PREFACE

This report is on “Mood Based Recommendation System”, which delivers media recommendations to a user based on their mood. This project was based on the need for the inclusion of Artificial Intelligence in our day-to-day lives thus making life easier. We are constantly recommended different products, movies and music whenever we use different web based services and thus the need for our project arises. To recommend pictures, music and movies to a person based solely on their mood.

Artificial Intelligence has growing applications of technology. We see the inclusion of Machine Learning online to recommend advertisements to users, for curating playlists based on listening history, etcetera, we went on to developing a recommendation system that delivers media based on the mood of the user, that is recognised by the computer. People love recommendations, and we made use of this to develop our prototype application. This led us to developing a mood based recommendation system that recommends media to a user based on their mood.

ABSTRACT

This project was based on the need for the inclusion of Artificial Intelligence in our day to day lives, to ease the life of people. It recommends pictures, music and movies to a person based on their mood. The prototype recommends media, that is, pictures, songs or movies based on the mood of the individual. The mood is recognised by facial expression recognition done by the computer.

Today, people want recommendations, may it be music, a restaurant, or even a movie, people love recommendations on the go. We thought of developing a media recommendation system, that recommends media to users based on their facial expression which translates to their mood. A webcam is used to take an image of the user's face once the user presses a key, that is recognised by the computer algorithm. This image that is captured is then fed to a machine learning model that has been trained by us on the fer2013 dataset.[1] The model employs a DenseNet121[2] convolutional neural network architecture. This model provides an output of the mood of the user using facial expression recognition. The model currently recognises five moods, that are, calm, angry, sad, happy and surprised. The output is then used to recommend relevant playlists, movies or pictures to the user.

The final model that we've trained has an accuracy of 76.18% and employs DenseNet121 architecture for the recognition of facial expression and translating it to a mood. The classification accuracy is high for a vision computing algorithm and thus predictions of mood are very accurate.

1 I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, and Y. Bengio, "Challenges in representation learning: A report on three machine learning contests,"*Neural Networks*, vol. 64, pp. 59–63, 2015. (arXiv:1307.0414)

2 Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, "Densely Connected Convolutional Network," *last revised 28 Jan 2018(v5).* (arXiv:1608.06993)

TABLE OF CONTENTS

1. INTRODUCTION

The face of a person conveys a magnitude of different expressions that make up what we perceive to be his mood at that given time. To a large extent, we can accurately describe what the mood of the person is at that given time through his facial expressions.[3] Thus, he can have a wide range of emotions like happy, angry, sad or suprised. In addition to this, he may have a neutral expression for which we can attribute it to being calm. Therefore, based on his emotions we communicate with that person accordingly. If one is sad we try and cheer the person so that they can get out of this "negative mood." And, when one is happy, we join in on their happiness and thus, we humans have a tendency to alleviate the sads from the world and bring joy into the lives of others.

Using this philosophy, we can train machines to do this exact same thing, albeit not in the same manner as a real person can. However, with the growing impact of technology and the relevance and important it has on one's life, the time is not far when machines will replace dogs to be the man's best friend. Based on these trends, there have been a tremendous amount of research and development being done in the field of computer vision and machine learning. Computer vision, is the ability of a computer or machine to see, detect and perceive objects that we humans can see. The computer has superior computational capabilities to extract, process and analyze digital images. Thus machines can technically have the same functionality as that of humans as long as we can train them. This is where Machine Learning comes into play. Once the computer has been able to process the information, we can train it to become better. By providing the computer relevant information in the form of data, we are able to train it to learn a new object, in this case the facial emotions of a person.

A successful integration of the two enables a machine to use the application to successfully recommend content deemed fit by the creator and thus can be used to essentially become a one-stop place for a person to get his daily dose of media according to his mood. The functionalities can further be used to curate content that is highly individualistic in nature and serves an integral part in his or her lives.

1.1. PROBLEM STATEMENT

The purpose of building a mood based recommendation system is so that it can serve as a one-stop place for an individual to get highly curated media content that is based on his real-time

[3] Knutson, B. J Nonverbal Behav (1996) 20: 165. https://doi.org/10.1007/BF02281954

mood. Thus, this serves to be a platform for a person to get on-demand movies, images and music suggestions that would be highly individualistic and unique. As we are aware, in today's world, personalization is the key driving factor behind every company. The internet is a pool for vast amounts of user-generated data that companies are using to provide recommendations. These recommendations are used in the form of market products to drive sales of the company. Companies are able to classify individuals into different groups and thus they are able to map them successfully.

Music listeners have tough time creating and segregating their playlist manually, especially since they have hundreds of songs available to them. It is also difficult to keep track of all the songs: songs that are added are often never played again, thus, this leads to a burden on the device and reduction in available memory. Therefore, users are forced to find and delete these songs. This leads to a waste of time for the individual. Instead, our system provides the user a highly personalized and curated playlists based on their mood.

A detailed Literature study can be found below where we have studied the different convolutional neural network architectures and the language used to code this prototype.

2. LITERATURE SURVEY

Literature survey includes the detailed study of all the existing technologies and systems which have been used for real-time viewing, detection and capture of the picture and finally processing it to get an output in the form of its emotions and mood. Further, this mood allows the user to access different recommended media content that has been curated for them. We have implemented the above using different technologies like computer vision and image processing using machine learning. The literature survey also includes the study on various different algorithms, neural networks and programs that have been used.

2.1. STUDY OF EXISTING SYSTEMS

A detailed study of the existing system which translates the image to mood. We have successfully incorporated different techniques and technologies. These have been stated, studied and documented below.

