CA-1
Sentiment Analysis of Facial Features using CNN | TensorFlow
<u>REPORT – *(GROUP-3)*</u>

Introduction:

With the advancements in machine and deep learning algorithms, the envision of various critical real-life applications in computer vision becomes possible. One of the applications is facial sentiment analysis.

Deep learning has made facial expression recognition the most trending research fields in computer vision area.

- Face recognition technology has widely attracted attention due to its enormous application value and market potential.
- It is being implemented in various fields like security system, digital video processing, and many such technological advances.
- Additionally, music is the form of art, which is known to have a greater connection with a person's emotion. It has got a unique ability to lift one's mood.

The human face is one of the most important parts of the anatomical body. It plays a great role in knowing the emotional status or the mood of the person. The mood of the person can be predicted, up to a certain accuracy, with the help of certain features visible on the face. With today's technologies, extracting or filtering the inputs can be done directly with the help of a webcam or an external device along with a few software.

This input can further be used for many applications and further studies. These technologies can be used to extract the moods of the user and those moods can be stored in a database. Relatively, this project focuses on building such a model that can recognise and differentiate between various facial expressions and is able to determine the emotion of user using Facial Recognition techniques.

Team Members:

Name	Roll no	Reg no	Role
Durga Mukesh	RKM047B64	11917727	**Exploratory Data Analysis, Creating and Training model**
Kartik Kumar Srivastava	RKM047B58	11913804	**Plotting accuracy and value loss graphs, Model testing and saving**

Problem Statement:

This project mainly focuses on creating a Facial Recognition system which can identify facial expressions of a sample images uploaded, based on the previously loaded datasets which are used for the model training.

This project can broadly be divided into:

1. Importing necessary Libraries and dataset.
2. Specifying train and test dataset paths.
3. Exploring the datasets: Finding out the number of images in each category of emotion in both the train and test datasets and plotting graphs and displaying random images of the same.
4. Creating Training/ Validation split using ImageGenerator.
5. Creating and training the model
6. Plotting the Accuracy and Value Loss graphs between the Validation and Training Datasets.
7. Testing the model accuracy and saving the model.

The final model is expected to be useful to detect facial expressions of people, when put up properly as a separate application or interface using Webcams or Cameras in general. This is the back-end analysis which can be used in general to detect emotions of people based on their facial expressions.

Libraries used:

1. NumPy
2. Pandas
3. Matplotlib Pyplot
4. OS
5. TensorFlow
6. Keras
7. Google Colab: Files
8. IO: BytesIO
9. PIL: Image

Integrated Development Environment (IDE) Used:

Google Colaboratory

Modules Proposed:

1. Convolutional Neural Networks – CNN
2. tensorflow.keras.models – Model

ABOUT PROJECT:

1) About Datasets:

This project has been done by a zip file containing train and test datasets, which further contain 7 different folders namely: Anger; Disgust; Fear; Happy; Neutral; Sad and Surprised, which contain various images showcasing each emotion.

The below code shows us how many images are there in each category of the train and test datasets:

```
def count_exp(path, set_):
    dict_ = {}
    for expression in os.listdir(path):
        dir_ = path + '/' + expression
        dict_[expression] = len(os.listdir(dir_))
    df = pd.DataFrame(dict_, index=[set_])
    return df
train_count = count_exp(train_dir, 'train')
test_count = count_exp(test_dir, 'test')
print(train_count)
print(test_count)
```

```
       disgust   sad  angry  happy  fear  surprise  neutral
train      436  4830   3995   7215  4097      3171     4965
      disgust   sad  angry  happy  fear  surprise  neutral
test      111  1247    958   1774  1024       831     1233
```

2) Project Steps:

1. Importing libraries and datasets:

   We have imported the following libraries:

   a. NumPy
   b. Pandas
   c. Matplotlib Pyplot - Visualisation
   d. OS
   e. TensorFlow
   f. Keras – Image Pre-processing, Validation Split and Model creation
   g. Google Colab: Files – Model Testing
   h. IO: BytesIO – Model Testing
   i. PIL: Image – Model Testing

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import os
    import tensorflow as tf
    from tensorflow.keras.preprocessing import image
    from keras.preprocessing.image import ImageDataGenerator, load_img
    from keras.layers import Conv2D, Dense, BatchNormalization, Activation, Dropout, MaxPooling2D, Flatten
    from keras import regularizers
    from keras.callbacks import ModelCheckpoint, CSVLogger, TensorBoard, EarlyStopping, ReduceLROnPlateau
    import datetime
    import warnings
    warnings.filterwarnings('ignore')
```

