Predictive Analytics

Module 6

Letter Dataset Text Analysis

Submitted to Dr. Mary Donhoffner

By Ananya Sharma

Introduction

To identify which pupils may need to improve their motor abilities at an early age, a school is attempting to determine whether they can utilize a writing exam. We are trying to understand what the images are from that writing test and derive model accuracy for the correct prediction. The data has been mentioned in the form of pixels and each pixel basically represents an image.

The figure below just gives us a glimpse of how the symbols, or the pictures could look before they have been put in the dataset in the form of numbers.

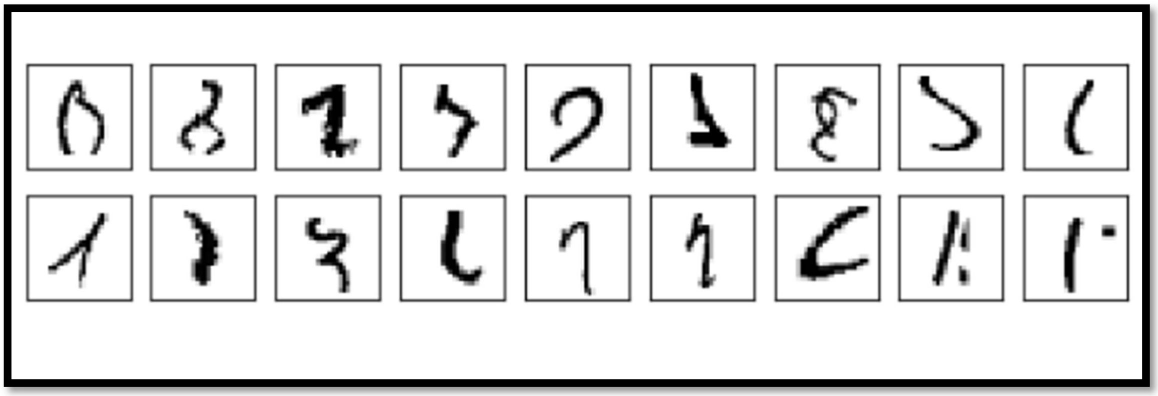

Figure 1: Example of an Image that can be used for Image to Text Analysis, source (Nielsen, 2015b)

Our dataset has 42000 rows, and 46 columns present. The names of the columns are as below.

```
#Columns
letter_analysis.columns
```

```
Index(['label', 'pixel43', 'pixel44', 'pixel92', 'pixel124', 'pixel125',
       'pixel126', 'pixel127', 'pixel128', 'pixel129', 'pixel130', 'pixel131',
       'pixel132', 'pixel133', 'pixel134', 'pixel135', 'pixel136', 'pixel137',
       'pixel138', 'pixel146', 'pixel147', 'pixel148', 'pixel149', 'pixel150',
       'pixel151', 'pixel152', 'pixel153', 'pixel154', 'pixel155', 'pixel156',
       'pixel157', 'pixel158', 'pixel159', 'pixel160', 'pixel327', 'pixel328',
       'pixel329', 'pixel351', 'pixel410', 'pixel411', 'pixel412', 'pixel413',
       'pixel414', 'pixel415', 'pixel416', 'pixel417'],
      dtype='object')
```

Figure 2: Column Names of the Letter Dataset

It is not a very conventional dataset, and all the columns are in number form. We don’t have any categorical values.

The dataset does not have any null values or duplicated values. We can see from figure3 and figure 4 below that the dataset seems to be quite clean.

```
#Checking for the null values
letter_analysis.isnull().sum()

#there are not many nukll values
```

```
label       0
pixel43     0
pixel44     0
pixel92     0
pixel124    0
pixel125    0
pixel126    0
pixel127    0
pixel128    0
pixel129    0
pixel130    0
pixel131    0
pixel132    0
pixel133    0
pixel134    0
pixel135    0
```

Figure 3: Null Values

```
#The dataframe does not have any duplicates
letter_analysis.duplicated()
```

```
0        False
1        False
2        False
3        False
4        False
         ...
41995    False
41996    False
41997    False
41998    False
41999    False
Length: 42000, dtype: bool
```

