Major Project Report on

Emotion Recognition and Personality Assessment using Multimodal Data

submitted in partial fulfillment of the

requirements for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

By:

Orendra Singh (2/CSE-3/2018[LE]/176807219)

Jasmeet Singh(?/CSE-3/2018/02376802718)

Medha (48/CSE-3/2018/40676802718)

Harsh Shawait Singh (59/CSE-3/2018/376807218)

under the guidance of

Dr. Aashish Bhardwaj

(HOD,CSE)

Department of Computer Science & Engineering

Guru Tegh Bahadur Institute of

Technology(Affiliated to Guru Gobind Singh

Indraprastha University Dwarka, New Delhi)Class of

Batch- 2018-2022

CERTIFICATE

This is to certify that the Major Project Report **on Emotion Recognition and Personality Assessment using Multimodal Data** , is submitted by **Orendra Singh (2/CSE-3/2018(LE)), Jasmeet Singh(?/CSE-3/2018/02376802718), Medha(48/CSE-3/2018) and Harsh Shawait Singh (59/CSE-3/2018)** who worked on and developed the project work under my supervision. I approve the Major Project for submission.

Dr. Aashish Bhardwaj

(HOD, CSE)

Department of Computer Science & Engineering

Guru Tegh Bahadur Institute of Technology

(Affiliated to Guru Gobind Singh Indraprastha University Dwarka, New Delhi)

Date: 02/06/2022

ACKNOWLEDGEMENT

I wish to express my deep sense of gratitude to everyone who supported me as it would not have been possible without the kind support and help of the involved individuals and organizations. I would like to extend my sincere thanks for the immense motivation and belief inculcated by each and every individual, throughput the process. I am highly indebted to ***Dr. Aashish Bhardwaj*** for his guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in improving the project as a whole. I would like to express my gratitude towards my parents & teachers and staff at ***Guru Tegh Bhadur Institute of Technology***, for their kind co-operation and encouragement which helped me in the process of developing the project. My thanks and appreciations also go to my project-mates and colleagues in developing the project and people who have willingly helped me out with their abilities.

Orendra Singh (2/CSE-3/176807219)
Jasmeet Singh(?/CSE-3/2018/02376802718)
Medha (48/CSE-3/40676802718)
Harsh Shawait Singh (59/CSE-3/376807218) Date: 02/6/2022

ABSTRACT

The techniques devised under the domain of Artificial Intelligence, have diverse utilization and hence a huge impact in the process to simplify day-to-day activities, in general. With vast availability of data in distinct forms, a recent problem, which is the basis of landmark research and multi-fold development amongst technical researchers and developers, aims to identify and classify human emotion and assess human behavior and personality, accurately, with an objective to support and enhance general human observation or even go beyond it to analyze certain characteristic traits for given multimodal data input. The project presents an effective analysis for three distinct media inputs, aiming at recognition and classification of facial expressions, following analysis of human emotion, perceived through voice and retrieval of psychological traits for a real time visual, audial and textual input respectively, deployed through an interactive web application. Accuracy has been used as a metric to evaluate the performance of the mathematical models designed. The basis and design of each one, is rooted at Convolutional Neural Networks, widely used as a Deep Learning methodology to provide bench-mark results for Image Classification and Segmentation. The model designs present an effective approach using Recurrent Neural Networks with Convolutional Neural Networks to provide accurate output. Further, Transfer Learning has been used as an additional technique to enhance the accuracy of result and present an improved model to preserve resources. The output has been analyzed and visualized through different functions and techniques.

LIST OF TABLES

LIST OF FIGURES

CONTENTS


GURU TEGH BAHADUR INSTITUTE OF TECHNOLOGY
DELHI

1. Introduction

1.1 Introduction to Project

The main purpose of this project is to design a web based application which allows the user to receive an instant respone and provide support for identification of emotion and personality from given visual,audial and textual inputs at real time. The backend of the project has been developed using Python and its Libraries and Flask whereas the frontend has been developed using basics of HTML,CSS and JavaScript. Key methodology to provide the functionalities has been developed using Deep Learning Technqiues. Convolutional Neural Networks and Recurrent Neural Networks have been used for classification of emotion and evaluation of personality traits for a given real time data input. Both the model designing techniques have been implemented using Tensorflow and Keras. Accuracy has been used a metric to check on the results using Train Set and Test Set Split. NLTK has been used to process text and extract physcological traits from the given textual data. The metric for evaluation is as specified below :-

Accuracy = (Number of correclty predicted images/Total number of tested images)×100%

The project also involves the deployment of the trained models to predict and serve appropriate results. The key features of the work in the project includes a deep analysis of the proposed model for facial emotiom recognition design on the basis of five distinct metrics and an effective comparison of the performance of a pre-trained model as ResNet 50 on the same Data-Set. Major tech giant companies like Google, Microsoft, Apple are trying to make their virtual assistants like Siri, Google Assistant, Alexa appear more like humans. These companies are doing great research and development to humanize their AI functionality of their virtual assistants. The idea is to incorporate digital assistants with a psychological machine learning model that is capable of detecting human facial emotions and acts accordingly.The same idea inspires this project and hence the project aims to develop the capability specified and check for appropriate metrics to present the best solution , amongst all possible solutions , all of which do not always work well and highly depend on the resources available rather than amplifying the appropriate results with whatever data is available, by a detailed analysis and absolute feature identification and extraction for further application.

1.2 Limitation of Existing Solutions

The core of the project involves classification and there are a number of techniques which serve the purpose. Here we disucuss different techniques which do not utilize the concept of neural networks and deep learning but still work best for general classification tasks. Classification is a typical supervised

learning task. We use it in those cases where we have to predict a categorical type, that is if a particular example belongs to a category or not (unlike regression, which is used to predict continuous values). For example, sentiment analysis, classify an email as spam or not, predicting if a person buys an SUV or not provided a training set containing salary, and buying an SUV. The types of Classification Models are as follows:-

1.2.1 **Logistic Regression** is a linear classification model (and hence, the prediction boundary is linear), which is used to model binary dependent variables. It is used to predict the probability (p) that an event occurs. If p >= 0.5, the output is 1 else 0. The sigmoid function maps the probability value to the discrete classes (0 and 1). For example, say our logistic regression model is trained on the dataset containing a person's salary and whether he buys an SUV or not. Now, given the person's salary, our model predicts whether or not the person buys an SUV. Few of the assumptions of logistic regression are – there is no high inter-correlation among the predictors, there is a linear relationship between the sigmoid of the outcome and the predictor variables.

1.1.2 **K – Nearest Neighbours** is a non – linear classifier (and hence, the prediction boundary is non-linear) that predicts which class a new test data point belongs to by identifying its k nearest neighbors' class. We select these k nearest neighbors based on Euclidean distance. Among these k neighbours, the number of data points in each category is counted, and the new data point is assigned to that category where we got the most neighbours in.

1.1.3 **Support Vector Machine (SVM)** is used as a linear or non-linear classifier based on the kernel used. If we use a linear kernel, then the classifier and hence the prediction boundary are linear. Here, to separate two classes, we need to draw a line. The line is such that there is a maximum margin. This line is drawn equidistant from both the sets. We draw two more lines on either side, and these are known as support vectors. SVMs learn from the support vectors, unlike other machine learning models that learn from the correct and incorrect data. For example, suppose we have two classes – apples and oranges. In that case, SVM learns those examples which are rightmost in apples (an apple resembling an orange) and leftmost in oranges (an orange resembling an apple); that is, they look at the extreme cases. Therefore, they perform better most of the time.

1.1.4 **Kernel SVM** is particularly useful when the data is not linearly separable. Therefore, we take our non – linearly separable dataset, map it to a higher dimension, get a linearly

separable dataset, invoke SVM classifier, build a decision boundary for the data, and then project it back into original dimensions. This mapping can be computationally expensive, and hence we use the Kernel Trick, which gives similar results. The kernels available are – The Gaussian RBF Kernel, Sigmoid Kernel, Polynomial Kernel (default value of kernel is 'RBF'). This is also known as the non – linear SVM.

1.1.5 **Naive Bayes Classifier** works on the basis of Bayes' Theorem. The fundamental assumptions made are that all the features are independent of one another and contribute equally to the outcome; all are of equal importance. But these assumptions are not always valid in real life (disadvantage of Naive Bayes). It is a probabilistic classifier model whose crux is the Bayes' theorem.

1.1.6 **Decision Tree Classification** is the most powerful classifier. A Decision tree is a flowchart like a tree structure, where each internal node denotes a test on an attribute (a condition), each branch represents an outcome of the test (True or False), and each leaf node (terminal node) holds a class label. Based on this tree, splits are made to differentiate classes in the original dataset given. The classifier predicts which of the classes a new data point belongs to based on the decision tree. The prediction boundaries are horizontal and vertical lines.

1.1.7 **Random Forest Classification** is an example of Ensemble learning, where multiple machine learning algorithms are put together to create one bigger and better performance ML algorithm. We randomly pick 'k' data points from the training set, build a decision tree associated with these k points. Then, we choose the number of trees 'n' we want to build and repeat. For a new data point, we take the predictions of each of the 'n' decision trees and and assign it to the majority vote category.

Advantages of Convolutional Neural Networks over traditional classification techniques are as follows: -

Image classification can be accomplished by any machine learning algorithms(logistic regression, random forest and SVM). But all the machine learning algorithms required proper features for doing the classification. If you feed the raw image into the classifier, it will fail to classify the images properly and the accuracy of the classifier would be less. CNN (convolution neural network) extract the features from the images and it handles the entire feature engineering part. In normal CNN architecture, beginning layers are extracting the low-level features and end level layers extract high-level features from the image.

Before CNN, we need to spend time on selecting the proper features for classifying the image. There are so many handcrafted features available(local feature, global feature), but it will take so much time to select the proper features for a solution(image classification) and selecting the proper classification model. CNN handles all these problems and the accuracy of the CNN is higher compared with the normal classifier.

1.3 Features of Project

Over the years, research on convolutional neural networks (CNNs) has progressed rapidly, however the real-world deployment of these models is often limited by computing resources and memory constraints. What has also led to extensive research in ConvNets is the accuracy of difficult classification tasks that require understanding abstract concepts in images.

Another reason why CNN are hugely popular is because of their architecture — the best thing is there is no need for feature extraction. The system learns to do feature extraction and the core concept of CNN is, it uses convolution of image and filters to generate invariant features which are passed on to the next layer. The features in next layer are convoluted with different filters to generate more invariant and abstract features and the process continues till one gets final feature / output (let say face of X) which is invariant to occlusions.

Also, another key feature is that deep convolutional networks are flexible and work well on image data. As one researcher points out, convolutional layers exploit the fact that an interesting pattern can occur in any region of the image, and regions are contiguous blocks of pixels. But one of the reasons why researchers are excited about deep learning is the potential for the model to learn useful features from raw data. Now, convolutional neural networks can extract informative features from images, eliminating the need of traditional manual image processing methods.

In fact, machine learning engineer Arden Dertat in an article in Towards Data Science states that CNN is the most popular deep learning model. According to Dertat, the recent surge of interest in deep learning is thanks to the effectiveness and popularity of convnets. Such is the accuracy that CNNs have become the go-to models for a lot of industry applications. For example, they are used for recommender systems, natural language processing and more. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. For example, given many pictures of cats and dogs, it can learn the key features for each class by itself.

Another area where we see the application of ConvNets is in the prevention of fraud, which is a big concern for telecom companies. In a bid to develop algorithms that detect early potential frauds and/or prevent them, deep learning techniques, especially ConvNets are being used to detect fraudsters in mobile communications. In a research paper, published in Science Direct, fraud datasets culled from customer details records (CDR) are used and learning features are extracted and classified to fraudulent and non-fraudulent events activity. The paper revealed how deep convolution neural networks surpassed other traditional machine learning algorithms such as random forest, support vector machines and gradient boosting classifier, especially in terms of accuracy.

According to AI evangelist, Alexander Del Toro Barba, convolutional neural networks revolutionized the industry, due to the ability to handle large, unstructured data.

Hence, ConvNets are extremely successful in areas where large, unstructured data is involved, such as image classification, speech recognition, natural language processing.

ConvNets are more powerful than machine learning algorithms and are also computationally efficient.

The trend was kickstarted in 2012 with AlexNet which was only 8 layers and how now progressed to the 152 layer ResNet.

In terms of architecture, the key building block of CNN is the convolutional layer. According to a MathWork post, a CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images. Since CNNs eliminate the need for manual feature extraction, one doesn't need to select features required to classify the images. How CNN work is by extracting features directly from images and the key features are not pretrained; they are learned while the network trains on a collection of images, the post notes. It is the automated feature extraction that makes CNNs highly suited for and accurate for computer vision tasks such as object/image classification.

1.4 Objective of Project

Convolutional Neural Networks (CNN) are everywhere. It is arguably the most popular deep learning architecture. The recent surge of interest in deep learning is due to the immense popularity and effectiveness of convnets. The interest in CNN started with AlexNet in 2012 and it

has grown exponentially ever since. In just three years, researchers progressed from 8 layer AlexNet to 152 layer ResNet.

CNN is now the go-to model on every image related problem. In terms of accuracy they blow competition out of the water. It is also successfully applied to recommender systems, natural language processing and more. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. For example, given many pictures of cats and dogs it learns distinctive features for each class by itself.

CNN is also computationally efficient. It uses special convolution and pooling operations and performs parameter sharing. This enables CNN models to run on any device, making them universally attractive.

All in all this sounds like pure magic. We are dealing with a very powerful and efficient model which performs automatic feature extraction to achieve superhuman accuracy (yes CNN models now do image classification better than humans). The objective is to utilise these features of a CNN and classify an image on basis of the facial expressions to mark the existenece of a specifc emotion.Moreover, the project also aims to comapre the idea of the two distict techniques as a model trained from scratch and the other, pre-trained and observe and visualize the output on metrics.

2. Tools and Technologies

2.1 Python

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.

python-programming-language Python-Tutorial

This specially designed Python tutorial will help you learn Python Programming Language in most efficient way, with the topics from basics to advanced (like Web-scraping, Django, Deep-Learning, etc.) with examples.Below are some facts about Python Programming Language:

i. Python is currently the most widely used multi-purpose, high-level programming language.
ii. Python allows programming in Object-Oriented and Procedural paradigms.

iii. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

iv. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard library which can be used for the following:

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia
- Scientific computing
- Text processing

2.2 Libraries

Normally, a library is a collection of books or is a room or place where many books are stored to be used later. Similarly, in the programming world, a library is a collection of precompiled codes that can be used later on in a program for some specific well-defined operations. Other than pre-compiled codes, a library may contain documentation, configuration data, message templates, classes, and values, etc.

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and convenient for the programmer. As we don't need to write the same code again and again for different programs. Python libraries play a very vital role in fields of Machine Learning, Data Science, Data Visualization, etc.

Working of Python Library

As is stated above, a Python library is simply a collection of codes or modules of codes that we can use in a program for specific operations. We use libraries so that we don't need to write the

code again in our program that is already available. But how it works. Actually, in the MS Windows environment, the library files have a DLL extension (Dynamic Load Libraries). When we link a library with our program and run that program, the linker automatically searches for that library. It extracts the functionalities of that library and interprets the program accordingly. That's how we use the methods of a library in our program. We will see further, how we bring in the libraries in our Python programs.

The Python Standard Library contains the exact syntax, semantics, and tokens of Python. It contains built-in modules that provide access to basic system functionality like I/O and some other core modules. Most of the Python Libraries are written in the C programming language. The Python standard library consists of more than 200 core modules. All these work together to make Python a high-level programming language. Python Standard Library plays a very important role. Without it, the programmers can't have access to the functionalities of Python. But other than this, there are several other libraries in Python that make a programmer's life easier. Let's have a look at some of the commonly used libraries:

1. **TensorFlow:** This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics.
2. **Matplotlib:** This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.
3. **Pandas:** Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.
4. **Numpy:** The name "Numpy" stands for "Numerical Python". It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library.

5. **SciPy:** The name "SciPy" stands for "Scientific Python". It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.
6. **Scrapy:** It is an open-source library that is used for extracting data from websites. It provides very fast web crawling and high-level screen scraping. It can also be used for data mining and automated testing of data.
7. **Scikit-learn**: It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy.
8. **PyGame:** This library provides an easy interface to the Standard Directmedia Library (SDL) platform-independent graphics, audio, and input libraries. It is used for developing video games using computer graphics and audio libraries along with Python programming language.
9. **PyTorch:** PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.
10. **PyBrain**: The name "PyBrain" stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks. It is so flexible and easily understandable and that's why is really helpful for developers that are new in research fields.

There are many more libraries in Python. We can use a suitable library for our purposes. Hence, Python libraries play a very crucial role and are very helpful to the developers.

As we write large-size programs in Python, we want to maintain the code's modularity. For the easy maintenance of the code, we split the code into different parts and we can use that code later ever we need it. In Python, modules play that part. Instead of using the same code in different programs and making the code complex, we define mostly used functions in modules and we can just simply import them in a program wherever there is a requirement. We don't need to write that code but still, we can use its functionality by importing its module. Multiple interrelated modules are stored in a library. And whenever

we need to use a module, we import it from its library. In Python, it's a very simple job to do due to its easy syntax. We just need to use import.

2.3 Matplotlib

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

2.4 Tensorflow

TensorFlow the massively popular open-source platform to develop and integrate large scale AI and Deep Learning Models has recently been updated to its newer form TensorFlow 2.0. This brings a massive boost in features in the originally feature-rich ML ecosystem created by the TensorFlow community.

What is Open-Source and how is it made TensorFlow so successful?

Open-Source means something that the people(mainly developers) can modify, share, integrate because all the original design features are open to all. This makes it very easy for a particular software, product to expand easily, effectively, and in a very little time. This feature allowed the original creator of TensorFlow i.e Google to easily port this into every platform available in the market that includes Web, Mobile, Internet of Things, Embedded Systems, Edge Computing and included support of various other languages such JavaScript, Node.js, F#, C++, C#, React.js, Go, Julia, Rust, Android, Swift, Kotlin and many other.Along with this came the support for hardware acceleration for running large scale Machine Learning codes. These include CUDA(library for running ML code on GPUs), TPUs(Tensor Processing Unit- Custom hardware provided by Google specially designed and developed to process tensors using TensorFlow) for multiple machine configuration, GPU, GPGPU, Cloud-based TPU's, ASIC (Application Specific Integrated Circuits) FPGAs(Field-Programmable Gate Arrays- These are exclusively used for

custom Programmable Hardware). This also includes new additions such as NVIDIA's Jetson TX2 and Intel's Movidius chips.

Now coming back to the newer and much feature rich TensorFlow2.0:

The things which are added include:

The main API is now non-other than the Keras: The fluid layer of Keras is now integrated on top of the raw TensorFlow code make it simple and easy to use. This would help bring a lot of progress and productivity in the field of Machine Learning and AI.

- Eager Command-line : This simple command line helps us to execute operation immediately without using Session.run command.
- Simplified and Integrated Workflow:
- Using tf.data for data loading(Or NumPy).
- Use Keras for model construction.(We can also use any premade Estimators).
- Use tf.function for DAG graph execution or use eager execution.
- Utilize distribution strategy for high-performance-computing and deep learning models. (For TPUs, GPUs etc).
- TF 2.0 standardizes Saved Model as a serialized version of a TensorFlow graph for a variety of different platforms ranging from Mobile, JavaScript, TensorBoard, TensorHub..etc.
- Support for TensorFlow Lite and TensorFlow Edge Computing: This would help the developers to give effective Machine Learning and AI services to the end devices. This would require very less computing power along with faster model implementation and end-users
- The new extensions for Web Applications and Node.js using TensorFlow.js for new and interactive AI-based websites and applications.
- TensorFlow optimization for Android.
- TensorFlow Integration for Swift and IOS based applications.
- Support for the most-awaited upcoming WebGPU Chrome RFC proposal.
- Unified Programming Paradigms(Directed Acyclic Graph/Functional and Stack/Sequential).
- TensorFlow AIY(Artificial Intelligence for Yourself) support.
- Integration of tf.contrib into separate repositories.
- Improved TPU and TPU support and distributed computation support and support for the same up to v3.

- Improved HPC integration for Parallel Computing.
- Community Integration for Development, Support and Research.
- Integration of tf.contrib best package implementation into the core package.
- Domain-Specific Community Support.
- Extra Support for Model Validation and Reuse.
- End-to-End ML Pipelines and Products available at TensorFlow Hub.
- At last we can now build big ML and Deep Learning models easily, effectively on TensorFlow2.0 for end-users, and implement them on a large scale.

2.4 Keras

Both Tensorflow and Keras are famous machine learning modules used in the field of data science. Here, we will look at the advantages, disadvantages and the difference between these libraries.

TensorFlow

TensorFlow is an open-source platform for machine learning and a symbolic math library that is used for machine learning applications.

Advantages of TensorFlow:

1. Tensor flow has a better graph representation for a given data rather than any other top platform out there.
2. Tensor flow has the advantage that it does support and uses many backend software like GUI and ASIC.
3. When it comes to community support tensor flow has the best.
4. Tensor flow also helps in debugging the sub-part of the graphs.
5. Tensor flow has shown a better performance when compared with other platforms.
6. Easy to extend as it gives freedom to add custom blocks to build on new ideas.

