**Convolution Neural Network: Facial Expression classification**

**Abstract**

Computer Vision (CV), in today’s world has become one of the most trending and discussed topics for research because of its application in huge number of fields. This amazing technology of precisely classifying and identifying an emotion provides an enormous opportunity for developing an advanced Human and Computer Interaction. This paper demonstrates one of the major application of CV which is Facial expression classification through facial emotions using Convolution Neural Network (CNN). We have also utilized Graphics Processing Unit (GPU) computation (available by notebook in the Google colab) so that we can accelerate the training process. Now we have built this CNN model with the help of Python using a well known Tensorflow module called Keras [2]. Different types of expressions that can be classified by our model is based on *fer2013* dataset [1]. This dataset was made available to the users on a public platform by a well known site for the dataset repositories called Kaggle (mainly the dataset was uploaded for a competition). The dataset was a grayscale image and the dimension of the face in the dataset that was available was forty-eight by forty eight. The labeling of the dataset was such that the expressions of the face were available in a numerical form where the number ‘0’ indicated the emotion Angry, number ‘1’ denoted the emotion Disgust, ‘2’ denoted the emotion Fear, ‘3’ denoted the Happy state of the face, ‘4’ denoted Sadness, ‘5’ denoted Surprise and finally the number ‘6’ denoted a Neutral state of the face. Thus a total of 7 different emotions can be classified through this model. Performance measures of the model and prediction results were obtained and visualized for discussion.

**Keywords**: Facial Emotion classification, Emotion detection, CNN

1. **Introduction**

When we observe the interaction between living entities (humans) we find that they exchange their emotions most of the time through facial expressions and this forms an integral part of body language such as gestures, mannerisms, bodily movements, etc. Facial expressions also plays a key role in interpersonal relations [3].

Different departments within the realm of science and technology such as behavioral science is in immense need for a model or mechanism which can automatically recognize the emotion of the person based on their facial expressions. We can find many fields of research work moving in this direction but development of an automated system that can be used to achieve this task is difficult [4].

The purpose for writing this paper is to accurately identify and classify the emotion of the person based on their facial expressions using Advanced Deep Learning Technologies such as Convolution Neural Network (CNN). After we train the model with significant amount of training; this CNN model provides us the solution to many issues pertaining to the field of Facial Expression classification. The input into the model is a grayscale image of size we have already discussed in the abstract part; and then with the help of CNN we predict one of these facial expression label: the first of these labels is anger, then comes disgust, followed by happiness, fear, sadness, disgust and lastly the emotion associated with neutral facial expression.

**Related Work**

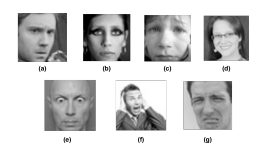
In classifying the expressions, significant development have been made in recent years [5]. Already some work has been done on classifying the emotions such as happiness, anger and sadness by building facial emotions models [6]. Next by using Action units or AU in short, we can classify the movements of the human face just by looking at their emotions on their face. This system or the model is called Facial Action Coding System (FACS) [7].

Now there are also different models built or developed based on Bayesian Networks and networks based on neurons (Graph neural network or Artificial Neural network) and HMM model to detect and classify the emotions based on the facial expressions. There are many drawbacks associated with the recognition rate and the accuracy of the model. We can then combine 2 or more techniques to achieve accurate classification and then, the most important features can be extracted as needed.

1. **Problem and Data set(s) description**

For this research, we used the *fer2013* dataset [1]. This dataset was made available to the users on a public platform by a well known site for the dataset repositories called Kaggle (mainly the dataset was uploaded for a competition). The dataset was a grayscale image and the dimension of the face in the dataset that was available was forty-eight by forty eight and a total of 36000 samples. The labeling of the dataset was such that the expressions of the face were available in a numerical form where the number ‘0’ indicated the emotion Angry, number ‘1’ denoted the emotion Disgust, ‘2’ denoted the emotion Fear, ‘3’ denoted the Happy state of the face, ‘4’ denoted Sadness, ‘5’ denoted Surprise and finally the number ‘6’ denoted a Neutral state of the face. Thus a total of 7 different emotions can be classified through this model. In these images, we find that faces almost take the same quantity of disk space for all the faces that we can find in the dataset and the faces are also aligned to the center with equal padding and margins from each sides.

