Real Time Facial Expression Recognition (March 2019)

Authors: Preetam Jain, Rupesh Acharya

***Abstract*— The main idea of the paper was to determine the facial expressions in real time video. We tried 3 different architecture to compare accuracy, how and where the model is going wrong while training the accuracy. The models which we used were "Convolutional Neural Network", "Residual Network Architecture" and "Visual Geometry Group Architecture". The fact that deep learning can be used to recognize the facial emotions of humans unlocks new wide-impact possibilities to the field of human-computer interaction. The main task here is to recognize constantly changing facial emotions of a person and segregation them into 1 of the 7 categories namely: happy, sad, neutral, surprised, anger, fear and disgust. The conditions of the image or video are not necessarily ideal with low illumination, scale, lesser pixels based on the camera quality, etc. We tried 8 layered architecture which gave us 58% testing accuracy after experimenting with various hyperparameters. Similarly, we also trained and curated the ResNet and VGG19 models. Experiments and trials based on different models related comparison show shat the best combination is achieved using the ResNet model with 32 layers, 500 epochs and Adam optimizer and data augmentation techniques obtaining a test accuracy around 63% on the FER2013 dataset.**

I. Introduction

It is easy to classify a person's facial expressions by looking at them through the eyes. However, this is not the same with computer. The accuracy of it will not be same in-fact. Generally, for humans too it is very difficult to recognize current expression and predict the mood. So, we thought of making computer to do this job. There are so many use cases for the project like, it can be used in restaurants to check the reactions of the customers. It can also be used for interviewers to check the current expression how confident they are.

II. The Dataset

The dataset was taken from Kaggle. It consists of 35K images split into Training and testing. The images were in format of pixels and not the actual images. The images were labeled with the emotions in the emotion's column. The task was to first convert the pixels into the images to check how the images look like and were the labeled properly. We took the data into the pandas data frame and then converted into the numpy array. After converting into an array, we reshaped it into 48x48. Using plotly, we were able to see the images. They looked like *Figure 1* below. Images were already cropped to headshots to train it properly.

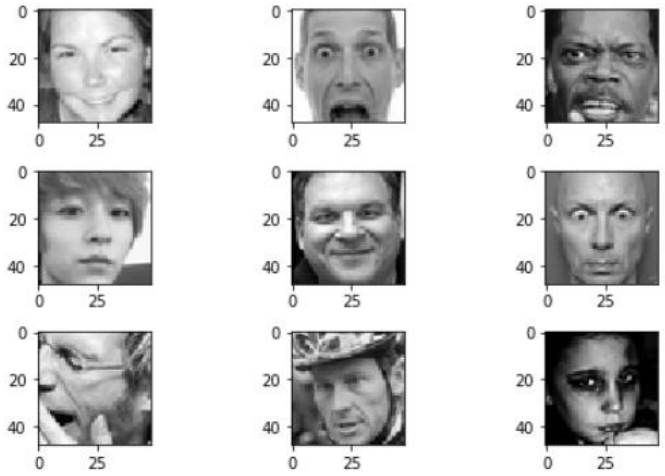


Figure 1

Figure 1, As we can see the images are 48x48 with different kind of expressions. The main expressions were of 7 categories: (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)

III. The Research.

A. *Convolutional Neural Networks:*

Convolution Neural Network is a class of deep, feed-forward artificial neural networks that has been applied to analyzing visual imagery. They are made up of neurons that have weights and biases. Each neuron receives some input, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. [1]

Layers in CNN:

- INPUT layer will hold the raw pixel values of the image, of width 32, height 32, and with grayscale values.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- RELU layer will apply an elementwise activation function, such as the max (0, x) thresholding at zero.
- POOL layer will perform a down sampling operation along the spatial dimensions (width, height)

- FC (i.e. fully-connected) layer will compute the class scores.

After training this model upto 200 epochs, we only got 85% training accuracy while the testing accuracy was around 50's. The model looks like this:

```
model.compile(loss='categorical_crossentropy'
    , optimizer=keras.optimizers.Adam()
    , metrics=['accuracy']
)
model.fit(train_generator, steps_per_epoch=batch_size, epochs=epochs)
```

After looking at the predictions we came to conclusion that the model is overfitting the data. To improve the accuracy, we started tweaking the hyperparameters like *Activation function, Cost functions, epochs, optimizers and every other parameter* but accuracy was not affected much.

