Title of Project: Facial Emotion Recognition using 5 different classifiers

Student Name: Eshaan Gupta

Enrolment Number: 14519051622

Email-ID: Eshaan.gupta.33@gmail.com

Contact Number: +91 98104 02669

Drive link:
https://drive.google.com/drive/folders/1RhT12Y4Pseg7SDSYjGKqbLmM5iDf5diY?usp=sharing

Title: Facial Emotion Recognition models using 5 different machine learning classifiers

Abstract:

This report proposes to expand the visual understanding capacity of computers by helping it to recognise various human emotions using machine learning algorithms. This report specially focuses on 7 prominent facial emotions. The idea or situation behind the emotions is not taken into consideration and only the image is used to train the models. The words used to express the emotions are not taken into consideration. From the perspective of a human, different emotions can have different meanings keeping the situation and words used with it. Computers can lend a hand to help humans recognise the emotion of the person and can take the course of conversation in a particular direction accordingly. The proposed idea is implemented using 5 different models like, Convolutional Neural Networks, Decision Tree, Support vector classification, K-Nearest Neighbour and Naïve Bayes Classifier. Throughout the report, these models will be compared and evaluated briefly. The pros and cons of each and every model will be then analysed on the basis of the outcome and accuracy score. However, since the data set is too low and the model development is not very complex, the models cannot be used in real life applications as of now, but can be furthered improved if worked upon.

Keywords: Deep learning, Decision Tree, Support Vector Classification, Naïve Bayes, Clustering.

Introduction:

Apart from the verbal communication power we possess, we also have the power to express our emotions through our facial expressions. The course of any conversation can be traced just by noticing the facial expressions of the people. Moreover, people having speaking impairment are also not able to express their emotions through words. Even a situation can be judged upon by seeing the facial expression of the people part participating in it. For example, if all the people are happy, then it can be said with higher probability that the environment is relaxed, fun and people are enjoying it. On the other hand if all the people around are scared or afraid, then it can be understood that something is not right. Therefore, facial expressions play a crucial role in our society and our communications.

A machine can be trained to learn various expressions of the people and then further predict. Although it may not always be right but it can lend a hand and help humans to think in a particular directions, narrowing the domain of possibilities and helping it to get optimal answer.

Proposed Methodology:

The methodology used in the models is as follows:

1. The dataset has been taken from Kaggle (https://www.kaggle.com/datasets/aadityasinghal/facial-expression-dataset).
2. The dataset comprises of 35.9k images of 7 expressions which are, angry, disgust, fear, happy, neutral, sad and surprise
3. The dataset was already clean and uniform (48x48, grayscale) and therefore, no cleaning and validation was required.
4. The dataset was already divided into 2 folders, testing and training. This made it easier to provide the training and testing data to the models to determine the accuracy of them.
5. Since the dataset comprised of images, features were extracted from them using TensorFlow Keras.preprocessing.images and stored in the form of numpy arrays.

6. For visualizing the data, popular python libraries were used i.e. Seaborn and Matplotlib
7. The accuracy of both the training data and testing data was computed to determine the cases of underfitting and overfitting.

With all the pre-requisites already understood, let us now discuss all the models one by one.

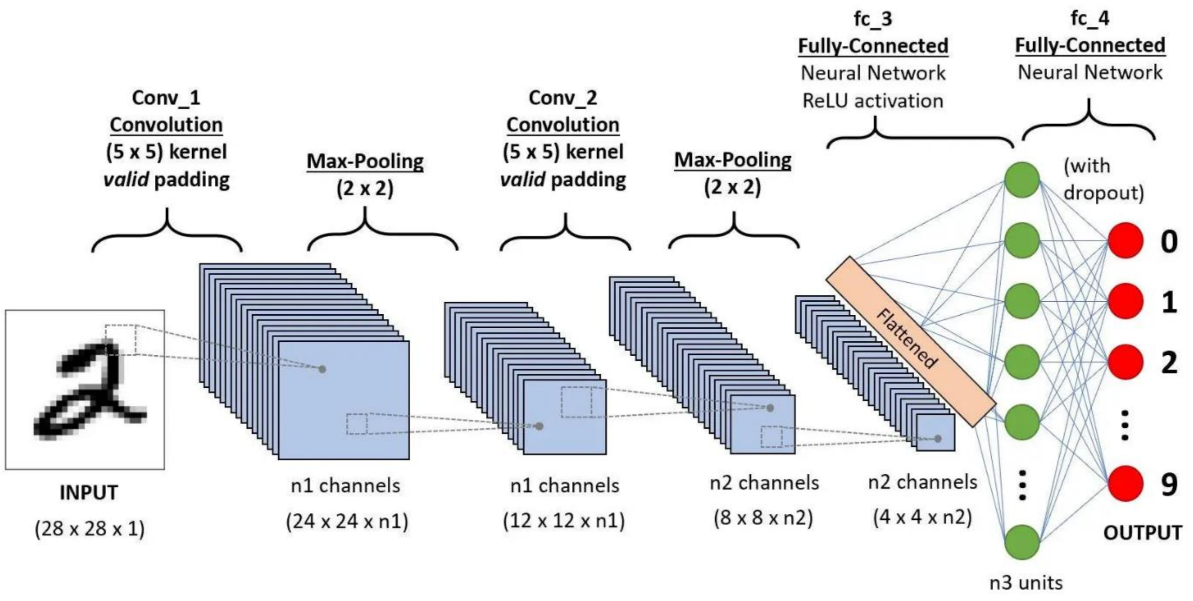
1. **Convolutional Neural Networks (CNN)**
   A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The architecture of CNNs is inspired by the visual processing in the human brain, and they are well-suited for capturing hierarchical patterns and spatial dependencies within images.

A. Convolutional Layers: These layers apply convolutional operations to input images, using filters (also known as kernels) to detect features such as edges, textures, and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.

B. Pooling Layers: Pooling layers downsample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network. Max pooling is a common pooling operation, selecting the maximum value from a group of neighboring pixels.

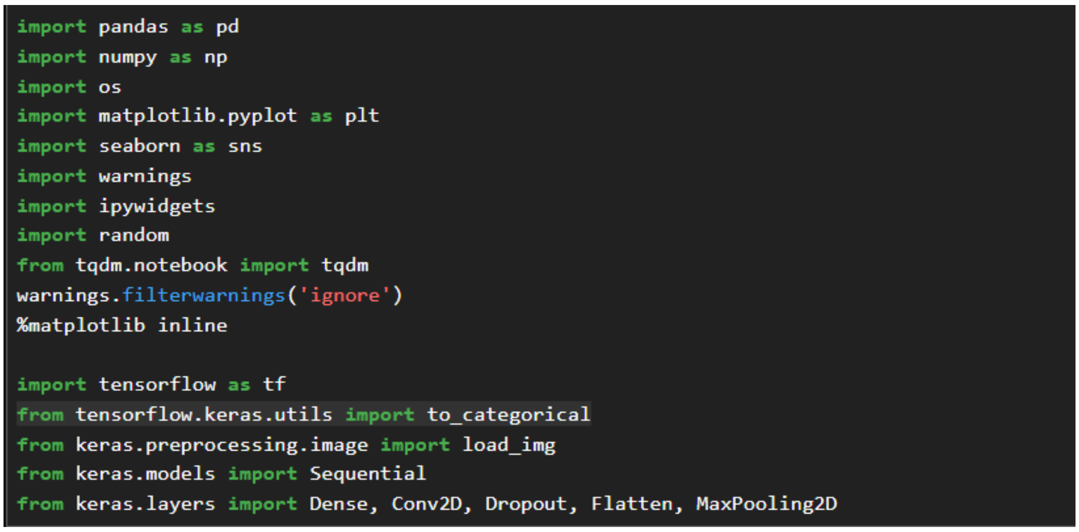
C. Activation Functions: Non-linear activation functions, such as Rectified Linear Unit (ReLU), introduce non-linearity to the model, allowing it to learn more complex relationships in the data.

D. Fully Connected Layers: These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.



A Guide to Convolutional Neural Networks — the ELI5 way | Saturn Cloud Blog
https://saturncloud.io/images/blog/a-cnn-sequence-to-classify-handwritten-digits.webp

Please note that the screenshots of the codes can be found in the jupyter notebooks and python files uploaded on the drive link.



```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import ipywidgets
import random
from tqdm.notebook import tqdm
warnings.filterwarnings('ignore')
%matplotlib inline

import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.image import load_img
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
```

The first step taken to build the model was to import all the necessary modules into the jupyter notebook. Let us look at each of them one by one

Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Numpy: NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

OS: Python has a built-in os module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.

