Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Human Face Detector and Analyzer

By:

Shehab Ahmed Bassiouni [CS Department]
Omar Essam Abdelhadi [CS Department]
Mohammed Mas'ad Saad [CS Department]
Mohammed Adel Abdullhamid [CS Department]
Ibrahim Abdelghani Mansour [CS Department]

Under Supervision of:

[Supervisor 1]
[Dr. **Nermine Naguib**],
Computer Science Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

[Supervisor 2]
[Dr. **Assma Bahi**],
Computer Science Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

Acknowledgement

First and foremost, we express our deepest gratitude to God Almighty for granting us the strength, guidance, and perseverance throughout the journey of completing our graduation project.

We would like to extend our heartfelt appreciation to Dr. Nermin and Dr. Asmaa Qasem for their invaluable guidance, support, and mentorship during the entire duration of our project. Their expertise, encouragement, and commitment to our success were instrumental in shaping our ideas, refining our methodology, and enhancing the overall quality of our work.

Their insightful feedback, constructive criticism, and tireless dedication to our project significantly contributed to its successful completion. We are truly grateful for their patience, as they generously shared their knowledge, challenged our thinking, and pushed us to excel beyond our limits.

We are indebted to their unwavering belief in our abilities and their commitment to fostering our intellectual growth. Their mentorship went beyond the scope of the project, leaving a profound impact on our personal and professional development.

We also extend our gratitude to the entire faculty and staff of the institution for providing us with a conducive environment and resources that facilitated our project's execution. Their continuous support and encouragement played a vital role in our journey.

Lastly, we would like to express our appreciation to our families, friends, and loved ones for their unwavering support, understanding, and encouragement during this challenging endeavor. Their belief in us fueled our determination and inspired us to overcome obstacles.

Completing this graduation project would not have been possible without the collective efforts and support of all those mentioned above. We are truly humbled and grateful for their contributions, and we look forward to carrying the knowledge and experience gained from this project into our future endeavors.

Thank you all once again for your immense support and assistance.

Abstract

The “Human Face Detector and Analyzer” is an API application to detect human face in an image and analyze it to predict Age, Gender, Emotions, and Ethnicity using implemented Deep learning models.

Some of the possible usages of this API application are:

1- Provides API Endpoint for developers to easily integrate the application features in their project.
2- Can be used in Entertainment and Marketing by analyzing the features of most-visiting customers.
3- Can be used in healthcare by monitoring Patients mental health through their emotional state.
4- Can be used in security by integrating it to Surveillance systems to track individuals with specific features.
5- Can be used in investigations and police-related tasks by providing estimations and predictions about individual specific features by their facial image.

The developed models achieved responsible results in detecting human face and predicting their Age, Gender, Emotions, and Ethnicity by utilizing the power of Deep Convolutional Neural Networks and Deep learning techniques and optimizations.

The Models Achieved the following results:

Model	method	Dataset	Loss	Accuracy
Age Prediction	CNN	Facial Age and UTK-Face	0.75	71%
Gender Prediction	CNN	UTK-Face and B3FD	0.8463	81%
Emotions Prediction	CNN	FER-2013	0.9411	67%
Ethnicity Prediction	Transfer learning and CNN	Ethnicity Aware and Arab Celebrity Faces	0.3744	86%
Face Detection	State-of-Art pretrained Model	———	——	——

Contents

Acknowledgement i

Abstract ii

List of Figures iv

List of Abbreviations vi

1- Introduction 1

1.1 Motivation 1

1.2 Problem Definition 1

1.3 Objective 3

1.4 Time Plan 4

1.5 Document Organization 6

2- Background 8

3- Analysis and Design 32

3.1 System Overview 32

3.1.1 System Architecture 32

3.1.2 System Users 33

3.2 System Analysis & Design 34

3.2.1 Use Case Diagram 34

3.2.2 Class Diagram 35

3.2.3 Sequence Diagram 35

3.2.4 Database Diagram 37

4- Implementation and Testing 38

7- User Manual 81

8- Conclusion and Future Work 83

6.1 Conclusion 83

6.2 Future Work 84

References 85

List of Figures

Figure 1: An approach to analyze human face based on CNN by Nippon and Juel. 8
Figure 2: CNN architecture proposed by Nippon and Juel. 8
Figure 3: types of AI Apps 10
Figure 4: Convolution Neural Network in Action. 12
Figure 5:Haar Cascade Algorithm Sliding Window. 14
Figure 6: The haar cascade algorithm in action. 14
Figure 7: The Haar Cascade Algorithm in action 2. 15
Figure 8: Data Augmentation in CNN. 17
Figure 9: Different Optimizers effect in Training Process. 18
Figure 10: Balanced and Imbalanced Classes. 20
Figure 11: Heat Maps in CNN. 23
Figure 12: Feature Maps in CNN 23
Figure 13: Convergence Plots in CNN. 24
Figure 14: Confusion Matrix in CNN. 25
Figure 15: Datasets in CNN. 26
Figure 16: VGG-16 Architecture. 28
Figure 17: 3-Tiers System Architecture. 32
Figure 18: Use Case Diagram. 34
Figure 19: Class Diagram. 35
Figure 20: User Sequence Diagram. 36
Figure 21: Developer Sequence Diagram. 37
Figure 22: import csv2 and read image from disk using it. 38
Figure 23: initializing Haar cascade Algorithm and apply it on image. 39
Figure 24: Cut the face from image using the given coordinates from Haar cascade algorithm. . 39
Figure 25: Displaying image using CV2. 39
Figure 26 Samples From "Facial Age" and "UTK Face" Datasets. 42
Figure 27 Distribution of Age Dataset after Augmentation 42
Figure 28 Age Model CNN Architecture 43
Figure 29: Split the dataset in the age model. 44
Figure 30: creating tensors from age dataset images and labels. 44
Figure 31:Apply preprocessing on tensors and specify the batch size to the data loader in age model 44
Figure 32: Age Model Preprocessing Function. 45
Figure 33: Age Model Architecture Sequential Implementation. 45
Figure 34: initialize the compile method and callbacks for age model. 46
Figure 35: Fitting the Age Model. 46
Figure 36 Age Model Convergence Plot 47
Figure 37 Age Model Confusion Matrix 47
Figure 38:Age Model Feature Map 48
Figure 39 Age Model Heat Map 48
Figure 40: Gender Model Train Dataset Class Distribution. 50
Figure 41: Gender Model Test and validate Dataset Class Distribution. 50
Figure 42: Samples of Gender Model Datasets. 51
Figure 43: Gender Model CNN Architecture. 52

Figure 44: Image Data Generator for Gender Model. 53
Figure 45:Gender Model Architecture in the Sequential Method. 53
Figure 46: Gender Model Compile and Callback Functions. 54
Figure 47: Fitting the Gender Model. 54
Figure 48: Gender Model Convergence Plot. 55
Figure 49: Gender Model Heat Map. 55
Figure 50: Gender Model Feature Map. 56
Figure 51:Gender Model Confusion Matrix. 57
Figure 52: Ethnicity Model Classes Distribution. 59
Figure 53: Ethnicity Model Dataset Samples. 59
Figure 54: Ethnicity Model CNN Architecture. 60
Figure 55: Ethnicity Model Image Data Generator. 61
Figure 56: Ethnicity Model Class weight Balancing. 62
Figure 57: Fine Tuning VGG16 in Ethnicity Model. 62
Figure 58: Compile and Call backs Functions in Ethnicity Model. 62
Figure 59: Fitting Ethnicity Model. 63
Figure 60: Ethnicity Model Convergence Plot. 63
Figure 61: Ethnicity Model Feature Map. 64
Figure 62:Ethnicity Model Heat Map. 65
Figure 63: Ethnicity Model Confusion Matrix. 66
Figure 64: Train Dataset Class Distribution in Emotion Model. 67
Figure 65: Test Dataset Class Distribution in Emotion Model. 68
Figure 66: Sample of Dataset used in Emotion Model. 68
Figure 67: Emotion Model CNN Architecture. 69
Figure 68: Image Data Generator in Emotion Model. 70
Figure 69: Compile and Call Backs Functions in Emotion Model. 71
Figure 70: Fitting Emotion Model. 71
Figure 71: Emotion Model Convergence Plot. 72
Figure 72: Emotion Model Confusion Matrix. 73
Figure 73: Emotion Model Confessions that are hard to classify. 73
Figure 74: API Running on port 8080. 74
Figure 75:Method Called When API Endpoint Receives a Request Message. 75
Figure 76: "Model" Class constructor Function when an object is created. 76
Figure 77: API predictions Methods. 77
Figure 78: Sending a request Message To the API Endpoint. 78
Figure 79: image sent by postman as request. 78
Figure 80: Reply from the API Endpoint to a Request 78
Figure 81: Web Application Port. 79
Figure 82: Web Application Upload Route. 79
Figure 83: displaying predictions received in html page in the Web Application. 80
Figure 84: Web Application Main Page. 82
Figure 85: Predictions From the API. 82

List of Abbreviations

Abbreviation

AI: Artificial Intelligence.

API: Application Programming Interface.

CNN: Convolutional Neural Network.

CV: Computer Vision.

FC: Fully Connected.

GAN: Generative Adversarial Networks

HTML: Hypertext Markup Language

H5: Hierarchical Data Format version 5.

IDE: Integrated development environment.

JASON: JavaScript Object Notation.

PR: Patter Recognition.

T&C: Terms and Conditions.

VSC: Visual Studio Code.

YOLO: You Can Only Look Once.

1- Introduction

1.1 Motivation

The field of computer vision has been witnessing remarkable advancements in recent years, opening new possibilities for applications that can revolutionize various industries. One area of particular interest is the analysis of human faces. The ability to detect and analyze human faces accurately has numerous practical implications in fields such as security, marketing, healthcare, and human-computer interaction. Considering these developments, our project titled "Human Face Detector and Analyzer" aims to leverage computer vision techniques to detect human faces and predict essential attributes such as age, gender, ethnicity, and emotions.

With the proliferation of digital devices equipped with cameras and the exponential growth of image and video data, there is an increasing demand for automated systems that can analyze human faces efficiently. Traditional methods of manually analyzing and categorizing faces are time-consuming, labor-intensive, and prone to human errors. By developing an automated human face detection and analysis system, we can streamline processes, save valuable time, and improve accuracy.

The recent advancements in computer vision, specifically in facial detection, recognition, and analysis, have served as the catalyst for our project. Deep learning techniques, such as convolutional neural networks, have demonstrated remarkable success in accurately detecting and analyzing human faces, outperforming traditional methods. These advancements have paved the way for the development of highly efficient and accurate face detection and analysis systems, motivating our idea for this project.

1.2 Problem Definition

The accurate detection and analysis of human faces plays a pivotal role in various domains. However, traditional manual methods of face analysis are time-consuming, subjective, and prone to errors. Additionally, the sheer volume of image and video data makes it challenging to process and extract meaningful information efficiently. Therefore, there is a pressing need for an automated system that can detect human faces and predict essential attributes such as age, gender, ethnicity, and emotions reliably.

The project titled "Human Face Detector and Analyzer" aims to address the challenges by leveraging computer vision techniques. By developing an automated system capable of accurately detecting human faces and analyzing their attributes, this project offers several significant benefits:

1- Efficiency and Timesaving:

The automated face detection and analysis system can process vast amounts of image and video data rapidly, surpassing human capabilities. This efficiency saves valuable time, especially in scenarios where real-time analysis is crucial, such as security monitoring or video-based customer analytics.

2- Enhanced Security and Surveillance:

The ability to detect and identify human faces accurately is of utmost importance in security and surveillance applications. By integrating the proposed system into existing surveillance networks, security personnel can efficiently identify individuals, track suspicious activities, and enhance overall public safety.

3- Personalized Marketing and User Experience:

The project's face analysis capabilities, including age, gender, and ethnicity prediction, offer valuable insights for marketers and businesses. This information can be utilized to deliver personalized marketing campaigns, tailor products and services to specific demographics, and provide enhanced user experiences that resonate with target audiences.

4- Healthcare and Mental Health Assessment:

Accurate analysis of facial expressions and emotions can have significant implications in healthcare, particularly in mental health assessment. The automated system can aid medical professionals in assessing patients' emotional states, contributing to early detection, precise diagnosis, and effective treatment of mental health disorders.

5- Human-Computer Interaction:

The project's face analysis module opens doors for improved human-computer interaction. By understanding user emotions and facial expressions, intelligent interfaces can respond dynamically, leading to more immersive and engaging experiences in gaming, virtual reality, and other interactive applications.

6- Research and Development:
The project holds value in the field of computer vision research and development. It contributes to advancing the understanding of facial analysis algorithms, deep learning techniques, and optimization methods, fostering innovation and driving progress in the broader discipline of computer vision.