After some research we read about random fit and *ImageDataGenerator* which generates the batches of tensor data with real-time augmentation which will help to increase the image quality and help to learn better.

We then re-organized our model by adding *dropouts and batchnormalization* in every layer so that the model doesn't overfit the data. Here is the updated model shown in *Figure 2*

```
model = Sequential()

#1st Convolution layer
model.add(Conv2D(64, (5, 5), activation='relu', input_shape=(48,48,1)))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

#2nd Convolution layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

#3rd Convolution layer
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

model.add(Flatten())

#Fully Connected Layers

model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(num_classes, activation='softmax'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py:263: c
(from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling dr
ensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

```
#batch process, generating batches of tensor image data with real-time data augmentation
gen = ImageDataGenerator()
train_generator = gen.flow(x_train, y_train, batch_size=batch_size)

model.compile(loss='categorical_crossentropy'
    , optimizer=keras.optimizers.Adam()
    , metrics=['accuracy']
)
model.fit_generator(train_generator, steps_per_epoch=batch_size, epochs=epochs) #train for randomly selected one
```

Figure 2

Initially, we trained this model for 200 epochs, but we found that the network reached *plateau* after almost 100 epochs. Since we did not want to overfit our model we trained it for 100 epochs.
After training, we got training accuracy around 98% and testing accuracy around 60%. Here is the history of loss and accuracy for both the sets.

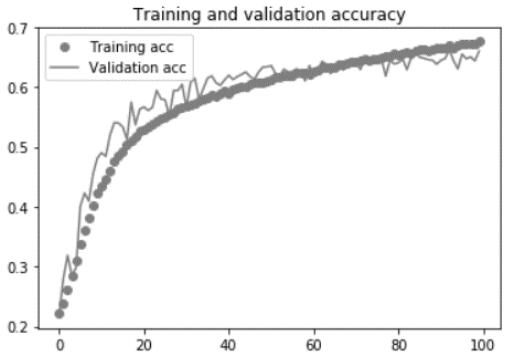


Figure 3

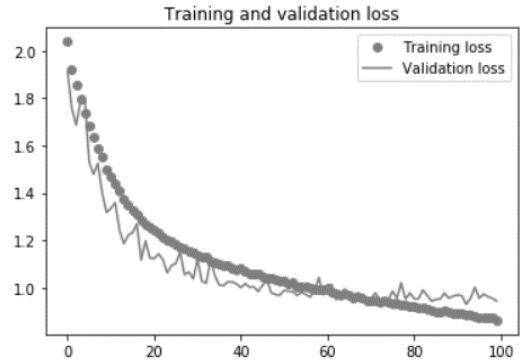


Figure 4

As we can see, the loss and accuracy are following similar pattern, the model is not overfitting the data. Also, the validation loss is still higher than the training loss. There is more scope of improvement and research.

Here are some results of our research, we tried on several images it was quite accurate.

Input Image:

Source: https://www.winningfaces.com.au/

Figure 5

Result:

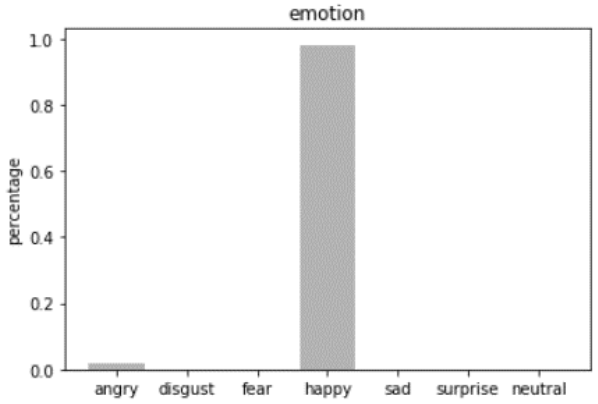


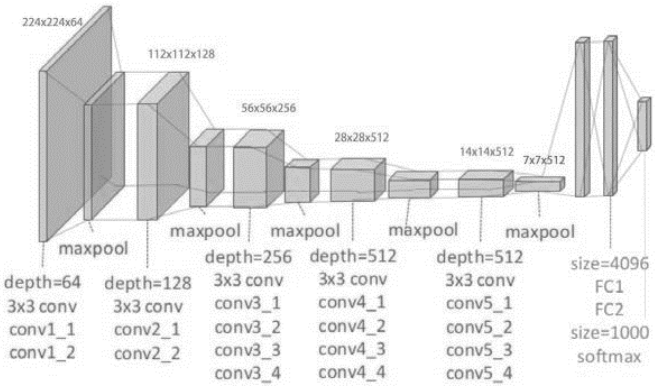
Figure 6

As we can observe, the input image was happy, and our model says it is 95% happy.

B. VGG19 Architecture:[3]

VGG-19 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 19 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224. [2]

Figure 7 represents the architecture of the VGG19



Source: https://www.researchgate.net/figure/llustration-of-the-network-architecture-of-VGG-19-model-conv-means-convolution-FC-means_fig2_325137356

Figure 7

Rather than using Keras VGG architecture we researched about tflearn library where we can create DNN's. We thought VGG19 would be better fit for image classification as it has won so many rewards in 2014.

Here is our model for VGG19, we have used 19 weighted layered model and then trained it on our data.

Here is our model looks like:

```
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.estimator import regression


# Building 'VGG Network'
network = input_data(shape=[None, 48, 48, 1])

network = conv_2d(network, 64, 3, activation='relu')
network = conv_2d(network, 64, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = conv_2d(network, 128, 3, activation='relu')
network = conv_2d(network, 128, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = conv_2d(network, 256, 3, activation='relu')
network = conv_2d(network, 256, 3, activation='relu')
network = conv_2d(network, 256, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = conv_2d(network, 512, 3, activation='relu')
network = conv_2d(network, 512, 3, activation='relu')
network = conv_2d(network, 512, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = conv_2d(network, 512, 3, activation='relu')
network = conv_2d(network, 512, 3, activation='relu')
network = conv_2d(network, 512, 3, activation='relu')
network = max_pool_2d(network, 2, strides=2)

network = fully_connected(network, 1024, activation='relu')
network = dropout(network, 0.5)
network = fully_connected(network, 1024, activation='relu')
network = dropout(network, 0.5)
network = fully_connected(network, 7, activation='softmax')

network = regression(network, optimizer='rmsprop',
                     loss='categorical_crossentropy',
                     learning_rate=0.0001)
```

Figure 8

After training the model on almost 600 epochs and tuning hyperparameters the training and testing accuracy came out to be 99% and 58% respectively.

Here is how we load the trained model and retrain it more for better accuracy.

```
In [7]: model = tflearn.DNN(network, checkpoint_path='model_vgg',
                            max_checkpoints=1, tensorboard_verbose=0)

        WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflo
        w.python.ops.math_ops) is deprecated and will be removed in a future version.
        Instructions for updating:
        Use tf.cast instead.

In [8]: #Loading the previously trained model to retrain it
        model.load('vgg19_emotion.tfl')

        WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/training/saver.py:1266: checkpoint_exists (fro
        m tensorflow.python.training.checkpoint_management) is deprecated and will be removed in a future version.
        Instructions for updating:
        Use standard file APIs to check for files with this prefix.
        INFO:tensorflow:Restoring parameters from /content/vgg19_final.tfl

In [9]: #training the model
        model.fit(x_train, y_train, n_epoch=200, shuffle=True,
                  show_metric=True, batch_size=128, snapshot_step=20,
                  snapshot_epoch=False, run_id='vgg_emotion')

