```
#importing libraries
import numpy as np
import pandas as pd
import random as rd

#data visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
from PIL import Image

#for the CNN model
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from keras.preprocessing.image import ImageDataGenerator

#setting seed for reproducability
from numpy.random import seed
seed(10)
tf.random.set_seed(20)

#for viewing filenames
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
#downloading the training data
train = pd.read_csv("/content/sign_mnist_train.csv")
train.head()
```

```
#downloading the test data
test = pd.read_csv("/content/sign_mnist_test.csv")
test.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...
0	6	149	149	150	150	150	151	151	150	151	...
1	5	126	128	131	132	133	134	135	135	136	...
2	10	85	88	92	96	105	123	135	143	147	...
3	0	203	205	207	206	207	209	210	209	210	...
4	3	188	191	193	195	199	201	202	203	203	...

5 rows × 785 columns

```
#summing the number of na in the training set for each column
print(sum(train.isna().sum()))

#summing the number of na in the test set for each column
print(sum(test.isna().sum()))
```

```
731
102
```

```
#summing the number of null values in the training set for each column
print(sum(train.isnull().sum()))

#summing the number of null values in the test set for each column
print(sum(test.isnull().sum()))
```

```
731
102
```

```
#creating our Y for the training data
Y_train = train["label"]

#creating our X for the training data
X_train = train.drop(labels = ["label"],axis = 1)


#creating our Y for the test data
Y_test = test["label"]

#creating our X for the training data
X_test = test.drop(labels = ["label"],axis = 1)
```

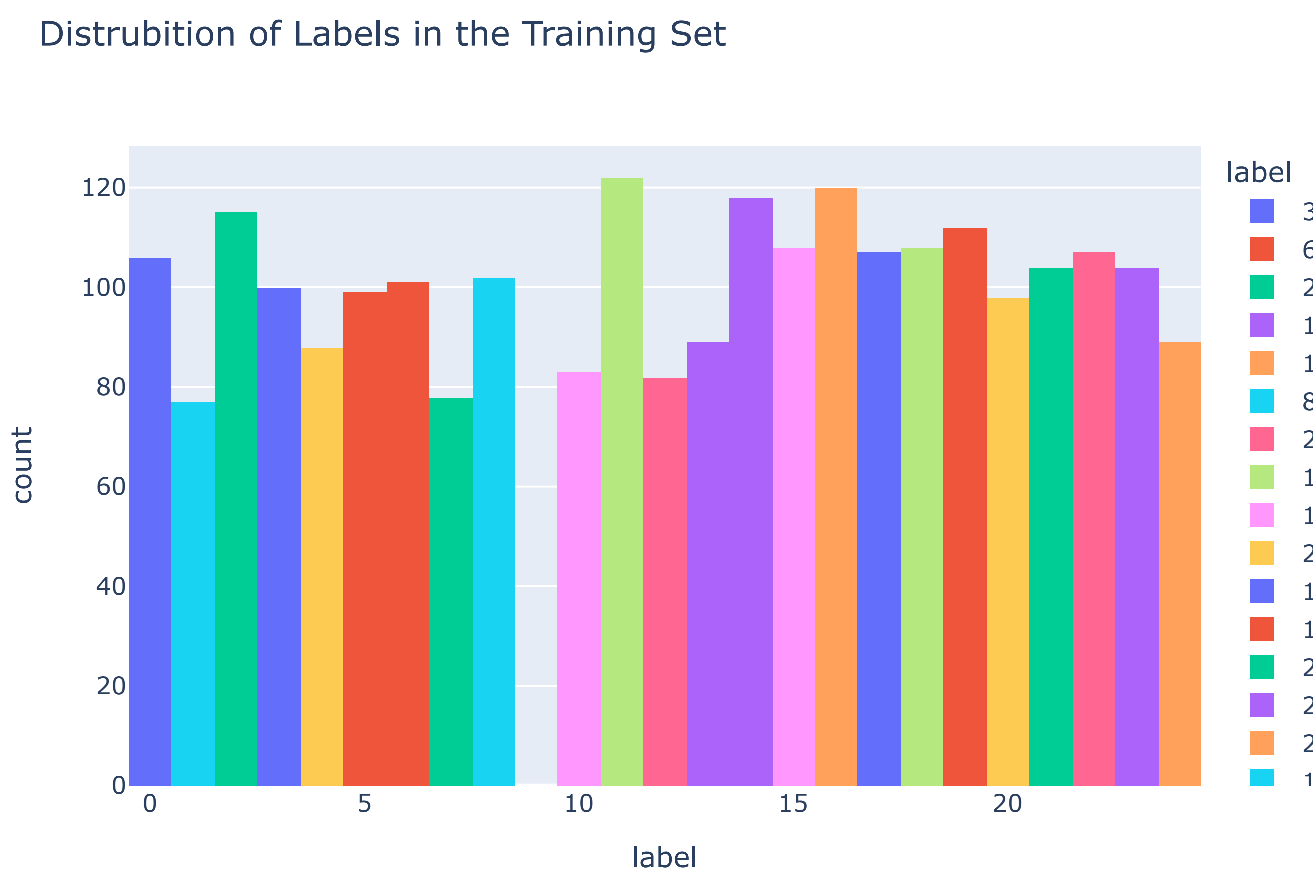
```
#converting the range of the pixel data from 0-255 to 0-1
X_train = X_train / 255.0

