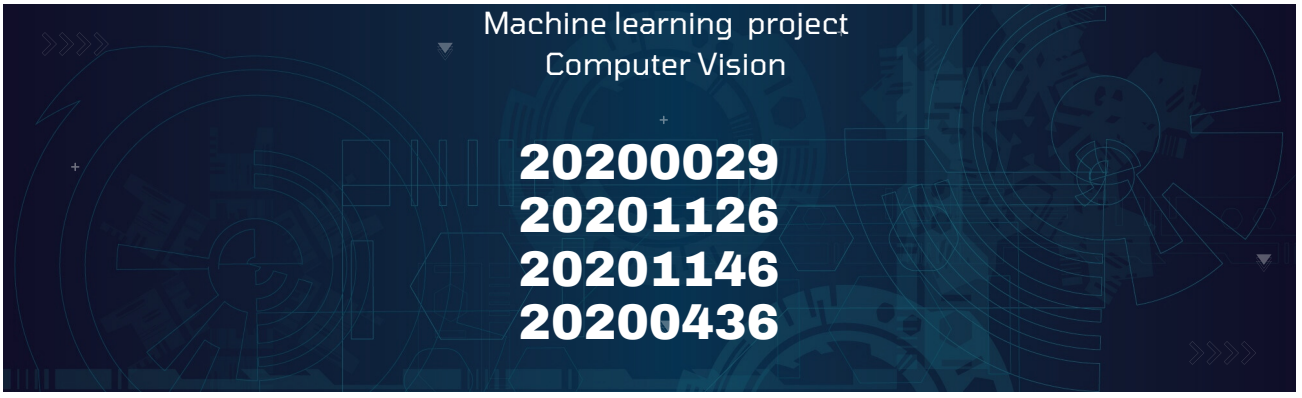


the project is about build two models to predicate the number exist on photo.
all data used you will find here : [MNIST in CSV (kaggle.com)]

build models.

we started with some data organization and preparation: finding nulls values, count the unique target classes, calculate the number of feature and others.

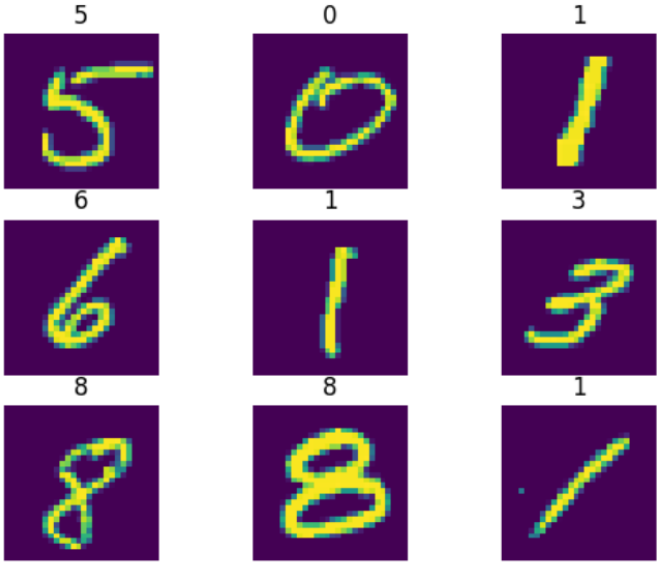
```
data = pd.read_csv("mnist_train.csv")
test_data = pd.read_csv("mnist_test.csv")
print("the number of unique classes is ",data['label'].nunique())
print("the number of feature is",data.shape[1]-1)
print("number of nulls in the data = ",data.isnull().sum().sum())
# the data of the image in a flat array not in matrix
print(data.shape)
```

```
the number of unique classes is  10
the number of feature is 784
number of nulls in the data =  0
(60000, 785)
```

```
#make allocation for the test and train data
X= data.iloc[:,1:]
X=X/255
y= data.iloc[:,0:1]
X_train,X_test,y_train,y_test= train_test_split(
        X,
        y,
        test_size=0.2,
        random_state=42
    )
X_train = X_train.copy().to_numpy()
X_test = X_test.copy().to_numpy()
y_train = y_train.copy().to_numpy().flatten()
y_test = y_test.copy().to_numpy().flatten()
print("The training digits data:\n", X_train.shape)
print("Digit labels: ", y_train.shape)
```

then we wanted to visualize some of the photos

```
]: for i in range(9):
       ax = plt.subplot(3, 3,i+1)
       plt.imshow(X_train[i].reshape(28, 28))
       plt.title(y_train[i])
       plt.axis("off")
```