```
from google.colab import files
from io import BytesIO
from PIL import Image
```

We then have **Mounted** our google drive contents to our colab notebook to be able to access the zip file that contains the information for our project.

We then unzip the zipped file and load its contents into the google colab notebook.

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

    Mounted at /content/gdrive

[ ] # unziping the dataset file:
    !unzip gdrive/My\ Drive/CA-1/Dataset/archive.zip

    Streaming output truncated to the last 5000 lines.
      inflating: train/sad/Training_65242339.jpg
      inflating: train/sad/Training_65267116.jpg
      inflating: train/sad/Training_65275626.jpg
      inflating: train/sad/Training_6529266.jpg
      inflating: train/sad/Training_65329617.jpg
      inflating: train/sad/Training_65338712.jpg
      inflating: train/sad/Training_65338797.jpg
      inflating: train/sad/Training_65387162.jpg
      inflating: train/sad/Training_65404494.jpg
      inflating: train/sad/Training_65426218.jpg
      inflating: train/sad/Training_65430136.jpg
      inflating: train/sad/Training_65437377.jpg
      inflating: train/sad/Training_6545735.jpg
      inflating: train/sad/Training_65463385.jpg
      inflating: train/sad/Training_65473985.jpg
      inflating: train/sad/Training_65502829.jpg
      inflating: train/sad/Training_65505359.jpg
      inflating: train/sad/Training_65508578.jpg
      inflating: train/sad/Training_65516023.jpg
      inflating: train/sad/Training_65524027.jpg
      inflating: train/sad/Training_65526454.jpg
      inflating: train/sad/Training_65531175.jpg
      inflating: train/sad/Training_65552921.jpg
```

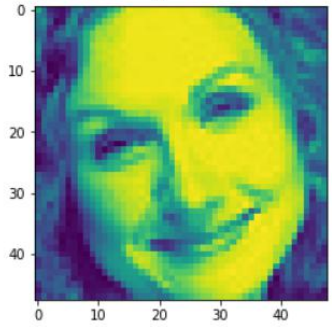
2. Specifying train and test dataset paths:

```
[ ] train_dir = 'train'
    test_dir = 'test'
```

Checking If the images have been loaded properly:

```
[ ] # Image viewing:
    img = image.load_img('train/happy/Training_10181727.jpg', target_size=(48,48), color_mode='grayscale')
    plt.imshow(img)
```

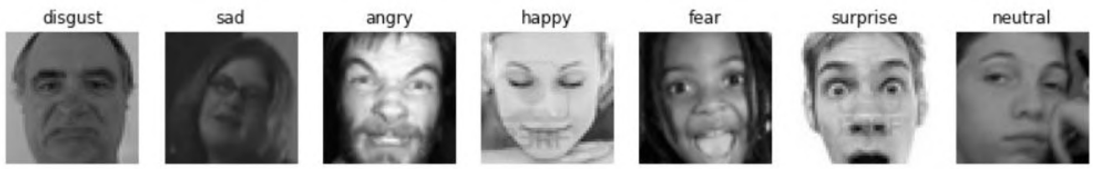
```
<matplotlib.image.AxesImage at 0x7f6108861890>
```



3. Exploring The Datasets:
    - The below table gives a brief understanding of the size of the two datasets and the contents they contain under each category of feeling:

	disgust	sad	angry	happy	fear	surprise	neutral
train	436	4830	3995	7215	4097	3171	4965
	disgust	sad	angry	happy	fear	surprise	neutral
test	111	1247	958	1774	1024	831	1233

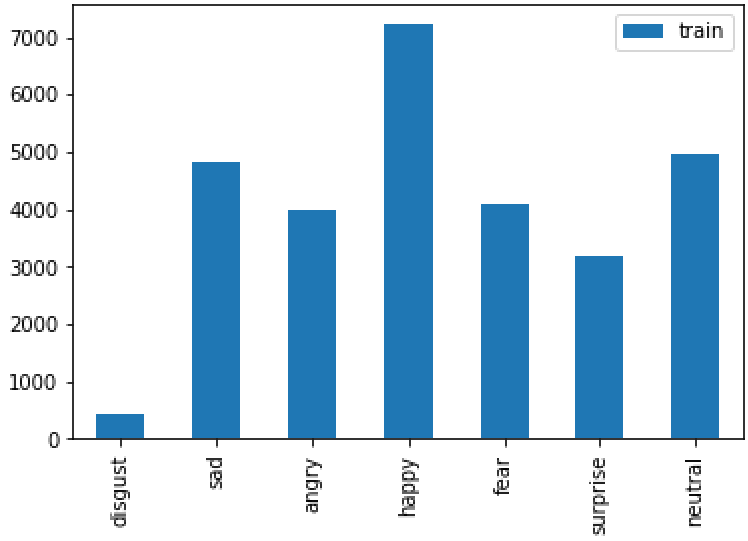
    - We then display random images from each category of feeling (one from each) to try to understand the contents of the datasets better:



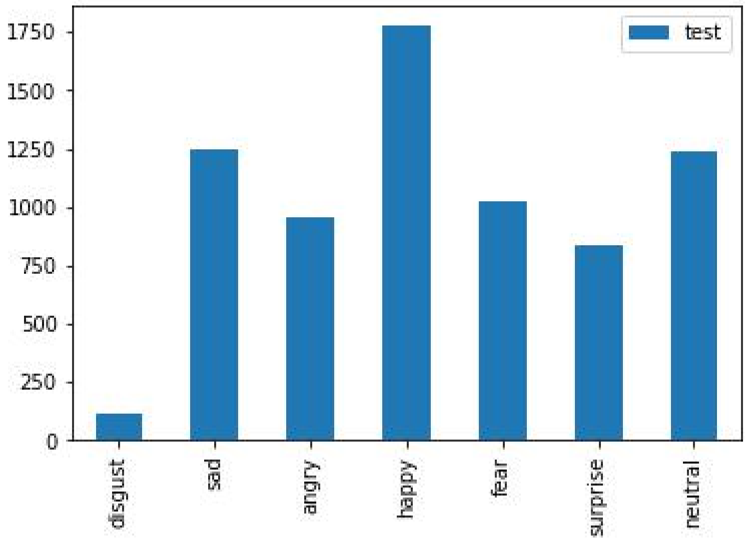
Train Dataset

Test Dataset

    - We then plot graphs of the number of images each category of the datasets consists of:



Here we can observe that the number of images in the **Train** dataset under the HAPPY category is the maximum, crossing 7000 images whereas the least number of images in the train dataset are in the DISGUST category, which don't even cross 500 images.



Similar graph can be observed for the **Test** dataset as well. The maximum images of the test dataset belong to the HAPPY category, crossing 1750 and the least number of images belong to the DISGUST category, not even crossing 250 images.

- We now display 14 images randomly from the Train and Test datasets respectively:

Train Dataset

Test Dataset

4. Validation Split:
   Validation split is a split of the dataset which allows us to verify how accurate our model is running. This split dataset consists of images segregated into different categories of emotions and allows us to

check/validate if our model can detect the emotions of the training dataset accurately or not.

```
[ ] from tensorflow.keras.preprocessing.image import ImageDataGenerator
    train_gen = ImageDataGenerator(rotation_range=20,
                                   rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1)

    training_data = train_gen.flow_from_directory(train_dir,
                                                  target_size=(224,224),
                                                  batch_size=64,
                                                  color_mode = "grayscale",
                                                  class_mode = "categorical")

    Found 28709 images belonging to 7 classes.
```

```
[ ] valid_gen = ImageDataGenerator(rescale=1./255)

    valid_data = valid_gen.flow_from_directory(test_dir,
                                               target_size=(224,224),
                                               batch_size=64,
                                               color_mode='grayscale',
                                               class_mode='categorical')

    Found 7178 images belonging to 7 classes.
```

```
[ ] from tensorflow.keras.applications.vgg19 import VGG19

    vgg = VGG19(weights='imagenet', include_top=False)

    for layer in vgg.layers:
        layer.trainable = False

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
    80142336/80134624 [==============================] - 1s 0us/step
    80150528/80134624 [==============================] - 1s 0us/step
```

```
[ ] training_data.class_indices

    {'angry': 0,
     'disgust': 1,
     'fear': 2,
     'happy': 3,
     'neutral': 4,
     'sad': 5,
     'surprise': 6}
```

5. Model Creation and Training:
   We create the model now and train it.

Model Creation:

```
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPool2D, Dropout, Flatten, Dense, Input
from tensorflow.keras.models import Model

input = Input(shape=(224,224,1))
#convolution layer
conv = Conv2D(3, kernel_size=(3,3), padding='same')(input)

vgg = vgg(conv)

x = Flatten()(vgg)

pred = Dense(7, activation='softmax')(x)

model = Model(inputs=input, outputs=pred)
```

Model Summary:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 1)]     0

 conv2d (Conv2D)             (None, 224, 224, 3)       30

 vgg19 (Functional)          (None, None, None, 512)   20024384

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 7)                 175623

=================================================================
Total params: 20,200,037
Trainable params: 175,653
Non-trainable params: 20,024,384
_________________________________________________________________
```

Model Compilation:

```
[ ] from tensorflow.keras.optimizers import Adam
    opt = Adam(learning_rate=0.0001)

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

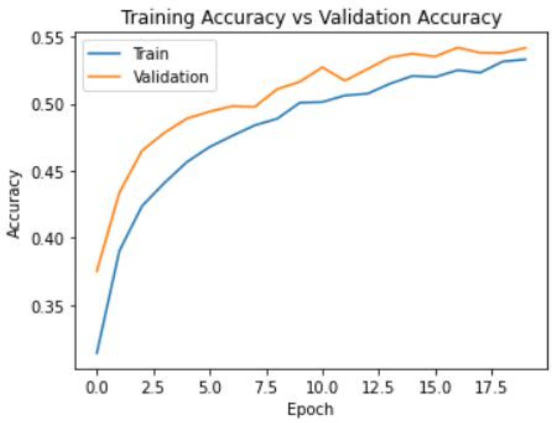
Model Training:

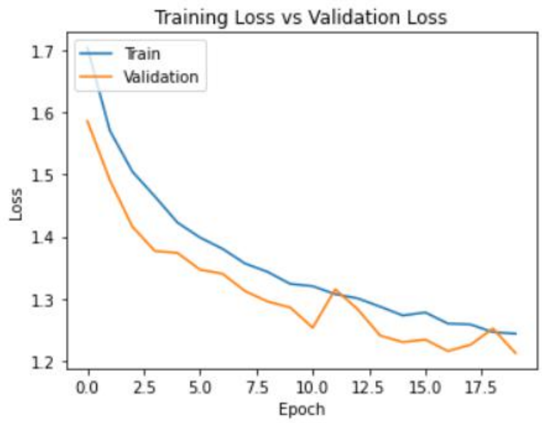
```
[ ] history = model.fit(training_data, epochs=20, validation_data = valid_data)
```

```
Epoch 1/20
449/449 [==============================] - 619s 1s/step - loss: 1.7045 - accuracy: 0.3143 - val_loss: 1.5860 - val_accuracy: 0.3753
Epoch 2/20
449/449 [==============================] - 553s 1s/step - loss: 1.5707 - accuracy: 0.3905 - val_loss: 1.4909 - val_accuracy: 0.4338
Epoch 3/20
449/449 [==============================] - 553s 1s/step - loss: 1.5047 - accuracy: 0.4237 - val_loss: 1.4159 - val_accuracy: 0.4649
Epoch 4/20
449/449 [==============================] - 554s 1s/step - loss: 1.4645 - accuracy: 0.4411 - val_loss: 1.3768 - val_accuracy: 0.4784
Epoch 5/20
449/449 [==============================] - 554s 1s/step - loss: 1.4226 - accuracy: 0.4567 - val_loss: 1.3737 - val_accuracy: 0.4890
Epoch 6/20
449/449 [==============================] - 554s 1s/step - loss: 1.3985 - accuracy: 0.4678 - val_loss: 1.3467 - val_accuracy: 0.4941
Epoch 7/20
449/449 [==============================] - 554s 1s/step - loss: 1.3803 - accuracy: 0.4761 - val_loss: 1.3402 - val_accuracy: 0.4982
Epoch 8/20
449/449 [==============================] - 554s 1s/step - loss: 1.3566 - accuracy: 0.4839 - val_loss: 1.3123 - val_accuracy: 0.4976
Epoch 9/20
449/449 [==============================] - 554s 1s/step - loss: 1.3432 - accuracy: 0.4888 - val_loss: 1.2954 - val_accuracy: 0.5109
Epoch 10/20
449/449 [==============================] - 554s 1s/step - loss: 1.3239 - accuracy: 0.5007 - val_loss: 1.2859 - val_accuracy: 0.5163
Epoch 11/20
449/449 [==============================] - 554s 1s/step - loss: 1.3202 - accuracy: 0.5012 - val_loss: 1.2532 - val_accuracy: 0.5272
Epoch 12/20
449/449 [==============================] - 554s 1s/step - loss: 1.3073 - accuracy: 0.5061 - val_loss: 1.3151 - val_accuracy: 0.5171
Epoch 13/20
449/449 [==============================] - 554s 1s/step - loss: 1.3005 - accuracy: 0.5074 - val_loss: 1.2829 - val_accuracy: 0.5258
Epoch 14/20
449/449 [==============================] - 554s 1s/step - loss: 1.2871 - accuracy: 0.5149 - val_loss: 1.2405 - val_accuracy: 0.5346
Epoch 15/20
449/449 [==============================] - 554s 1s/step - loss: 1.2729 - accuracy: 0.5207 - val_loss: 1.2301 - val_accuracy: 0.5372
Epoch 16/20
449/449 [==============================] - 555s 1s/step - loss: 1.2778 - accuracy: 0.5199 - val_loss: 1.2340 - val_accuracy: 0.5351
Epoch 17/20
449/449 [==============================] - 554s 1s/step - loss: 1.2600 - accuracy: 0.5250 - val_loss: 1.2156 - val_accuracy: 0.5418
Epoch 18/20
449/449 [==============================] - 555s 1s/step - loss: 1.2585 - accuracy: 0.5231 - val_loss: 1.2256 - val_accuracy: 0.5379
Epoch 19/20
449/449 [==============================] - 555s 1s/step - loss: 1.2462 - accuracy: 0.5314 - val_loss: 1.2519 - val_accuracy: 0.5378
Epoch 20/20
449/449 [==============================] - 553s 1s/step - loss: 1.2436 - accuracy: 0.5330 - val_loss: 1.2124 - val_accuracy: 0.5415
```

