatten())

fig, axes = plt.subplots(1,4 , figsize=(10, 3))
for i in range(4):
    img = resized_image[i].reshape(28, 28)
    axes[i].imshow(img)
    axes[i].axis('off')
plt.show()
```



```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=100)
```

```
# Create a k-NN classifier
knn = KNeighborsClassifier()


# define the parameter values that should be searched
k_range = list(range(1, 20, 2))  # Odd values from 1 to 20
weight_options = ['uniform', 'distance']
# create a parameter grid: map the parameter names to the values that should be searched
param_grid = dict(n_neighbors=k_range, weights=weight_options)
print(param_grid)
```

```
{'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19], 'weights': ['uniform', 'distance']}
```

```
# instantiate the grid
grid_search = GridSearchCV(knn, param_grid, cv=3, scoring='accuracy', n_jobs=-1)

grid_search.fit(x_train, y_train)

# view the results
pd.DataFrame(grid_search.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
```

| | mean_test_score | std_test_score | params |
|---|---|---|---|
| 0 | 0.966708 | 0.003120 | {'n_neighbors': 1, 'weights': 'uniform'} |
| 1 | 0.966708 | 0.003120 | {'n_neighbors': 1, 'weights': 'distance'} |
| 2 | 0.966604 | 0.002022 | {'n_neighbors': 3, 'weights': 'uniform'} |
| 3 | 0.968354 | 0.002534 | {'n_neighbors': 3, 'weights': 'distance'} |
| 4 | 0.965729 | 0.003036 | {'n_neighbors': 5, 'weights': 'uniform'} |
| 5 | 0.967354 | 0.002822 | {'n_neighbors': 5, 'weights': 'distance'} |
| 6 | 0.964396 | 0.002434 | {'n_neighbors': 7, 'weights': 'uniform'} |
| 7 | 0.965813 | 0.002424 | {'n_neighbors': 7, 'weights': 'distance'} |
| 8 | 0.962458 | 0.002921 | {'n_neighbors': 9, 'weights': 'uniform'} |
| 9 | 0.963646 | 0.002968 | {'n_neighbors': 9, 'weights': 'distance'} |
| 10 | 0.960667 | 0.003011 | {'n_neighbors': 11, 'weights': 'uniform'} |
| 11 | 0.961938 | 0.002954 | {'n_neighbors': 11, 'weights': 'distance'} |
| 12 | 0.959417 | 0.002527 | {'n_neighbors': 13, 'weights': 'uniform'} |
| 13 | 0.960688 | 0.002491 | {'n_neighbors': 13, 'weights': 'distance'} |
| 14 | 0.958167 | 0.002519 | {'n_neighbors': 15, 'weights': 'uniform'} |
| 15 | 0.959167 | 0.002494 | {'n_neighbors': 15, 'weights': 'distance'} |
| 16 | 0.956792 | 0.002092 | {'n_neighbors': 17, 'weights': 'uniform'} |
| 17 | 0.957917 | 0.002094 | {'n_neighbors': 17, 'weights': 'distance'} |
| 18 | 0.955583 | 0.002297 | {'n_neighbors': 19, 'weights': 'uniform'} |
| 19 | 0.956833 | 0.001978 | {'n_neighbors': 19, 'weights': 'distance'} |

```
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'n_neighbors': 3, 'weights': 'distance'}
```

```
best_knn = KNeighborsClassifier(**best_params)
best_knn.fit(x_train, y_train)



y_pred = best_knn.predict(x_test)


best_knn_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", (best_knn_accuracy*100), "%")
```

```
Accuracy: 97.24166666666667 %
```

```
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.99      0.99      1185
           1       0.96      1.00      0.98      1365
           2       0.98      0.97      0.98      1152
           3       0.97      0.96      0.97      1247
           4       0.98      0.97      0.97      1196
           5       0.96      0.97      0.96      1089
           6       0.98      0.99      0.99      1198
           7       0.97      0.97      0.97      1230
           8       0.99      0.95      0.96      1206
           9       0.95      0.96      0.96      1132