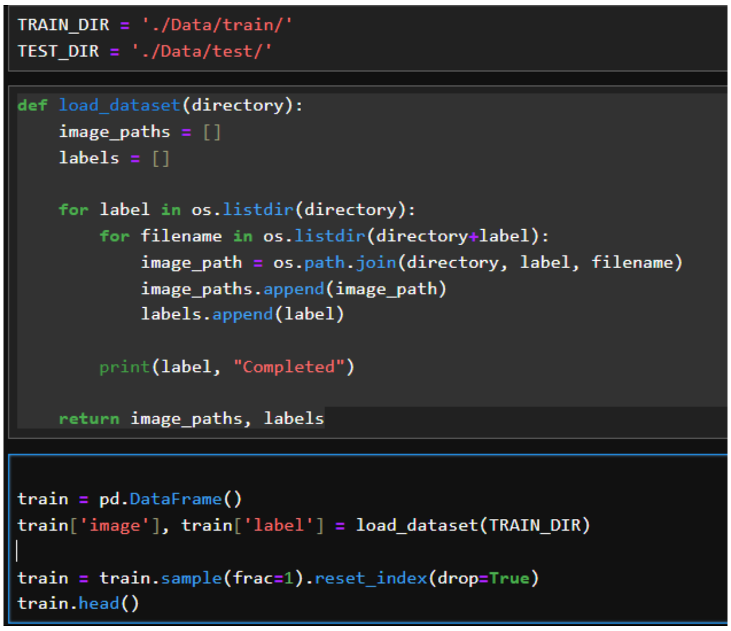
Matplotlib: Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely.

Seaborn: Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

Ipywidgets and tqdm: It is used to add UI interface to the jupyter notebook. It has been used to display the loading bar.

Tensorflow: TensorFlow is an open-source machine learning library developed by Google. TensorFlow is used to build and train deep learning models as it facilitates the creation of computational graphs and efficient execution on various hardware platforms.

Keras: Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.



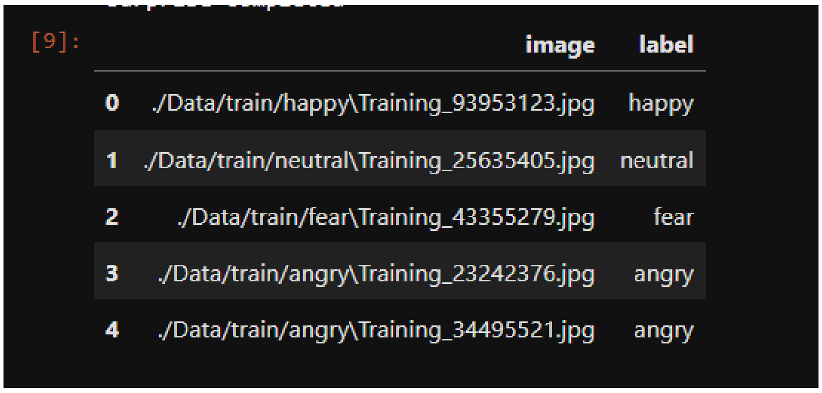
```python
TRAIN_DIR = './Data/train/'
TEST_DIR = './Data/test/'
```

```python
def load_dataset(directory):
    image_paths = []
    labels = []

    for label in os.listdir(directory):
        for filename in os.listdir(directory+label):
            image_path = os.path.join(directory, label, filename)
            image_paths.append(image_path)
            labels.append(label)

        print(label, "Completed")

    return image_paths, labels
```

```python
train = pd.DataFrame()
train['image'], train['label'] = load_dataset(TRAIN_DIR)

train = train.sample(frac=1).reset_index(drop=True)
train.head()
```

The directory of the dataset has been stored in two variables, TRAIN_DIR and TEST_DIR. A pandas dataframe is created with the name of train with two columns “image” and “label” and the data is then stored into this data frame using the user defined load_Dataset function. Similarly, the testing data is also being loaded and stored using the same method but TEST_DIR directory.

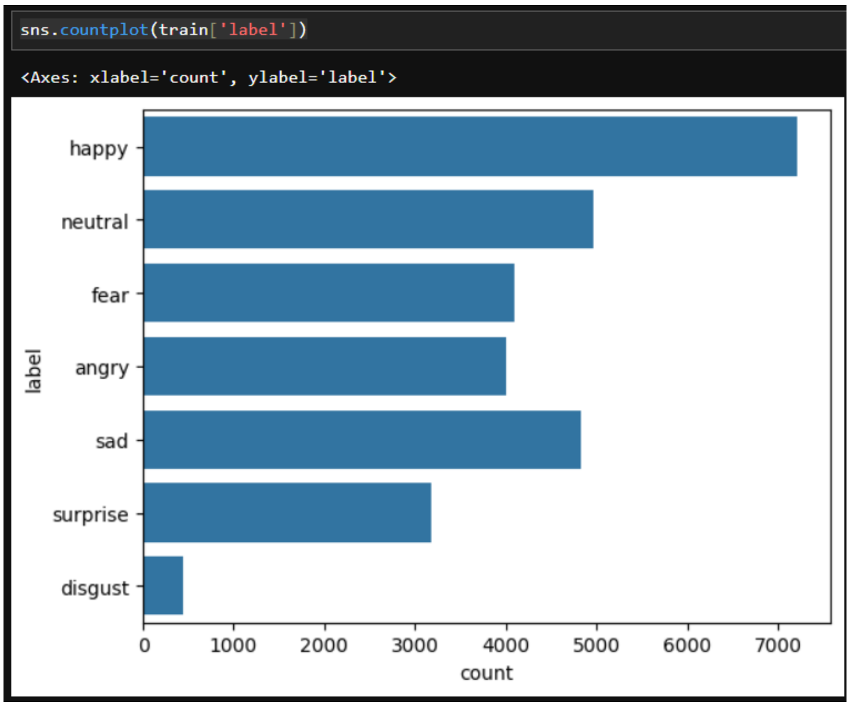
The image column is storing the name of the image and the label is storing the expression it showcases. The following image can be referred to understand it better.



[9]:

	image	label
0	./Data/train/happy\Training_93953123.jpg	happy
1	./Data/train/neutral\Training_25635405.jpg	neutral
2	./Data/train/fear\Training_43355279.jpg	fear
3	./Data/train/angry\Training_23242376.jpg	angry
4	./Data/train/angry\Training_34495521.jpg	angry

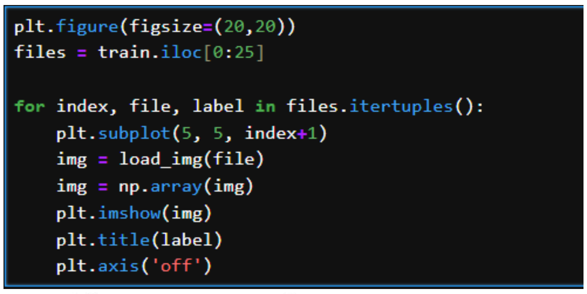
To analyse the dataset, a count plot was generated using Seaborn and following was the result.



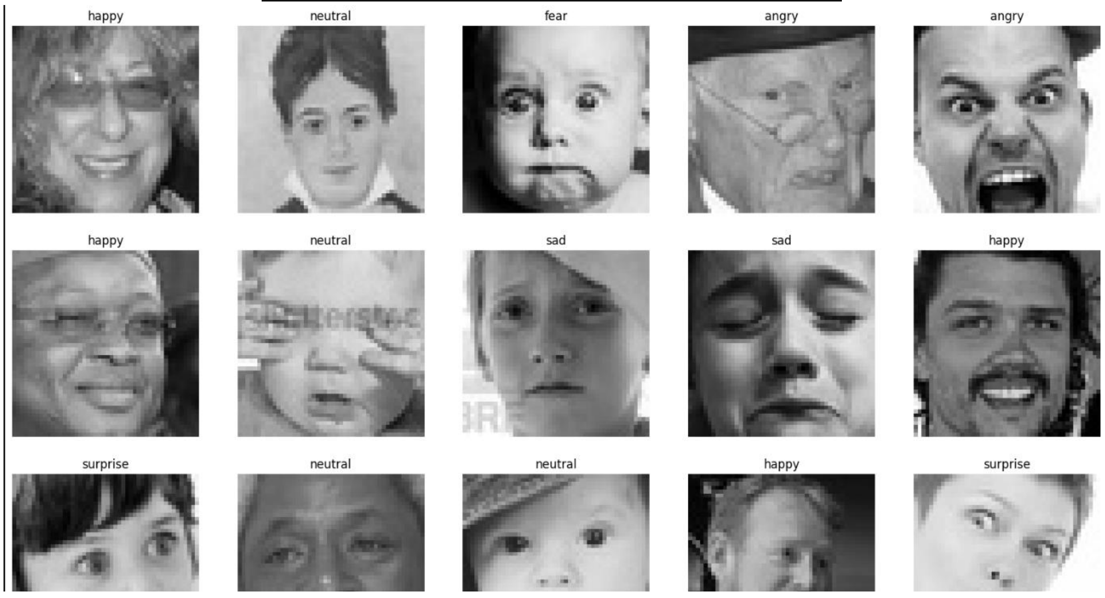
```
sns.countplot(train['label'])
```

```
<Axes: xlabel='count', ylabel='label'>
```