and then we built out first model and it`s KNN and used grid search to find the best hyperparameters for that mode

```
[24]: knn = KNeighborsClassifier()
      # train the classifier
      knn.fit(X_train, y_train)
```

```
[24]: ▾ KNeighborsClassifier
      KNeighborsClassifier()
```

```
[227]:
space = [
     {"weights": ["uniform", "distance"],
      "n_neighbors": [2, 3, 5, 7]
     }
 ]

knn_grid_search = GridSearchCV(knn, param_grid=space, cv=3, scoring="accuracy", n_jobs=2)
knn_grid_search.fit(X_train, y_train)
print(knn_grid_search.best_estimator_)
```

```
KNeighborsClassifier(n_neighbors=3, weights='distance')
```

```
[25]: Knn_best_parameters = KNeighborsClassifier(n_neighbors=3, weights='distance')
      Knn_best_parameters.fit(X_train, y_train)
      y_pred = Knn_best_parameters.predict(X_test)
      knn_accuracy = accuracy_score(y_test, y_pred)
      print(knn_accuracy)
```

```
0.9735833333333334
```

and then we built two artificial neural networks using TensorFlow

```
[6]: # the dense don't need to put the shape of the input
     #MODEL 1
     model1= keras.Sequential([
          Dense(units=64,activation="relu"),
          Dense(units=10,activation="softmax")
     ])
     model1.compile(optimizer=Adam(learning_rate=0.001),loss=SparseCategoricalCrossentropy(),metrics=['accuracy'])
     model1.fit(X_train,y_train,epochs=10,batch_size=32)
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 881us/step - accuracy: 0.8444 - loss: 0.5533
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 1s 844us/step - accuracy: 0.9485 - loss: 0.1810
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 1s 831us/step - accuracy: 0.9636 - loss: 0.1269
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 1s 877us/step - accuracy: 0.9698 - loss: 0.1018
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 1s 911us/step - accuracy: 0.9765 - loss: 0.0788
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 1s 836us/step - accuracy: 0.9807 - loss: 0.0631
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 1s 868us/step - accuracy: 0.9844 - loss: 0.0515
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 1s 829us/step - accuracy: 0.9865 - loss: 0.0451
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 1s 886us/step - accuracy: 0.9899 - loss: 0.0368
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 2s 1000us/step - accuracy: 0.9902 - loss: 0.0320
6]: <keras.src.callbacks.history.History at 0x1f8c4052910>
```

and second one

```
[27]: #MODEL 2
      model2= keras.Sequential([
           Dense(units=32,activation="relu"),
           Dense(units=10,activation="softmax")
      ])
      model2.compile(optimizer=Adam(learning_rate=0.01),loss=SparseCategoricalCrossentropy(),metrics=['accuracy'])
      model2.fit(X_train,y_train,epochs=10,batch_size=128)
```

```
Epoch 1/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8558 - loss: 0.4687
Epoch 2/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 977us/step - accuracy: 0.9527 - loss: 0.1643
Epoch 3/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 964us/step - accuracy: 0.9610 - loss: 0.1293
Epoch 4/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 962us/step - accuracy: 0.9659 - loss: 0.1084
Epoch 5/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 958us/step - accuracy: 0.9708 - loss: 0.0944
Epoch 6/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 977us/step - accuracy: 0.9718 - loss: 0.0895
Epoch 7/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 973us/step - accuracy: 0.9734 - loss: 0.0846
Epoch 8/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 968us/step - accuracy: 0.9744 - loss: 0.0782
Epoch 9/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9778 - loss: 0.0708
Epoch 10/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9756 - loss: 0.0722
[27]: <keras.src.callbacks.history.History at 0x1f8e18aac90>
```

and used a comparation on them to choose the best one in term of accuracy.

```
[28]: loss1, accuracy1=model1.evaluate(X_test,y_test)
```

```
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 685us/step - accuracy: 0.9743 - loss: 0.0931
```

```
[29]: loss2, accuracy2=model2.evaluate(X_test,y_test)
```

```
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 646us/step - accuracy: 0.9630 - loss: 0.1476
```