1.3 Objective

The objective of the "Human Face Detector and Analyzer" project is to develop an automated system that can detect human faces in images and videos, and accurately analyze their age, gender, ethnicity, and emotions. The primary goal is to leverage computer vision techniques to provide efficient, reliable, and real-time face detection and analysis capabilities.

Specific objectives include:

1- Face Detection:
Implement robust algorithms to accurately detect human faces in various types of images and videos, considering different lighting conditions, angles, and occlusions. The system should be capable of detecting multiple faces simultaneously and handle complex scenarios effectively.

2- Age Prediction:
Develop a model that can estimate the age of individuals based on their facial features. The system should be able to categorize age into appropriate groups, such as children, teenagers, adults, and seniors, providing valuable demographic information.

3- Gender Classification:
Create a gender classification module that can identify the gender of detected faces accurately. The system should distinguish between male and female individuals, enabling gender-based analysis and personalized targeting in various applications.

4- Ethnicity Recognition:
Design a model capable of recognizing the ethnicity or race of individuals based on their facial features. The system should provide insights into the distribution of ethnic groups, facilitating targeted marketing campaigns and demographic analysis.

5- Emotion Analysis:
Implement an emotion analysis component that can detect and classify facial expressions, capturing a range of emotions such as happiness, sadness, anger, surprise, and more. The system should accurately identify and analyze the emotional states of individuals in real-time.

6- Integration and Real-time Performance:
Integrate the developed face detection and analysis components into a unified system that operates in real-time, ensuring efficient processing and analysis of image and video data. The system should be optimized for performance, balancing accuracy, and speed to deliver timely results.

7- Evaluation and Validation:
Evaluate the performance and accuracy of the system using benchmark datasets and appropriate evaluation metrics. Conduct comprehensive testing to ensure reliable results and validate the system's effectiveness in face detection and attribute analysis tasks.

8- Build API Endpoint:
Build an API end point capable of delivering the project features to other systems easily.

By achieving these objectives, the "Human Face Detector and Analyzer" project aims to contribute to the advancement of computer vision techniques in the context of face analysis. The resulting system will offer practical applications in security, marketing, healthcare, and human-computer interaction, enabling efficient decision-making, personalized experiences, and improved understanding of human behavior.

1.4 Time Plan

The "Human Face Detector and Analyzer" project was executed within a timeframe of 9 months, divided into three distinct phases: studying computer vision and deep learning, implementing the project, and testing and deploying the system. The following is a breakdown of the time plan:

Phase 1: Studying Computer Vision and Deep Learning (3 months)
During the initial phase, spanning three months, the project team dedicated their efforts to acquiring a strong foundation in computer vision and deep learning concepts. The activities included:

1- Literature Review:

Conducting an in-depth study of relevant research papers, books, and online resources to gain comprehensive knowledge about face detection, age estimation, gender classification, ethnicity recognition, and emotion analysis algorithms.

2- Online Courses and Tutorials:

Completing online courses and tutorials focused on computer vision, deep learning frameworks, and relevant libraries such as OpenCV, TensorFlow, or PyTorch. These resources provided practical insights and hands-on experience in implementing face analysis algorithms.

3- Experimentation and Prototyping:

Engaging in small-scale experimentation and prototyping to familiarize themselves with the chosen tools, frameworks, and algorithms. This phase involved implementing and testing basic face detection and analysis techniques.

Phase 2: Project Implementation (4 months)

Following the completion of the studying phase, the project team dedicated four months to implementing the "Human Face Detector and Analyzer" system. The activities during this phase included:

1- System Architecture Design:

Designing the overall system architecture, including the modules for face detection, age prediction, gender classification, ethnicity recognition, and emotion analysis. Defining the necessary data flow and integration between these components.

2- Algorithm Implementation:

Implementing the selected face detection and analysis algorithms based on the acquired knowledge from the study phase. Ensuring proper integration and compatibility of different algorithms and optimizing their performance.

3- User Interface Development:

Creating a user-friendly interface to interact with the system, allowing users to input images or videos for face analysis, displaying the results, and providing options for further analysis or data export.

Phase 3: Testing and Deployment (2 months)
In the final phase of the project, spanning two months, the team focused on testing and deploying the "Human Face Detector and Analyzer" system. The activities included:

1- System Testing:

Conducting rigorous testing to evaluate the system's accuracy, performance, and reliability. Assessing the face detection and attribute analysis results against benchmark datasets and established evaluation metrics.

2- Bug Fixing and Refinement:

Addressing any identified issues, bugs, or performance bottlenecks in the system. Refining the algorithms, optimizing the code, and improving the user interface based on user feedback and testing outcomes.

3- Deployment and Documentation:

Preparing the system for deployment, ensuring its compatibility with the target hardware or platform. Generating comprehensive documentation, including user manuals, installation instructions, and technical specifications.

By adhering to this time plan, the project team successfully completed the "Human Face Detector and Analyzer" project within the allocated 9-month timeframe. The systematic division of time into studying, implementing, testing, and deploying phases enabled efficient progress, ensuring a well-rounded and functional system.

1.5 Document Organization

In Chapter 2 we give a quick background about the project’s field and the recent studies and research about the project field.
We will also give a short introduction about tools, techniques and approaches used in this project.

In Chapter 3 we analyze the system design and functionality using popular methods like UML diagrams.

In Chapter 4 we talk in detail about how the project tasks are implemented and the results of each task besides the tool used and snapshots of the project source codes.

In Chapter 5 we review the project User Manual and How to use or integrate the Application in your Project

In Chapter 6 we conclude what we talked about in this documentation, results of each model, and some of the future ideas we intend to implement and use in this project.

2- Background

There are many research and experiments done to detect the human face and analyze it using CNN.

In 2022, Nippon and Juel proposed an architecture to predict human age, gender and emotion using Deep Learning [1].
Their method suggests using YoloV3 and CNN architecture of 29 layers with 5 convolution blocks for features extraction and a SoftMax function with 11 units.

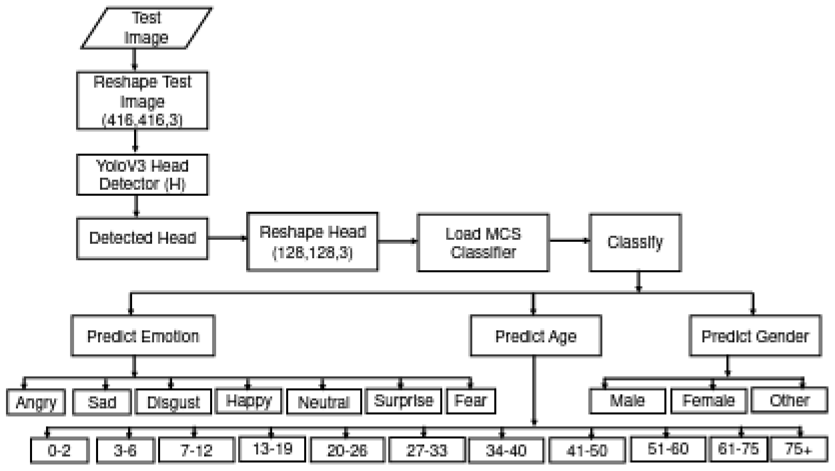


Figure 1: An approach to analyze human face based on CNN by Nippon and Juel.

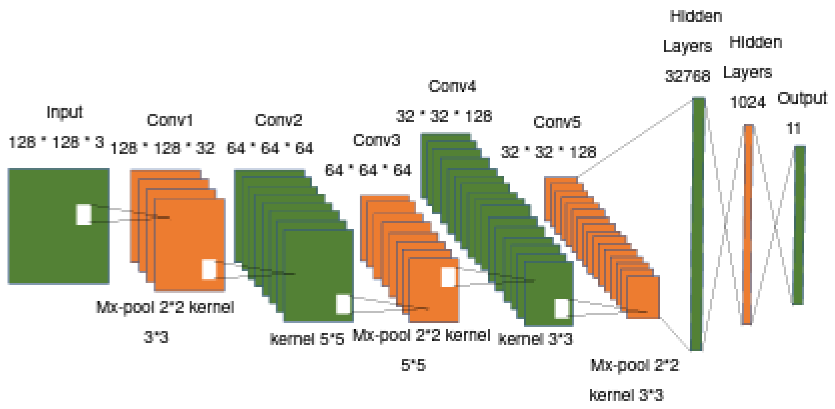


Figure 2: CNN architecture proposed by Nippon and Juel.

This approach achieved accuracy of 88.56% in predicting emotions, 91.67% in gender prediction and 87.95 in age prediction.

Various research and experiments were done in the same field with the following results.

RF.	Detection	Methodology	Accuracy
[2]	Gender	VGG Face CNN	97.45%
[3]	Gender	Deep-CNN	90.33%
[4]	Emotions	CNN	55%
[5]	Emotions	CNN	65%
[3]	Age	Deep-CNN	80.1%
[6]	Ethnicity	CNN	76%

The Choice and the Quality of the Datasets used in training of CNN model have a direct impact on the accuracy of the model.

Most of the research and experiments are done using Commercial private datasets which are hard to get or have high cost.

All experiments done in this project use non-commercial free datasets which sometime lower in quality and have misclassifications and noises which require heavy preprocessing.

Background on Approaches, methodologies and technologies used:

The field of predicting and analyzing human faces using artificial intelligence (AI) has witnessed remarkable advancements over the years.

The exploration of AI in face prediction and analysis began with traditional computer vision techniques. These methods relied on handcrafted features, such as Haar-like features and local binary patterns, combined with machine learning algorithms like support vector machines (SVM) and random forests. Viola and Jones' work on face detection using Haar-like features and the associated cascade classifier [7] laid the foundation for subsequent research in the field.

The advent of deep learning, particularly convolutional neural networks (CNNs), revolutionized the field of face prediction and analysis. Deep learning models enabled automatic feature learning from raw image data, eliminating the need for manual feature engineering. Krizhevsky et al. introduced the influential AlexNet architecture for image classification, which demonstrated the power of deep CNNs [8]

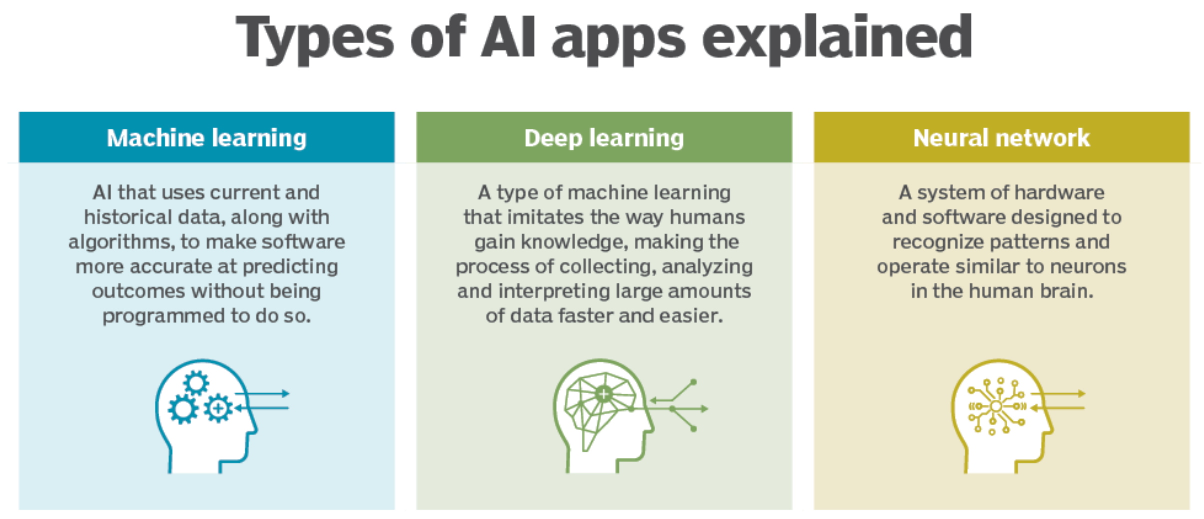

Types of AI apps explained
Machine learning
AI that uses current and historical data, along with algorithms, to make software more accurate at predicting outcomes without being programmed to do so.
Deep learning
A type of machine learning that imitates the way humans gain knowledge, making the process of collecting, analyzing and interpreting large amounts of data faster and easier.
Neural network
A system of hardware and software designed to recognize patterns and operate similar to neurons in the human brain.

Figure 3: types of AI Apps.

A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.

There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice. This makes them highly suitable for computer vision (CV) tasks and for applications where object recognition is vital, such as self-driving cars and facial recognition. Rahul Awati, convolutional neural network (CNN) [9]

CNN’s architecture is analogous to the connectivity pattern of the human brain. Just like the brain consists of billions of neurons, CNNs also have neurons arranged in a specific way. In fact, CNN’s neurons are arranged like the brain's frontal lobe, the area responsible for processing visual stimuli. This arrangement ensures that the entire visual field is covered, thus avoiding the piecemeal image processing problem of traditional neural networks, which must be fed images in reduced-resolution pieces. Compared to the older networks, a CNN delivers better performance with image inputs, and with speech or audio signal inputs.