2.2. PYTHON 3

Python[4] is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until July 2018. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python features a comprehensive standard library, and is referred to as “batteries included.” Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open-source software and has a community-based development model. Python and CPython are managed by the non-profit Python Software Foundation.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution. Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable

4 G. van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

interfaces to existing applications. Thus, Python supports a wide range of libraries. All in all, python is fast, extensive and open-source.

2.3. OPENCV LIBRARY

OpenCV[5] (Open Source Computer Vision Library) is released under a BSD license and hence it is free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous computing platform. We have made use of the Python based library for our project.

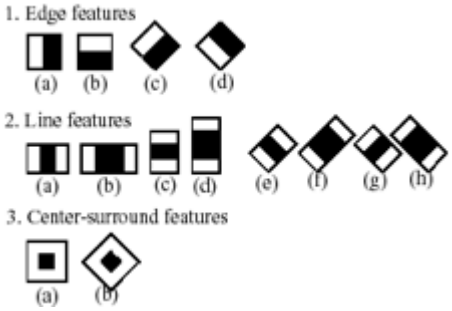
OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe. It is aimed at making computer vision accessible to programmers and users in the area of real-time human-computer interaction and mobile robotics. Thus, the library comes with source code and hand-tuned assembly language binaries optimized for Intel processors, so that users can both learn from the library and make use of its performance. OpenCV addresses the areas of object/human/face segmentation, detection, recognition, and tracking, as well as camera calibration, stereovision, and 2D/3D shape reconstruction. A full matrix algebra package is also included in the library to support algorithms in these areas. As computers increase in performance and cameras decrease in cost, tools such as OpenCV will help set the infrastructure in place for new ways of interacting with computing devices, especially as computers move into set-top boxes, handhelds, projection screens, and wearable computers.

2.4. HAAR CASCADES

A Haar Cascade[6] is basically a classifier which is used to detect the object for which it has been trained for, from the source. The Haar Cascade is the process of superimposing the positive image over a set of negative images. The training is generally done on a server and on various stages. Better results are obtained by using high quality images and increasing the amount of stages for which the classifier is trained. One can also used predefined Haar Cascades which are available on Github. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

5 Gary Bradski in Dr. Dobbs Journal, 2000, The OpenCV Library (https://github.com/opencv/opencv)

6 Paul Viola and Michael Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, 2001



Here we work with face detection using a 'frontalface_alt2' classifier. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, haar features shown in below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

2.5. IMAGEAI LIBRARY

ImageAI[7] is an easy to use Computer Vision Python library that empowers developers to easily integrate state-of-the-art Artificial Intelligence features into their new and existing applications and systems. It is used by thousands of developers, students, researchers, tutors and experts in corporate organizations around the world. Built with simplicity in mind, ImageAI supports a list of state-of-the-art Machine Learning algorithms for image prediction, custom image prediction, object detection, video detection, video object tracking and image predictions trainings. ImageAI currently supports image prediction and training using 4 different Machine Learning algorithms trained on the ImageNet-1000 dataset. They are ResNet, DenseNet, SqueezeNet and InceptionV3.

This library uses Tensorflow Machine Learning library as the backend and Keras for the front end computation. This library takes the input train and test sets as images.

2.5.1. TENSORFLOW LIBRARY

TensorFlow[8] is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine

7 John Olafenwa and Moses Olafenwa, Introduction to Deep Computer Vision, 2018

8 Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike

learning applications such as neural networks. TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy. Using Tensorflow, one can build and train state-of-the-art models without sacrificing speed or performance. TensorFlow gives you the flexibility and control with features like the Keras Functional API and Model Subclassing API for creation of complex topologies. For easy prototyping and fast debugging, use eager execution. Tensorflow forms the backend of the ImageAI library.

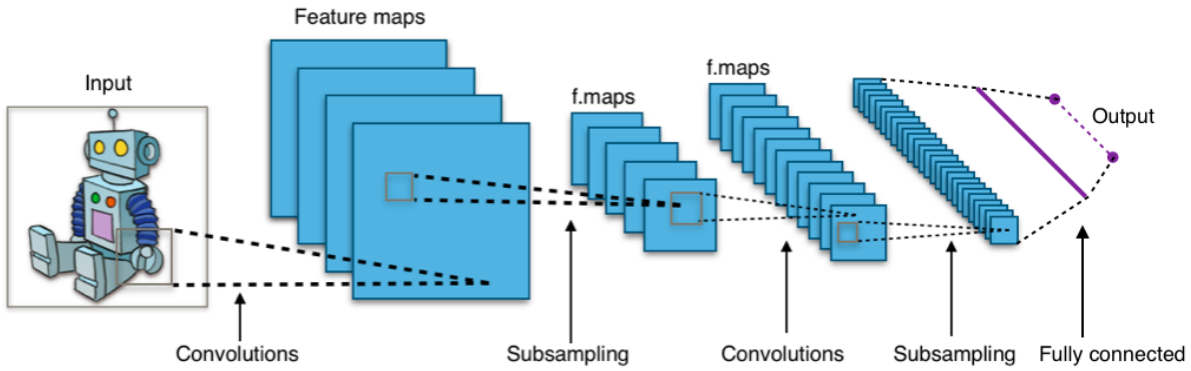
2.5.2. KERAS LIBRARY

Keras[9] is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model. In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine-learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0. By default, Keras will use TensorFlow as its tensor manipulation library. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

9 Chollet, F. (2015) keras, GitHub. https://github.com/fchollet/keras

2.6. CONVOLUTIONAL NEURAL NETWORKS



Convolutional Neural Network Architecture

In deep learning, a convolutional neural network[10] (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layer i.e. activation function, pooling layers, fully connected layers and normalization layers. Description of the process as a convolution in neural networks is by convention. Mathematically it is a cross-correlation rather than a convolution (although cross-correlation is a related operation). This only has significance for the indices in the matrix, and thus which weights are placed at which index.

