# Facial Emotion Recognition and Music Recommendation based on Detected Emotion

*By Kush Ramesh Jain (2017A3PS0425G)*

**Submitted to: Prof. Sujith Thomas**

In Partial Fulfillment of the Course
**Study Oriented Project (CS F266)**

# INTRODUCTION

The inspiration behind this project was to create a better experience for people who listen to music. Music is an integral part of everyone's life, and what would be a better way to recommend music to someone based on how they are feeling now?

This Study Project involved the development of a Deep Learning Model to identify the emotion from an image of a person's face through his/her expressions. Using the identified emotion, another model would recommend songs based on the person's emotion. To implement the second aspect of the project, a dataset based on songs were classified into emotions. Further explanation is provided in later sections of the report.

# Facial Emotion Recognition

## DATASET

The **Karolinska Directed Emotional Faces** (**KDEF**) has been used. It is a set of totally 4900 pictures of human facial expressions. The set of pictures contains 70 individuals displaying seven different emotional expressions. Each expression is viewed from 5 different angles. The subjects used are 70 amateur actors (35 males and 35 females). The subjects have been chosen if they match a specific criterion of selection:

Selection criteria*:* Age between 20 and 30 years of age. No beards, mustaches, earrings or eyeglasses, and preferably no visible make-up during photo-session.

Procedure for shooting of images for data collection:

All subjects received written instructions in advance. These instructions entailed a description of the seven different expressions that they were to pose during the photo session. The subject was asked to rehearse the different expressions for 1 hour before coming to the photo session. It was emphasized that the subject should try to evoke the emotion that was to be expressed, and - while maintaining a way of expressing the emotion that felt natural to them - try to make the expression strong and clear. All subjects wore special gray T-shirts. After a session of rehearsal, the subjects were shot in one expression at the time until all seven expressions had been shot (series one). The subjects were shot once again in all expressions and angles (series two).

Naming of each photograph:

*Example:* AF01ANFL.JPG

Letter 1: Session (A = series one, B = series two)

Letter 2: Gender (F = Female, M = Male)

Letter 3 and 4: Identity Number (01 to 35)

Letter 5 and 6: Expression (AF = afraid, AN = angry, DI = disgusted, HA = happy, NE = neutral, SA = sad, SU = surprised)

Letter 7 and 8: Angle (FL = full left profile, HL = half left profile, S = straight, HR = half right profile, FR = full right profile)

*Extension: Picture format* JPG = jpeg (Joint Photographic Experts Group)

Reference: Lundqvist, D., & Litton, J. E. (1998). The Averaged Karolinska Directed Emotional Faces - AKDEF, CD ROM from Department of Clinical Neuroscience, Psychology Section, Karolinska Institute, ISBN 91-630-7164-9.

