Realtime Gender and Emotion detection using Convolutional Neural Networks

Course Project Report

EE769
Introduction to Machine Learning

By,
Jainil Shah (193100071)
Vivek P Revi (193100076)

Indian Institute of Technology, Bombay

June 2020

Contents

Goal:

To create a Convolutional Neural Network based model which will be able to identify the gender and emotion of a person in real time.

Description:

For achieving our goal, we need to create two separate models. One which will be trained to classify faces based on gender and the other one which will be able to detect emotion of the face. Both these models will be combined to give a real-time detection and prediction of gender and emotion. We have used Tensorflow with Keras for our model training and OpenCV for real-time face detection and prediction.

Dataset Information:

As we need to create two models, One for gender and the other for emotion. We required two datasets. The data set chosen for this was FER-2013 for Emotion prediction and UTK-Face Dataset for gender prediction.

Facial Emotion Recognition Dataset (FER-2013):

This dataset was taken from a kaggle competition. It consists of 35,685 grayscale images with resolution 48x48 pixel of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. It is classified based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

The dataset has a csv file, which has three columns. One is “lablel” which specify the emotion corresponding to the image. Second one is “pixels” which has the pixel values of the 48x48 grayscale image. The third one is “usage” which corresponds to Training or Testing.

UTK-Face Dataset:

This dataset was taken from AICIP (Advanced Imaging and Collaborative Information Processing) web page. UTKFace dataset is a large-scale face dataset with a long age span (range from 0 to 116 years old). The dataset consists of 21,800 RGB face images of resolution 200x200 with annotations of age, gender, and ethnicity. The images cover large variation in pose, facial expression, illumination, occlusion, resolution, etc. This dataset could be used on a variety of tasks, e.g., face detection, age estimation, age progression/regression, landmark localization, etc. The key features of the dataset are:

- The images are aligned and cropped
- Images are labeled as “[age]_[gender]_[race]_[date&time].jpg”
- “[age]” is an integer from 0 to 116, indicating the age
- “[gender]” is either 0 (male) or 1 (female)

- “[race]” is an integer from 0 to 4, denoting White, Black, Asian, Indian, and Others (like Hispanic, Latino, Middle Eastern).
- “[date&time]” is in the format of yyyymmddHHMMSSFFF, showing the date and time an image was collected to UTKFace

Preparation of Dataset for Model Creation:

To create and train our model in CNN, we need the image data in a specific format. The data needs to be categorized into folders with names “test, train and valid”, to represent use cases of Testing,Training and Validation respectively with each of them containing folders which represent the classes to which images belong.

Preparing FER-2013:

The target is to read the csv file, and classify the data in the csv files according to the use case and labels into the specified folder format mentioned above.

```
def unique_name(pardir,prefix,suffix='jpg'):
    filename = '{0}_{1}.{2}'.format(prefix,random.randint(1,10**8),suffix)
    filepath = os.path.join(pardir,filename)
    if not os.path.exists(filepath):
        return filepath
```

The above function determines the path to which the image file needs to be saved. We need to pass the parent directory and prefix, which is “training” or “testing”. This will return the file location to save the image.

```
if __name__ == '__main__':
    filename = 'fer2013.csv'
    filename = os.path.join(curdir,filename)
    gen_record(filename,1)
```

This is a code execution function, which executes the code for extracting and formatting the images to specified directory.

```
def gen_record(csvfile,channel):
    data = pd.read_csv(csvfile,delimiter=',',dtype='a')
    labels = np.array(data['emotion'],np.float)

    imagebuffer = np.array(data['pixels'])
    images = np.array([np.fromstring(image,np.uint8,sep=' ') for image in imagebuffer])
    del imagebuffer
    num_shape = int(np.sqrt(images.shape[-1]))
    images.shape = (images.shape[0],num_shape,num_shape)
    dirs = set(data['Usage'])
    subdirs = set(labels)
    class_dir = {}
    for dr in dirs:
        dest = os.path.join(curdir,dr)
        class_dir[dr] = dest
        if not os.path.exists(dest):
            os.mkdir(dest)

    data = zip(labels,images,data['Usage'])

    for d in data:
        destdir = os.path.join(class_dir[d[-1]],str(int(d[0])))
        if not os.path.exists(destdir):
            os.mkdir(destdir)
        img = d[1]
        filepath = unique_name(destdir,d[-1])
        print('[^_^] Write image to %s' % filepath)
        if not filepath:
            continue
        sig = cv2.imwrite(filepath,img)
        if not sig:
            print('Error')
            exit(-1)
```

The above function is responsible for reading the csv file, extracting the image pixel data and transforming it into actual image and rearranging the extracted image into the specific location.

Preparing UTK-Face:

Here since all the images are put together into a single folder in this dataset, we need to format them into specific folders according to use case.

```
os.chdir(r'C:\Users\VR94\Desktop\Python Files\DATA\UTKFace\Data')
if os.path.isdir('male/') is False:
    os.mkdir('male')
    os.mkdir('female')
```

The above code checks weather the directory for “male” and “female” classes exist, if not it creates those directory.