Disadvantages of TensorFlow:

1. Tensor flow not specifically designed for the Windows operating systems but it is designed for other OS like Linux but tensor flow can be installed in windows with the help of a python package installer(pip).
2. The speed of the tensor flow is less when it is compared to other platforms of the same type.
3. For a better understanding of tensor flow, the user must have the fundamentals of calculus.
4. Tensor flow does not support OpenCL.

Keras

It is an Open Source Neural Network library that runs on top of Theano or Tensorflow. It is designed to be fast and easy for the user to use. It is a useful library to construct any deep learning algorithm of whatever choice we want.

Advantages of Keras:

1. Keras is the best platform out there to work on neural network models.
2. The API that Keras has a user-friendly where a beginner can easily understand.
3. Keras has the advantage that it can choose any libraries which support it for its backend support.
4. Keras provides various pre-trained models which help the user in further improving the models the user is designing.
5. When it comes to community support Keras has the best like stack overflow.

Disadvantages of Keras:

The major drawback of Keras is it is a low-level application programming interface.

1. Few of the pre-trained models that the Keras has been not much supportive when it comes to designing of some models.
2. The errors given by the Keras library were not much helpful for the user.

2.5 Flask

- What is Web Framework?
  - Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

- What is Flask?
  - Flask is a web application framework written in Python. It is developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

- WSGI

- Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

- Werkzeug
  - It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

- Jinja2
  - Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.

  - Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the extensions to add such functionality to the application. Some of the popular Flask extensions are discussed later in the tutorial.

2.6 Transfer Learning

Image classification is a task where a computer will predict an image belongs to which class. Before deep learning starts booming, tasks like image classification cannot achieve human-level performance. It's because the machine learning model cannot learn the neighbor information of an image. The model only gets pixel-level information.Thanks to the power of deep learning, image classification task can reach a human level performance using a model called Convolutional Neural Network (CNN).

CNN is a type of deep learning model that learns representation from an image. This model can learn from low to high-level features without human involvement.

The model learns not only information on a pixel level. The model also learns the neighbor information from an image by a mechanism called convolution. Convolution will aggregate neighborhood information by multiplying the collection of pixels in a region and sum them into a value. Those features will be used to classify the image into a class.

Although deep learning can achieve human-level performance, it needs a large amount of data. What if we don't have them? We can use a concept called transfer learning.Transfer learning

is a method where we will use a model that has been trained on large scale data for our problem. Therefore, we only train them by fine-tuning the model. The benefit that we will get is the model will train in a short time. An example for Transfer Learning is as below:-

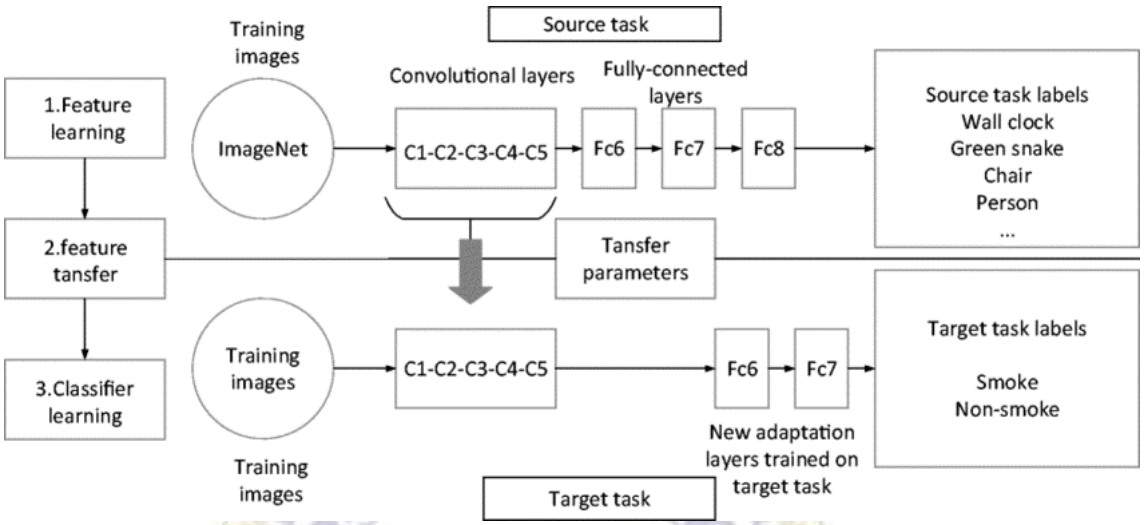


Fig 2.6.1

2.7 Deep Learning Techniques

2.7.1 ResNet50 (Pre-Trained Model for Image Classification)

ResNet, the winner of ILSVRC-2015 competition are deep networks of over 100 layers. Residual networks are similar to VGG nets however with a sequential approach they also use “Skip connections” and “batch normalization” that helps to train deep layers without hampering the performance. After VGG Nets, as CNNs were going deep, it was becoming hard to train them because of vanishing gradients problem that makes the derivate infinitely small. Therefore, the overall performance saturates or even degrades. The idea of skips connection came from highway network where gated shortcut connections were used.

Architecture of ResNet-50

Now we’ll talk about the architecture of ResNet50. The architecture of ResNet50 has 4 stages as shown in the diagram below. The network can take the input image having height, width as multiples of 32 and 3 as channel width. For the sake of explanation, we will consider the input size as 224 x 224 x 3. Every ResNet architecture performs the initial convolution and max-pooling using 7×7 and 3×3 kernel sizes respectively. Afterward, Stage 1 of the network starts and

it has 3 Residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block of stage 1 are 64, 64 and 128 respectively. The curved arrows refer to the identity connection. The dashed connected arrow represents that the convolution operation in the Residual Block is performed with stride 2, hence, the size of input will be reduced to half in terms of height and width but the channel width will be doubled. As we progress from one stage to another, the channel width is doubled and the size of the input is reduced to half.

For deeper networks like ResNet50, ResNet152, etc, bottleneck design is used. For each residual function F, 3 layers are stacked one over the other. The three layers are 1×1, 3×3, 1×1 convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions.

Finally, the network has an Average Pooling layer followed by a fully connected layer having 1000 neurons (ImageNet class output).

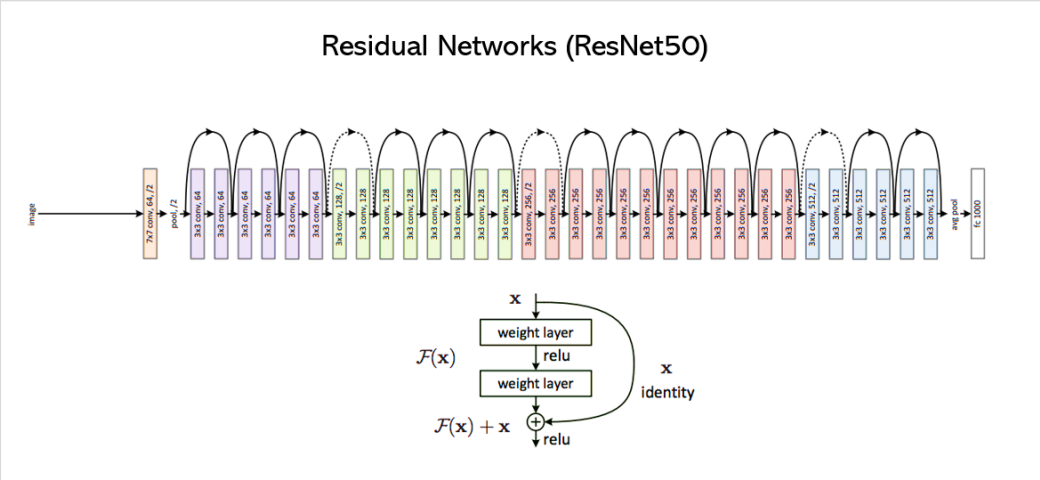


Fig 2.7.1

2.7.2 Time-Distributed Convolutional Neural Nets and Recurrent Neural Nets (For Speech Emotion Recognition)

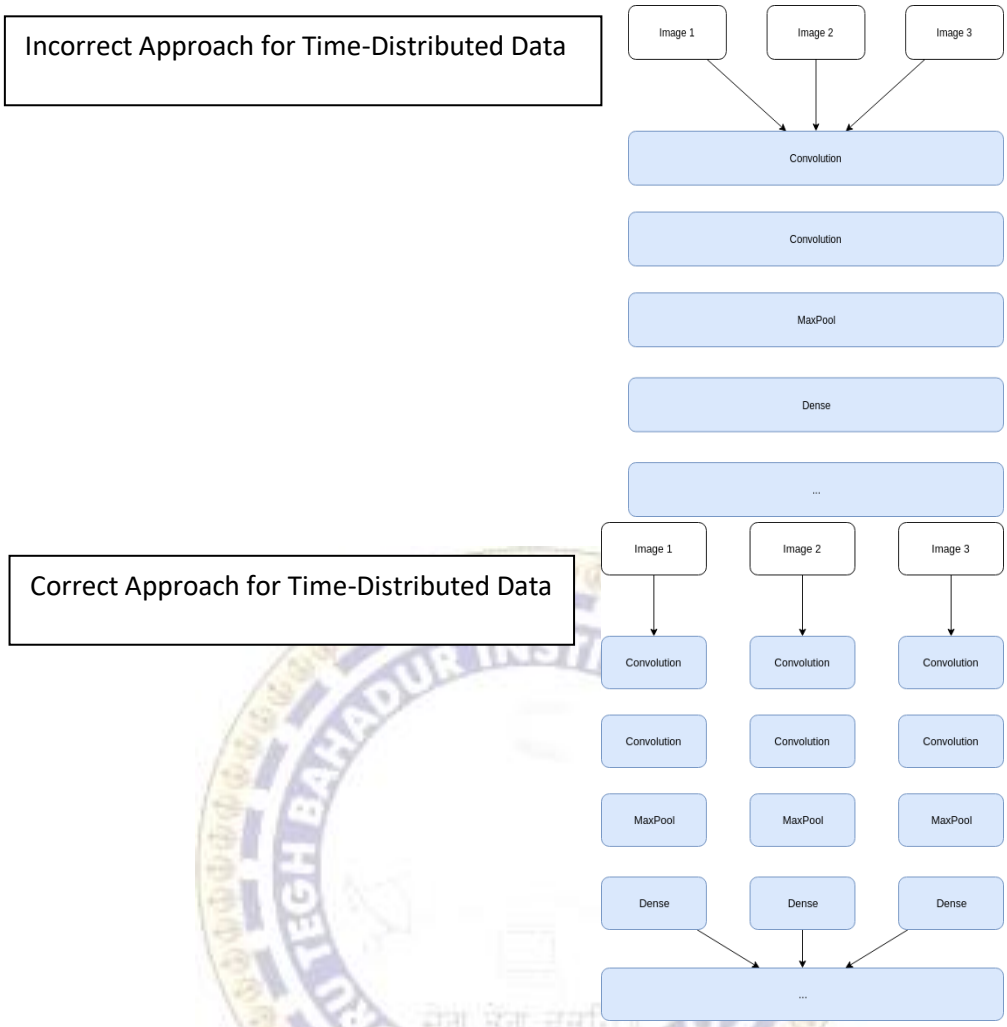


Fig 2.7.2.1

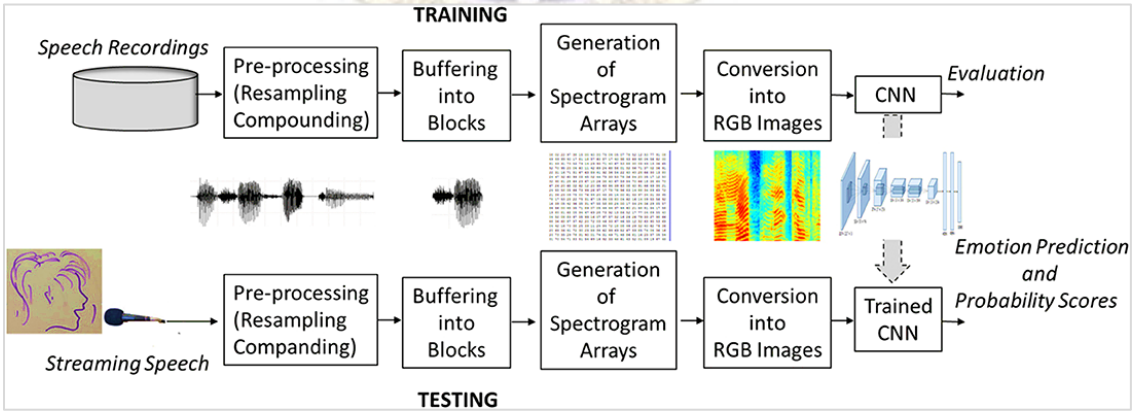


Fig 2.7.2.2

2.7.3 1-D CNN and LSTM for Perosnality Assessment on the basis of Text

2.7.3.1 Difference between Traditional Machine Learning Approach and Deep Learning Approach for Text Sentiment Analysis

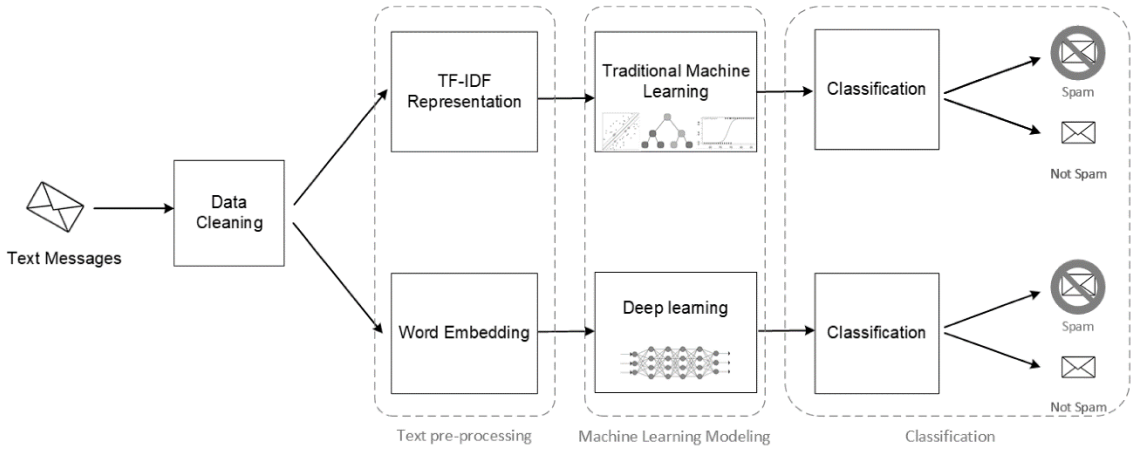


Fig 2.7.3.1

2.7.3.2 Pipeline for Text Classification

Convolutional neural networks excel at learning the spatial structure in input data. The IMDB review data does have a one-dimensional spatial structure in the sequence of words in reviews and the CNN may be able to pick out invariant features for good and bad sentiment. This learned spatial features may then be learned as sequences by an LSTM layer.

We can easily add a one-dimensional CNN and max pooling layers after the Embedding layer which then feed the consolidated features to the LSTM. We can use a smallish set of 32 features with a small filter length of 3. Using model checkpoint and callbacks, we are saving the model weights when validation accuracy is maximum. The pooling layer can use the standard length of 2 to halve the feature map size.

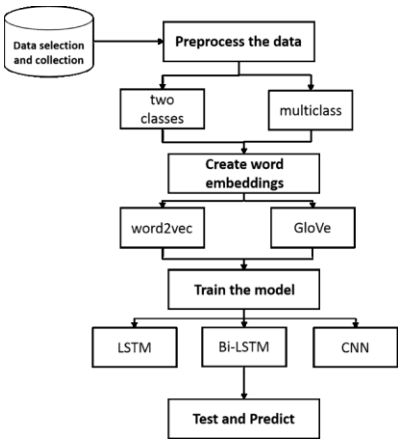


Fig 2.7.3.2

2.7.3.3 Pre-Trained Word2Vec Embeddings

Word embeddings are widely used in machine learning based natural language processing systems. It is common to use pre-trained word embeddings which provide benefits such as reduced training time and improved overall performance. There has been a recent interest in applying natural language processing techniques to programming languages. However, none of this recent work uses pre-trained embeddings on code tokens. Using extreme summarization as the downstream task, it has been show that using pre-trained embeddings on code tokens provides the same benefits as it does to natural languages, achieving: over 1.9x speedup, 5\% improvement in test loss, 4\% improvement in F1 scores, and resistance to over-fitting. It has also been shown that the choice of language used for the embeddings does not have to match that of the task to achieve these benefits and that even embeddings pre-trained on human languages provide these benefits to programming languages

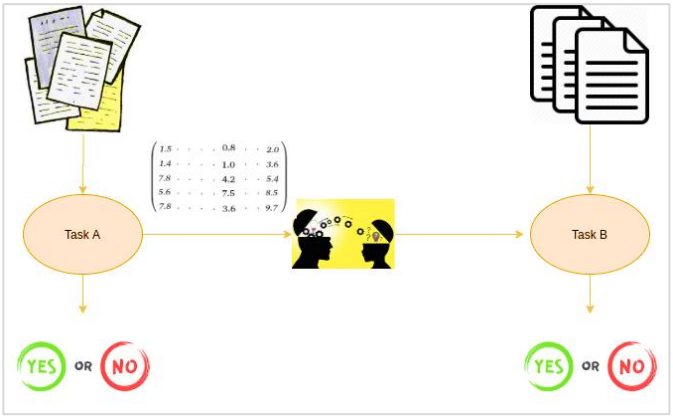


Fig. 2.7.3.3

3. Development Environment

3.1 Google Colab

What is Colab?

Colab, or 'Colaboratory', allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

The interface is as below:-

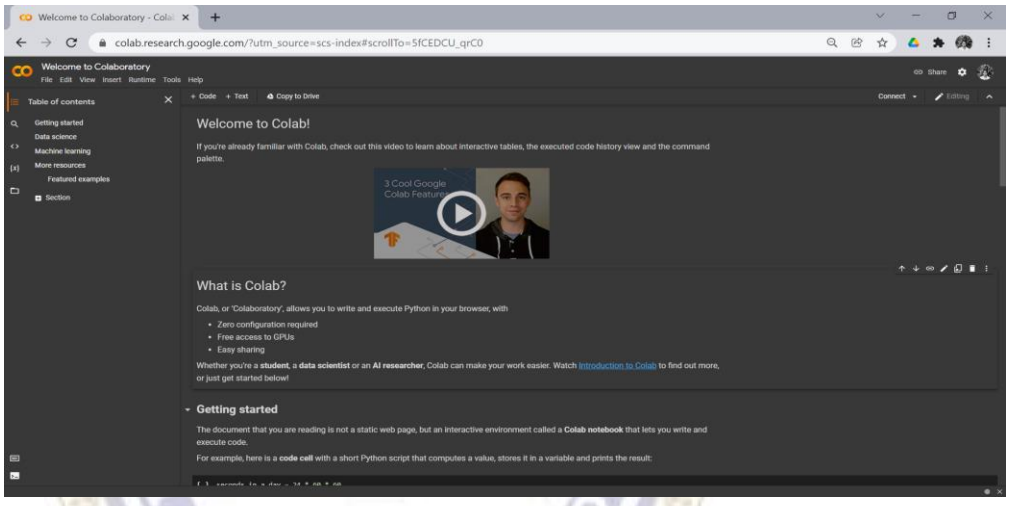


Fig 3.1.1

Few Snippets from the Project are as below:-

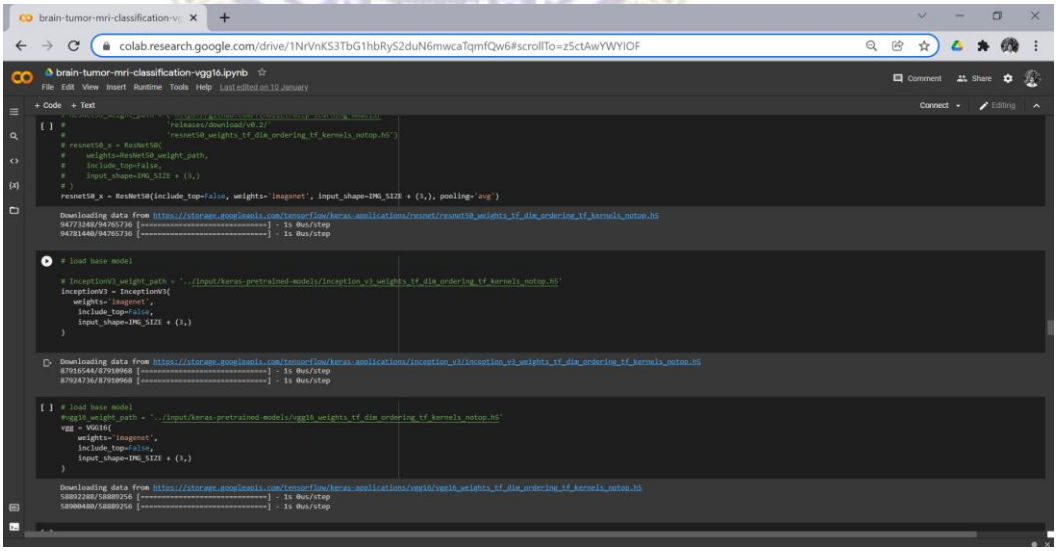


Fig 3.1.2

3.1.1 GPU Support

Google is quite aggressive in AI research. Over many years, Google developed AI framework called TensorFlow and a development tool called Colaboratory. Today TensorFlow is open-sourced and since 2017, Google made Colaboratory free for public use. Colaboratory is now known as Google Colab or simply Colab.Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold per-use basis.Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

3.2 Kaggle

Kaggle is an AirBnB for Data Scientists – this is where they spend their nights and weekends. It's a crowd-sourced platform to attract, nurture, train and challenge data scientists from all around the world to solve data science, machine learning and predictive analytics problems. It has over 536,000 active members from 194 countries and it receives close to 150,000 submissions per month. Started from Melbourne, Australia Kaggle moved to Silicon Valley in 2011, raised some 11 million dollars from the likes of Hal Varian (Chief Economist at Google), Max Levchin (Paypal), Index and Khosla Ventures and then ultimately been acquired by the Google in March of 2017. Kaggle is the number one stop for data science enthusiasts all around the world who compete for prizes and boost their Kaggle rankings. There are only 94 Kaggle Grandmasters in the world to this date.

