```
#Team
#Nora Mohamed Hussein 20201196
#Mariam mahomed elmoazen 20200528
#Heba Abdelwahab Sayed Abdelwahab 20201208
#Kholoud mohamed alkamkhli 20200846

# !pip install tensorflow==2.3.0
import pandas as pd
import numpy as np
import matplotlib.pyplot as plot
import tensorflow as tf
from sklearn.model_selection import train_test_split
import joblib
```

```
WARNING:tensorflow:From C:\Users\Kholoud\AppData\Local\Programs\
Python\Python310\lib\site-packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

```
data = pd.read_csv("mnist_train.csv")
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]
```

```
unique_classes = data.nunique()
print(f"Number of unique classes: {unique_classes}")
```

```
Number of unique classes: label    10
1x1        1
1x2        1
```

```
1x3       1
1x4       1
         ..
28x24     3
28x25     1
28x26     1
28x27     1
28x28     1
Length: 785, dtype: int64
```

```
num_of_features = data.shape[0]
print(f"Number of features: {num_of_features}")
```

```
Number of features: 60000
```

```
missing = data.isnull().sum()
print(f"Number of missing values: {missing}")
```

```
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
```

```
Y = data['label']
print(Y.head)
```

```
<bound method NDFrame.head of 0         5
1         0
2         4
3         1
4         9
        ..
59995     8
59996     3
59997     5
59998     6
59999     8
Name: label, Length: 60000, dtype: int64>
```

```
X = data.iloc[:, 1:]
print(X.head)
```

```
<bound method NDFrame.head of        1x1  1x2  1x3  1x4  1x5  1x6  1x7
1x8  1x9  1x10  ...  28x19  28x20  \
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
2        0    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        0    0    0    0    0    0    0    0    0     0  ...       0
0
...    ...  ...  ...  ...  ...  ...  ...  ...  ...   ...  ...     ...
...
59995    0    0    0    0    0    0    0    0    0     0  ...       0
0
59996    0    0    0    0    0    0    0    0    0     0  ...       0
0
59997    0    0    0    0    0    0    0    0    0     0  ...       0
0
59998    0    0    0    0    0    0    0    0    0     0  ...       0
0
59999    0    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 784 columns]>
```

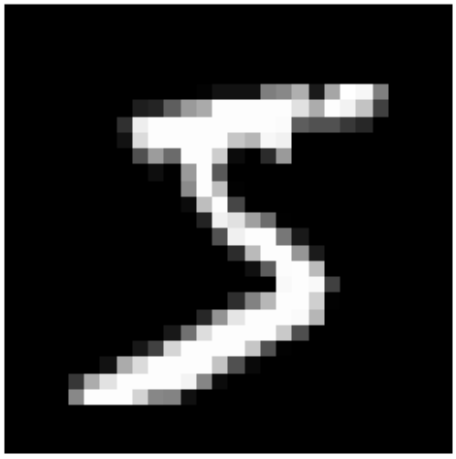
```
X = X/255.0
print(X)
```

```
       1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  1x10  ...  28x19
28x20  \
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
2      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
```

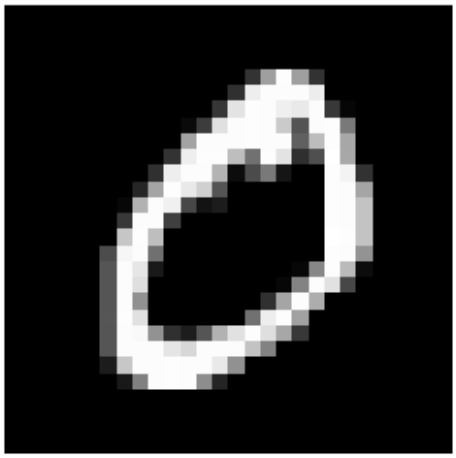
```
0.0
3      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
4      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
...    ...  ...  ...  ...  ...  ...  ...  ...  ...   ...  ...    ...
...
59995  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
59996  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  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  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  0.0  0.0  0.0  0.0  0.0  0.0  0.0  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    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
2        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
...      ...    ...    ...    ...    ...    ...    ...    ...
59995    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
59996    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
59997    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
59998    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
59999    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