X_test = X_test / 255.0
```

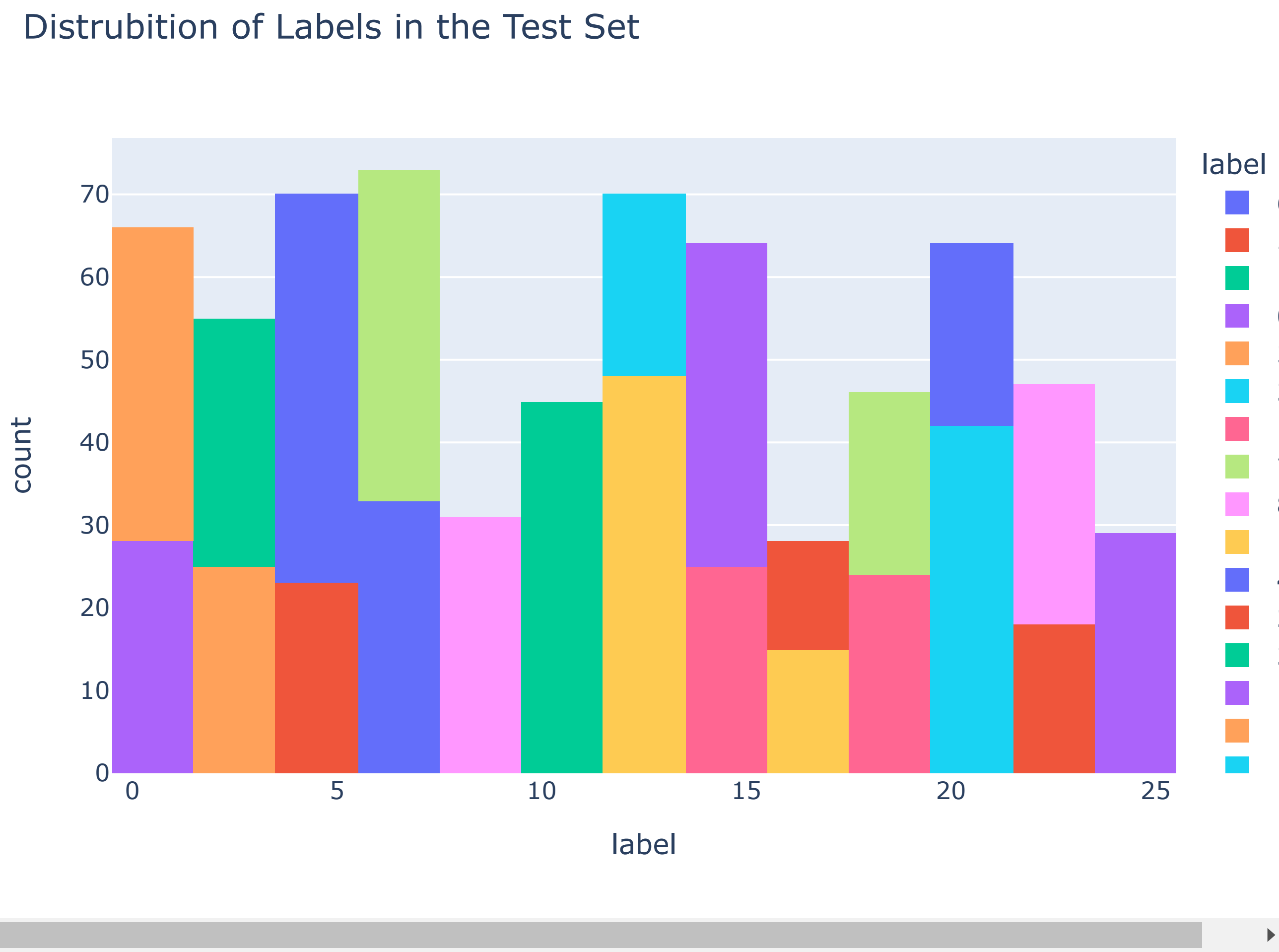
```
X_train = X_train.values.reshape(-1,28,28,1)
X_test = X_test.values.reshape(-1,28,28,1)
print(X_train.shape)
print(X_test.shape)
```

```
(2417, 28, 28, 1)
(688, 28, 28, 1)
```

```
#creating an interactive bar graph that shows the distrubition of labels within the training
fig = px.histogram(train,
                   x='label',
                   color = 'label',
                   title="Distrubition of Labels in the Training Set",
                   width=700, height=500)
fig.show()
```