A deep learning CNN consists of three layers: a convolutional layer, a pooling layer and a fully connected (FC) layer. The convolutional layer is the first layer while the FC layer is the last.

From the convolutional layer to the FC layer, the complexity of the CNN increases. It is this increasing complexity that allows CNN to successively identify larger portions and more complex features of an image until it finally identifies the object in its entirety.

Convolutional layer. Most computations happen in the convolutional layer, which is the core building block of a CNN. A second convolutional layer can follow the initial convolutional layer. The process of convolution involves a kernel or filter inside this layer moving across the receptive fields of the image, checking if a feature is present in the image.

Over multiple iterations, the kernel sweeps over the entire image. After each iteration a dot product is calculated between the input pixels and the filter. The final output from the series of dots is known as a feature map or convolved feature. Ultimately, the image is converted into numerical values in this layer, which allows CNN to interpret the image and extract relevant patterns from it.

Pooling layer. Like the convolutional layer, the pooling layer also sweeps a kernel or filter across the input image. But unlike the convolutional layer, the pooling layer reduces the number of parameters in the input and results in some information loss. On the positive side, this layer reduces complexity and improves the efficiency of CNN.

Fully connected layer. The FC layer is where image classification happens in the CNN based on the features extracted in the previous layers. Here, fully connected means that all the inputs or nodes from one layer are connected to every activation unit or node of the next layer.

All the layers in CNN are not fully connected because it would result in an unnecessarily dense network. It also would increase losses and affect the output quality, and it would be computationally expensive.

A CNN can have multiple layers, each of which learns to detect the different features of an input image. A filter or kernel is applied to each image to produce an output that gets progressively better and more detailed after each layer. In the lower layers, the filters can start as simple features.

At each successive layer, the filters increase in complexity to check and identify features that uniquely represent the input object. Thus, the output of each convolved image -- the partially recognized image after each layer -- becomes the input for the next layer. In the last layer, which is an FC layer, the CNN recognizes the image or the object it represents.

With convolution, the input image goes through a set of these filters. As each filter activates certain features from the image, it does its work and passes on its output to the filter in the next layer. Each layer learns to identify different features and the operations end up being repeated for dozens, hundreds or even thousands of layers. Finally, all the image data progressing through the CNN's multiple layers allow the CNN to identify the entire object.

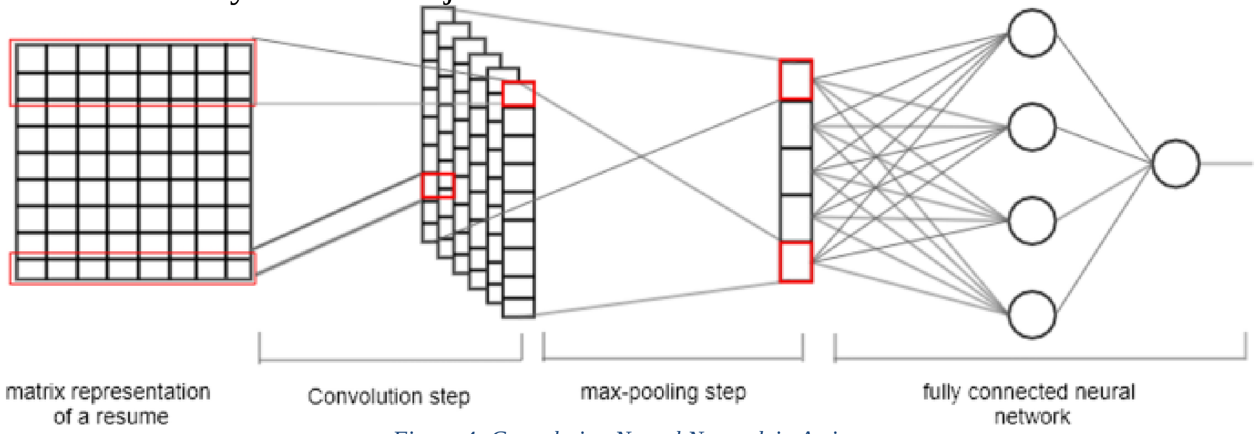

matrix representation of a resume
Convolution step
max-pooling step
fully connected neural network

Figure 4: Convolution Neural Network in Action.

Face Detection:
There are several famous methods for detecting human faces from images like:

Viola-Jones Algorithm: The Viola-Jones algorithm is a widely used method for face detection. It utilizes Haar-like features and a cascade classifier to detect faces in images. This method, introduced by Viola and Jones in 2001, marked a significant advancement in real-time face detection. [7]

Convolutional Neural Networks (CNNs): CNNs have revolutionized face prediction and analysis. Models like AlexNet, VGGNet, and ResNet have been applied to tasks such as face recognition, age estimation, and facial expression analysis. These deep learning models can automatically learn features from raw image data, enabling accurate predictions. [8] [10]

DeepFace: DeepFace, introduced by Facebook's AI Research team, is a deep learning model that achieves near-human performance in face verification tasks. It employs deep CNNs combined with a Siamese network architecture to learn face embeddings for accurate face recognition. [11]

FaceNet: FaceNet is another influential method for face recognition developed by Google researchers. It employs deep CNNs and triplet loss to learn discriminative face embeddings. FaceNet's embeddings can be used to measure similarity between faces, enabling accurate face identification. [12]

Generative Adversarial Networks (GANs): GANs have been applied to various face-related tasks, including face generation and face attribute manipulation. Models like StyleGAN and CycleGAN can generate highly realistic and diverse faces, demonstrating the potential of GANs in face prediction. [13]

The Method used in this project is Viola-Jones Algorithm also called haar cascade algorithm. Despite being one of oldest algorithms in face detection field, it is still one of easiest and most accurate methods.

The Haar cascade front face algorithm is a computer vision technique used for face detection. It takes advantage of a mathematical concept known as Haar features, which are rectangular regions in an image that capture different patterns of contrast.

The algorithm works by sliding a window-like structure, called a cascade, over an image in a step-by-step manner. At each step, the algorithm calculates the Haar-like features within the window to determine if they resemble a face. These features can include variations in brightness, edge orientations, and texture patterns.

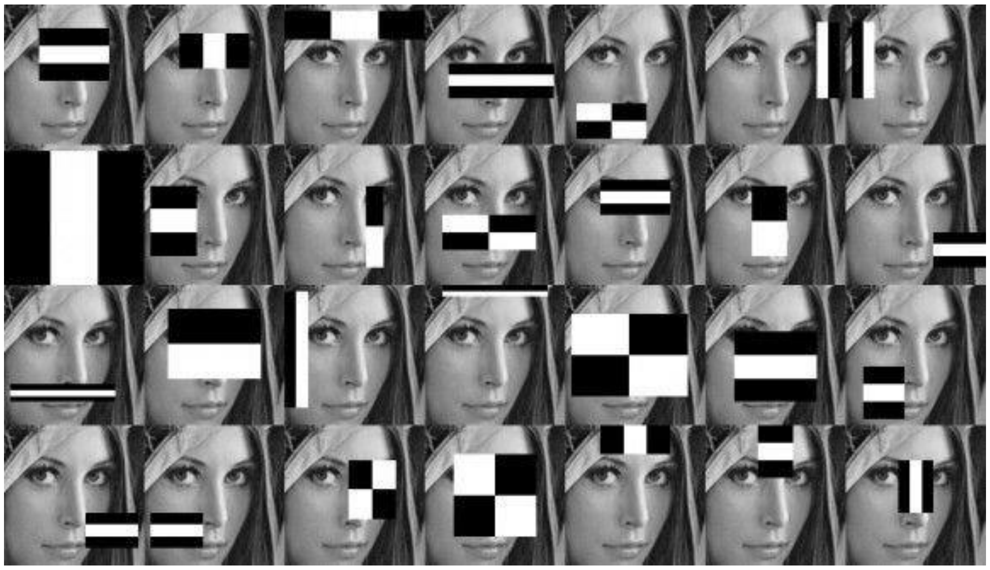


Figure 5:Haar Cascade Algorithm Sliding Window.

To identify a face, the algorithm compares the calculated Haar features with a pre-trained model, which contains patterns commonly found in human faces. The model consists of positive and negative samples that help the algorithm distinguish between face and non-face regions.

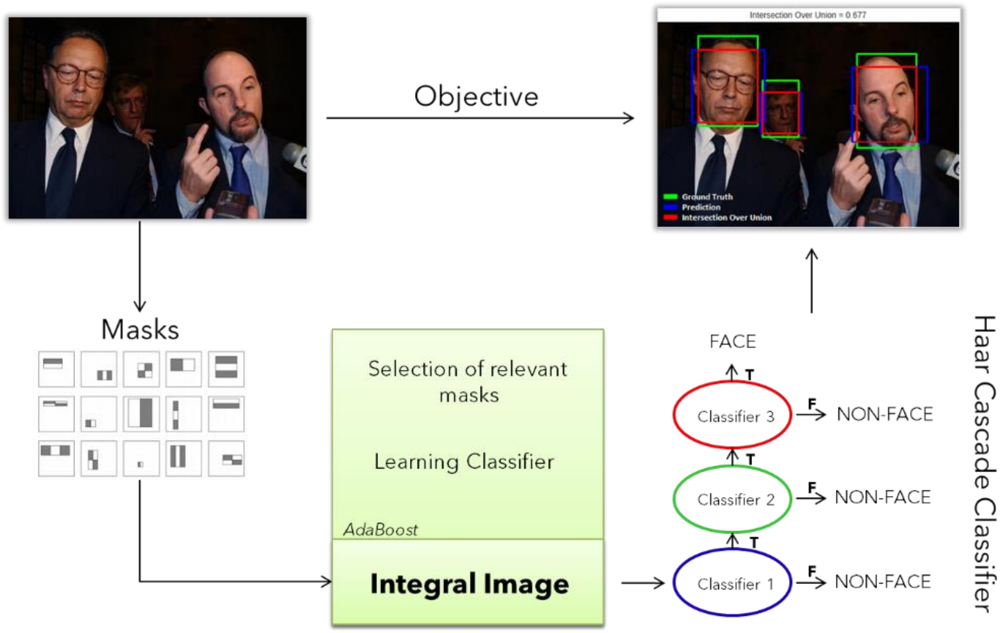


Figure 6: The haar cascade algorithm in action.

At the beginning of the process, the algorithm scans the image at a larger scale, looking for simpler features like edges and corners. As it progresses, it narrows down the search to regions that are more likely to contain faces. The algorithm applies a series of classifiers that refine the detection, eliminating false positives and enhancing the accuracy.

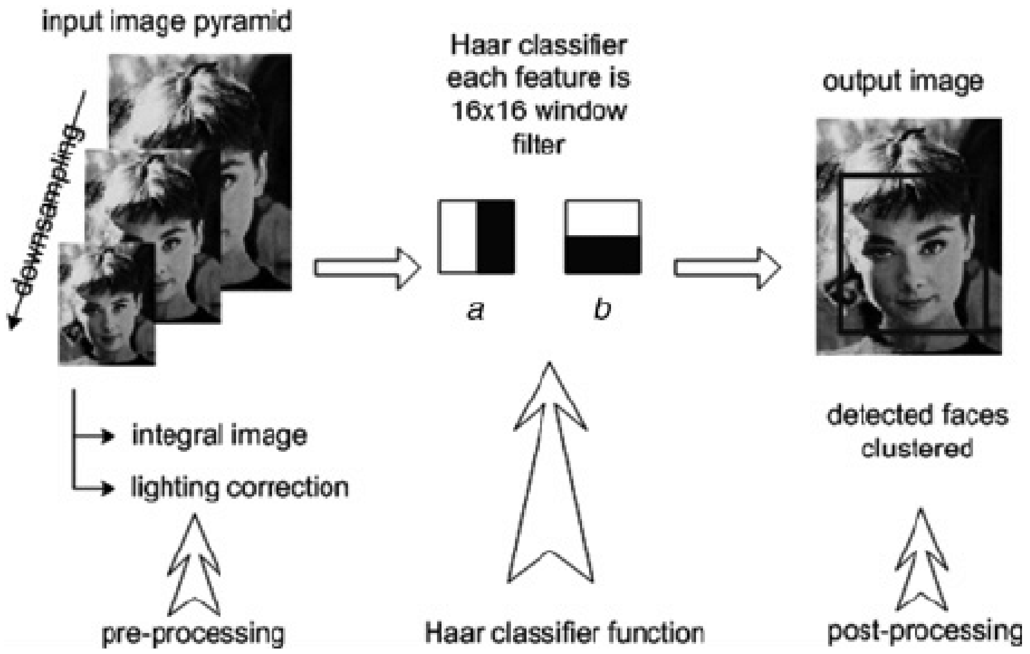


Figure 7: The Haar Cascade Algorithm in action 2.