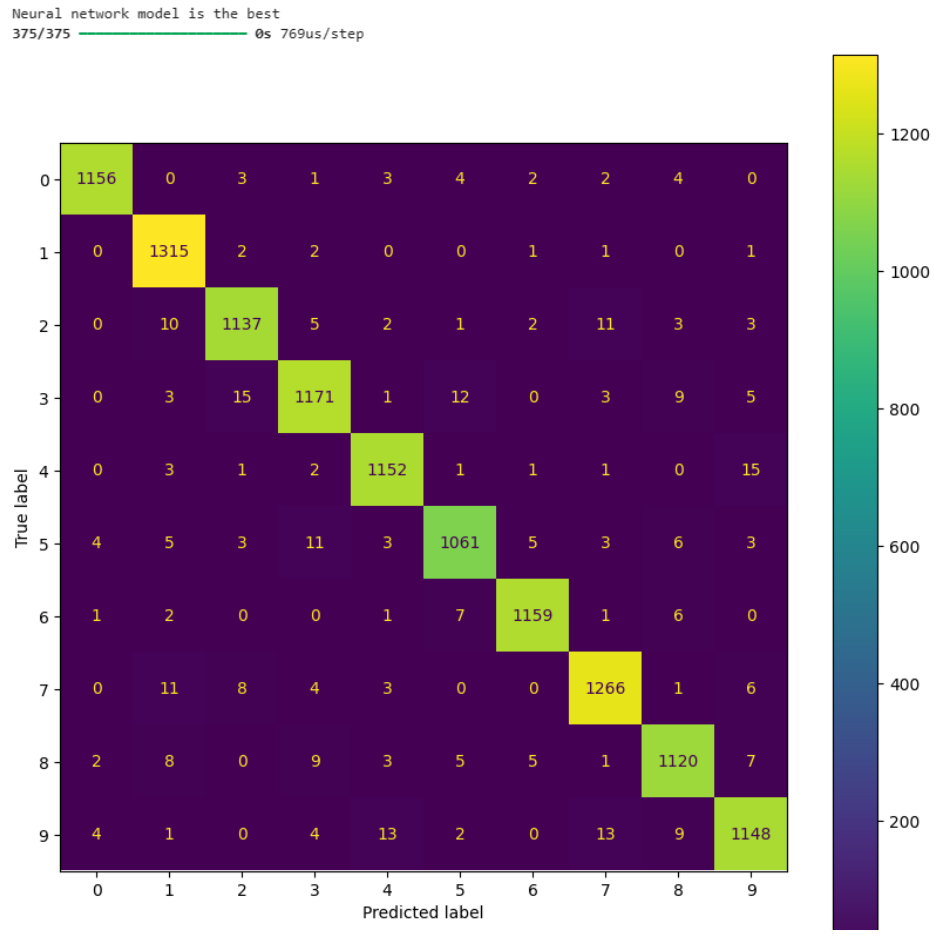
```
[30]: #get the best between the 2 different ANN
      best_ANN_model=model1
      best_ANN_accuracy = accuracy1
      if accuracy1<accuracy2:
          best_ANN_model=model2
          best_ANN_accuracy = accuracy2

      y_predict=best_ANN_model.predict(X_test)
      print(y_predict[0])
      #to get the argument of the max value in the y_predict[0]
      print("the prediction for the first example in test data = ",np.argmax(y_predict[0]))
```

```
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 759us/step
[3.2235343e-12 1.5515610e-09 8.6068624e-10 1.1859846e-09 2.3553405e-12
 8.7606381e-08 1.6092489e-14 9.9999797e-01 1.1299005e-11 1.9017278e-06]
the prediction for the first example in test data =  7
```

and finally, we compared the model of KNN with best hyperparameters with the model of ANN and save the model of it and compute the confusion matrix.

```
:  if(knn_accuracy>best_ANN_accuracy):
       print("KNN model is the best")
       best_model = Knn_best_parameters
       with open("best_model.joblib", "wb") as f:
           import joblib
           joblib.dump(best_model, f)
   else:
       print("Neural network model is the best")
       best_model = best_ANN_model
       best_model.save("best_model.keras")
       predicted_labels = best_model.predict(X_test)
       predicted_labels = np.argmax(predicted_labels, axis=1)
   conf_matrix = confusion_matrix(y_test, predicted_labels)
   cm_fig, cm_ax = plt.subplots(figsize=(10, 10))
   cf_mat_disp = ConfusionMatrixDisplay(conf_matrix)
   cf_mat_disp.plot(ax=cm_ax)
   plt.show()
```

and here`s the cm



test model.

we test our model by load the best one of them and test it on the test data

[11]:
```
try:
    with open("best_model.joblib", "rb") as f:
        import joblib
        knn_model = joblib.load(f)
except:
    print("no KNN model file exist")
    knn_model = None
if knn_model:
        knn_predictions = knn_model.predict(X)
        print("KNN accuracy:", accuracy_score(y, knn_predictions))
try:
    loaded_nn_model = load_model("best_model.keras")
except :
    print("no NN model file exist")
    loaded_nn_model = None
if loaded_nn_model:
        nn_predictions = loaded_nn_model.predict(X)
        loss, accuracy=loaded_nn_model.evaluate(X,y)
        print("Neural Network accuracy:", accuracy)
```

```
no KNN model file exist
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 667us/step - accuracy: 0.9645 - loss: 0.1153
Neural Network accuracy: 0.972100019454956
```