There are about 28,700 training images, 3,600 validation images, and 3,600 images for testing. Out of these, approximately 5000 images are reserved for anger, 550 images are reserved for disgust emotion, 5100 for fear, 9000 for happy face, 6000 for sad emotion, 4000 for surprise, and 6200 for neutral faces. Figure -1 demonstrates the examples of all the different emotions for facial expressions (seven in total) that we consider to detect and identify in our problem.



**Figure 1**: Examples of all the different emotions for facial expressions (seven in total) that we consider to detect and identify in our problem. They are (a) angry, (b) neutral, (c) sad, (d) happy, (e) surprise, (f) fear, (g) disgust

1. **Methods**

We have built a CNN model with the help of Python using a well known Tensorflow module called Keras with different convolution blocks to measure different performance metrics such as accuracy, precision, recall, F1 score, classification matrix, etc of these models for facial expression identification and detection through facial emotions.

**Input Layer**: The column in the csv file with the name 'raw pixel values’ which contains the information of the image in pixel format are given as an input to our model. Before we feed the images into this layer, preprocessing is done beforehand.

**Dense layer**: The final layer obtained from the max pooling layer is flattened and is transformed into 1D feature vector to send it to the Fully connected layers followed by the hidden layers in order to classify the emotions

**Convolution layer**: When the images are provided in 2D input shape, the model has different parameters such as weights, biases and filters. These parameters along with the input (in the form of neurons) connect with each other through a mathematical expression (basically a dot product between weights, biases and filters). Now the number of convolution layers is very big, so the computational time for evaluating the mathematical expressions increases with increasing layers and therefore to reduce this time, these layers have to go through the technique called 'max pooling' which decreases the size (basically the height and width of the image) of the sample and thus we can successfully decrease the huge number of these layers which eventually decreases the computational time for evaluating those mathematical expressions.

**Output layer**: This is the last layer in our model which is added to the fully connected layers that we have developed in the previous section. This section basically outputs the required emotions or the probability of each of the emotions using an activation function called softmax function.

The network architecture that the paper demonstrates is as follows:

We initialized our model by dividing it the convolution layers into four different blocks. The first block contains two convolution layers with 64 different filters and ‘ReLU’ activation unit followed by a Max pooling layer with two strides. The next block contains two convolution layers with 128 different filters and ‘ReLU’ activation unit followed by a Max pooling layer with two strides. Third block contains three convolution layers with 128 different filters and ‘ReLU’ activation unit followed by a Max pooling layer with two strides. Finally, the fourth block contains 3 convolution layers with 512 different filters and ‘ReLU’ activation unit followed by a Max pooling layer with two strides.

Then the model was flattened and two dense layer having 4096 neurons along with ‘ReLU’ activation function and a dropout of 0.3 was added. Lastly, an output layer with 7 neurons (each neuron representing a specific emotion) was added to the dense layers (or the fully connected layer that we have just developed) and outputs the required facial emotions or their probabilities using an activation function called softmax activation function.

These gave us a total of 43,322,567 trainable parameters. Our model which is developed is fully flexible in terms of the number of convolutional layers that we want to choose or the number of hidden layer neurons that we would like to add, as well as the type of batch normalization, or the dropout fraction and number of max-pooling layers that we would like to add.

Not only the number of convolutional layers, but also the freedom was provided to add the number of filters, strides, and zero-padding as per the requirements of the user. In case the user does not provide these values then we have made arrangements to use the default values.

We implemented the aforementioned model with the help of Python using a well known Tensorflow module called Keras [2] and we have also utilized Graphics Processing Unit (GPU) computation (available by notebook in the Google colab) so that we can accelerate the training process.

As mentioned before the original data was divided into images (X) and label (y), which were then splitted into train, validation and test sets. Thus, the algorithms would be trained on one set of data and tested out on a completely different set of data (not seen before by the algorithm). We used all the 28700 samples in train dataset in order to train the model. The number of epochs and batch size (which are also flexible and can be adjusted by the user) was set to a default value of 30 and 32 respectively. Now along with training the model, we were simultaneously calculating the validation accuracy (for validating our models) using all the available 3,600 validation images. Finally we tested our model (the performance of our model in order to evaluate it) using all the available 3,600 test images. The model gave us 60% accuracy on both the validation set and on the test set.