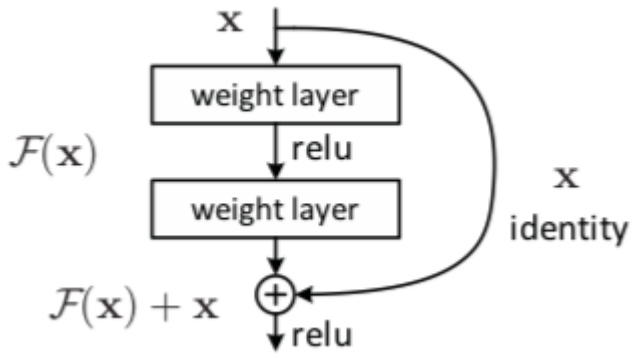
Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. Convolutional neural networks usually require a large amount of training data in order to avoid overfitting. A common technique is to train the network on a larger data set from

[10] M. D. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Neural Networks, In ECCV, 2014

a related domain. Once the network parameters have converged an additional training step is performed using the in-domain data to fine-tune the network weights. This allows convolutional networks to be successfully applied to problems with small training sets.

CNNs are often used in image recognition systems. When applied to facial recognition, CNNs achieved a large decrease in error rate. Even with this, the best algorithms still struggle with objects that are small or thin, such as a small ant on a stem of a flower or a person holding a quill in their hand. They also have trouble with images that have been distorted with filters, an increasingly common phenomenon with modern digital cameras. By contrast, those kinds of images rarely trouble humans. Humans, however, tend to have trouble with other issues. For example, they are not good at classifying objects into fine-grained categories such as the particular breed of dog or species of bird, whereas convolutional neural networks handle this. In 2015 a many-layered CNN demonstrated the ability to spot faces from a wide range of angles, including upside down, even when partially occluded, with competitive performance. The network was trained on a database of 200,000 images that included faces at various angles and orientations and a further 20 million images without faces. They used batches of 128 images over 50,000 iterations.

2.6.1. RESNET50 ARCHITECTURE



ResNet Building Block

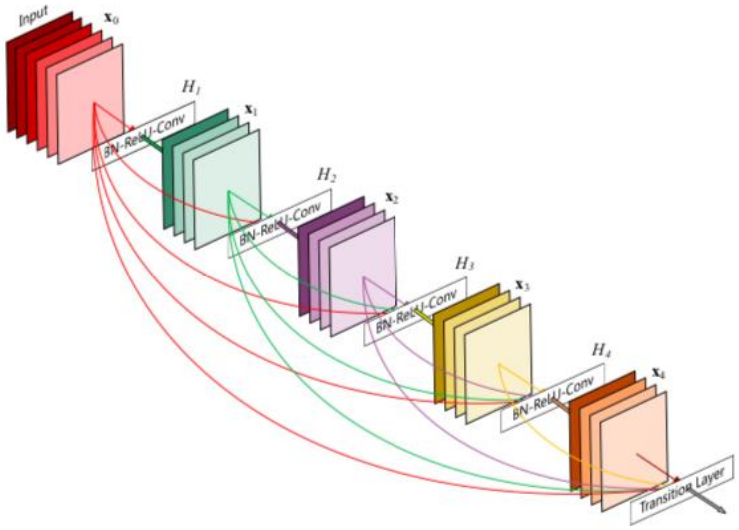
ResNet[11] is a short name for Residual Network. As the name of the network indicates, the new terminology that this network introduces is residual learning. Deep convolutional neural networks have led to a series of breakthroughs for image classification. Many other visual

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, "Deep Residual Learning for Image Recognition," 2015 (arXiv:1512.03385)

recognition tasks have also greatly benefited from very deep models. So, over the years there is a trend to go more deeper, to solve more complex tasks and to also increase/improve the classification/recognition accuracy. But, as we go deeper; the training of neural network becomes difficult and also the accuracy starts saturating and then degrades also. Residual Learning tries to solve both these problems. In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low, mid or high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet does this using shortcut connections (directly connecting input of nth layer to some (n+x)th layer. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved. This is the fundamental concept of ResNet.

ResNet50 is a 50 layer Residual Network. There are other variants like ResNet101 and ResNet152 also.

2.6.2. DENSENET121 ARCHITECTURE



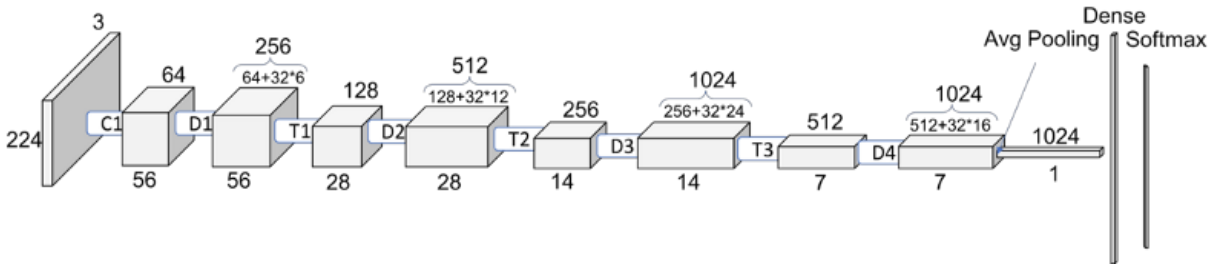
DenseNet Architecture, 5-layer dense block

