TUNEX

(Tunes for Expressions)

DRISHTI

A Revolutionary Concept

Mentors

Arjun Parmar
Bhavna Matwani
Manav Khorasiya
Meet Soni
Riya Singhal
Parth Garasiya
Yatin Aditya T

Members

Anshoo Rajput
Deepanshu Prasad
Gaurav Kumar
Himanshu Pandey

Acknowledgment

We have made efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are highly indebted to “**DRISHTI-A Revolutionary Concept**” for their guidance and constant supervision and for providing necessary information regarding the project & for their support in completing the project.

We would like to express our gratitude towards our **mentors** for their kind cooperation and encouragement, which helped us complete this project.

We would like to express special gratitude and thanks to industry persons for giving us such attention and time.

Thanks and appreciation to the people who have willingly helped us out with their abilities.

Abstract

Music is a powerful language to express our feelings and, in many cases, is used as therapy to deal with challenging moments in our lives. The music we choose to play can easily reflect our moods and emotions.
We tend to listen to energetic music when physical work is involved but at the same time prefer serene music to calm our anxious minds.
We realize that our moods sometimes unconsciously define the genre of music we listen to.
This system prompts us to explore the therapeutic aspects of music, which improves one's health in cognitive, motor, and emotional domains.

The human face plays a vital role in knowing an individual's mood. Using the camera's feed, one can easily extract the required facial features, hence deducing an individual's mood. We can further utilize this data to classify and tag songs that comply with the "mood" derived from the earlier input.
This system not only eliminates the time-consuming and tedious task of segregating and grouping music but also equips us with music therapy techniques.

Thus, our application focuses on detecting and analyzing human emotions and providing music recommendations based on the user's current mood.

Table of Contents

Workflow and Research

- Our preliminary research work includes reading **Blogs** on Medium and a few **articles** on IEEE to gain a clear insight and understanding of different models and the logic behind their workings.

  All the **research papers** are mentioned in the project's GitHub Repository, which we read initially and took inspiration for starting the project.

- To further prepare ourselves for coding and implementing the emotion detection models and learning in-depth about the neural networks, we all did several courses that are part of **Deep Learning Specialization by Andrew Ng.**

  Those courses were:
    - Neural Networks and Deep Learning
    - Hyperparameter Tuning, Regularization & Optimisation
    - Convolutional Neural Networks (CNN)

- Getting finished with the courses, we moved further on **developing an approach** for how the project will progress.

  We researched and learned about human emotions and how they are related to music genres.

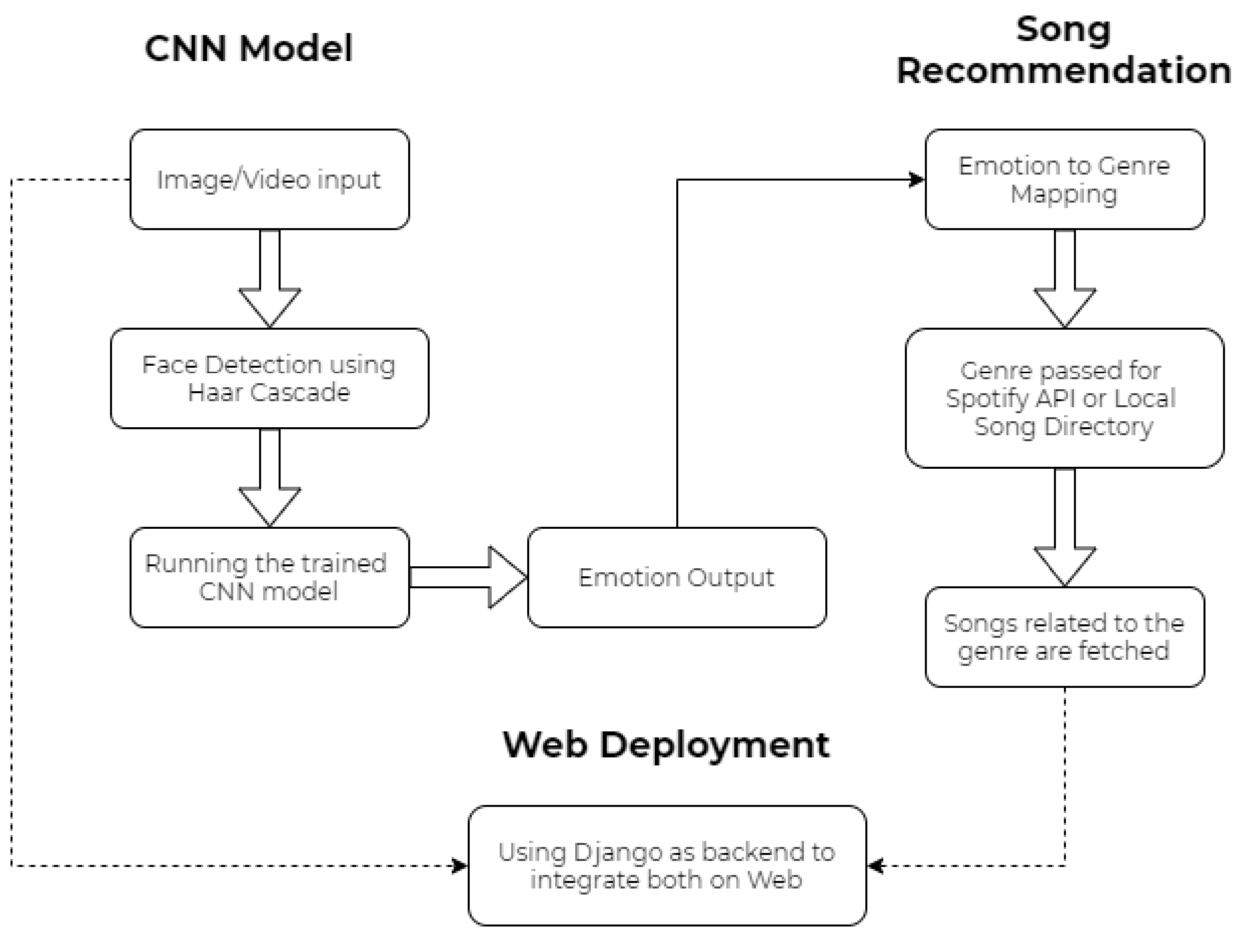
  After brainstorming on this and observing all the available datasets, finally, we decided that we will output the following seven emotions:
    1. Afraid
    2. Angry
    3. Disgust
    4. Happy
    5. Neutral
    6. Sad
    7. Surprised

- Deeply analyzing all the papers, articles, and blogs that we read earlier, we selected some methods which would work for the project and finalized our approach.