Next we need to read the filenames of all images in our dataset, as it is the filename which contains information to which class the image belongs. According to the extracted file name, we can classify the images to respective folders. The code below saves all the filename into the array “file_name”

```
os.chdir(r"C:\Users\VR94\Desktop\Python Files\DATA\UTKFace\Data")

file_name = []

for file in glob.glob("*.jpg"):
    file_name.append(file)
```

Now we need to read the file name and extract the information about gender from it. Then we move the file into the desired location. The code below does exactly that.

```
for name in file_name:
    split = name.split('_')
    if split[1] == '0':
        shutil.move(f'{name}', 'male')
    else:
        shutil.move(f'{name}', 'female')
```

Emotion Prediction - Model Training:

For our model creation and training we have used Keras library with TensorFlow as backend. We will be doing our training with CUDA-GPU on our laptop device. The versions we are running are:

Python - 3.7
TensorFlow 2.0
CudaToolkit - 9.0
Keras - 2.3.1

```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.optimizers import Adam, RMSprop
from keras.regularizers import l2
from keras.layers.normalization import BatchNormalization
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score as asc
import matplotlib.pyplot as plt
%matplotlib inline
```

Shown above are the libraries that are used for our model and training of image dataset.

Initially we specify the location of the data for Training, Testing and Validation.

```
train_path = 'FER2013-Data_Extracted_7_emo/train'
valid_path = 'FER2013-Data_Extracted_7_emo/valid'
test_path = 'FER2013-Data_Extracted_7_emo/test'
```

```
train_batches = ImageDataGenerator(rescale=1./255, horizontal_flip=True).flow_from_directory(
    directory=train_path, target_size=(48,48), color_mode = 'grayscale', batch_size=70)
valid_batches = ImageDataGenerator(rescale=1./255).flow_from_directory(
    directory=valid_path, target_size=(48,48), color_mode = 'grayscale', batch_size=50)
test_batches = ImageDataGenerator(rescale=1./255).flow_from_directory(
    directory=test_path, target_size=(48,48), color_mode = 'grayscale', batch_size=50, shuffle=False)
```

```
Found 28700 images belonging to 7 classes.
Found 3500 images belonging to 7 classes.
Found 3500 images belonging to 7 classes.
```

ImageDataGenerator is a module from keras.preprocessing library which allows us to load image data from a directory. Previously we had arranged our images into the following splits shown above. For Training we have 28700 images and for Validation and Testing we have 3500 images each. The images belong to 7 classes of emotion. We are using a normalization parameter for the pixel values, typically the brightness in an image is represented from a range of 0 to 255, 0 being black and 255 being white. This is normalized into a range of 0 to 1, this gives the network more stability for convergence. Horizontal flip for training set is enabled, which is a data augmentation technique for better generalization. This randomly flips some of the images horizontally. We have also specified the size of the image (48x48), color mode and batch size. As the image in the dataset is grayscale, we have specified it here as well. We have given a batch size of 70 for training and 50 for testing and validation. Generally higher batch size results in faster training, and less generalization. It is also limited by the hardware specifications. We found the aforementioned batch sizes good for our model.

CNN Models Tried (Experiments and Observation):

Before getting the best ML model for this dataset, we had tried several model architectures, parameters and training cycles. This enabled us to draw inferences regarding what model architecture and the effect of various parameters involved.

Model 1: Sequential model with 6 Conv Layers

A basic CNN architecture was used, starting from larger filter size 5x5 for the first two layers and decreasing the filter size to 3x3 in the subsequent layers at the same time increasing the number of filters to extract the small features in the image data. Relu activation was used in every layers, and the output from the Conv layers was passed to a shallow dense layer with just 2 fully connected layers. A dropout feature was utilized in the fully connected layers to make some generalizations and reduce overfit.

```
input_shape = (48,48,1)
model.add(Conv2D(64, (5, 5), input_shape=input_shape,activation='relu', padding='same'))
model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (5, 5),activation='relu',padding='same'))
model.add(Conv2D(128, (5, 5),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(7))
model.add(Activation('softmax'))
```

The image above shows the structure of the model implemented. The model thus obtained consisted of 2,785,863 Trainable parameters and 1,152 Non-trainable parameters.

Several test runs were performed on this by varying the number of epochs and learning rate. Studying the performance of the obtained model and testing it with the real time prediction program, we were able to get insights into how to improve the model. The observations were.

- The model was severely overfitting, this was evident from the above 90% training accuracy and the validation and testing accuracy values were only close to 50%.
- Even though the training loss was showing a good trend by starting at a higher value in the beginning and showing a steady decrease, the validation loss was showing large fluctuations even from the beginning. This also shows the overfitting and maybe higher learning rates.
- The confusion matrix obtained was not good enough, in some of the test runs the model showing bias. This was evident from getting good accuracy for some classes and poor accuracy for others.
- In the real time prediction, the model was giving good predictions for easily identifiable emotions such as Happy, Angry and Surprise but did poorly on others.