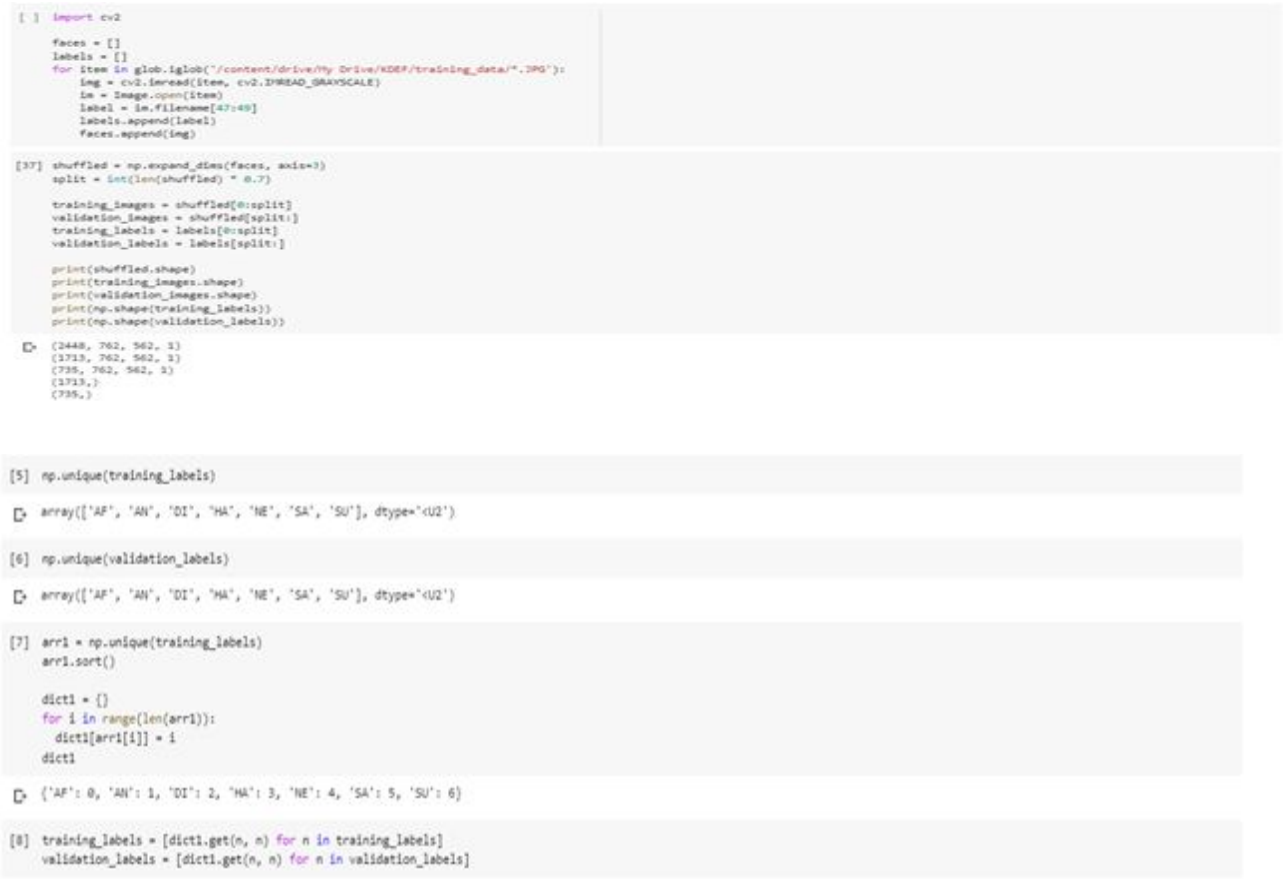
Refer to: http://kdef.se/

# APPROACH AND METHODOLOGY

The dataset was mounted on Google Drive, and all the work was done on the Google Colaboratory platform since it provides free access to GPUs. Colab allows the writing and executing of Python code in the browser, eliminating the need for a compiler or IDE on your computer.

Frameworks such as Tensorflow and Keras can be used in it easily, and other Data Science libraries, such as numpy, pandas, and seven, can also be imported for use.

Data Preprocessing:



```
[ ] import cv2

    faces = []
    labels = []
    for item in glob.iglob("/content/drive/My Drive/KDEF/training_data/*.JPG"):
        img = cv2.imread(item, cv2.IMREAD_GRAYSCALE)
        im = Image.open(item)
        label = im.filename[47:49]
        labels.append(label)
        faces.append(img)
```

```
[37] shuffled = np.expand_dims(faces, axis=3)
     split = int(len(shuffled) * 0.7)

     training_images = shuffled[0:split]
     validation_images = shuffled[split:]
     training_labels = labels[0:split]
     validation_labels = labels[split:]

     print(shuffled.shape)
     print(training_images.shape)
     print(validation_images.shape)
     print(np.shape(training_labels))
     print(np.shape(validation_labels))
```

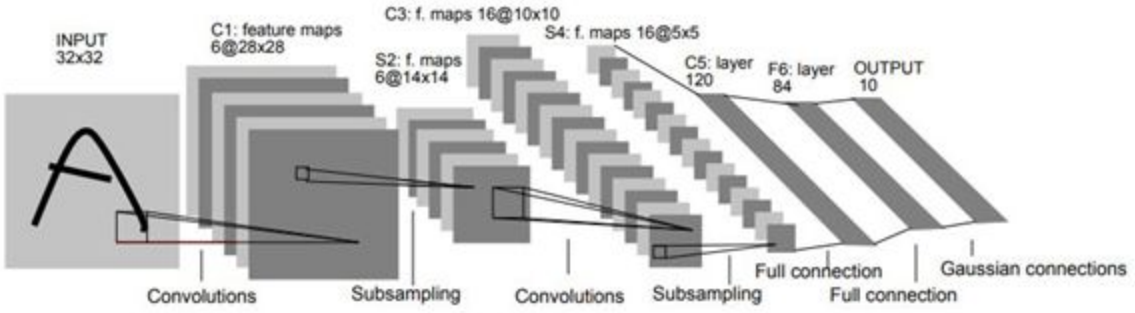
```
(2448, 762, 562, 1)
(1713, 762, 562, 1)
(735, 762, 562, 1)
(1713,)
(735,)
```

```
[5] np.unique(training_labels)

array(['AF', 'AN', 'DI', 'HA', 'NE', 'SA', 'SU'], dtype='<U2')

[6] np.unique(validation_labels)

array(['AF', 'AN', 'DI', 'HA', 'NE', 'SA', 'SU'], dtype='<U2')

[7] arr1 = np.unique(training_labels)
    arr1.sort()

    dict1 = {}
    for i in range(len(arr1)):
      dict1[arr1[i]] = i
    dict1

{'AF': 0, 'AN': 1, 'DI': 2, 'HA': 3, 'NE': 4, 'SA': 5, 'SU': 6}

[8] training_labels = [dict1.get(n, n) for n in training_labels]
    validation_labels = [dict1.get(n, n) for n in validation_labels]
```