- The approach was straightforward for us.
  - First, search for a good, unbiased, and reliable dataset because there were not many available publicly, particularly for this case.
    We also kept an option to create our own dataset by collecting photos from friends and colleagues, in case we did not get sufficient data.
  - After the collection of data, perform augmentation if required, and pre-process the data to feed into the CNN model.
  - Third, learn and implement haar cascading, i.e., detecting the face and facial features from an image, finding the region of interest (ROI).
  - Fourth, train the neural network models and create an architecture with the best possible accuracy. Use of transfer learning if needed.
  - After training, test the model, first on a static image and then on the live video feed.
    Finally, the model will output an emotion.
  - Build a song recommendation system, which will take emotion as input and output a playlist of songs or a Spotify playlist based on genres related to the emotion.
    Finally, integrating this system with the model trained.
  - At last, deploy this whole system on the website.
    Also, learn Django for providing the backend to our website.

- With the approach set in our minds, we started with the implementation of our project.
  During this, we faced many problems and learned many things. We also had to change our approach completely, multiple times.

  These are all mentioned in detail in the upcoming sections.


CNN Model
Song Recommendation
Image/Video input
Face Detection using Haar Cascade
Running the trained CNN model
Emotion Output
Emotion to Genre Mapping
Genre passed for Spotify API or Local Song Directory
Songs related to the genre are fetched
Web Deployment
Using Django as backend to integrate both on Web

Workflow Diagram, explaining the progression of this project.

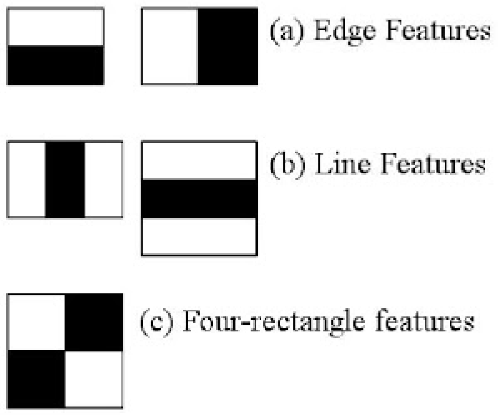
Face Detection

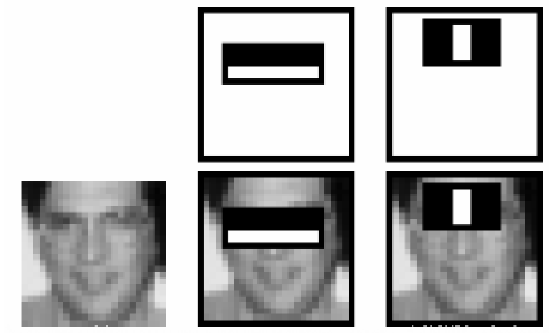
Face detection was the first step towards the recognition of facial emotion.

For this, we need to detect the user's face and make a bounding box on the **Region of Interest**(ROI, i.e., face). We then feed the cropped ROI to our emotion recognition model.
We crop the image for the model to get only the required region, and not the extra noise from the image.

For this purpose, we used the Haar Cascade classifiers of the **OpenCV GitHub repository.**

Haar Cascade is an Object Detection Algorithm used to identify faces in an image or a real-time video. The algorithm uses edge or line detection features on the face. These features on the image make it easy to find out the edges or the lines in the image or pick areas where there is a sudden change in the intensities of the pixels.





Extracting facial features using Haar classifier.

Now, taking an image, if we apply these features even on the subsets of pixels, there will be thousands and lakhs of features. And all of these, just to detect a face! That is why we use the concept of **Cascade of Classifiers.** Instead of applying all features on a window, the features are grouped into different stages of classifiers and applied one by one.

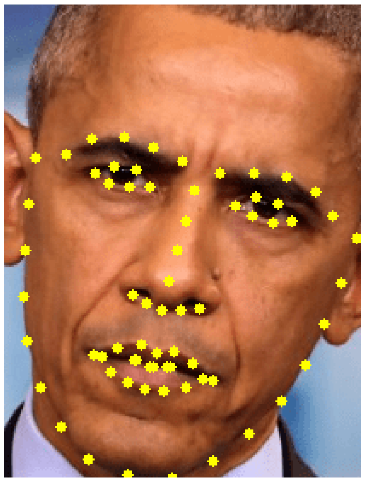
Usually, the first few stages will contain very many fewer features. If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a **face region.**

So this is how the Haar Cascade Classifier works. It is an old method/algorithm but is still relevant and efficient.
And also, this is the one we continued with on this project.

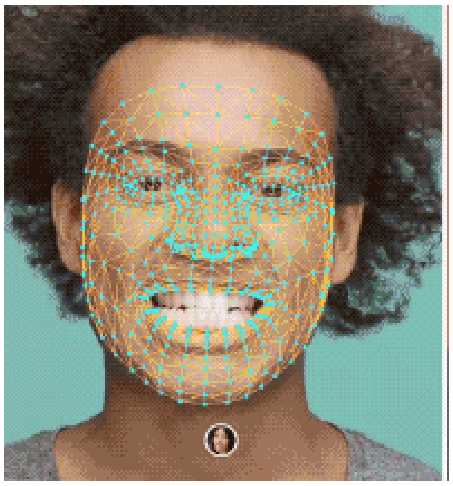
We also tried **68 point facial landmarks** available in **dlib** to annotate the face and feed it directly to the model.
Similarly, we also tried **Face Mesh** from Mediapipe(Google).