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and

those close to the output. In DenseNet[12] architecture, each layer is connected to every other layer in a feed-forward fashion. For each layer, the feature maps of all preceding layers are treated as separate inputs whereas its own feature maps are passed on as inputs to all subsequent layers. This connectivity pattern yields state-of-the-art accuracies on CIFAR10/100 (with or without data augmentation) and SVHN. On the large scale ILSVRC 2012 (ImageNet) dataset, DenseNet achieves a similar accuracy as ResNet, but using less than half the amount of parameters and roughly half the number of FLOPs. Counter-intuitive effect of this dense connectivity pattern is that it requires fewer parameters than traditional convolutional networks, as there is no need to relearn redundant feature maps. Traditional feed-forward architectures can be viewed as algorithms with a state, which is passed on from layer to layer. Each layer reads the state from its preceding layer and writes to the subsequent layer. It changes the state but also passes on information that needs to be preserved.

DenseNet-121 Dx: Dense Block x. Tx: Transition Block x.

Concatenating feature maps learned by different layers increases variation in the input of



subsequent layers and improves efficiency. This constitutes a major difference between DenseNets and ResNets. Compared to Inception networks [35, 36], which also concatenate features from different layers, DenseNets are simpler and more efficient. Bottleneck layers. Although each layer only produces k output feature maps, it typically has many more inputs. It has been noted in [36, 11] that a 1×1 convolution can be introduced as bottleneck layer before each 3×3 convolution to reduce the number of input feature-maps, and thus to improve computational efficiency. We find this design especially effective for DenseNet and we refer to our network with such a bottleneck layer, i.e., to the BN-ReLU-Conv(1× 1)-BN-ReLU-Conv(3×3) version of H`, as DenseNet-B. Unless otherwise specified, each 1×1 convolution reduces the

[12] Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Laurens van der Maaten and Kilian Q. Weinberger, "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition," 2017 (arXiv:1608.06993)

input to 4k feature-maps in all experiments. Compression. To further improve model compactness, we can reduce the number of feature-maps at transition layers. If a dense block contains m feature-maps, we let the following transition layer generate bθmc output feature maps, where 0 <θ ≤1 is referred to as the compression factor. When θ = 1, the number of feature-maps across transition layers remains unchanged. We refer the DenseNet with θ <1 as DenseNet-C, and we set θ = 0.5 in our experiment. When both the bottleneck and transition layers with θ < 1 are used, we refer to our model as DenseNet-BC. Model compactness. As a direct consequence of the input concatenation, the feature maps learned by any of the DenseNet layers can be accessed by all subsequent layers. This encourages feature reuse throughout the network, and leads to more compact models. Implicit Deep Supervision. One explanation for the improved accuracy of dense convolutional networks may be that individual layers receive additional supervision from the loss function through the shorter connections. One can interpret DenseNets to perform a kind of “deep supervision”. The benefits of deep supervision have previously been shown in deeply-supervised nets (DSN; [20]), which have classifiers attached to every hidden layer, enforcing the intermediate layers to learn discriminative features. DenseNets perform a similar deep supervision in an implicit fashion: a single classifier on top of the network provides direct supervision to all layers through at most two or three transition layers. However, the loss function and gradient of DenseNets are substantially less complicated, as the same loss function is shared between all layers.

2.6.3. COMPARISON BETWEEN RESNET50 & DENSENET121

When compared to ResNet-50, DenseNet-121 architecture CNN performs with more accuracy. It also has less trainable parameters as compared to ResNet architecture as discussed above. This overall file size of the model also reduces by almost one thirds. The only downside is that DenseNet takes more time to give an output from the training model.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

DenseNet architectures for ImageNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet architectures for ImageNet

```
Total params: 23,597,957
Trainable params: 23,544,837
Non-trainable params: 53,120
```

```
Total params: 7,042,629
Trainable params: 6,958,981
Non-trainable params: 83,648
```

ResNet-50 Trainable parameters (first image) v/s DenseNet-50 Trainable parameters (second image). The ResNet architecture employs more trainable parameters (approximately 24 million for FER2013 dataset) which leads to more training time and a larger model file size when compared to DenseNet architecture (approximately 7 million) for the same dataset. This is due to the lack of feed-forward loops in ResNet architecture. Overall the accuracy of ResNet is lower than DenseNet when trained under the same parametric limitations. It can be seen in the images below.

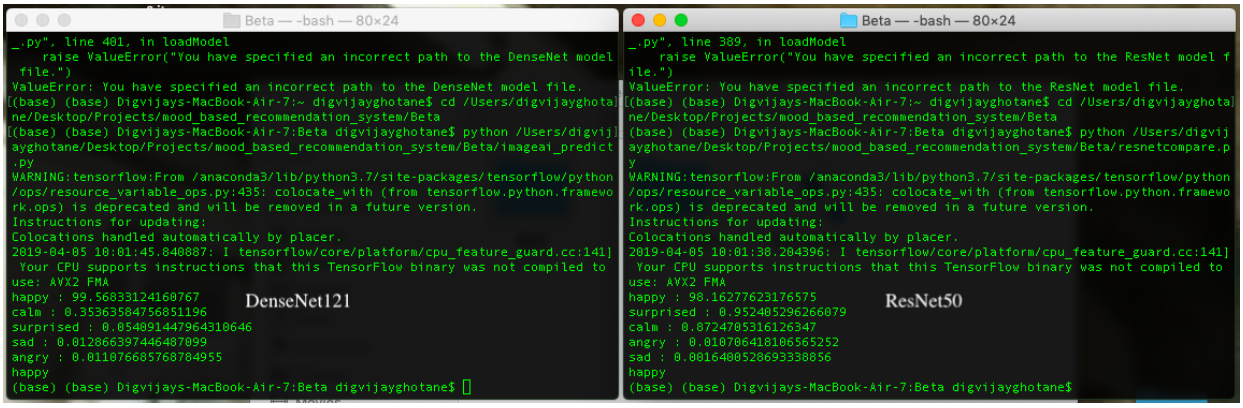
```
Epoch 00017: saving model to /content/drive/My Drive/images/models/model_ex-017_acc-0.749134.h5
757/757 [==============================] - 714s 944ms/step - loss: 0.6818 - acc: 0.7792 - val_loss: 0.7655 - val_acc: 0.7491
Epoch 18/30
190/190 [==============================] - 49s 257ms/step - loss: 0.7697 - acc: 0.7472
```