Do you know that most data scientists are only theorists and rarely get a chance to practice before being employed in the real-world? Kaggle solves this problem by giving data science enthusiasts a platform to interact and compete in solving real-life problems. The experience you get on Kaggle is invaluable in preparing you to understand what goes into finding feasible solutions for big data.

Kaggle enables data scientists and other developers to engage in running machine learning contests, write and share code, and to host datasets. The types of data science problems posted on Kaggle can be anything from attempting to predict cancer occurrence by examining patient

records to analyzing sentiment to evoke by movie reviews and how this affects audience reaction.

Different sources post projects on this trailblazing platform. While some are just for educational purposes and fun brain exercises, others are genuine issues that companies are trying to solve. Kaggle makes the environment competitive by awarding prizes and rankings for winners and participants. The prizes are not only monetary but can also include attractive rewards such as jobs or free products from the company hosting the competition.

Monetary prices are exciting to most Kagglers. For instance, Home Depot was offering a winning prize of a whopping $40,000 in search of an algorithm to improve search results on homedepot.com. For most data science enthusiasts, this innovative website is not only a monetary resource, but it is also an indispensable learning tool that helps improve the experience, gain knowledge, elevate and enhance the skills, and learn from mistakes by resubmitting the code. It is the perfect platform to practice consistently.

The Kaggle community is growing fast. There are currently over one million Kaggle members (Kagglers). This data community has submitted above four million learning models to different competitions. Kaggle users have shared over one thousand datasets, more than 170,000 forum posts and over 250 kernels. According to the founder, this incredibly fast growth can be attributed to high-quality content, data, and code shared by Kagglers.

Most Kaggle users are committed and active hence the 4,000 forum posts per month and more than 3,500 competition submissions on a daily basis. This platform is the place to be for data scientists and machine learning engineers worldwide.

3.2.1 Dataset

The Dataset for the Facial Emotion Recognition has been taken from Kaggle. The Dataset is the FER-2013 Dataset.

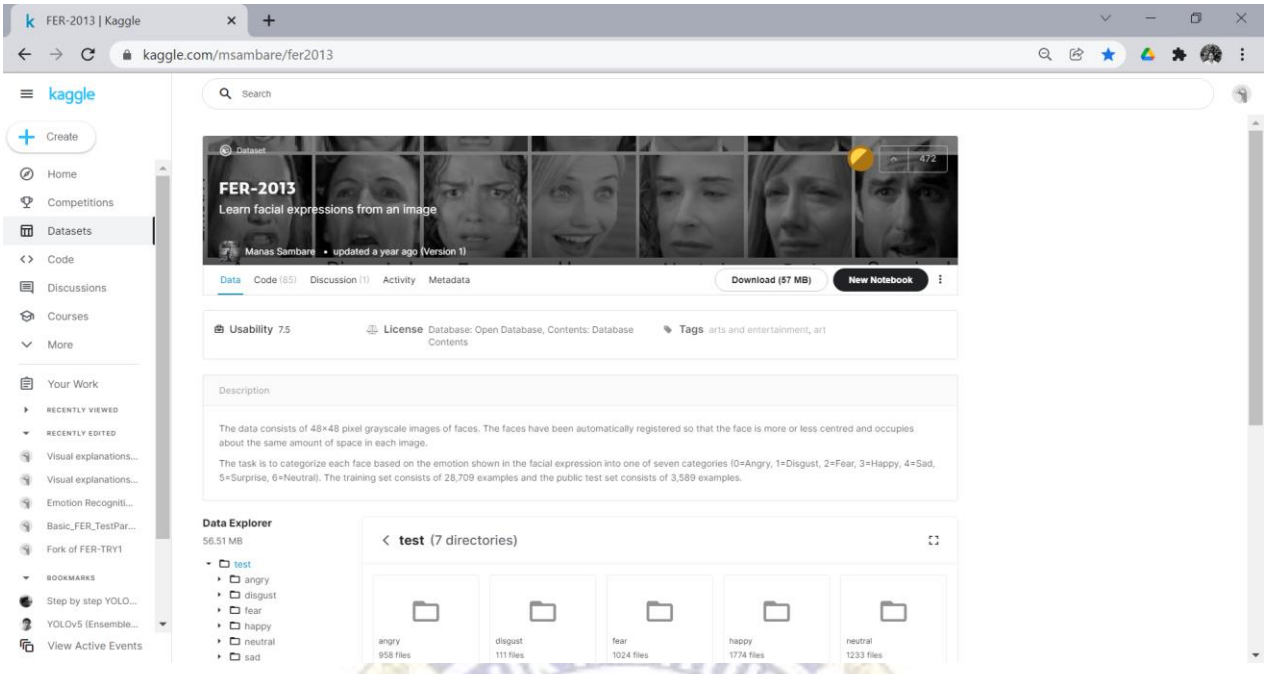
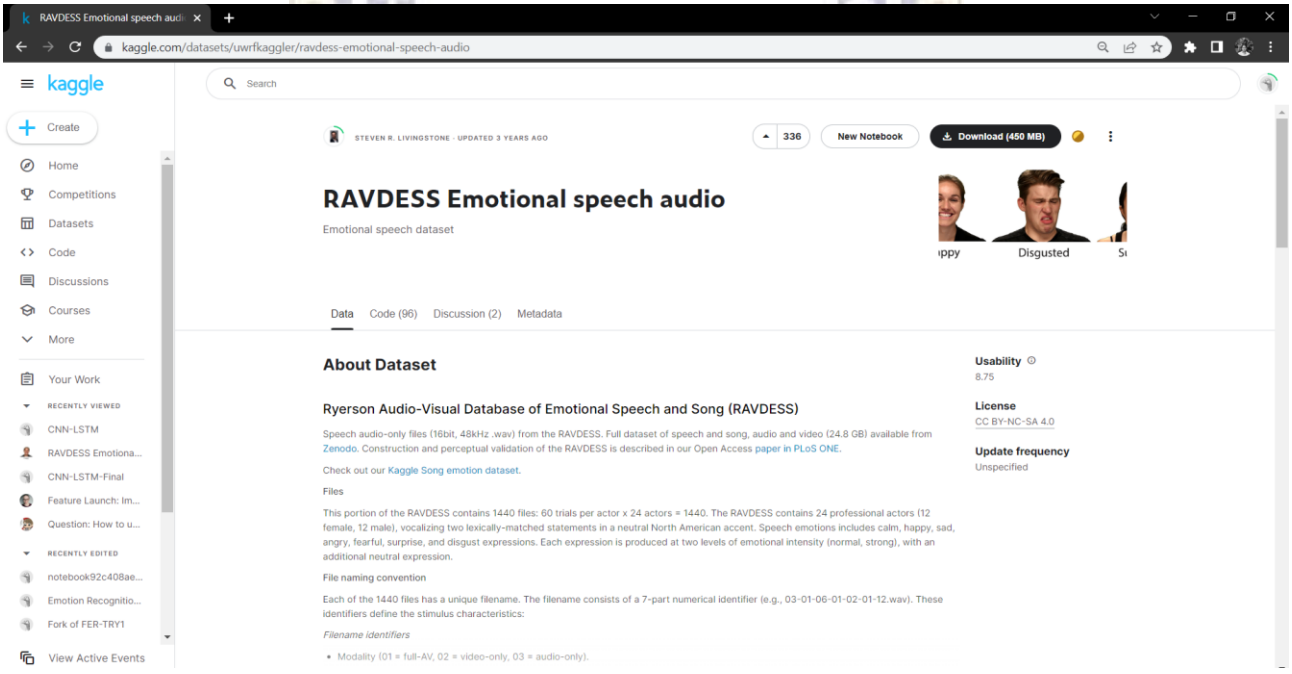


Fig 3.2.1

The Dataset for the Audial Emotion Recognition has been taken from Kaggle. The Dataset is the RAVDESS Dataset for Emotional Speech.



4. Methodology

An overview of the entire process for visual input analysis, developed as a part of Major Project is as specified:

Task 1: Introduction and Overview

- Introduction to the data and and overview of the task which is visual input analysis, audial input analysis and textual input analysis.
- Import amd understand the use of essential modules and helper functions from NumPy, Matplotlib, and Keras.

Task 2: Explore the Dataset

- Display and Plot the dataset to understand imbalanced dataset or clean the dataset or augment the dataset for efficient results.
- Check for class imbalance problems in the training data.

Task 3: Generate Training and Validation Batches

- Generate batches of tensor image data with real-time data augmentation.
- Specify paths to training and validation image directories and generates batches of augmented data.

Task 4: Create a Model Design

- Design an appropriate Neural Network Model according as the requirement of classification purposes.
- Use Adam as the optimizer, categorical crossentropy as the loss function, and accuracy as the evaluation metric.

Task 5: Train and Evaluate Model

- Train the CNN by invoking the model.fit() method.
- Use ModelCheckpoint() to save the weights associated with the higher validation accuracy.
- Observe live training loss and accuracy plots in Jupyter Notebook for Keras.

Task 6: Save and Serialize Model as JSON String

- Sometimes, you are only interested in the architecture of the model, and you don't need to save the weight values or the optimizer.
- Use to_json(), which uses a JSON string, to store the model architecture.

Task 7: Create a Flask App to Serve Predictions

- Use open-source code from Flask Repositories to create a flask app to serve the model's prediction images directly to a web interface.

Task 8: Create a Class to Output Model Predictions

- Create a FacialExpressionModel class to load the model from the JSON file, load the trained weights into the model, and predict facial expressions.

Task 9: Design an HTML Template for the Flask App. Add CSS and JS to add interactive Web Interface.

Task 10: Use Model to Recognize Facial Expressions in Videos, Emotion in audial input and psycological traits for text.

- Run the main.py script to create the Flask app and serve the model's predictions to a web interface.
- Apply the model to Real-Time Input.

Task11: Analyze the output for both the models for Accuracy and Loss, Confusion Matrix, and Top-K Accuracy for wherever required.

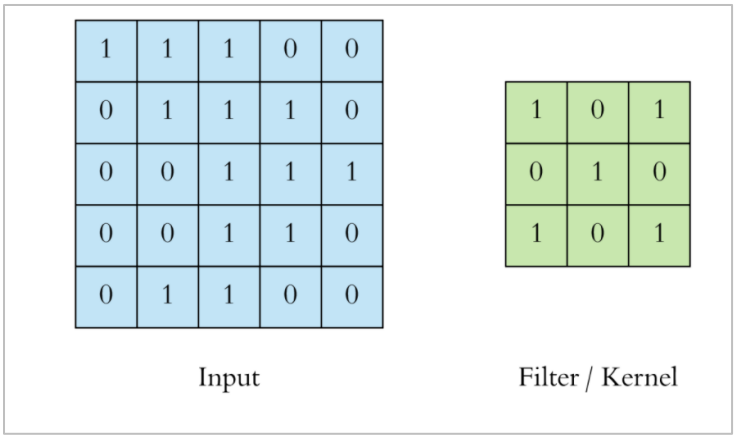
Task12: Visualize the Output by GRAD-CAM for understanding the mechanism of CNNs.

We can use Grad-CAM to identify bias in our model's predictions which was passed by the training dataset. Considering the sample example as in the paper, suppose we've trained a nurse vs. doctor image classifier from images taken from the ImageNet dataset. The classifier would perform pretty well on the testing dataset but doesn't generalize well. Well, how do we know this? Look at the Grad-CAM heatmaps in figure 12, our the biased model, we can observe that the model focuses on unwanted features (part of the face, hair) which seem insignificant while predicting labels 'nurse' and 'doctor'. The model seems to learn gender-based features which aren't appropriate in our use-case.

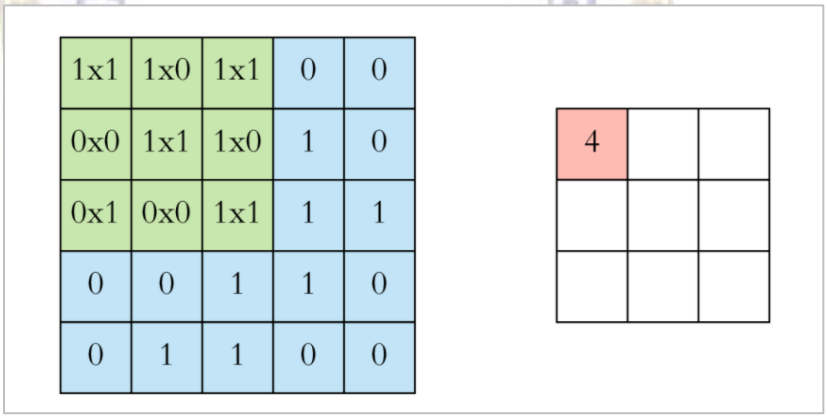
The model can be generalised or can be viewed as the model below.All CNN models follow a similar architecture, as shown in the figure above and now we disucss the architechtural design.

Convolution

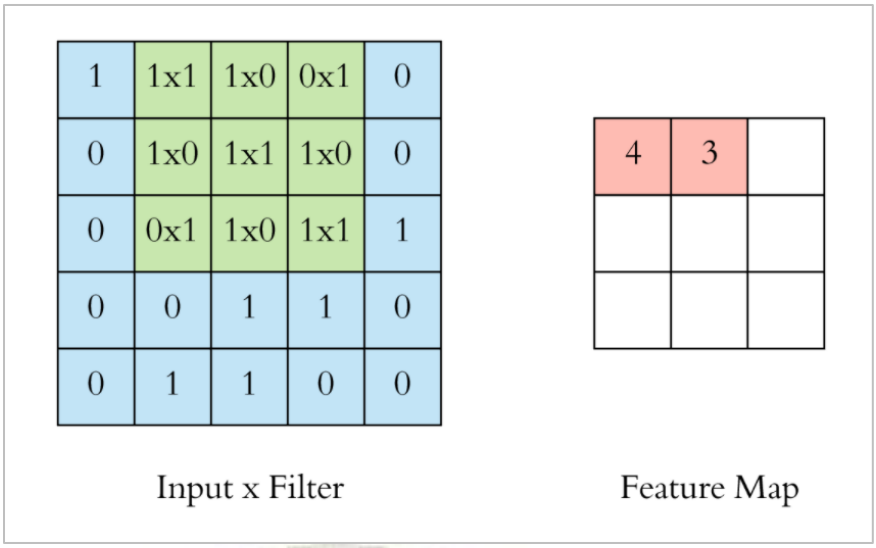
The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. In our case the convolution is applied on the input data using a convolution filter to produce a feature map.



On the left side is the input to the convolution layer, for example the input image. On the right is the convolution filter, also called the kernel, we will use these terms interchangeably. This is called a 3x3 convolution due to the shape of the filter.We perform the convolution operation by sliding this filter over the input. At every location, we do element-wise matrix multiplication and sum the result. This sum goes into the feature map. The green area where the convolution operation takes place is called the receptive field. Due to the size of the filter the receptive field is also 3x3 .

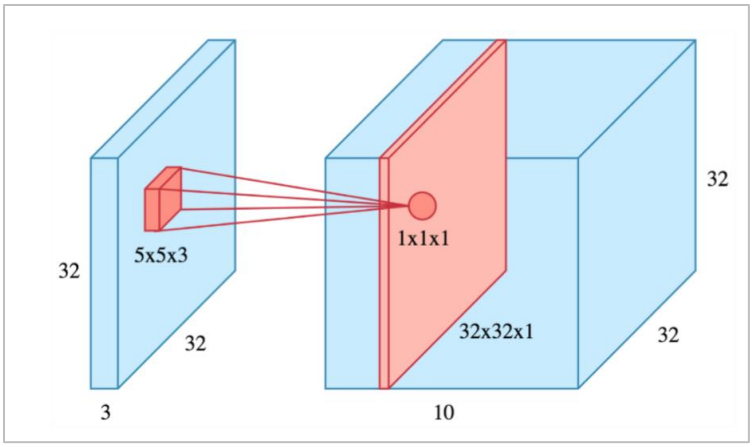


Here the filter is at the top left, the output of the convolution operation “4” is shown in the resulting feature map. We then slide the filter to the right and perform the same operation, adding that result to the feature map as well.



This was an example convolution operation shown in 2D using a 3x3 filter. But in reality these convolutions are performed in 3D. In reality an image is represented as a 3D matrix with dimensions of height, width and depth, where depth corresponds to color channels (RGB). A convolution filter has a specific height and width, like 3x3 or 5x5, and by design it covers the entire depth of its input so it needs to be 3D as well. This was an example convolution operation shown in 2D using a 3x3 filter. But in reality these convolutions are performed in 3D. In reality an image is represented as a 3D matrix with dimensions of height, width and depth, where depth corresponds to color channels (RGB). A convolution filter has a specific height and width, like 3x3 or 5x5, and by design it covers the entire depth of its input so it needs to be 3D as well.

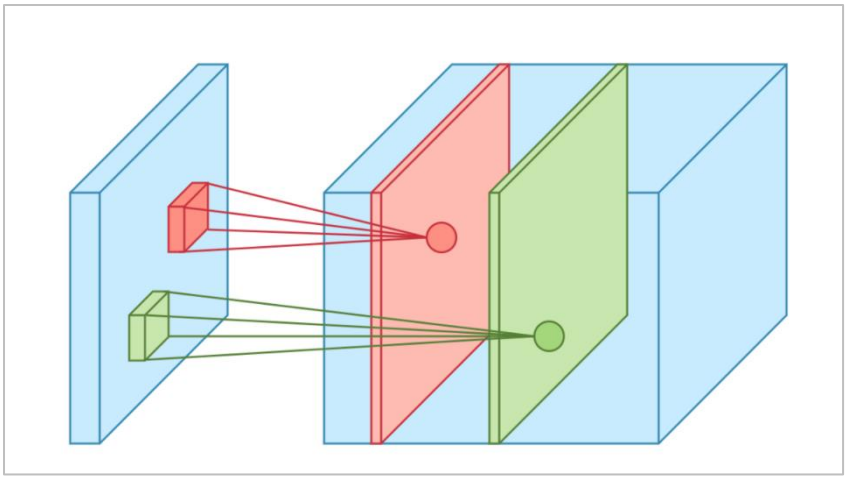
One more important point before we visualize the actual convolution operation. We perform multiple convolutions on an input, each using a different filter and resulting in a distinct feature map. We then stack all these feature maps together and that becomes the final output of the convolution layer. But first let's start simple and visualize a convolution using a single filter.



Let's say we have a 32x32x3 image and we use a filter of size 5x5x3 (note that the depth of the convolution filter matches the depth of the image, both being 3). When the filter is at a particular location it covers a small volume of the input, and we perform the convolution operation described above. The only difference is this time we do the sum of matrix multiply in 3D instead of 2D, but the result is still a scalar. We slide the filter over the input like above and perform the convolution at every location aggregating the result in a feature map. This feature map is of size 32x32x1, shown as the red slice on the right.

If we used 10 different filters we would have 10 feature maps of size 32x32x1 and stacking them along the depth dimension would give us the final output of the convolution layer: a volume of size 32x32x10, shown as the large blue box on the right. Note that the height and width of the feature map are unchanged and still 32, it's due to padding and we will elaborate on that shortly. To help with visualization, we slide the filter over the input as follows. At each location we get a scalar and we collect them in the feature map. The animation shows the sliding operation at 4 locations, but in reality it's performed over the entire input.

Below we can see how two feature maps are stacked along the depth dimension. The convolution operation for each filter is performed independently and the resulting feature maps are disjoint.