[60000 rows x 784 columns]
```

```python
from PIL import Image
import matplotlib.pyplot as plt
def convert_to_image(row,width = 28 , height = 28):
    pixels = np.array(row).reshape(28,28)
    image = Image.fromarray(pixels.astype('uint8'))
    resized_image = image.resize((width, height))
    fig, axes = plt.subplots(1, 1, figsize=(3, 3))
    axes.imshow(pixels, cmap='gray')
    axes.set_title('Original Image')
    axes.axis('off')
for i in range(7):
    convert_to_image(X.iloc[i])
```

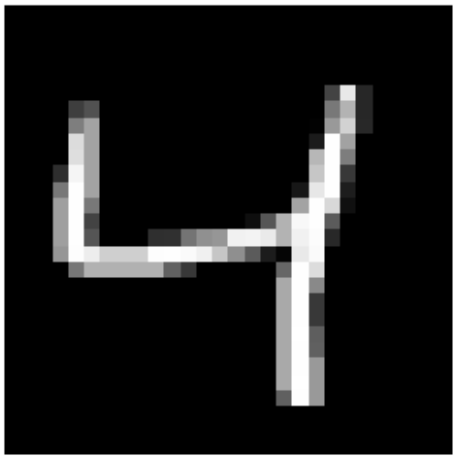
Original Image



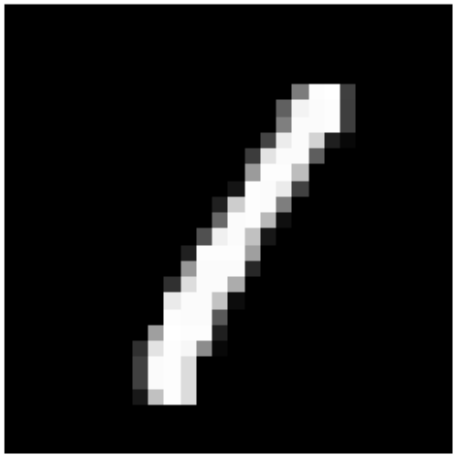
Original Image



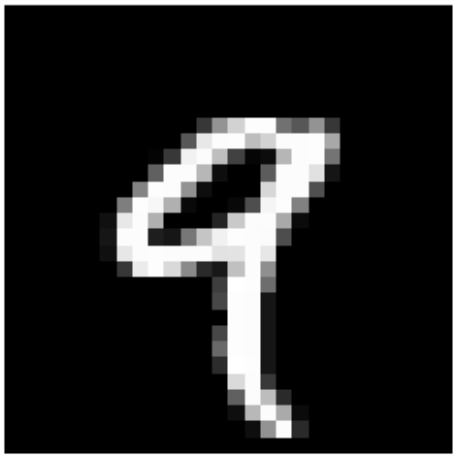
Original Image



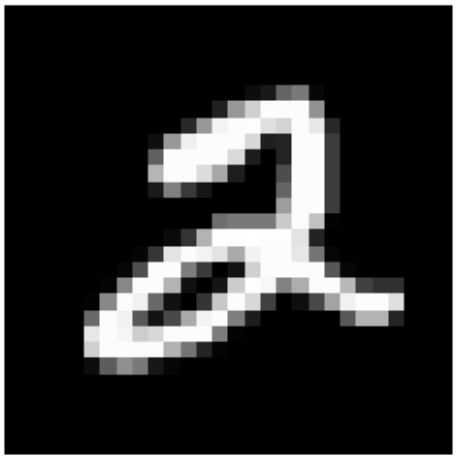
Original Image

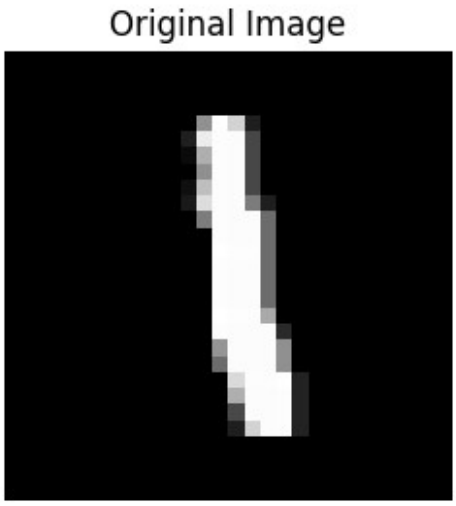


Original Image



Original Image