Models used for training the data:

1. **LeNet-5:**

Refer to: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf




Since we have seven categories, the output layer used is of size 7.

```
[42] #LeNet-5
     from keras.models import Sequential,Model,load_model
     from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,AveragePooling2D

     model1 = Sequential([
         Conv2D(6, (3, 3), activation='relu', input_shape = (762, 562, 1)),
         AveragePooling2D(2, 2),
         Conv2D(16, (3, 3), activation='relu'),
         AveragePooling2D(2, 2),
         Flatten(),
         Dense(120, activation=tf.nn.relu),
         Dropout(0.5),
         Dense(84, activation=tf.nn.relu),
         Dropout(0.5),
         Dense(7, activation=tf.nn.softmax)])
```

```
[43] from keras.optimizers import Adam
     adam = Adam(lr = 0.0001)
     model1.compile(optimizer = adam, loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])

     seed = 7
     np.random.seed(seed)

     from keras.callbacks import EarlyStopping, ModelCheckpoint
     es = EarlyStopping(monitor = 'val_acc', mode='max', patience = 50)
     mc = ModelCheckpoint('model_lenet5.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
     model1.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 760, 560, 6)       60
_________________________________________________________________
average_pooling2d_7 (Average (None, 380, 280, 6)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 378, 278, 16)      880
_________________________________________________________________
average_pooling2d_8 (Average (None, 189, 139, 16)      0
_________________________________________________________________
flatten_4 (Flatten)          (None, 420336)            0
_________________________________________________________________
dense_10 (Dense)             (None, 120)               50440440
_________________________________________________________________
dropout_7 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_11 (Dense)             (None, 84)                10164
_________________________________________________________________
dropout_8 (Dropout)          (None, 84)                0
_________________________________________________________________
dense_12 (Dense)             (None, 7)                 595
=================================================================
Total params: 50,452,139
Trainable params: 50,452,139
Non-trainable params: 0
_________________________________________________________________
```

Callbacks used in training the model:

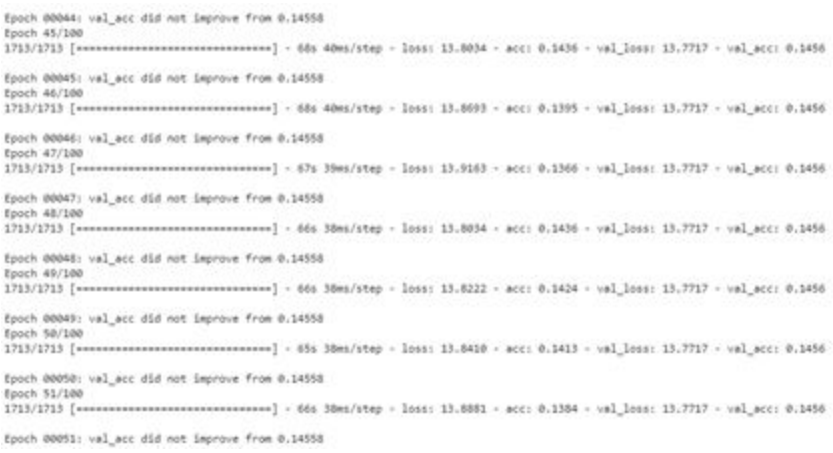
1. **Early Stopping:**

- It monitors the performance measure specified by the “monitor” parameter and ends training the model when the value of this performance measure stops increasing.
- Another parameter, “patience,” is the number of epochs that produced the monitored quantity with no improvement after which training will be stopped.