Note: The Epoch value determines the number of times we want to run the model to train it. The more we run it, the more the accuracy of the model. Here we have set our EPOCH value to 20- so our model ran 20 times to get trained.

6. Plotting the Accuracy and Value Loss graphs:
   We plot the graph between the Training Accuracy and Validation Accuracy, AND the Training Loss and Validation Loss.





The X axis represents the Epochs, and the Y axis represents the Accuracy and the Loss values in each graph.
We can observe from the Training accuracy Vs Validation Accuracy graph in the left, where the Blue line represents Train Accuracy, and the Yellow one represents the Validation Accuracy, that both the lines are relatively close. If they coincide, it means that our Model id of Maximum accuracy. By running our Model to more epochs, we can attain that accuracy.

Same goes with the Training Loss Vs Validation Loss graph in the right. By running the model more times, we can make sure the two lines coincide giving our model the perfect loss values as well.

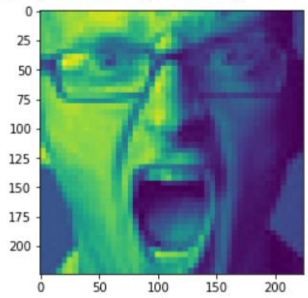
```
train_loss, train_acc = model.evaluate(training_data)
test_loss, test_acc = model.evaluate(valid_data)
```

```
449/449 [==============================] - 230s 511ms/step - loss: 1.2290 - accuracy: 0.5406
113/113 [==============================] - 58s 510ms/step - loss: 1.2124 - accuracy: 0.5415
```

7. Model Testing:
   We now load an image into our model and try to figure out the expression of the person in that image

```
[ ] test_img = image.load_img('test/angry/PrivateTest_13278552.jpg',target_size = (224,224),color_mode = "grayscale")
    plt.imshow(test_img)
```

```
<matplotlib.image.AxesImage at 0x7f6084994350>
```



Importing an image into the Model:

```
from google.colab import files
from io import BytesIO
from PIL import Image

uploaded = files.upload()
test_img = Image.open(BytesIO(uploaded['happy-810x540.png']))


plt.imshow(test_img)
plt.show()
```

```
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving happy-810x540.png to happy-810x540 (1).png
```

Code for Model usage on the Test Image uploaded:

```
[ ] label_dict = {0:'Angry',1:'Disgust',2:'Fear',3:'Happy',4:'Neutral',5:'Sad',6:'Surprise'}

    test_img = np.expand_dims(test_img,axis = 0)
    test_img = test_img.reshape(35,224,224,1)
    result = model.predict(test_img)
    result = list(result[0])

    img_index = result.index(max(result))
    print(label_dict[img_index])
```

This code gives us the output as **HAPPY** based on the previous analyses and model training of the Training model with the sample dataset Images. The Label_dict gives us a means to assign each expression name to a "key". We then expand the loaded image in terms of dimensions, reshape it and run the model on the test image. We store the result of the model prediction in a variable named RESULT. The index of this result variable will be stored in another variable named img_index, which consists of an Index value that matches with one of the Key values of the label_dict. We then print the Category of the image based on this Key value by printing the VALUE of the KEY stored in the dictionary and hence, get the category of the Test Image uploaded.

REFERENCES:

Datasets: https://opendatacommons.org/licenses/dbcl/1-0/
Google-Colab Link: https://colab.research.google.com/drive/1Mug1FcpI8RYsVDU2aW45-YfRk_AZZX9S?usp=sharing

THANK YOU