DenseNet-121 model, with accuracy of 74.91% trained for 30 epochs.

```
Epoch 00020: saving model to /content/drive/My Drive/images/models/model_ex-020_acc-0.732311.h5
757/757 [==============================] - 746s 985ms/step - loss: 0.6242 - acc: 0.7613 - val_loss: 0.7056 - val_acc: 0.7323
Epoch 21/30
190/190 [==============================] - 52s 273ms/step - loss: 0.7057 - acc: 0.7302
```

ResNet-50 model, with accuracy of 73.23% trained for 30 epochs.

The live performance on our project was also tested on the same image and can be seen from the image below.



3. IMPLEMENTATION

3.1. INITIAL SETTING UP

```
import numpy as np
import cv2
from imageai.Prediction.Custom import CustomImagePrediction
import os
import webbrowser
```

The above libraries are imported.

Since images are converted to bit arrays, numpy is used extensively by OpenCV. The ‘cv2’ library is the OpenCV library. ImageAI library is also imported. ‘os’ library is built in and is imported for functions like defining paths. The ‘webbrowser’ library is imported to open web links.

A few functions are defined for the purpose of ease in the code later on as shown below.

```
def int_check(l,h,message): #for inputting integers
    while True:
        s = input(message)
        if s.isdigit():
            s=int(s)
            if l<=s<=h:
                return s
            else:
                print("\nError in input, please try again.")
        else:
            print("\nError in input, please try again.")
```

The function ‘int_check’ checks whether the input received is an integer input lying between two integer values, ideally a maximum and minimum value. The code below defines a function to set the resolution of the OpenCV output to 720p.

```
def make_720p(): #defining resolution to 720p
   cap.set(3, 1280)
   cap.set(4, 720)
```

3.2. FACE DETECTION

3.2.1. SETTING UP OF OPENCV

We use the webcam of the computer to detect and capture the image of the person. We further use this image to detect the mood of the person using facial expression recognition.

Firstly, we have to open the webcam of the computer. For this we need make use of OpenCV which is an open source computer vision and machine learning library software. OpenCV is used to develop real-time computer vision applications like video capture analysis and image processing like face and object detection.

Firstly, we import OpenCV[13] as "cv2" using the code below:

```
import cv2
```

To capture the video from webcam we use the code as stated below. Since we are using the webcam, we initialize it using '0' as the integer parameter.

```
cap = cv2.VideoCapture(0)
```

We can control the resolution and can set it depending on the resolution supported by the camera. In this case we make use of 720 pixels. In order to set the resolution as 720p we use the aspect ratio 16:9 thus it is 1280 x 720 lines.

```
def make_720p():
            cap.set(3, 1280)
            cap.set(4, 720)
```

Here, we are defining a function to set the resolution to 720p.

[13] https://docs.opencv.org/2.4/index.html

As long as each “frame” is received from the previously defined capture “cap” function is returned “ret” will be set to a boolean value of ‘True.’ We make use of this and initiate it inside a ‘while’ loop.

```
ret, frame = cap.read()
```

We convert the image frame captured from color to grayscale for better distinction in the bit values. OpenCV library itself makes use of grayscale images. RGB values have 24-bit values whereas Grayscale values are 8bit. The “frame” is converted using the function “cv2.COLOR_BGR2GRAY” i.e we convert the Blue, Green and Red captured frame to Grayscale.

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

3.2.2. DETECTION OF FACE

For face detection, we use the required XML classifier file ‘frontalface_alt2’ using the following line of code:

```
face_cascade = cv2.CascadeClassifier('directory')
```

We use “v2.CascadeClassifier.detectMultiScale()” to find faces. The parameters are as follows:

1. image: Matrix of the type CV_8U containing an image where objects are detected.

2. scaleFactor: Parameter specifying how much the image size is reduced at each image scale. This scale factor is used to create scale pyramid as shown in the picture. Suppose, the scale factor is 1.5, it means we're using a small step for resizing, i.e. reduce size by 50 percent, we increase the chance of a matching size with the model for detection is found.