The term "cascade" refers to the structure of the algorithm, where multiple stages are organized in a hierarchical manner. Each stage consists of a set of weak classifiers that collectively contribute to the decision of whether a region contains a face or not. The cascade structure allows for efficient processing, as regions that are unlikely to contain faces are quickly rejected, reducing the computational load.

This approach was chosen for this project due to the computation effectiveness of it and its accuracy.

Data Augmentation:

To use approaches like CNN, large Datasets are needed. At early times there was a lack in datasets to use in CNN applications so many techniques were proposed to deal with these problems.

One of the most famous techniques is the Data Augmentation Technique.

Data augmentation refers to the process of expanding a given dataset by generating new samples through various transformations or modifications. It is commonly used in machine learning and deep learning tasks to increase the size and diversity of the training data.

The purpose of data augmentation is to address the limitations of a small or limited dataset by artificially creating additional training examples. By introducing variations and augmenting the dataset, models can learn to generalize better and perform more accurately on unseen data.

Data augmentation techniques can be applied to various types of data, including images, text, audio, and more. Here, I'll focus on image dataset augmentation, which is one of the most widely used types.

Image data augmentation involves applying different transformations to the existing images in the dataset, generating new images with modified attributes. Some common image dataset augmentation techniques include:

1- Image flipping: Horizontally or vertically flipping the image.

2- Image rotation: Rotating the image by a certain angle.

3- Image scaling: Resizing the image to a larger or smaller size.

4- Image translation: Shifting the image horizontally or vertically.

5- Image shearing: Skewing the image along a specified axis.

6- Image cropping: Selecting a specific region of interest within the image.

7- Image zooming: Zooming in or out of the image.

8- Image brightness and contrast adjustment: Changing the brightness and contrast levels of the image.

9- Image noise addition: Adding random noise to the image.

By applying these transformations to the original images and adding them to the dataset, the model gets exposed to a wider range of variations and becomes more robust to different conditions during training. It helps prevent overfitting and improves the model's ability to generalize well on unseen data.

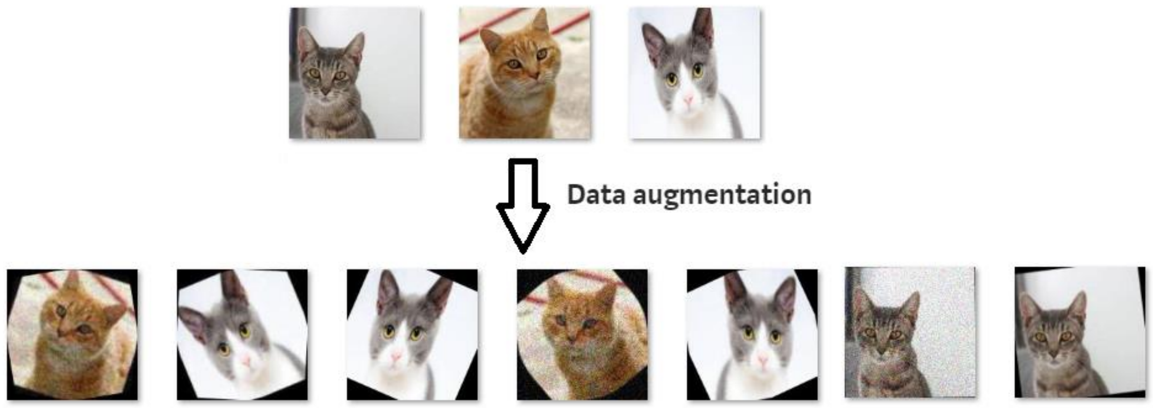


Figure 8: Data Augmentation in CNN.

Optimizers:
In the field of computer vision and CNN there is a need to optimize the hyper parameters used in the training of the models.

Some of those hyper parameters like learning rate are hard to optimize.
There are many techniques proposed to optimize such parameters like Adam Optimizer proposed by Diederik P. Kingma and Jimmy Ba in their paper titled "Adam: A Method for Stochastic Optimization" published in 2014 [14].

The learning rate is a hyperparameter that determines the step size at which the parameters of a machine learning model are updated during the training process. In simpler terms, it controls how quickly or slowly a model learns from the data.

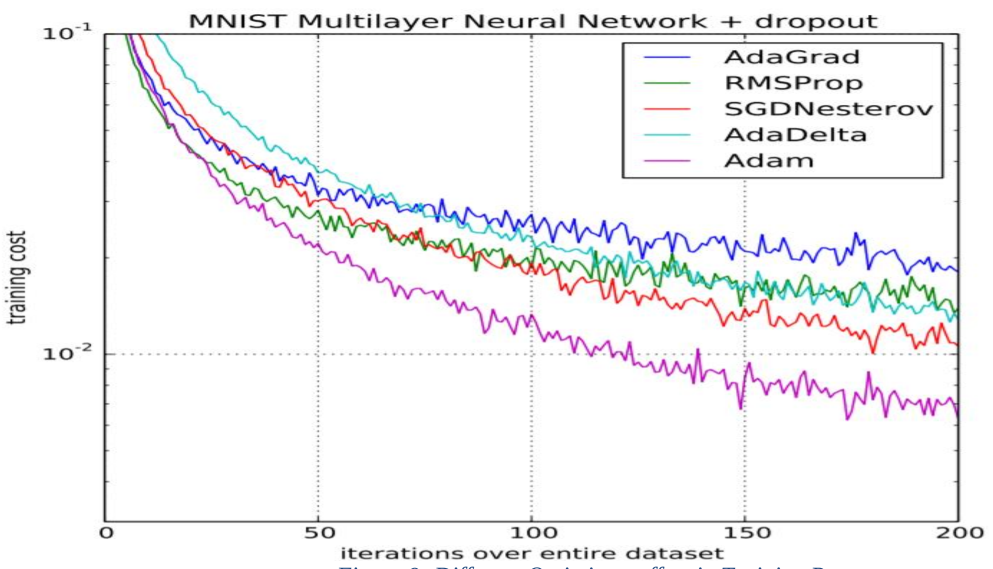


Figure 9: Different Optimizers effect in Training Process.

The Adam optimizer is a popular optimization algorithm used in machine learning and deep learning. It is an extension of the stochastic gradient descent (SGD) algorithm, which is commonly used for updating the parameters of a neural network during the training process.

Adam stands for "Adaptive Moment Estimation." It is called adaptive because it adapts the learning rate for each parameter individually based on the past gradients and squared gradients. It estimates the moments of the gradients, including both the first moment (the mean) and the second moment (the uncentered variance), to compute the adaptive learning rates.

To understand how the Adam optimizer works, let's break down its key components:

1- Gradients: In the training process, the gradients of the model's parameters are calculated to determine the direction and magnitude of the updates. These gradients indicate the slope of the loss function with respect to each parameter.

2- Moments: The moments are statistical measures that describe the distribution of data. In Adam, two moments are computed for each parameter: the first moment, also known as the moving average of the gradients, and the second moment, which is the moving average of the squared gradients.

3- Bias correction: Since the moments are initialized to zero, they are biased towards zero, particularly at the beginning of training. To address this, Adam employs a bias correction mechanism to adjust the moments during the initial iterations, making them closer to their true values.

4- Adaptive learning rate: The Adam optimizer computes individual learning rates for each parameter by combining the first and second moments. The first moment provides information about the average gradient, while the second moment indicates how the gradients vary. By taking these factors into account, Adam adapts the learning rate for each parameter accordingly. This adaptivity is beneficial because it allows the optimizer to adjust the step size for different parameters, which can result in faster convergence and better performance.

5- Parameter updates: Finally, the Adam optimizer uses the computed learning rates to update the parameters of the model. It multiplies the learning rate by the first moment (mean) of the gradient to determine the step size and divides it by the square root of the second moment (variance) to normalize the update. This normalization helps prevent the updates from being too large or too small.

Class Balancing:

Class balancing in AI refers to the process of addressing the imbalance of class distribution in a dataset during training. Class imbalance occurs when the number of samples in each class of a classification problem is significantly different. This imbalance can affect the learning process and bias the model towards the majority class, resulting in poorer performance on the minority class.

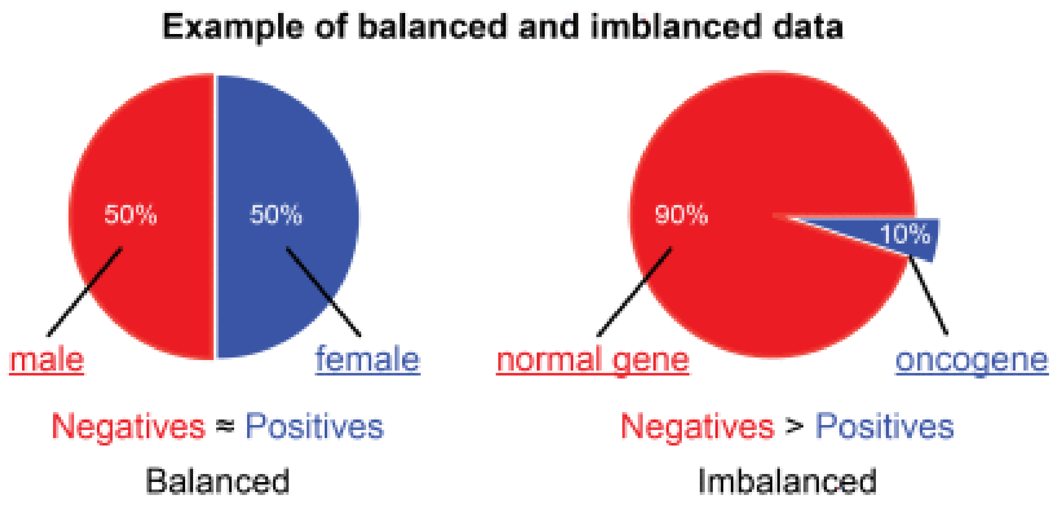


Figure 10: Balanced and Imbalanced Classes.

Class balancing techniques aim to mitigate the impact of class imbalance by ensuring that the model receives sufficient exposure to samples from the minority class. Some common methods for class balancing include:

1- Oversampling: Oversampling involves replicating samples from the minority class to increase their representation in the dataset. This can be done randomly or using more advanced techniques like SMOTE (Synthetic Minority Over-sampling Technique), which creates synthetic samples based on the characteristics of existing minority class samples.

2- Under sampling: Under sampling involves reducing the number of samples from the majority class to match the size of the minority class. Random under sampling randomly removes samples from the majority class, but care must be taken to avoid losing important information. More sophisticated methods like Tomek links or Edited Nearest Neighbors (ENN) can be used for selective under sampling.

3- Class weights: Class weights assign higher weights to samples from the minority class during training. By adjusting the weights, the model pays more attention to the minority class, effectively balancing the contribution of each class to the overall loss function. Class weights can be incorporated in various machine learning algorithms, such as decision trees, support vector machines, and neural networks.

4- Data augmentation: Data augmentation techniques artificially increase the size of the minority class by applying transformations such as rotation, scaling, or flipping to existing samples. This helps in introducing diversity and reducing the class imbalance.

5- Ensemble methods: Ensemble methods combine multiple models to improve performance. When dealing with class imbalance, ensemble techniques can create a diverse set of models that specialize in different aspects of the problem, including handling the minority class. By aggregating predictions from multiple models, ensemble methods can achieve better overall performance.

In this project we use class weights Balancing to balance our data.

Transfer Learning:
Transfer learning in Convolutional Neural Networks (CNNs) is a technique that leverages the knowledge and learned representations from pre-trained models to improve the performance of a target task or dataset.

In traditional deep learning approaches, CNN models are trained from scratch on large datasets, requiring substantial computational resources and labeled data. However, transfer learning allows us to benefit from existing knowledge and models that have been trained on similar or related tasks.

The main idea behind transfer learning is to utilize the learned representations, or features, from a pre-trained CNN model as a starting point for the new task. By doing so, we can take advantage of the generalization power and feature extraction capabilities of the pre-trained model, which has typically been trained on large-scale datasets such as ImageNet.

The transfer learning process typically involves the following steps:

1- Pre-training: A CNN model is trained on a large dataset for a different but related task, such as image classification. This pre-training step helps the model learn meaningful and generic features that can be applied to various tasks.

2- Feature extraction: The pre-trained CNN model is used as a feature extractor. The learned weights and architecture of the model are frozen, and only the early layers (convolutional layers) are used to extract features from the new dataset. These extracted features capture higher-level representations of the data.