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

```
#Traning a Neural network
from sklearn.neural_network import MLPClassifier

# Define the first ANN architecture
ann1 = MLPClassifier(
    hidden_layer_sizes=(100,),
    max_iter=500,
    random_state=100
)

# Train the first ANN
ann1.fit(x_train, y_train)

# Evaluate the first ANN on the validation set
y_pred_ann1 = ann1.predict(x_test)

accuracy_ann1 = accuracy_score(y_test, y_pred_ann1)
print("Accuracy (ANN1):", (accuracy_ann1*100), "%")
```

```
Accuracy (ANN1): 97.175 %
```

```
# Define the second ANN architecture with different hyperparameters
ann2 = MLPClassifier(
    hidden_layer_sizes=(50,),    # Single hidden layer with 50 neurons
    learning_rate_init=0.01,      # Initial learning rate
    batch_size=128,               # Batch size
    max_iter=500,                 # Maximum number of iterations
    random_state=100
)

# Train the second ANN
ann2.fit(x_train, y_train)

# Evaluate the second ANN on the validation set
y_pred_ann2 = ann2.predict(x_test)

accuracy_ann2 = accuracy_score(y_test, y_pred_ann2)
print("Accuracy (ANN2):", (accuracy_ann2*100), "%")
```

```
Accuracy (ANN2): 96.51666666666667 %
```

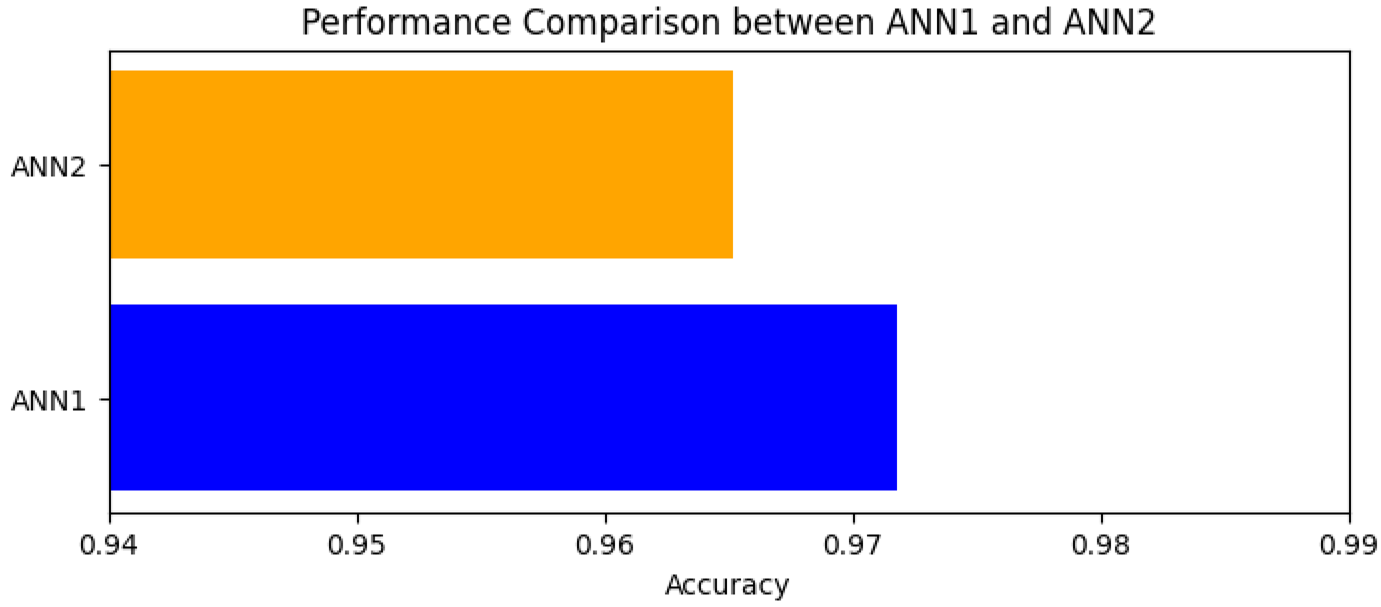
```
# Choose the best model based on validation accuracy
best_ann,best_ann_accuracy = (ann1,accuracy_ann1) if accuracy_ann1 >= accuracy_ann2 else (ann2,accuracy_ann2)

print("Best ANN architecture:", "ANN1" if accuracy_ann1 >= accuracy_ann2 else "ANN2")
```

```
Best ANN architecture: ANN1
```

```
# Plot the performance comparison as a horizontal bar plot with a smaller y-axis scale
labels = ['ANN1', 'ANN2']
accuracies = [accuracy_ann1, accuracy_ann2]

plt.figure(figsize=(8, 3))  # Smaller height for the plot
plt.barh(labels, accuracies, color=['blue', 'orange'])
plt.xlabel('Accuracy')
plt.title('Performance Comparison between ANN1 and ANN2')
plt.xlim(0.94, 0.99)  # Adjusted x-axis limit to focus on differences
plt.show()
```