Model 2: Transfer Learning with MobileNet

Transfer learning imports the model architecture from a well known and better performing model along with the weights. A selected number of layers in this model is then trained with our dataset along with modification to the output layer to obtain new weights and predict for our use case. Here we have used a popular ML model called MobileNet, which is capable of classifying 1000 classes. The model is similar to a very popular ML model called VGG16, also capable of classifying 1000 classes. However, MobileNet is optimized to be able to work on mobile devices with very less hardware requirement. This was achieved by optimizing and reducing the number of parameters in the model. MobileNet has 4.24 million parameters compared to 138 million in VGG16. This means MobileNet can give predictions very fast. However, this comes with a slight reduction in accuracy than VGG16 gives. MobileNet achieves 89% accuracy in ImageNet. The size of this model is also less with 16MB. This makes it very convenient for our application for real time prediction where speed of prediction is also important.

```
mobile = keras.applications.mobilenet.MobileNet()
mobile.summary()
```

```
x = mobile.layers[-6].output #dropping last 6 layers
predictions = Dense(7, activation='softmax')(x) # set output layer as 7 class
model = Model(inputs=mobile.input, outputs=predictions)
```

```
for layer in model.layers[:-23]: #here we are training the last 23 layers, this is a hyperparameter (to play around)
    layer.trainable = False
```

The model was imported from keras library, the output layer was set to a 6 class predictiction with softmax activation. The layers of the network except the last 23 were set to trainable. This number is a hyperparameter and several other numbers were tried to analyse the results. The total number of trainable parameters are 2,401,799 and non trainable parameters are 834,240, in this case.

Several test runs were performed on this by varying the number of epochs and learning rate. Studying the performance of the obtained model and testing it with the realtime prediction program, we were able to get insights.The observations were.

- The model was showing good training accuracy, but the validation accuracy was even worse than the previous models.
- The loss function was showing better trends than the previous models. This was because MobileNet was using techniques for generalizing the data.
- The testing results were not good, giving an accuracy of only 40%.
- The results were poor. MobileNet was not designed to classify human faces. The strength of MobileNet is in classifying animals and objects. So it is not good at extracting human facial emotion features.

Model 3: Trying 5 class predictions

The dataset that we are using was not balanced. There are more images for popular emotions such as Happy, Angry, Neutral and Surprised. The other emotions such as Disgust, Sad and Fear were having less number of images. This results in a bias towards the classes with more data. We decided to overcome this by reducing the number of classes into 5, namely Angry, Sad, Happy, Surprise and Neutral. As emotions such as Fear is closely similar to Surprise and Angry, also Disgust is closely related to Sad. We manually sorted these images to the new 5 class dataset to make it a better balanced model, with good number of datapoints in each class.

```
test_batches.class_indices
```

```
{'0': 0, '1': 1, '2': 2, '3': 3, '4': 4}
```

(0=Angry, 1=Happy, 2=Sad, 3=Surprise, 4=Neutral)

The above labels show each class that will be used, we have dropped Disgust and Fear. The previously experimented models were implemented in this to analyse the results. The observations were.

- The accuracy value was improved, with the highest accuracy obtained with a 5 Cov layer model, running for about 30 epochs giving an accuracy of 66% in the best case.
- The obtained accuracy was not high as we expected. There was only a 10% improvement even after reducing the classes from 7 to 5.
- Transfer learning was also tried with MobileNet with this dataset, and it was also not performing well enough.

So, considering all this, we decided to stick with the original 7 class dataset and try to make it better combining what we learned from all the above experimentation.

Finalized Model:

According to the inferences obtained from testing various model architectures and methods. We have decided to use a 3 Conv-Layer network with Batch Normalization to reduce the imbalance of weights along with Max Pooling, Drop Outs and Regularization. The first two Conv layers use a larger filter size of 5x5, with 32 filters in the first layer and 64 filters in the second. This is to recognize various features in the image appropriately. ‘Relu’ activation is used in both the cases with padding as ‘same’ which maintains the image size across layers. Batch normalization and Max pooling is also used for normalizing the weights and for keeping only prominent features. The 3rd Conv layer uses a smaller 3x3 filter size with 128 filters. This increases the facial feature extraction from the previous max pooled layer. Drop outs have been added in this layer for better generalization and reduce overfitting.

```
model = Sequential()

input_shape = (48,48,1)
model.add(Conv2D(32, (5, 5), input_shape=input_shape, activation='relu', padding='same'))
model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),activation='relu',padding='same'))
model.add(Dropout(rate = 0.1))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(units = 1024, activation = 'relu', kernel_regularizer = l2(0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(units = 512, activation = 'relu', kernel_regularizer = l2(0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(units = 256, activation = 'relu', kernel_regularizer = l2(0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(7))
model.add(Activation('softmax'))
```

After the Conv-Layers, we flatten the layers to pass it through a dense fully connected layer. As this is a facial recognition network, there are various minute features that distinguish the facial expression. So, we need a large fully connected layer to classify such small features. A 3 layered fully connected network is used for this purpose. The first dense layer has 1024 nodes which is followed by a 512 node and 256 node layer before passing to the final 7 class output layer. Each of the 3 layers has an activation of ‘relu’ along with a regularization parameter and dropout feature. This helps to reduce overfitting which was observed to be occurring in the previous trial runs. Batch Normalization is also applied to normalize the weights in the network and reduce imbalance. The output layer is a softmax activation layer with 7 classes, each for predicting the 7 emotions that constitutes the dataset.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 32)        832
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 64)        51264
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 128)       73856
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 128)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18432)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              18875392
_________________________________________________________________
batch_normalization_3 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 7)                 1799
_________________________________________________________________
activation_1 (Activation)    (None, 7)                 0
=================================================================
Total params: 19,667,207
Trainable params: 19,663,239
Non-trainable params: 3,968
_________________________________________________________________
```

The list above shows the information about each layer in the network. In total we can see there are 19,667,207 parameters in which 19,663,239 are trainable.