        Training Step: 67499  | total loss: 0.05512 | time: 24.189s
        | RMSProp | epoch: 200 | loss: 0.05512 - acc: 0.9936 -- iter: 28672/28709
        Training Step: 67500  | total loss: 0.05089 | time: 24.298s
        | RMSProp | epoch: 200 | loss: 0.05089 - acc: 0.9935 -- iter: 28709/28709
        --
```

Figure 9

But, the problem with VGG was as the epochs increases the loss was also increasing instead of decreasing. Also, the VGG19 architecture is very heavy and takes lot of computation power. So it was difficult to train VGG again and try to improve accuracy.

Where it went wrong and why?

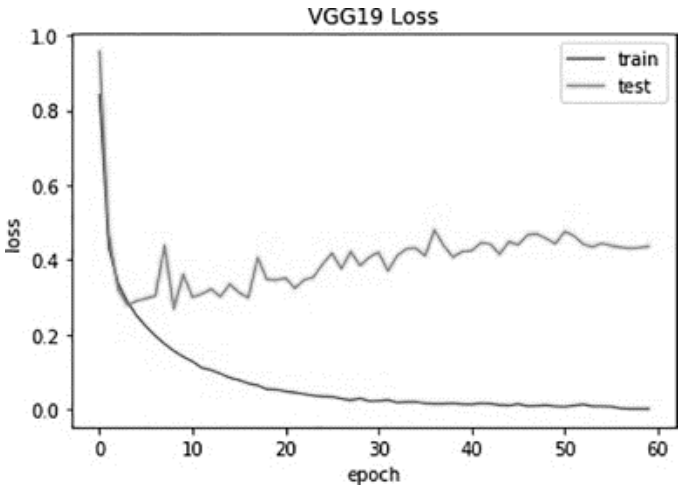


Figure 10

Figure 10 shows the training vs testing loss of VGG19 epochs when trained.

It went wrong because it needs more training time and computation as it becomes heavy as we train more. Also, the *vanishing gradient* problem is also not solved in VGG. But, as compared to VGG, if we apply so many layers to regular CNN, it will have more testing error.

To improve this model, we need to have good computation powered system which can handle huge load.

Here is the result for VGG19:

Input image and result:

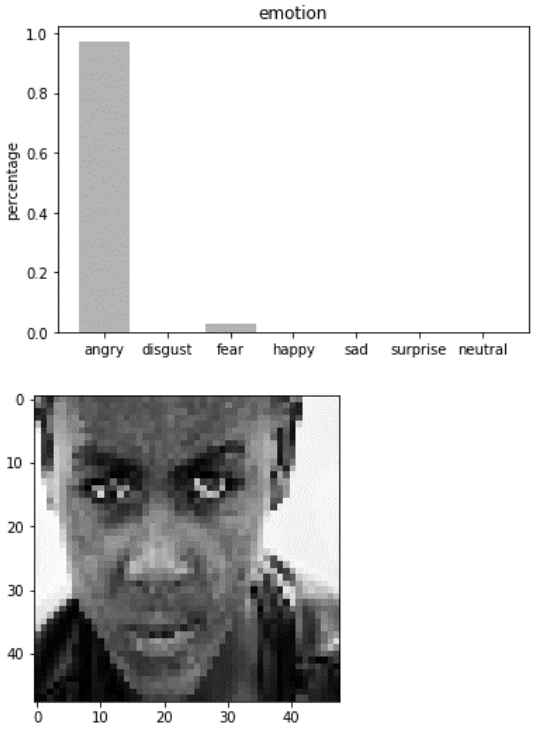


Figure 11

Figure 11 shows result for angry input and it shows result as expected

C. Residual Network Architecture:

The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more layers. The authors argue that stacking layers shouldn't degrade the network performance, because we could simply stack identity mappings (layer that doesn't do anything) upon the current network, and the resulting architecture would perform the same. This indicates that the deeper model should not produce a training error higher than its shallower counterparts. They hypothesize that letting the stacked layers fit a residual mapping is easier than letting them directly fit the desired underlaying mapping. And the residual block above explicitly allows it to do precisely that. [4]