3. minNeighbors: Parameter specifying how many neighbors each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces: higher value results in less detections but with higher quality. We're using 5 in the code.

```
faces =
face_cascade.detectMultiScale(gray,scaleFactor=1.5,minNeighbors = 5)
```

In order to break or close the windows, we have given the function “cv2.waitKey()” and a hexadecimal constant “0xFF” which returns 11111111 i.e 8-bit in binary. This is because, waitkey() returns a 32-bit integer value and the input key “z” depends only on the 8 bits and not the rest. Thus, if the user presses ‘z,’ the image is captured and saved at that instant to be fed into the machine learning model.

```
if cv2.waitKey(20) & 0xFF == ord('z'):
                break
```

To end the program and release the captured “*frame*” and to destroy the windows.

```
cap.release()
cv2.destroyAllWindows()
```

The complete code is given below.

```
#Capturing Face

face_cascade =
cv2.CascadeClassifier('/usr/local/Cellar/opencv//4.0.1/share/opencv4
/haarcascades/haarcascade_frontalface_alt2.xml')
cap = cv2.VideoCapture(0)
make_720p()
while(True): #Capture frame-by-frame
   ret,frame = cap.read()
   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   faces = face_cascade.detectMultiScale(gray, scaleFactor=1.5,
minNeighbors = 5)
   for(x,y,w,h) in faces:
       roi_gray=gray[y:y+h,x:x+w]
       roi_color=frame[y:y+h,x:x+w]
       color = (255,0,0)
       stroke = 2
       cv2.rectangle(frame,(x,y),(x+w,y+h), color, stroke)
       cv2.imwrite("image.png", roi_gray)

   cv2.imshow('Live Webcam',frame) #Display the resulting frame
   if cv2.waitKey(1) & 0xFF == ord('z'):
       break
cap.release()
cv2.destroyAllWindows()
```

3.3. EMOTION DETECTION

The next step is to detect emotions. For this we used a trained Machine Learning model that is based on DenseNet-121 architecture. The '121' in the DenseNet stands for 121 convolution layers present in the said architecture. We've compared the two widely used architectures in the Literature Survey. The first step was to train a model. After the model is trained, we used it to insert the captured image into the trained model to receive an output of the closest emotion that the person has in accordance with their facial expression.

3.3.1 TRAINING OF MODEL

```
from imageai.Prediction.Custom import ModelTraining
import os

model_trainer = ModelTraining()
model_trainer.setModelTypeAsDenseNet()
model_trainer.setDataDirectory("directory to dataset organized as
train and test")
model_trainer.trainModel(num_objects=5, num_experiments=10,
enhance_data=True, batch_size=32, show_network_summary=True)
```

The above code is used to train the model. This model was trained on Google Colab[14] for 50 epochs to attain 76.18% accuracy at the 38th epoch.

```
Epoch 00038: saving model to /content/drive/My Drive/images/models/model_ex-038_acc-0.761834.h5
757/757 [==============================] - 720s 952ms/step - loss: 0.4624 - acc: 0.8635 - val_loss: 0.7987 - val_acc: 0.7618
Epoch 39/50
190/190 [==============================] - 49s 258ms/step - loss: 0.7983 - acc: 0.7610
```

The training process generates a JSON file that maps the objects types in the image dataset and creates lots of models. There is then a peak in the model with the highest accuracy and custom image prediction can be performed using the model and the JSON file generated.

Firstly, we organise our dataset images in 'Train' and 'Test' folders with each emotion(label) in a sub folder and the corresponding images in that folder. The directory folder looks something like this:

user/images/train/happy *user/images/train/sad*

14 https://research.google.com/colaboratory/faq.html

user/images/train/angry	*user/images/test/sad*
user/images/train/calm	*user/images/test/angry*
user/images/train/surprised	*user/images/test/calm*
user/images/test/happy	*user/images/test/surprised*

Once the dataset is ready, the first line of code creates an instance of the 'Model Training' class. The second line of code sets the model type to 'DenseNet' architecture, since as discussed above, this library allows the use of 4 different deep learning algorithms (SqueezeNet , ResNet , InceptionV3 and DenseNet). The function '.setDataDirectory()' accepts a string which must be the path to the folder that contains the test and train subfolder of your image dataset.

The function '.trainModel()' starts the training process. Once it starts, it will create a JSON file in the dataset/json folder (e.g user/images/json) which contains the mapping of the classes of the dataset. The parameters associated with the '.trainModel()' class are as follows:

1. parameter num_objects (required) : This refers to the number of different classes in your image dataset.

2. parameter num_experiments (required) : This is the number of times the algorithm will be trained on your image dataset. The accuracy of your training does increases as the number of times it trains increases. However, it does peak after a certain number of trainings;and that point depends on the size and nature of the dataset.

3. parameter enhance_data (optional) : This parameter is used to transform your image dataset in order to generate more sample for training. It is set to False by default. However, it is useful to set it to True if your image dataset contains less than 1000 images per class.

4. parameter batch_size (optional) : During training, the algorithm is trained on a set of images in parallel. Because of this, the default value is set to 32. You can increase or reduce this value if you understand well enough to know the capacity of the system you are using to train. Should you intend to change this value, you should set it to values that are in multiples of 8 to optimize the training process.

5. parameter show_network_summary (optional) : This parameter when set to True displays the structure of the algorithm you are training on your image dataset in the CLI before training starts. It is set to False by default.

6. parameter initial_learning_rate (optional) : This parameter is a highly technical value. It determines and control the behaviour of your training which is critical to the accuracy that can be achieved. Change this parameter's value only if you understand its function fully.

7. parameter initial_learning_rate (optional) : This parameter is a highly technical value. It determines and control the behaviour of your training which is critical to the accuracy that can be achieved. Change this parameter's value only if you understand its function fully.

8. training_image_size initial_learning_rate (optional) : This is the size at which the images in your image dataset will be trained, irrespective of their original sizes. The default value is 224 and must not be set to less than 100. Increasing this value increases accuracy but increases training time, and vice-versa.