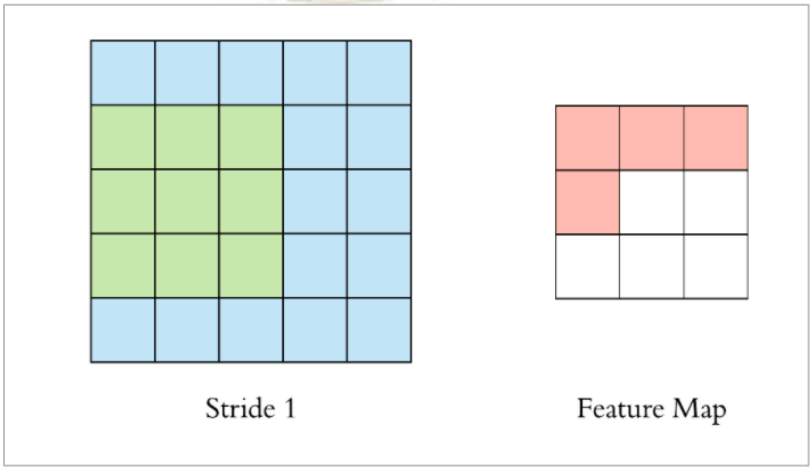


4.a Non-linearity

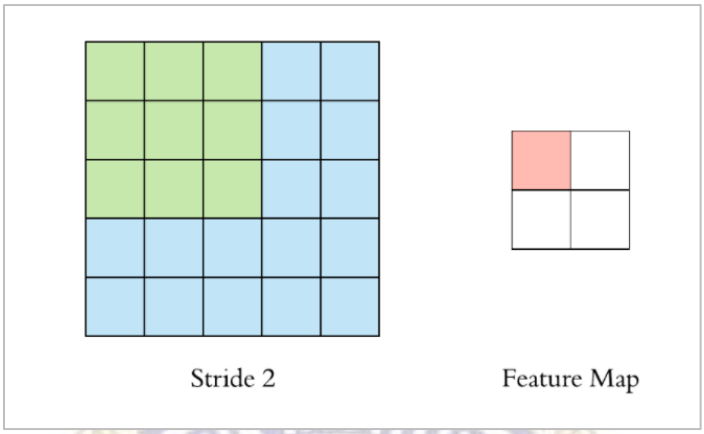
For any kind of neural network to be powerful, it needs to contain non-linearity. Both the ANN and auto-encoder we saw before achieved this by passing the weighted sum of its inputs through an activation function, and CNN is no different. We again pass the result of the convolution operation through relu activation function. So the values in the final feature maps are not actually the sums, but the relu function applied to them. But keep in mind that any type of convolution involves a relu operation, without that the network won't achieve its true potential.

4.b Stride and Padding

Stride specifies how much we move the convolution filter at each step. By default the value is 1, as you can see in the figure below.

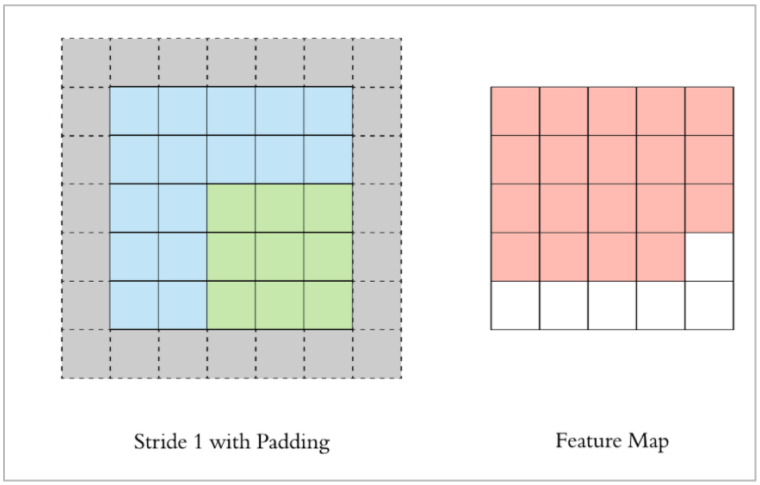


We can have bigger strides if we want less overlap between the receptive fields. This also makes the resulting feature map smaller since we are skipping over potential locations. The following figure demonstrates a stride of 2. Note that the feature map got smaller.



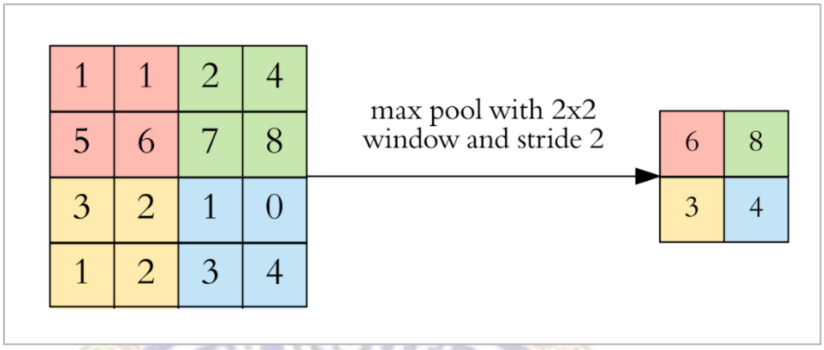
We see that the size of the feature map is smaller than the input, because the convolution filter needs to be contained in the input. If we want to maintain the same dimensionality, we can use padding to surround the input with zeros.

The gray area around the input is the padding. We either pad with zeros or the values on the edge. Now the dimensionality of the feature map matches the input. Padding is commonly used in CNN to preserve the size of the feature maps, otherwise they would shrink at each layer, which is not desirable. The 3D convolution figures we saw above used padding, that's why the height and width of the feature map was the same as the input (both 32x32), and only the depth changed.

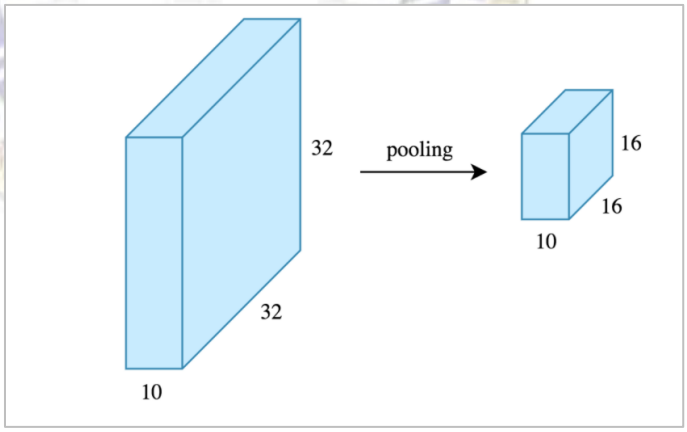


4.c Pooling

After a convolution operation we usually perform pooling to reduce the dimensionality. This enables us to reduce the number of parameters, which both shortens the training time and combats over-fitting. Pooling layers down-sample each feature map independently, reducing the height and width, keeping the depth intact.



The most common type of pooling is max pooling which just takes the max value in the pooling window. Contrary to the convolution operation, pooling has no parameters. It slides a window over its input, and simply takes the max value in the window. Similar to a convolution, we specify the window size and stride.



Here is the result of max pooling using a 2x2 window and stride 2. Each color denotes a different window. Since both the window size and stride are 2, the windows are not overlapping.

Note that this window and stride configuration halves the size of the feature map. This is the main use case of pooling, down-sampling the feature map while keeping the important information.

Now let's work out the feature map dimensions before and after pooling. If the input to the pooling layer has the dimensionality 32x32x10, using the same pooling parameters described above, the result will be a 16x16x10 feature map. Both the height and width of the feature map are halved, but the depth doesn't change because pooling works independently on each depth slice the input.

4.d Hyper-parameters

Let's now only consider a convolution layer ignoring pooling, and go over the hyper-parameter choices we need to make. We have 4 important hyper-parameters to decide on:

- Filter size: we typically use 3x3 filters, but 5x5 or 7x7 are also used depending on the application. There are also 1x1 filters which we will explore in another article, at first sight it might look strange but they have interesting applications. Remember that these filters are 3D and have a depth dimension as well, but since the depth of a filter at a given layer is equal to the depth of its input, we omit that.
- Filter count: this is the most variable parameter, it's a power of two anywhere between 32 and 1024. Using more filters results in a more powerful model, but we risk over-fitting due to increased parameter count. Usually we start with a small number of filters at the initial layers, and progressively increase the count as we go deeper into the network.
- Stride: we keep it at the default value 1.
- Padding: we usually use padding.

4.e Fully Connected

After the convolution + pooling layers we add a couple of fully connected layers to wrap up the CNN architecture. This is the same fully connected ANN architecture we talked about in Part 1.

Remember that the output of both convolution and pooling layers are 3D volumes, but a fully connected layer expects a 1D vector of numbers. So we flatten the output of the final pooling layer to a vector and that becomes the input to the fully connected layer. Flattening is simply arranging the 3D volume of numbers into a 1D vector, nothing fancy happens here.

4.f Training

CNN is trained the same way like ANN, back-propagation with gradient descent. Due to the convolution operation it's more mathematically involved, and it's out of the scope for this article. If you're interested in the details refer here.

4.g Intuition

A CNN model can be thought as a combination of two components: feature extraction part and the classification part. The convolution + pooling layers perform feature extraction. For example given an image, the convolution layer detects features such as two eyes, long ears, four legs, a short tail and so on. The fully connected layers then act as a classifier on top of these features, and assign a probability for the input image being a dog.

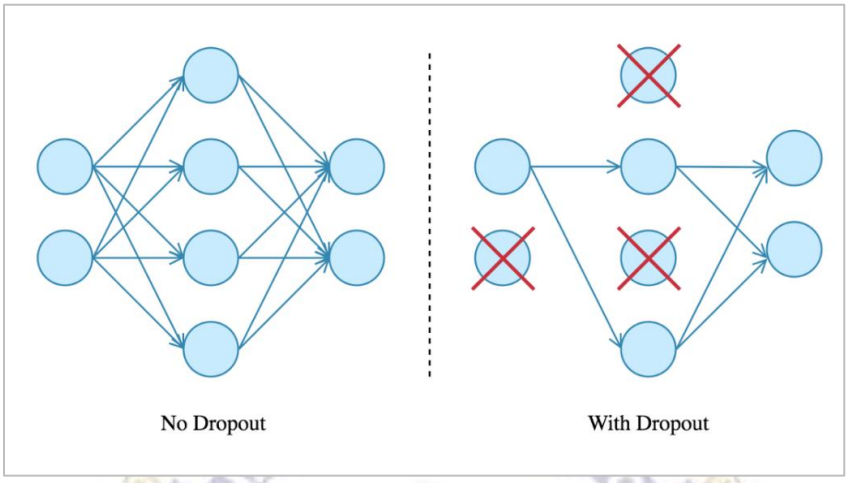
The convolution layers are the main powerhouse of a CNN model. Automatically detecting meaningful features given only an image and a label is not an easy task. The convolution layers learn such complex features by building on top of each other. The first layers detect edges, the next layers combine them to detect shapes, to following layers merge this information to infer that this is a nose. To be clear, the CNN doesn't know what a nose is. By seeing a lot of them in images, it learns to detect that as a feature. The fully connected layers learn how to use these features produced by convolutions in order to correctly classify the images.

Structurally the code looks similar to the ANN we have been working on. There are 4 new methods we haven't seen before:

- Conv2D: this method creates a convolutional layer. The first parameter is the filter count, and the second one is the filter size. For example in the first convolution layer we create 32 filters of size 3x3. We use Re-Lu non-linearity as activation. We also enable padding. In Keras , there are two options for padding: same or valid. Same means we pad with the number on the edge and valid means no padding. Stride is 1 for convolution layers by default so we don't change that. This layer can be customized further with additional parameters; you can check the documentation here.
- MaxPooling2D: creates a max-pooling layer, the only argument is the window size. We use a 2x2 window as it's the most common. By default stride length is equal to the window size, which is 2 in our case, so we don't change that.
- Flatten: After the convolution + pooling layers we flatten their output to feed into the fully connected layers as we discussed above.
- Dropout: we will explain this in the next section.

4.h Dropout

Dropout is by far the most popular regularization technique for deep neural networks. Even the state-of-the-art models which have 95% accuracy get a 2% accuracy boost just by adding dropout, which is a fairly substantial gain at that level.



Dropout is used to prevent over-fitting and the idea is very simple. During training time, at each iteration, a neuron is temporarily "dropped" or disabled with probability p. This means all the inputs and outputs to this neuron will be disabled at the current iteration. The dropped-out neurons are resampled with probability p at every training step, so a dropped out neuron at one step can be active at the next one. The hyper-parameter p is called the dropout-rate and it's typically a number around 0.5, corresponding to 50% of the neurons being dropped out.

4.1 Visulaize the Data

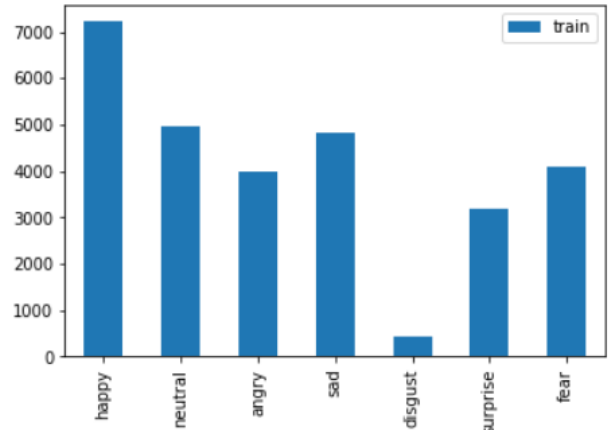
The FER-2013 Data can be visualized as follows :-

The data consists of 48*48 pixel grayscale images of faces.The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

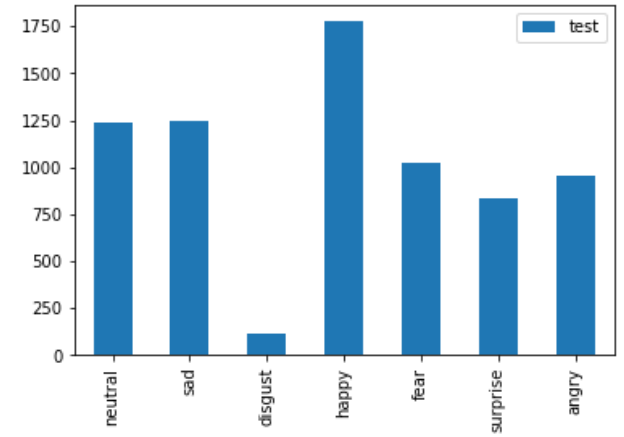
Emotion	Neutral	Sad	Disgust	Happy	Fear	Surprise	Angry
Train Test	4965	4830	436	7215	4097	3171	3995
Test Set	1233	1247	111	1774	1024	831	958

Table

4.1.1 – Data Enumeration (Overview) – For each of the LABEL



Plot – I (Images in Training Set) – For 7 Classes

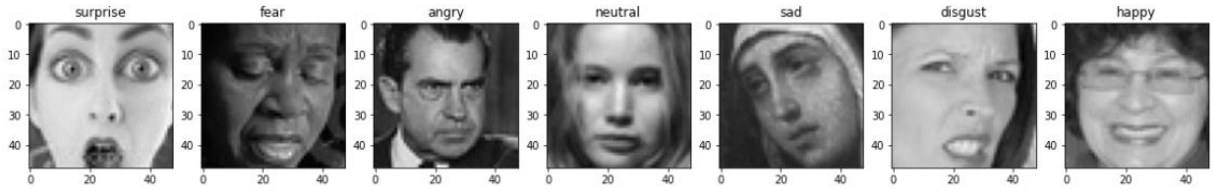


Plot – I (Images in Test Set) – For 7 Classes

(Fig-4.1.1)

The images in the Data-Set are shown as below as plot(s):-

Visualizations of the Data as Input Image(s) in the FER2013 Data-Set , without plots.



Data in the FER2013, displayed as PLOTS (48 x 48). Note that all the images are gray-scale images.

(Fig-4.1.2)

The RAVDESS Dataset has been used to train the model for Speech Emotion Analysis for vocal input. The RAVDESS Dataset is introduced and used as per the format specified below: -

Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)

Speech audio-only files (16bit, 48kHz .wav) from the RAVDESS. Full dataset of speech and song, audio and video (24.8 GB) available from Zenodo. Construction and perceptual validation of the RAVDESS is described in our Open Access paper in PLoS ONE.

Files

This portion of the RAVDESS contains 1440 files: 60 trials per actor x 24 actors = 1440. The RAVDESS contains 24 professional actors (12 female, 12 male), vocalizing two lexically-matched statements in a neutral North American accent. Speech emotions includes calm, happy, sad, angry, fearful, surprise, and disgust expressions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression.

File naming convention

Each of the 1440 files has a unique filename. The filename consists of a 7-part numerical identifier (e.g., 03-01-06-01-02-01-12.wav). These identifiers define the stimulus characteristics:

Filename identifiers

- Modality (01 = full-AV, 02 = video-only, 03 = audio-only).
- Vocal channel (01 = speech, 02 = song).
- Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).

- Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
- Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
- Repetition (01 = 1st repetition, 02 = 2nd repetition).
- Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

Filename example: 03-01-06-01-02-01-12.wav

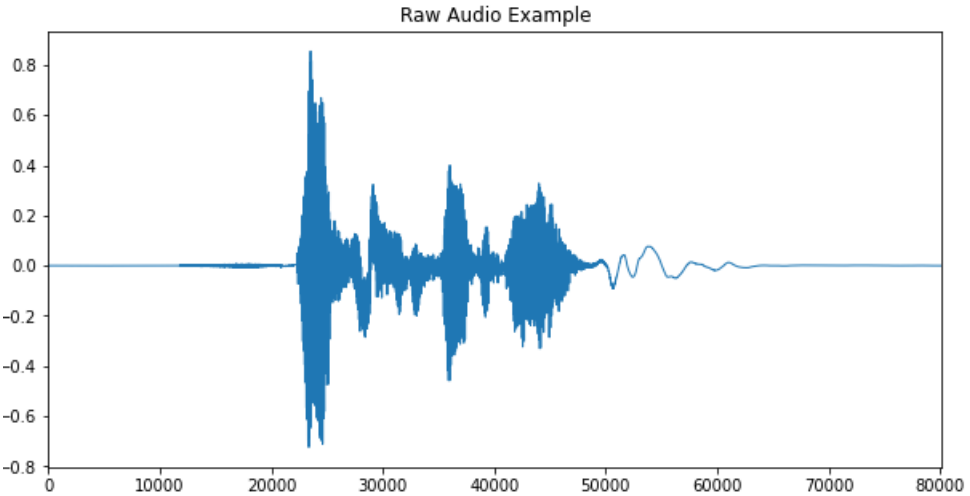
1. Audio-only (03)
2. Speech (01)
3. Fearful (06)
4. Normal intensity (01)
5. Statement "dogs" (02)
6. 1st Repetition (01)
7. 12th Actor (12)
   Female, as the actor ID number is even.

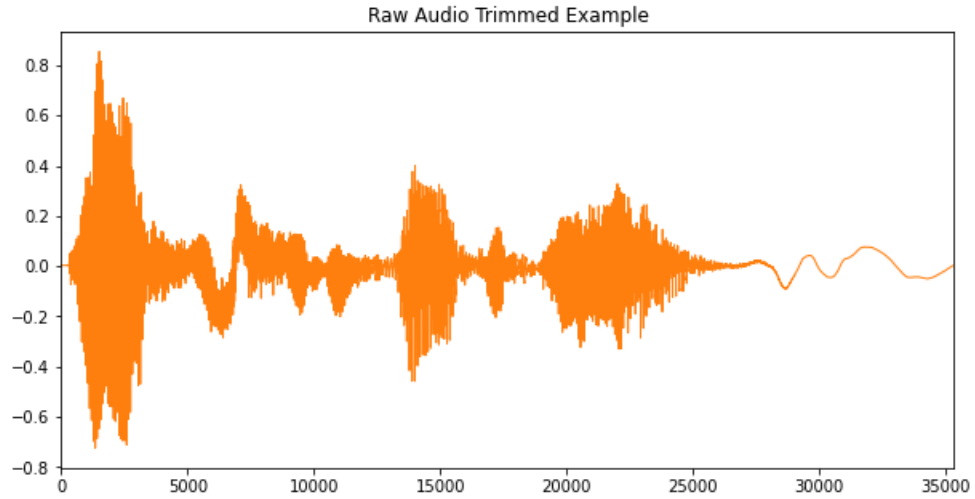
The Dataset is distributed across different labels as follows: -

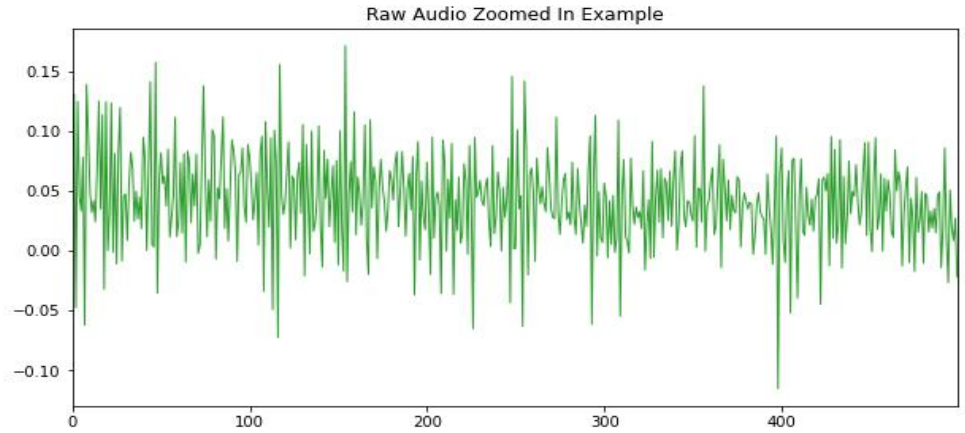
female_neutral	144
male_neutral	144
male_sad	96
male_happy	96
male_fear	96
female_happy	96
female_angry	96
female_fear	96
female_disgust	96
male_disgust	96
male_angry	96
female_surprise	96
male_surprise	96
female_sad	96

(Fig 4.1.3)