2. **Model Checkpoint:**

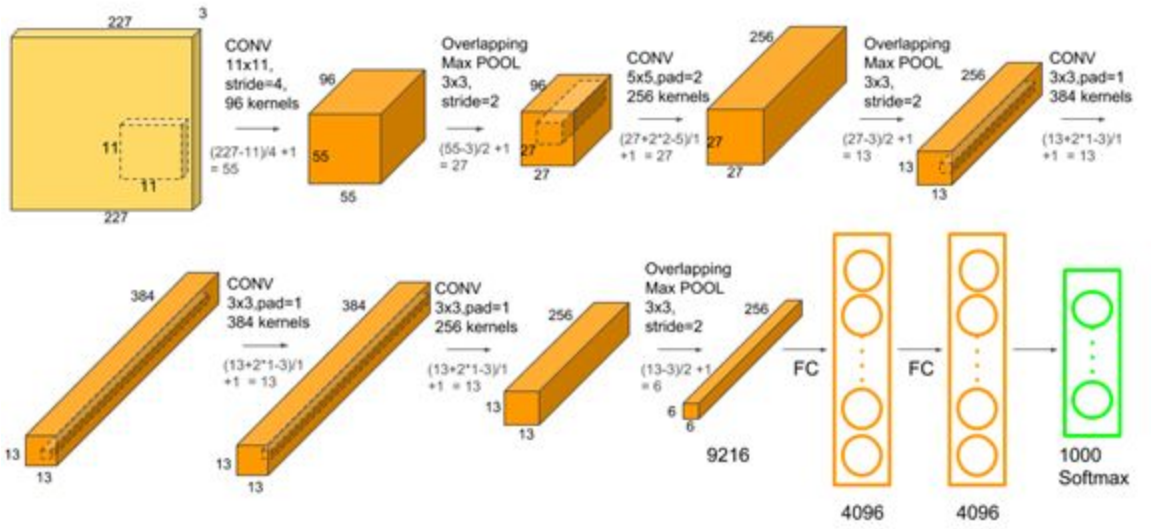
- I have used it to save the best model observed during training as defined by a chosen performance measure on the validation dataset.

The performance measure used is ‘val_acc’ (accuracy of the model on the validation dataset). We have called these two callbacks in the fit() function of the model. When this model was trained for 100 epochs, it stopped at epoch 51 (due to Early Stopping) and gave us a validation accuracy of 0.14558, which is not very good.

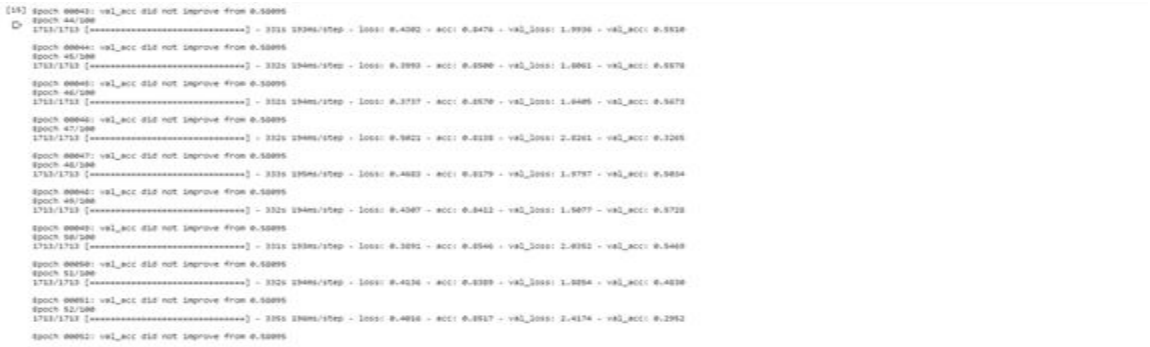


## 2. AlexNet:

Refer to:
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf




Since we have 7 categories, the output layer used is of size seven. AlexNet gave a validation accuracy of 58.095%, which is an improvement compared to LeNet-5.

