## **Face Expression Recognition Description**

Computer animated agents and robots bring new dimension in human computer interaction which makes it vital as how computers can affect our social life in day-to-day activities. Face to face communication is a real-time process operating at a a time scale in the order of milliseconds. The level of uncertainty at this time scale is considerable, making it necessary for humans and machines to rely on sensory rich perceptual primitives rather than slow symbolic inference processes.

In this project we are presenting the real time facial expression recognition of seven most basic human expressions: ANGER, DISGUST, FEAR, HAPPY, NEUTRAL SAD, SURPRISE.

This model can be used for prediction of expressions of both still images and real time video. However, in both the cases we have to provide image to the model. In case of real time video the image should be taken at any point in time and feed it to the model for prediction of expression. The system automatically detects face using HAAR cascade then its crops it and resize the image to a specific size and give it to the model for prediction. The model will generate seven probability values corresponding to seven expressions. The highest probability value to the corresponding expression will be the predicted expression for that image.

## **Business Problem**

However, our goal here is to predict the human expressions, but we have trained our model on both human and animated images. Since, we had only approx 1500 human images which are very less to make a good model, so we took approximately 9000 animated images and leverage those animated images for training the model and ultimately do the prediction of expressions on human images.

For better prediction we have decided to keep the size of each image 350\*350.

For any image our goal is to predict the expression of the face in that image out of seven basic human expression

## **Problem Statement**

CLASSIFY THE EXPRESSION OF FACE IN IMAGE OUT OF SEVEN BASIC HUMAN EXPRESSION

## **Source Data**

We have downloaded data from 4 different sources.

1. Human Images Source-1: <http://www.consortium.ri.cmu.edu/ckagree/>
2. Human Images Source-2: <http://app.visgraf.impa.br/database/faces/>
3. Human Images Source-3: <http://www.kasrl.org/jaffe.html>
4. Animated Images Source: <https://grail.cs.washington.edu/projects/deepexpr/ferg-db.html>

## **Real-World Business Objective & Constraints**

1. Low-latency is required.
2. Interpretability is important for still images but not in real time. For still images, probability of predicted expressions can be given.
3. Errors are not costly.

## **Prerequisites**

You need to have installed following softwares and libraries in your machine before running this project.

1. Python 3
2. Anaconda: It will install ipython notebook and most of the libraries which are needed like sklearn, pandas, seaborn, matplotlib, numpy, PIL.
3. OpenCV
4. keras

## **Installing**

1. Python 3: <https://www.python.org/downloads/>
2. Anaconda: <https://www.anaconda.com/download/>
3. OpenCV: pip install opencv-python
4. Keras: pip install keras

## **Built With**

* ipython-notebook - Python Text Editor
* OpenCV - It is used for processing images
* Keras - Deep Learning Library
* Sklearn: It is a Machine Learning library but here it is used just to calculate accuracy and confusion matrix.

## **Real-World Business Objective & Constraints**

1. Low-latency is required.
2. Interpretability is important for still images but not in real time. For still images, probability of predicted expressions can be given.
3. Errors are not costly.

## **Y- Encoded Labels**

**Angry--1**

**Disgust --2**

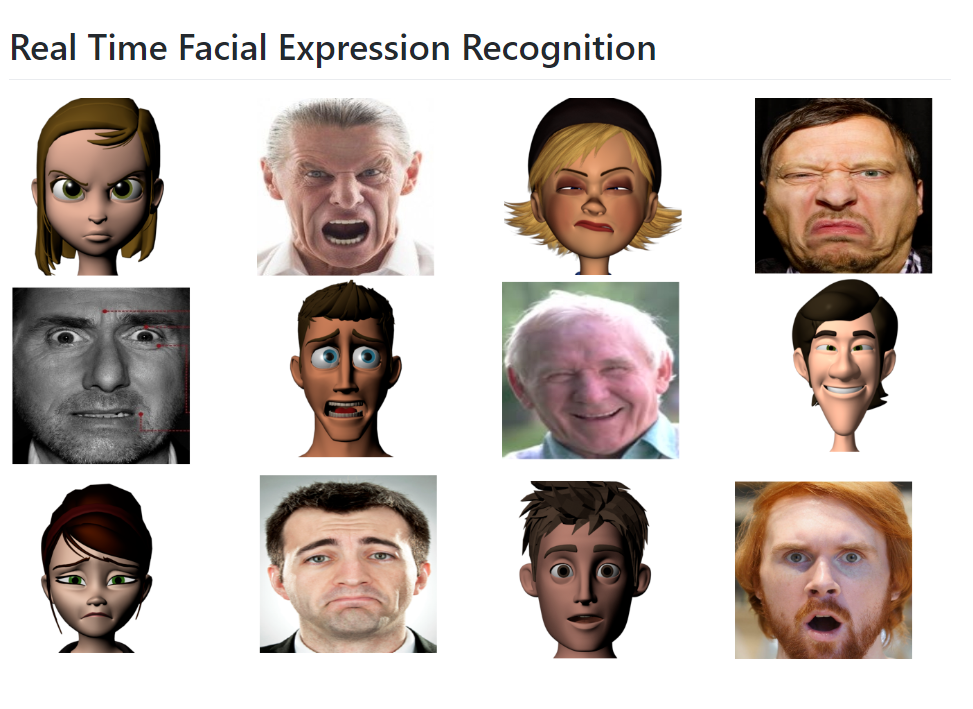
**Fear--3**

**Happy--4**

**Neutral--5**

**Sad--6**

**Surprise--7**



**import** **os**

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

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

**import** **seaborn** **as** **sns**

**import** **matplotlib.pyplot** **as** **plt**

**from** **PIL** **import** Image

**import** **glob**

**import** **cv2**

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **keras.layers** **import** Dropout, Dense