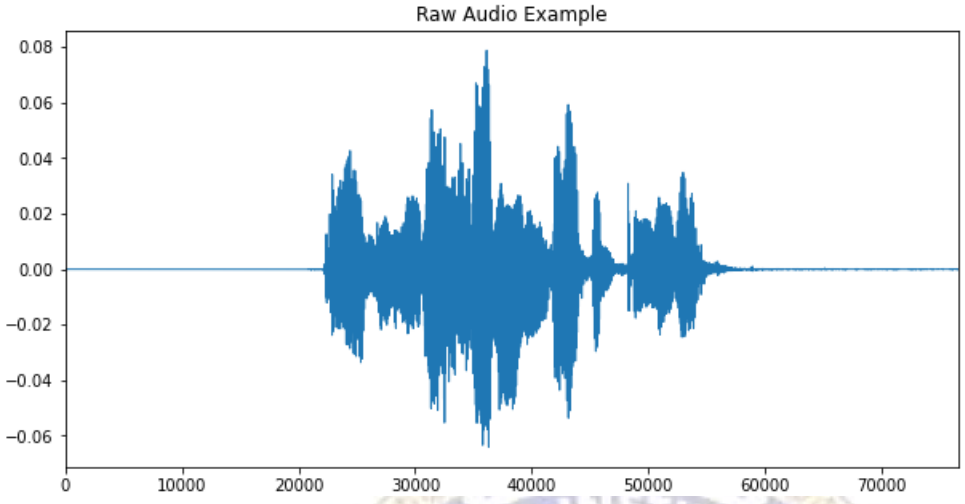
Example - 1

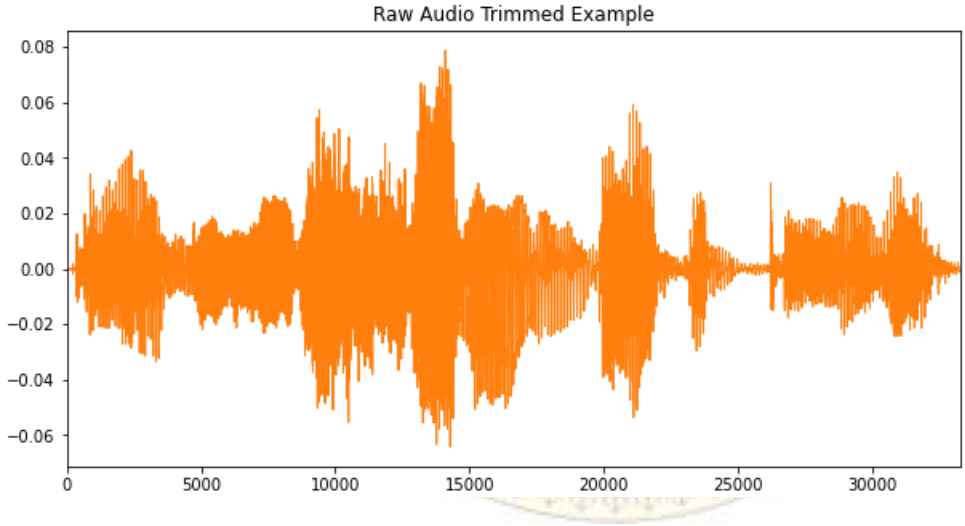


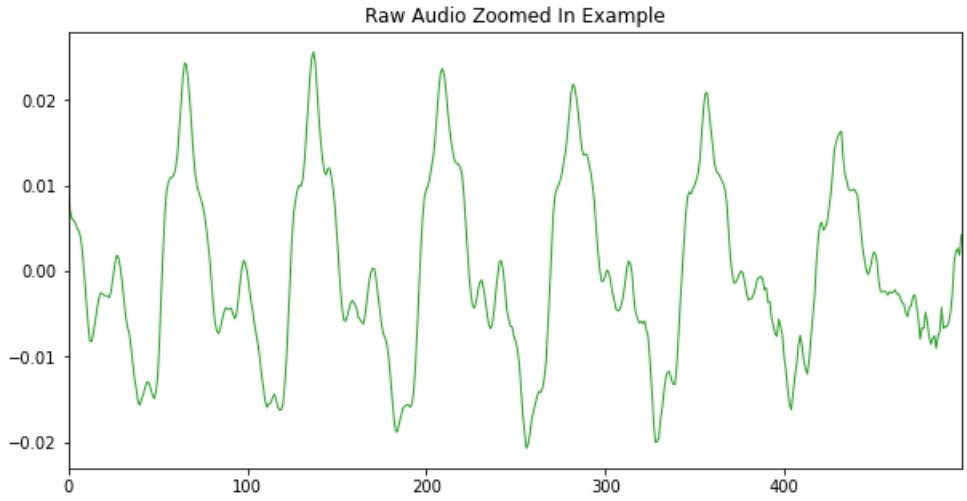




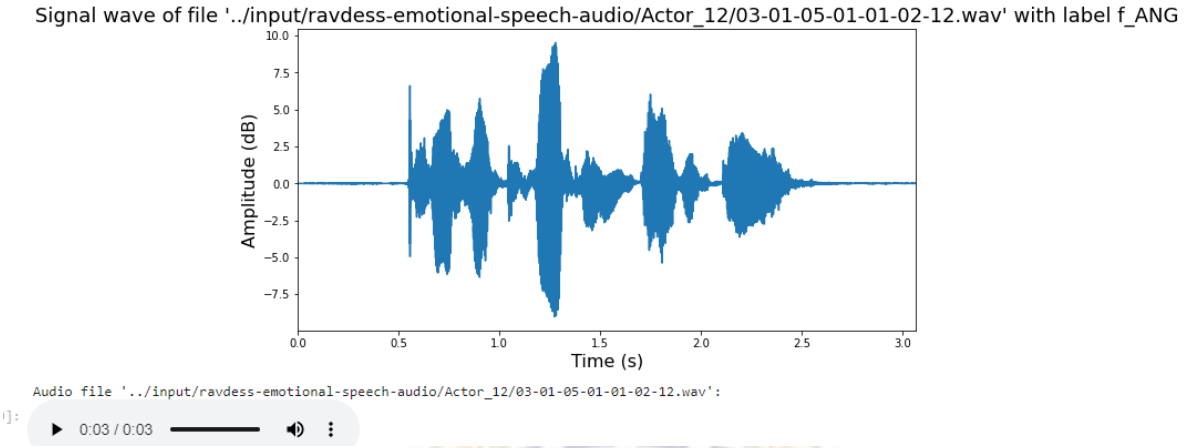
Example-2


Raw Audio Example
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0
10000
20000
30000
40000
50000
60000
70000

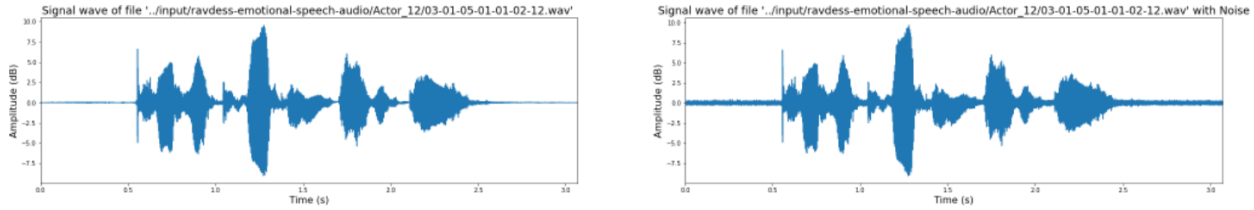

Raw Audio Trimmed Example
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0
5000
10000
15000
20000
25000
30000


Raw Audio Zoomed In Example
0.02
0.01
0.00
−0.01
−0.02
0
100
200
300
400

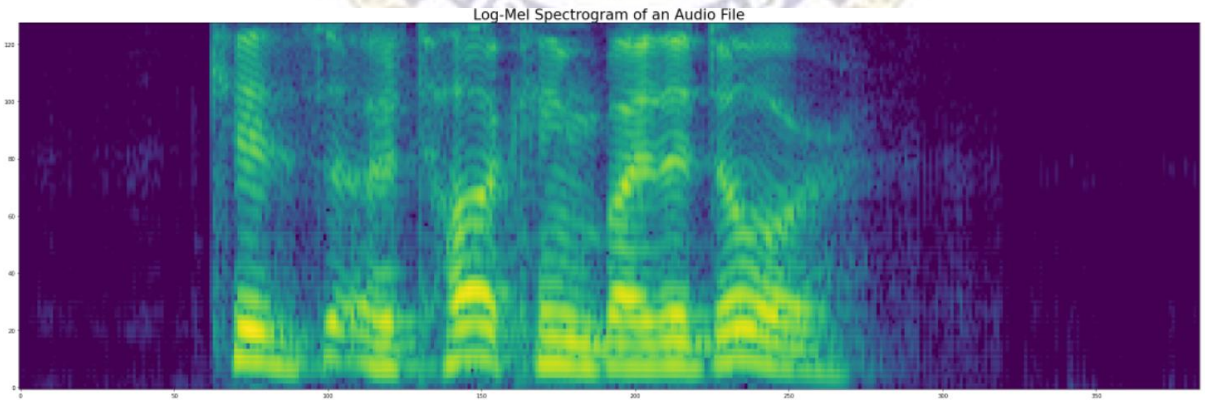
(Fig 4.1.4)



Data Augmentation used to add noise to the data is used as follows: -



The corresponding Log-Mel Spectrogram is as follows:-



(Fig 4.1.5)

The Dataset used for Personality Assessment to identify the percentage of existence of five characteristic personality traits is the **Stream of Consciousness** Dataset described and used as specified below.

In the context of our study, we chose to use text mining in order not to detect regular emotions such as disgust or surprise, but to recognize personality traits based on the "Big Five" model in psychology. Even though emotion recognition and personality traits classification are two separate fields of studies based on different theoretical underpinnings, they use similar learning-based methods and literature from both areas can be interesting. The main motivation behind this choice is to offer a broader assessment to the user : as emotions can only be understood in the light of a person's own characteristics, we thought that analyzing personality traits would provide a new key to understanding emotional fluctuations. Our final goal is to enrich the user experience and improve the quality of our analysis : any appropriate and complementary information deepening our understanding of the user's idiosyncrasies is welcomed.

Many psychology researchers (starting with D. W. Fiske [1949], then Norman [1963] and Goldberg [1981]), believe that it is possible to exhibit five categories, or core factors, that determine one's personality. The acronym OCEAN (for openness, conscientiousness, extraversion, agreeableness, and neuroticism) is often used to refer to this model. We chose to use this precise model as it is nowadays the most popular in psychology : while the five dimensions don't capture the peculiarity of everyone's personality, it is the theoretical framework most recognized by researchers and practitioners in this field.

Many linguistic-oriented tools can be used to derive a person's personality traits, for instance the individual's linguistic markers (obtained using text analysis, psycholinguistic databases and lexicons for instance). Since one of the earliest studies in this particular field [Mairesse et al., 2007], researchers have introduced multiple linguistic features and have shown correlations between them and the Big Five. These features could therefore have a non-negligible impact on classification performances, but as we stated before, we will mainly focus machine learning methods and leave out the linguistic modeling as it does not fit into the spectrum of our study.

We are using data that was gathered in a study by Pennebaker and King [1999]. It consists of a total of 2,468 daily writing submissions from 34 psychology students (29 women and 5 men whose ages ranged from 18 to 67 with a mean of 26.4). The writing submissions were in the form of a course unrated assignment. For each assignment, students were expected to write a minimum of 20 minutes per day about a specific topic. The data was collected during a 2-week summer course between 1993 to 1996. Each student completed their daily writing for 10 consecutive days. Students' personality scores were assessed by answering the Big Five Inventory (BFI) [John et al., 1991]. The BFI is a 44-item self-report questionnaire that provides a score for each of the five personality traits. Each item consists of short phrases and is rated using a 5-point scale that ranges from 1 (disagree strongly) to 5 (agree strongly). An instance in the data source consists of an ID, the actual essay, and five classification labels of the Big Five personality traits.

Labels were originally in the form of either yes ('y') or no ('n') to indicate scoring high or low for a given trait. It is important to note that the classification labels have been applied according to answers to a rather short self-report questionnaire: there might be a non-negligible bias in the data due to both the relative simplicity of the BFI test compared to the complexity of psychological features, and the cognitive biases preventing users from providing a perfectly accurate assessment of their own characteristics.

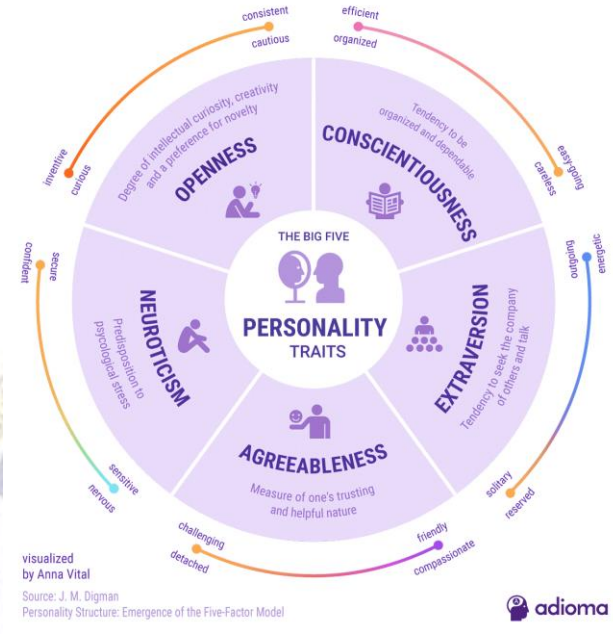
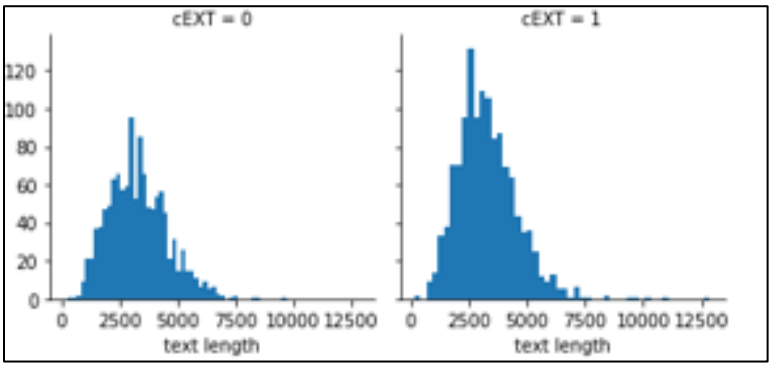


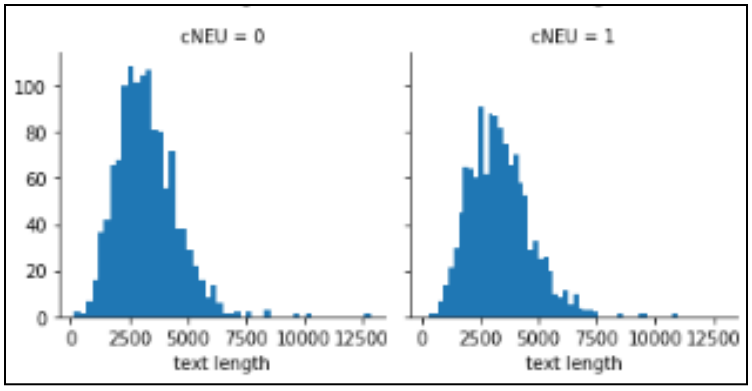
Fig 4.1.6

Histograms of Text length Distribution for each Label is as follows: -

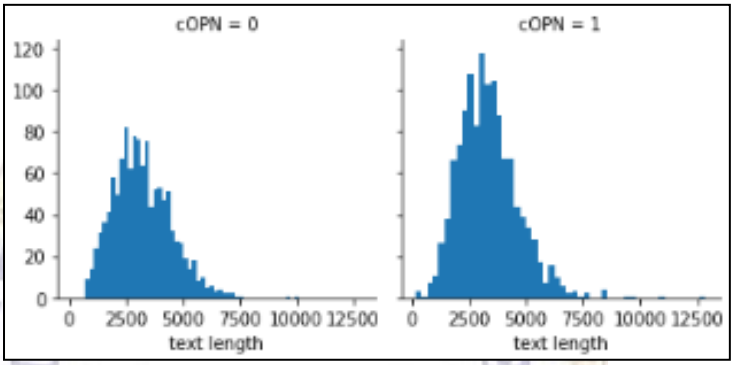
Histograms of Text length Distribution for the label EXTRAVERSION.



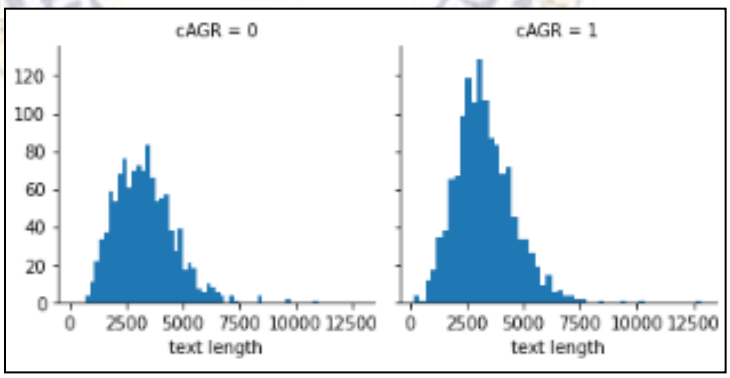
Histograms of Text length Distribution for the label NEUROTICISM.



Histograms of Text length Distribution for the label OPENESS.



Histograms of Text length Distribution for the label AGREEABLENESS.



Histograms of Text length Distribution for the label CONSCIENTIOUSNESS

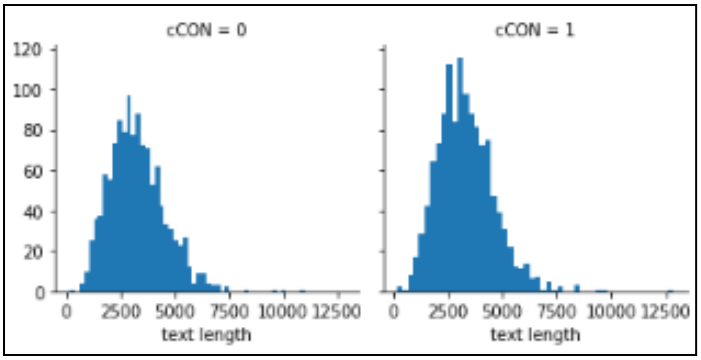
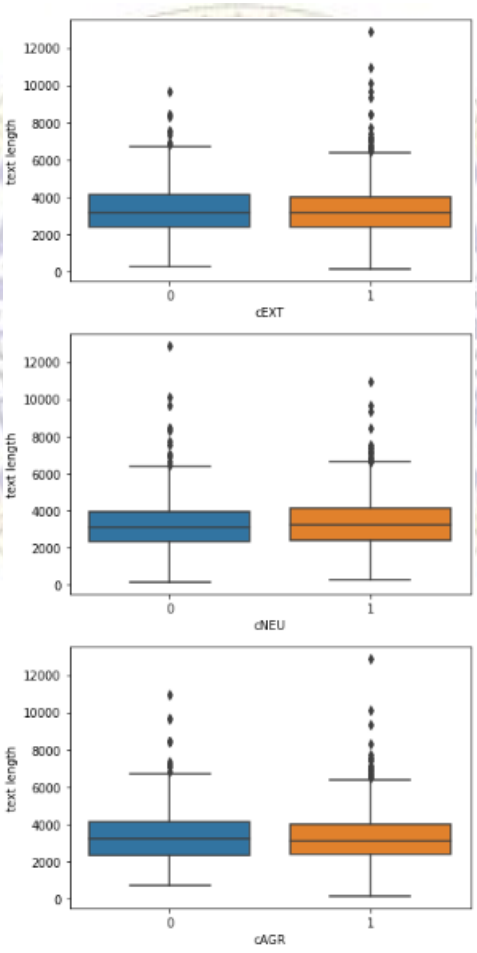


Fig 4.1.7

Boxplots of text length Distribution for each Label



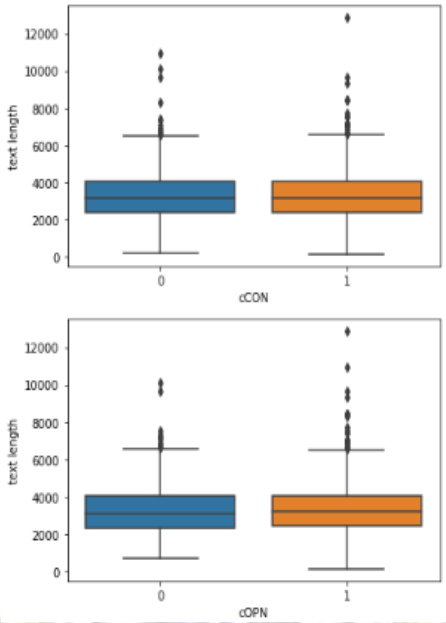


Fig 4.1.8

List of 100 most frequent words/counts

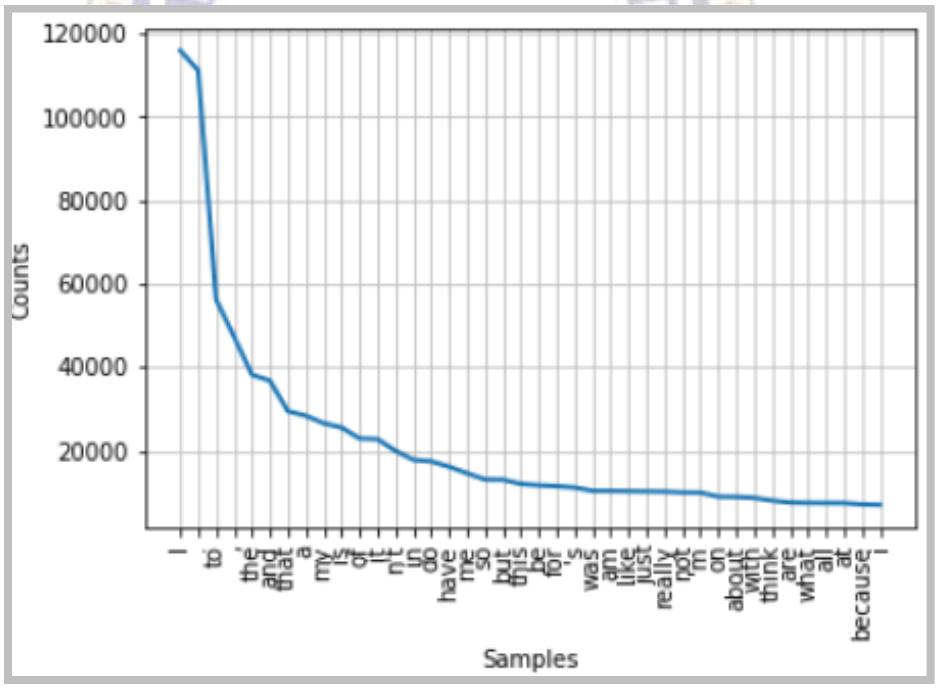


Fig 4.1.9

4.2.1 Model Design

Design a convolutional neural network with 4 convolution layers and 2 fully connected layers to predict 7 types of facial expressions.Use Adam as the optimizer, categorical crossentropy as the loss function, and accuracy as the evaluation metric. Wr compare the performance of the model with ResNet50 Model and deploy the efficient model.