```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score,mean_squared_error,r2_score

X_train, X_test , y_train , y_test = train_test_split(X,Y,test_size =
0.2 , random_state = 42)
knn_model = KNeighborsClassifier(n_neighbors = 5)
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
accuracy = accuracy_score(y_pred,y_test)

print(y_pred[0:5])
print(y_test[0:5])

Hello
[7 3 8 9 3]
12628    7
37730    3
39991    8
8525     9
8279     3
Name: label, dtype: int64

print(f"accuracy score: {accuracy}")

accuracy score: 0.9715

ranges_of_k  =  [3, 5, 9, 11]

knn = KNeighborsClassifier()
param_grid = dict(n_neighbors=ranges_of_k)
grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy',
```

```
return_train_score=False,verbose=1)
# fitting the model for grid search
grid_search=grid.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```

```
print(grid_search.best_params_)
```

```
{'n_neighbors': 3}
```

```
best_knn_model = grid_search.best_estimator_
y_pred_ = best_knn_model.predict(X_test)
accuracy = accuracy_score(y_pred_, y_test)
print(f"Validation Accuracy: {accuracy:.4f}")
```

```
Validation Accuracy: 0.9727
```

```
test_data = pd.read_csv('mnist_test.csv')
```

```
Y_test_data = test_data['label']
X_test_data = test_data.iloc[:, 1:]
print(Y_test_data)
print(X_test_data)
```

```
0       7
1       2
2       1
3       0
4       4
       ..
9995    2
9996    3
9997    4
9998    5
9999    6
Name: label, Length: 10000, dtype: int64
      1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  1x10  ...  28x19
28x20  \
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
2       0    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       0    0    0    0    0    0    0    0    0     0  ...      0
0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...   ...  ...    ...
...
9995    0    0    0    0    0    0    0    0    0     0  ...      0
```

```
0
9996    0    0    0    0    0    0    0    0    0    0  ...       0
0
9997    0    0    0    0    0    0    0    0    0    0  ...       0
0
9998    0    0    0    0    0    0    0    0    0    0  ...       0
0
9999    0    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
...     ...    ...    ...    ...    ...    ...    ...    ...
9995      0      0      0      0      0      0      0      0
9996      0      0      0      0      0      0      0      0
9997      0      0      0      0      0      0      0      0
9998      0      0      0      0      0      0      0      0
9999      0      0      0      0      0      0      0      0

[10000 rows x 784 columns]
```

```
X_test_data = X_test_data/255.0
print(X_test_data)
```

```
      1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  1x10  ...  28x19
28x20  \
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
2     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     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     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...   ...  ...    ...
...
9995  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
9996  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
9997  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
9998  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
```

```
9999  0.0  0.0  0.0  0.0  0.0  0.0  0.0  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    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
2       0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3       0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4       0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
...     ...    ...    ...    ...    ...    ...    ...    ...
9995    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9996    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9997    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9998    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9999    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

[10000 rows x 784 columns]

y_test_predictions = best_knn_model.predict(X_test_data)

accuracy = accuracy_score(y_test_predictions, Y_test_data)
print(f"Validation Accuracy: {accuracy:.4f}")
print(y_test_predictions)
print(Y_test_data)

Validation Accuracy: 0.9681
[7 2 1 ... 4 5 6]
0       7
1       2
2       1
3       0
4       4
       ..
9995    2
9996    3
9997    4
9998    5
9999    6
Name: label, Length: 10000, dtype: int64

X_train, X_test , y_train , y_test = train_test_split(X,Y,test_size =
0.2 , random_state = 42)

X_train = np.asarray(X_train).astype(np.float32)
y_train = np.asarray(y_train).astype(np.int32)
X_test = np.asarray(X_test).astype(np.float32)
y_test = np.asarray(y_test).astype(np.int32)
```