**from** **keras.layers.normalization** **import** BatchNormalization

**from** **keras.models** **import** Sequential, load\_model

**from** **keras.applications** **import** VGG16

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix

## **1. Reading the Data of Human Images**

### **Angry**

human\_angry = glob.glob("../Data/Human/Angry/\*")

human\_angry.remove('../Data/Human/Angry**\\**Thumbs.db')

print("Number of images in Angry emotion = "+str(len(human\_angry)))

Number of images in Angry emotion = 168

human\_angry\_folderName = [str(i.split("**\\**")[0])+"/" **for** i **in** human\_angry]

human\_angry\_imageName = [str(i.split("**\\**")[1]) **for** i **in** human\_angry]

human\_angry\_emotion = [["Angry"]\*len(human\_angry)][0]

human\_angry\_label = [1]\*len(human\_angry)

len(human\_angry\_folderName), len(human\_angry\_imageName), len(human\_angry\_emotion), len(human\_angry\_label)

(168, 168, 168, 168)

df\_angry = pd.DataFrame()

df\_angry["folderName"] = human\_angry\_folderName

df\_angry["imageName"] = human\_angry\_imageName

df\_angry["Emotion"] = human\_angry\_emotion

df\_angry["Labels"] = human\_angry\_label

df\_angry.head()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **folderName** | **imageName** | **Emotion** | **Labels** |  |
| **0** | ../Data/Human/Angry/ | KA.AN1.39.tiff | Angry | 1 |
| **1** | ../Data/Human/Angry/ | KA.AN2.40.tiff | Angry | 1 |
| **2** | ../Data/Human/Angry/ | KA.AN3.41.tiff | Angry | 1 |
| **3** | ../Data/Human/Angry/ | KL.AN1.167.tiff | Angry | 1 |
| **4** | ../Data/Human/Angry/ | KL.AN2.168.tiff | Angry | 1 |

### **Disgust**

human\_disgust = glob.glob("../Data/Human/Disgust/\*")

human\_disgust.remove('../Data/Human/Disgust**\\**Thumbs.db')

print("Number of images in Disgust emotion = "+str(len(human\_disgust)))

Number of images in Disgust emotion = 221

human\_disgust\_folderName = [str(i.split("**\\**")[0])+"/" **for** i **in** human\_disgust]

human\_disgust\_imageName = [str(i.split("**\\**")[1]) **for** i **in** human\_disgust]

human\_disgust\_emotion = [["Disgust"]\*len(human\_disgust)][0]

human\_disgust\_label = [2]\*len(human\_disgust)

len(human\_disgust\_folderName), len(human\_disgust\_imageName), len(human\_disgust\_emotion), len(human\_disgust\_label)

(221, 221, 221, 221)

df\_disgust = pd.DataFrame()

df\_disgust["folderName"] = human\_disgust\_folderName

df\_disgust["imageName"] = human\_disgust\_imageName

df\_disgust["Emotion"] = human\_disgust\_emotion

df\_disgust["Labels"] = human\_disgust\_label

df\_disgust.head()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **folderName** | **imageName** | **Emotion** | **Labels** |  |
| **0** | ../Data/Human/Disgust/ | KA.DI1.42.tiff | Disgust | 2 |
| **1** | ../Data/Human/Disgust/ | KA.DI2.43.tiff | Disgust | 2 |
| **2** | ../Data/Human/Disgust/ | KA.DI3.44.tiff | Disgust | 2 |
| **3** | ../Data/Human/Disgust/ | KL.DI1.170.tiff | Disgust | 2 |
| **4** | ../Data/Human/Disgust/ | KL.DI2.171.tiff | Disgust | 2 |

### **Fear**

human\_fear = glob.glob("../Data/Human/Fear/\*")

human\_fear.remove('../Data/Human/Fear**\\**Thumbs.db')

print("Number of images in Fear emotion = "+str(len(human\_fear)))

Number of images in Fear emotion = 122

human\_fear\_folderName = [str(i.split("**\\**")[0])+"/" **for** i **in** human\_fear]

human\_fear\_imageName = [str(i.split("**\\**")[1]) **for** i **in** human\_fear]

human\_fear\_emotion = [["Fear"]\*len(human\_fear)][0]

human\_fear\_label = [3]\*len(human\_fear)

len(human\_fear\_folderName), len(human\_fear\_imageName), len(human\_fear\_emotion), len(human\_fear\_label)

(122, 122, 122, 122)

df\_fear = pd.DataFrame()

df\_fear["folderName"] = human\_fear\_folderName

df\_fear["imageName"] = human\_fear\_imageName

df\_fear["Emotion"] = human\_fear\_emotion

df\_fear["Labels"] = human\_fear\_label

df\_fear.head()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **folderName** | **imageName** | **Emotion** | **Labels** |  |
| **0** | ../Data/Human/Fear/ | KA.FE3.47.tiff | Fear | 3 |
| **1** | ../Data/Human/Fear/ | KA.FE4.48.tiff | Fear | 3 |
| **2** | ../Data/Human/Fear/ | KL.FE1.174.tiff | Fear | 3 |
| **3** | ../Data/Human/Fear/ | MK.FE1.131.tiff | Fear | 3 |
| **4** | ../Data/Human/Fear/ | MK.FE2.132.tiff | Fear | 3 |