As a matter of fact, ResNet was not the first to make use of shortcut connections, Highway Network introduced gated shortcut connections. These parameterized gates control how much information can flow across the shortcut. Similar idea can be found in the Long-Term Short Memory (LSTM) cell, in which there is a parameterized forget gate that controls how much information will flow to the next time step. Therefore, ResNet can be thought of as a special case of Highway Network. [4]

Figure 12 shows the network architecture

As we can see, how the identity mapping helps in the maintaining the accuracy and decreasing the loss.

For example,
If we give a[l] as a input to the model, the output of the first layer would give *a[l+1]*. Now, at the output of the second block, the identity of input a[l] will be given and the output will be shown as *(a[l+2] + a[l])*. This a[l] referred is known as residual.
Due to this property, the model will not overfit even if we train them extensively for more than 200 layers. Here is the sample residual block look like:

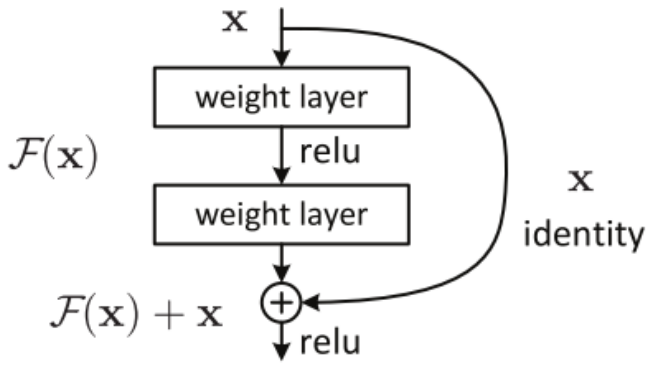


Figure 13

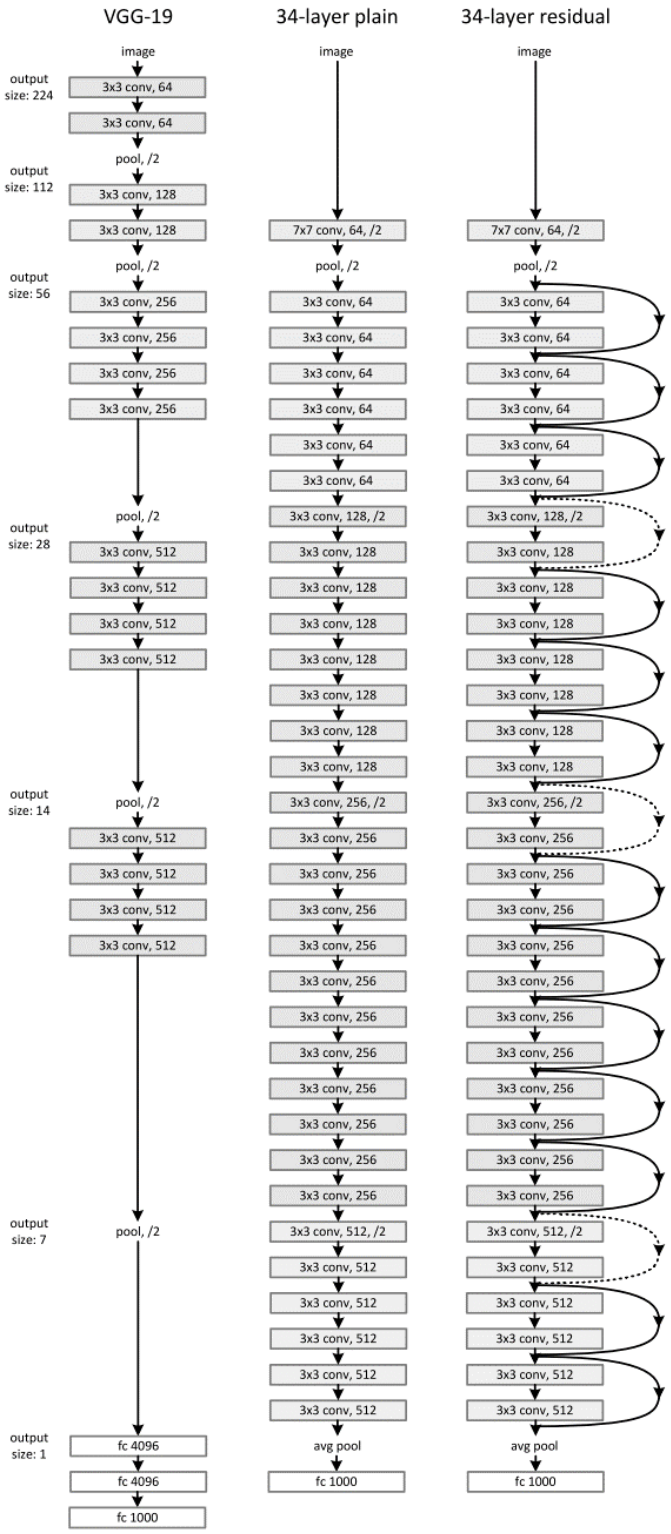


Figure 12 Source: Towards DataScience

What we did in our model?

Firstly, the model will preprocess the image using *ImageDataPreocessing()* library where the images are processed in real time. After that, the images will be augmented using *ImageAugmentation()* for better performance. Lastly, the images will be flipped randomly to train it better and give better results.

Here is our architecture look like:

```
#RealTime image preprocessing
img_prep = ImagePreprocessing()

# Zero Center (With mean computed over the whole dataset)
img_prep.add_featurewise_zero_center()

# STD Normalization (With standard deviation computed over the whole dataset)
img_prep.add_featurewise_stdnorm()
```

```
#This will augment the given image
img_aug = tflearn.ImageAugmentation()
#This will randomly flip the image for better training
img_aug.add_random_flip_leftright()
```

```
#Creating residual architecture
n = 5  #[3]
net = tflearn.input_data(shape=[None, 48, 48, 1], data_preprocessing=img_prep, data_augmentation=img_aug)
net = tflearn.conv_2d(net, nb_filter=16, filter_size=3, regularizer='L2', weight_decay=0.0001)
net = tflearn.residual_block(net, n, 16)
net = tflearn.residual_block(net, 1, 32, downsample=True)
net = tflearn.residual_block(net, n-1, 32)
net = tflearn.residual_block(net, 1, 64, downsample=True)
net = tflearn.residual_block(net, n-1, 64)
# Normalizing the previous batches by converting activation function close to 0 and loss function close to
net = tflearn.batch_normalization(net)
net = tflearn.activation(net, 'relu')
net = tflearn.global_avg_pool(net)

# Creating final layer
net = tflearn.fully_connected(net, 7, activation='softmax')
mom = tflearn.Momentum(learning_rate=0.1, lr_decay=0.0001, decay_step=32000, staircase=True, momentum=0.9)
net = tflearn.regression(net, optimizer=mom,
                         loss='categorical_crossentropy')
```

Figure 14

We trained our model on 32 layers and around 700 epochs and still the model doesn't overfit and improved continuously.

ResNet produced the testing accuracy of 63% (Highest in our case). We can improve it more by training it again for some more time.

Here are some results, as we can see in *figure 15* the input image given was neutral and out was also as expected. Input and output images are as follows:

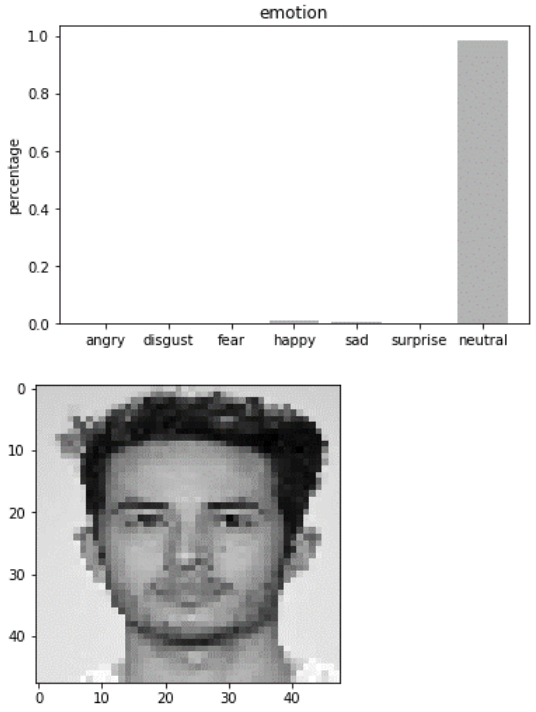


Figure 15

D. Live Video and Loaded Video implementation:

For this, we used python OpenCV library which opens the live web camera and captures every frame and *"haarcascade"* file helped us to detect the faces and the detected faces were then send to the model.

Here is the output looks like:

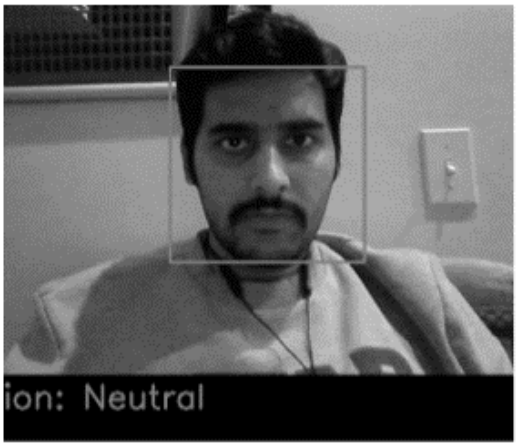


Figure 16

How did that work? Here is the code which explains everything:

```
In [19]: def smooth_emotions(prediction):
             """
             As the model will provide the mixture of results this function will give average of the emotions to 1 emotion