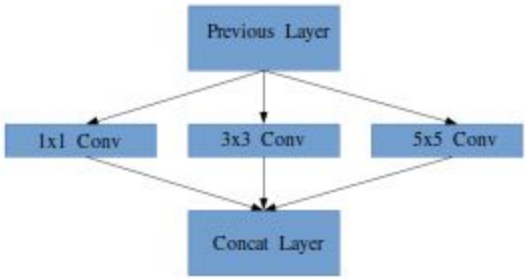


3. **Inception V3:**

Reference Paper: https://arxiv.org/abs/1409.4842

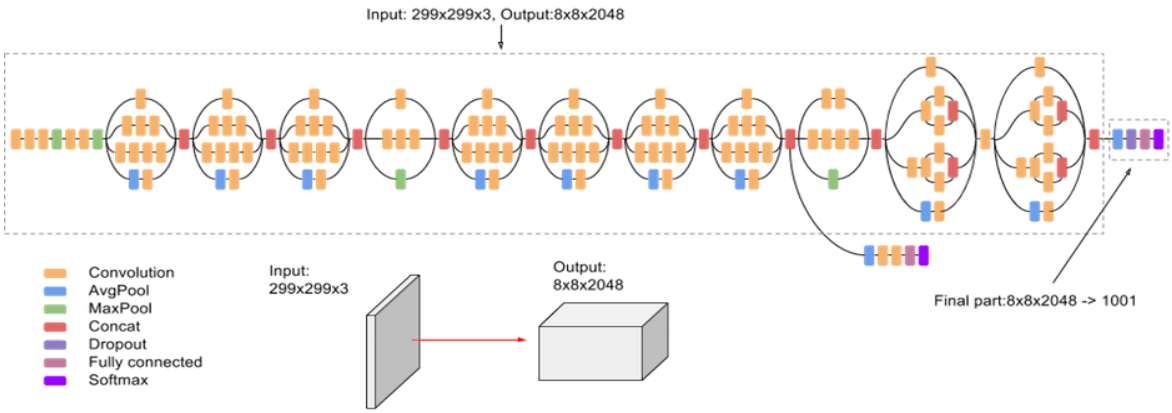
The inception layer is the core concept of a sparsely connected architecture.

An Inception Module:




Inception layer allows the internal layers to pick and choose which filter size will be relevant to learn the required information. the author calls on the Hebbian principle from human learning. This says that “neurons that fire together, wire together”. The author suggests that when creating a subsequent layer in a deep learning model, one should pay attention to the learning of the previous layer.

Inception v3 is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years. It is based on the original paper: "Rethinking the Inception Architecture for Computer Vision" by Szegedy, et. al.




I have used a Keras pre-trained model of Inception v3. This model is pre-trained on the famous ImageNet dataset. On the dataset I have used, this model has given an accuracy of 94.375%, which is very good.

```
from keras import applications
from keras.models import Sequential,Model,load_model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,GlobalAveragePooling2D

m = applications.inception_v3.InceptionV3(include_top=False, weights='imagenet', input_tensor=None, input_shape=(299, 299, 3), pooling=None)
x = m.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
predictions = Dense(7, activation= 'softmax')(x)
incep = Model(inputs = m.input, outputs = predictions)
```

```
Epoch 00090: val_acc did not improve from 0.94375
Epoch 91/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0034 - acc: 0.9988 - val_loss: 0.2404 - val_acc: 0.9250

Epoch 00091: val_acc did not improve from 0.94375
Epoch 92/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0035 - acc: 0.9988 - val_loss: 0.2406 - val_acc: 0.9250

Epoch 00092: val_acc did not improve from 0.94375
Epoch 93/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0034 - acc: 0.9988 - val_loss: 0.2411 - val_acc: 0.9250

Epoch 00093: val_acc did not improve from 0.94375
Epoch 94/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0033 - acc: 0.9988 - val_loss: 0.2414 - val_acc: 0.9250

Epoch 00094: val_acc did not improve from 0.94375
Epoch 95/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0027 - acc: 0.9988 - val_loss: 0.2397 - val_acc: 0.9250

Epoch 00095: val_acc did not improve from 0.94375
Epoch 96/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0030 - acc: 0.9988 - val_loss: 0.2403 - val_acc: 0.9250

Epoch 00096: val_acc did not improve from 0.94375
Epoch 97/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0032 - acc: 0.9988 - val_loss: 0.2423 - val_acc: 0.9250

Epoch 00097: val_acc did not improve from 0.94375
Epoch 98/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0033 - acc: 0.9988 - val_loss: 0.2425 - val_acc: 0.9250

Epoch 00098: val_acc did not improve from 0.94375
Epoch 99/200
2448/2448 [==============================] - 87s 35ms/step - loss: 0.0034 - acc: 0.9988 - val_loss: 0.2421 - val_acc: 0.9313

Epoch 00099: val_acc did not improve from 0.94375
```

This model was saved to my Google Drive.

