Importing libraries

```
import pathlib
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics, tree, ensemble
from sklearn.utils import shuffle
import tensorflow as tf
from google.colab import files
```

Checking if device is GPU

```
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

Uploading Kaggle token

```
files.upload()
```

```
Choose Files kaggle.json
• kaggle.json(application/json) - 71 bytes, last modified: 11/19/2023 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json':
b'{"username":"mahmuedalardawi","key":"dd071d7a5b7237c6d363d0cd48ba2095"}'}
```

Importing Kaggle dataset

```
! pip install kaggle
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.16)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.2.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.2)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7)
```

Downloading the Arabic Handwritten Characters Dataset

```
! kaggle datasets download -d mloey1/ahcd1
```

```
Downloading ahcd1.zip to /content
 21% 5.00M/24.0M [00:00<00:00, 46.2MB/s]
100% 24.0M/24.0M [00:00<00:00, 121MB/s]
```

Unzipping the dataset to folder Datasets

```
!unzip ahcd1.zip -d Datasets
```

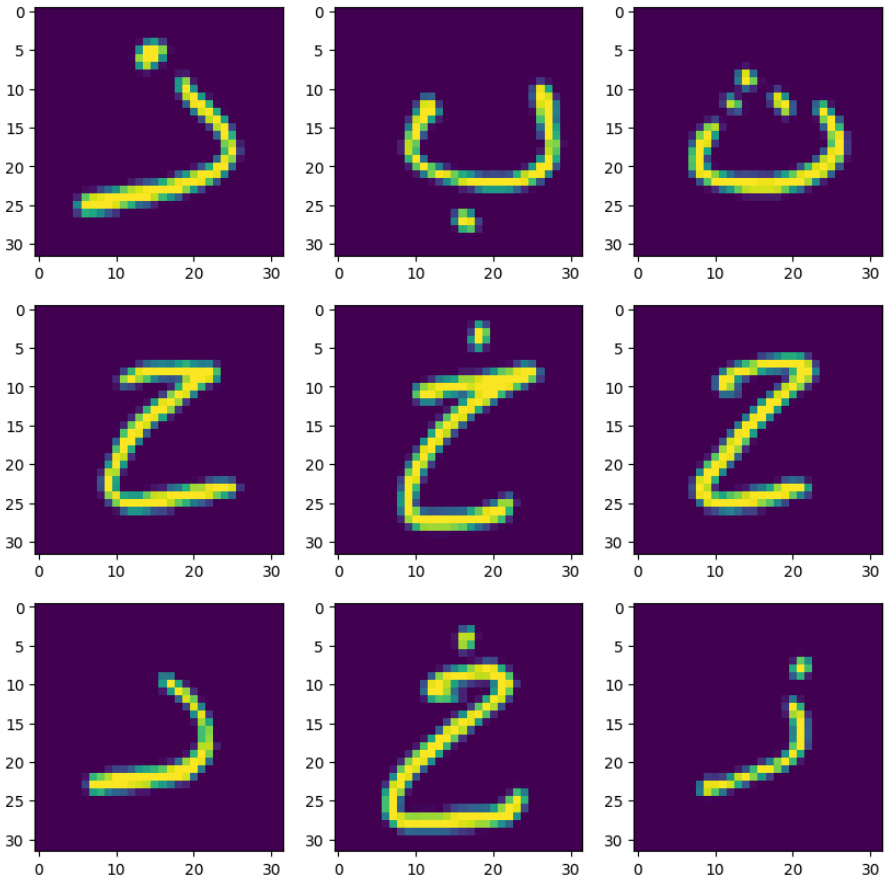
```
inflating: Datasets/train images 13440x32x32/train/id_9958_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_9959_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_995_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_9960_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_9961_label_14.png
inflating: Datasets/train images 13440x32x32/train/id_9962_label_14.png
inflating: Datasets/train images 13440x32x32/train/id_9963_label_14.png
inflating: Datasets/train images 13440x32x32/train/id_9964_label_14.png
inflating: Datasets/train images 13440x32x32/train/id_9965_label_14.png
inflating: Datasets/train images 13440x32x32/train/id_9966_label_14.png
inflating: Datasets/train images 13440x32x32/train/id_9967_label_14.png
inflating: Datasets/train images 13440x32x32/train/id_9968_label_14.png
inflating: Datasets/train images 13440x32x32/train/id_9969_label_15.png
inflating: Datasets/train images 13440x32x32/train/id_996_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_9970_label_15.png
inflating: Datasets/train images 13440x32x32/train/id_9971_label_15.png
inflating: Datasets/train images 13440x32x32/train/id_9972_label_15.png
inflating: Datasets/train images 13440x32x32/train/id_9973_label_15.png
inflating: Datasets/train images 13440x32x32/train/id_9974_label_15.png
inflating: Datasets/train images 13440x32x32/train/id_9975_label_15.png
inflating: Datasets/train images 13440x32x32/train/id_9976_label_15.png
inflating: Datasets/train images 13440x32x32/train/id_9977_label_16.png
inflating: Datasets/train images 13440x32x32/train/id_9978_label_16.png
inflating: Datasets/train images 13440x32x32/train/id_9979_label_16.png
inflating: Datasets/train images 13440x32x32/train/id_997_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_9980_label_16.png
inflating: Datasets/train images 13440x32x32/train/id_9981_label_16.png
inflating: Datasets/train images 13440x32x32/train/id_9982_label_16.png
inflating: Datasets/train images 13440x32x32/train/id_9983_label_16.png
inflating: Datasets/train images 13440x32x32/train/id_9984_label_16.png
inflating: Datasets/train images 13440x32x32/train/id_9985_label_17.png
inflating: Datasets/train images 13440x32x32/train/id_9986_label_17.png
inflating: Datasets/train images 13440x32x32/train/id_9987_label_17.png
inflating: Datasets/train images 13440x32x32/train/id_9988_label_17.png
inflating: Datasets/train images 13440x32x32/train/id_9989_label_17.png
inflating: Datasets/train images 13440x32x32/train/id_998_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_9990_label_17.png
inflating: Datasets/train images 13440x32x32/train/id_9991_label_17.png
inflating: Datasets/train images 13440x32x32/train/id_9992_label_17.png
inflating: Datasets/train images 13440x32x32/train/id_9993_label_18.png
inflating: Datasets/train images 13440x32x32/train/id_9994_label_18.png
inflating: Datasets/train images 13440x32x32/train/id_9995_label_18.png
inflating: Datasets/train images 13440x32x32/train/id_9996_label_18.png
inflating: Datasets/train images 13440x32x32/train/id_9997_label_18.png
inflating: Datasets/train images 13440x32x32/train/id_9998_label_18.png
inflating: Datasets/train images 13440x32x32/train/id_9999_label_18.png
inflating: Datasets/train images 13440x32x32/train/id_999_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_99_label_13.png
inflating: Datasets/train images 13440x32x32/train/id_9_label_2.png
```

Loading the data