### **Happy**

human\_happy = glob.glob("../Data/Human/Happy/\*")

human\_happy.remove('../Data/Human/Happy**\\**Thumbs.db')

print("Number of images in Happy emotion = "+str(len(human\_happy)))

Number of images in Happy emotion = 280

human\_happy\_folderName = [str(i.split("**\\**")[0])+"/" **for** i **in** human\_happy]

human\_happy\_imageName = [str(i.split("**\\**")[1]) **for** i **in** human\_happy]

human\_happy\_emotion = [["Happy"]\*len(human\_happy)][0]

human\_happy\_label = [4]\*len(human\_happy)

len(human\_happy\_folderName), len(human\_happy\_imageName), len(human\_happy\_emotion), len(human\_happy\_label)

(280, 280, 280, 280)

df\_happy = pd.DataFrame()

df\_happy["folderName"] = human\_happy\_folderName

df\_happy["imageName"] = human\_happy\_imageName

df\_happy["Emotion"] = human\_happy\_emotion

df\_happy["Labels"] = human\_happy\_label

df\_happy.head()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **folderName** | **imageName** | **Emotion** | **Labels** |  |
| **0** | ../Data/Human/Happy/ | KA.HA1.29.tiff | Happy | 4 |
| **1** | ../Data/Human/Happy/ | KA.HA2.30.tiff | Happy | 4 |
| **2** | ../Data/Human/Happy/ | KA.HA3.31.tiff | Happy | 4 |
| **3** | ../Data/Human/Happy/ | KL.HA1.158.tiff | Happy | 4 |
| **4** | ../Data/Human/Happy/ | KL.HA2.159.tiff | Happy | 4 |

## **Analysing Data of Animated Images**

### **Distribution of class labels in Train, CV and Test**

df\_temp\_train = df\_anime\_train.sort\_values(by = "Labels", inplace = **False**)

df\_temp\_cv = df\_anime\_cv.sort\_values(by = "Labels", inplace = **False**)

df\_temp\_test = df\_anime\_test.sort\_values(by = "Labels", inplace = **False**)

TrainData\_distribution = df\_anime\_train["Emotion"].value\_counts().sort\_index()

CVData\_distribution = df\_anime\_cv["Emotion"].value\_counts().sort\_index()

TestData\_distribution = df\_anime\_test["Emotion"].value\_counts().sort\_index()

TrainData\_distribution\_sorted = sorted(TrainData\_distribution.items(), key = **lambda** d: d[1], reverse = **True**)

CVData\_distribution\_sorted = sorted(CVData\_distribution.items(), key = **lambda** d: d[1], reverse = **True**)

TestData\_distribution\_sorted = sorted(TestData\_distribution.items(), key = **lambda** d: d[1], reverse = **True**)

fig = plt.figure(figsize = (10, 6))

ax = fig.add\_axes([0,0,1,1])

ax.set\_title("Count of each Emotion in Train Data", fontsize = 20)

sns.countplot(x = "Emotion", data = df\_temp\_train)

plt.grid()

**for** i **in** ax.patches:

ax.text(x = i.get\_x() + 0.185, y = i.get\_height()+1.6, s = str(i.get\_height()), fontsize = 20, color = "grey")

plt.xlabel("")

plt.ylabel("Count", fontsize = 15)

plt.tick\_params(labelsize = 15)

plt.xticks(rotation = 40)

plt.show()

**for** i **in** TrainData\_distribution\_sorted:

print("Number of training data points in class "+str(i[0])+" = "+str(i[1])+ "("+str(np.round(((i[1]/df\_temp\_train.shape[0])\*100), 4))+"%)")

print("-"\*80)

fig = plt.figure(figsize = (10, 6))

ax = fig.add\_axes([0,0,1,1])

ax.set\_title("Count of each Emotion in Validation Data", fontsize = 20)

sns.countplot(x = "Emotion", data = df\_temp\_cv)

plt.grid()

**for** i **in** ax.patches:

ax.text(x = i.get\_x() + 0.21, y = i.get\_height()+0.3, s = str(i.get\_height()), fontsize = 20, color = "grey")

plt.xlabel("")

plt.ylabel("Count", fontsize = 15)

plt.tick\_params(labelsize = 15)

plt.xticks(rotation = 40)

plt.show()

**for** i **in** CVData\_distribution\_sorted:

print("Number of training data points in class "+str(i[0])+" = "+str(i[1])+ "("+str(np.round(((i[1]/df\_temp\_cv.shape[0])\*100), 4))+"%)")

print("-"\*80)

fig = plt.figure(figsize = (10, 6))

ax = fig.add\_axes([0,0,1,1])

ax.set\_title("Count of each Emotion in Test Data", fontsize = 20)

sns.countplot(x = "Emotion", data = df\_temp\_test)

plt.grid()

**for** i **in** ax.patches:

ax.text(x = i.get\_x() + 0.21, y = i.get\_height()+0.3, s = str(i.get\_height()), fontsize = 20, color = "grey")

plt.xlabel("")

plt.ylabel("Count", fontsize = 15)