Rather than detecting the edges, these methods focus on catching the various points/coordinates on the faces. These points together define many vectors, distance, and geometries. Using those, we can get to analyze the formations and shapes of Eyes, Lips, Teeth, Glabellar lines, and other facial features.

We had to reject the **dlib** and **Mediapipe** methods later, as the model's accuracy was not satisfactory.



68 point landmark detection (Dlib)



Face mesh(Media pipe)

Data Collection and Preprocessing

Collecting a reasonable and unbiased Dataset was a tedious job. As we had already decided that we were going to work with seven labels, i.e., seven emotions, we required a dataset according to that.
After researching the internet and reading several research papers, we found many datasets, ranging from 1000 samples to 5 million. Many among them were either private to company, paid, or contained no labelings, and in some cases, the dataset was biased towards one or two emotion categories. Some datasets even consisted of paintings, images of cartoons, and all kinds of weird things that resemble a face. And according to our requirements, i.e., detecting the emotion of an actual human through a webcam, we did not need those datasets.

So, continuing with the search for the best suitable dataset that is also public to use, we stopped on **KDEF & AKDEF Facial Expression Dataset**.

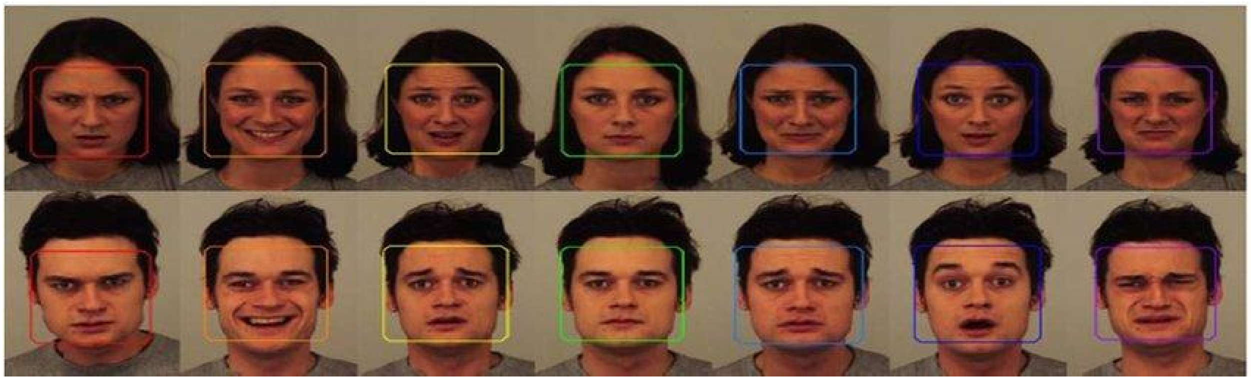
The **Karolinska Directed Emotional Faces (KDEF)** is a set of a total of 4900 pictures of human facial expressions. The collection of images contains 70 individuals displaying seven different emotional expressions. They shot each expression from 5 different angles.

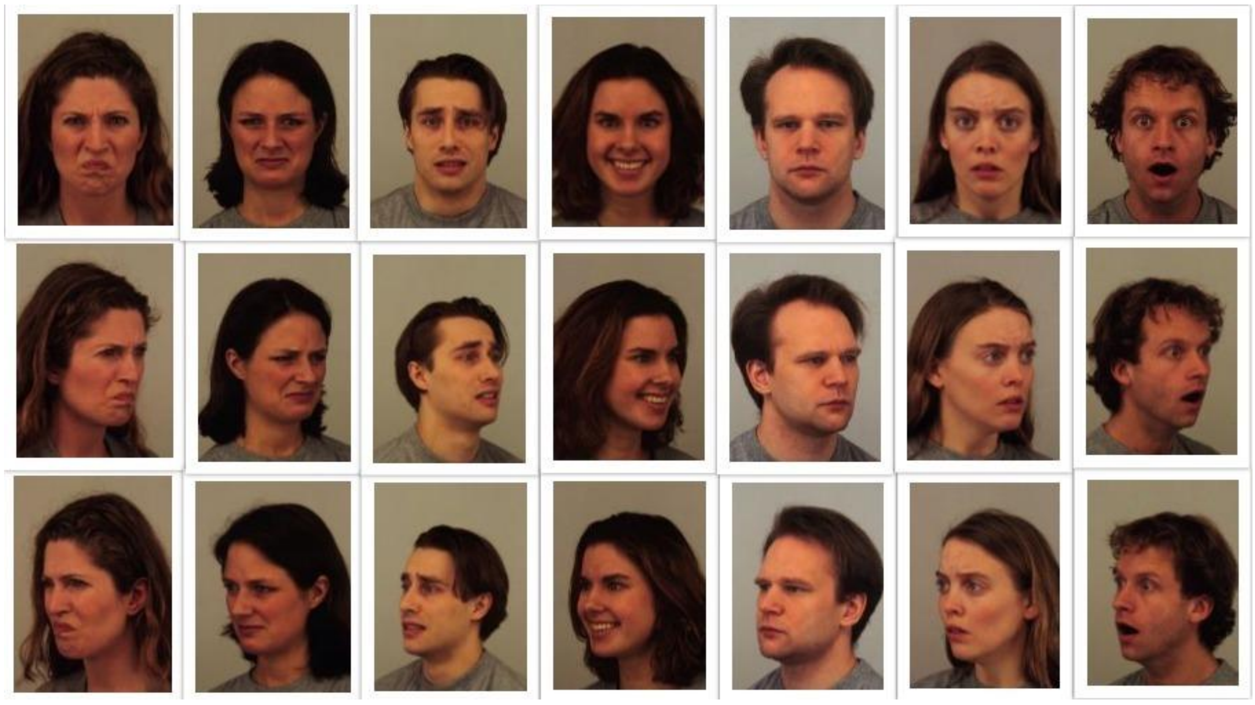
These images are of 70 amateur actors, 35 females and 35 males. Age between 20 and 30 years of age. No beards, mustaches, earrings, or eyeglasses, and also no visible make-up during photo-session.