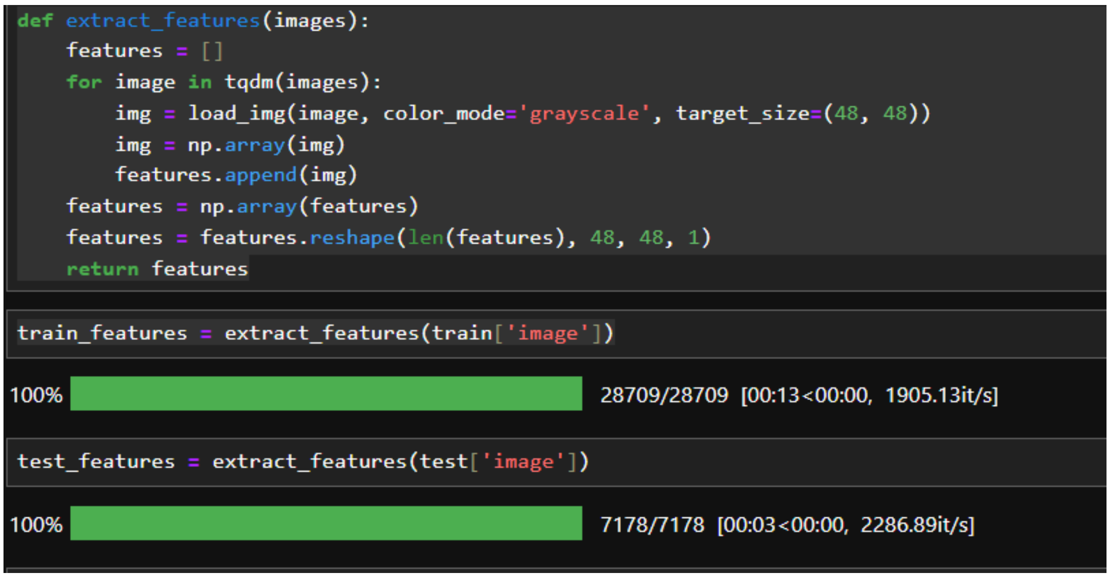
It is very clear from the above graph that the data set is not uniform. The number of images for the expression “happy” is around 7000, On the other hand, the number of images for the expression “disgust” is around 400-500. This variability will naturally affect the working and prediction of the model. On way to tackle this problem is by using class balancing techniques like Random Under Sampling, Random Oversampler, Synthetic Minority Oversampling Technique [SMOTE]. The mentioned techniques have not been implemented in this report.



```
plt.figure(figsize=(20,20))
files = train.iloc[0:25]

for index, file, label in files.itertuples():
    plt.subplot(5, 5, index+1)
    img = load_img(file)
    img = np.array(img)
    plt.imshow(img)
    plt.title(label)
    plt.axis('off')
```



To verify that the images and labels were correctly implemented and matched, a matrix of 5x5 images was generated with image and its corresponding label. The above screenshot is not showing all the 25 images but only 15 images. However, all 25 were analysed before constructing the model.



```
def extract_features(images):
    features = []
    for image in tqdm(images):
        img = load_img(image, color_mode='grayscale', target_size=(48, 48))
        img = np.array(img)
        features.append(img)
    features = np.array(features)
    features = features.reshape(len(features), 48, 48, 1)
    return features
```

```
train_features = extract_features(train['image'])
```

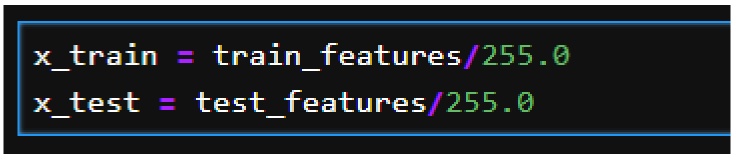
```
100%  28709/28709 [00:13<00:00, 1905.13it/s]
```

```
test_features = extract_features(test['image'])
```

```
100%  7178/7178 [00:03<00:00, 2286.89it/s]
```

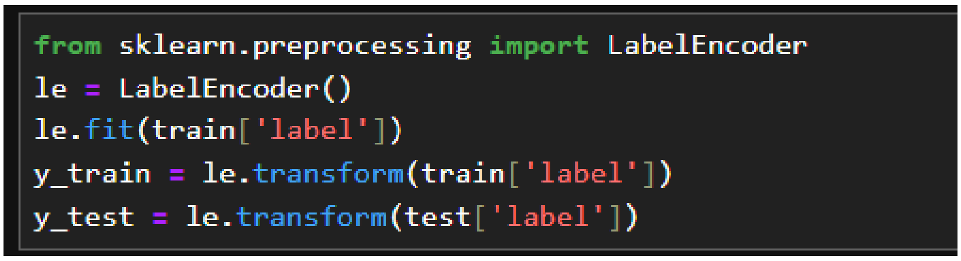
After the verification of images and its corresponding labels, the features were extracted from the images and stored in the form of numpy array. Here, load_img was used. It is a pre defined function in Tensorflow Keras. It is used to store images in the form of numpy arrays. After all the images were stored in the numpy array, it was reshaped into its length, 48x48 pixels height and width and 1 (colour attribute which is grayscale in this case).

The progress bar is generated using tqdm module in python which is used to display these bars for loops.



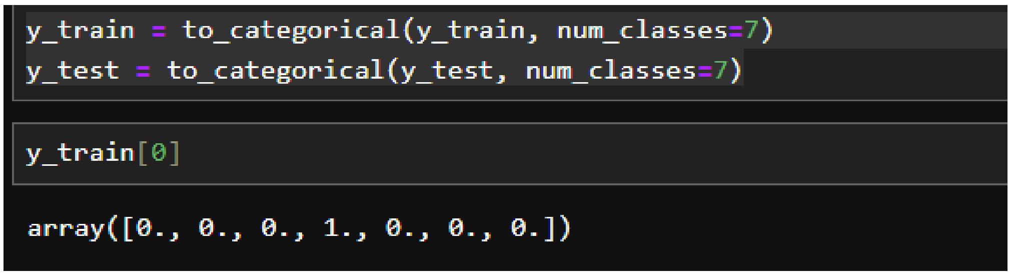
```
x_train = train_features/255.0
x_test = test_features/255.0
```

The features extracted are then normalized by dividing all the values by 255. It is usually done to RGB pixels or grayscale pixels. Without normalizing, the neural network will get too confused and tend to give wrong predictions. Normalizing the values will bring then between 0 and 1.



```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(train['label'])
y_train = le.transform(train['label'])
y_test = le.transform(test['label'])
```

Since a machine cannot understand words but numbers, it is crucial to convert all the labels (happy, sad, etc.) in the form of number (0,1,2,…). To achieve this task, Label Encoder from Sci Kit learn has been used.