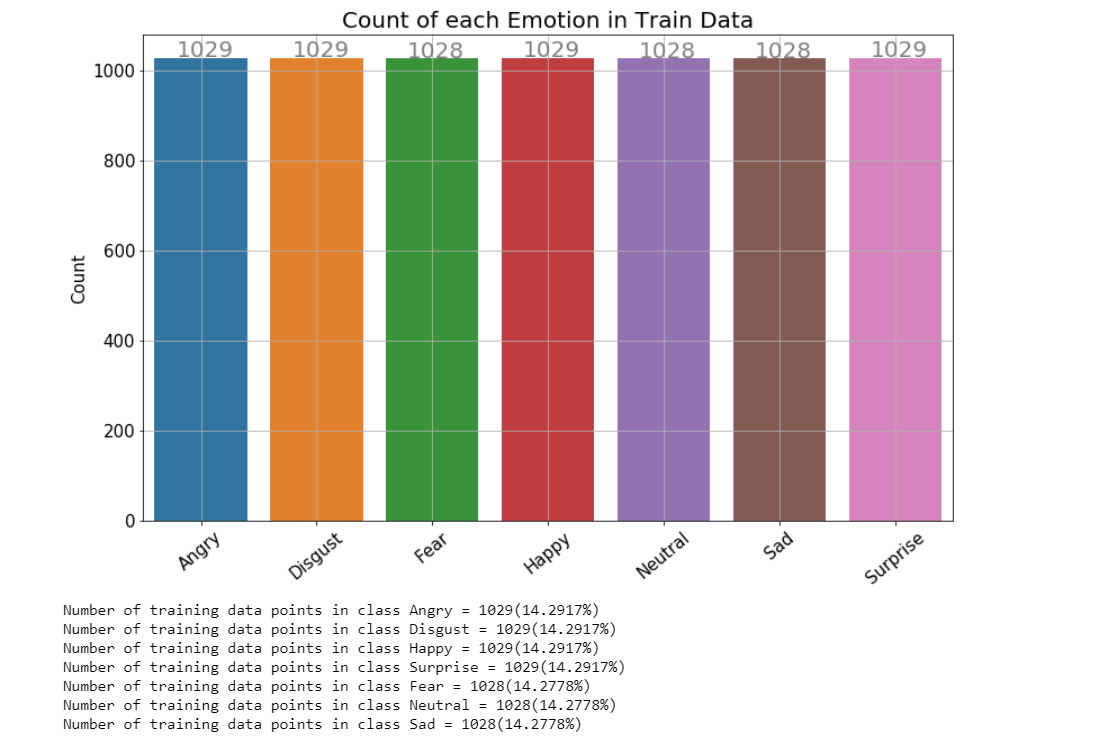
plt.tick\_params(labelsize = 15)

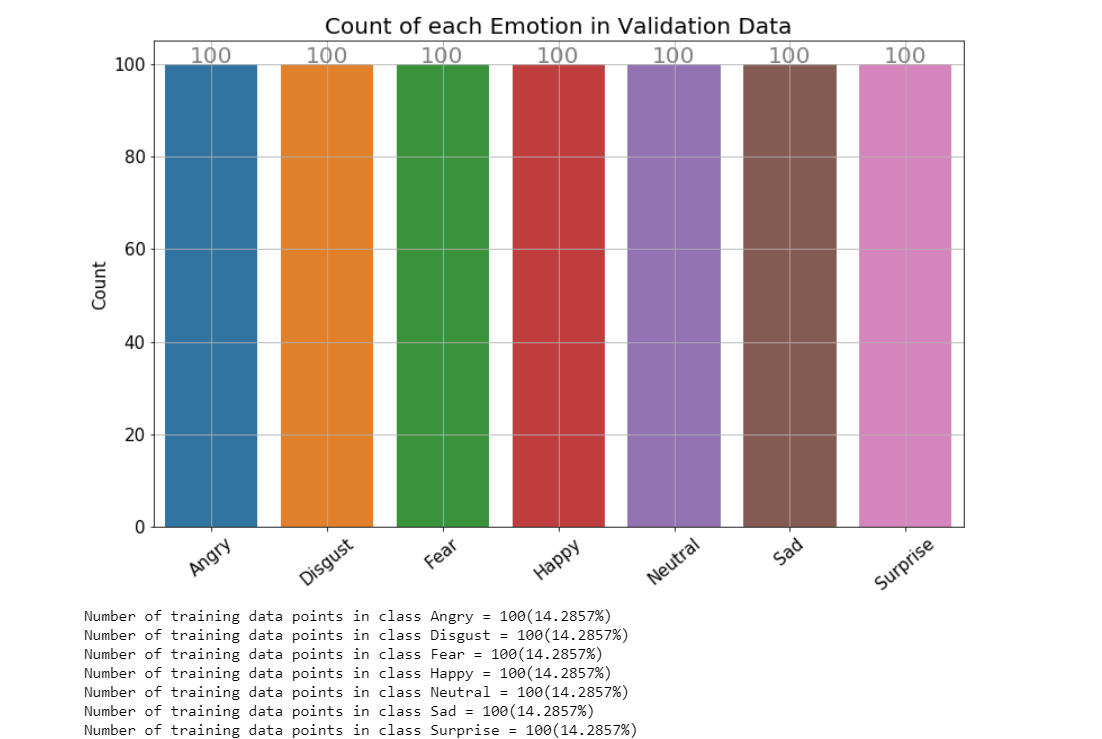
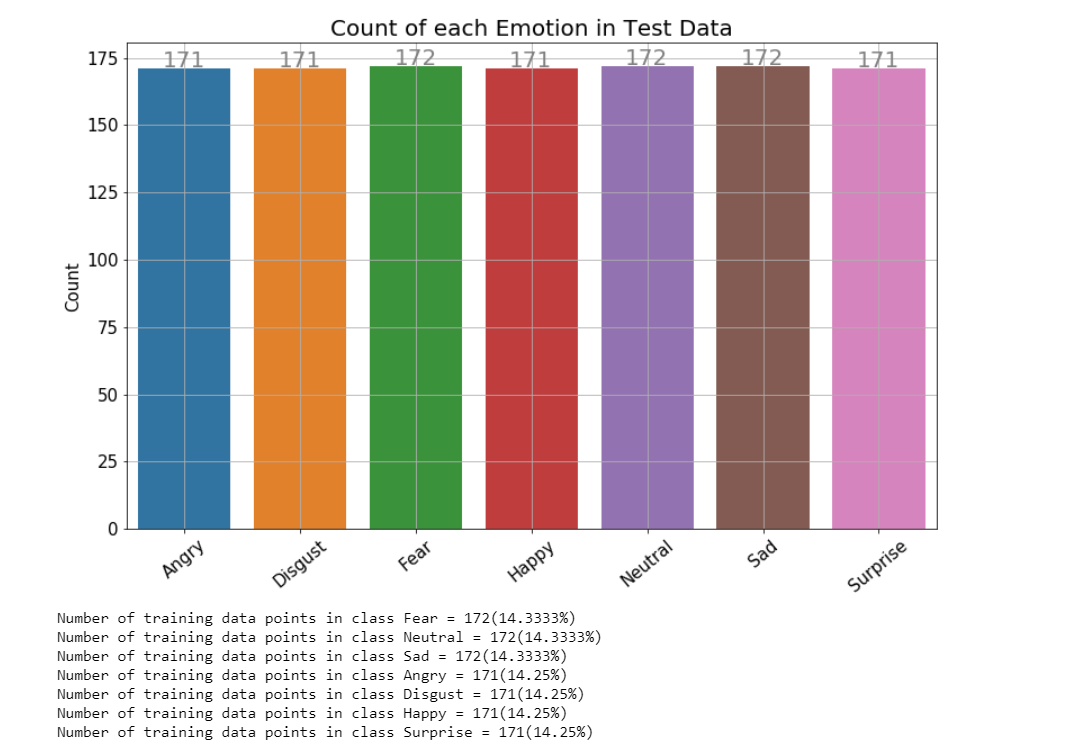
plt.xticks(rotation = 40)