```
dir = pathlib.Path("/content/Datasets/Arabic Handwritten Characters Dataset CSV")

# training data features
train_data = pd.read_csv(dir / "csvTrainImages 13440x1024.csv", header=None)

# training data target
train_target = pd.read_csv(dir / "csvTrainLabel 13440x1.csv", header=None)

# testing data features
test_data = pd.read_csv(dir / "csvTestImages 3360x1024.csv", header=None)

# testing data target
test_target = pd.read_csv(dir / "csvTestLabel 3360x1.csv", header=None)
```

showing the datasets

```
shuff = shuffle(train_data[:100])
fig, ax = plt.subplots(3,3, figsize = (10,10))
axes = ax.flatten()
for i in range(9):
  axes[i].imshow(shuff.values[i].reshape(32,32).transpose(1,0))
plt.show()
```



```
print(train_data)
print(train_target)
```

```
          0     1     2     3     4     5     6     7     8     9     ...  1014  \
0         0     0     0     0     0     0     0     0     0     0  ...     0
1         0     0     0     0     0     0     0     0     0     0  ...     0
2         0     0     0     0     0     0     0     0     0     0  ...     0
3         0     0     0     0     0     0     0     0     0     0  ...     0
4         0     0     0     0     0     0     0     0     0     0  ...     0
...     ...   ...   ...   ...   ...   ...   ...   ...   ...   ...  ...   ...
13435     0     0     0     0     0     0     0     0     0     0  ...     0
13436     0     0     0     0     0     0     0     0     0     0  ...     0
13437     0     0     0     0     0     0     0     0     0     0  ...     0
13438     0     0     0     0     0     0     0     0     0     0  ...     0
13439     0     0     0     0     0     0     0     0     0     0  ...     0