Figure 4: Duplicated Values

Analysis

The dataset has majorly integer types of data type and occupies a space of 14.7 MB

```
letter_analysis.info()
# All the columns are of type Integer and the dataset occupie
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Data columns (total 46 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   label     42000 non-null  int64
 1   pixel43   42000 non-null  int64
 2   pixel44   42000 non-null  int64
 3   pixel92   42000 non-null  int64
 4   pixel124  42000 non-null  int64
 5   pixel125  42000 non-null  int64
 6   pixel126  42000 non-null  int64
 7   pixel127  42000 non-null  int64
 8   pixel128  42000 non-null  int64
 9   pixel129  42000 non-null  int64
 10  pixel130  42000 non-null  int64
 11  pixel131  42000 non-null  int64
 12  pixel132  42000 non-null  int64
 13  pixel133  42000 non-null  int64
 14  pixel134  42000 non-null  int64
 15  pixel135  42000 non-null  int64
 16  pixel136  42000 non-null  int64
```

Figure: Information of the dataset

```
# Summary of the Dataset
letter_analysis.describe()
```

	label	pixel43	pixel44	pixel92	pixel124	pixel125	pixel126	pixel127	pixel128	pixel129	...
count	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	...
mean	4.456643	0.171357	0.164476	1.192833	28.043952	36.084976	42.713952	46.092310	44.542452	38.948524	...
std	2.887730	5.726352	5.515774	14.692403	70.505431	78.631145	84.390533	87.287033	85.740313	81.223946	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	10.000000	29.000000	21.000000	0.000000	...
max	9.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	...

8 rows × 46 columns

Figure: Summary statistics

From the above figure we derive that the summary statistics do not give us a lot of information about the dataset

<u>Importing Other Libraries:</u>

```
from sklearn import datasets, neighbors
from sklearn.model_selection import train_test_split
from keras.datasets import mnist
from random import randint
import time
from sklearn import datasets, neighbors
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from keras import models
from keras import layers
from keras.utils import to_categorical
```

```
np.random.seed(1212)
import keras
from keras.models import Model
from keras.layers import *
from keras import optimizers
```

```
import tensorflow as tf
from tensorflow.keras import layers
```

Figure: Importing libraries

We are importing certain libraries for the testing and training of the dataset. Furthermore, we import Keras and TensorFlow to fit the best model for our dataset.

TensorFlow is an open-source collection of frameworks for building and utilizing neural networks, such as those employed in projects involving machine learning (ML) and deep learning.

In contrast, Keras is a high-level API that utilizes TensorFlow. With its simple-to-use framework, Keras makes it easier to create sophisticated neural networks. (Active State, 2022)

```
#We are doing the value count of the label
letter_analysis['label'].value_counts()

1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

Figure: Value Count for Target Variable Label

<u>Test and Train the Model:</u>

We prepare the dataset for training and testing the model. We provide 80% of the dataset for training and 20 % for testing.

Variable X gets the data set without the label whereas Y gets the data set after adding the label column.

```
X = letter_analysis.drop(columns = ['label'], axis = 1)
y = letter_analysis['label']

X_train,X_test,y_train,y_test = train_test_split(X,y,train_size = 0.8, random_state = 12)
```

Figure: Split the dataset

After splitting we get the following data frame. We get a length of 33600 for the purpose of fitting the dataset. Furthermore, we try to understand how the data has been split up for testing and training.

```
X_train.shape,X_test.shape

((33600, 45), (8400, 45))
```

Figure: X_train and X_test

```
y_train.shape,y_test.shape

((33600,), (8400,))
```

Figure: y_train and y_test

From the figures above we understand that the columns have not been added for the y_train and y_test.

Exploring the Models

We explore several models to understand which can give us the result accurately.

1) Random Forest Classifier
2) KNN Model
3) XGBoost
4) Support Vector Machine
5) Neural Networks using TensorFlow and Keras library

Random Forest Classifier:

We import the random forest classifier library to run this model. We select the entropy criteria for running our nodes. Node estimators are taken to be around 245. We have considered the random_state as 0 and the minimum sample split to be 2

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=245,criterion='entropy',random_state=0,min_samples_split=2)

rfc.fit(X_train,y_train)

RandomForestClassifier(criterion='entropy', n_estimators=245, random_state=0)
```

Figure: Random Forest Classification

```
y_pred=rfc.predict(X_test)

from sklearn.metrics import accuracy_score
ac= accuracy_score(y_test,y_pred)
print('Accuracy is :',ac*100)

Accuracy is : 69.69047619047619
```

Figure: Accuracy Score of Random Forest classifier

KNN Model

The accuracy of the K Nearest Neighbour Model comes out to be 60%. The algorithm uses the Brute force method instead of the Euclidean Method to calculate the distance between the data points.