3- Fine-tuning: Optionally, the frozen layers of the pre-trained model can be fine-tuned on the target task or dataset. This step involves unfreezing some of the layers, typically the later layers, to allow them to adapt and learn task-specific features. The fine-tuning process helps in optimizing the model's parameters specifically for the new task.

Transfer learning offers several advantages:

1- Reduced training time: By utilizing pre-trained models, the need for training from scratch on a large dataset is reduced, leading to significant time savings.

2- Improved performance: The pre-trained model has already learned generic features from a large dataset, which can be highly beneficial, especially when the target dataset is small. It helps to overcome the scarcity of labeled data and can improve the model's generalization ability.

3- Transfer of knowledge: Transfer learning allows the transfer of knowledge from one task to another. The pre-trained model captures useful representations that can be applicable to different but related tasks, even if the specific target task has different classes or characteristics.

4- Effective feature extraction: The early layers of a CNN can extract low-level visual features that are generally useful across tasks, such as detecting edges or textures. Leveraging these learned features saves time and computational resources.

Transfer learning has found widespread use in various computer vision tasks, such as image classification, object detection, and image segmentation. It enables practitioners to build accurate models with limited resources and data, making it a valuable technique in the field of deep learning.

We use VGG-16 architecture with VGG-Face weights in this project.

Heat Maps:
Heat maps are often used for visualizing the importance or relevance of different regions in an input image that contributes to the network's output or prediction.

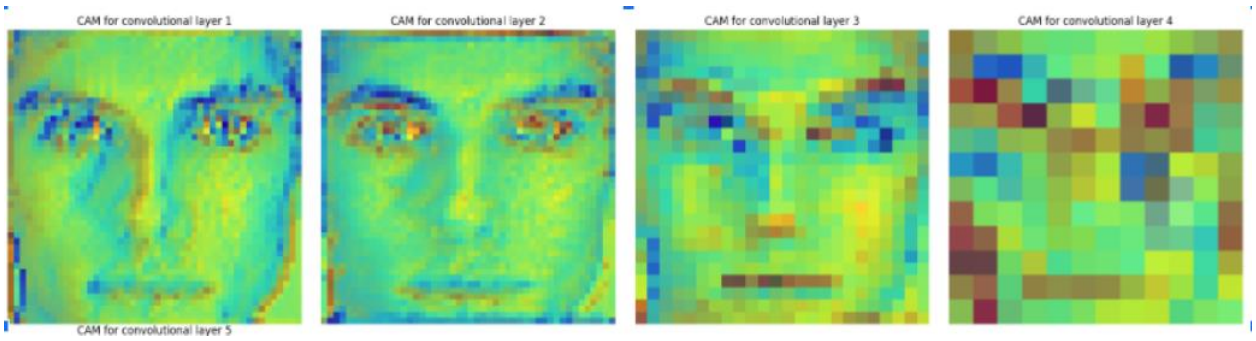


Figure 11: Heat Maps in CNN.

Heat maps in CNNs are typically generated using a technique called Grad-CAM (Gradient-weighted Class Activation Mapping) or similar methods. Grad-CAM computes the gradients of the target class score with respect to the feature maps in the final convolutional layer of the CNN. These gradients highlight the importance of different spatial locations or feature maps in determining the final prediction.

Feature Maps:
Feature maps refer to the outputs of the convolutional layers in the network. These feature maps capture the learned representations of the input data at various levels of abstraction.

Each feature map corresponds to a specific filter or kernel applied to the input image or previous layer's feature maps. The filter slides across the input data, computing the dot product between its weights and the local receptive field, resulting in a single value. By applying multiple filters, multiple feature maps are generated, each capturing different patterns or features in the input data.

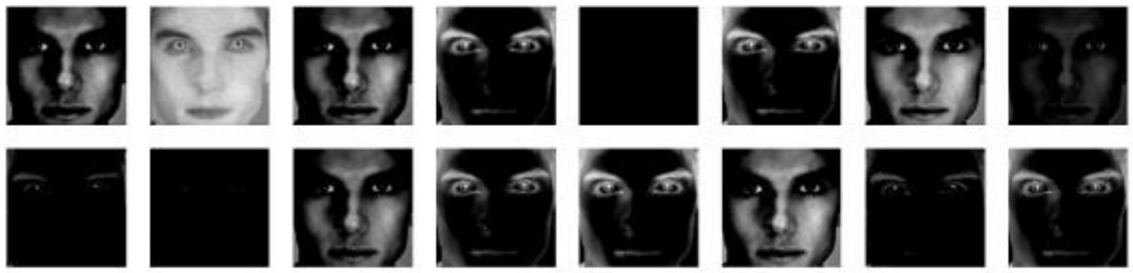


Figure 12: Feature Maps in CNN.

Convergence Plots:
Convergence plots in CNN refer to graphical representations that show how the performance of a CNN model changes over the course of training iterations or epochs. These plots provide insights into the convergence behavior of the model during the training process.

Convergence plots typically display the values of a specific metric, such as the training loss or accuracy, on the y-axis, and the number of training iterations or epochs on the x-axis. The metric can be calculated based on the comparison between the model's predictions and the ground truth labels for a given dataset.

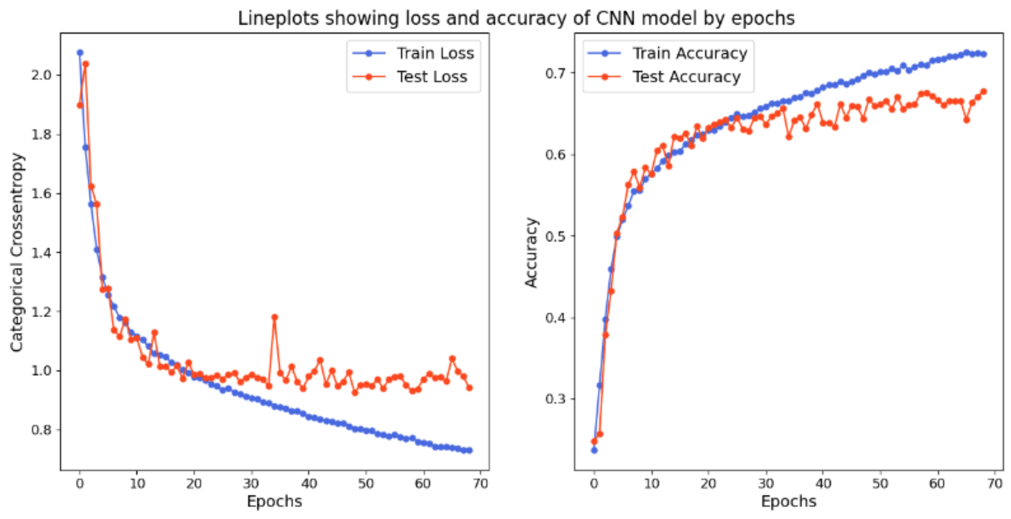


Figure 13: Convergence Plots in CNN.

Confusion Matrix:
A confusion matrix in CNN is a table that summarizes the performance of a classification model by showing the count or proportion of correct and incorrect predictions made by the model for each class in the dataset.

The confusion matrix is especially useful when dealing with multi-class classification problems, where there are more than two classes or categories. It provides a detailed breakdown of the model's predictions and helps to evaluate its performance in terms of different types of errors.

A confusion matrix typically has the following components:

1- True Positives (TP): The number of instances correctly predicted as positive (belonging to a particular class).

2- True Negatives (TN): The number of instances correctly predicted as negative (not belonging to a particular class).

3- False Positives (FP): The number of instances incorrectly predicted as positive (false alarms or Type I errors).

4- False Negatives (FN): The number of instances incorrectly predicted as negative (misses or Type II errors).

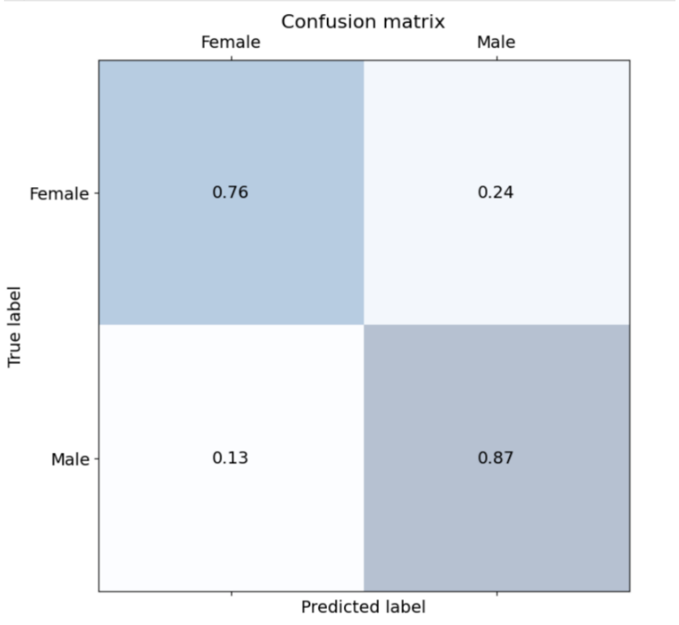


Figure 14: Confusion Matrix in CNN.

Datasets:
In CNN, a dataset refers to a collection of input samples and their corresponding labels that are used to train, validate, and test the CNN model. The dataset plays a fundamental role in machine learning, as it provides the model with examples to learn from and evaluate its performance.

A typical CNN dataset consists of two main components:

1- Input Samples: The input samples in a CNN dataset are usually images or other multidimensional data, such as audio spectrograms or 3D volumes. Each sample represents an individual instance or observation. The input samples are structured as tensors with specific dimensions, such as height, width, and depth (for color images) or channels (for other types of data).

2- Labels: The labels in a CNN dataset are the corresponding ground truth values or categories that indicate the desired output or class for each input sample. For example, in an image classification task, the labels might represent the object category depicted in the image (e.g., "cat," "dog," "car," etc.). The labels are often represented as numerical values or one-hot encodings, where each label corresponds to a unique index or binary vector.

Datasets for CNN training typically involves three main subsets:

1- Training Dataset: The training dataset is used to train the CNN model. It consists of many labeled samples, providing the model with diverse examples to learn from. The training dataset is essential for the model to capture patterns, extract features, and optimize its parameters.

2- Validation Dataset: The validation dataset is used to monitor the model's performance and fine-tune its hyperparameters during training. It contains labeled samples that the model has not seen during training. The validation dataset helps assess the model's generalization and prevents overfitting by providing an independent evaluation of its performance.

3- Test Dataset: The test dataset is used to evaluate the final performance of the trained CNN model. It consists of labeled samples that are independent of both the training and validation sets. The test dataset provides an unbiased assessment of the model's accuracy, allowing practitioners to estimate its performance on unseen data.

Figure 15: Datasets in CNN.

Python:
Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released

in 1991. Python emphasizes code readability, with a clear and concise syntax that allows programmers to express concepts in fewer lines of code compared to other programming languages. It is considered the best programming language for AI Tasks.

Native Keras:
Native Keras refers to the original implementation of the Keras deep learning library, which was developed as an independent project by François Chollet [15]. Native Keras was initially built on top of Theano and later expanded to support TensorFlow as well.

Native Keras offered a user-friendly, high-level API for designing, training, and evaluating neural networks. It provided a simplified interface that made it easy to define network architectures, specify layer configurations, and compile models with various optimization algorithms and loss functions.

TensorFlow:
TensorFlow is an open-source machine learning framework developed by Google. It provides a comprehensive set of tools, libraries, and resources for building and deploying machine learning models, including deep learning models. TensorFlow is widely used in both research and production environments due to its flexibility, scalability, and extensive ecosystem.

CV2:
cv2, or OpenCV (Open-Source Computer Vision Library), is an open-source computer vision and image processing library. It provides a comprehensive set of functions and algorithms for various computer vision tasks, such as image and video manipulation, feature detection, object recognition, and more.

OpenCV is written in C++ and has interfaces for various programming languages, including Python. The Python interface, known as cv2, allows developers to utilize OpenCV functionalities in Python applications, making it a popular choice for computer vision tasks in the Python ecosystem.

VGG-16:
VGG16 refers to a convolutional neural network (CNN) architecture that was introduced by the Visual Geometry Group (VGG) at the University of Oxford [10]. VGG16 is specifically named after the fact that it has 16 layers, including 13 convolutional layers and 3 fully connected layers.