         1015  1016  1017  1018  1019  1020  1021  1022  1023
0         0     0     0     0     0     0     0     0     0
1         0     0     0     0     0     0     0     0     0
2         0     0     0     0     0     0     0     0     0
3         0     0     0     0     0     0     0     0     0
4         0     0     0     0     0     0     0     0     0
...     ...   ...   ...   ...   ...   ...   ...   ...   ...
13435     0     0     0     0     0     0     0     0     0
13436     0     0     0     0     0     0     0     0     0
13437     0     0     0     0     0     0     0     0     0
13438     0     0     0     0     0     0     0     0     0
13439     0     0     0     0     0     0     0     0     0

[13440 rows x 1024 columns]
        0
0       1
1       1
2       1
3       1
```

```
4       1
...    ..
13435  28
13436  28
13437  28
13438  28
13439  28

[13440 rows x 1 columns]
```

```
print(test_data)
print(test_target)
```

```
      0    1    2    3    4    5    6    7    8    9    ...  1014  \
0        0    0    0    0    0    0    0    0    0    0  ...     0
1        0    0    0    0    0    0    0    0    0    0  ...     0
2        0    0    0    0    0    0    0    0    0    0  ...     0
3        0    0    0    0    0    0    0    0    0    0  ...     0
4        0    0    0    0    0    0    0    0    0    0  ...     0
...    ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...   ...
3355     0    0    0    0    0    0    0    0    0    0  ...     0
3356     0    0    0    0    0    0    0    0    0    0  ...     0
3357     0    0    0    0    0    0    0    0    0    0  ...     0
3358     0    0    0    0    0    0    0    0    0    0  ...     0
3359     0    0    0    0    0    0    0    0    0    0  ...     0

      1015  1016  1017  1018  1019  1020  1021  1022  1023
0        0     0     0     0     0     0     0     0     0
1        0     0     0     0     0     0     0     0     0
2        0     0     0     0     0     0     0     0     0
3        0     0     0     0     0     0     0     0     0
4        0     0     0     0     0     0     0     0     0
...    ...   ...   ...   ...   ...   ...   ...   ...   ...
3355     0     0     0     0     0     0     0     0     0
3356     0     0     0     0     0     0     0     0     0
3357     0     0     0     0     0     0     0     0     0
3358     0     0     0     0     0     0     0     0     0
3359     0     0     0     0     0     0     0     0     0

[3360 rows x 1024 columns]
       0
0      1
1      1
2      2
3      2
4      3
...   ..
3355  26
3356  27
3357  27
3358  28
3359  28

[3360 rows x 1 columns]
```

ML(Supervised): Decision Tree Classifier

Creating a decision tree classifier

```
dtc = tree.DecisionTreeClassifier().fit(train_data, train_target)
```

Making predictions

```
dtc_predections = dtc.predict(test_data)
```

Finding model accuarcy

```
accuracy = metrics.accuracy_score(test_target, dtc_predections)

print(accuracy)
```

```
0.36339285714285713
```

Generating classification report

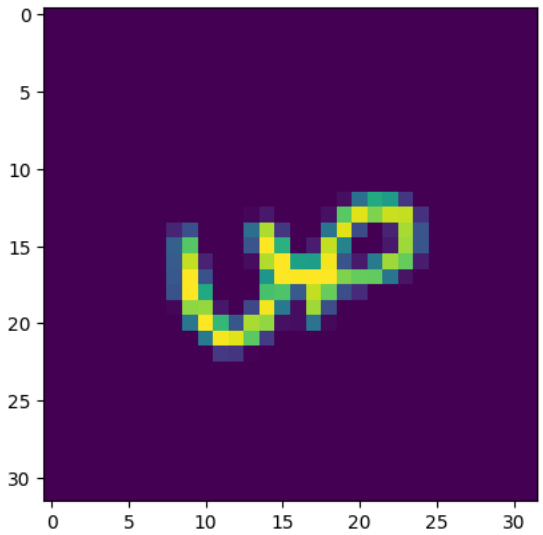
```
dtc_report = metrics.classification_report(test_target, dtc_predections)
print(dtc_report)
```