**4. Results**

Its time to find the performance of our model (which we evaluated after training the model and fitting the model with test dataset). Using the feature of the Keras model.history, we were able to find the accuracy history (for both training and validation) of the model with respect to the number of epochs for which it was trained and plotted the same (see Figure 2)

against the number of epochs.

The model was able to achieve the validation accuracy of 60% after just training for 10 to 15 epochs (Figure 2 is evident for the same). The over fitting of the model was also reduced due to addition of the non-linear feature to the model

Next the confusion matrix was evaluated for the CNN model which is plotted in Figure - 3 [8]. The confusion matrix plot shows that the accuracy (or the true prediction) for most of the facial expressions or labels is high (as opposed to the random guess of 14.28%) which shows that the model performed quite well on identifying the emotions.

It is quite amazing to note that the happy and Surprise facial expressions were detected with high precision and recall as opposed to the other categories or labels or emotions, which shows that predicting the happy or a surprise face is easier as compared to other expressions.

As the name implies, this confusion matrix shows in which expressions the model was confused in identifying the facial emotions (for the trained model). The expressions for which the model did not performed as expected (or the model showed less precision and recall rate) were sad, fear and neutral emotions.

The mode misidentified the predicted label as neutal or fear expressions when the actual label was sad and vice versa for almost half of the samples.

These observations are indeed interesting to see because these mis-identifications are consistent with what we see when we look at different images in the dataset; The same case happens with us living entities also that we most of the time misidentify the sad person or a fearful person as having a neutral emotion because the humans do not always show their expressions or emotions in the similar fashion and thus the mis-identification occurs.

Finally, we computed the accuracy of each model for every expression or class. Table-1 shows these results. The facial emotion present in the data are (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). From this table we find that, the happy and Surprise facial expressions were detected with high accuracy as opposed to the other categories or labels or emotion.

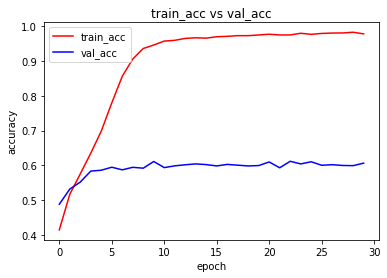
After performing the task of classification, not only the test dataset accuracy but also other performance metrics, namely , precision, recall, and F1 score were evaluated. Table-2 shows the performance of the CNN in the other performance metrics.

**Table-1**: Class wise accuracy

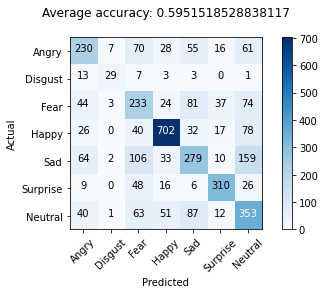
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Class** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| **Accuracy** | 0.49 | 0.52 | 0.47 | 0.78 | 0.43 | 0.75 | 0.58 |

**Table-2:** Precision, Recall, and F1 values

|  |  |  |  |
| --- | --- | --- | --- |
| **Emotions** | **Precision** | **Recall** | **F1** |
| Anger | 0.54 | 0.49 | 0.51 |
| Disgust | 0.69 | 0.52 | 0.59 |
| Fear | 0.41 | 0.47 | 0.44 |
| Happy | 0.82 | 0.78 | 0.80 |
| Sad | 0.51 | 0.43 | 0.47 |
| Surprise | 0.77 | 0.75 | 0.76 |
| Neutral | 0.47 | 0.58 | 0.52 |



**Figure 2**: Training and Validation accuracy of CNN model vs number of epochs



**Figure 3**: Confusion Matrix

1. **Discussions and Conclusion**

This amazing technology of precisely classifying and identifying an emotion provides an enormous opportunity for developing an advanced Human and Computer Interaction. Thus in this paper we demonstrated one of the major application of Computer Vision which is Emotion Detection and we implemented the same with the help of facial expressions dataset and using Convolution Neural Network (CNN).