plt.show()

**for** i **in** TestData\_distribution\_sorted:

print("Number of training data points in class "+str(i[0])+" = "+str(i[1])+ "("+str(np.round(((i[1]/df\_temp\_test.shape[0])\*100), 4))+"%)")





## **11. Modelling & Training**

no\_of\_classes = 7

*#model architecture*

**def** model(input\_shape):

model = Sequential()

model.add(Dense(512, activation='relu', input\_dim = input\_shape))

model.add(Dropout(0.1))

model.add(Dense(256, activation='relu'))

model.add(Dense(128, activation='relu'))

model.add(BatchNormalization())

model.add(Dense(64, activation='relu'))

model.add(Dense(output\_dim = no\_of\_classes, activation='softmax'))

**return** model

*#training the model*

SAVEDIR\_COMB\_TRAIN = "../Data/Bottleneck\_Features/Bottleneck\_CombinedTrain/"

SAVEDIR\_COMB\_TRAIN\_LABELS = "../Data/Bottleneck\_Features/CombinedTrain\_Labels/"

SAVEDIR\_CV\_HUMANS = "../Data/Bottleneck\_Features/Bottleneck\_CVHumans/"

SAVEDIR\_CV\_HUMANS\_LABELS = "../Data/Bottleneck\_Features/CVHumans\_Labels/"

SAVEDIR\_CV\_ANIME = "../Data/Bottleneck\_Features/Bottleneck\_CVAnimated/"

SAVEDIR\_CV\_ANIME\_LABELS = "../Data/Bottleneck\_Features/CVAnimated\_Labels/"

SAVER = "../Data/Model\_Save/"

input\_shape = 10\*10\*512 *#this is the shape of bottleneck feature of each image which comes after passing the image through VGG-16*

model = model(input\_shape)

*# model.load\_weights(os.path.join(SAVER, "model.h5"))*

model.summary()

model.compile(loss = 'categorical\_crossentropy', optimizer = "adam", metrics = ["accuracy"])

epochs = 20

batch\_size = 10

step = 0

combTrain\_bottleneck\_files = int(len(Train\_Combined) / batch\_size)

CVHuman\_bottleneck\_files = int(len(CV\_Humans) / batch\_size)

CVAnime\_bottleneck\_files = int(len(CV\_Animated) / batch\_size)

epoch\_number, CombTrain\_loss, CombTrain\_acc, CVHuman\_loss, CVHuman\_acc, CVAnime\_loss, CVAnime\_acc = [], [], [], [], [], [], []

**for** epoch **in** range(epochs):

avg\_epoch\_CombTr\_loss, avg\_epoch\_CombTr\_acc, avg\_epoch\_CVHum\_loss, avg\_epoch\_CVHum\_acc, avg\_epoch\_CVAnime\_loss, avg\_epoch\_CVAnime\_acc = 0, 0, 0, 0, 0, 0

epoch\_number.append(epoch + 1)

**for** i **in** range(combTrain\_bottleneck\_files):

step += 1

*#loading batch of train bottleneck features for training MLP.*

X\_CombTrain\_load = np.load(os.path.join(SAVEDIR\_COMB\_TRAIN, "bottleneck\_**{}**.npy".format(i+1)))