# Emotion Recognition of Music

The idea to recommend songs based on emotion was to detect the emotion of a song based on its Valence and Arousal values. The dataset I used is called the PMEmo Dataset. It contains emotion annotations of 794 songs as well as the simultaneous electrodermal activity (EDA) signals. These songs were classified into emotions based on their valence and arousal values through the "Circumplex model of effect":

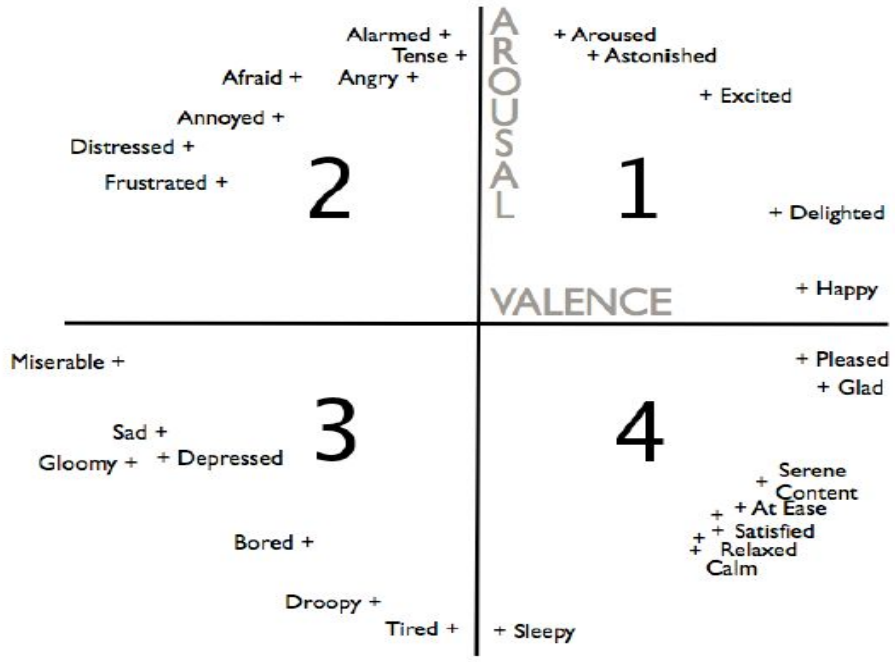



*Figure 2.* "Circumplex model of affect", adapted from Russel (1980)

Refer to:

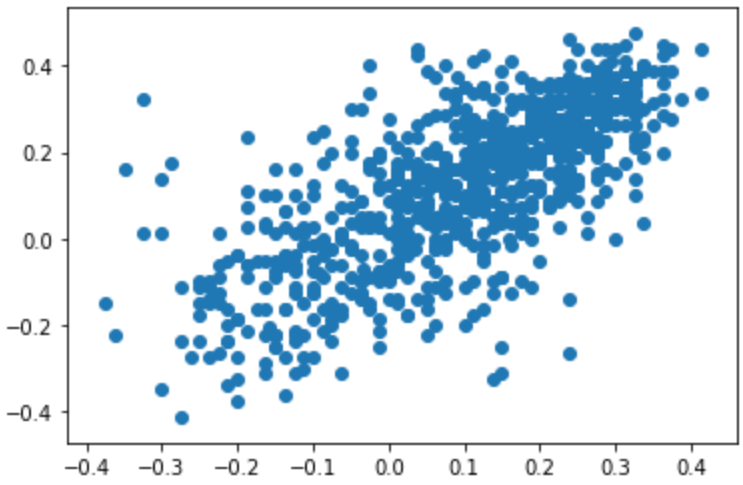
1. http://www.mtg.upf.edu/system/files/publications/Laurier_Herrera_Detection_Emotion_Music-original.pdf

2. http://delivery.acm.org/10.1145/3210000/3206037/p135-zhang.pdf?ip=103.225.100.51&id=3206037&acc=ACTIVE%20SERVICE&key=045416EF4DDA69D9%2EB8A9898014426BF2%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1573736260_4ef133390bf7db5c22a6217ed5a231dd

Since our dataset contains seven emotions, I thought that the emotion “disgusted” would not be associated with any song, so I have not classified songs in that emotion.

The Static Annotations given in the PMEmo dataset were used. This dataset contains 767 rows, one for each song. Each row contains 3 fields: the song ID, Mean Valence value and Mean Arousal value.

A visualization of the valence (X-axis) and arousal (Y-axis) values of all songs:




The songs were classified into emotions evoked by them based on the following code:

```
[ ]  dict1 = {1:'angry', 2:'sad', 3:'happy', 4:'surprised', 5:'neutral', 6:'afraid'}
     y = []
     for row in df2.index:
       if(df2['Valence(mean)'][row] > 0):
         if(df2['Arousal(mean)'][row] < 0.2 and df2['Arousal(mean)'][row] > 0):          #happy
           y.append(3)
           df2['temp'].loc[row] = 1
         elif(df2['Arousal(mean)'][row] > 0.2):                                          #surprised
           y.append(4)
           df2['temp'].loc[row] = 1
         elif(df2['Arousal(mean)'][row] < 0 and df2['Arousal(mean)'][row] > -0.1):       #happy
           y.append(3)
           df2['temp'].loc[row] = 1
         elif(df2['Arousal(mean)'][row] < -0.1 and df2['Arousal(mean)'][row] > -0.4):    #neutral, calm
           y.append(5)
           df2['temp'].loc[row] = 1

       else:
         if(df2['Arousal(mean)'][row] > 0 and df2['Arousal(mean)'][row] < 0.3):          #angry
           y.append(1)
           df2['temp'].loc[row] = 1
         elif(df2['Arousal(mean)'][row] > 0.3):                                          #afraid
           y.append(6)
           df2['temp'].loc[row] = 1
         elif(df2['Arousal(mean)'][row] < 0 and df2['Arousal(mean)'][row] > -0.3):       #sad
           y.append(2)
           df2['temp'].loc[row] = 1
```