We developed CNN model for accurately identifying the emotions using facial expressions and also looked at different performance metrics and plots for evaluating our model. The results demonstrated that very subtle facial features or characteristics can be learned by CNNs and the results can further be improved by adjusting or tuning in certain hyperparameters which results in high accuracy of the model.

However, there are still various unanswered questions that need to be addressed regarding the execution of emotion classification. They are as follows:

• Which feature is both most important (for facial emotion classification) and scalable?

• How to reduce dimensionality of the images over large training samples.

For the successful completion of the project, we trained all the models from scratch using CNN packages with the help of Python using a well known Tensorflow module called Keras. Now we would like to extend our model to an RGB images (i.e. color images). This way we can scrutinize the effectiveness of already pre-trained models such as ALEX-NET [9] or VGGNet [10] for identifying the emotion through facial expressions. We can also extend this project further by implementing face detection model prior to the emotion prediction.

1. **References**
2. <https://www.kaggle.com/deadskull7/fer2013>
3. Chollet, F. & others, 2015. Keras. Available at: <https://github.com/fchollet/keras>

[3] Bettadapura, Vinay (2012). Face expression

detection, identification and analysis: the state of the art. arXiv preprint arXiv:1203.6722

[4] Yafei Sun, Theo Gevers, et al. (2007) Authentic Facial Expression Analysis. Image and Vision Computing 25.12: 1856-1863

[5] Y. Tian, T. Kanade, and J. Cohn. Recognizing

action units for facial expression analysis. IEEE

Transactions on Pattern Analysis and Machine Intelligence, 23(2), 2001.

[6] G. Littlewort, M. Bartlett, I. Fasel, J. Susskind,

and J. Movellan. Dynamics of facial expression

extracted automatically from video. Image and Vision Computing, 24(6), 2006.

[7] P. Ekman, W. Friesen, Facial Action Coding System: A Technique for the Measurement of Facial Movement, Consulting Psychologists Press, 1978

1. <http://scikit-learn.org/stable/>

[9] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey

E. Hinton. ”Imagenet classification with deep convolutional neural networks.” Advances in neural information processing systems. 2012.

[10] Andrew Zisserman. ”Deep convolutional networks for large-scale image recognition.” arXiv preprint arXiv:1409.1556 (2014).

**Appendix**

'''

We utilized GPU in the "Google Colab"

to get fast processing. We can enable it by going to "Runtime in Google Colab

and then clicking on Change runtime type" and select GPU. Once it is enabled we will now

import the required libraries for building the network.

'''

**import** pandas **as** pd

**from** matplotlib **import** pyplot

**from** math **import** sqrt

**import** numpy **as** np

**from** IPython**.**display **import** display

**from** keras**.**utils **import** np\_utils

**from** keras**.**utils **import** plot\_model

**from** keras**.**models **import** Sequential

**from** keras**.**layers **import** Conv2D**,** MaxPool2D**,** Dense**,** Flatten**,** Dropout

**from** keras**.**optimizers **import** Adam**,** SGD

**from** matplotlib **import** pyplot **as** plt

**from** sklearn**.**metrics **import** confusion\_matrix

**import** itertools

**import** warnings

warnings**.**filterwarnings**(**'ignore'**)**

%matplotlib inline

'''

The name of the data set is fer2013. The dataset was a grayscale image and the dimension of the face in the dataset that was available was forty-eight by forty eight. The labeling of the dataset was such that the expressions of the face were available in a numerical form where the number ‘0’ indicated the emotion Angry, number ‘1’ denoted the emotion Disgust, ‘2’ denoted the emotion Fear, ‘3’ denoted the Happy state of the face, ‘4’ denoted Sadness, ‘5’ denoted Surprise and finally the number ‘6’ denoted a Neutral state of the face. Thus a total of 7 different emotions can be classified through this model.

The CSV file contains two columns that are emotion that contains numeric code from 0-6 and a pixel column that includes a string surrounded in quotes for each image.

'''

data**=** pd**.**read\_csv**(**'fer2013.csv'**)**

data**.**head**()**

**from** pylab **import** rcParams

rcParams**[**'figure.figsize'**]** **=** 15**,** 10

'''

There are about 28,700 training images, 3,600 validation images, and 3,600 images for testing.