```
model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Now we compile our model with the specified parameters. We have used Categorical Cross Entropy as our desired loss function with Adam as the optimizer and a constant learning rate as 0.0001.

```
%%time
H = model.fit_generator(generator=train_batches, steps_per_epoch=410,
    validation_data=valid_batches, validation_steps=70, epochs=50, verbose=2)
```

Now we do the actual training, with model.fit function. We specify the training and validation batches. The steps needed per epoch is obtained by dividing the total number of images in the set with the number of images per batch in the specified set.

Few Trial Runs:

- **Test 1:**
  epochs = 30
  lr = 0.0001
  trainable parameters = 19,663,239
  non trainable parameters = 3,968

  Epoch 30/30
  - 255s - loss: 0.9182 - accuracy: 0.7721 - val_loss: 2.1123 - val_accuracy: 0.5331

  Test Accuracy: 0.5124

- ***Test 2:***
  epochs = 20
  lr = 0.0001
  trainable parameters = 19,663,239
  non trainable parameters = 3,968

  Epoch 20/20 - 258s - loss: 1.1713 - accuracy: 0.6769 - val_loss: 1.7387 - val_accuracy: 0.5451

  Test Accuracy: 0.5482

- ***Test 3:***
  epochs = 50
  lr = 0.0001
  trainable parameters = 19,663,239
  non trainable parameters = 3,968

  Epoch 50/50 - 259s - loss: 0.4964 - accuracy: 0.9440 - val_loss: 2.4844 - val_accuracy: 0.5643

  Test Accuracy: 0.582

The Model and Results from Test 3 was used, after 50 epochs the results obtained are:
Epoch 50/50 - 259s - loss: 0.4964 - accuracy: 0.9440 - val_loss: 2.4844 - val_accuracy: 0.5643

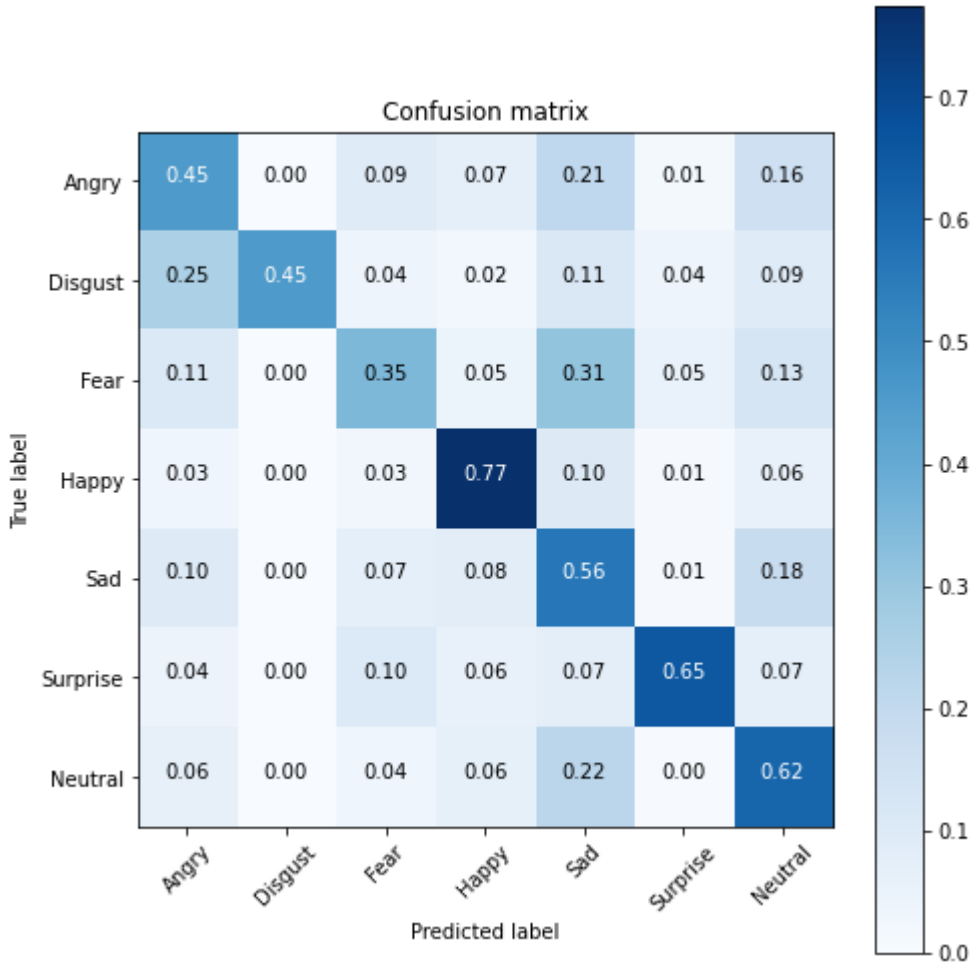
From the above result we can see that the model achieves a validation accuracy of around 56% and the training loss attains almost constant value after 35 epochs. However the validation loss shows an increasing trend as the number of epochs increases with fluctuations.

```
cm = confusion_matrix(y_true=test_labels, y_pred=predictions.argmax(axis=1))
print(f"Confusion Matrix:\n{cm}")
accuracy = asc(y_true=test_labels, y_pred=predictions.argmax(axis=1))
print(f"\nAccuracy:{accuracy}")
```

Test Accuracy: 0.582

```
Confusion Matrix:
[[223   0  46  34 102   6  80]
 [ 14  25   2   1   6   2   5]
 [ 56   1 186  24 164  28  69]
 [ 24   0  21 612  79   7  47]
 [ 57   1  40  48 334   5 109]
 [ 18   1  43  23  31 272  28]
 [ 39   0  22  40 138   2 385]]
```

We are getting a test accuracy of 58.2% with this model. The confusion matrix shows how the model predicts each classes. Inorder to better visualize the prediction a normalized confusion matrix is displayed, as shown below.