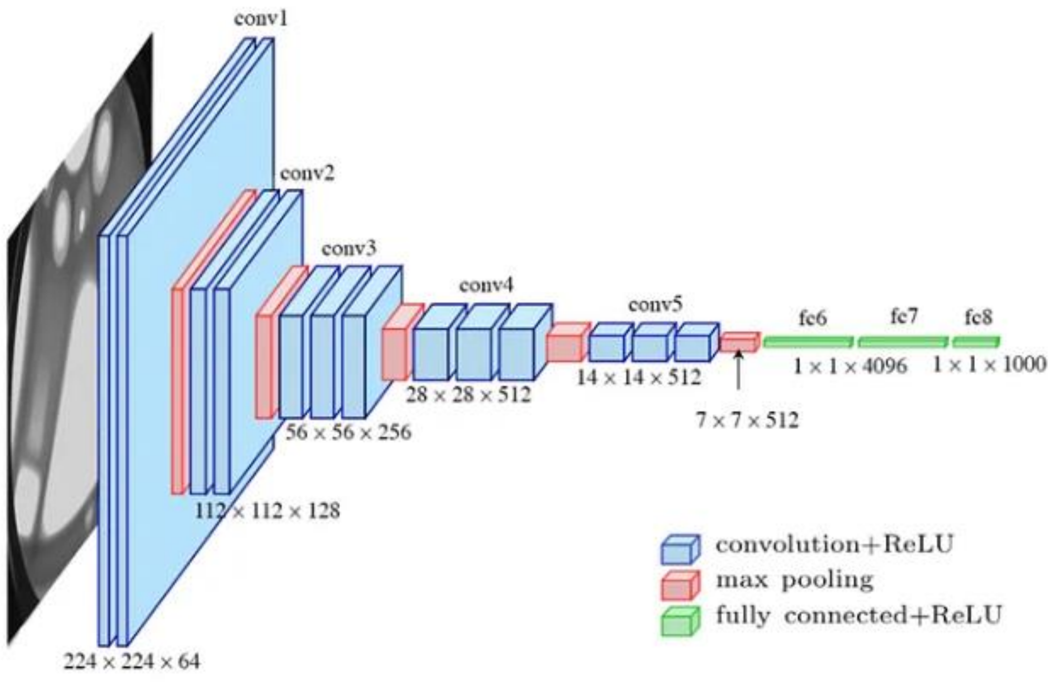


Figure 16: VGG-16 Architecture.

The VGG16 architecture is renowned for its simplicity and effectiveness in image classification tasks. It played a crucial role in demonstrating the power of deeper CNN architectures and their ability to achieve high accuracy on image recognition benchmarks, such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

Key characteristics of the VGG16 architecture include:

1- Sequential Convolutional Layers: VGG16 consists of a sequence of convolutional layers, followed by max-pooling layers. These layers perform feature extraction by applying convolutional filters to input images, capturing different levels of image features.

2- Small Convolutional Filters: VGG16 mainly employs 3x3 convolutional filters throughout the network. The use of smaller filters instead of larger ones enables deeper networks while keeping the number of parameters manageable.

3- Deep Architecture: With its 16 layers, VGG16 has a relatively deep architecture compared to previous CNN models at the time of its introduction. The deeper network allows for the modeling of more complex and hierarchical features in the data.

4- Fully Connected Layers: The last few layers of VGG16 consist of fully connected layers that learn high-level representations from the extracted features. These layers combine the extracted features and perform the final classification.

The pre-trained weights of VGG16 on large-scale datasets like ImageNet have been made publicly available, allowing researchers and practitioners to use transfer learning for various computer vision tasks. By utilizing these pre-trained weights, it becomes possible to leverage the knowledge and learned representations from VGG16 for different image classification or feature extraction tasks, even with limited amounts of task-specific training data.

VGG16 has been widely adopted as a benchmark model in computer vision research and has influenced subsequent CNN architectures. It has demonstrated excellent performance in tasks such as image classification, object detection, and localization. However, due to its depth and number of parameters, VGG16 can be computationally expensive and memory-intensive to train from scratch on large-scale datasets. Transfer learning with pre-trained weights is often preferred to benefit from its learned representations while reducing training time and resource requirements.

VGG-Face:

VGG Face refers to a convolutional neural network (CNN) model that was specifically designed for face recognition tasks. It is based on the VGG16 architecture, which was originally developed for image classification. VGG Face extends the VGG16 model by fine-tuning it on face-related datasets to learn representations that are effective for face recognition.

The VGG Face model aims to capture and encode facial features in a way that is useful for identifying and verifying faces. It utilizes the deep architecture and hierarchical features of VGG16 to extract discriminative facial characteristics at various levels of abstraction.

H5 format:
The H5 format refers to the Hierarchical Data Format version 5, which is a file format commonly used in AI and TensorFlow to store model weights.

JSON:
JSON (JavaScript Object Notation) is a lightweight data interchange format that is widely used for transmitting and storing data. It is a text-based format that is easy for humans to read and write, and it is also easy for machines to parse and generate. JSON is language-independent, which means it can be used with various programming languages.
JSON represents data as key-value pairs, where the keys are strings and the values can be of different types, including strings, numbers, Booleans, arrays, and nested objects. The key-value pairs are enclosed in curly braces {}, and multiple pairs are separated by commas.

API:
API stands for Application Programming Interface. It is a set of rules and protocols that allows different software applications to communicate and interact with each other. APIs define the methods and data formats that applications can use to request and exchange information.
In the context of web development, an API typically refers to a set of rules and endpoints that enable communication between a client (such as a web browser or a mobile app) and a server. The server exposes specific functionalities or data through the API, allowing clients to make requests and receive responses.

Flask:
Flask is a lightweight and flexible web framework for Python. It provides a simple and minimalistic approach to building web applications. Flask is known for its simplicity, ease of use, and extensibility, making it a popular choice for developers of all levels of experience.

Postman:
Postman is a popular API development and testing tool that allows developers to simplify the process of building, testing, and documenting APIs. It provides a user-friendly interface for making HTTP requests to APIs and inspecting the responses.

Yolo:
YOLO (You Only Look Once) is an object detection algorithm that is widely used in computer vision and deep learning. It is known for its speed and accuracy in

real-time object detection tasks. YOLO takes an input image or video and outputs bounding boxes and class labels for the objects it detects.

The main idea behind YOLO is to divide the input image into a grid and predict bounding boxes and class probabilities for each grid cell. YOLO performs this prediction in a single pass through the neural network, which makes it highly efficient and fast compared to other object detection algorithms.

Bootstrap:
Bootstrap is a popular front-end framework for building responsive and mobile-first websites and web applications. It provides a collection of pre-designed HTML, CSS, and JavaScript components, along with a grid system and various utility classes, that help developers create visually appealing and consistent user interfaces.

HTML:
HTML (Hypertext Markup Language) is the standard markup language used for creating and structuring web pages. It provides a set of tags or elements that define the structure and content of a web document. HTML elements are represented by opening and closing tags, and they can contain text, images, links, tables, forms, and various other types of multimedia.

3- Analysis and Design

3.1 System Overview

3.1.1 System Architecture

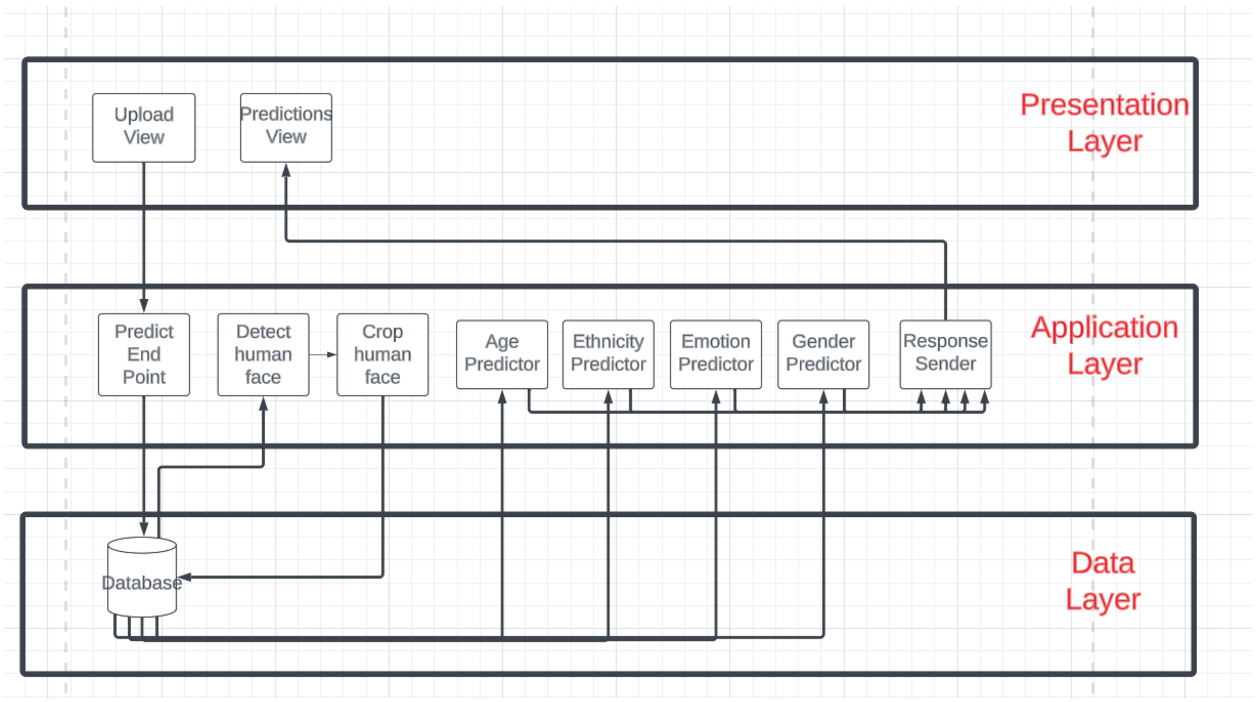


Figure 17: 3-Tiers System Architecture.

The presentation layer consists of the upload view which is the view that is used to upload the images for predictions and send it to the API Application Endpoint and the prediction view which is used to show the received predictions from the API Application.

The application layer consists of the processes that is used to make the predictions using each model and the main process "Predict Endpoint" which is used to receive the requests and the process "Detect Human Face" which apply haar cascade algorithm to detect human face then extract it using "crop Human Face" process. the predictions are received in the "response sender" process which sends the predictions back to the "prediction view".

The image received through "predict Endpoint" Process is saved in the database.

The saved image is retrieved in the “Detect Human Face” process which then call the process “Crop human Face” which extract the human face then save the extracted face in the database.

Each prediction process retrieves a version of the human face image saved in the database and applies preprocessing and corresponding model’s weights on it and send predictions to “response sender” process.

All images are deleted from the database after the predictions are sent.

3.1.2 System Users

A. Intended Users:

1- Any User with access to the internet who wants to receive any of the project services.

2- Any Developer that wants to integrate the project services in their system using the API Application.

B. User Characteristics

1- Users with access to the internet and have the basic knowledge of how to use the web browser and internet that want to receive any of the project services.

2- Developers with technical knowledge about backend and API technologies.

3.2 System Analysis & Design

3.2.1 Use Case Diagram

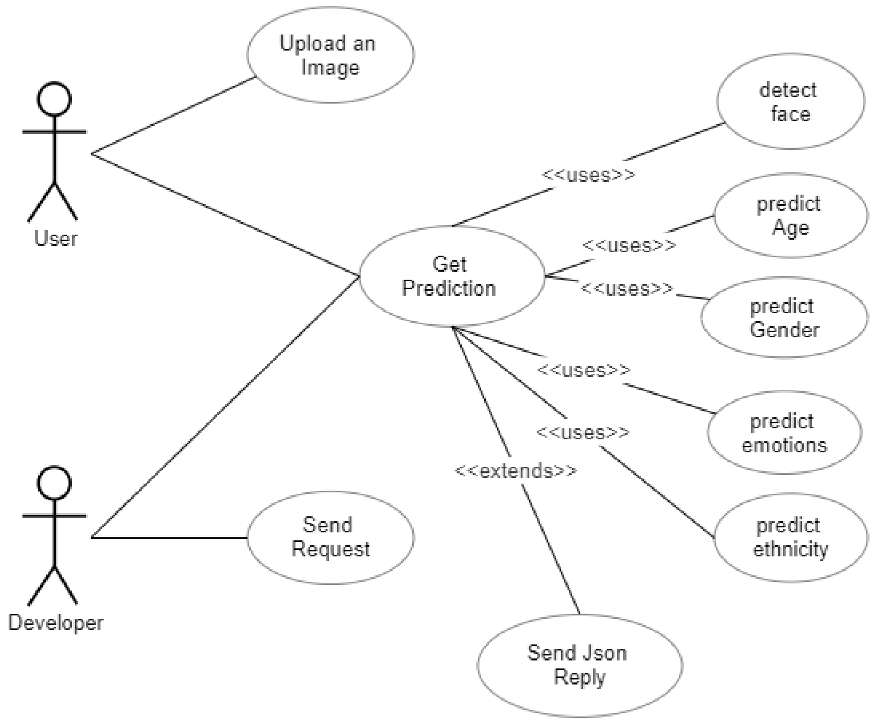


Figure 18: Use Case Diagram.

