ANSHUMAN PRATIK

'Artificial Intelligence Sign Language Recognition'

Edunet Foundation & IBM SkillsBuild Internship

```
#Importing essential libraries for Data visualization
import pandas as pd
import numpy as np
import keras
import cv2
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D, Dense,Flatten
from keras.datasets import mnist
import random as rd
import os
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
from PIL import Image
from keras.utils import np_utils
from keras.optimizers import SGD
```

```
#Libraries for Image processing
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from keras.preprocessing.image import ImageDataGenerator
```

```
#Initializing reproducibility of seed by TensorFlow
from numpy.random import seed
seed(10)
tf.random.set_seed(20)
```

```
#Joining path of Kaggle File to Directory
for dirname, _, filenames in os.walk('/kaggle/input'):
  for filename in filenames:
    print(os.path.join(dirname, filename))
```

```
#Accessing Training and Testing data from Google Drive
train_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/sign_mnist_train.csv')
test_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/sign_mnist_test.csv')
```

```
train_data.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782
0	3	107	118	127	134	139	143	146	150	153	...	207	207	207	207	206	206	206	204
1	6	155	157	156	156	156	157	156	158	158	...	69	149	128	87	94	163	175	103
2	2	187	188	188	187	187	186	187	188	187	...	202	201	200	199	198	199	198	195
3	2	211	211	212	212	211	210	211	210	210	...	235	234	233	231	230	226	225	222
4	13	164	167	170	172	176	179	180	184	185	...	92	105	105	108	133	163	157	163

5 rows × 785 columns

```
test_data.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782
0	6	149	149	150	150	150	151	151	150	151	...	138	148	127	89	82	96	106	112
1	5	126	128	131	132	133	134	135	135	136	...	47	104	194	183	186	184	184	184
2	10	85	88	92	96	105	123	135	143	147	...	68	166	242	227	230	227	226	225
3	0	203	205	207	206	207	209	210	209	210	...	154	248	247	248	253	236	230	240
4	3	188	191	193	195	199	201	202	203	203	...	26	40	64	48	29	46	49	46

5 rows × 785 columns

```
#Computing Null Value in each column Data set after Preprocessing
print("Null values in Each column in Training data set = ",sum(train_data.isnull().sum()))
print("Null values in Each column in Testing data set = ", sum(test_data.isnull().sum()))
```

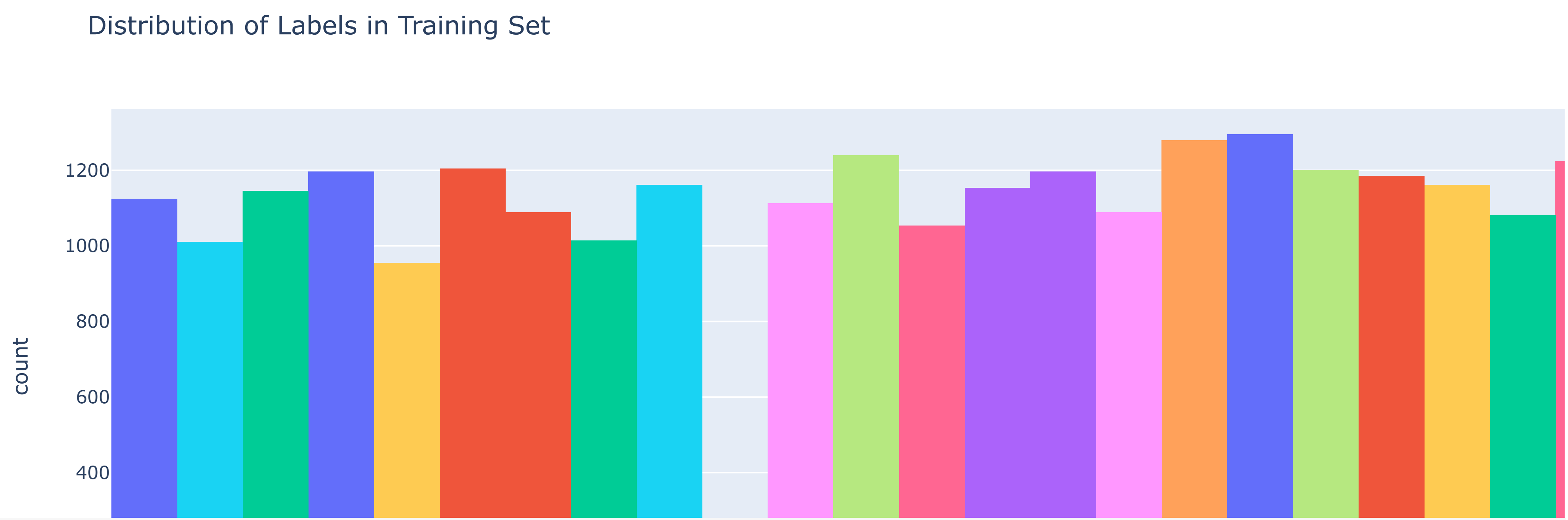
```
Null values in Each column in Training data set =  0
Null values in Each column in Testing data set =  0
```

```
#Creating data labels for training and testing sets
y_train = train_data["label"]
x_train = train_data.drop(labels = ["label"],axis=1)
y_test = test_data["label"]
x_test = test_data.drop(labels = ["label"],axis=1)
```