```
                   precision    recall  f1-score   support

                1       0.70      0.75      0.72       120
                2       0.41      0.49      0.45       120
                3       0.26      0.26      0.26       120
                4       0.28      0.27      0.27       120
                5       0.37      0.38      0.38       120
                6       0.46      0.42      0.44       120
                7       0.28      0.28      0.28       120
                8       0.36      0.42      0.39       120
                9       0.32      0.36      0.34       120
               10       0.50      0.57      0.54       120
               11       0.40      0.44      0.42       120
               12       0.26      0.28      0.27       120
               13       0.33      0.25      0.29       120
               14       0.30      0.31      0.30       120
               15       0.27      0.24      0.25       120
               16       0.32      0.32      0.32       120
               17       0.26      0.23      0.24       120
               18       0.28      0.30      0.29       120
               19       0.36      0.34      0.35       120
               20       0.31      0.33      0.32       120
               21       0.19      0.17      0.18       120
               22       0.46      0.40      0.43       120
               23       0.52      0.53      0.53       120
               24       0.50      0.51      0.50       120
               25       0.30      0.28      0.29       120
               26       0.41      0.38      0.39       120
               27       0.40      0.38      0.39       120
               28       0.28      0.27      0.27       120

         accuracy                           0.36      3360
        macro avg       0.36      0.36      0.36      3360
     weighted avg       0.36      0.36      0.36      3360
```

Implementation

```
img = random.randint(0, len(dtc_predections) - 1)
print("Predection =", dtc_predections[img])
plt.imshow(test_data.values[img].reshape(32,32).transpose(1,0))
```

```
Predection = 14
<matplotlib.image.AxesImage at 0x7e70e07d2c50>
```



ML(Supervised): Random Forest Classifier

Creating a random forest classifier

```
rfc = ensemble.RandomForestClassifier().fit(train_data, train_target)
```

```
<ipython-input-16-d5ed9d04c9b6>:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the s
  rfc = ensemble.RandomForestClassifier().fit(train_data, train_target)
```

Making predictions

```
rfc_predections = rfc.predict(test_data)
```

Finding model accuarcy

```
accuracy = metrics.accuracy_score(test_target, rfc_predections)

print(accuracy)
```

```
0.6806547619047619
```

Generating classification report

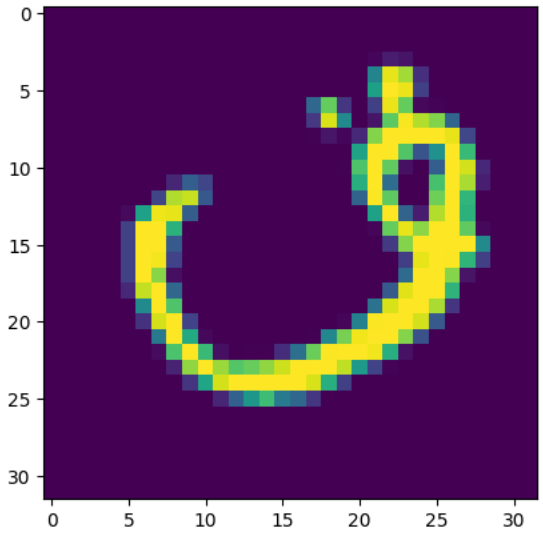
```
rfc_report = metrics.classification_report(test_target, rfc_predections)
print(rfc_report)
```

```
                   precision    recall  f1-score   support

                1       0.79      0.96      0.86       120
                2       0.67      0.90      0.77       120
                3       0.51      0.58      0.55       120
                4       0.58      0.52      0.55       120
                5       0.75      0.67      0.71       120
                6       0.69      0.70      0.70       120
                7       0.67      0.56      0.61       120
                8       0.60      0.78      0.68       120
                9       0.67      0.65      0.66       120
               10       0.62      0.84      0.72       120
               11       0.69      0.65      0.67       120
               12       0.72      0.75      0.73       120
               13       0.78      0.67      0.72       120
               14       0.61      0.70      0.65       120
               15       0.65      0.52      0.57       120
               16       0.60      0.69      0.64       120
               17       0.60      0.47      0.53       120
               18       0.61      0.59      0.60       120
               19       0.68      0.63      0.66       120
               20       0.56      0.57      0.57       120
               21       0.69      0.59      0.64       120
               22       0.88      0.72      0.79       120
               23       0.85      0.87      0.86       120
               24       0.71      0.83      0.77       120
               25       0.64      0.61      0.62       120
               26       0.76      0.70      0.73       120
               27       0.74      0.81      0.77       120
               28       0.90      0.53      0.66       120

         accuracy                           0.68      3360
        macro avg       0.69      0.68      0.68      3360
     weighted avg       0.69      0.68      0.68      3360
```

Implementation