The subjects were asked to rehearse the different expressions for 1 hour before coming to the photo session. They emphasized that the subject should try to evoke the emotion that was going to be expressed. While maintaining a way of expressing the sentiment that felt natural to them, try to make the expression solid and transparent.

All subjects wore unique gray T-shirts. They were seated at a distance of approximately three meters from the camera. The absolute distance was adapted for each subject by adjusting the camera position until the subject’s eyes and mouth were at specific, pre-defined vertical and horizontal positions on the camera’s grid screen. The lights were set to cast a soft indirect light evenly distributed at both sides of the face. After a rehearsal session, the subjects were shot in one expression at a time until they had shot all seven expressions.

These are a few examples of how all the images in our dataset look.





As per the original dataset, we had 4900 images available with us, equally distributed among all the seven emotion categories. Compared to our estimate of 15000-20000(which we considered being ideal in our case), the number was significantly less.

Therefore, now we needed to perform Data Augmentation. But before that, another problem needed to be solved. Each individual in this dataset had shot photos on five different angles. Out of which, we did not require the absolute left and the absolute right faced images. First, we eliminated all those and were left with 2940 samples, which was even lesser.

To perform the augmentation, we had two ways: One, use the inbuilt function of the Keras library. Two, use the albumentation library of python, which makes it easy to augment images. We used both of them but finally considered the one performed with albumentation.

Following were the augmentations we used:

1. Rotate 15 degrees right
2. Rotate 15 degrees left
3. Rotate 30 degrees right
4. Rotate 30 degrees left
5. Horizontal Flip
6. Random Brightness and Contrast

This gave us a total of 17640 sets of images in our dataset, each category containing 2520 images. We continued with this data on the project.

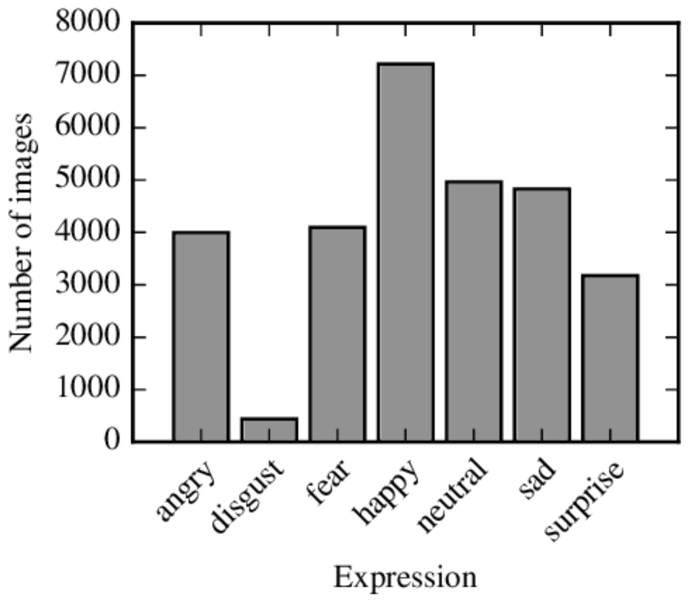
Later, after training the model on several basic and advanced architectures, we didn't get the desired accuracy on test images and the live video feed. Hence, this dataset was rejected.

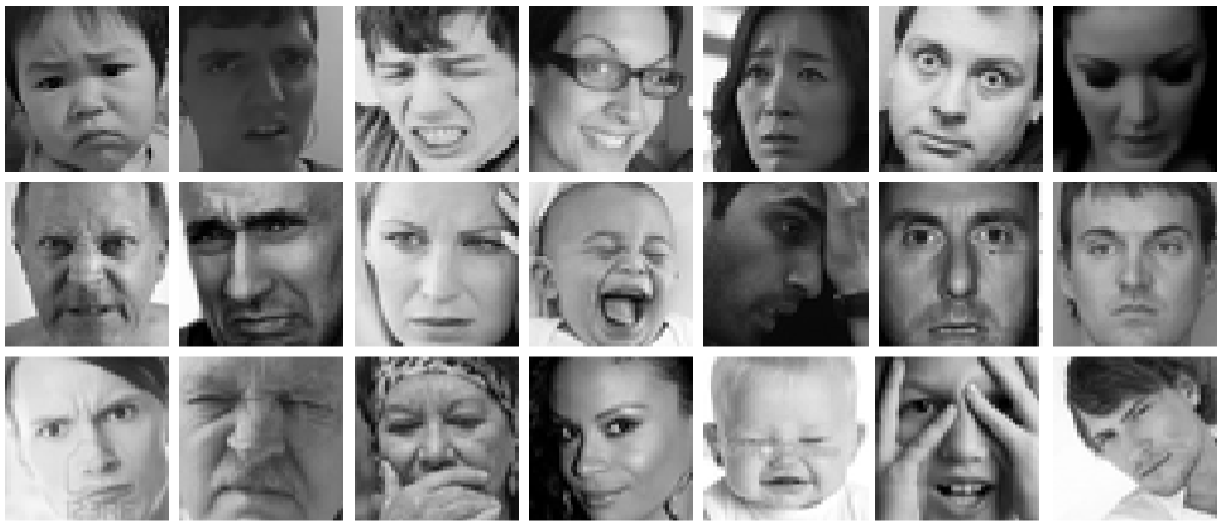
The reason can be stated that this dataset does not have much variety and is not relevant to the Indian situation. For e.g., most of the individuals were of European descent. They didn't keep any beard or mustache. These facial features are prevalent in India's scenario. And there could also be a possibility that the expressions given by these actors are not generally found in the actual situation.