After the processing, I got 746 songs that fit into the approximate values assigned above. These values are prone to error since they have been approximated from the graph.

Since we will be extracting features from the song lyrics using Natural Language Processing, the

song lyrics are read. The dataset contains song lyrics of 629 songs out of the 794 songs.

From the dataframe of 746 songs, I only picked those songs whose lyrics are available with us for feature extraction, so I got a total of 592 songs after this.

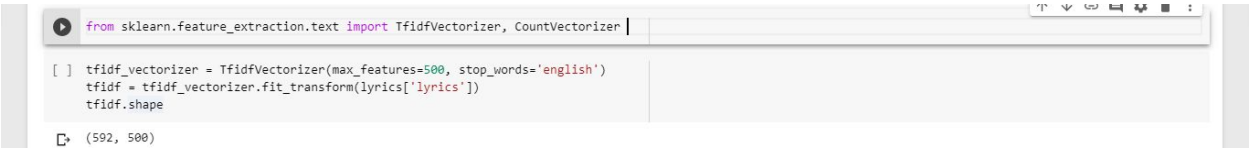
The non-alphabetic characters and symbols were removed from the lyrics using the Regular Expression Operations (re) library in Python.

The lyrics were tokenized into words by splitting them at the spaces.

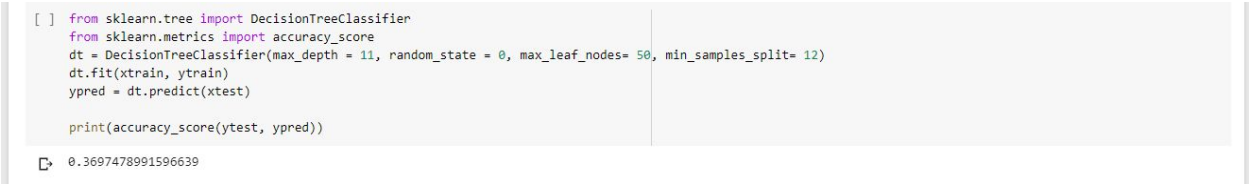
I have used the TfidfVectorizer to vectorize the words. This is an acronym that stands for "Term Frequency – Inverse Document" Frequency which are the components of the resulting scores assigned to each word.

- **Term Frequency**: This summarizes how often a given word appears within a document.
- **Inverse Document Frequency**: This downscales words that appear a lot across documents.

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. I have kept max_features parameter to 500.




Using the Decision Tree Classifier I got an accuracy of 36.97%:




Using Multi-Layer Perceptron and Random Forest Classifier gave an accuracy of 32.77 % and 36.97 % respectively.

```
[51] from sklearn.neural_network import MLPClassifier

     mlp = MLPClassifier(hidden_layer_sizes=(100, 100), activation='logistic', random_state= 0, max_iter = 650)
     mlp.fit(xtrain, ytrain)
     ypred2 = mlp.predict(xtest)

     print(accuracy_score(ytest, ypred2))
```

```
0.3277310924369748
```

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score
    rf = RandomForestClassifier(n_estimators = 800, random_state = 0)
    rf.fit(xtrain, ytrain)
    ypred3 = dt.predict(xtest)

    print(accuracy_score(ytest, ypred3))
```

```
0.3697478991596639
```

**Another Approach:**

Using the Keras text-processing functions, I extracted all the words from the lyrics (using the text_to_word_sequence function) and then

```
[52] from keras.preprocessing.text import text_to_word_sequence
     from keras.preprocessing.text import one_hot

     text = l
     result = text_to_word_sequence(text)
     print(result)

     words = set(result)
     vocab_size = len(words)
     print(vocab_size)
     result = one_hot(text, round(vocab_size*1.3))
     print(result)
```