3.3.2. USING TRAINED MODEL FOR PREDICTION

The following code is used to predict the mood of the person using the trained model. The trained model has a '.h5' file extension with a json file.

```
from imageai.Prediction.Custom import CustomImagePrediction
import os

execution_path = os.getcwd()

prediction = CustomImagePrediction()
prediction.setModelTypeAsDenseNet()
prediction.setModelPath(os.path.join(execution_path, "model_ex-
038_acc-0.761834.h5"))
prediction.setJsonPath(os.path.join(execution_path,
"model_class.json"))
prediction.loadModel(num_objects=5)

predictions, probabilities =
prediction.predictImage(os.path.join(execution_path, "image.png"),
result_count=5)

for eachPrediction, eachProbability in zip(predictions,
probabilities):
    print(eachPrediction + " : " + eachProbability)
location_of_emo = probabilities.index(max(probabilities))
print(predictions[location_of_emo])
```

Parameters:

1. .setModelTypeAsDenseNet(): This function sets the model type of the image recognition instance you created to the DenseNet model, which means you will be performing your image prediction tasks using the “DenseNet” model generated during your custom training.

2. .setModelPath(): This function accepts a string which must be the path to the model file generated during your custom training and must corresponds to the model type you set for your image prediction instance.

3. .setJsonPath(): This function accepts a string which must be the path to the JSON file generated during your custom training.

4. .loadModel(): This function loads the model from the path you specified in the function call above into your image prediction instance. You will have to set the parameter num_objects to the number of classes in your image dataset which is 5 in this case.

5. parameter prediction_speed (optional): This parameter allows you to reduce the time it takes to predict in an image by up to 80% which leads to slight reduction in accuracy. This parameter accepts string values. The available values are “normal”, “fast”, “faster” and “fastest”. The default values is “normal”

6. .predictImage(): This is the function that performs actual prediction of an image. It can be called many times on many images once the model as been loaded into your prediction instance.

7. parameter image_input (required): This refers to the path to your image file, Numpy array of your image or image file stream of your image, depending on the input type you specified.

8. parameter result_count (optional): This refers to the number of possible predictions that should be returned. The parameter is set to 5 by default.

9. parameter input_type (optional): This refers to the type of input you are parse into the image_input parameter. It is “file” by default and it accepts “array” and “stream” as well.

10. returns prediction_results (a python list): The first value returned by the predictImagefunction is a list that contains all the possible prediction results. The results are arranged in descending order of the percentage probability.

11. returns prediction_probabilities (a python list): The second value returned by the predictImage function is a list that contains the corresponding percentage probability of all the possible predictions in the prediction_results.

3.4. MEDIA RECOMMENDATION

The media recommendation currently is done by using web applications such as Buzzfeed[15] to recommend images, Spotify[16] to recommend music and Netflix[17] to recommend movies to the user. The user is presented with an option to select what they want to be recommended and the recommendation is done using these services.

15 https://www.buzzfeed.com/
16 https://www.spotify.com/
17 https://www.netflix.com/

4. COMPONENTS

4.1. HARDWARE

The only hardware components this project requires is a personal computer or laptop with a working webcam.

4.1.1. Webcam

A webcam is a video camera that feeds or streams its image in real time to or through a computer to a computer network. When "captured" by the computer, the video stream may be saved, viewed or sent on to other networks travelling through systems such as the internet, and e-mailed as an attachment. When sent to a remote location, the video stream may be saved, viewed or on sent there. Unlike an IP camera (which connects using Ethernet or Wi-Fi), a webcam is generally connected by a USB cable, or similar cable, or built into computer hardware, such as laptops.

4.2. SOFTWARE

The language used for this project has been Python 3.7.

4.2.1. Python

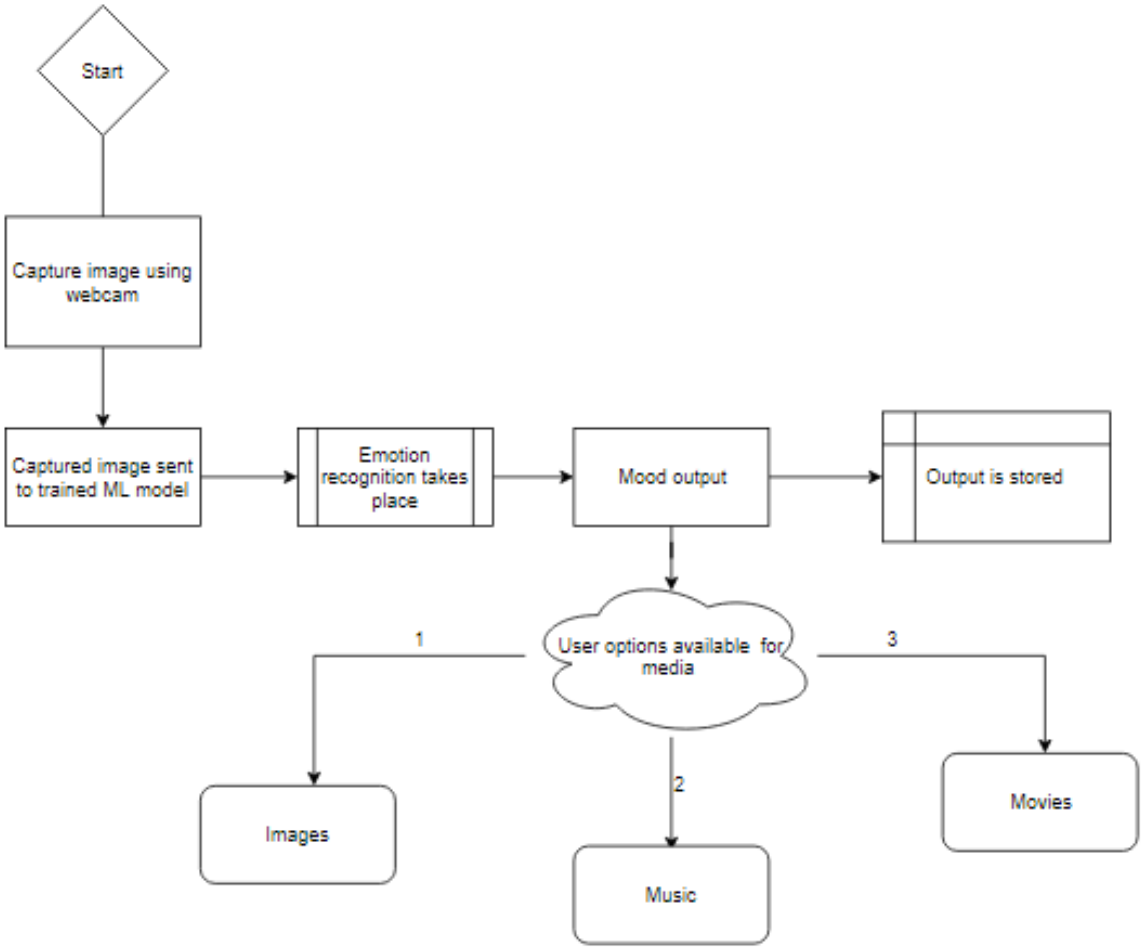
Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