```
#creating an interactive bar graph that shows the distrubition of labels within the test set
fig = px.histogram(test,
                   x='label',
                   color = 'label',
                   title="Distrubition of Labels in the Test Set",
                   width=700, height=500)
fig.show()
```




```
#creating a 5x5 grid of the first 25 photos in the training images
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(15, 10),
                        subplot_kw={'xticks': [], 'yticks': []})

for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(X_train[i], cmap='gray')
    plt.title(Y_train[i])
plt.show()
```

```
#creating a 5x5 grid of the first 25 photos in the test images
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(15, 10),
                        subplot_kw={'xticks': [], 'yticks': []})

for i in range(25):
```

```
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(X_test[i], cmap='gray')
    plt.title(Y_test[i])
plt.show()
```

```
#spliting training images into the images we will use for training the model and validating t
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.3, random_s
```

```
#showing the shapes of our train, validate, and test images
print(X_train.shape)
print(Y_train.shape)
print(X_val.shape)
print(Y_val.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(1691, 28, 28, 1)
(1691,)
(726, 28, 28, 1)
(726,)
(688, 28, 28, 1)
(688,)
```

```
#creating our CNN model
model = keras.Sequential([

    layers.BatchNormalization(),
    layers.Conv2D(filters=32, kernel_size=(5,5), activation="relu", padding='same',
                  input_shape=[28, 28, 1]),
    layers.MaxPool2D(),
    layers.Dropout(.25),

    layers.BatchNormalization(),
    layers.Conv2D(filters=32, kernel_size=(3,3), activation="relu", padding='same'),
    layers.MaxPool2D(),
    layers.Dropout(.25),

    layers.BatchNormalization(),
    layers.Conv2D(filters=64, kernel_size=(3,3), activation="relu", padding='same'),
    layers.MaxPool2D(),
    layers.Dropout(.25),
    layers.BatchNormalization(),
    layers.Conv2D(filters=128, kernel_size=(3,3), activation="relu", padding='same'),
    layers.MaxPool2D(),
    layers.Dropout(.25),

    layers.Flatten(),
    layers.Dropout(.25),
    layers.Dense(units=64, activation="relu"),
```

```
    layers.Dense(units=26, activation="softmax"),
])

#compiling the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(epsilon=0.01),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)


#Training the model
history = model.fit(
    x = X_train,
    y = Y_train,
    validation_data= (X_val,Y_val),
    batch_size = 128,
    epochs=50,
    verbose=2,
)
```

```
Epoch 1/50
14/14 - 6s - loss: nan - accuracy: 0.0438 - val_loss: nan - val_accuracy: 0.0455 - 6s
Epoch 2/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 3/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 4/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 5/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 6/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 7/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 8/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 9/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 10/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 11/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 12/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 13/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 14/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 15/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 16/50
14/14 - 4s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 4s
Epoch 17/50
```

```
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 18/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 19/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 20/50
14/14 - 4s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 4s
Epoch 21/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 22/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 23/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 24/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 25/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 26/50
14/14 - 4s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 4s
Epoch 27/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
Epoch 28/50
14/14 - 3s - loss: nan - accuracy: 0.0432 - val_loss: nan - val_accuracy: 0.0455 - 3s
```

```
#Viewing the training results
history_frame = pd.DataFrame(history.history)
history_frame.loc[:, ['loss', 'val_loss']].plot()
history_frame.loc[:, ['accuracy', 'val_accuracy']].plot();
```


```
#creating our predictions using the test pixel values
predictions = model.predict(X_test)
predictions = np.argmax(predictions,axis = 1)

#creating a report that show how our predictions compare with actual values
print(classification_report(Y_test, predictions))
```

```
22/22 [==============================] - 0s 15ms/step
              precision    recall  f1-score   support

           0       0.04      1.00      0.08        28
           1       0.00      0.00      0.00        38
           2       0.00      0.00      0.00        30
           3       0.00      0.00      0.00        25
           4       0.00      0.00      0.00        47
           5       0.00      0.00      0.00        23
           6       0.00      0.00      0.00        33
           7       0.00      0.00      0.00        40
           8       0.00      0.00      0.00        31
          10       0.00      0.00      0.00        28
          11       0.00      0.00      0.00        17
          12       0.00      0.00      0.00        48
          13       0.00      0.00      0.00        22
          14       0.00      0.00      0.00        25
          15       0.00      0.00      0.00        39
          16       0.00      0.00      0.00        13
          17       0.00      0.00      0.00        15
          18       0.00      0.00      0.00        22
          19       0.00      0.00      0.00        24
          20       0.00      0.00      0.00        22
          21       0.00      0.00      0.00        42
          22       0.00      0.00      0.00        18
          23       0.00      0.00      0.00        29
          24       0.00      0.00      0.00        29

    accuracy                           0.04       688
   macro avg       0.00      0.04      0.00       688
weighted avg       0.00      0.04      0.00       688

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
```

```
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted s
```