X\_CombTrain = X\_CombTrain\_load.reshape(X\_CombTrain\_load.shape[0], X\_CombTrain\_load.shape[1]\*X\_CombTrain\_load.shape[2]\*X\_CombTrain\_load.shape[3])

Y\_CombTrain = np.load(os.path.join(SAVEDIR\_COMB\_TRAIN\_LABELS, "bottleneck\_labels\_**{}**.npy".format(i+1)))

*#loading batch of Human CV bottleneck features for cross-validation.*

X\_CVHuman\_load = np.load(os.path.join(SAVEDIR\_CV\_HUMANS, "bottleneck\_**{}**.npy".format((i % CVHuman\_bottleneck\_files) + 1)))

X\_CVHuman = X\_CVHuman\_load.reshape(X\_CVHuman\_load.shape[0], X\_CVHuman\_load.shape[1]\*X\_CVHuman\_load.shape[2]\*X\_CVHuman\_load.shape[3])

Y\_CVHuman = np.load(os.path.join(SAVEDIR\_CV\_HUMANS\_LABELS, "bottleneck\_labels\_**{}**.npy".format((i % CVHuman\_bottleneck\_files) + 1)))

*#loading batch of animated CV bottleneck features for cross-validation.*

X\_CVAnime\_load = np.load(os.path.join(SAVEDIR\_CV\_ANIME, "bottleneck\_**{}**.npy".format((i % CVAnime\_bottleneck\_files) + 1)))

X\_CVAnime = X\_CVAnime\_load.reshape(X\_CVAnime\_load.shape[0], X\_CVAnime\_load.shape[1]\*X\_CVAnime\_load.shape[2]\*X\_CVAnime\_load.shape[3])

Y\_CVAnime = np.load(os.path.join(SAVEDIR\_CV\_ANIME\_LABELS, "bottleneck\_labels\_**{}**.npy".format((i % CVAnime\_bottleneck\_files) + 1)))

CombTrain\_Loss, CombTrain\_Accuracy = model.train\_on\_batch(X\_CombTrain, Y\_CombTrain) *#train the model on batch*

CVHuman\_Loss, CVHuman\_Accuracy = model.test\_on\_batch(X\_CVHuman, Y\_CVHuman) *#cross validate the model on CV Human batch*

CVAnime\_Loss, CVAnime\_Accuracy = model.test\_on\_batch(X\_CVAnime, Y\_CVAnime) *#cross validate the model on CV Animated batch*

print("Epoch: **{}**, Step: **{}**, CombTr\_Loss: **{}**, CombTr\_Acc: **{}**, CVHum\_Loss: **{}**, CVHum\_Acc: **{}**, CVAni\_Loss: **{}**, CVAni\_Acc: **{}**".format(epoch+1, step, np.round(float(CombTrain\_Loss), 2), np.round(float(CombTrain\_Accuracy), 2), np.round(float(CVHuman\_Loss), 2), np.round(float(CVHuman\_Accuracy), 2), np.round(float(CVAnime\_Loss), 2), np.round(float(CVAnime\_Accuracy), 2)))

avg\_epoch\_CombTr\_loss += CombTrain\_Loss / combTrain\_bottleneck\_files

avg\_epoch\_CombTr\_acc += CombTrain\_Accuracy / combTrain\_bottleneck\_files

avg\_epoch\_CVHum\_loss += CVHuman\_Loss / combTrain\_bottleneck\_files

avg\_epoch\_CVHum\_acc += CVHuman\_Accuracy / combTrain\_bottleneck\_files

avg\_epoch\_CVAnime\_loss += CVAnime\_Loss / combTrain\_bottleneck\_files

avg\_epoch\_CVAnime\_acc += CVAnime\_Accuracy / combTrain\_bottleneck\_files

print("Avg\_CombTrain\_Loss: **{}**, Avg\_CombTrain\_Acc: **{}**, Avg\_CVHum\_Loss: **{}**, Avg\_CVHum\_Acc: **{}**, Avg\_CVAnime\_Loss: **{}**, Avg\_CVAnime\_Acc: **{}**".format(np.round(float(avg\_epoch\_CombTr\_loss), 2), np.round(float(avg\_epoch\_CombTr\_acc), 2), np.round(float(avg\_epoch\_CVHum\_loss), 2), np.round(float(avg\_epoch\_CVHum\_acc), 2), np.round(float(avg\_epoch\_CVAnime\_loss), 2), np.round(float(avg\_epoch\_CVAnime\_acc), 2)))

CombTrain\_loss.append(avg\_epoch\_CombTr\_loss)

CombTrain\_acc.append(avg\_epoch\_CombTr\_acc)

CVHuman\_loss.append(avg\_epoch\_CVHum\_loss)

CVHuman\_acc.append(avg\_epoch\_CVHum\_acc)

CVAnime\_loss.append(avg\_epoch\_CVAnime\_loss)