Failing to get accuracy on the KDEF data, we then moved on to a new one, the FER-2013 dataset available on Kaggle.

The data consists of 48x48 pixel grayscale images 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. The training set consists of 28,709 examples, and the public test set consists of 3,589 samples.

This dataset has a substantial bias towards some categories. Still, we went with this one, as it gave better accuracy on the live test than the KDEF data.

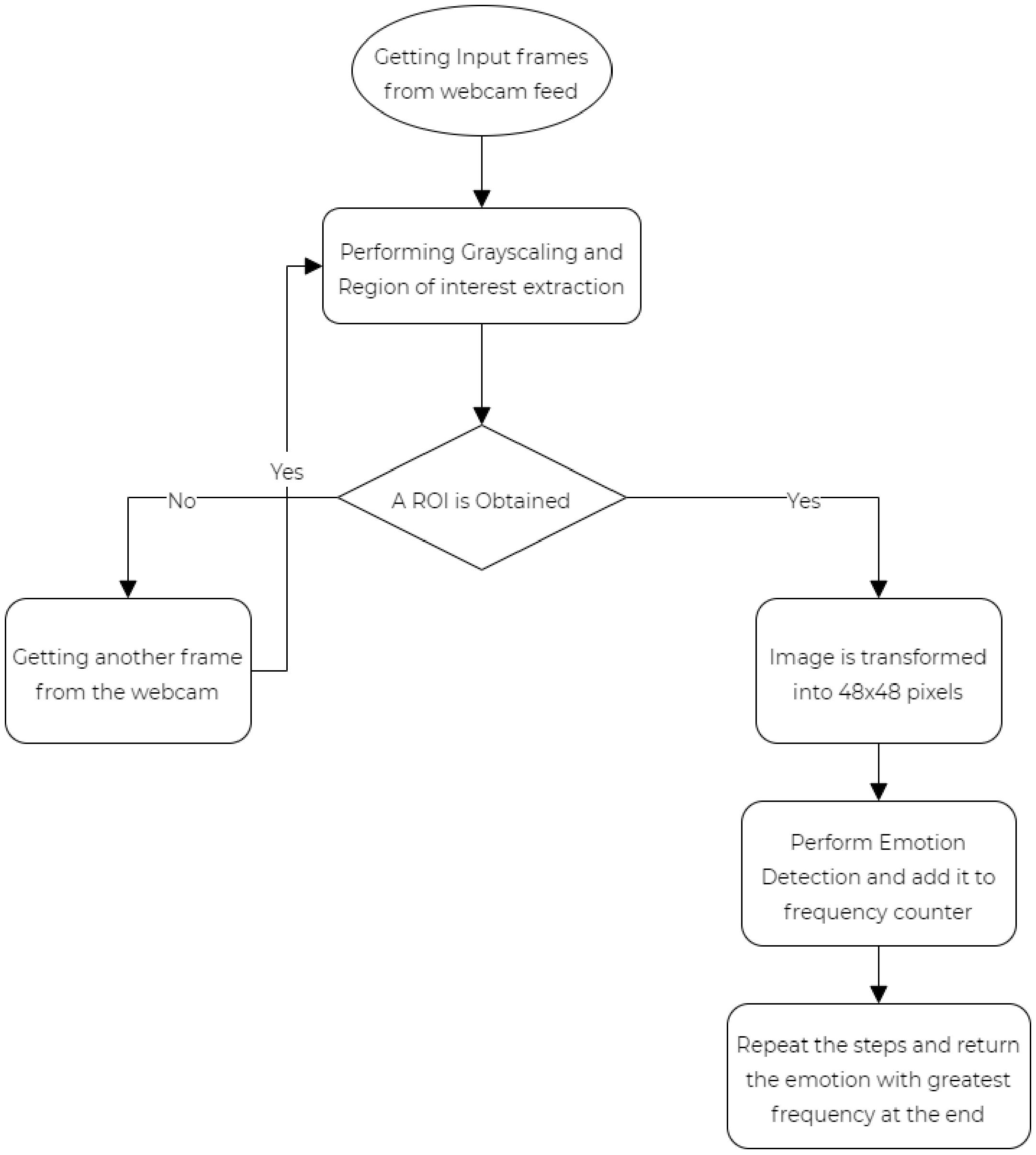




Emotion Recognition Models

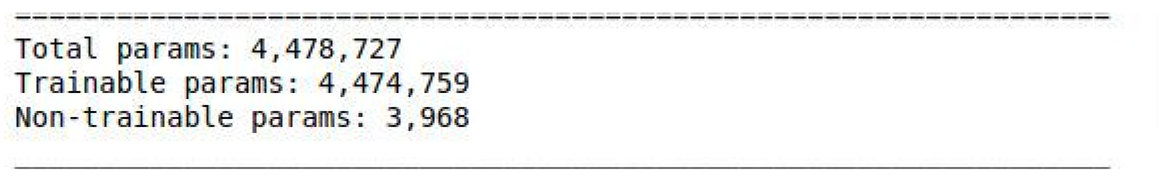
Since, emotion recognition is being performed on the basis of facial recognition we hence choose to use a Convolutional Neural Network as the initial layers and after a few blocks of convolution we are flattening the layer and taking it as an input for a neural network and at the end there is a softmax layer for classification of emotion into one of the seven categories. The intuition behind this model architecture can be derived from the definition of Convolutional Neural Network which is generally used as the first layer of model and is consistent with the input image shape of nxmx3, where nxm is the pixel of image and 3 represents the RGB channels. Imagine that the reading of the input matrix begins at the top left of the image. Next the software selects a smaller matrix there, which is called a filter (or neuron, or core). Then the filter produces convolution, i.e. moves along the input image. The filter’s task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end. Since the filter has read the image only in the upper left corner, it moves further and further right by 1 unit performing a similar operation. After passing the filter across all positions, a matrix is obtained, but smaller than an input matrix.