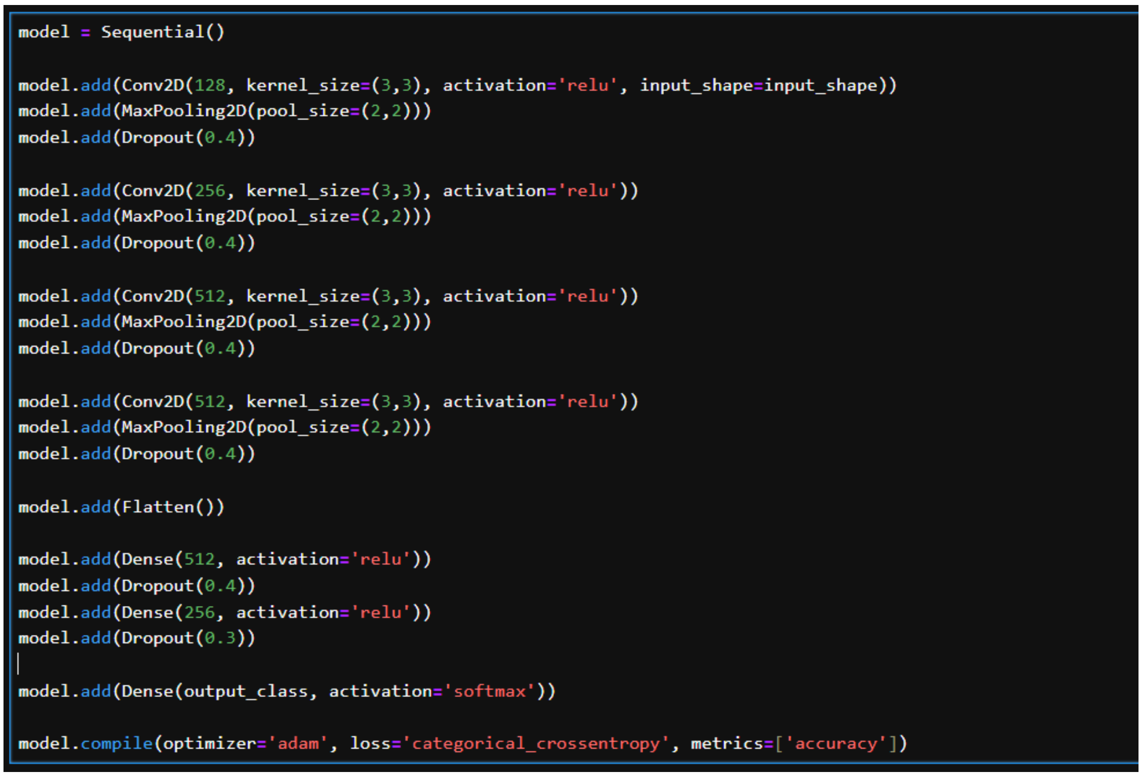


```
y_train = to_categorical(y_train, num_classes=7)
y_test = to_categorical(y_test, num_classes=7)
```

```
y_train[0]
```

```
array([0., 0., 0., 1., 0., 0., 0.])
```

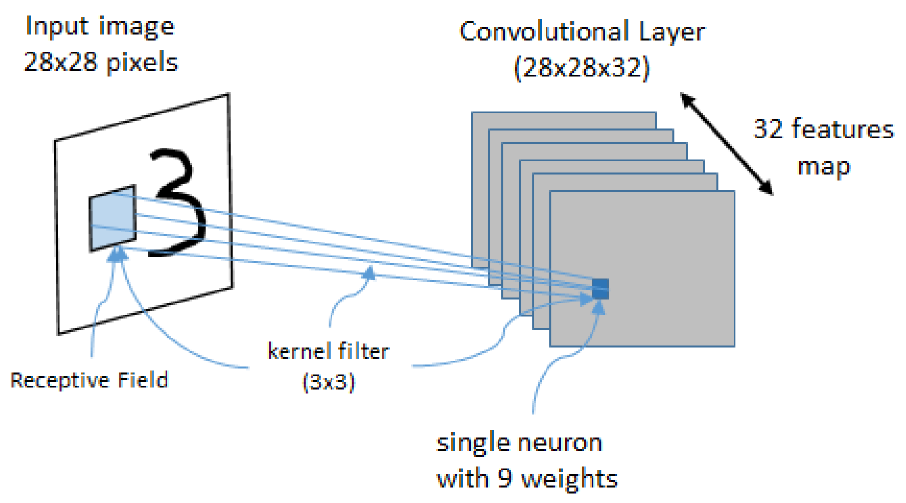
After the labels have been encoded, they are categorically stored in 7 classes. In the above snippet, it has been shown how the data structure of the y train list is. 1 represents the category it belongs to while 0 represents the false values.



```
model = Sequential()

model.add(Conv2D(128, kernel_size=(3,3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(256, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(512, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(512, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(output_class, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

The above snippet shows how the model is architectured. It comprises of 5 convolutional layers. In the first convolutional layer, there are 128 filters with a kernel size of 3x3. Each filter detects different patterns and features in the input data. This parameter specifies the size of the convolutional kernel (filter). The convolutional kernel is a small matrix that slides over the input data to perform the convolution operation. (3, 3) indicates that each kernel has a height and width of 3 pixels. Like this, there are 4 convolutional layers with 256, 512 and 512 filter layers with a kernel of size 3x3. ReLU stands for Rectified Linear Unit. It is an activation function commonly used in neural networks, particularly in deep learning models. It is a simple non-linear function that introduces non-linearity to the model, enabling it to learn complex patterns and relationships in the data. The Dropout function is used to randomly drop some of the neurons to avoid overfitting and get better accuracy of the results.

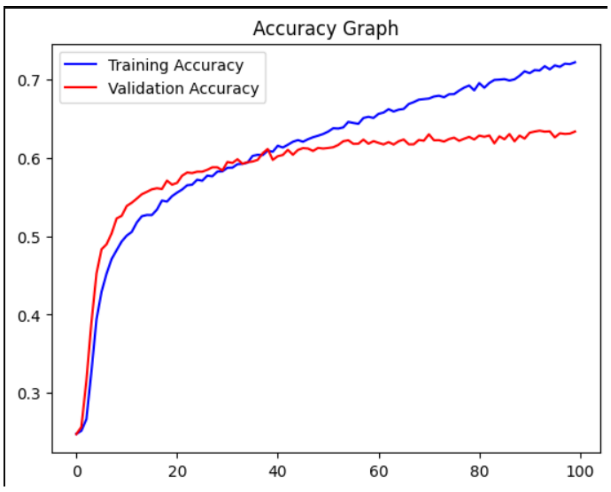
The Flatten layer is used to flatten the input data into a one-dimensional array. It takes the multi-dimensional input data, such as the output of convolutional layers, and reshapes it into a flat vector. This is necessary when transitioning from convolutional layers (which operate on multi-dimensional data) to fully connected layers (which require one-dimensional input). The Dense layer is a fully connected layer, also known as a densely connected layer. Each neuron in a dense layer is connected to every neuron in the previous layer.

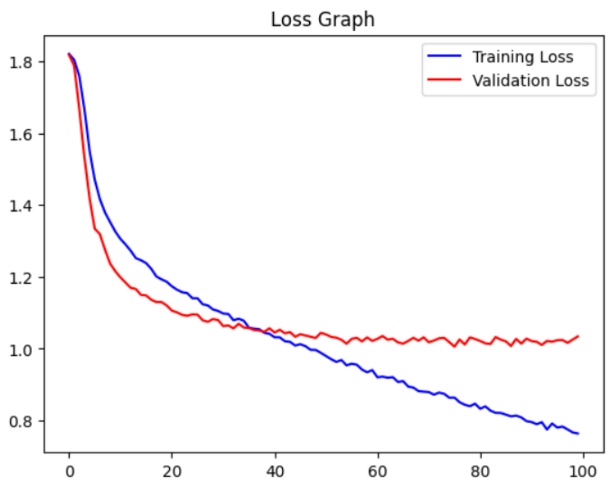


https://www.filepicker.io/api/file/J9SaSzrQ1gEiT4d001eQ

After the model was constructed, it was executed for 100 epochs with 128 as batch size. A total of 225*128 images were processed per epoch.

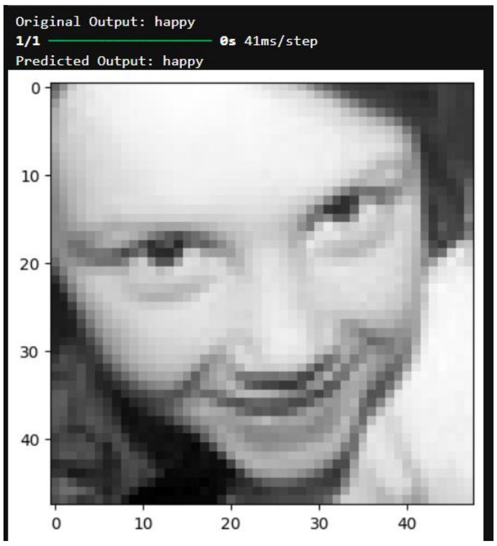
The model took around 5 hours to get trained and complete 100 epochs. After the model training was completed, the accuracy was calculated to be 0.713 and loss of 0.7677. The accuracy of first epoch was 0.2403 and it gradually increased to 0.713. To visualize these numbers, an accuracy and a loss graph was plotted and following was the result.



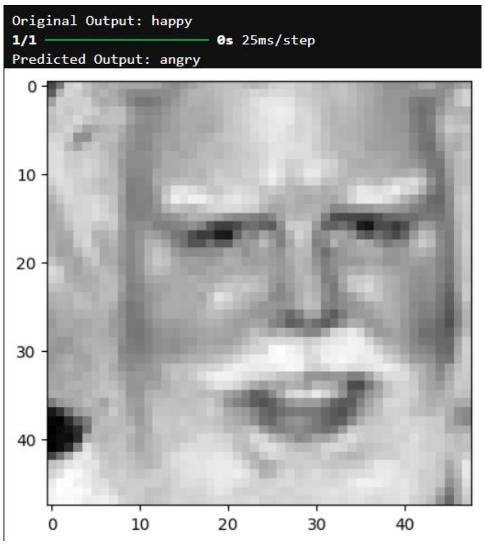


After visualizing the graphs, it can be concluded that the model tend to overfit and thus the results of the testing data was not up to the mark. To tackle this problem, the model can be trained further by increasing the epochs and dropping more neurons during the construction of the model. It can also be visualized that around 35th epoch, the graphs are intersecting each other.

To check the working of the model manually, some images were randomly loaded and predicted label and original label. The following were the results.