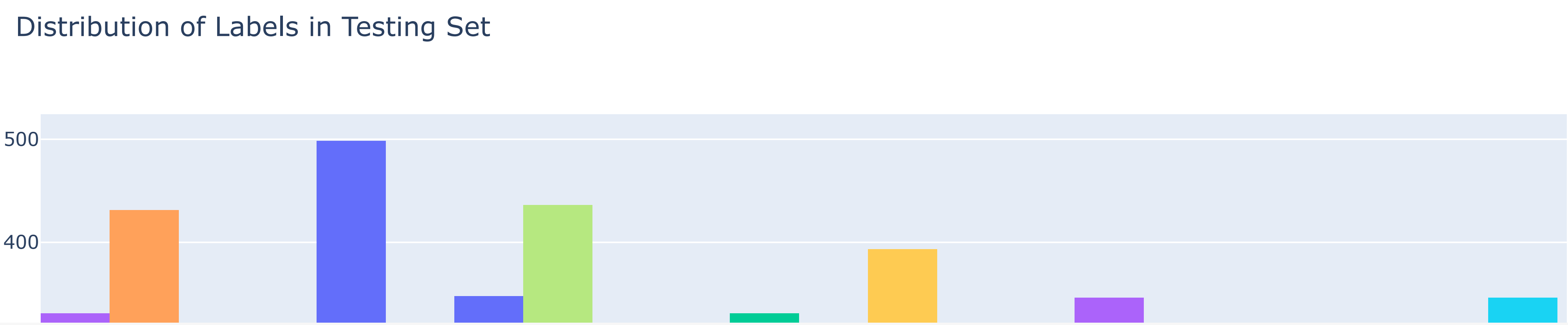
```
#Converting pixel range to 0-1 from initial 0-255
x_train = x_train /255.0
x_test = x_test /255.0
#Making number of elements in set same when reshaped
x_train = x_train.values.reshape(-1,28,28,1)
x_test = x_test.values.reshape(-1,28,28,1)
print(x_train.shape)
print(x_test.shape)
```

```
(27455, 28, 28, 1)
(7172, 28, 28, 1)
```