```
Using TensorFlow backend.
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %tensorflow_version 1.x magic: more info.
['by', 'trap', 'girl', 'n', 'uhh', 'yeah', 'alright', 'by', 'octobersveryown', 'n', 'yeah', 'yeah', 'ye', 'by', 'i', 'lll', 'n', 'created', 'in', 'lyricseditor'
48
[56, 15, 37, 61, 48, 22, 15, 56, 19, 61, 22, 22, 40, 56, 60, 1, 61, 32, 46, 27, 61, 13, 56, 12, 61, 60, 36, 52, 61, 56, 47, 61, 35, 33, 61, 56, 61, 48, 14, 48,
```

I used GloVe embeddings of 100 dimensions (100d). GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

(Refer to Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. https://nlp.stanford.edu/pubs/glove.pdf)

Code to process the embeddings and the model used:

```
[ ] embeddings_index = dict()
    f = open('/content/glove.6B.100d.txt')
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    f.close()
```

```
[ ] vocabulary_size = 55
    embedding_matrix = np.zeros((vocabulary_size, 100))
    for word, index in t.word_index.items():
        if index > vocabulary_size - 1:
            break
        else:
            embedding_vector = embeddings_index.get(word)
            if embedding_vector is not None:
                embedding_matrix[index] = embedding_vector
```

```
from keras.models import Sequential, Model
from keras.layers import Embedding, SpatialDropout1D, Dense, LSTM, MaxPooling1D, Conv1D, GlobalMaxPooling1D, Dropout

model = Sequential()
model.add(Embedding(vocabulary_size, 100, weights=[embedding_matrix], input_length=X.shape[1], trainable = False))
model.add(Dropout(0.2))
model.add(Conv1D(64, 5, activation='relu'))
model.add(MaxPooling1D())
model.add(Dropout(0.2))
model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(6, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Model Summary:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 200, 100)          5500
_________________________________________________________________
dropout_1 (Dropout)          (None, 200, 100)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 196, 64)           32064
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 98, 64)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 98, 64)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 94, 128)           41088
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 79,426
Trainable params: 73,926
Non-trainable params: 5,500
_________________________________________________________________
```

The Model gave a validation accuracy of 38.202 %.

```
Epoch 00198: val_acc did not improve from 0.38202
Epoch 199/400
414/414 [==============================] - 0s 332us/step - loss: 0.1501 - acc: 0.9300 - val_loss: 4.2023 - val_acc: 0.3371

Epoch 00199: val_acc did not improve from 0.38202
Epoch 200/400
414/414 [==============================] - 0s 327us/step - loss: 0.1744 - acc: 0.9203 - val_loss: 4.3314 - val_acc: 0.3202

Epoch 00200: val_acc did not improve from 0.38202
Epoch 201/400
414/414 [==============================] - 0s 307us/step - loss: 0.1536 - acc: 0.9275 - val_loss: 4.3669 - val_acc: 0.2697

Epoch 00201: val_acc did not improve from 0.38202
Epoch 202/400
414/414 [==============================] - 0s 322us/step - loss: 0.1745 - acc: 0.9444 - val_loss: 4.1423 - val_acc: 0.3202

Epoch 00202: val_acc did not improve from 0.38202
Epoch 203/400
414/414 [==============================] - 0s 309us/step - loss: 0.1386 - acc: 0.9420 - val_loss: 4.1138 - val_acc: 0.3708

Epoch 00203: val_acc did not improve from 0.38202
Epoch 204/400
414/414 [==============================] - 0s 311us/step - loss: 0.1648 - acc: 0.9275 - val_loss: 4.2156 - val_acc: 0.2809

Epoch 00204: val_acc did not improve from 0.38202
Epoch 205/400
414/414 [==============================] - 0s 318us/step - loss: 0.1399 - acc: 0.9275 - val_loss: 4.1217 - val_acc: 0.3371

Epoch 00205: val_acc did not improve from 0.38202
Epoch 206/400
414/414 [==============================] - 0s 384us/step - loss: 0.1437 - acc: 0.9444 - val_loss: 3.9535 - val_acc: 0.3596

Epoch 00206: val_acc did not improve from 0.38202
Epoch 207/400
414/414 [==============================] - 0s 347us/step - loss: 0.1637 - acc: 0.9300 - val_loss: 4.0551 - val_acc: 0.3427

Epoch 00207: val acc did not improve from 0.38202
```

# FUTURE WORK

- Try to collect a dataset from scratch which has songs from recent years and those which are popular and collect better labels for the same.
- Test other Natural Language Processing models and approaches to increase validation accuracy.
- Try to use a varied dataset for facial emotion recognition which is more relevant to real-world data.
- Develop an application by combining both the models.