```
Original Output: happy
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Predicted Output: happy
```

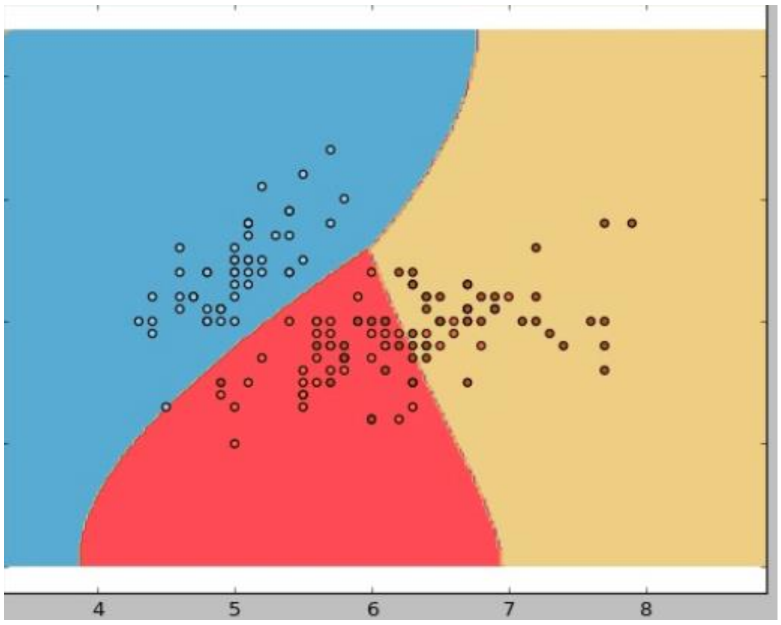


```
Original Output: happy
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
Predicted Output: angry
```

The first image was predicted correctly while the second image was predicted as “angry” but the original label was “happy.”

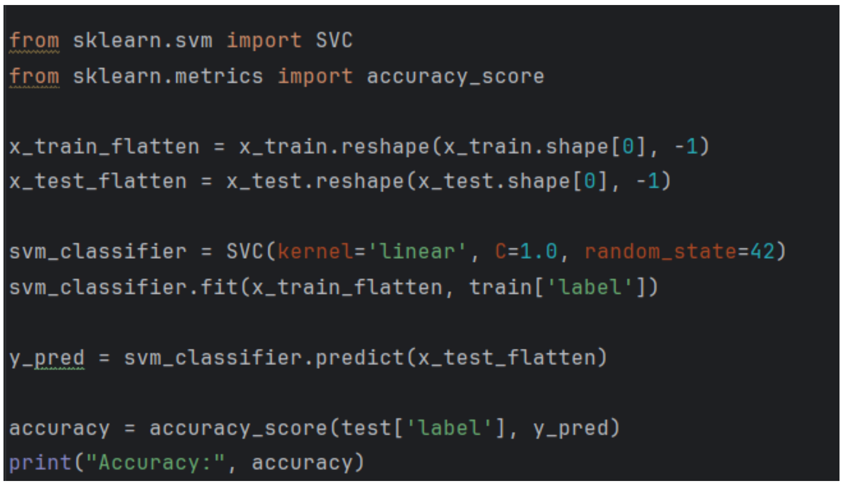
2. **Support Vector Classification:**

   In machine learning, SVC stands for Support Vector Classifier, which is a type of supervised learning model used for classification tasks. Specifically, SVC is a variant of Support Vector Machine (SVM) that is tailored for classification. SVC is widely used in various domains, including text classification, image classification, bioinformatics, and more. It is particularly effective for binary classification tasks and can be extended to handle multi-class classification using strategies such as one-vs-one or one-vs-rest.



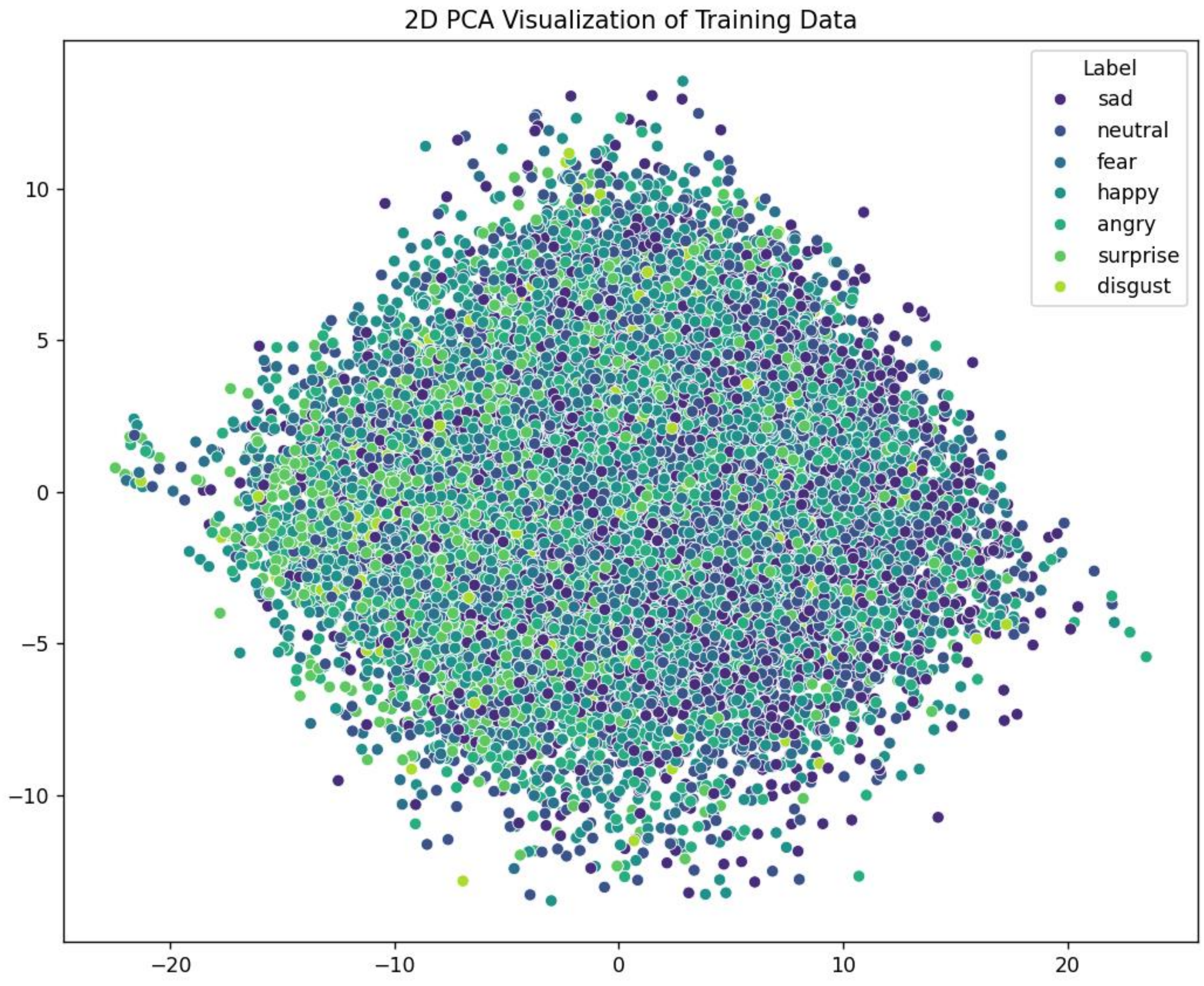
https://cdn.analyticsvidhya.com/wp-content/uploads/2024/01/image-157.png

The data was loaded and extracted in the same way as it was done for the CNN model.



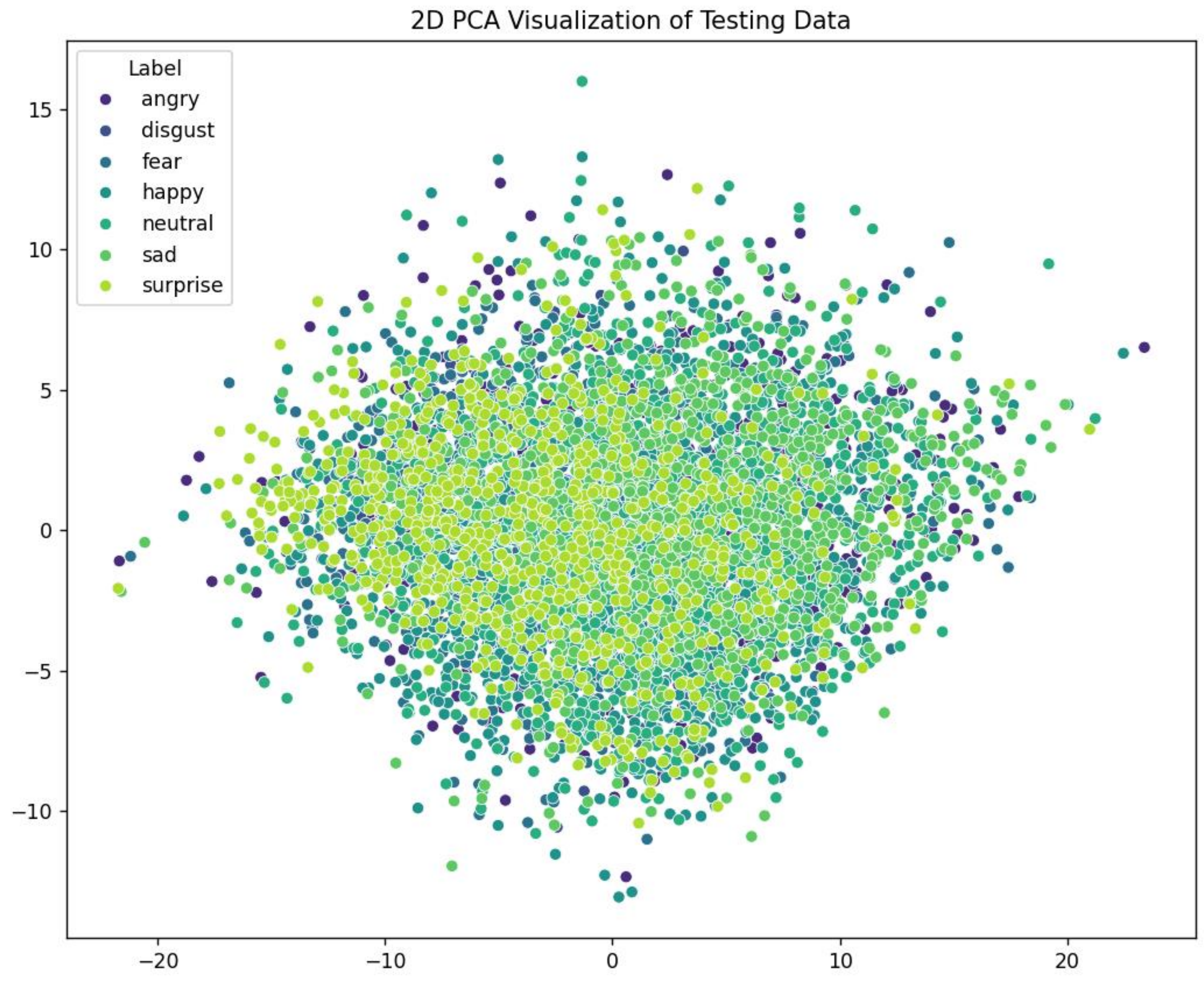
```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