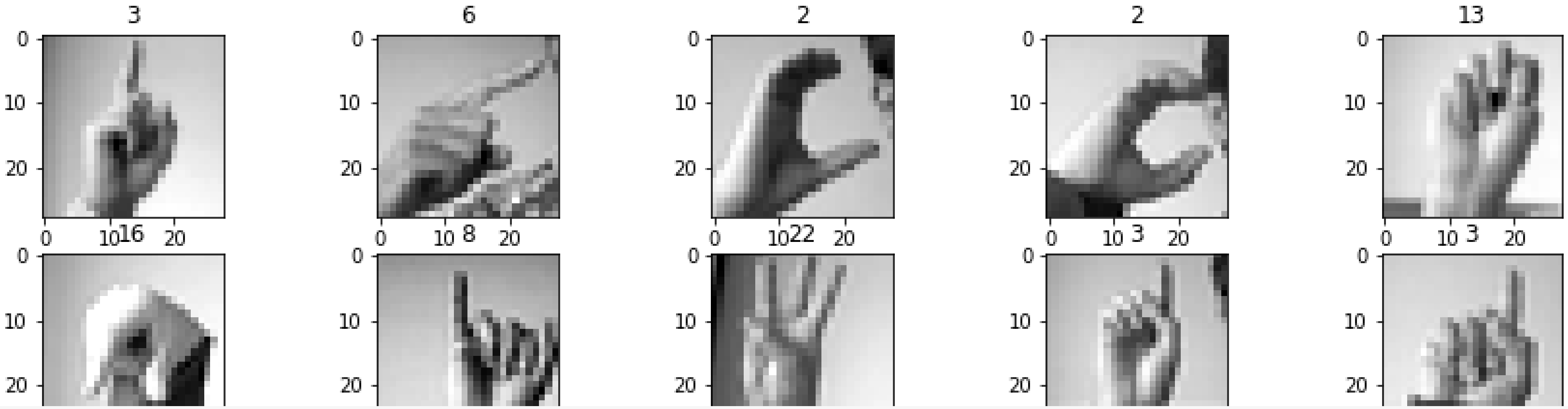
```
#Interactive Bar graph showing distribution of Labels count in training set
bar_graph= px.histogram(train_data, x='label', color='label', title='Distribution of Labels in Training Set')
bar_graph.show()
```



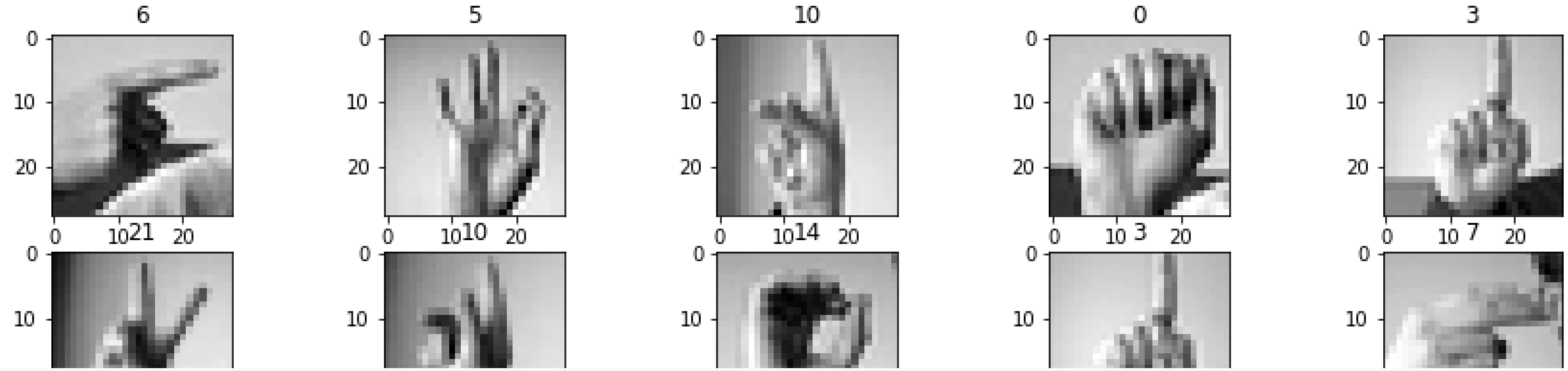

```
#Interactive Bar graph showing distribution of Labels count in testing set
bar_graph_test= px.histogram(test_data, x='label', color='label', title='Distribution of Labels in Testing Set')
bar_graph_test.show()
```




```
#Creating 5 x 5 Grid of  25 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(np.squeeze(x_train[i]), cmap ='gray')
  plt.title(y_train[i])
plt.show()
```




```
#Testing new images on Model
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(np.squeeze(x_test[i]), cmap ='gray')
  plt.title(y_test[i])
plt.show()
```




```
#Splitting of training images for modeling and validation of sign images.
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.3, random_state=104, shuffle=True)
print(x_train.shape)
print(y_train.shape)

print(x_val.shape)
print(y_val.shape)

print(x_test.shape)
print(y_test.shape)
```

```
(9416, 28, 28, 1)
(9416,)
(4036, 28, 28, 1)
(4036,)
(7172, 28, 28, 1)
(7172,)
```

```
#Modellig Rectified Linear Unit Convolution Neural Networks
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(0.25),
            layers.Dense(units=64, activation="relu"),
            layers.Dense(units=26, activation="softmax"),
             ])
```

```
#Compilation of CNN Model
model.compile(
   optimizer=tf.keras.optimizers.Adam(epsilon=0.01),
   loss= 'sparse_categorical_crossentropy',
   metrics =['accuracy']
)
```

```
#Tarining model against Epoch and Batch Size
history = model.fit(
   x=x_train,
   y=y_train,
   validation_data=(x_val,y_val),
   batch_size=128,
   epochs=50,
   verbose=2
)
```