5. PROPOSED SYSTEM

5.1. APPLICATION FLOW

First, the input is taken through the webcam. The picture of the user is taken when the user presses the ‘z’ key on the keyboard which has been programmed to end the loop for capturing the face of the user. When the key is pressed, the image is saved as a PNG image file, ‘image.png.’ This image is a grayscale image that is fed to the trained machine learning model. This model has been trained on the FER2013 dataset, with a train-test split of about 75 percent train and 25 percent test images. The moods that we’re recognizing are happy, sad, angry, surprised and calm. The trained model gives us an output, that is probabilities of the mood of the person. The highest probability is then chosen and the corresponding mood is taken into consideration. This mood is then used to recommend media, as chosen by the user, that is either Images, Songs or Movies.

5.2. BLOCK DIAGRAM



6. CONCLUSION

Even though facial recognition has been around for at least a decade, it's application in this field has been either very limited or even close to zero. There are very limited number of services that make use of facial recognition technology with integrated machine learning algorithms that detect the mood of a person based on their facial expressions and render to the same user media based on it. Face recognition is at a pioneering stage to solve many problems pertaining to our everyday lives. We are the crux of revolutionizing the way we can use image processing. It can be used to identify and recognise a facial expression.

The mood based recommendation system is used to provide a personalized approach to how one media is delivered to individuals based on their mood. Thus, ensuring that the person can experience highly relevant and curated content when they so desire. The project allows the user to sit back and relax while the machine learning algorithm takes care of the media recommendation for the user according to their mood. This technology can be applied to not only alleviate ones bad mood but also play media that could help boost ones mood.

The architecture used not only creates a model that is accurate (76.18% accuracy), but also one that has less trainable parameters and has a low file size. We wanted to learn how to employ machine learning into this kind of a recommendation system using open vision computing. This project was an attempt at employing Artificial Intelligence in the day to day leisure activities of a human and we believe that the prototype for the same is ready. In the distant future, this technology can also be used for certain applications to treat people with depression, anxiety or experiencing sadness. Since this technology aims at presenting media to the user, it is assumed that the user may make use of it on a daily basis, and the moods that are recorded of the user, may be used for further applications.

7. FUTURE SCOPE

The main aim of the project was to develop a program that could predict your mood and thus provide recommendations in the form of media content. The application can further be employed in an array of scenarios. The current prototype opens the media content on a web browser, this can further be deployed using a mobile application or a web application by integrating it with Spotify, Apple Music and Netflix API's. Making the content available without the need of the internet would be a major breakthrough. This would serve a multitude of purposes and allow people to use this when they are offline as well. Report the mood of a person after constant usage and detect for many types of illnesses that are associated with erratic mood patterns or constant mood patterns.

All in all, this particular prototype can be taken forward for not only a full-fledged media recommendation web application that runs on the web browser or a mobile phone application but has a wide range of other applications. By integrating it with other API's this can be implemented for usage by the people online from other parts of the world, every single day. Not only that, but this facial expression recognition system can also be employed for other things such as by car manufacturers to check whether a person is falling asleep and if that happens, the car should not start. Other than that, collaborative filtering can be employed for recommending similar type of media to people with similar taste in media.

8. REFERENCES

1. I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, and Y. Bengio, "Challenges in representation learning: A report on three machine learning contests,"Neural Networks, vol. 64, pp. 59–63, 2015. (arXiv:1307.0414)
2. Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, "Densely Connected Convolutional Network," last revised 28 Jan 2018(v5). (arXiv:1608.06993)
3. Knutson, B. J Nonverbal Behav (1996) 20: 165. https://doi.org/10.1007/BF02281954
4. G. van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.
5. Gary Bradski in Dr. Dobbs Journal, 2000, The OpenCV Library (https://github.com/opencv/opencv)
6. Paul Viola and Michael Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, 2001
7. John Olafenwa and Moses Olafenwa, Introduction to Deep Computer Vision, 2018
8. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
9. Chollet, F. (2015) keras, GitHub. https://github.com/fchollet/keras
10. M. D. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Neural Networks, In ECCV, 2014

11. Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, “Deep Residual Learning for Image Recognition,” 2015 (arXiv:1512.03385)

12. Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Laurens van der Maaten and Kilian Q. Weinberger, “Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,” 2017 (arXiv:1608.06993)

13. https://docs.opencv.org/2.4/index.html

14. https://research.google.com/colaboratory/faq.html

15. https://www.buzzfeed.com/

16. https://www.spotify.com/

17. https://www.netflix.com/