Architecture 1

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(128, activation = 'relu'))
model.add(tf.keras.layers.Dense(128, activation = 'relu'))

# output layer
model.add(tf.keras.layers.Dense(10, activation = 'softmax'))

model.compile(optimizer="adam",
loss="sparse_categorical_crossentropy", metrics=["accuracy"])

model.fit(X_train, y_train, epochs=5, validation_split=0.3)
```

```
Epoch 1/5
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at
0x0000029C02F08550> and will run it as-is.
Cause: Unable to locate the source code of <function
Model.make_train_function.<locals>.train_function at
0x0000029C02F08550>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their
source code. If that is the case, you should define them in a .py
source file. If you are certain the code is graph-compatible, wrap the
call using @tf.autograph.experimental.do_not_convert. Original error:
could not get source code
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at
0x0000029C02F08550> and will run it as-is.
Cause: Unable to locate the source code of <function
Model.make_train_function.<locals>.train_function at
0x0000029C02F08550>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their
source code. If that is the case, you should define them in a .py
source file. If you are certain the code is graph-compatible, wrap the
call using @tf.autograph.experimental.do_not_convert. Original error:
could not get source code
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING:tensorflow:From C:\Users\Kholoud\AppData\Local\Programs\
Python\Python310\lib\site-packages\keras\src\utils\tf_utils.py:492:
The name tf.ragged.RaggedTensorValue is deprecated. Please use
tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Kholoud\AppData\Local\Programs\
Python\Python310\lib\site-packages\keras\src\engine\
base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
```

```
tf.compat.v1.executing_eagerly_outside_functions instead.

1021/1050 [============================>.] - ETA: 0s - loss: 0.3036 -
accuracy: 0.9106WARNING:tensorflow:AutoGraph could not transform
<function Model.make_test_function.<locals>.test_function at
0x0000029C054E49D0> and will run it as-is.
Cause: Unable to locate the source code of <function
Model.make_test_function.<locals>.test_function at
0x0000029C054E49D0>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their
source code. If that is the case, you should define them in a .py
source file. If you are certain the code is graph-compatible, wrap the
call using @tf.autograph.experimental.do_not_convert. Original error:
could not get source code
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_test_function.<locals>.test_function at 0x0000029C054E49D0>
and will run it as-is.
Cause: Unable to locate the source code of <function
Model.make_test_function.<locals>.test_function at
0x0000029C054E49D0>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their
source code. If that is the case, you should define them in a .py
source file. If you are certain the code is graph-compatible, wrap the
call using @tf.autograph.experimental.do_not_convert. Original error:
could not get source code
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
1050/1050 [==============================] - 5s 2ms/step - loss:
0.3001 - accuracy: 0.9117 - val_loss: 0.1697 - val_accuracy: 0.9504
Epoch 2/5
1050/1050 [==============================] - 2s 2ms/step - loss:
0.1272 - accuracy: 0.9618 - val_loss: 0.1321 - val_accuracy: 0.9584
Epoch 3/5
1050/1050 [==============================] - 2s 2ms/step - loss:
0.0861 - accuracy: 0.9739 - val_loss: 0.1041 - val_accuracy: 0.9667
Epoch 4/5
1050/1050 [==============================] - 2s 2ms/step - loss:
0.0604 - accuracy: 0.9812 - val_loss: 0.1124 - val_accuracy: 0.9660
Epoch 5/5
1050/1050 [==============================] - 2s 2ms/step - loss:
0.0479 - accuracy: 0.9848 - val_loss: 0.1359 - val_accuracy: 0.9601