CVAnime\_acc.append(avg\_epoch\_CVAnime\_acc)

model.save(os.path.join(SAVER, "model.h5")) *#saving the model on each epoc*

model.save\_weights(os.path.join(SAVER, "model\_weights.h5")) *#saving the weights of model on each epoch*

print("Model and weights saved at epoch **{}**".format(epoch + 1))

log\_frame = pd.DataFrame(columns = ["Epoch", "Comb\_Train\_Loss", "Comb\_Train\_Accuracy", "CVHuman\_Loss", "CVHuman\_Accuracy", "CVAnime\_Loss", "CVAnime\_Accuracy"])

log\_frame["Epoch"] = epoch\_number

log\_frame["Comb\_Train\_Loss"] = CombTrain\_loss

log\_frame["Comb\_Train\_Accuracy"] = CombTrain\_acc

log\_frame["CVHuman\_Loss"] = CVHuman\_loss

log\_frame["CVHuman\_Accuracy"] = CVHuman\_acc

log\_frame["CVAnime\_Loss"] = CVAnime\_loss

log\_frame["CVAnime\_Accuracy"] = CVAnime\_acc

log\_frame.to\_csv("../Data/Logs/Log.csv", index = **False**)

log = pd.read\_csv("../Data/Logs/Log.csv")

log

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Epoch** | **Comb\_Train\_Loss** | **Comb\_Train\_Accuracy** | **CVHuman\_Loss** | **CVHuman\_Accuracy** | **CVAnime\_Loss** | **CVAnime\_Accuracy** |
| **0** | 1 | 2.45 | 0.17 | 2.89 | 0.15 | 2.21 | 0.19 |
| **1** | 2 | 2.19 | 0.21 | 2.25 | 0.21 | 1.55 | 0.34 |
| **2** | 3 | 1.98 | 0.25 | 1.97 | 0.25 | 1.09 | 0.51 |
| **3** | 4 | 1.78 | 0.31 | 1.77 | 0.29 | 0.70 | 0.71 |
| **4** | 5 | 1.63 | 0.35 | 1.65 | 0.35 | 0.40 | 0.85 |
| **5** | 6 | 1.55 | 0.41 | 1.44 | 0.48 | 0.22 | 0.91 |
| **6** | 7 | 1.32 | 0.51 | 1.34 | 0.50 | 0.11 | 0.93 |
| **7** | 8 | 1.11 | 0.57 | 1.22 | 0.54 | 0.07 | 0.97 |
| **8** | 9 | 0.95 | 0.61 | 1.20 | 0.55 | 0.04 | 0.98 |
| **9** | 10 | 0.77 | 0.69 | 1.10 | 0.57 | 0.01 | 0.99 |
| **10** | 11 | 0.65 | 0.75 | 1.09 | 0.60 | 0.01 | 0.99 |
| **11** | 12 | 0.44 | 0.81 | 1.03 | 0.61 | 0.01 | 0.99 |
| **12** | 13 | 0.21 | 0.89 | 0.89 | 0.69 | 0.01 | 1.00 |
| **13** | 14 | 0.11 | 0.93 | 0.77 | 0.75 | 0.00 | 1.00 |
| **14** | 15 | 0.07 | 0.97 | 0.65 | 0.83 | 0.00 | 1.00 |
| **15** | 16 | 0.04 | 0.99 | 0.50 | 0.85 | 0.00 | 1.00 |
| **16** | 17 | 0.03 | 0.99 | 0.54 | 0.84 | 0.00 | 1.00 |
| **17** | 18 | 0.04 | 0.99 | 0.49 | 0.86 | 0.00 | 1.00 |
| **18** | 19 | 0.03 | 0.99 | 0.49 | 0.87 | 0.00 | 1.00 |
| **19** | 20 | 0.04 | 0.99 | 0.46 | 0.87 | 0.00 | 1.00 |

**def** plotting(epoch, train\_loss, CVHuman\_loss, CVAnimated\_loss, title):

fig, axes = plt.subplots(1,1, figsize = (12, 8))

axes.plot(epoch, train\_loss, color = 'red', label = "Train")

axes.plot(epoch, CVHuman\_loss, color = 'blue', label = "CV\_Human")

axes.plot(epoch, CVAnimated\_loss, color = 'green', label = "CV\_Animated")

axes.set\_title(title, fontsize = 25)

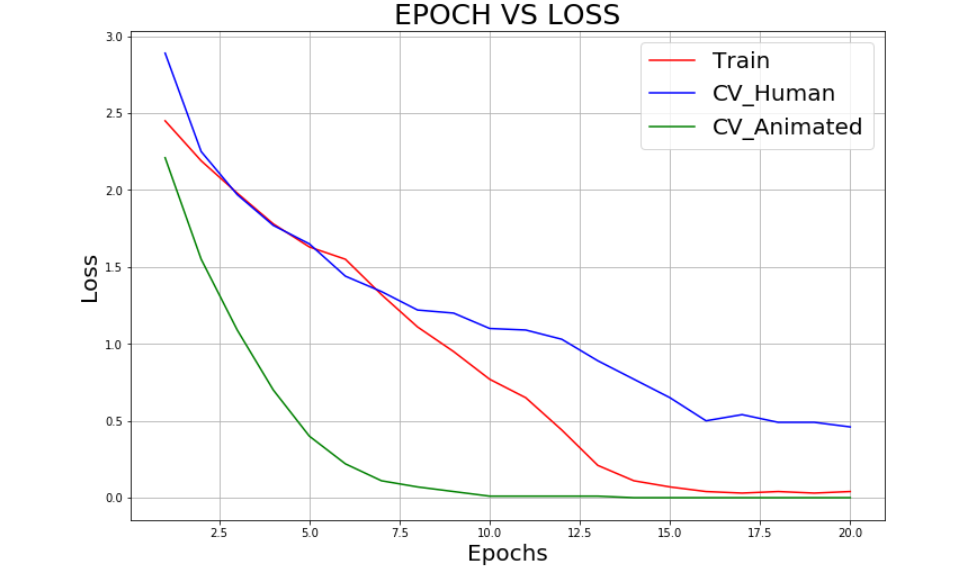
axes.set\_xlabel("Epochs", fontsize = 20)

axes.set\_ylabel("Loss", fontsize = 20)

axes.grid()

axes.legend(fontsize = 20)

plotting(list(log["Epoch"]), list(log["Comb\_Train\_Loss"]), list(log["CVHuman\_Loss"]), list(log["CVAnime\_Loss"]), "EPOCH VS LOSS")