```
img = random.randint(0, len(rfc_predections) - 1)
print("Predection =", rfc_predections[img])
plt.imshow(test_data.values[img].reshape(32,32).transpose(1,0))
```

```
Predection = 21
<matplotlib.image.AxesImage at 0x7e70e06a2380>
```



DL: Neural Network (NN)

Normalizing the data

```
train_data = tf.keras.utils.normalize(train_data, axis=1)
test_data = tf.keras.utils.normalize(test_data, axis=1)
```

Neural Network model

```
mode_nn = tf.keras.models.Sequential([
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(29, activation='softmax')
])
```

compiling the model

```
mode_nn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')
```

Training the model

```
mode_nn.fit(train_data, train_target, validation_split=0.3, epochs=10)
```

```
Epoch 1/10
294/294 [==============================] - 4s 5ms/step - loss: 2.2047 - accuracy: 0.3204 - val_loss: 1.6999 - val_accuracy: 0.4345
Epoch 2/10
294/294 [==============================] - 1s 4ms/step - loss: 1.2023 - accuracy: 0.5933 - val_loss: 1.2941 - val_accuracy: 0.5657
Epoch 3/10
294/294 [==============================] - 1s 3ms/step - loss: 0.7955 - accuracy: 0.7215 - val_loss: 1.1450 - val_accuracy: 0.6081
Epoch 4/10
294/294 [==============================] - 1s 4ms/step - loss: 0.5670 - accuracy: 0.7979 - val_loss: 1.1217 - val_accuracy: 0.6399
Epoch 5/10
294/294 [==============================] - 2s 5ms/step - loss: 0.4093 - accuracy: 0.8602 - val_loss: 1.0994 - val_accuracy: 0.6555
Epoch 6/10
294/294 [==============================] - 1s 4ms/step - loss: 0.3003 - accuracy: 0.9016 - val_loss: 1.1161 - val_accuracy: 0.6642
Epoch 7/10
294/294 [==============================] - 1s 3ms/step - loss: 0.2256 - accuracy: 0.9257 - val_loss: 1.1509 - val_accuracy: 0.6729
Epoch 8/10
294/294 [==============================] - 1s 4ms/step - loss: 0.1505 - accuracy: 0.9549 - val_loss: 1.1723 - val_accuracy: 0.6833
Epoch 9/10
294/294 [==============================] - 1s 4ms/step - loss: 0.1104 - accuracy: 0.9679 - val_loss: 1.2287 - val_accuracy: 0.6860
Epoch 10/10
294/294 [==============================] - 1s 3ms/step - loss: 0.0759 - accuracy: 0.9800 - val_loss: 1.3463 - val_accuracy: 0.6778
<keras.src.callbacks.History at 0x7e70cdb804f0>
```

Finding loss and accuracy

```
loss, accuracy = mode_nn.evaluate(test_data, test_target)

print("Accuracy =", accuracy)
print("Loss =", loss)
```

```
105/105 [==============================] - 0s 2ms/step - loss: 1.0054 - accuracy: 0.7366
Accuracy = 0.7366071343421936
Loss = 1.0053635835647583
```

Making predictions

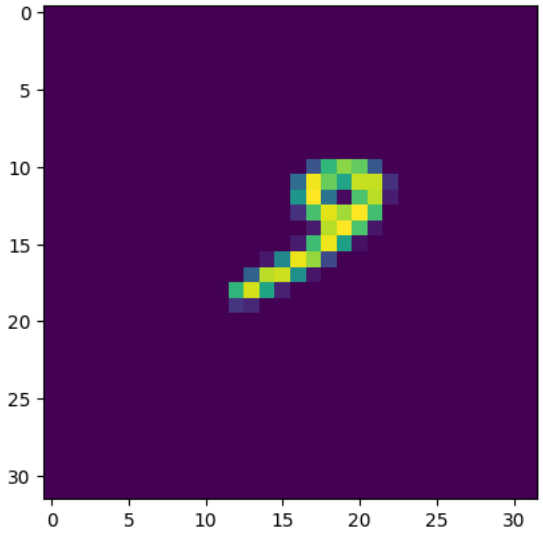
```
ann_predections = mode_nn.predict(test_data)
```

```
105/105 [==============================] - 0s 2ms/step
```

Implementation

```
img = random.randint(0, len(ann_predections) - 1)
print("Predection =", np.argmax(ann_predections[img]))
plt.imshow(test_data.values[img].reshape(32,32).transpose(1,0))
```