<keras.src.callbacks.History at 0x29c02edbf10>
```

Architecture 2

```
model2 = tf.keras.models.Sequential()
```

```
model2.add(tf.keras.layers.Dense(256, activation = 'relu'))
model2.add(tf.keras.layers.Dense(128, activation = 'relu'))
model2.add(tf.keras.layers.Dense(64, activation='relu'))

# output layer
model2.add(tf.keras.layers.Dense(10, activation = 'softmax'))

custom_optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model2.compile(optimizer=custom_optimizer,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model2.fit(X_train, y_train, epochs=5, validation_split=0.3)

Epoch 1/5
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at
0x0000029C054E52D0> and will run it as-is.
Cause: Unable to locate the source code of <function
Model.make_train_function.<locals>.train_function at
0x0000029C054E52D0>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their
source code. If that is the case, you should define them in a .py
source file. If you are certain the code is graph-compatible, wrap the
call using @tf.autograph.experimental.do_not_convert. Original error:
could not get source code
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at
0x0000029C054E52D0> and will run it as-is.
Cause: Unable to locate the source code of <function
Model.make_train_function.<locals>.train_function at
0x0000029C054E52D0>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their
source code. If that is the case, you should define them in a .py
source file. If you are certain the code is graph-compatible, wrap the
call using @tf.autograph.experimental.do_not_convert. Original error:
could not get source code
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
1029/1050 [============================>.] - ETA: 0s - loss: 0.2760 -
accuracy: 0.9188WARNING:tensorflow:AutoGraph could not transform
<function Model.make_test_function.<locals>.test_function at
0x0000029C054E6680> and will run it as-is.
Cause: Unable to locate the source code of <function
Model.make_test_function.<locals>.test_function at
0x0000029C054E6680>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their
source code. If that is the case, you should define them in a .py
source file. If you are certain the code is graph-compatible, wrap the
```

```
call using @tf.autograph.experimental.do_not_convert. Original error:
could not get source code
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_test_function.<locals>.test_function at 0x0000029C054E6680>
and will run it as-is.
Cause: Unable to locate the source code of <function
Model.make_test_function.<locals>.test_function at
0x0000029C054E6680>. Note that functions defined in certain
environments, like the interactive Python shell, do not expose their
source code. If that is the case, you should define them in a .py
source file. If you are certain the code is graph-compatible, wrap the
call using @tf.autograph.experimental.do_not_convert. Original error:
could not get source code
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
1050/1050 [==============================] - 3s 2ms/step - loss:
0.2731 - accuracy: 0.9195 - val_loss: 0.1370 - val_accuracy: 0.9572
Epoch 2/5
1050/1050 [==============================] - 3s 2ms/step - loss:
0.1150 - accuracy: 0.9651 - val_loss: 0.1135 - val_accuracy: 0.9662
Epoch 3/5
1050/1050 [==============================] - 2s 2ms/step - loss:
0.0786 - accuracy: 0.9755 - val_loss: 0.1151 - val_accuracy: 0.9675
Epoch 4/5
1050/1050 [==============================] - 2s 2ms/step - loss:
0.0587 - accuracy: 0.9810 - val_loss: 0.1278 - val_accuracy: 0.9639
Epoch 5/5
1050/1050 [==============================] - 2s 2ms/step - loss:
0.0449 - accuracy: 0.9853 - val_loss: 0.1307 - val_accuracy: 0.9641

<keras.src.callbacks.History at 0x29c0bd5ce20>
```

```
ann_model1_accuracy=model.evaluate( X_test,y_test)[1]  #evaluate
return loss - accuracy
ann_model2_accuracy=model2.evaluate( X_test,y_test)[1]

if(ann_model1_accuracy>ann_model2_accuracy):
  best_ann = model
else:
  best_ann = model2

print("ann_model1: ",ann_model1_accuracy)
print("ann_model2: ",ann_model2_accuracy)

print(best_ann.evaluate( X_test,y_test)[1])
```

```
375/375 [==============================] - 0s 875us/step - loss:
0.1309 - accuracy: 0.9618
```