**def** plotting(epoch, train\_acc, CVHuman\_acc, CVAnimated\_acc, title):

fig, axes = plt.subplots(1,1, figsize = (12, 8))

axes.plot(epoch, train\_acc, color = 'red', label = "Train\_Accuracy")

axes.plot(epoch, CVHuman\_acc, color = 'blue', label = "CV\_Human\_Accuracy")

axes.plot(epoch, CVAnimated\_acc, color = 'green', label = "CV\_Animated\_Accuracy")

axes.set\_title(title, fontsize = 25)

axes.set\_xlabel("Epochs", fontsize = 20)

axes.set\_ylabel("Accuracy", fontsize = 20)

axes.grid()

axes.legend(fontsize = 20)

plotting(list(log["Epoch"]), list(log["Comb\_Train\_Accuracy"]), list(log["CVHuman\_Accuracy"]), list(log["CVAnime\_Accuracy"]), "EPOCH VS ACCURACY")

## **12. Checking Test Accuracy**

**def** print\_confusionMatrix(Y\_TestLabels, PredictedLabels):

confusionMatx = confusion\_matrix(Y\_TestLabels, PredictedLabels)

precision = confusionMatx/confusionMatx.sum(axis = 0)

recall = (confusionMatx.T/confusionMatx.sum(axis = 1)).T

sns.set(font\_scale=1.5)

*# confusionMatx = [[1, 2],*

*# [3, 4]]*

*# confusionMatx.T = [[1, 3],*

*# [2, 4]]*

*# confusionMatx.sum(axis = 1) axis=0 corresponds to columns and axis=1 corresponds to rows in two diamensional array*

*# confusionMatx.sum(axix =1) = [[3, 7]]*

*# (confusionMatx.T)/(confusionMatx.sum(axis=1)) = [[1/3, 3/7]*

*# [2/3, 4/7]]*

*# (confusionMatx.T)/(confusionMatx.sum(axis=1)).T = [[1/3, 2/3]*

*# [3/7, 4/7]]*

*# sum of row elements = 1*

labels = ["ANGRY", "DISGUST", "FEAR", "HAPPY", "NEUTRAL", "SAD", "SURPRISE"]

plt.figure(figsize=(16,7))

sns.heatmap(confusionMatx, cmap = "Blues", annot = **True**, fmt = ".1f", xticklabels=labels, yticklabels=labels)

plt.title("Confusion Matrix", fontsize = 30)

plt.xlabel('Predicted Class', fontsize = 20)

plt.ylabel('Original Class', fontsize = 20)

plt.tick\_params(labelsize = 15)

plt.xticks(rotation = 90)

plt.show()

print("-"\*125)

plt.figure(figsize=(16,7))

sns.heatmap(precision, cmap = "Blues", annot = **True**, fmt = ".2f", xticklabels=labels, yticklabels=labels)

plt.title("Precision Matrix", fontsize = 30)

plt.xlabel('Predicted Class', fontsize = 20)

plt.ylabel('Original Class', fontsize = 20)

plt.tick\_params(labelsize = 15)

plt.xticks(rotation = 90)

plt.show()

print("-"\*125)

plt.figure(figsize=(16,7))

sns.heatmap(recall, cmap = "Blues", annot = **True**, fmt = ".2f", xticklabels=labels, yticklabels=labels)

plt.title("Recall Matrix", fontsize = 30)

plt.xlabel('Predicted Class', fontsize = 20)

plt.ylabel('Original Class', fontsize = 20)

plt.tick\_params(labelsize = 15)

plt.xticks(rotation = 90)

plt.show()

### **Test Data of Human Images**

model = load\_model("../Data/Model\_Save/model.h5")

predicted\_labels = []

true\_labels = []

batch\_size = 10

total\_files = int(len(Test\_Humans) / batch\_size) + 2 *#here, I have added 2 because there are 30 files in Test\_Humans*

**for** i **in** range(1, total\_files, 1):

img\_load = np.load("../Data/Bottleneck\_Features/Bottleneck\_TestHumans/bottleneck\_**{}**.npy".format(i))

img\_label = np.load("../Data/Bottleneck\_Features/TestHumans\_Labels/bottleneck\_labels\_**{}**.npy".format(i))

img\_bundle = img\_load.reshape(img\_load.shape[0], img\_load.shape[1]\*img\_load.shape[2]\*img\_load.shape[3])

**for** j **in** range(img\_bundle.shape[0]):

img = img\_bundle[j]

img = img.reshape(1, img\_bundle.shape[1])

pred = model.predict(img)

predicted\_labels.append(pred[0].argmax())

true\_labels.append(img\_label[j].argmax())

acc = accuracy\_score(true\_labels, predicted\_labels)

print("Accuracy on Human Test Data = **{}**%".format(np.round(float(acc\*100), 2)))

Accuracy on Human Test Data = 82.43%

print\_confusionMatrix(true\_labels, predicted\_labels)