User Can upload an image and get prediction.
Developers can send an API request and get predictions.
Get prediction Function use:

1- Detect Face to detect human face from image and extract it.
2- Predict Age Function to predict human age using the age model.
3- Predict Emotions Function to predict human emotions using the Emotion model.
4- Predict Gender Function to predict human gender using the gender model.
5- Predict Ethnicity Function to predict human ethnicity using the ethnicity model.
6- Send Json Reply to send the predictions to Developer in JSON format.

3.2.2 Class Diagram

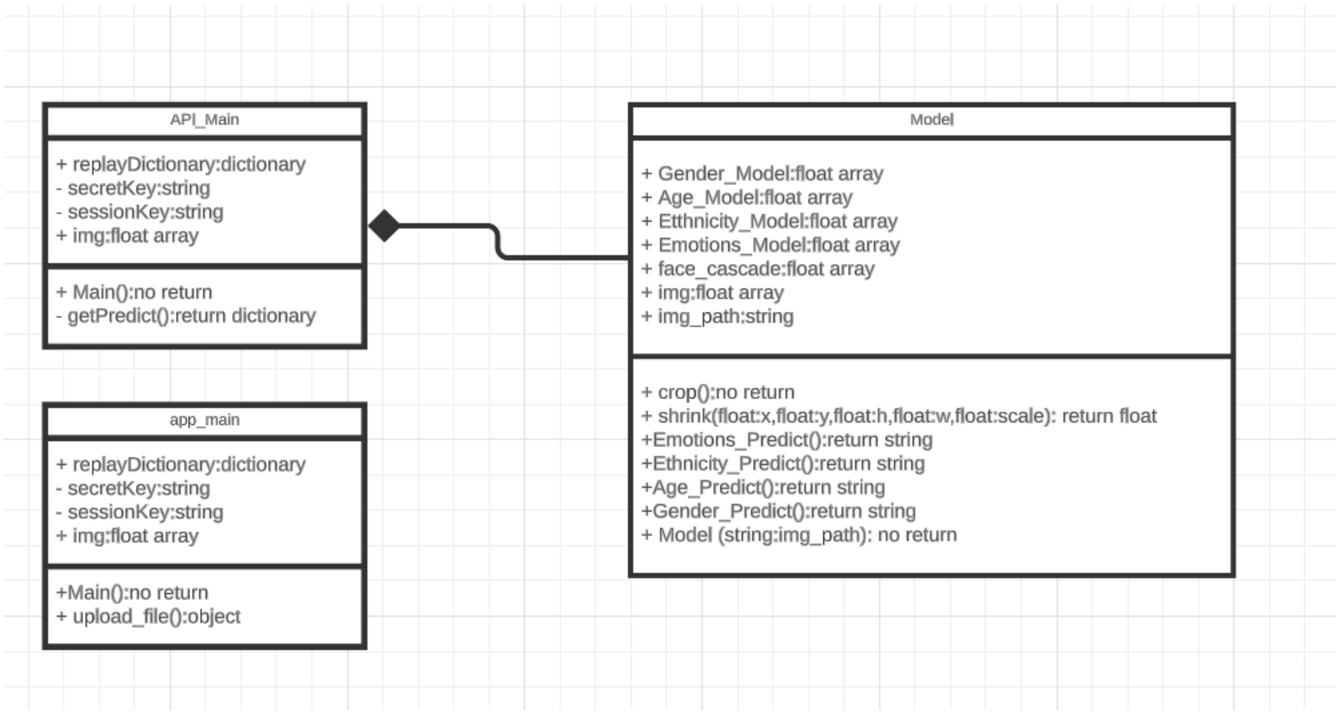

API_Main
+ replayDictionary:dictionary
- secretKey:string
- sessionKey:string
+ img:float array
+ Main():no return
- getPredict():return dictionary
app_main
+ replayDictionary:dictionary
- secretKey:string
- sessionKey:string
+ img:float array
+Main():no return
+ upload_file():object
Model
+ Gender_Model:float array
+ Age_Model:float array
+ Etthnicity_Model:float array
+ Emotions_Model:float array
+ face_cascade:float array
+ img:float array
+ img_path:string
+ crop():no return
+ shrink(float:x,float:y,float:h,float:w,float:scale): return float
+Emotions_Predict():return string
+Ethnicity_Predict():return string
+Age_Predict():return string
+Gender_Predict():return string
+ Model (string:img_path): no return

Figure 19: Class Diagram.

The API_Main class is responsible for receiving requests and sending replies using getPredict class which initializes an object from Model Class and calls its methods. The Model Class is responsible for receiving an image from the API_Main Class and apply face detection and predictions on it and send the predictions to the caller class.

The app_main class is responsible for getting the image from user and sending it to the API Endpoint and when it receives the predictions, it shows it to the user in HTML page.

3.2.3 Sequence Diagram

1- Web APP User:

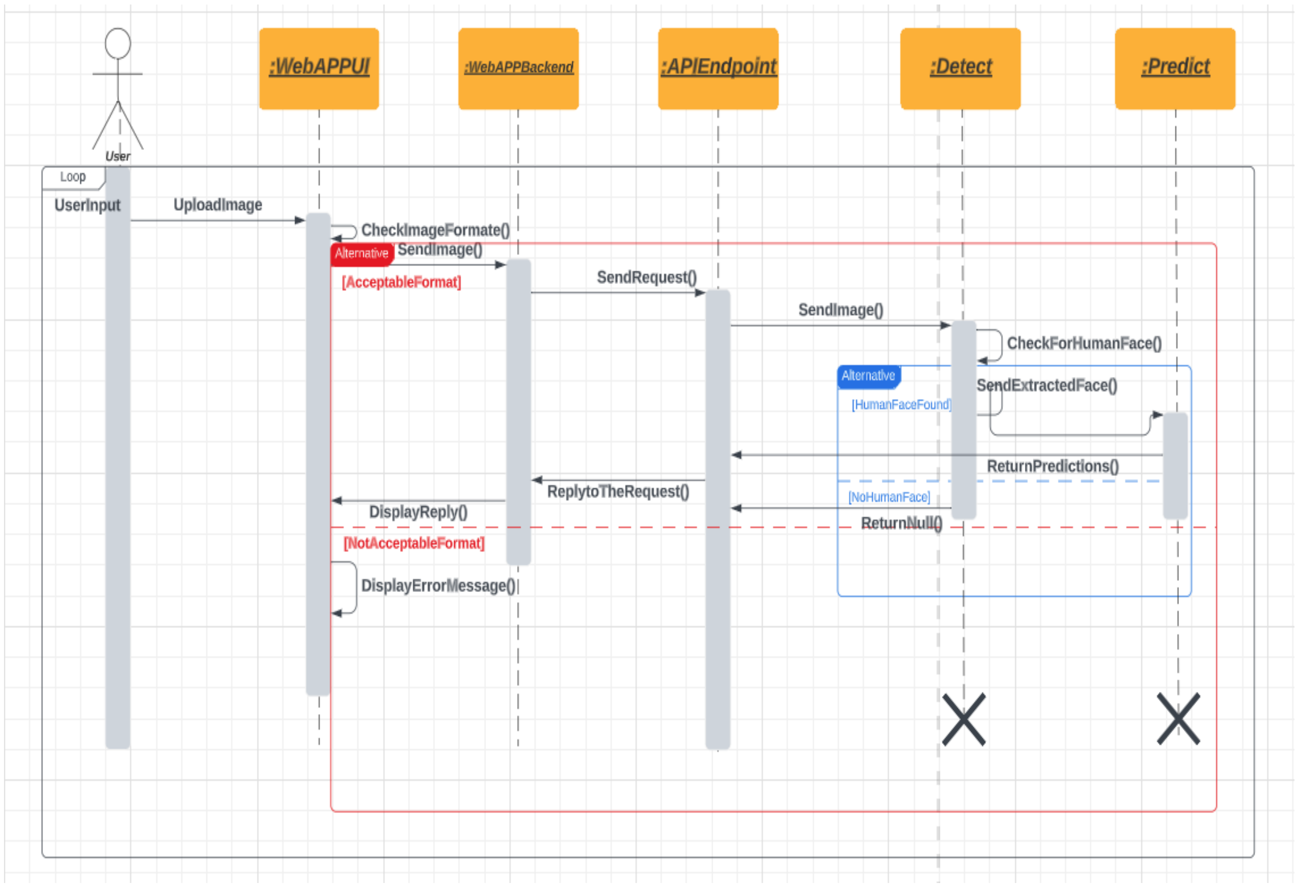


Figure 20: User Sequence Diagram.

1- The actor uploads an image to the web APP UI.
2- Web App UI checks the image format, if it’s not the required format, it displays an error message.
3- If the format is correct, the image is sent to the Web App Backend.
4- Web App Backend sends the image to the AI Endpoint as a request.
5- The API Endpoint method sends the received image from the request to the detector to detect human face.
6- If no human face detected a Null reply is sent back to the API Endpoint method which sent it to the web page backend, then to the web page UI where an error message is displayed.
7- If a human face is detected, the detector extracts it and sends it to the predictor.
8- Predictor analyzes and predicts human features and returns it to the API Endpoint method where it is sent back to the Web App backend as a reply.
9- When the Web App Receives the reply it sends it to Web App UI where it displayed.

2- API User:

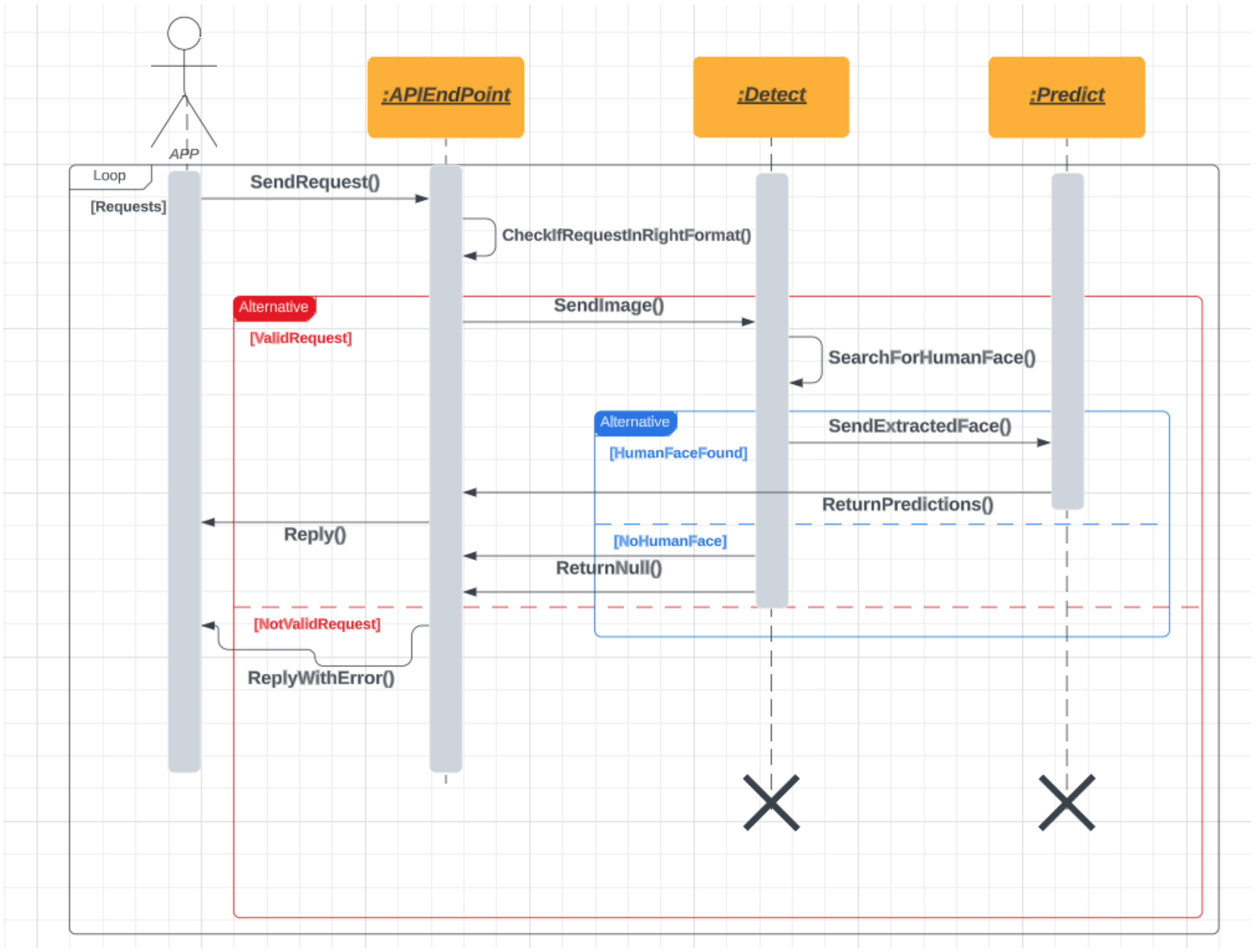


Figure 21: Developer Sequence Diagram.