We have used the n_neighbours as 5 because it gives better accuracy than when we take the n_neighbours as 1. The leaf size has been taken to be 100. This is the min size that can fit in a class with the various parameters or features and must be of optimum size.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(algorithm='brute',n_neighbors =1 ,leaf_size=100,p=30)
knn.fit(X_train, y_train)

knn_predictions = knn.predict(X_test)

acc=accuracy_score(y_test,knn_predictions)
print('Accuracy is :',acc*100)

Accuracy is : 60.023809523809526
```

Figure: Knn with n_neighbour as 1

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(algorithm='brute',n_neighbors =5 ,leaf_size=100,p=30)
knn.fit(X_train, y_train)

knn_predictions = knn.predict(X_test)

acc=accuracy_score(y_test,knn_predictions)
print('Accuracy is :',acc*100)

Accuracy is : 63.90476190476191
```

Figure: KNN Model with n_neighbour as 5

<u>XGBoost:</u>

```
from xgboost import XGBClassifier
model = XGBClassifier(learning_rate=1.0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

Accuracy: 70.55%
```

Figure: XG Boost Model

A shrinkage of 0.1 is given to avoid the model from learning fast and becoming prone to overfitting.
By examining the distribution of features over all data points in a leaf, XGBoost can address the potential loss for all the possible splits by narrowing the search space for potential feature splits. (Tseng, 2018)

<u>Support Vector Machine</u>
RBF kernel projects high-dimensional data before looking for a linear separation for it. (Awasthi, 2020)
Support vector machine can be sensitive to outliers and requires a very high time to train the model hence we explore other options to train and recognize the images.
The accuracy given by this model is 70.17%

```
from sklearn.svm import SVC
model = SVC(kernel='rbf', C=1E01,tol=0.1)
model.fit(X_train, y_train)
predicted= model.predict(X_test)

from sklearn.metrics import confusion_matrix,accuracy_score
accuracy = accuracy_score(y_test, predicted)
print("Accuracy: %.2f%%" % (accuracy * 100.0))#97.28

Accuracy: 70.17%
```

Figure: Support Vector Machine

<u>Feature Normalization using Keras and tensor flow before fitting Neural Network:</u>

```
# Feature Normalization
X_train /= 255; X_test /= 255

# Convert labels to One Hot Encoded
num_digits = 10

# Converting labels into one hot encoded values so that our model
y_train = keras.utils.to_categorical(y_train, num_digits)
y_test = keras.utils.to_categorical(y_test, num_digits)
```

Figure: Feature Normalization

In the above diagram, we define a variable that equals our features. The Labels column has ten features. Henceforth, we take num_digits=10.
Using the to_categorical we have been able to create an array that contains ten elements.

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

Figure: Checking the values of y_test

As we discussed above, y_test shows an array containing ten elements because our target variable ‘label’ contained ten parameters starting from 0-9.

```
# Input Parameters
n_input = 45 # number of features
n_hidden_1 = 300
n_hidden_2 = 100
n_hidden_3 = 100
n_hidden_4 = 200
num_digits = 10
```

Figure: Taking in the input values

In the figure above, we have taken the n_input as 45 because our total number of columns after the feature extraction comes out to be 45.

```
model = tf.keras.Sequential([
  layers.Flatten(input_shape = (45,)),
  layers.Dense(128, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dropout(.1),
  layers.Dense(10, activation = 'softmax')
])
```

Figure: Keras Sequential

After this, we fit our model where we describe the activation to ‘relu’. Relu makes sure that it will give only positive output and will suppress the negative output to zero.
The activation function in the output layer of neural network models, that forecasts a multinomial classification problem is the SoftMax function. (Jason Brownlee,2020)
Furthermore, we use a sequential function so that our model layers are added to the instance of the sequential class. The dense layers are the hidden layers. The input layer contains the shape of the input data. (Jason Brownlee,2017)

The input is passed as an argument to achieve the summary. We get 23,690 parameters for the purpose of training. The figure below shows the hidden and input layers.

```
# Model Summary

model.summary() # We have 23,690 parameters to estimate

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 45)                0

 dense (Dense)               (None, 128)               5888

 dense_1 (Dense)             (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 23,690
Trainable params: 23,690
Non-trainable params: 0
```

Figure: Model Summary

```
# Stochastic Gradient
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Figure: Stochastic gradient

We use the above model compilation for computing any losses between the true and the predicted labels.