x_train_flatten = x_train.reshape(x_train.shape[0], -1)
x_test_flatten = x_test.reshape(x_test.shape[0], -1)

svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
svm_classifier.fit(x_train_flatten, train['label'])

y_pred = svm_classifier.predict(x_test_flatten)

accuracy = accuracy_score(test['label'], y_pred)
print("Accuracy:", accuracy)
```

The accuracy of the model was coming out to be 0.3523. The reason behind such a low accuracy was analysed after visualizing the data points using matplotlib. The results were as follows:



As it can be seen that the training points are too clustered and thus the support vector margins cannot be drawn distinctly. Therefore, the learning procedure of the machine is not very productive

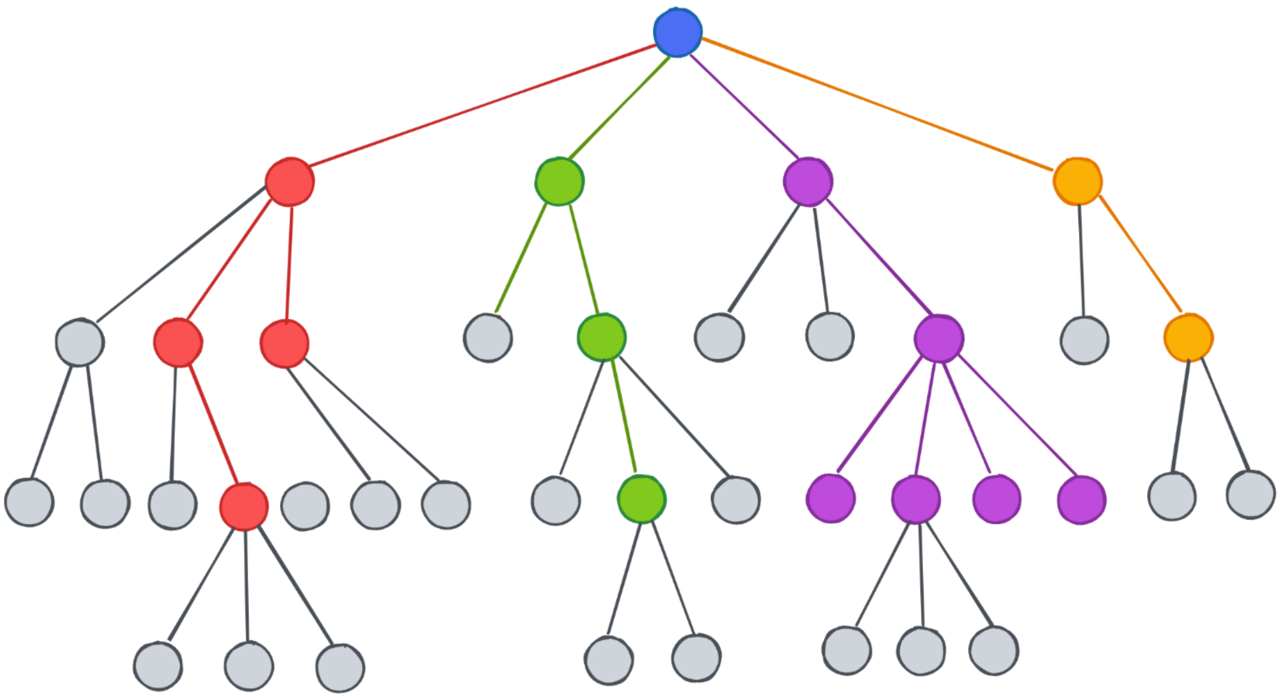
Moreover, when the points were plotted for testing data, they were too, very clustered. Since the model could not learn well from the training data, it ought to fail for the testing data.



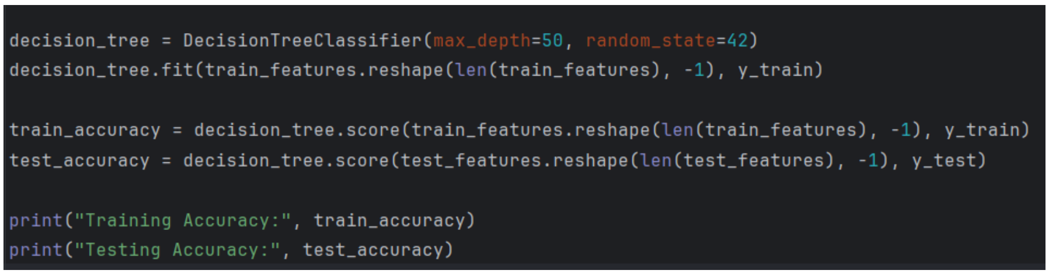
It can be concluded that SVC is not a good choice for this dataset as it performs very poorly.

3. **Decision Tree**

   A decision tree is one of the most powerful tools of supervised learning algorithms used for both classification and regression tasks. It builds a flowchart-like tree structure where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. It is constructed by recursively splitting the training data into subsets based on the values of the attributes until a stopping criterion is met, such as the maximum depth of the tree or the minimum number of samples required to split a node.



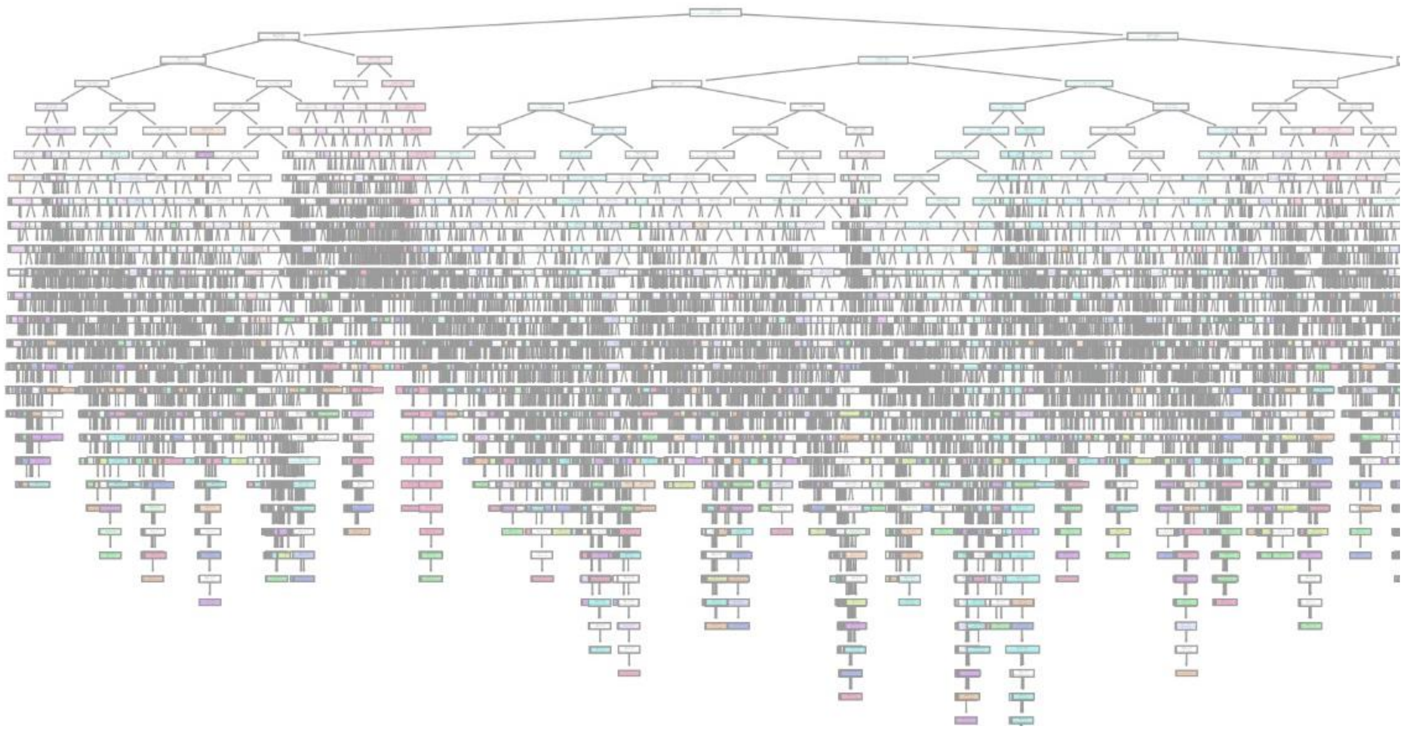
The steps for data extraction and loading were same as done in CNN classifier