The above confusion matrix shows clearly how well the predictions match the true labels. We can observe good accuracy for emotions such as Happy, Surprise, Neutral and Sad. The prediction accuracy for Fear, Disgust and Angry are on the lower side. We can also see correlations between emotions such as Fear-Sad, Disgust-Angry, Sad-Neutral, Angry-Sad etc.. This is understandable while taking a closer look at the dataset. The differences between these emotions are very minute making it harder for even humans to classify.

Now the obtained model structure and weights needs to be saved, to be used for real-time prediction later on. The architecture is saved into json format and the weights are saved into h5 file, using the code below.

```
fer_json = model.to_json()
with open("fer.json","w") as json_file:
    json_file.write(fer_json)
model.save_weights("fer_wts.h5")
```

Gender Prediction - Model Training:

For gender prediction models, we are using the same set of libraries. Only the dataset and parameters that we are training changes.

```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.optimizers import Adam
from keras.layers.normalization import BatchNormalization
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score as asc
import matplotlib.pyplot as plt
%matplotlib inline
```

The above are the libraries that we will be using to create and train our model.

```
train_path = 'Data/train'
valid_path = 'Data/valid'
test_path = 'Data/test'
```

We specify the location of the data for Training, Testing and Validation.

```
train_batches = ImageDataGenerator(rescale=1./255, horizontal_flip=True).flow_from_directory(
    directory=train_path, target_size=(48,48), batch_size=50)
valid_batches = ImageDataGenerator(rescale=1./255).flow_from_directory(
    directory=valid_path, target_size=(48,48), batch_size=20)
test_batches = ImageDataGenerator(rescale=1./255).flow_from_directory(
    directory=test_path, target_size=(48,48), batch_size=20, shuffle=False)
```

```
Found 21200 images belonging to 2 classes.
Found 400 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
```

As before we use ImageDataGenerator module from keras.preprocessing library which allows us to load image data from a directory. We load the data from the previously arranged folders.Here for Training we have 21200 images, Validation we have 400 and Testing we have 200 images. The images belong to 2 classes of gender. We have also specified the size of the image (48x48), this is to be on par with the size of image in the previous model. It is convenient to use this later on when we predict real-time with OpenCV. We are using a normalization parameter for the pixel values to even out the highs and lows in the image. Horizontal flip for training set is enabled, which is a data augmentation technique for better generalization.We have given a batch size of 50 for training and 20 for testing and validation.

CNN Models Tried (Experiment and Observations):

Before getting the best ML model for this dataset, we had tried several model architectures, parameters and training cycles. This enabled us to draw inferences regarding what model architecture and the effect of various parameters involved.

Model 1: Sequential model with 6 Conv Layers

A basic CNN architecture was used, same one as used initially in the previous model. The model starts from larger filter size 5x5 for the first two layers and decreasing the filter size to 3x3 in the subsequent layers at the same time increasing the number of filters to extract the small features in the image data. Relu activation was used in every layers, and the output from the Conv layers was passed to a shallow dense layer with just 2 fully connected layers. A dropout feature was utilized in the fully connected layers to make some generalizations and reduce overfit.

```
input_shape = (48,48,3)
model.add(Conv2D(64, (5, 5), input_shape=input_shape,activation='relu', padding='same'))
model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (5, 5),activation='relu',padding='same'))
model.add(Conv2D(128, (5, 5),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(2))
model.add(Activation('softmax'))
```

The image above shows the structure of the model implemented. The model thus obtained consisted of 2,785,863 Trainable parameters and 1,152 Non-trainable parameters.

Several test runs were performed on this by varying the number of epochs and learning rate. Studying the performance of the obtained model and testing it with the realtime prediction program, we were able to get insights into how to improve the model. The observations were.

- The model gave good predictions right from the start. Since there were only 2 classes to predict, this did not turn to be a complicated problem even for a basic convolutional model.
- The Train and Validation accuracy was above 95% for most of the test runs.
- The Training and Validation Loss shows steady decrease, with validation loss showing slight fluctuations.

- In real time testing, the program was showing good results being able to predict male and female faces in various age categories.
- The model could be further improved by implementing some normalization and regularization parameters then training for higher epochs.