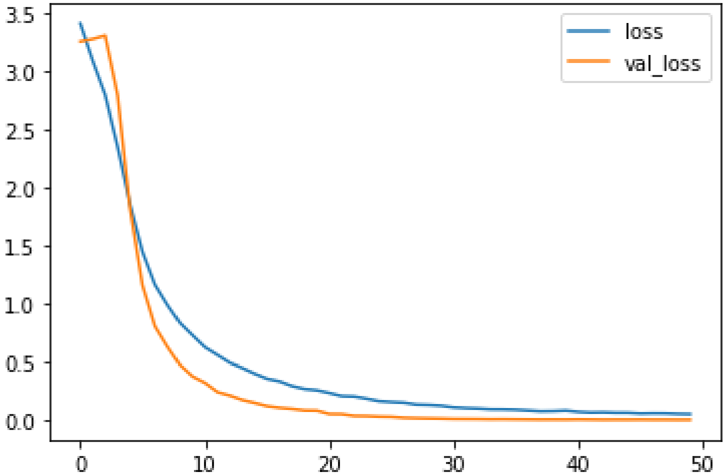
```
Epoch 1/50
74/74 - 19s - loss: 3.4119 - accuracy: 0.0559 - val_loss: 3.2566 - val_accuracy: 0.0538 - 19s/epoch - 251ms/step
Epoch 2/50
74/74 - 16s - loss: 3.0894 - accuracy: 0.1008 - val_loss: 3.2778 - val_accuracy: 0.0483 - 16s/epoch - 222ms/step
Epoch 3/50
```