Out of these, approximately 5000 images are reserved for anger, 550 images are reserved for disgust emotion, 5100 for fear, 9000 for happy face, 6000 for sad emotion, 4000 for surprise, and 6200 for neutral faces.

'''

data**.**emotion**.**value\_counts**()**

num\_classes **=** 7

width **=** 48

height **=** 48

emotion\_labels **=** **[**"Angry"**,** "Disgust"**,** "Fear"**,** "Happy"**,** "Sad"**,** "Surprise"**,** "Neutral"**]**

classes**=**np**.**array**((**"Angry"**,** "Disgust"**,** "Fear"**,** "Happy"**,** "Sad"**,** "Surprise"**,** "Neutral"**))**

data**.**Usage**.**value\_counts**()**

'''

We then create different lists of storing the testing and training image pixels.

After this, we check if the pixel belongs to training then we append it into the

training list & training labels. Similarly, for pixels belonging to the Private and Public test,

we append it to validation and testing lists.

'''

X\_train**,** y\_train **=** **[],** **[]**

X\_val**,** y\_val **=** **[],** **[]**

X\_test**,** y\_test **=** **[],** **[]**

**for** index**,** row **in** data**.**iterrows**():**

k **=** row**[**'pixels'**].**split**(**" "**)**

**if** row**[**'Usage'**]** **==** 'Training'**:**

X\_train**.**append**(**np**.**array**(**k**))**

y\_train**.**append**(**row**[**'emotion'**])**

**elif** row**[**'Usage'**]** **==** 'PublicTest'**:**

X\_test**.**append**(**np**.**array**(**k**))**

y\_test**.**append**(**row**[**'emotion'**])**

**elif** row**[**'Usage'**]** **==** 'PrivateTest'**:**

X\_val**.**append**(**np**.**array**(**k**))**

y\_val**.**append**(**row**[**'emotion'**])**

'''

Once we have added the pixel to the lists then we convert them into NumPy arrays and reshape X\_train,

X\_val and X\_test. After doing this we convert the training labels,

validation labels and testing labels into categorical ones.

'''

X\_train **=** np**.**array**(**X\_train**).**astype**(**float**)**

y\_train **=** np**.**array**(**y\_train**)**

X\_val **=** np**.**array**(**X\_val**).**astype**(**float**)**

y\_val **=** np**.**array**(**y\_val**)**

X\_test **=** np**.**array**(**X\_test**).**astype**(**float**)**

y\_test **=** np**.**array**(**y\_test**)**

X\_train **=** X\_train**.**reshape**(**X\_train**.**shape**[**0**],** 48**,** 48**,** 1**)**

X\_val **=** X\_val**.**reshape**(**X\_val**.**shape**[**0**],** 48**,** 48**,** 1**)**

X\_test **=** X\_test**.**reshape**(**X\_test**.**shape**[**0**],** 48**,** 48**,** 1**)**

y\_train**=** np\_utils**.**to\_categorical**(**y\_train**,** num\_classes**=**num\_classes**)**

y\_val**=** np\_utils**.**to\_categorical**(**y\_val**,** num\_classes**=**num\_classes**)**

y\_test **=** np\_utils**.**to\_categorical**(**y\_test**,** num\_classes**=**num\_classes**)**

'''

We considered the following network architecture in our investigation:

We initialized our model by dividing it the convolution layers into four different blocks.

The first block contains two convolution layers with 64 different filters and

‘ReLU’ activation unit followed by a Max pooling layer with two strides.

The next block contains two convolution layers with 128 different filters and

‘ReLU’ activation unit followed by a Max pooling layer with two strides.

Third block contains three convolution layers with 128 different filters and

‘ReLU’ activation unit followed by a Max pooling layer with two strides.

Finally, the fourth block contains 3 convolution layers with 512 different filters and

‘ReLU’ activation unit followed by a Max pooling layer with two strides.

Then the model was flattened and two dense layer having 4096 neurons along with

‘ReLU’ activation function and a dropout of 0.3 was added. Lastly, an output layer

with 7 neurons (each neuron representing a specific class) was added to the previous

fully connected layer and outputs the required classes or their probabilities using softmax activation function.