Model 2: Transfer Learning with MobileNet

As explained above, Transfer learning imports the model architecture from a well known and better performing model along with the weights. A selected number of layers in this model is then trained with our dataset along with modification to the output layer to obtain new weights and predict for our use case. Here we have used a popular ML model called MobileNet, which is capable of classifying 1000 classes. The MobileNet model along with weights were imported from keras library and trained with the gender dataset.

```
mobile = keras.applications.mobilenet.MobileNet()
mobile.summary()
```

```
x = mobile.layers[-6].output #dropping last 6 layers
predictions = Dense(2, activation='softmax')(x) # set output layer as 2 class
model = Model(inputs=mobile.input, outputs=predictions)
```

```
for layer in model.layers[:-30]: #here we are training the last 30 layers, this is a hyperparameter (to play around)
    layer.trainable = False
```

The output layer was set to a 2 class predicticition with softmax activation. The layers of the network except the last 30 were set to trainable. This number is a hyperparameter and several other numbers were tried to analyse the results. The total number of trainable parameters are 2,396,674 and non trainable parameters are 834,240 in this case.

Several test runs were performed on this by varying the number of epochs and learning rate. Studying the performance of the obtained model and testing it with the realtime prediction program, we were able to get insights.The observations were.

- The results were not as good as the previous model. The best accuracy obtained was close to 90%.
- Several variations were tried, by changing the number of trainable layers and epochs. But the test accuracy was not seen to improve above 92%.
- We came to the conclusion that MobileNet was not a good model when it comes to human faces, weather it is emotions or identifying gender.
- We decided to apply the learnings from the Emotion model and add normalization, regularization and dropout feature so that a better generalization can be achieved, which hopefully will result in a better model.

Finalized Model:

```
model = Sequential()

input_shape = (48,48,3)
model.add(Conv2D(32, (5, 5), input_shape=input_shape, activation='relu', padding='same'))
model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),activation='relu',padding='same'))
model.add(Dropout(rate = 0.1))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(units = 256, activation = 'relu', kernel_regularizer = l2(0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units = 128, activation = 'relu', kernel_regularizer = l2(0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units = 64, activation = 'relu', kernel_regularizer = l2(0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(2))
model.add(Activation('softmax'))
```

According to the inferences obtained from testing various model architectures and methods, we created the model structure for the Convolutional Neural Network. We have used a 3 Conv-Layer network as in the previous case, the convolution layers follow the same sets of parameters as in facial recognition model. It starts with a 32 filter of 5x5 size as the first layer which is connected to a 64 filter 5x5 filter layer including batch normalization and max pooling. The 3rd conv layer is a 128 filter with 3x3 size which can identify minute features of the facial image. This is followed by batch normalization and dropouts to reduce overfitting. The conv layers are then flattened and passed through fully connected layers to classify the features. Since we only require a two class classification, we need not use a large layer with many nodes as before. So the number of units was reduced to 256 in the first dense layer followed by 128 in the second and 64 in the third before going to the 2 class output layer. This reduces the parameters to around 4 million instead of 19 million in earlier case, resulting in faster training and prediction. Note that regularization terms, drop outs and batch normalization is also used here for decreasing overfitting and normalizing the weights in the network.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 48, 48, 32)        2432
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 48, 48, 64)        51264
_________________________________________________________________
batch_normalization_11 (Batc (None, 48, 48, 64)        256
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 24, 24, 128)       73856
_________________________________________________________________
dropout_9 (Dropout)          (None, 24, 24, 128)       0
_________________________________________________________________
batch_normalization_12 (Batc (None, 24, 24, 128)       512
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
flatten_3 (Flatten)          (None, 18432)             0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               4718848
_________________________________________________________________
batch_normalization_13 (Batc (None, 256)               1024
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_14 (Batc (None, 128)               512
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_15 (Batc (None, 64)                256
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 130
_________________________________________________________________
activation_3 (Activation)    (None, 2)                 0
=================================================================
Total params: 4,890,242
Trainable params: 4,888,962
Non-trainable params: 1,280
_________________________________________________________________
```

The list above shows the information about each layer in the network. In total we can see there are 4,890,242 parameters in which 4,888,962 are trainable.

```
model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Now we compile our model with the specified parameters. We have used Categorical Cross Entropy as our desired loss function with Adam as the optimizer and a constant learning rate as 0.0001.

```
%%time
H = model.fit_generator(generator=train_batches, steps_per_epoch=424,
    validation_data=valid_batches, validation_steps=20, epochs=30, verbose=2)
```

Now we do the actual training, with model.fit function. We specify the training and validation batches. The steps needed per epoch is obtained by dividing the total number of images in the set with the number of images per batch in the specified set.

Few Trial Runs:
(Test runs were done both by changing the number of epochs and the number of nodes in the fully connected layers, which changes the trainable of parameters.)

- ***Test 1:***
  epochs = 20
  lr = 0.0001
  trainable parameters = 2,788,418
  non trainable parameters = 1,152

  Epoch 20/20 - 197s - loss: 0.0259 - accuracy: 0.9905 - val_loss: 0.1035 - val_accuracy: 0.9450

  Test Accuracy:0.94

- ***Test 2:***
  epochs = 30
  lr = 0.0001
  trainable parameters = 9,715,202
  non trainable parameters = 2,048

  Epoch 30/30 - 160s - loss: 0.1732 - accuracy: 0.9724 - val_loss: 0.1030 - val_accuracy: 0.9475

  Test Accuracy: 0.93

- ***Test 3:***
  epochs = 30
  lr = 0.0001
  trainable parameters = 4,888,962
  non trainable parameters = 1,280