```
Predection = 10
<matplotlib.image.AxesImage at 0x7e70cd266b90>
```



DL: Convolutional Neural Networks (CNN)

Reshaping the data to a numpy array

```
train_data_cnn = np.reshape(train_data.values, (train_data.shape[0], 32, 32))
print(train_data_cnn.shape)

test_data_cnn = np.reshape(test_data.values, (test_data.shape[0], 32, 32))
print(test_data_cnn.shape)
```

```
(13440, 32, 32)
(3360, 32, 32)
```

Reshaping the data to a convolutional shape

```
train_data_cnn = train_data_cnn.reshape(train_data_cnn.shape[0], train_data_cnn.shape[1], train_data_cnn.shape[2], 1)
print(train_data_cnn.shape)

test_data_cnn = test_data_cnn.reshape(test_data_cnn.shape[0], test_data_cnn.shape[1], test_data_cnn.shape[2], 1)
print(test_data_cnn.shape)
```

```
(13440, 32, 32, 1)
(3360, 32, 32, 1)
```

Convolutional Neural Network model

```
model_cnn = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=2),

    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=2),

    tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'same'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=2),

    tf.keras.layers.Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding = 'same'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(1024,activation ="selu"),
    tf.keras.layers.Dense(512,activation ="selu"),
    tf.keras.layers.Dense(29,activation ="softmax")
])
```

compiling the model

```
model_cnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')
```

Training the model

```
model_cnn.fit(train_data_cnn, train_target, validation_split=0.3, epochs=10)
```

```
Epoch 1/10
294/294 [==============================] - 7s 8ms/step - loss: 1.7430 - accuracy: 0.4229 - val_loss: 0.9030 - val_accuracy: 0.6771
Epoch 2/10
294/294 [==============================] - 2s 5ms/step - loss: 0.6601 - accuracy: 0.7641 - val_loss: 0.5771 - val_accuracy: 0.7984
Epoch 3/10
294/294 [==============================] - 2s 5ms/step - loss: 0.4204 - accuracy: 0.8523 - val_loss: 0.4405 - val_accuracy: 0.8492
Epoch 4/10
294/294 [==============================] - 2s 6ms/step - loss: 0.2880 - accuracy: 0.9019 - val_loss: 0.4955 - val_accuracy: 0.8383
Epoch 5/10
294/294 [==============================] - 2s 6ms/step - loss: 0.2214 - accuracy: 0.9253 - val_loss: 0.3812 - val_accuracy: 0.8832
Epoch 6/10
294/294 [==============================] - 2s 6ms/step - loss: 0.1656 - accuracy: 0.9418 - val_loss: 0.4848 - val_accuracy: 0.8671
Epoch 7/10
294/294 [==============================] - 2s 8ms/step - loss: 0.1487 - accuracy: 0.9512 - val_loss: 0.5056 - val_accuracy: 0.8666
Epoch 8/10
294/294 [==============================] - 2s 6ms/step - loss: 0.1370 - accuracy: 0.9534 - val_loss: 0.5613 - val_accuracy: 0.8693
Epoch 9/10
294/294 [==============================] - 2s 6ms/step - loss: 0.1039 - accuracy: 0.9657 - val_loss: 0.4471 - val_accuracy: 0.8874
Epoch 10/10
294/294 [==============================] - 2s 5ms/step - loss: 0.1207 - accuracy: 0.9584 - val_loss: 0.4669 - val_accuracy: 0.8894
<keras.src.callbacks.History at 0x7e70cd2ed930>
```

Finding loss and accuracy

```
loss, accuracy = model_cnn.evaluate(test_data_cnn, test_target)

print("Accuracy =", accuracy)
print("Loss =", loss)
```

```
105/105 [==============================] - 0s 3ms/step - loss: 0.4112 - accuracy: 0.9062
Accuracy = 0.90625
Loss = 0.41121718287467957
```

Making predections

```
cnn_predections = model_cnn.predict(test_data_cnn)
```

```
105/105 [==============================] - 0s 2ms/step
```

Implementation

```
img = random.randint(0, len(cnn_predections) - 1)
print("Predection =", np.argmax(cnn_predections[img]))
plt.imshow(test_data.values[img].reshape(32,32).transpose(1,0))
```

```
Predection = 9
<matplotlib.image.AxesImage at 0x7e70cc79c460>
```