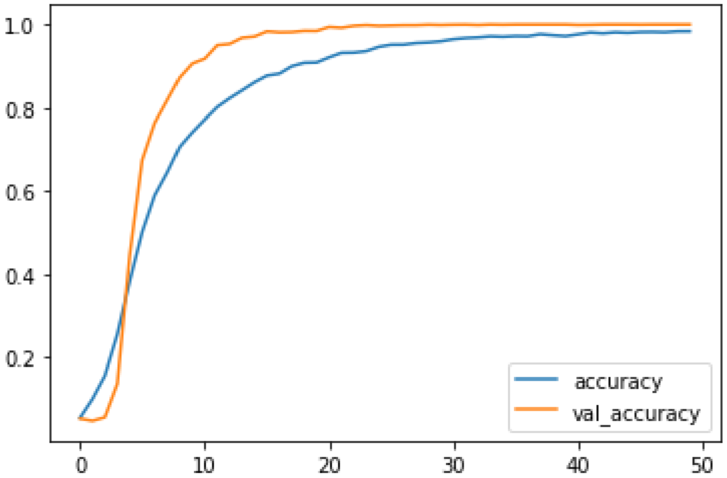
```
74/74 - 20s - loss: 2.7959 - accuracy: 0.1568 - val_loss: 3.3083 - val_accuracy: 0.0570 - 20s/epoch - 268ms/step
Epoch 4/50
74/74 - 16s - loss: 2.3520 - accuracy: 0.2563 - val_loss: 2.7928 - val_accuracy: 0.1378 - 16s/epoch - 220ms/step
Epoch 5/50
74/74 - 17s - loss: 1.8581 - accuracy: 0.3835 - val_loss: 1.8189 - val_accuracy: 0.4467 - 17s/epoch - 224ms/step
Epoch 6/50
74/74 - 16s - loss: 1.4473 - accuracy: 0.5016 - val_loss: 1.1588 - val_accuracy: 0.6747 - 16s/epoch - 222ms/step
Epoch 7/50
74/74 - 22s - loss: 1.1632 - accuracy: 0.5897 - val_loss: 0.8056 - val_accuracy: 0.7634 - 22s/epoch - 298ms/step
Epoch 8/50
74/74 - 16s - loss: 0.9882 - accuracy: 0.6445 - val_loss: 0.6297 - val_accuracy: 0.8181 - 16s/epoch - 222ms/step
Epoch 9/50
74/74 - 16s - loss: 0.8321 - accuracy: 0.7067 - val_loss: 0.4711 - val_accuracy: 0.8729 - 16s/epoch - 222ms/step
Epoch 10/50
74/74 - 19s - loss: 0.7272 - accuracy: 0.7410 - val_loss: 0.3730 - val_accuracy: 0.9061 - 19s/epoch - 252ms/step
Epoch 11/50
74/74 - 16s - loss: 0.6247 - accuracy: 0.7719 - val_loss: 0.3180 - val_accuracy: 0.9177 - 16s/epoch - 220ms/step
Epoch 12/50
74/74 - 16s - loss: 0.5584 - accuracy: 0.8019 - val_loss: 0.2391 - val_accuracy: 0.9502 - 16s/epoch - 222ms/step
Epoch 13/50
74/74 - 16s - loss: 0.4913 - accuracy: 0.8230 - val_loss: 0.2095 - val_accuracy: 0.9532 - 16s/epoch - 221ms/step
Epoch 14/50
74/74 - 18s - loss: 0.4457 - accuracy: 0.8419 - val_loss: 0.1719 - val_accuracy: 0.9680 - 18s/epoch - 249ms/step
Epoch 15/50
74/74 - 17s - loss: 0.3968 - accuracy: 0.8611 - val_loss: 0.1463 - val_accuracy: 0.9710 - 17s/epoch - 224ms/step
Epoch 16/50
74/74 - 16s - loss: 0.3534 - accuracy: 0.8770 - val_loss: 0.1188 - val_accuracy: 0.9834 - 16s/epoch - 221ms/step
Epoch 17/50
74/74 - 18s - loss: 0.3312 - accuracy: 0.8817 - val_loss: 0.1045 - val_accuracy: 0.9812 - 18s/epoch - 240ms/step
Epoch 18/50
74/74 - 17s - loss: 0.2911 - accuracy: 0.8995 - val_loss: 0.0951 - val_accuracy: 0.9817 - 17s/epoch - 230ms/step
Epoch 19/50
74/74 - 16s - loss: 0.2657 - accuracy: 0.9078 - val_loss: 0.0822 - val_accuracy: 0.9846 - 16s/epoch - 221ms/step
Epoch 20/50
74/74 - 16s - loss: 0.2549 - accuracy: 0.9086 - val_loss: 0.0806 - val_accuracy: 0.9844 - 16s/epoch - 221ms/step
Epoch 21/50
74/74 - 21s - loss: 0.2317 - accuracy: 0.9209 - val_loss: 0.0514 - val_accuracy: 0.9941 - 21s/epoch - 278ms/step
Epoch 22/50
74/74 - 16s - loss: 0.2055 - accuracy: 0.9317 - val_loss: 0.0502 - val_accuracy: 0.9921 - 16s/epoch - 219ms/step
Epoch 23/50
74/74 - 16s - loss: 0.2019 - accuracy: 0.9322 - val_loss: 0.0348 - val_accuracy: 0.9968 - 16s/epoch - 221ms/step
Epoch 24/50