```
history1 = model.fit(X_train, y_train,
                     batch_size = 64,
                     epochs = 20,
                     verbose = 2,
                     validation_data=(X_test, y_test))

Epoch 1/20
525/525 - 2s - loss: 2.1158 - accuracy: 0.3240 - val_loss: 1.9203 - val_accuracy: 0.4423 - 2s/epoch - 4ms/step
Epoch 2/20
525/525 - 1s - loss: 1.7386 - accuracy: 0.4624 - val_loss: 1.5732 - val_accuracy: 0.4951 - 1s/epoch - 2ms/step
Epoch 3/20
525/525 - 1s - loss: 1.4876 - accuracy: 0.5101 - val_loss: 1.4044 - val_accuracy: 0.5292 - 1s/epoch - 2ms/step
Epoch 4/20
525/525 - 1s - loss: 1.3752 - accuracy: 0.5343 - val_loss: 1.3226 - val_accuracy: 0.5496 - 1s/epoch - 2ms/step
Epoch 5/20
525/525 - 1s - loss: 1.3083 - accuracy: 0.5511 - val_loss: 1.2682 - val_accuracy: 0.5667 - 1s/epoch - 2ms/step
Epoch 6/20
525/525 - 1s - loss: 1.2625 - accuracy: 0.5651 - val_loss: 1.2278 - val_accuracy: 0.5769 - 1s/epoch - 2ms/step
Epoch 7/20
525/525 - 1s - loss: 1.2264 - accuracy: 0.5764 - val_loss: 1.1947 - val_accuracy: 0.5886 - 1s/epoch - 2ms/step
Epoch 8/20
525/525 - 1s - loss: 1.1956 - accuracy: 0.5907 - val_loss: 1.1672 - val_accuracy: 0.5963 - 1s/epoch - 2ms/step
```

Figure: Model fitting

Epoch is the number of times the dataset is sent back ward and forward in a dataset.The accuracy towards the end comes to 67%. We have considered a batch size of 67 with 20 iterations to get to accuracy.

```
y_pred_nn = model.predict(X_test)

263/263 [==============================] - 0s 1ms/step
```

Figure: Predicting the Model

```
print(classification_report(y_pred_nn,pd.DataFrame(y_test).idxmax(axis = 1)))

              precision    recall  f1-score   support

           0       0.86      0.81      0.83       873
           1       0.93      0.79      0.85      1091
           2       0.59      0.64      0.62       758
           3       0.50      0.61      0.55       704
           4       0.46      0.63      0.53       590
           5       0.56      0.59      0.57       743
           6       0.88      0.85      0.86       866
           7       0.71      0.47      0.57      1335
           8       0.51      0.58      0.55       718
           9       0.41      0.47      0.44       722

    accuracy                           0.65      8400
   macro avg       0.64      0.64      0.64      8400
weighted avg       0.67      0.65      0.65      8400
```

Figure: Accuracy of Neural Network

Finally, the classification report gives us an accuracy of 67% for Neural Networks.

Recommendations

There is no right or wrong number of batch sizes to get the perfect epoch. The model can easily get overfitted. It is good to use gradient descent as an iterative feature and a limited dataset to get an optimum epoch value.

Conclusion

In this model, we get that XG Boost gives a good accuracy percentage of 70%.

We have been able to recognize the labels that need to be considered while understanding whether a student's motor skills need enhancement or not.

The recall and the precision in the figure above help us to give percentages against each label.

Model	Accuracy
Random Forest Classifier	69.69%
KNN	60.023%
XGBoost	70.55%
Support Vector Machine	70.17%
Neural Network	67%

References

ActiveState. (2022, July 11). *What is a Keras model and how to use it to make predictions*. https://www.activestate.com/resources/quick-reads/what-is-a-keras-model/

Wood, T. (2020, June 25). *Softmax Function*. DeepAI. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

Tseng, G. (2018, November 29). *Gradient Boosting and XGBoost - Gabriel Tseng*. Medium. https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5

Awasthi, S. (2020, December 17). *Seven Most Popular SVM Kernels*. Dataaspirant. https://dataaspirant.com/svm-kernels/

Jason Brownlee. (2020, October 19). Retrieved October 27, 2022, from https://machinelearningmastery.com/softmax-activation-function-with-python/

Jason Brownlee. (2017, May 28). Retrieved October 27, 2022, from https://machinelearningmastery.com/keras-functional-api-deep-learning/

Annexure

Import the Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from google.colab import drive
drive.mount('/content/drive')

# Read the file
letter_analysis=pd.read_csv("/content/drive/MyDrive/letters.csv")

letter_analysis.head(n=10)

#The dataframe does not have any duplicates
letter_analysis.duplicated()

#Columns
letter_analysis.columns
```