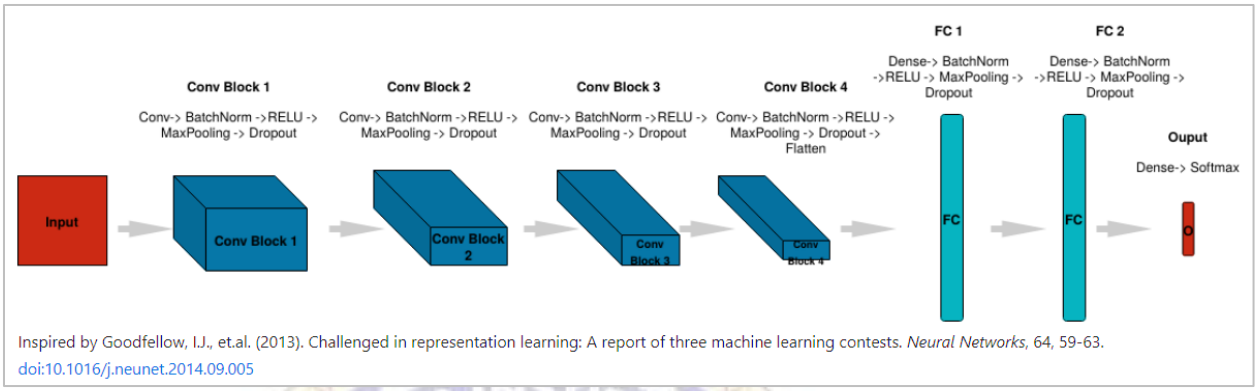


Fig – 4.2.1 – Inspiration of the model designed as Base Model.The model is trained from scratch.

An overview of the entire process for processing and analysis of audial input is as specified:

TimeDistributed CNNs

The main idea of a Time Distributed Convolutional Neural Network is to apply a rolling window (fixed size and time-step) all along the log-mel-spectrogram. Each of these windows will be the entry of a convolutional neural network, composed by four Local Feature Learning Blocks (LFLBs) and the output of each of these convolutional networks will be fed into a recurrent neural network composed by 2 cells LSTM (Long Short Term Memory) to learn the long-term contextual dependencies. Finally, a fully connected layer with softmax activation is used to predict the emotion detected in the voice**.**

TimeDistributed CNNs pipeline consists of the following process: -

- Voice recording
- Audio signal discretization
- Log-mel-spectrogram extraction
- Split spectrogram with a rolling window
- Make a prediction using our pre-trained model

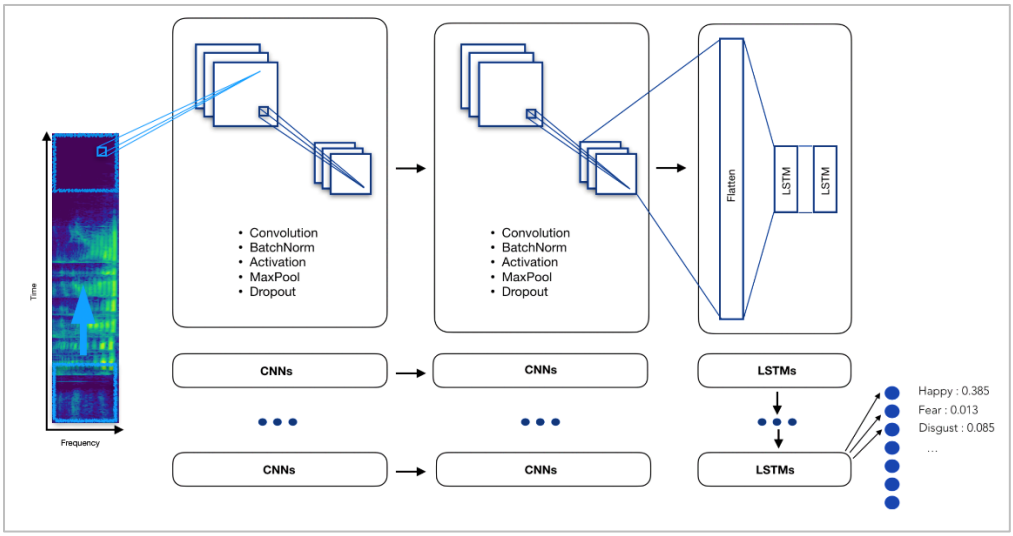


Fig – 4.2.2

An overview of the entire process for textual input is as specified:

The text-based personality recognition pipeline has the following structure :

- Text data retrieving
- Custom natural language preprocessing :
    - Tokenization of the document
    - Cleaning and standardization of formulations using regular expressions (for instance replacing "can't" by "cannot", "'ve" by "have")
    - Deletion of the punctuation
    - Lowercasing the tokens
    - Removal of predefined stopwords (such as 'a', 'an' etc.)
    - Application of part-of-speech tags on the remaining tokens
    - Lemmatization of tokens using part-of-speech tags for more accuracy.
    - Padding the sequences of tokens of each document to constrain the shape of the input vectors. The input size has been fixed to 300 : all tokens beyond this index are deleted. If the input vector has less than 300 tokens, zeros are added at the beginning of the vector in order to normalize the shape. The dimension of the padded sequence has been determine using the characteristics of our training data. The average number of words in each essay was 652 before any preprocessing. After the standardization of formulations, and the removal of punctuation characters and stopwords, the average number of words dropped to 168 with a standard deviation of 68. In order to make sure we incorporate in our classification the right number of words without discarding too much information, we set

the padding dimension to 300, which is roughly equal to the average length plus two times the standard deviation.

- 300-dimension Word2Vec trainable embedding
- Prediction using our pre-trained model

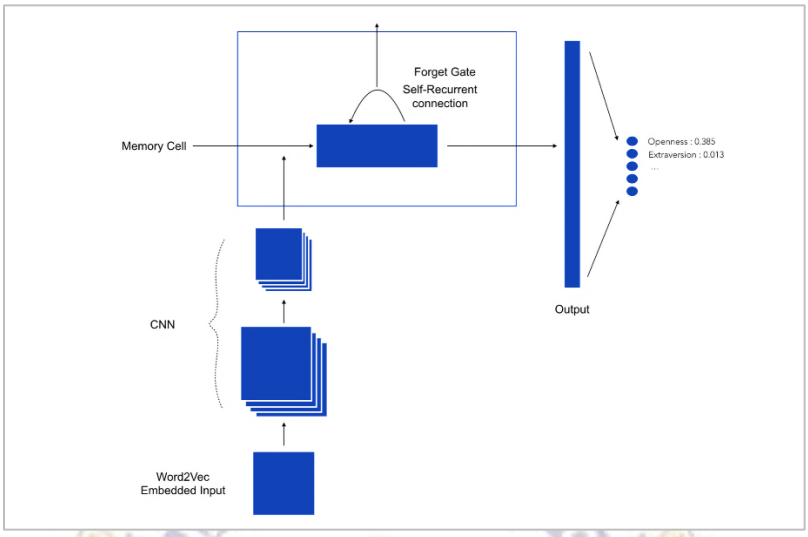


Fig – 4.2.3

Model

We have chosen a neural network architecture based on both one-dimensional convolutional neural networks and recurrent neural networks. The one-dimensional convolution layer plays a role comparable to feature extraction : it allows finding patterns in text data. The Long-Short Term Memory cell is then used in order to leverage on the sequential nature of natural language : unlike regular neural network where inputs are assumed to be independent of each other, these architectures progressively accumulate and capture information through the sequences. LSTMs have the property of selectively remembering patterns for long durations of time. Our final model first includes 3 consecutive blocks consisting of the following four layers : one-dimensional convolution layer - max pooling - spatial dropout - batch normalization. The numbers of convolution filters are respectively 128, 256 and 512 for each block, kernel size is 8, max pooling size is 2 and dropout rate is 0.3. Following the three blocks, we chose to stack 3 LSTM cells with 180 outputs each. Finally, a fully connected layer of 128 nodes is added before the last classification layer.

4.3 Training and Web Application

The User Interface for the application is as follows:-

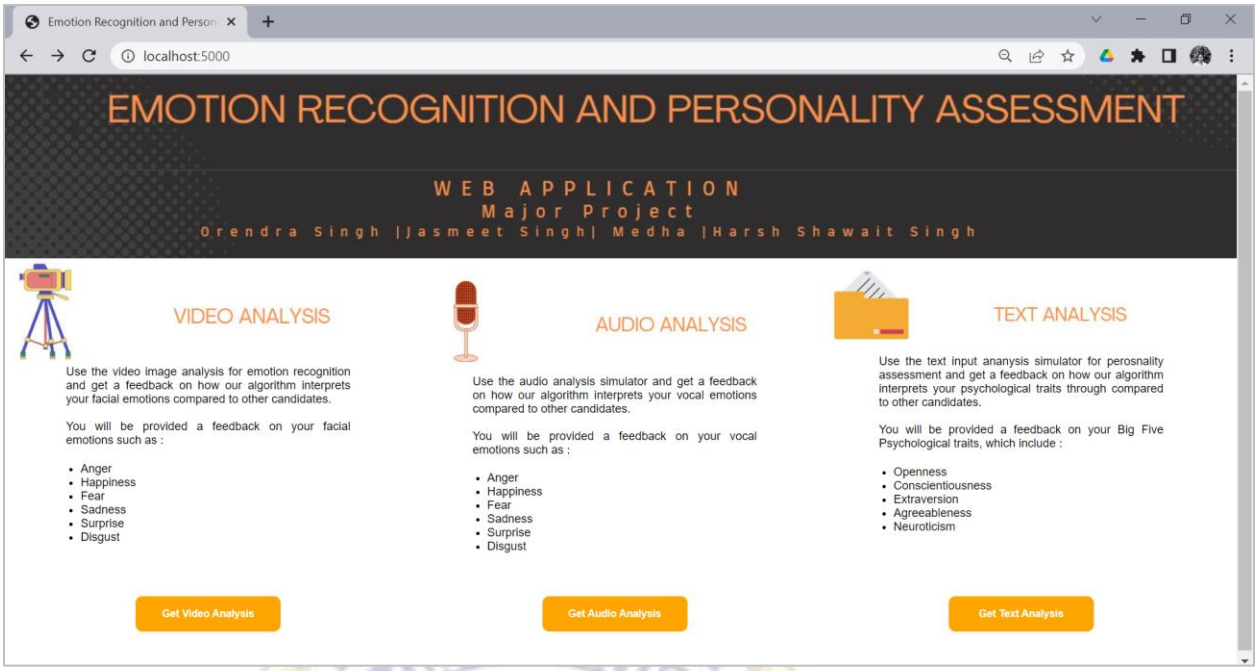
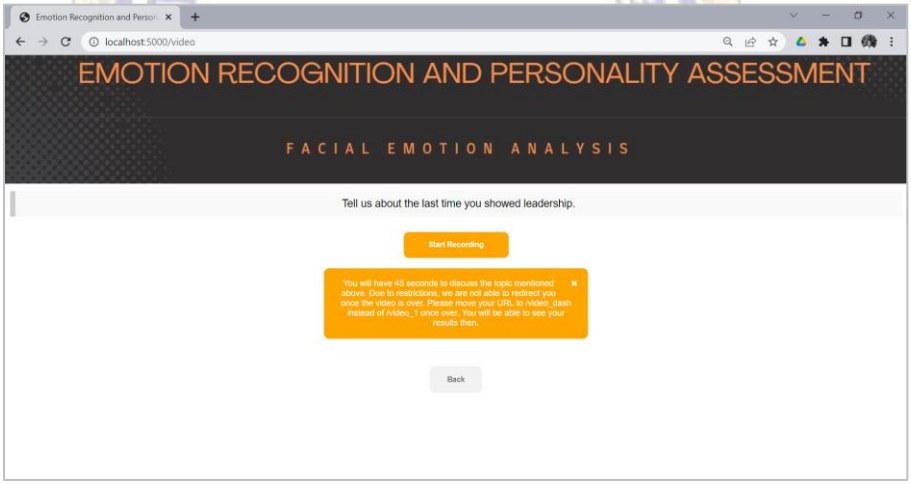


Fig 4.3.1

The output for Emotion Recognition through Real-Time Video Input is as follows: -



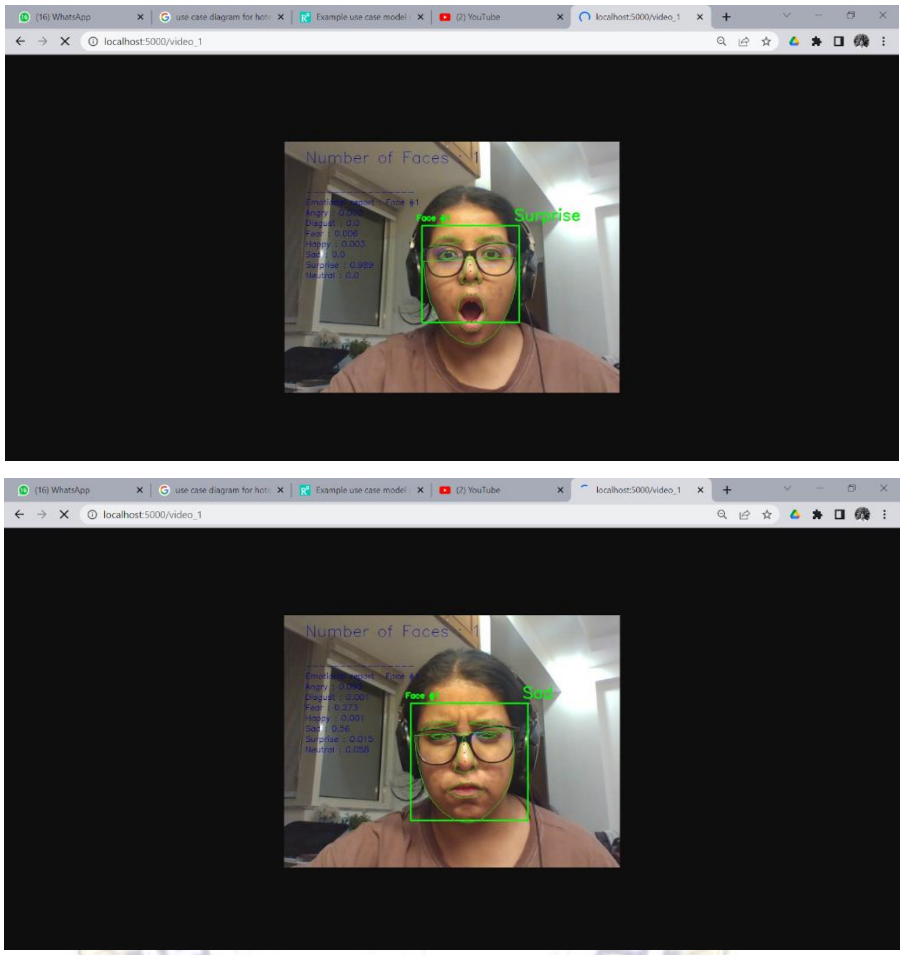

localhost:5000/video_1
Number of Faces : 1
Face #1
Surprise
localhost:5000/video_1
Number of Faces : 1
Face #1
Sad

Fig 4.3.2

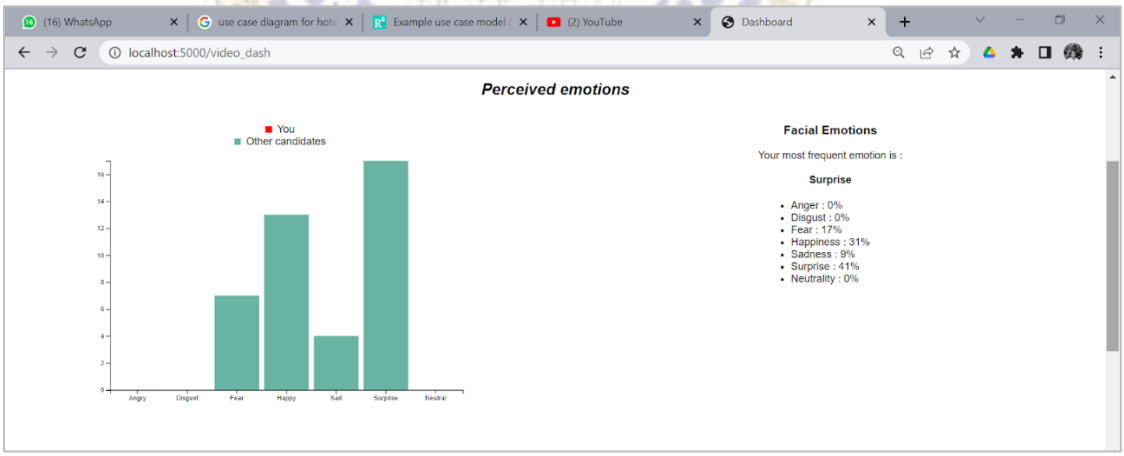

localhost:5000/video_dash
Perceived emotions
You
Other candidates
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
Facial Emotions
Your most frequent emotion is :
Surprise
Anger : 0%
Disgust : 0%
Fear : 17%
Happiness : 31%
Sadness : 9%
Surprise : 41%
Neutrality : 0%

Fig 4.3.4

The output for Emotion Recognition through Real-Time Audio Input is as follows: -