The network will consist of several convolutional networks mixed with nonlinear and pooling layers. When the image passes through one convolution layer, the output of the first layer becomes the input for the second layer. And this happens with every further convolutional layer. The nonlinear layer is added after each convolution operation. It has an activation function, which brings nonlinear property. Without this property a network would not be sufficiently intense and will not be able to model the response variable (as a class label). The pooling layer follows the nonlinear layer. It works with width and height of the image and performs a downsampling operation on them. As a result the image volume is reduced. This means that if some features (as for example boundaries) have already been identified in the previous convolution operation, than a detailed image is no longer needed for further processing, and it is compressed to less detailed pictures. After completion of series of convolutional, nonlinear and pooling layers, it is necessary to attach a fully connected layer. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the amount of classes from which the model selects the desired class.

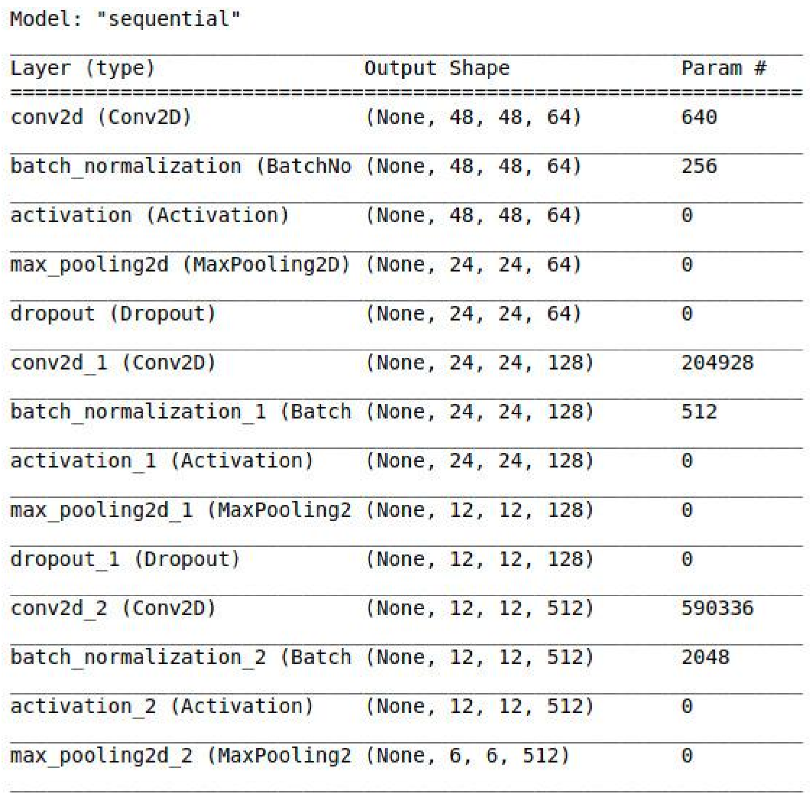


Getting Input frames from webcam feed
Performing Grayscaling and Region of interest extraction
A ROI is Obtained
No
Yes
Yes
Getting another frame from the webcam
Image is transformed into 48x48 pixels
Perform Emotion Detection and add it to frequency counter
Repeat the steps and return the emotion with greatest frequency at the end

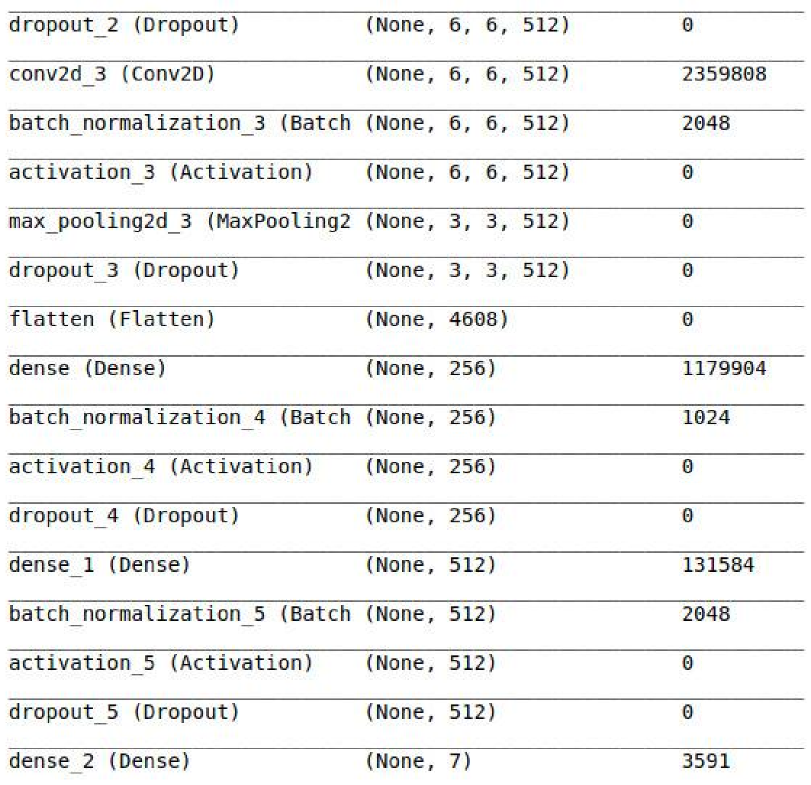
These were our clear motives behind selection of a CNN based model, and after this each team member started trials with different models. Training and validation accuracy were theoretically good (all of us were getting above 95% accuracy) initially with the first simple CNN model but when tested on live feed all the predictions were giving emotion as "neutral". We analysed the training graph and found that our trained models were overfitting and hence we started trying the ways we could implement the changes. We then added dropout layers and reduced the density of our network but outcomes didn't change much and we then started working with pre-trained models and trying transfer learning. We tried out resnet50, VGG16, Xception and Mobilenet models, for our dataset. But the accuracy on some of these models were very low, such as after many attempts and trials VGG16 gave an test accuracy of 45%, resnet was giving greater accuracy of around 90% on test, but was again inefficient on testing. At this point we tried a different dataset and instead of KDEF we used FER dataset, because we thought maybe the lack of Indian samples from KDEF dataset might be a possible reason for this inefficiency. Hence, we again trained all the past models on FER dataset and the accuracy using a simple CNN model was although 65%, giving a much better result when being tested on web feed. Thus, we decided to stick with this model and moved forward with the Song Recommendation part.


```
=================================================================
Total params: 4,478,727
Trainable params: 4,474,759
Non-trainable params: 3,968
_________________________________________________________________
```