```
decision_tree = DecisionTreeClassifier(max_depth=50, random_state=42)
decision_tree.fit(train_features.reshape(len(train_features), -1), y_train)

train_accuracy = decision_tree.score(train_features.reshape(len(train_features), -1), y_train)
test_accuracy = decision_tree.score(test_features.reshape(len(test_features), -1), y_test)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

The accuracy of the training data set is coming out to be 0.9985 while the accuracy of the testing data is coming out to be 0.3275. This shows that the model is overfitted and performing poorly for the test data.

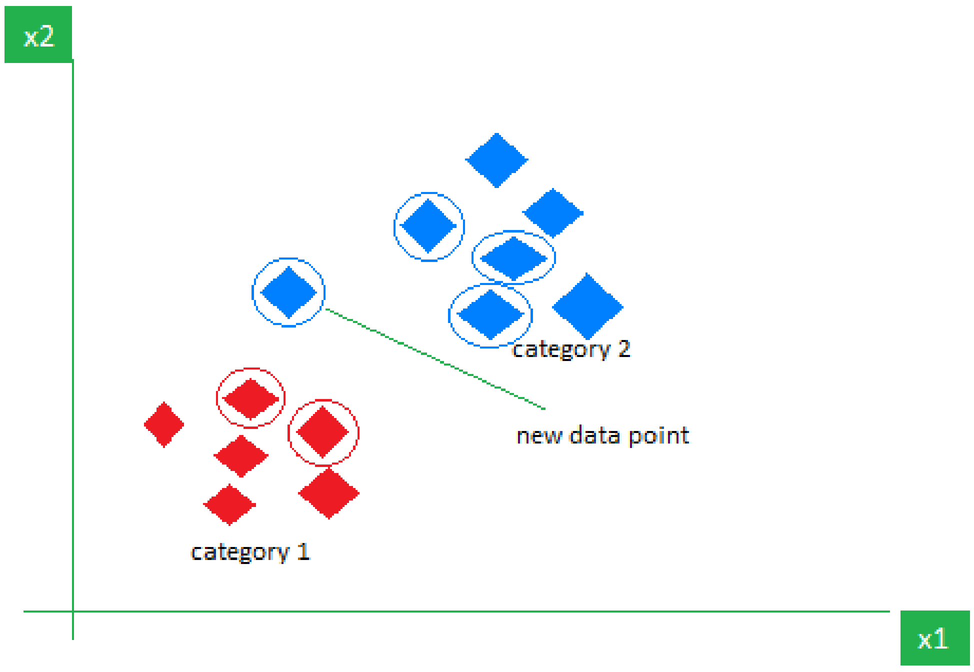
On visualizing the decision tree, it was not very clear what the issue was as the tree was too complex.



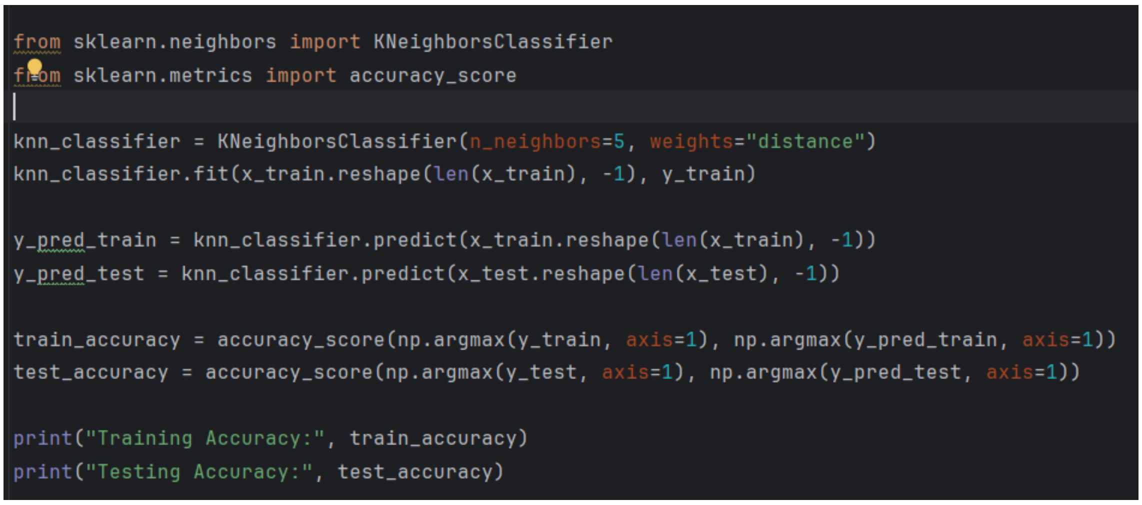
To conclude, decision tree tend to overfit the model and thus perform poorly for image classification. The max depth attribute was set to 10, 20 and 50 and out of these, the best accuracy coming out was by 50.

4. **K nearest neighbours**

   The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning method employed to tackle classification and regression problems. Evelyn Fix and Joseph Hodges developed this algorithm in 1951, which was subsequently expanded by Thomas Cover.