             """
             emotions = ["Angry", "Disgust", "Fear", "Happy", "Sad", "Surprise", "Neutral"]
             emotion_values = {'Angry': 0.0, 'Disgust': 0.0,'Fear': 0.0, 'Happy': 0.0, 'Sad': 0.0, 'Surprise': 0.0, 'Neutral': 0.0}

             emotion_probability, emotion_index = max((val, idx) for (idx, val) in enumerate(prediction[0]))
             emotion = emotions[emotion_index]

             # Append the new emotion and if the max length is reached pop the oldest value out
             emotion_queue.appendleft((emotion_probability, emotion))

             # Iterate through each emotion in the queue and create an average of the emotions
             for pair in emotion_queue:
                 emotion_values[pair[1]] += pair[0]

             # Select the current emotion based on the one that has the highest value
             average_emotion = max(emotion_values.items(), key=operator.itemgetter(1))[0]
             return average_emotion

In [20]: # preprocessing the input image
         def process_image(roi_gray, img):
                 image_scaled = np.array(cv2.resize(roi_gray, (48, 48)), dtype=float)
                 image_processed = image_scaled.flatten()
                 image_processed = image_processed.reshape([-1, 48, 48, 1])

                 prediction = model.predict(image_processed)
                 emotion = smooth_emotions(prediction)

                 font = cv2.FONT_HERSHEY_SIMPLEX
                 cv2.putText(img, "Emotion: " + emotion, (50, 450), font, 1, (0, 255, 255), 2)
                 cv2.imshow('img', img)

In [27]: # detecting human faces using HAAR cascade [1]
         # face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
         cap = cv2.VideoCapture(0) # 0 for webcam
         #cap = cv2.VideoCapture("../face_detection.mp4") # input the name of your video file here

         while True:
             ret, img = cap.read()
             gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
             faces = face_cascade.detectMultiScale(gray, 1.3, 5)
             for (x, y, w, h) in faces:
                 cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
                 roi_gray = gray[y:y + h, x:x + w]
                 roi_color = img[y:y + h, x:x + w]
                 process_image(roi_gray, img)
             if cv2.waitKey(1) & 0xFF == ord('q'):
                 break
         cap.release()
         cv2.destroyAllWindows()
```

Figure 17

Basically, here is how it is worked:
The face is first detected by the haarcascade, the face is then converted to grayscale and reshaped to 48x48 for the input to the model. This is then, converted into the np array and augmented and passed to the model. As this is a regression, the model generates the prediction percentage for each class. It is then averaged using the libraries. This value is then shown on the user's screen.

IV. Conclusion

To conclude, we will firstly show the accuracy of all the 3 models together,

Model No.	training accuracy	testing accuracy
CNN Sequential	99.7 %	57.2%
VGG19	99.3 %	58.0%
ResNet	97.4 %	62.0 %

Figure 17

The model which gave us the best result was ResNet model. Though the training accuracy is slightly less the better results were shown in testing accuracy.

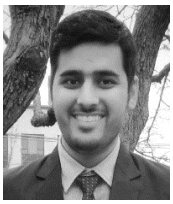
Future scope of this application can be for the professors to inspect in class who all are listening carefully using their expressions. One can make it more dynamic by just one click application will run.

References

[1] Human AI, https://becominghuman.ai/convolution-neural-network-in-modular-approach-e3a0969cf825?gi=6a55c5bfaa01
[2] MathWorks, https://www.mathworks.com/help/deeplearning/ref/vgg19.html;jsessionid=ccf9599bd865b423281a56299a68
[3] Research Gate, https://www.researchgate.net/figure/llustration-of-the-network-architecture-of-VGG-19-model-conv-means-convolution-FC-means_fig2_325137356
[4] Towards DataScience, https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035
Other References:
https://www.youtube.com/watch?v=PmZ29Vta7Vc
https://github.com/tflearn/tflearn/tree/master/examples/images
http://tflearn.org/tutorials/
http://sefiks.com/2018/01/01/facial-expression-recognition-with-keras/

Authors

Preetam Jain, Author is currently a Graduate student in Northeastern University and intended towards research and work in Data Science. This paper's research was done by him under the guidance of *Professor Nik Bear Brown*



Rupesh Acharya, Author is currently a Graduate student in Northeastern University and intended towards research and work in Data Science. This paper's research was done by him under the guidance of *Professor Nik Bear Brown*

Advisor: Professor Nik Bear Brown