  Epoch 30/30 - 133s - loss: 0.1162 - accuracy: 0.9793 - val_loss: 0.4681 - val_accuracy: 0.9625

  Test Accuracy: 0.945

Results from Test 2 was used, after 20 epochs the obtained results are:
Epoch 30/30 - 133s - loss: 0.1162 - accuracy: 0.9793 - val_loss: 0.4681 - val_accuracy: 0.9625

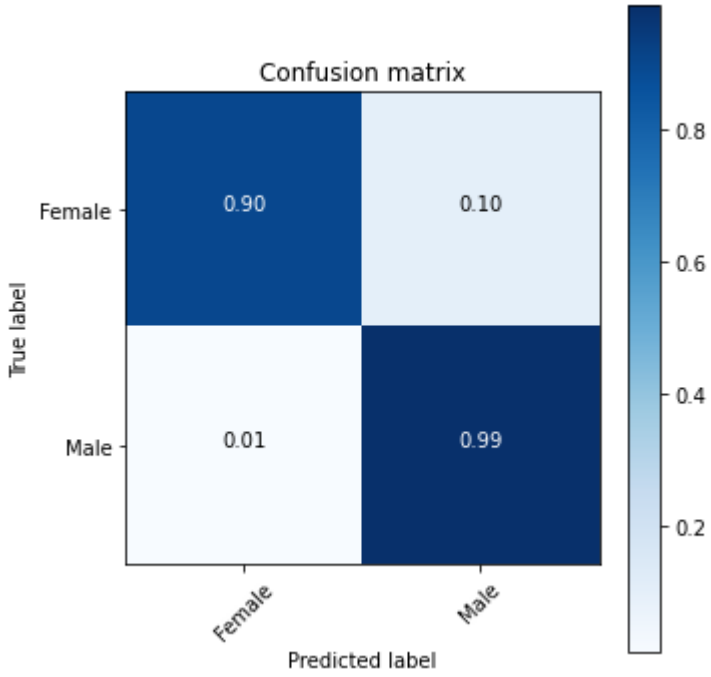
From the above result we can see that the model achieves a validation accuracy of around 96.25%. The training loss decreases gradually to 0.1 and the validation loss also shows a similar trend with fluctuations.

```
cm = confusion_matrix(y_true=test_labels, y_pred=predictions.argmax(axis=1))
print(f"Confusion Matrix:\n{cm}")
accuracy = asc(y_true=test_labels, y_pred=predictions.argmax(axis=1))
print(f"\nAccuracy:{accuracy}")
```

Test Accuracy: 0.945

```
Confusion Matrix:
[[90 10]
 [ 1 99]]
```

We are getting a test accuracy of 0.945 with this model.The confusion matrix shows how the model predicts each classes. Inorder to better visualize the prediction a normalized confusion matrix is displayed, as shown below.



From the above normalized confusion matrix, we can see the model works very well in predicting both the classes. However the prediction for 'Male' class seems a bit better than the 'Female' class. But an overall accuracy of 94.5% is to be considered pretty good.

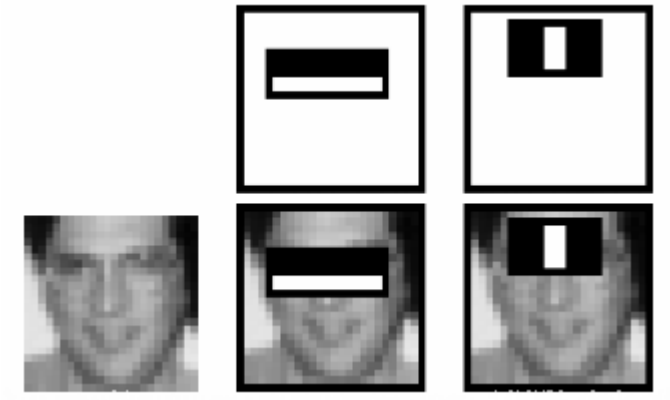
Now the obtained model structure and weights needs to be saved, to be used for real-time prediction later on. The architecture is saved into json format and the weights are saved into h5 file, using the code below.

```
gender_json = model.to_json()
with open("gender.json","w") as json_file:
    json_file.write(gender_json)
model.save_weights("gender_wts.h5")
```

Realtime Prediction:

Now we need to make a real-time prediction from the webcam. In order to implement this, first we need to detect the faces. This is done by making use of 'Haar Cascade' machine learning object detection algorithm. There exist a specific Haar Cascade function called 'Haar Cascade Frontal Face Default', which is able to detect human faces easily from a live feed.

A Haar feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.