'''

**def** CNN\_Model**(** input\_shape **=** **(**48**,**48**,**1**)** **):**

# first input model

num\_classes **=** 7

model **=** Sequential**()**

#the 1-st block

model**.**add**(**Conv2D**(**64**,** 3**,** activation **=** 'relu'**,** padding **=** "same"**,** input\_shape **=** input\_shape**))**

model**.**add**(**Conv2D**(**64**,** 3**,** padding **=** "same"**,** activation**=**'relu'**))**

model**.**add**(**MaxPool2D**(** **(**2**,**2**),** strides **=** 2**))**

model**.**add**(**Conv2D**(**128**,** 3**,** padding **=** "same"**,** activation **=** 'relu'**))**

model**.**add**(**Conv2D**(**128**,** 3**,** padding **=** "same"**,** activation **=** 'relu'**))**

model**.**add**(**MaxPool2D**(** **(**2**,**2**),** strides **=** 2**))**

model**.**add**(**Conv2D**(**256**,** 3**,** padding **=** "same"**,** activation **=** 'relu'**))**

model**.**add**(**Conv2D**(**256**,** 3**,** padding **=** "same"**,** activation **=** 'relu'**))**

model**.**add**(**Conv2D**(**256**,** 3**,** padding **=** "same"**,** activation **=** 'relu'**))**

model**.**add**(**MaxPool2D**(** **(**2**,**2**),** strides **=** 2**))**

model**.**add**(**Conv2D**(**512**,** 3**,** padding **=** "same"**,** activation **=** 'relu'**))**

model**.**add**(**Conv2D**(**512**,** 3**,** padding **=** "same"**,** activation **=** 'relu'**))**

model**.**add**(**Conv2D**(**512**,** 3**,** padding **=** "same"**,** activation **=** 'relu'**))**

model**.**add**(**MaxPool2D**(** **(**2**,**2**),** strides **=** 2**))**

model**.**add**(**Flatten**())**

model**.**add**(**Dense**(**4096**,** activation **=** 'relu'**))**

model**.**add**(**Dropout**(**0.3**))**

model**.**add**(**Dense**(**4096**,** activation **=** 'relu'**))**

model**.**add**(**Dropout**(**0.3**))**

model**.**add**(**Dense**(**num\_classes**,** activation **=** 'softmax'**))**

# summary layers

**print(**model**.**summary**())**

**return** model

model **=** CNN\_Model**()**

'''

After this, we compile the model using Adam as an optimizer,

loss as categorical cross-entropy, and metrics as accuracy as shown below.

'''

model**.**compile**(**loss**=**'categorical\_crossentropy'**,** optimizer**=**Adam**(** lr**=**0.0001**,** decay**=**1e-6 **),** metrics **=** **[**'accuracy'**])**

**import** tensorflow **as** tf

tf**.**config**.**run\_functions\_eagerly**(True)**

'''

After compiling the model we then fit the data for training and validation.

Here, we are taking the batch size to be 32 with 30 epochs.

You can tune them according to your wish.

'''

history **=** model**.**fit**(** X\_train**,** y\_train**,** batch\_size **=** 32**,** epochs **=** 30**,** verbose **=** 1**,** validation\_data **=** **(**X\_val**,** y\_val**))**

# Commented out IPython magic to ensure Python compatibility.

# visualizing losses and accuracy

# %matplotlib inline

'''

We are plotting the training and validation accuracy of thye model

with respect to the number of epochs

'''

train\_acc **=** history**.**history**[**'accuracy'**]**

val\_acc **=** history**.**history**[**'val\_accuracy'**]**

epochs **=** range**(**len**(**train\_acc**))**

plt**.**plot**(**epochs**,** train\_acc**,** 'r'**,** label **=** 'train\_acc'**)**

plt**.**plot**(**epochs**,** val\_acc**,** 'b'**,** label **=** 'val\_acc'**)**

plt**.**title**(**'train\_acc vs val\_acc'**)**

plt**.**xlabel**(** 'epoch' **)**

plt**.**ylabel**(** 'accuracy' **)**

plt**.**legend**()**

plt**.**figure**()**

'''

We now serialize the model to JSON and save the model weights in an hd5 file so

that we can make use of this file to make predictions rather than training the network again.