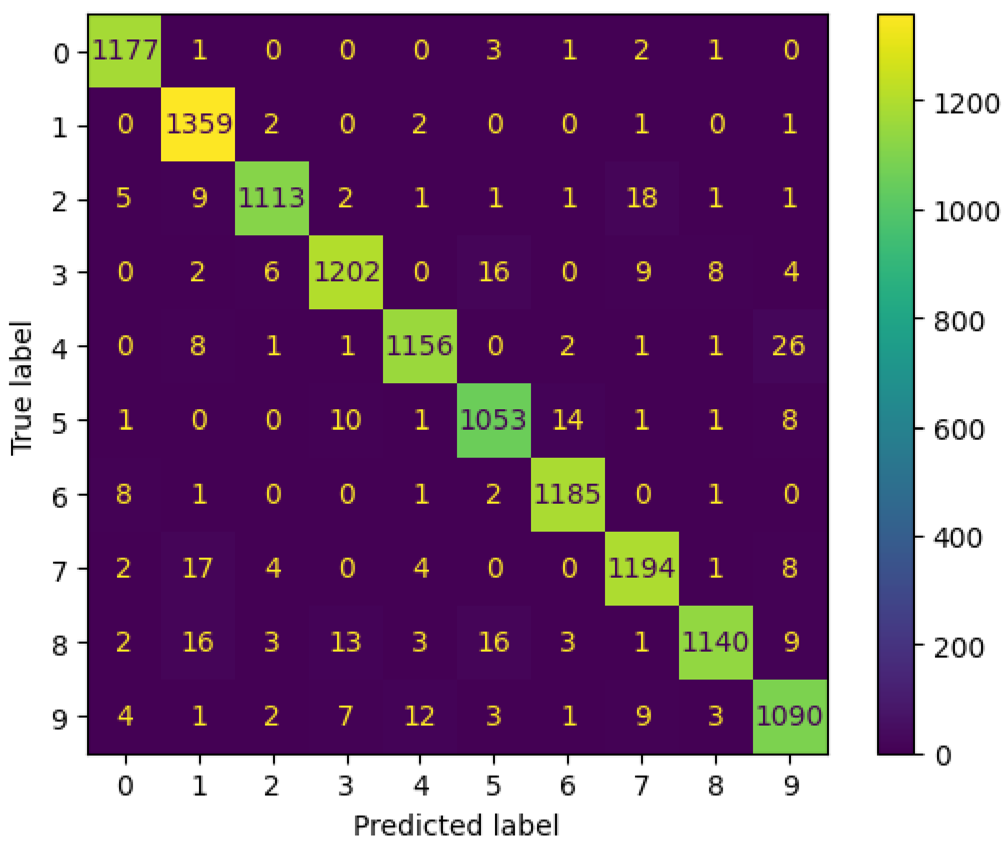
```
# Choose the best model based on validation accuracy
best_model,best_accuracy = (best_ann,best_ann_accuracy) if best_ann_accuracy >= best_knn_accuracy else (best_knn,best_knn_
print("Best Mode architecture:", "ANN" if best_ann_accuracy >= best_knn_accuracy else "K-NN")
```

```
Best Mode architecture: K-NN
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

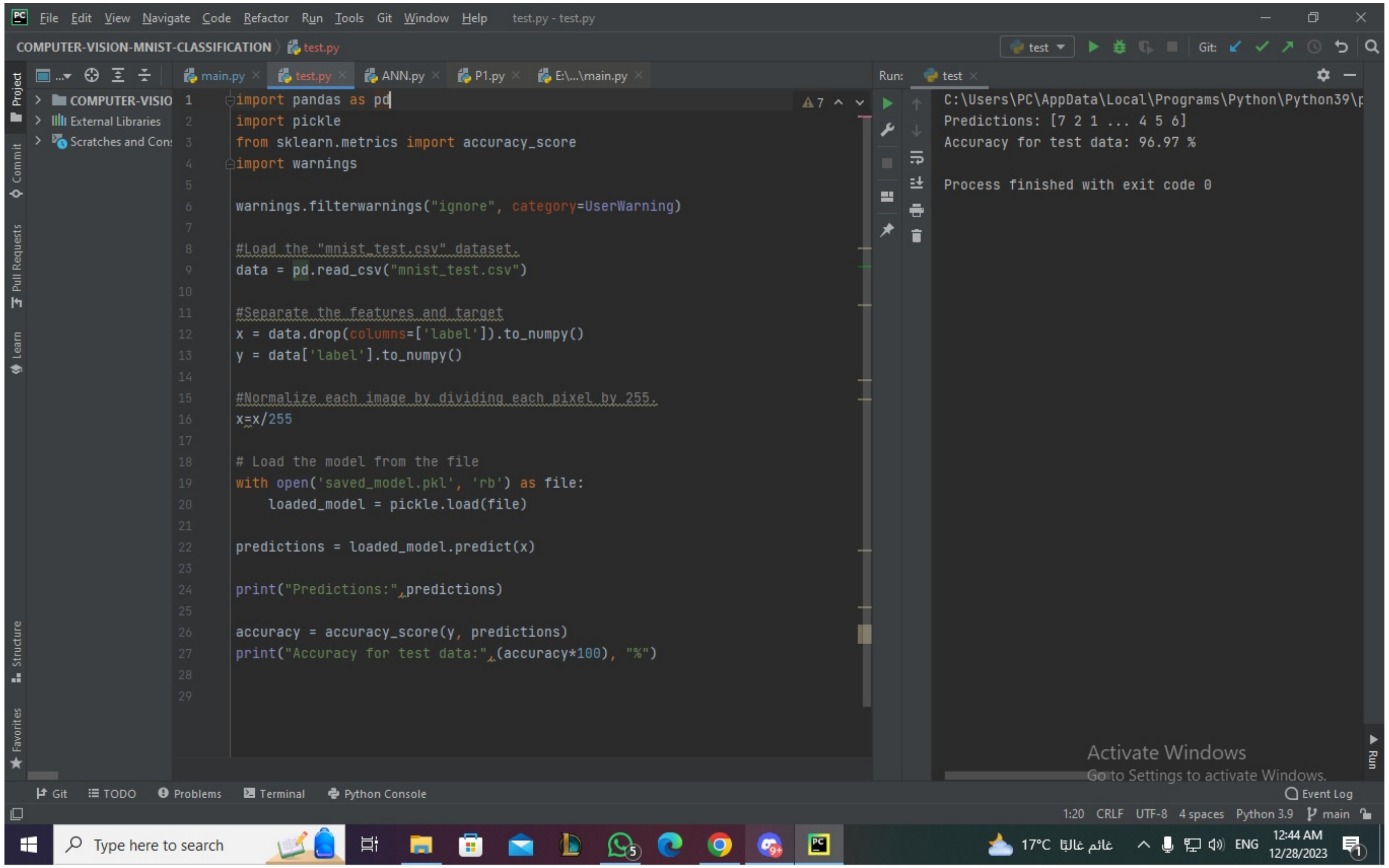
```
#Confusion matrix of the best model
predictions = best_model.predict(x_test)
confusion = confusion_matrix(y_test, predictions)
ConfusionMatrixDisplay(confusion, display_labels=[str(i) for i in range(10)]).plot(cmap='viridis')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7dd1588d0a60>
```




```
#Save the best model
import pickle

with open('saved_model.pkl', 'wb') as file:
    pickle.dump(best_model, file)
```


COMPUTER-VISION-MNIST-CLASSIFICATION
test.py
import pandas as pd
import pickle
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
#Load the "mnist_test.csv" dataset.
data = pd.read_csv("mnist_test.csv")
#Separate the features and target
x = data.drop(columns=['label']).to_numpy()
y = data['label'].to_numpy()
#Normalize each image by dividing each pixel by 255.
x=x/255
# Load the model from the file
with open('saved_model.pkl', 'rb') as file:
loaded_model = pickle.load(file)
predictions = loaded_model.predict(x)
print("Predictions:",predictions)
accuracy = accuracy_score(y, predictions)
print("Accuracy for test data:",(accuracy*100), "%")
Predictions: [7 2 1 ... 4 5 6]
Accuracy for test data: 96.97 %
Process finished with exit code 0
Activate Windows
Go to Settings to activate Windows.