In face detection, the first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. This is how Haar Cascade detects faces easily. After the faces are extracted we can leverage OpenCV and Keras to load our trained model and do predictions with it.

```
import os
import cv2
import numpy as np
from keras.models import model_from_json
from skimage import transform
```

Given above are the necessary imports needed for our real time prediction.

```
#Load model
model_emotion = model_from_json(open("fer.json", "r").read())
model_gender = model_from_json(open("gender.json", "r").read())
#Load weights
model_emotion.load_weights("fer_wts.h5")
model_gender.load_weights("gender_wts.h5")
```

Next we load the model architecture and the corresponding weights that we had saved before after training the model. This is what the above code does.

```
face_haar_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

cap=cv2.VideoCapture(0)

while True:
    ret,test_img=cap.read() # captures frame and returns boolean value and captured image
    if not ret:
        continue
    color_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)
    gray_img= cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)

    faces_detected = face_haar_cascade.detectMultiScale(gray_img, 1.03, 10, minSize=(180,180))

    for (x,y,w,h) in faces_detected:
        cv2.rectangle(test_img,(x-10,y-10),(x+w+10,y+h+10),(255,0,0), thickness=1)
        roi_g = gray_img[y-20:y+h+20,x-20:x+w+20] # cropping region of interest i.e. face area from  image
        np_image_g = np.array(roi_g).astype('float32')/255
        np_image_g = transform.resize(np_image_g, (48, 48, 1))
        np_image_g = np.expand_dims(np_image_g, axis=0)

        predicted_emotion = model_emotion.predict(np_image_g)

        roi_c = color_img[y-20:y+h+20,x-20:x+w+20] # cropping region of interest i.e. face area from  image
        np_image_c = np.array(roi_c).astype('float32')/255
        np_image_c = transform.resize(np_image_c, (48, 48, 3))
        np_image_c = np.expand_dims(np_image_c, axis=0)

        predicted_gender = model_gender.predict(np_image_c)[0][0]

        #find max indexed array
        max_index = np.argmax(predicted_emotion[0])

        emotions = ('Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral')

        if predicted_gender > 0.5:
            gender = "F"
        else:
            gender = "M"

        predicted = emotions[max_index] + ', ' + gender

        cv2.putText(test_img, predicted, (int(x-10), int(y-10)), cv2.FONT_HERSHEY_SIMPLEX, .8, (0,0,255), 2)

    resized_img = cv2.resize(test_img, (800, 550))
    cv2.imshow('Facial emotion analysis ',resized_img)

    if cv2.waitKey(10) == ord('q'):#wait until 'q' key is pressed
        break

cap.release()
cv2.destroyAllWindows
```

Now we go into predicting the live face image detected. First, we load the ‘haarcascade_frontalface_default.xml’ file, which will help us detect faces. Now we take the live feed capture using OpenCV Video Capture function. Both the gray and color image of the captured frame is taken, gray image is used for emotion prediction as we trained the model with grayscale images and the RGB one is used for gender prediction. We call the haarcascade function to detect faces in the image, and pass the cropped rectangular image to the ‘for’ loop which continuously gives predictions given a face is detected. Image transformations and resizing is performed on this image before doing the actual prediction. The predicted results are stored in the variable and are displayed using the OpenCV ‘putText’ function. The application is set to terminate on the press of ’q’ key, which jumps out of the ‘for’ loop.


Facial emotion analysis
Neutral, M
Facial emotion analysis
Happy, M
Facial emotion analysis
Angry, M
Facial emotion analysis
Sad, M
Facial emotion analysis
Fear, M
Facial emotion analysis
Surprise, M


Facial emotion analysis
Sad, M
Happy, F

The above images are snapshots of the real-time feed. The model seems to perform well in predicting Male and Female faces. When it comes to emotion detection the model works very well in predicting Happy, Angry and Surprise. The emotions Neutral, Sad and Fear are good, even though not the best. Disgust is the poorest. Looking into the images of the dataset, this is understandable because the emotions such as Fear and Surprise are very similar, also the similarity between Sad and Disgust is also significant making it difficult even for a human to distinguish between them.

Conclusion:

In this project we have been able to implement a program which can identify the gender and emotion of a human face with good accuracy. We have utilized Tensorflow with Keras API to achieve the result. The model was trained on NVIDIA CUDA device with the help of cuda-toolkit-9.0 for faster computation. By implementing Convolutional Neural Network, we were able to train models on both emotion dataset and gender dataset. The models were trained to achieve good accuracy by experimenting with various parameters and model architecture. The obtained best model from both the dataset was combined using OpenCV to give real-time facial recognition on the webcam feed.

We have an accurate model for predicting gender, However the emotion model can be improved further. We believe tuning the model architecture of the current CNN model to perfectly fit the given problem will result in better performance. This can include removing redundant model parameters and adding new parameters in the useful places of the CNN structure. Experimenting with learning rates and location of dropout layers can also lead to different results. We were unable to do more experiments due to the limitation of hardware that we are using, All the above training was done on our laptop devices powered by an NVIDIA GPU which takes significant time to train.

The model can be further improved if the dataset we are using closely reflected to realtime conditions. For example, the model does not work well on low light and noisy image conditions. Also the recognition is not accurate if the face is not properly aligned to the camera. So for realtime applications we need a dataset that includes all these factors.

This project was a great learning experience for us, and we are confident now to use and implement Convolutional Neural Networks on Keras with Tensorflow as backend. We are now aware of various data preparation techniques, model structure, parameters, tuning, analysing results, exporting and importing models and also how to combine them for our use cases. We also got to learn about OpenCV, and how it helps to deploy CNN models for real use cases.

References

1. “Understanding Neural Networks Through Deep Visualization” - Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson

2. “Real-time Emotion Recognition From Facial Expressions” - Minh-An Quinn, Grant Sivesind, Guilherme Reis

3. “Keras - Python Deep Learning Neural Network API” - www.deeplizard.com

4. “A Beginner's Guide To Understanding Convolutional Neural Networks” - Adit Deshapande - www.adeshpande3.github.io