'''

model\_json **=** model**.**to\_json**()**

**with** open**(**"model.json"**,** "w"**)** **as** json\_file**:**

json\_file**.**write**(**model\_json**)**

model**.**save\_weights**(**"model.h5"**)**

**print(**"Saved model to disk"**)**

loss **=** model**.**evaluate**(**X\_test**,** y\_test**)**

**print(**"Test Loss " **+** str**(**loss**[**0**]))**

**print(**"Test Acc: " **+** str**(**loss**[**1**]))**

'''

As the name of the function suggests, we are plotting the confusion matrix

with the help of this function

'''

**def** plot\_confusion\_matrix**(**y\_test**,** y\_pred**,** classes**,**

normalize**=False,**

title**=**'Unnormalized confusion matrix'**,**

cmap**=**plt**.**cm**.**Blues**):**

cm **=** confusion\_matrix**(**y\_test**,** y\_pred**)**

**if** normalize**:**

cm **=** np**.**round**(**cm**.**astype**(**'float'**)** **/** cm**.**sum**(**axis**=**1**)[:,** np**.**newaxis**],** 2**)**

np**.**set\_printoptions**(**precision**=**2**)**

plt**.**imshow**(**cm**,** interpolation**=**'nearest'**,** cmap**=**cmap**)**

plt**.**title**(**title**)**

plt**.**colorbar**()**

tick\_marks **=** np**.**arange**(**len**(**classes**))**

plt**.**xticks**(**tick\_marks**,** classes**,** rotation**=**45**)**

plt**.**yticks**(**tick\_marks**,** classes**)**

thresh **=** cm**.**min**()** **+** **(**cm**.**max**()** **-** cm**.**min**())** **/** 2.

**for** i**,** j **in** itertools**.**product**(**range**(**cm**.**shape**[**0**]),** range**(**cm**.**shape**[**1**])):**

plt**.**text**(**j**,** i**,** cm**[**i**,** j**],**

horizontalalignment**=**"center"**,**

color**=**"white" **if** cm**[**i**,** j**]** **>** thresh **else** "black"**)**

plt**.**tight\_layout**()**

plt**.**ylabel**(**'Actual'**)**

plt**.**xlabel**(**'Predicted'**)**

plt**.**show**()**

**return** cm

'''

Once the training has been done we can evaluate the model and compute loss and accuracy using the below code.

'''

y\_pred\_ **=** model**.**predict**(**X\_test**,** verbose**=**1**)**

y\_pred **=** np**.**argmax**(**y\_pred\_**,** axis**=**1**)**

y\_te **=** np**.**argmax**(**y\_test**,** axis**=**1**)**

cm **=** plot\_confusion\_matrix**(**y\_test**=**y\_te**,** y\_pred**=**y\_pred**,**

classes**=**classes**,**

normalize**=False,**

#cmap=plt.cm.Greys,

title**=**'Average accuracy: ' **+** str**(**np**.**sum**(**y\_pred **==** y\_te**)/**len**(**y\_te**))** **+** '\n'**)**

'''

After performing the task of classification, not only the test dataset accuracy

but also other performance metrics, namely , precision, recall, and F1 score were evaluated.

'''

TP **=** np**.**diag**(**cm**)**

FP **=** np**.**sum**(**cm**,** axis**=**0**)** **-** TP

FN **=** np**.**sum**(**cm**,** axis**=**1**)** **-** TP

num\_classes **=** len**(**cm**)**

TN **=** **[]**

**for** i **in** range**(**num\_classes**):**

temp **=** np**.**delete**(**cm**,** i**,** 0**)** # delete ith row

temp **=** np**.**delete**(**temp**,** i**,** 1**)** # delete ith column

TN**.**append**(**sum**(**sum**(**temp**)))**

precision **=** TP**/(**TP**+**FP**)**

recall **=** TP**/(**TP**+**FN**)**

F1 **=** 2**\*(**precision**\***recall**)/(**precision**+**recall**)**

**print(**precision**)**

**print(**recall**)**

**print(**F1**)**

cm **=** cm**.**astype**(**'float'**)** **/** cm**.**sum**(**axis**=**1**)[:,** np**.**newaxis**]**

**print(**np**.**diag**(**cm**))**