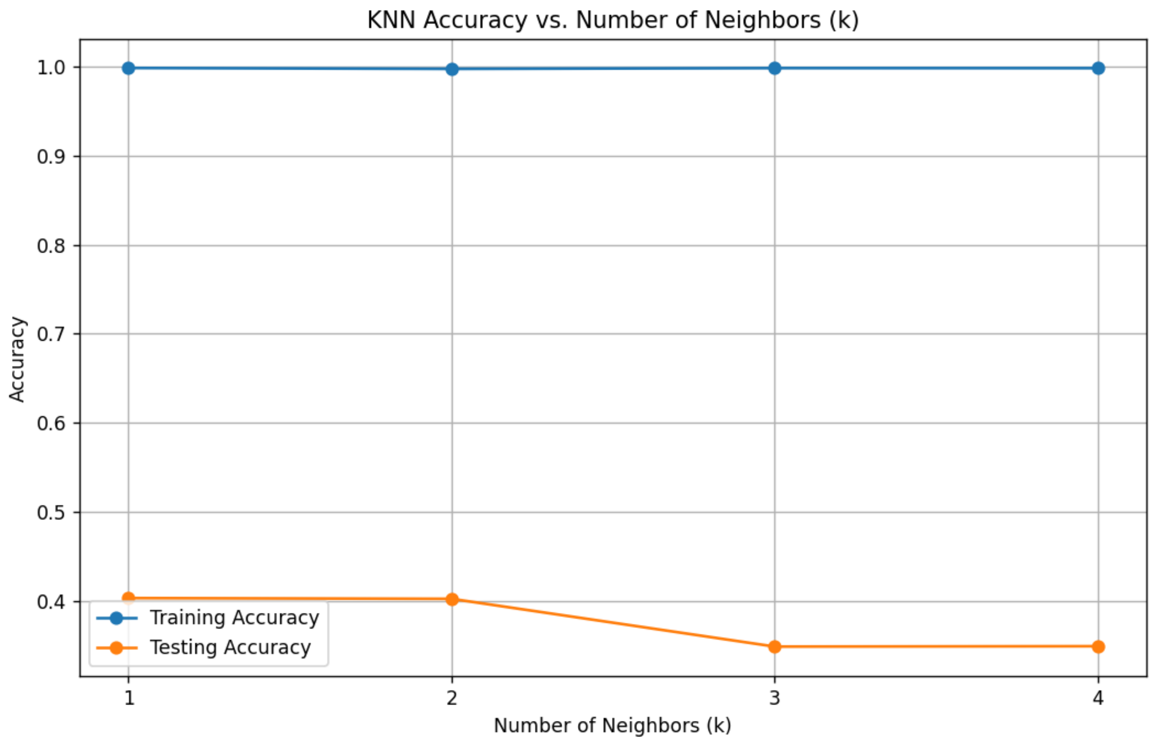
The method of extracting and loading the data is same as CNN classifier.



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn_classifier = KNeighborsClassifier(n_neighbors=5, weights="distance")
knn_classifier.fit(x_train.reshape(len(x_train), -1), y_train)

y_pred_train = knn_classifier.predict(x_train.reshape(len(x_train), -1))
y_pred_test = knn_classifier.predict(x_test.reshape(len(x_test), -1))

train_accuracy = accuracy_score(np.argmax(y_train, axis=1), np.argmax(y_pred_train, axis=1))
test_accuracy = accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_pred_test, axis=1))

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

The accuracy of the model for the training data set was coming out to be 0.998 while of the testing data was 0.315. This shows that K nearest model too, is tending to overfit the data.

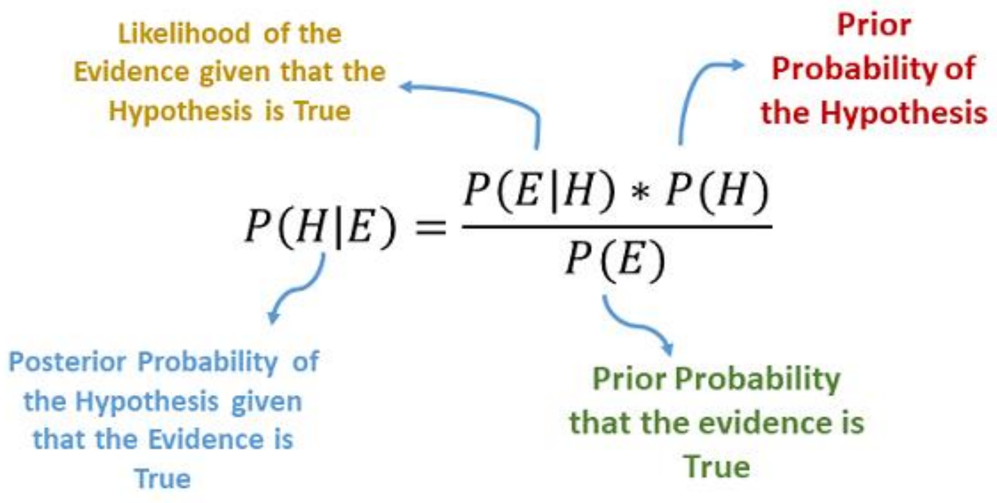
This can be understood by the graph previously attached in SVC classifier where the data points were too clustered. Almost all the neighbours were different and thus no correct prediction can be made out from it



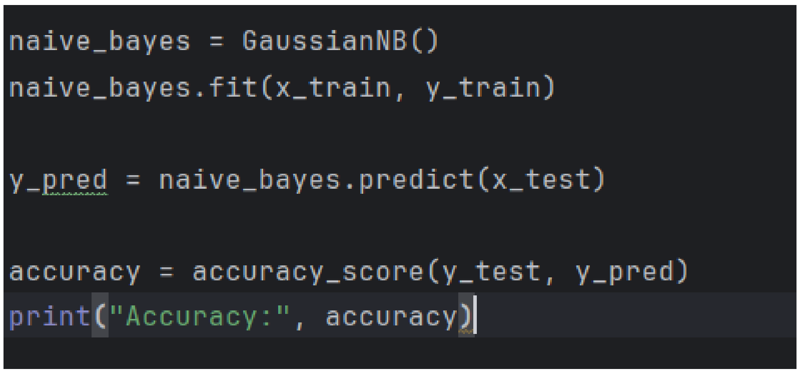
To conclude, K nearest is also failing to classify this data set and tending to overfit the model. One of the way to prevent overfitting is by penalizing the model but that has not been discussed in this report.

5. **Naïve Bayes Classifier.**

   Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset. One of the most simple and effective classification algorithms, the Naïve Bayes classifier aids in the rapid development of machine learning models with rapid prediction capabilities.
   Naïve Bayes algorithm is used for classification problems. It is highly used in text classification. In text classification tasks, data contains high dimension (as each word represent one feature in the data). It is used in spam filtering, sentiment detection, rating classification etc. The advantage of using naïve Bayes is its speed. It is fast and making prediction is easy with high dimension of data.


Likelihood of the Evidence given that the Hypothesis is True
Prior Probability of the Hypothesis
$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$
Posterior Probability of the Hypothesis given that the Evidence is True
Prior Probability that the evidence is True

The method of extracting and loading the data was same as in the CNN Classifier.



```
naive_bayes = GaussianNB()
naive_bayes.fit(x_train, y_train)

y_pred = naive_bayes.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Although the model was the fastest among all the models, the accuracy of the model was the least. The accuracy for the training data was 0.222 and testing data was 0.226. The model was not able to learn from the training data set and is tending to under fit.

Conclusion:

From this report, it can be concluded that CNN had the best accuracy rate among all the models.

Classifier	Accuracy on training data	Accuracy on testing data
Convolutional Neural Network	0.637	0.719
Support Vector Classification	0.5582	0.3523
Decision Tree	0.9985	0.3275
K Nearest Neighbours	0.998	0.315
Naïve Bayes Classifier	0.222	0.226

Future Work:

The first and foremost thing is to improve all the models and try to achieve as much accuracy as possible. There is a possibility that one model may be overpower the other in the future. The project, after completion can be deployed and be used to study facial recognition in judiciary, administration, etc. It also has potential to improve communication skills of people, especially of people with speaking disability.

References:

https://www.youtube.com/watch?v=PPeaRc-r1OI&list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw&index=15
https://www.kaggle.com/datasets/aadityasinghal/facial-expression-dataset

https://www.geeksforgeeks.org/naive-bayes-classifiers/

https://arxiv.org/pdf/1711.06303

https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py

https://www.tensorflow.org/guide/keras/sequential_model

https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/

https://www.geeksforgeeks.org/decision-tree/

https://www.youtube.com/watch?v=PHxYNGo8NcI&list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw&index=10

https://www.youtube.com/watch?v=aoCIoumbWQY

https://www.youtube.com/watch?v=CQveSaMyEwM&list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw&index=19

https://www.w3schools.com/python/pandas/pandas_intro.asp

https://medium.com/@nerdjock/convolutional-neural-network-lesson-4-filters-kernels-c64f11a18419#:~:text=A%20filter%2C%20or%20kernel%2C%20in,process%20is%20known%20as%20convolution.