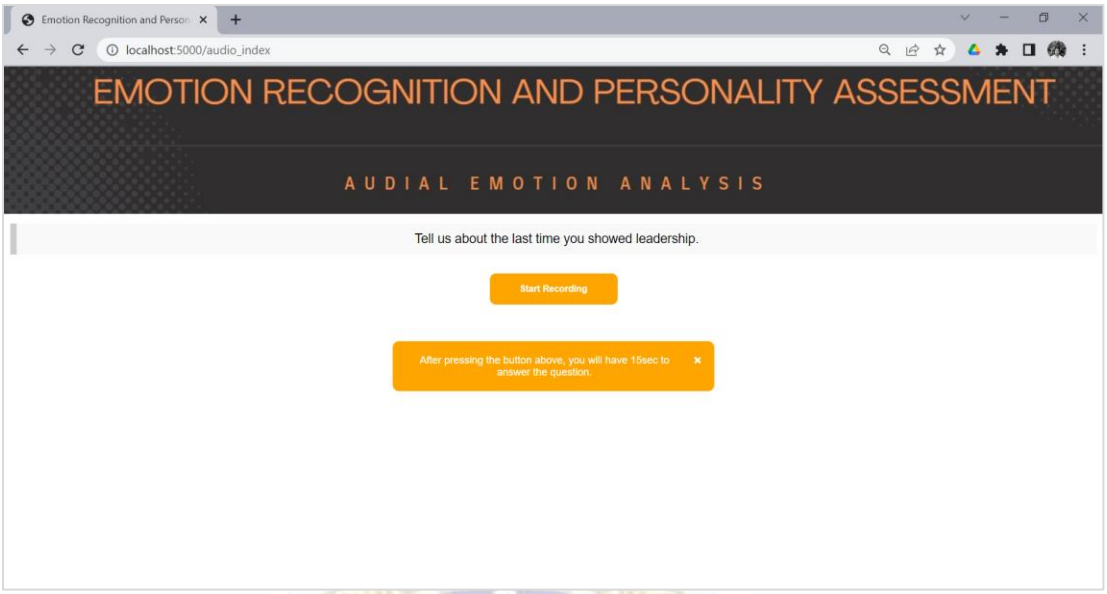


Fig 4.3.5

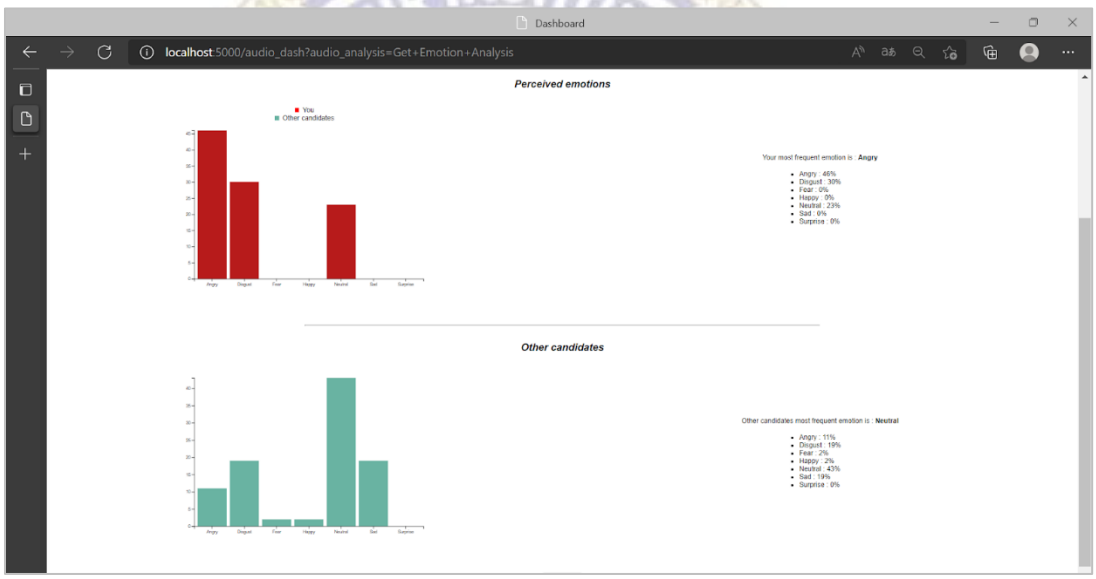


Fig 4.3.6

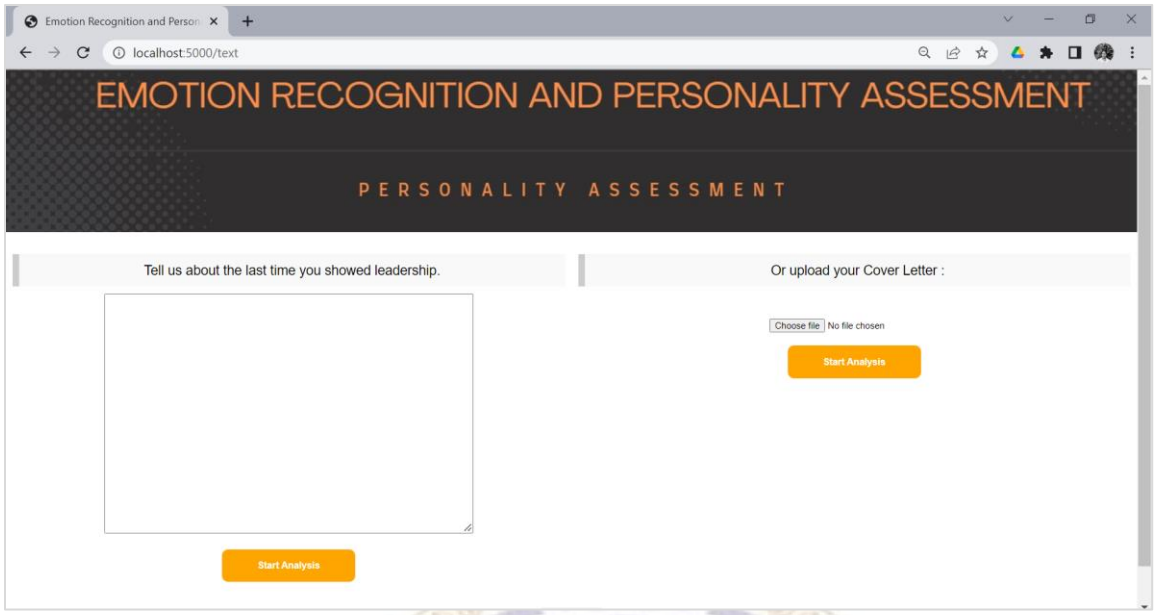


Fig 4.3.7

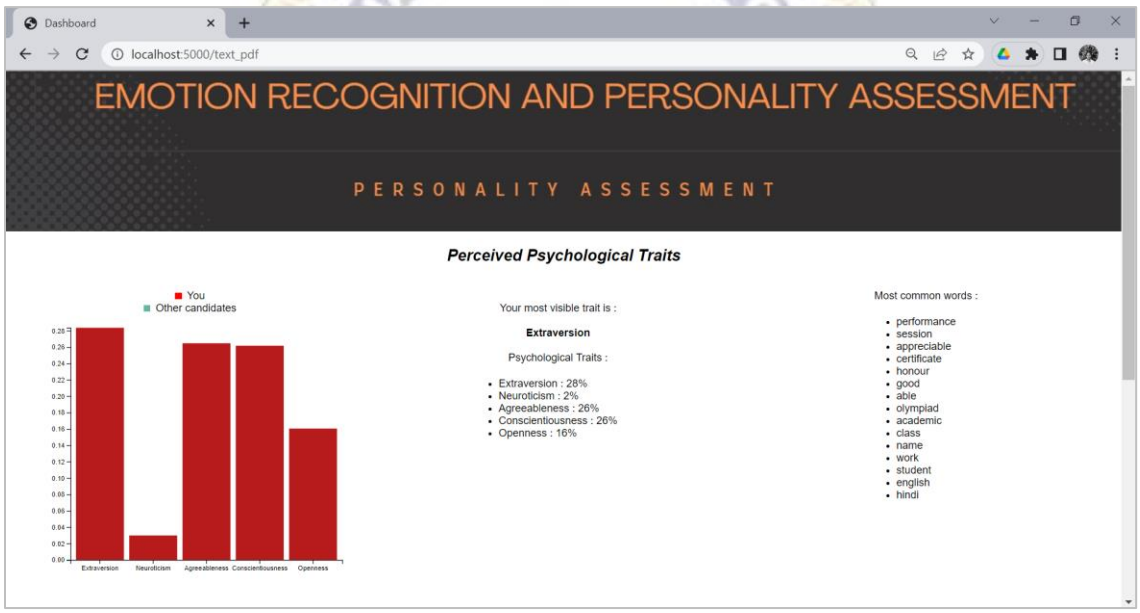


Fig 4.3.8

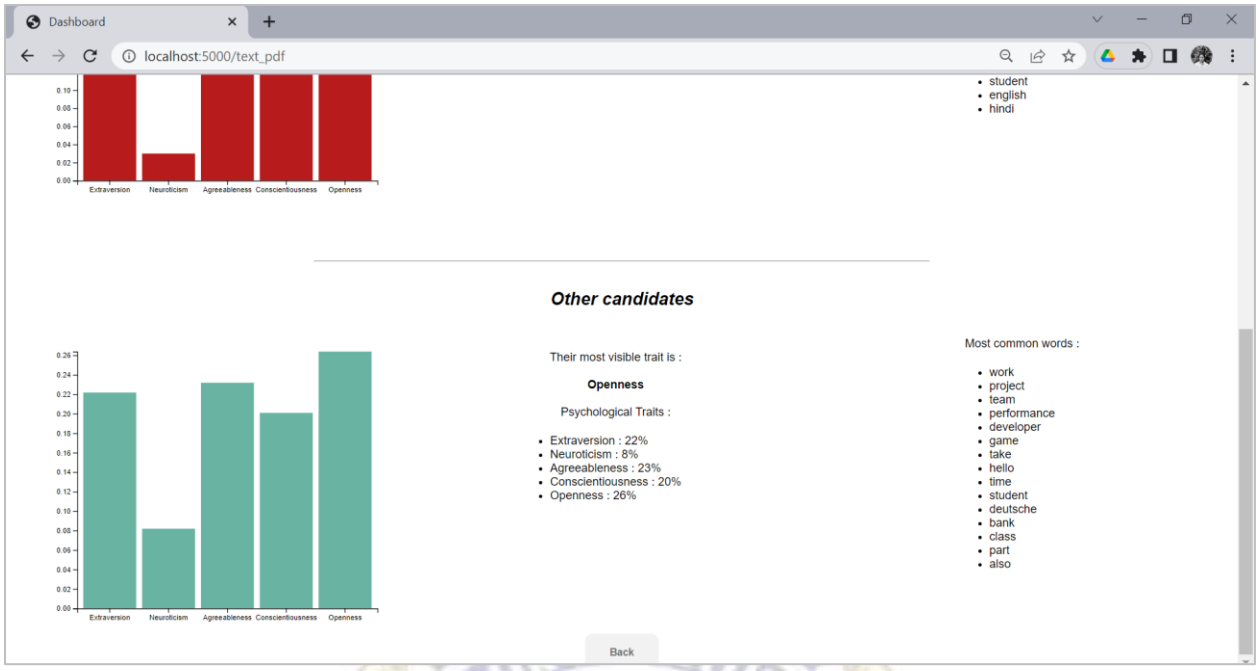


Fig 4.3.9

5. Implementation and Testing

5.1 Model Architecture

The design of the base model design is as follows :-

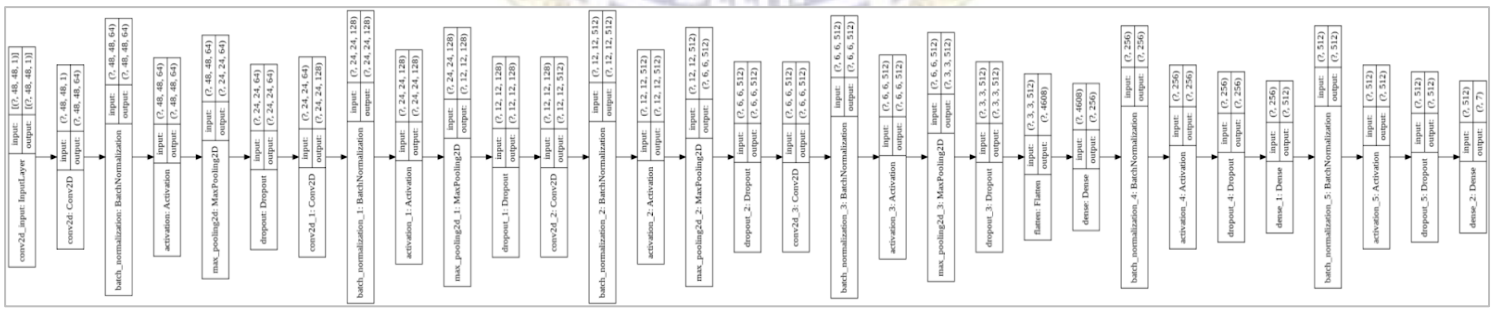


Fig 5.1.1

The design of the model using ResNet-50 as the pre-trained model is as:-

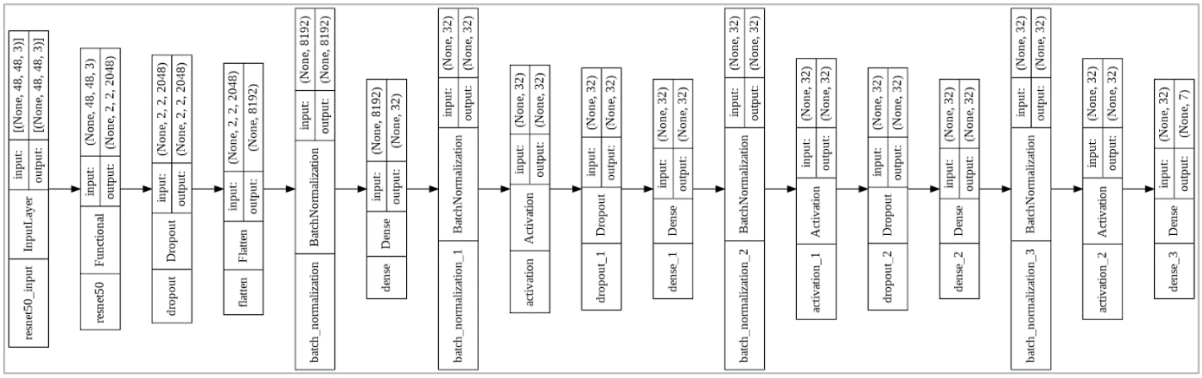


Fig 5.1.2

The design of Time Distributed CNN and LSTM for Speech Emotion Recognition is as follows: -

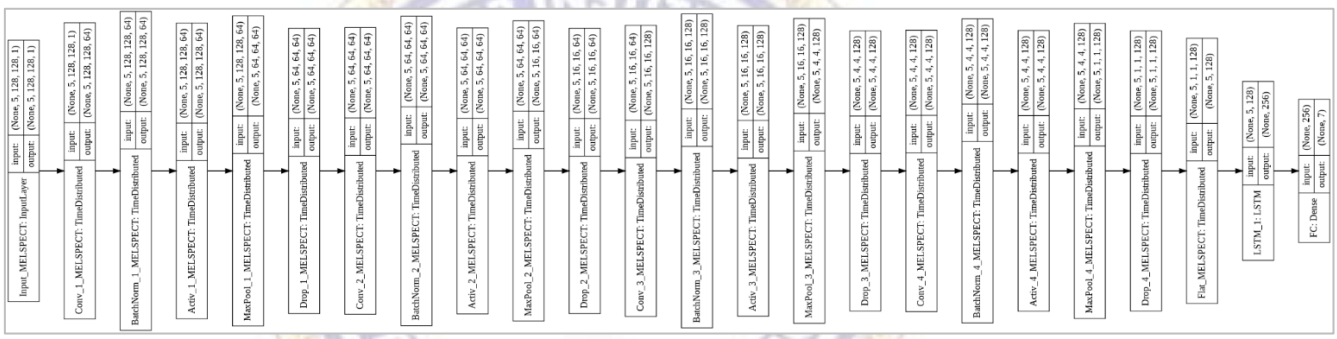


Fig 5.1.3

5.2 Analysis of Model Design

5.2.1 Analysis of Base Model for FER

The model is trained for 10 EPOCHS only.

The plot for ***accuracy*** and ***loss*** after 2 EPOCHS :-

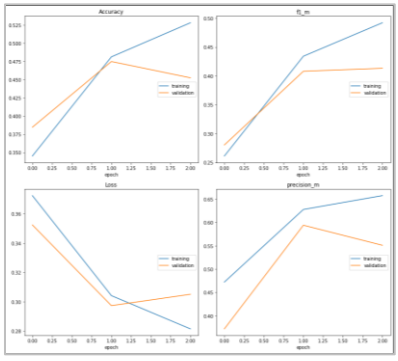


Fig 5.2.1.a

The plot for ***accuracy*** and ***loss*** after 4 EPOCHS :-

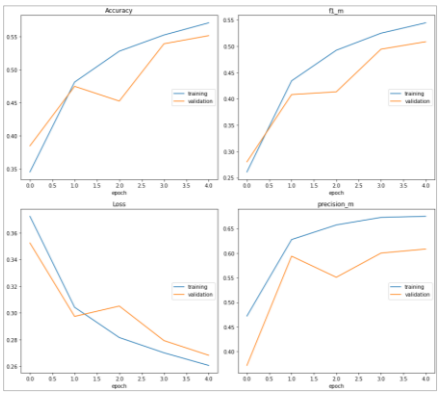


Fig 5.2.1.b

The plot for ***accuracy*** and ***loss*** after 6 EPOCHS :-

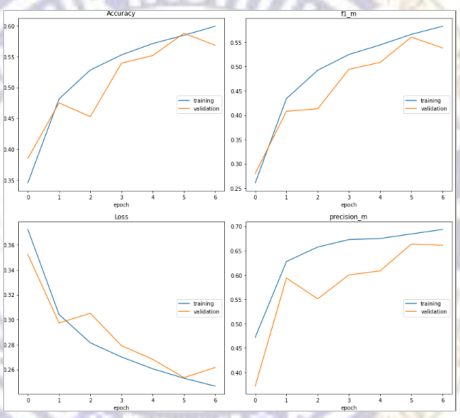


Fig 5.2.1.c

The plot for ***accuracy*** and ***loss*** after 10 EPOCHS :-

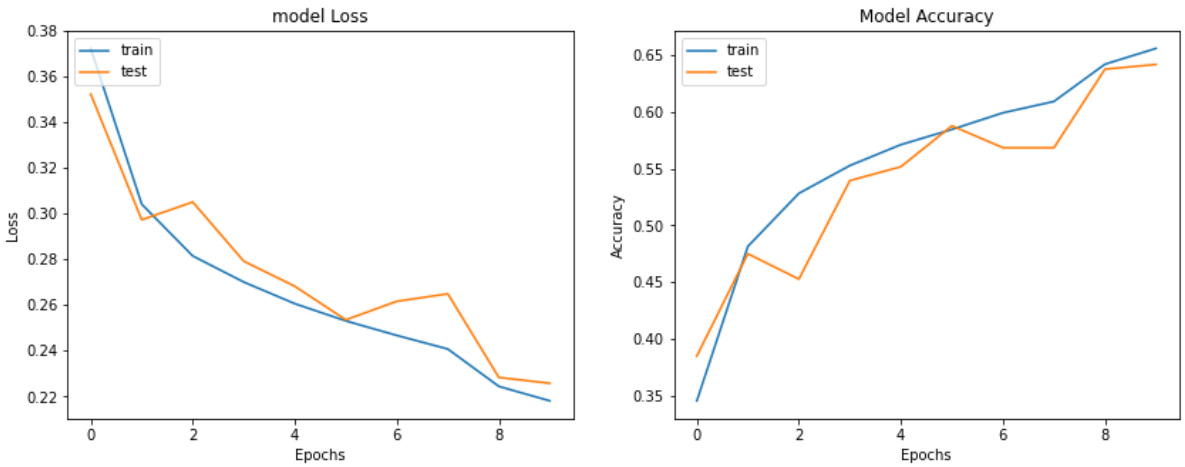


Fig 5.2.1.d

The output of the Web-App is as :-

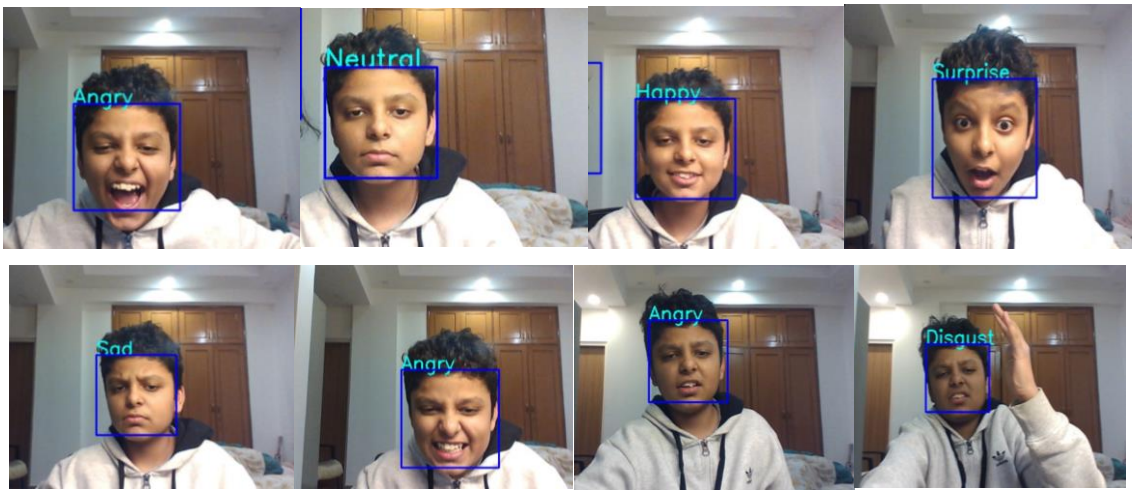


Fig 5.2.1.e

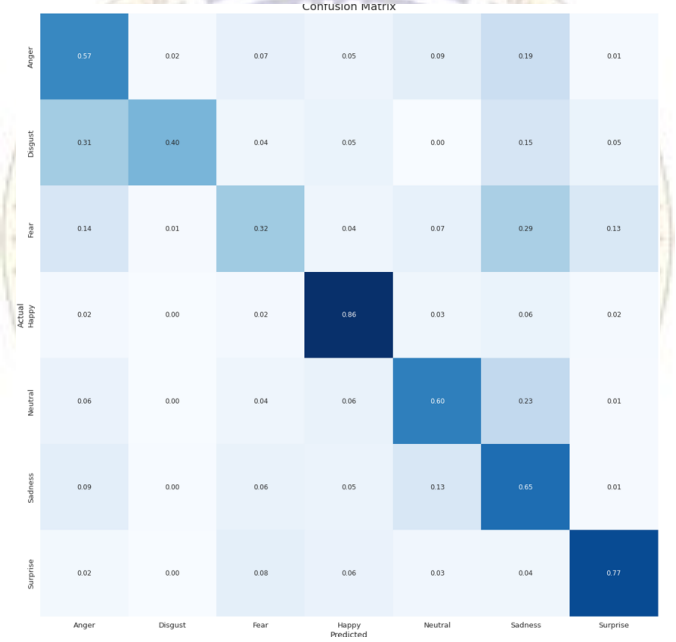


Fig 5.2.1.f

Fig 5.2.1

Confusion Matrix on Test Set with the the actual count of images, which have been classified as a certain label along with their original label which is either or incorrect depending upon the cell position being referred