Checking for Null Values

```
#Checking for the null values
letter_analysis.isnull().sum()
#there are not many nukll values
```

Checking the Dimension

```
#Checking the Dimensions of the Data Set
letter_analysis.shape

from sklearn import datasets, neighbors
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
from random import randint
import time

from keras import models
from keras import layers
from keras.utils import to_categorical

np.random.seed(1212)
import keras
from keras.models import Model
from keras.layers import *
from keras import optimizers

import tensorflow as tf
from tensorflow.keras import layers

#We are doing the value count of the label
letter_analysis['label'].value_counts()
```

```
X = letter_analysis.drop(columns = ['label'], axis = 1)
y = letter_analysis['label']
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size = 0.8, random_state = 12)
y_train
```

Understanding the Dimension

```
X_train.shape,X_test.shape

y_train.shape,y_test.shape
```

Inserting the Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=245,criterion='entropy',random_state=0,min_sam
ples_split=2)
rfc.fit(X_train,y_train)
y_pred=rfc.predict(X_test)
from sklearn.metrics import accuracy_score
ac= accuracy_score(y_test,y_pred)
print('Accuracy is :',ac*100)
```

KNN Model

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(algorithm='brute',n_neighbors =1 ,leaf_size=100,p=30)
knn.fit(X_train, y_train)
knn_predictions = knn.predict(X_test)
acc=accuracy_score(y_test,knn_predictions)
print('Accuracy is :',acc*100)
```

XGBoost Model

```
from xgboost import XGBClassifier
model = XGBClassifier(learning_rate=1.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Support Vector Machine

```
from sklearn.svm import SVC
model = SVC(kernel='rbf', C=1E01,tol=0.1)
model.fit(X_train, y_train)
predicted= model.predict(X_test)
from sklearn.metrics import confusion_matrix,accuracy_score
accuracy = accuracy_score(y_test, predicted)
print("Accuracy: %.2f%%" % (accuracy * 100.0))#97.28
# Feature Normalization
```

```
X_train /= 255; X_test /= 255

# Convert labels to One Hot Encoded
num_digits = 10
# Converting labels into one hot encoded values so that our model is able to train on it, for
softmax it is required that we convert multiclass labels into their one-hot encoded form.
y_train = keras.utils.to_categorical(y_train, num_digits)
y_test = keras.utils.to_categorical(y_test, num_digits)

y_test

# Input Parameters
n_input = 45 # number of features
n_hidden_1 = 300
n_hidden_2 = 100
n_hidden_3 = 100
n_hidden_4 = 200
num_digits = 10

model = tf.keras.Sequential([
  layers.Flatten(input_shape = (45,)),
  layers.Dense(128, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dropout(.1),
  layers.Dense(10, activation = 'softmax')
])
# Inp = Input(shape=(len(X_train.columns),))
# x = Dense(n_hidden_1, activation='relu', name = "Hidden_Layer_1")(Inp)
# x = Dense(n_hidden_2, activation='relu', name = "Hidden_Layer_2")(x)
# x = Dense(n_hidden_3, activation='relu', name = "Hidden_Layer_3")(x)
# x = Dense(n_hidden_4, activation='relu', name = "Hidden_Layer_4")(x)
# output = Dense(num_digits, activation='softmax', name = "Output_Layer")(x)

# Our model would have '6' layers - input layer, 4 hidden layer and 1 output layer
model.summary() # We have 297,910 parameters to estimate

# Insert Hyperparameters
learning_rate = 0.1
batch_size = 100
sgd = optimizers.SGD(learning_rate)

#
model.compile(loss='categorical_crossentropy',
            optimizer='sgd',
            metrics=['accuracy'])


history1 = model.fit(X_train, y_train,
                  batch_size = 64,
                  epochs = 20,
                  verbose = 2,
                  validation_data=(X_test, y_test))


y_pred_nn = model.predict(X_test)
```

```
# This step is necessary to convert our one hot encoded classes into digits category column.
y_pred_nn = pd.DataFrame(y_pred_nn)
y_pred_nn = y_pred_nn.idxmax(axis = 1)

# Classification report to see the precision of our model.
print(classification_report(y_pred_nn,pd.DataFrame(y_test).idxmax(axis = 1))) # We have converted y_test back to its categorical form from one hot encoded obtained from to_categical() function.
```