Model Architecture

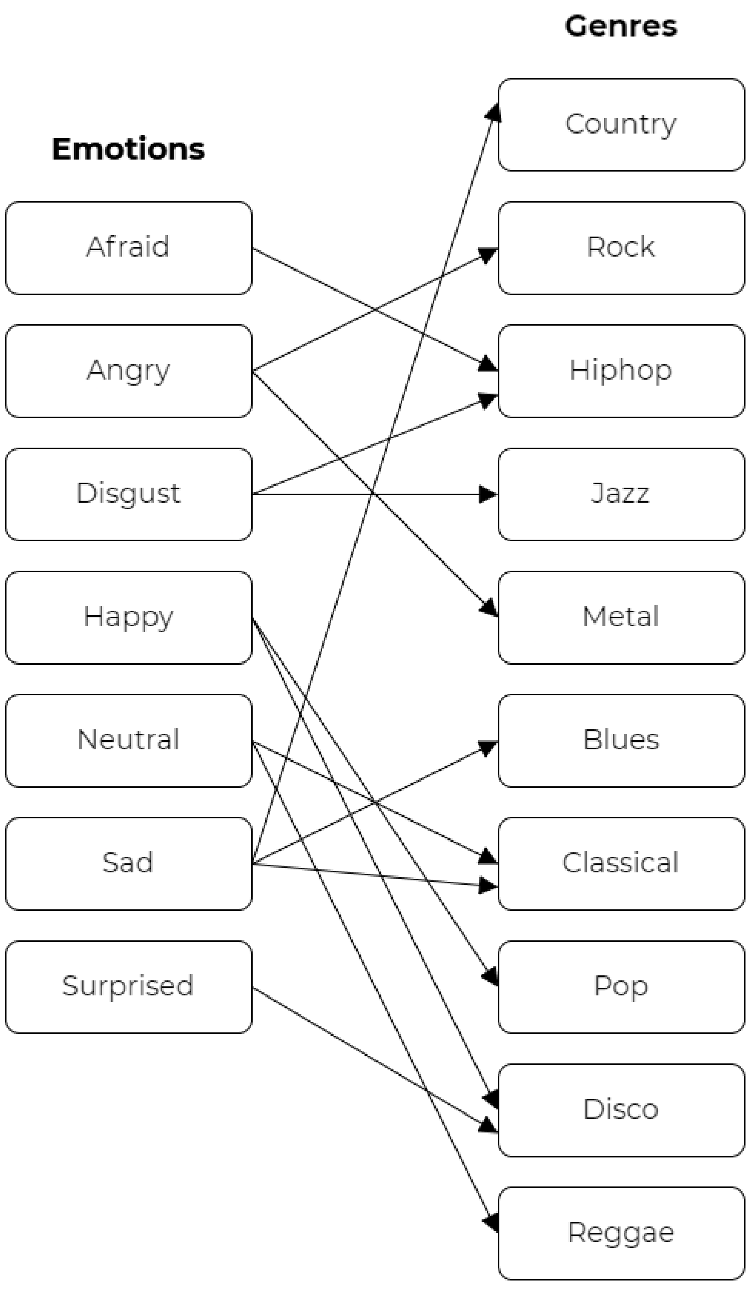


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 64)        640
_________________________________________________________________
batch_normalization (BatchNo (None, 48, 48, 64)        256
_________________________________________________________________
activation (Activation)      (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 128)       204928
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
activation_1 (Activation)    (None, 24, 24, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 512)       590336
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 512)       2048
_________________________________________________________________
activation_2 (Activation)    (None, 12, 12, 512)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
```