Result:- The TOP-5 Accuracy is as specified :-

Top 0 Accuracy: 0.641404290888827

Top 1 Accuracy: 0.82418500975202

Top 2 Accuracy: 0.9080523822791864

Top 3 Accuracy: 0.9651713569239343

Top 4 Accuracy: 0.9891334633602675

Top 5 Accuracy: 0.9972137085539148

Top 6 Accuracy: 1

Analysis of ResNet 50

RESNET-

The plot for ***accuracy*** and ***loss*** after 10 EPOCHS :-

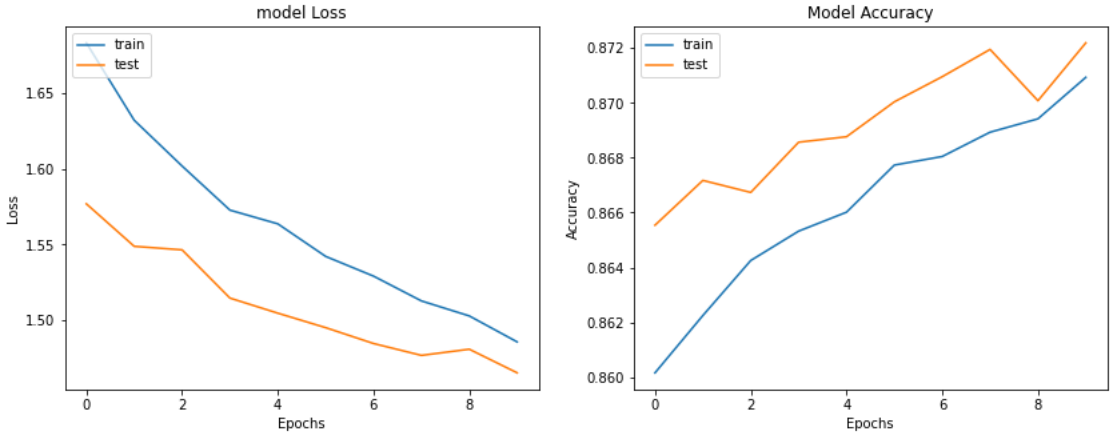


Fig 5.2.2.a

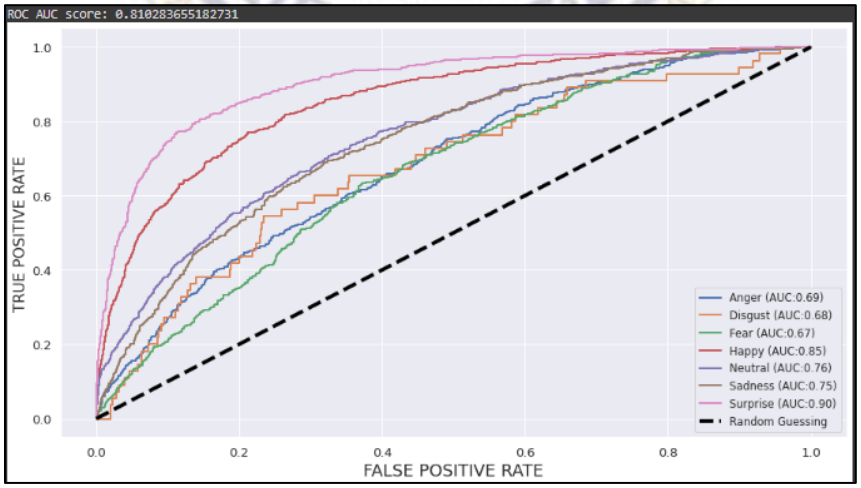


Fig 5.2.2.b

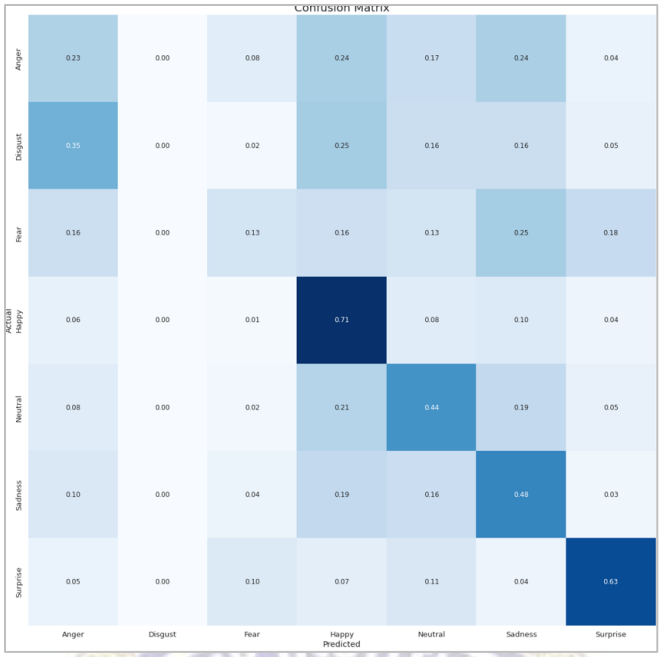


Fig 5.2.2.c

Fig 5.2.2

Confusion Matrix on Test Set with the the actual count of images, which have been classified as a certain label along with their original label which is either or incorrect depending upon the cell position being referred.

Result:- The TOP-5 Accuracy is as specified :-

Top 0 Accuracy: 0.44692114795207577

Top 1 Accuracy: 0.641961549178044

Top 2 Accuracy: 0.7762607968793536

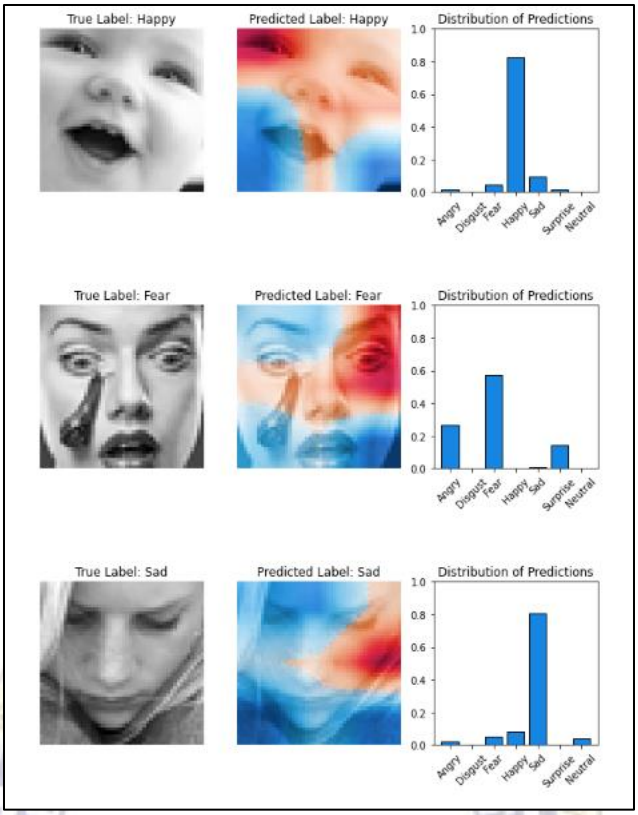
Top 3 Accuracy: 0.8790749512398996

Top 4 Accuracy: 0.9509612705488995

Top 5 Accuracy: 0.983560880468097

Top 6 Accuracy: 1.0

The distribution of predictions is as follows:-


True Label: Happy
Predicted Label: Happy
Distribution of Predictions
True Label: Fear
Predicted Label: Fear
Distribution of Predictions
True Label: Sad
Predicted Label: Sad
Distribution of Predictions
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral

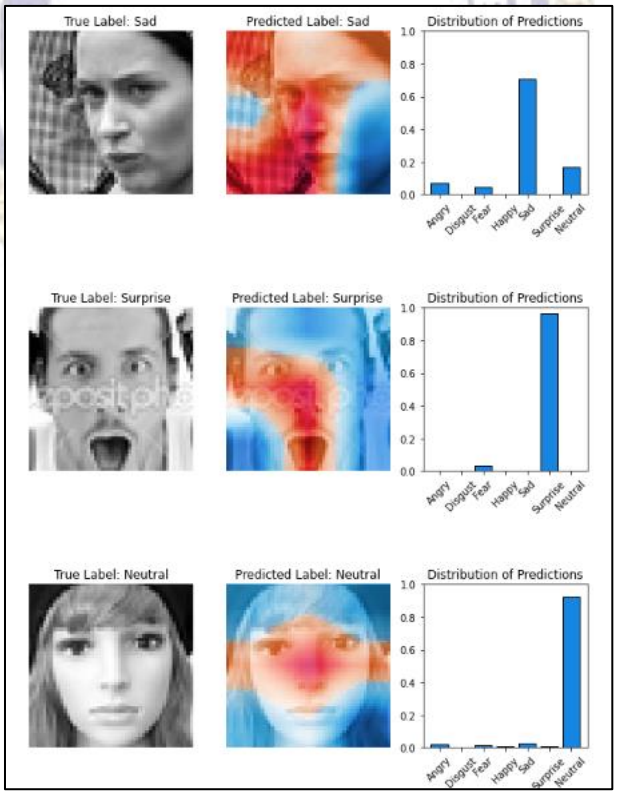
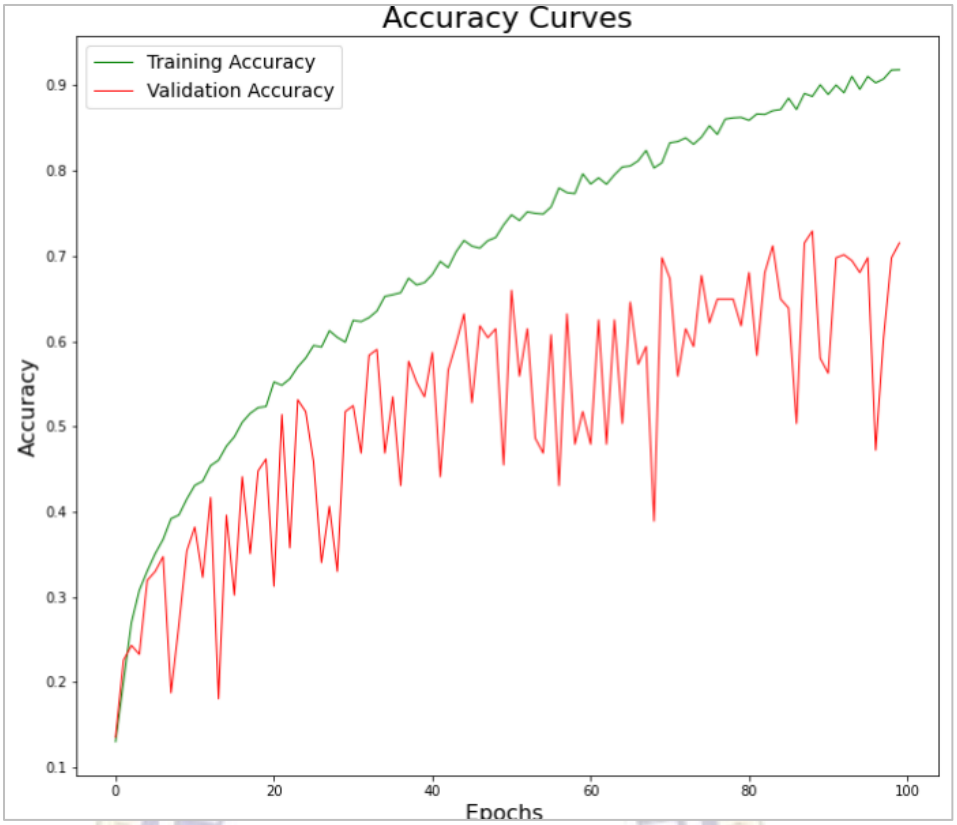

True Label: Sad
Predicted Label: Sad
Distribution of Predictions
True Label: Surprise
Predicted Label: Surprise
Distribution of Predictions
True Label: Neutral
Predicted Label: Neutral
Distribution of Predictions
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral

Fig 5.2.3

The GRAD-CAM Visulaizations are as follows :-

Fig 5.2.4


Accuracy Curves
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100

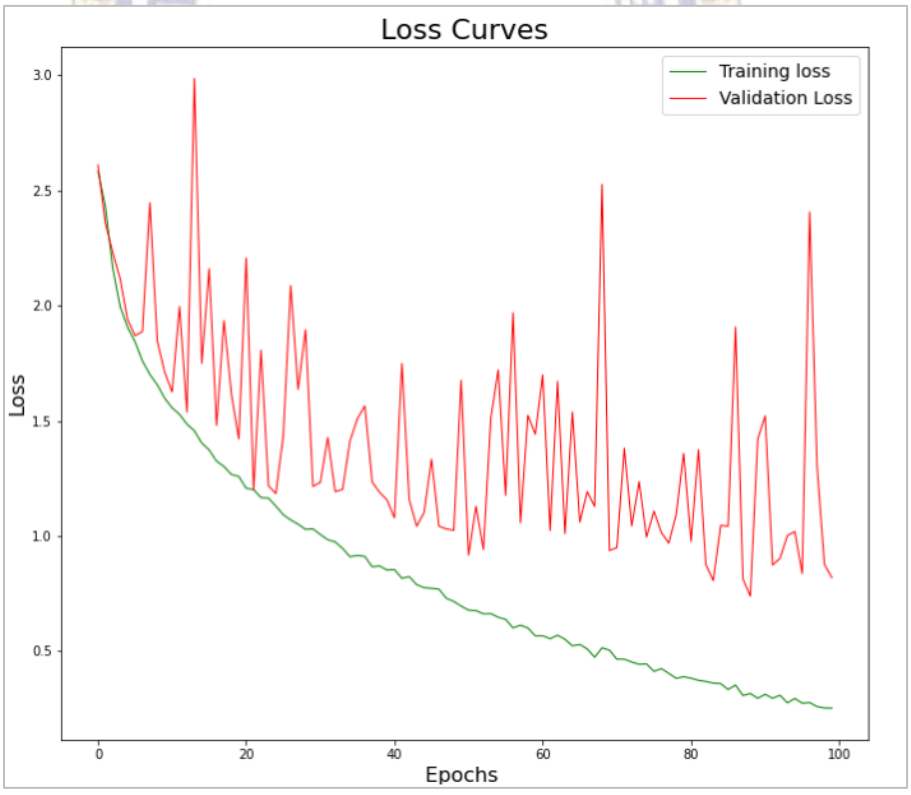

Loss Curves
Training loss
Validation Loss
Loss
Epochs
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

Fig 5.2.5

6. Conclusion, Summary and Future Scope

This project was a combination of CNN model classification problem where we have to identify and recognize appropriate human emotion & Computer Vision problem (to automate the process of identifying the features of human face). The final accuracy is much higher than 50% baseline However, it could be increased by larger number of train images or through model hyper-parameters tuning. ResNet-50 at present , has the best accuracy on the Training Set, for the present number of iterations. Further , by adding an appropriate model design on top of the pre-trained model, it is possible to get a better accuracy. Accuracy can also be modified by a sufficiently large dataset and appropriate split of the dataset in TRAIN, VALIDATION and SPLIT Datasets. Also, by appropriately handling class imbalance, accuracy of the pre-trained model can be increased multi-folds. This project consisted of advanced and recent challenges presented as a problem statement to be solved by efficient utilization of CNN model as a classification problem where we have to identify and recognize human emotion and get a measure of distinct psychological traits playing a key-factor for personality analysis using Multimodal Data. CNNs and RNNs provide a great scope of improvement in design for better results. Additional datasets can be used to train the model for better analysis of facial, vocal and textual input. Other important parameters which can be improved to enhance accuracy of the result are as Data Augmentation, Hyper Parameter Tuning, Early Stopping and Activation Functions. The web application user interface can be removed by designing an ensemble model to extract three distinct features from only one stream of input rather than taking a distinct input for each mode of information extraction. Additional features can be introduced at frontend to enhance the overall user experience

APPENDIX

(CODE)

MODEL-1

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#import utils
import os
%matplotlib inline

import keras

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Input, Dropout,Flatten, Conv2D
from tensorflow.keras.layers import BatchNormalization, Activation,
MaxPooling2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import plot_model

from IPython.display import SVG, Image
#from livelossplot import PlotLossesKeras
# from livelossplot import PlotLossesTensorFlowKeras
import tensorflow as tf
print("Tensorflow version:", tf.__version__)
```

```
img_size = 48
batch_size = 64

# Data generator to augment data for training
datagen_train = ImageDataGenerator(horizontal_flip=True)
train_generator = datagen_train.flow_from_directory("/content/train/train",

target_size=(img_size,img_size),
                                                    color_mode='grayscale',
                                                   batch_size=batch_size,
                                                   class_mode='categorical',
                                                   shuffle=True)

# Data generator to augment data for validation
datagen_validation = ImageDataGenerator(horizontal_flip=True)
validation_generator = datagen_train.flow_from_directory("/content/test/test-
private",

target_size=(img_size,img_size),
                                                    color_mode='grayscale',
                                                   batch_size=batch_size,
```

```
                                                    class_mode='categorical',
                                                    shuffle=False)


model = Sequential()

# Conv Block 1
model.add(Conv2D(64, (3,3), padding='same', input_shape=(48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Conv Block 2
model.add(Conv2D(128,(5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Conv Block 3
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Conv Block 3
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())

# Fully connected Block 1
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected Block 2
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(7, activation='softmax'))

opt = Adam(lr=0.0005)
model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
model.summary()
```

```
from livelossplot import PlotLossesKeras
#from livelossplot import PlotLossesTensorFlowKeras
from keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['acc',f1_m,precision_m, recall_m])

epochs = 10
steps_per_epoch= train_generator.n//train_generator.batch_size
validation_steps = validation_generator.n//validation_generator.batch_size

checkpoint = ModelCheckpoint("model_weights.h5",monitor='val_accuracy',
                             save_weights_only=True, mode='max',verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss' , factor=0.1, patience=2,
min_lr=0.00001,model='auto')

callbacks = [PlotLossesKeras(), checkpoint, reduce_lr]

history = model.fit(
        x= train_generator,
        steps_per_epoch=steps_per_epoch,
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=validation_steps,
        callbacks=callbacks
)
```

```
plt.figure(figsize=(14,5))
plt.subplot(1,2,2)
plt.plot(history.history['acc'])
```

```
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['train', 'test'], loc='upper left')

plt.subplot(1,2,1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
y_pred = model.predict(train_generator)
y_pred = np.argmax(y_pred, axis=1)
class_labels = train_generator.class_indices
class_labels = {v:k for k,v in class_labels.items()}
Class_labels  = ['Anger', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sadness',
"Surprise"]
from sklearn.metrics import classification_report, confusion_matrix
cm_train = confusion_matrix(train_generator.classes, y_pred)
print('Confusion Matrix')
print(cm_train)
print('Classification Report')
target_names = list(CLASS_LABELS)
print(classification_report(train_generator.classes, y_pred,
target_names=target_names))

plt.figure(figsize=(8,8))
plt.xlabel('Predictions', fontsize=10)
plt.ylabel('Actuals', fontsize=10)
plt.title('Confusion Matrix', fontsize=10)
plt.grid(False)
plt.imshow(cm_train, interpolation='nearest')
plt.colorbar()
tick_mark = np.arange(len(target_names))
_ = plt.xticks(tick_mark, target_names, rotation=90)
_ = plt.yticks(tick_mark, target_names)


y_pred = model.predict(validation_generator)
y_pred = np.argmax(y_pred, axis=1)
class_labels = validation_generator.class_indices
class_labels = {v:k for k,v in class_labels.items()}
Class_labels  = ['Anger', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sadness',
"Surprise"]
#from sklearn.metrics import classification_report, confusion_matrix
cm_test = confusion_matrix(validation_generator.classes, y_pred)
print('Confusion Matrix')
print(cm_test)
```

```
print('Classification Report')
target_names = list(CLASS_LABELS)
print(classification_report(validation_generator.classes, y_pred,
target_names=target_names))

plt.figure(figsize=(8,8))
plt.xlabel('Predictions', fontsize=10)
plt.ylabel('Actuals', fontsize=10)
plt.title('Confusion Matrix', fontsize=10)
plt.grid(False)
plt.imshow(cm_test, interpolation='nearest')
plt.colorbar()
tick_mark = np.arange(len(target_names))
_ = plt.xticks(tick_mark, target_names, rotation=90)
_ = plt.yticks(tick_mark, target_names)
```

MODEL -2

```
from keras.preprocessing import image
img = image.load_img("/content/test/test-
private/happy/PrivateTest_10077120.jpg",target_size=(48,48))
img = np.array(img)
plt.imshow(img)
print(img.shape)

img = np.expand_dims(img, axis=0)
from keras.models import load_model
print(img.shape)
```

```
(48, 48, 3)
(1, 48, 48, 3)
```

In [9]:

```
base_model =
tf.keras.applications.ResNet50(input_shape=(48,48,3),include_top=False,weight
s="imagenet")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applica
tions/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 3s 0us/step
94781440/94765736 [==============================] - 3s 0us/step
```

In [10]:

```
# Freezing Layers

for layer in base_model.layers[:-4]:
    layer.trainable=False
```

In [11]:

```
# Building Model

model=Sequential()
model.add(base_model)
model.add(Dropout(0.5))
```

```
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(7,activation='softmax'))
```

REFERENCES

1. https://www.geeksforgeeks.org/
2. https://arxiv.org/abs/2105.03588
3. https://www.kaggle.com/yasserhessein/emotion-recognition-with-resnet50
4. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2