```
375/375 [==============================] - 0s 930us/step - loss:
0.1260 - accuracy: 0.9668
ann_model1:  0.9618333578109741
ann model2:  0.9667500257492065
375/375 [==============================] - 0s 1ms/step - loss: 0.1260
- accuracy: 0.9668
0.9667500257492065
```

```
best_ann_accu=best_ann.evaluate( X_test,y_test)[1]
y_predict = best_knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test,y_predict)

print("best_ann_accu: ",best_ann_accu)
print("knn_accu: ",knn_accuracy)

if(knn_accuracy > best_ann_accu):
  print("knn_accu is better than ann acuuracy: ",knn_accuracy)
  best_model = best_knn_model
else:
  print("ann_accu is better than knn acuuracy: ",best_ann_accu)

  best_model = best_ann

conf_matrix = confusion_matrix(y_test, best_model.predict(X_test))
print ("conf_matrix: \n",conf_matrix)
```

```
375/375 [==============================] - 1s 1ms/step - loss: 0.1260
- accuracy: 0.9668

C:\Users\Kholoud\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:465: UserWarning: X does not have valid
feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(

best_ann_accu:  0.9667500257492065
knn_accu:  0.9726666666666667
knn_accu is better than ann acuuracy:  0.9726666666666667

C:\Users\Kholoud\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:465: UserWarning: X does not have valid
feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(

conf_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]]
```

```
joblib.dump(best_model, 'best_model_.joblib')
```

```
['best_model_.joblib']
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plot
from sklearn.model_selection import train_test_split
import joblib
from sklearn.metrics import accuracy_score,mean_squared_error,r2_score

# Reload model from the file
loaded_bestmodel = joblib.load('best_model_.joblib')

test_data = pd.read_csv('mnist_test.csv')

Y_test_data = test_data['label']
X_test_data = test_data.iloc[:, 1:]
print(Y_test_data)
print(X_test_data)
```

```
0       7
1       2
2       1
3       0
4       4
       ..
9995    2
9996    3
9997    4
9998    5
9999    6
Name: label, Length: 10000, dtype: int64
      1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  1x10  ...  28x19
28x20  \
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
2       0    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       0    0    0    0    0    0    0    0    0     0  ...      0
0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...   ...  ...    ...
...
9995    0    0    0    0    0    0    0    0    0     0  ...      0
0
9996    0    0    0    0    0    0    0    0    0     0  ...      0
0
9997    0    0    0    0    0    0    0    0    0     0  ...      0
0
9998    0    0    0    0    0    0    0    0    0     0  ...      0
```

```
0
9999    0    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
...     ...    ...    ...    ...    ...    ...    ...    ...
9995      0      0      0      0      0      0      0      0
9996      0      0      0      0      0      0      0      0
9997      0      0      0      0      0      0      0      0
9998      0      0      0      0      0      0      0      0
9999      0      0      0      0      0      0      0      0

[10000 rows x 784 columns]
```

```
X_test_data = X_test_data/255.0
print(X_test_data)
```

```
      1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  1x10  ...  28x19
28x20  \
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
2     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     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     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...   ...  ...    ...
...
9995  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
9996  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
9997  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
9998  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0  ...    0.0
0.0
9999  0.0  0.0  0.0  0.0  0.0  0.0  0.0  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    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
```

```
2       0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3       0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4       0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
...     ...    ...    ...    ...    ...    ...    ...    ...
9995    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9996    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9997    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9998    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9999    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

[10000 rows x 784 columns]
```

```
y_predictions = loaded_bestmodel.predict(X_test_data)

# Calculate and print the accuracy and confusion matrix
accuracy = accuracy_score(Y_test_data, y_predictions)
print("accuracy of best model on mnist_test.csv : ",accuracy)
```

```
accuracy of best model on mnist_test.csv :  0.9681
```