```
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 512)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 512)         2359808
_________________________________________________________________
batch_normalization_3 (Batch (None, 6, 6, 512)         2048
_________________________________________________________________
activation_3 (Activation)    (None, 6, 6, 512)         0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 512)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 3, 3, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1179904
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
activation_4 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               131584
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 3591
```

Integration with Song Recommendation



Mapping Emotions to the Genres

After building the model that finally outputs an emotion out of seven, the next task was to define genres that will correspond to each emotion. Above is the diagram, mapping each emotion with a specific genre.

Since the genres were now defined, we then started downloading songs that were considered best in their respective genres. All the songs were arranged in their respective genre folders, and the folders were mapped to the emotions, as per the diagram above.

So, after the model outputs an emotion, it is then given as an input to a **recommendation** function. This function then randomly selects a song from the specified genre folder and plays it.

Web Deployment (Locally)

For Deployment we had the option between two popular frameworks i.e flask and django.

What is Flask?

Flask is an micro framework offering basic features of web app. This framework has no dependencies on external libraries. The framework offers extensions for form validation, object-relational mappers, open authentication systems, uploading mechanism, and several other tools.

What is Django?

Django is a web development framework for Python. This framework offers a standard method for fast and effective website development. It helps you in building and maintaining quality web applications. It enables you to make the development process smooth and time-saving.

It is a high-level web framework which allows performing rapid development. The primary goal of this web framework is to create complex database-driven websites.

Which is Better?
You should prefers flask if you want the granular level of control while a Django developer relies on an extensive community to create unique website.

Django combined with the REST Framework helps you to build powerful APIs, whereas Flask requires more work, so there are high chances to make a mistake.

The best method is to build a few basic CRUD apps with both frameworks and decide which framework fits your project style better.

Our team explored the differences between the django and flask and after discussion on which framework should be used we decided to go with django.

During deployment phase, there was also an issue of providing good recommendations. To overcome this we tried different approaches like trying to classify songs locally stored in a device and also explored the possibility of hosting few songs in the cloud. But all of this methods seemed time consuming to the song recommendation phase. So we decided to use the Spotify APIs for this purpose.

Spotify APIs

Based on simple REST principles, the Spotify Web API endpoints return JSON metadata about music artists, albums, and tracks, directly from the Spotify Data Catalogue.

Web API also provides access to user related data, like playlists and music that the user saves in the Your Music library. Such access is enabled through selective authorization, by the user.

The base address of Web API is https://api.spotify.com. The API provides a set of endpoints, each with its own unique path. To access private data through the Web API, such as user profiles and playlists, an application must get the user’s permission to access the data. Authorization is via the Spotify Accounts service.

Using a pre existing service posed a challenge of having to accept user’s data to allow the app to recommend the songs, So we decided to use such a service which would just require an authorization token and would not need the user to authorize our application. So we used Get Recommendations Based on Seeds feature provided by the spotify APIs. Only downside is that we have to update the Token after its expiry period.

Problems Faced and Solutions

Although we tried to work on this project with a tremendous amount of research, we still felt that there are a few shortcomings that we could improve in future editions of the project in general and the entire Deep Learning algorithm in particular.

Some of them will be:

- **Improving Model Architecture:**

  In the end, we somehow had to work with just the basic CNN model with a 65% validation accuracy.
  Reasons for this were a slight bit lack of time, and computational resources, as we were unable to train other models locally or even in google colab due to session crashes.

  So, we believe there should definitely be a better architecture that can achieve better accuracy, and hence it will be a good shot to try those.

- **Music Recommendation System:**

  A recommender system can improve the User experience rather than just listening to a randomly chosen song from the genre we predict.

  Also, we did the mapping of genres with emotion for fewer genres than what exists in reality.

- **Spotify or other API's integration:**

  The current version of Spotify API we tried was paid and had fewer request limits than the dev version; hence the full deployment of the web page was hampered, and thus it leaves a scope of improvement in the future.

- **Facial Landmarks unutilized:**

  Although we tried utilizing them in various models, there were still many problems when making predictions due to unbalanced output from the pre-existing models. Hence we ran a CNN on the simple unmarked face; thus, Facial landmarks can be a way to check for other be-fitting models if utilized.

- **Streaming webcam feed:**

  We need to take a pause of 10 seconds to capture the user’s emotion. Multi-processing can possibly remove this gap.

- **Other Emotion detection techniques:**

  Facial emotion detection is a good thing, but we still need to consider our user’s situation, so people cannot show emotions on their face. Also, it may be annoying for someone to bring the camera in front of them after every song. Therefore, we can utilize the other ways such as pulse rate or sentiments from the user’s texts for this purpose, and we might achieve a better result.

Conclusion

Selection of a proper dataset not just includes selecting the one with many samples, but also the ones that have many diverse samples, because it may end up being a cause for bias in the model while we classify or make predictions. Giving proper time to this process, in the beginning, is very important and indeed saves a lot of time while working on the actual model. Also, the default hyperparameters need to be explored before we start training our model as they will not be good enough for every model requirement and use case. Instead of changing them after hours of training the model, it is better to check if they will be able to match our requirements, at the start.

It is also essential to have consistent work on any project after it has started because a large number of small breaks in between result in a lot of inefficiencies, and people tend to start losing the motivation behind the thing. It is much better if the team stays together and works on the project consistently with fewer breaks and in an offline mode. Communication is a very tough task to achieve when things are Online, and you can see each other just online. We will surely take note of these points when working on a project afterward, as these are the things that determine the success or failure of the project.

It was an excellent experience for us working on this project. We learned a lot about deep learning models, building them from scratch, and using Keras to set them up. We also got a great insight on working with Image Processing tasks as our project revolves around images and emotion detection a lot. Also, the intuition behind these models and the convolutional process as a whole was something that really excited us, and it filled us with joy to see and learn a lot of new things through this project. Hyper-parameter tuning, avoiding overfitting, using dropout for the same, and efficiently determining the project’s goal before starting it are some of the key takeaways from this project.

Bibliography

We went through a lot of materials in the form of blogs and research papers and hence we would like to mention a few of notable once here:

- https://ieeexplore.ieee.org/document/7753378 - Emotion Detections using facial expression recognition and EEG
- https://www.sciencedirect.com/science/article/pii/S235291482030201X - Real Time Emotion Recognition System using Facial Expressions and EEG
- B2b.musicovery.com- Music Recommendation System
- https://ieeexplore.ieee.org/abstract/document/8299738 - Smart Music Player Integrating facial emotion recognition and music mood recommendationBibliography
- https://www.researchgate.net/publication/325674764_Facial_Expression_Recognition_Using_Facial_Landmarks_and_Random_Forest_Classifier - Facial Expression Recognition using Facial Landmarks and Random Forest Classifier
- https://arxiv.org/pdf/1812.04510.pdf - Facial Expression Recognition using Facial Landmark Detection and Feature Extraction via Neural Networks
- https://www.ri.cmu.edu/pub_files/2015/5/intraface_final.pdf - Intraface model to extract features
- https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47- SVM model
- https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn - XGB Model