74/74 - 16s - loss: 0.1832 - accuracy: 0.9359 - val_loss: 0.0335 - val_accuracy: 0.9985 - 16s/epoch - 222ms/step
Epoch 25/50
74/74 - 18s - loss: 0.1617 - accuracy: 0.9462 - val_loss: 0.0286 - val_accuracy: 0.9968 - 18s/epoch - 249ms/step
```

```
Epoch 26/50
74/74 - 17s - loss: 0.1543 - accuracy: 0.9514 - val_loss: 0.0266 - val_accuracy: 0.9975 - 17s/epoch - 231ms/step
Epoch 27/50
74/74 - 16s - loss: 0.1488 - accuracy: 0.9515 - val_loss: 0.0182 - val_accuracy: 0.9983 - 16s/epoch - 221ms/step
Epoch 28/50
74/74 - 18s - loss: 0.1324 - accuracy: 0.9553 - val_loss: 0.0154 - val_accuracy: 0.9983 - 18s/epoch - 248ms/step
Epoch 29/50
```

```
#Obtaining Training Data Frame
history_frame = pd.DataFrame(history.history)
history_frame.loc[:, ['loss', 'val_loss']].plot()
history_frame.loc[:, ['accuracy','val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f727a637210>
```





```
#Estimation of prediction using Test pixels
pred = model.predict(x_test)
prediction = np.argmax(pred,axis=1)
```

```
225/225 [==============================] - 4s 17ms/step
```

```
#Prediction final report describing Precision, Recall and F1-Score of 25 Images
print(classification_report(y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       331
           1       1.00      1.00      1.00       432
           2       1.00      1.00      1.00       310
           3       1.00      1.00      1.00       245
           4       1.00      1.00      1.00       498
           5       1.00      1.00      1.00       247
           6       0.93      1.00      0.96       348
           7       1.00      0.95      0.98       436
           8       1.00      1.00      1.00       288
          10       1.00      1.00      1.00       331
          11       1.00      1.00      1.00       209
          12       1.00      1.00      1.00       394
          13       1.00      1.00      1.00       291
          14       1.00      1.00      1.00       246
          15       1.00      1.00      1.00       347
          16       1.00      1.00      1.00       164
          17       0.97      1.00      0.98       144
          18       1.00      1.00      1.00       246
          19       1.00      0.97      0.98       248
          20       1.00      0.98      0.99       266
          21       1.00      1.00      1.00       346
          22       1.00      1.00      1.00       206
          23       1.00      1.00      1.00       267
          24       1.00      1.00      1.00       332

    accuracy                           1.00      7172
   macro avg       1.00      1.00      1.00      7172
weighted avg       1.00      1.00      1.00      7172
```

Reference Link:

https://colab.research.google.com/drive/1cOArLgppKDnF_z8oeAQQidlVB3Nf1JAN#scrollTo=T9E_b1srgnkZ