1- The App sends requests to the API Endpoint.
2- The API Endpoint checks if the request and image are in valid format.
3- If the request or image is not in valid format, it replies with error or else it sends the image to the detector.
4- The detector checks for human face in the image, if no human face is found it returns Null to the API Endpoint method.
5- If there is a human face, it extracts it and sends it to the predictor.
6- The predictor analyzes and predicts human features and sends it to the API Endpoint method where it sent back as reply to the requester Application.

3.2.4 Database Diagram

No database Used in the project. The flask static folder is used for temporarily storing received image then delete it after sending the reply to the requester.

4- Implementation and Testing

The project consists of four Modules:

1- Face Detection
2- Age Prediction.
3- Gender Prediction.
4- Ethnicity Prediction.
5- Emotions Prediction.
6- The API Endpoint.
7- The Web Application.

This chapter describes each module Implementation, Testing and Results.

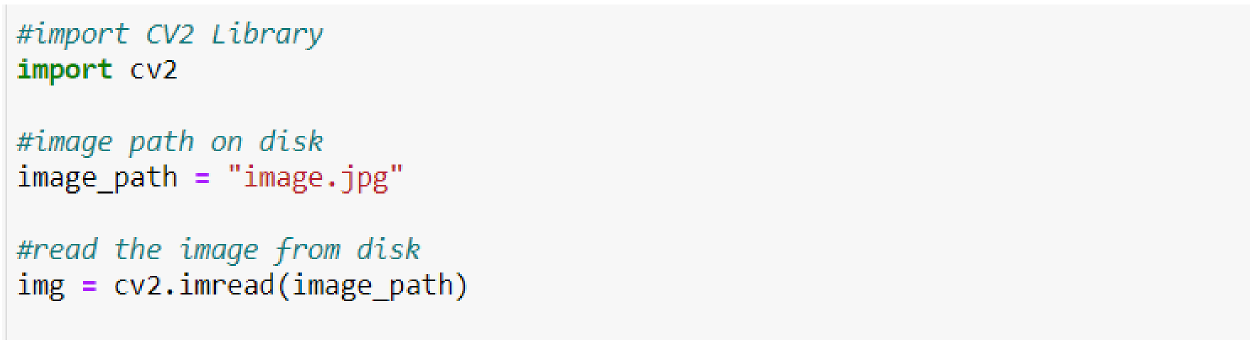
1- Face Detection

The objective of this module is to develop a model capable of detecting human faces and extracting them from an image.

The algorithm used is The Haar cascade algorithm [7].

Implementation:

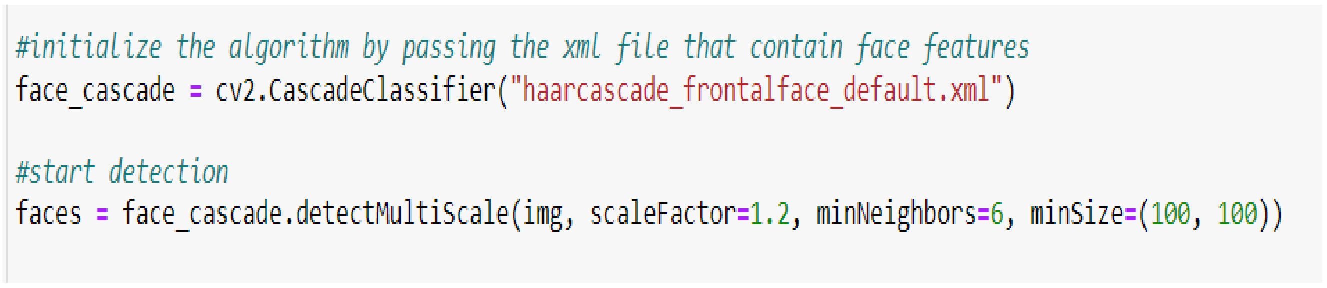
The Haar cascade algorithm is part of CV2 library.
First, we import the cv2 library and read the target image from the disk.



```python
#import CV2 Library
import cv2

#image path on disk
image_path = "image.jpg"

#read the image from disk
img = cv2.imread(image_path)
```

Figure 22: import csv2 and read image from disk using it.

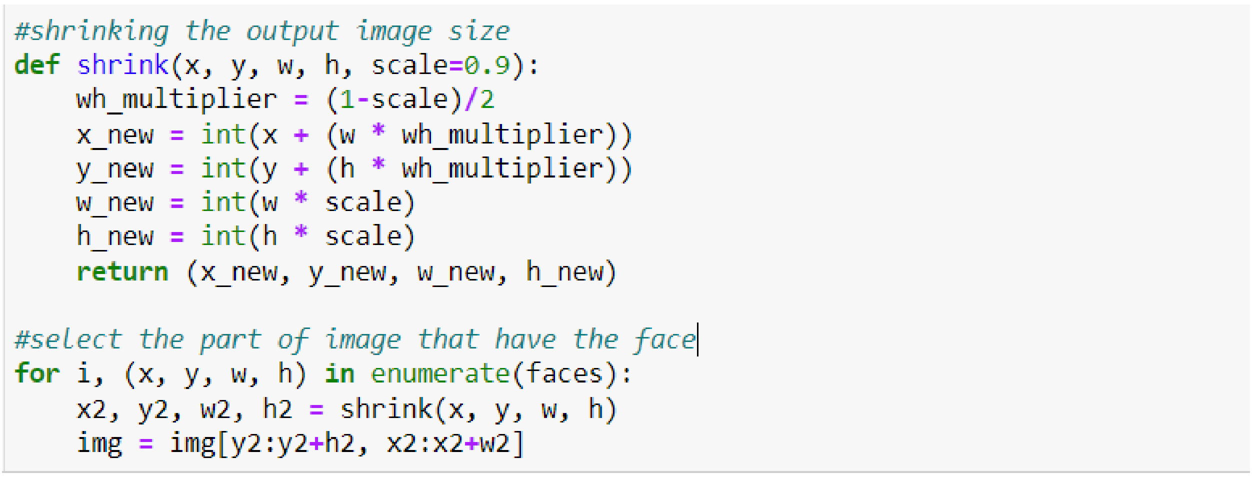
Then we initialize the algorithm by passing the xml file containing human face features.
The xml file can be found in cv2 library repository on GitHub.
And we pass the image and start detection.



```
#initialize the algorithm by passing the xml file that contain face features
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

#start detection
faces = face_cascade.detectMultiScale(img, scaleFactor=1.2, minNeighbors=6, minSize=(100, 100))
```

Figure 23: initializing Haar cascade Algorithm and apply it on image.

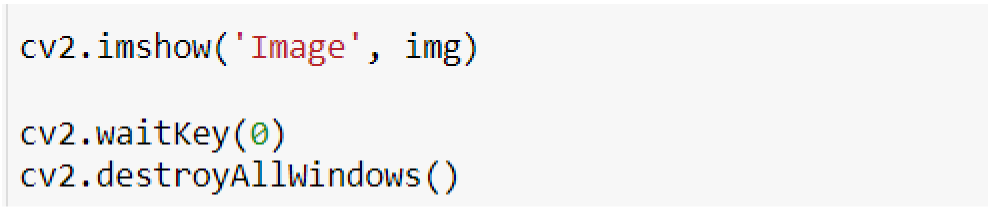
After detection we get the window size of the human face in the image.
We start cropping the human face from image and using the coordinates and shrink the output image to reduce computational power later when we use CNN.



```
#shrinking the output image size
def shrink(x, y, w, h, scale=0.9):
    wh_multiplier = (1-scale)/2
    x_new = int(x + (w * wh_multiplier))
    y_new = int(y + (h * wh_multiplier))
    w_new = int(w * scale)
    h_new = int(h * scale)
    return (x_new, y_new, w_new, h_new)

#select the part of image that have the face
for i, (x, y, w, h) in enumerate(faces):
    x2, y2, w2, h2 = shrink(x, y, w, h)
    img = img[y2:y2+h2, x2:x2+w2]
```

Figure 24: Cut the face from image using the given coordinates from Haar cascade algorithm.

After cutting the human face from the image we can then display or save it.



```
cv2.imshow('Image', img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 25: Displaying image using CV2.

2- Age Prediction

The objective of this module is to Detect human Age from Facial features.

Those features can be:

1- Facial Wrinkles and Texture: Wrinkles and texture patterns on the face can provide valuable cues for age estimation. Various texture analysis techniques, such as local binary patterns (LBP) or Gabor filters, can be employed to capture these features.

2- Facial Landmarks: Facial landmarks refer to specific points on the face, such as the corners of the eyes, the tip of the nose, or the corners of the mouth. The positions and distances between these landmarks can be used to extract geometric features that are indicative of age.

3- Facial Shape and Contour: Changes in facial shape and contour can be informative for age estimation. Shape analysis techniques, such as principal component analysis (PCA) or active shape models (ASM), can be employed to capture the variations in facial shape across different age groups.

4- Skin Color and Pigmentation: The distribution and variations of skin color and pigmentation can be useful for age estimation. Color-based features, such as histogram of oriented gradients (HOG) or color moments, can be extracted to capture these characteristics.

5- Hair Color and Style: Hair-related features, such as hair color, hair texture, or hairline, can also contribute to age estimation. These features can be extracted using image processing techniques or deep learning-based approaches.

6- Eye-related Features: Characteristics such as eye bags, crow's feet, or sagging eyelids can provide age-related information. These features can be detected using computer vision techniques or deep learning models.

7- Contextual Features: In addition to facial features, contextual information can also be valuable for age estimation. For example, clothing style, accessories, or background environment can be considered as supplementary cues.

The choice of these features depends on the approaches and algorithms used.

By utilizing the power of CNN models, we create a CNN model to predict the human age.

One of the challenges faced during this module is deciding wither to use a regression model or classification model.

Detecting the exact age of humans can be challenging due to the large similarities between age groups and abnormalities like having a younger or older appearance despite the real age being older or younger. Regression models also tend to need more data to train and more computational power.

Due to the limited available computational power and datasets and the challenges of predicting the exact age with minimum error a classification model has been chosen.

Human age groups can be split to 7 classes based on the similarity in facial features.
Those classes are:

1- Ages from 1 to 2
2- Ages from 3 to 9
3- Ages from 10 to 20
4- Ages from 21 to 27
5- Ages from 28 to 45
6- Ages from 46 to 65
7- Ages from 66 to 116

Datasets used for training are “Facial Age” and “UTK face”.
A total of 33,486 images which is critically low for tasks like age classification.

Samples of the Datasets:

Figure 26 Samples From "Facial Age" and "UTK Face" Datasets.

Due to the low number of images in the Datasets and to reduce overfitting/underfitting during training, both Dataset and image Augmentation Techniques were used.

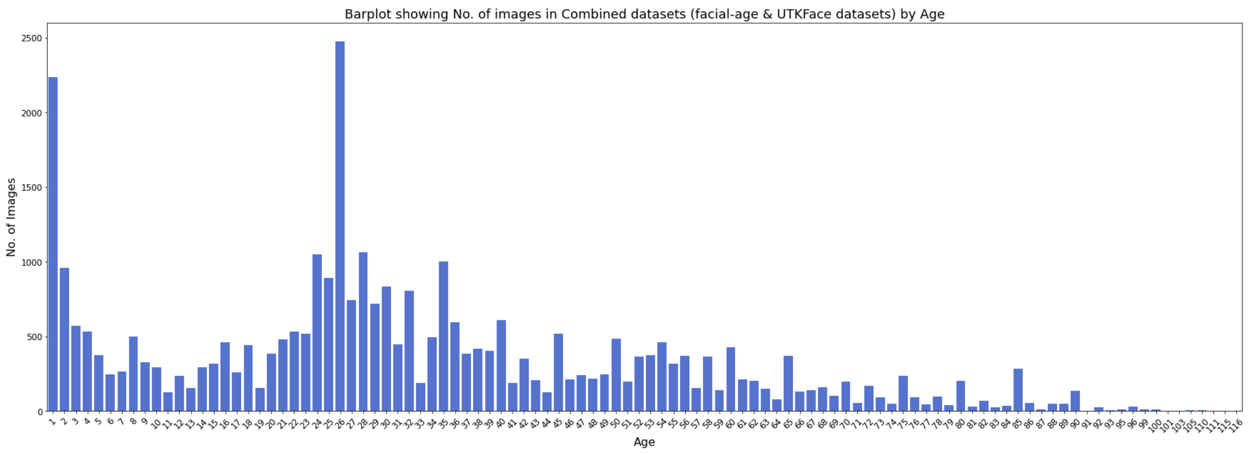


Figure 27 Distribution of Age Dataset after Augmentation.

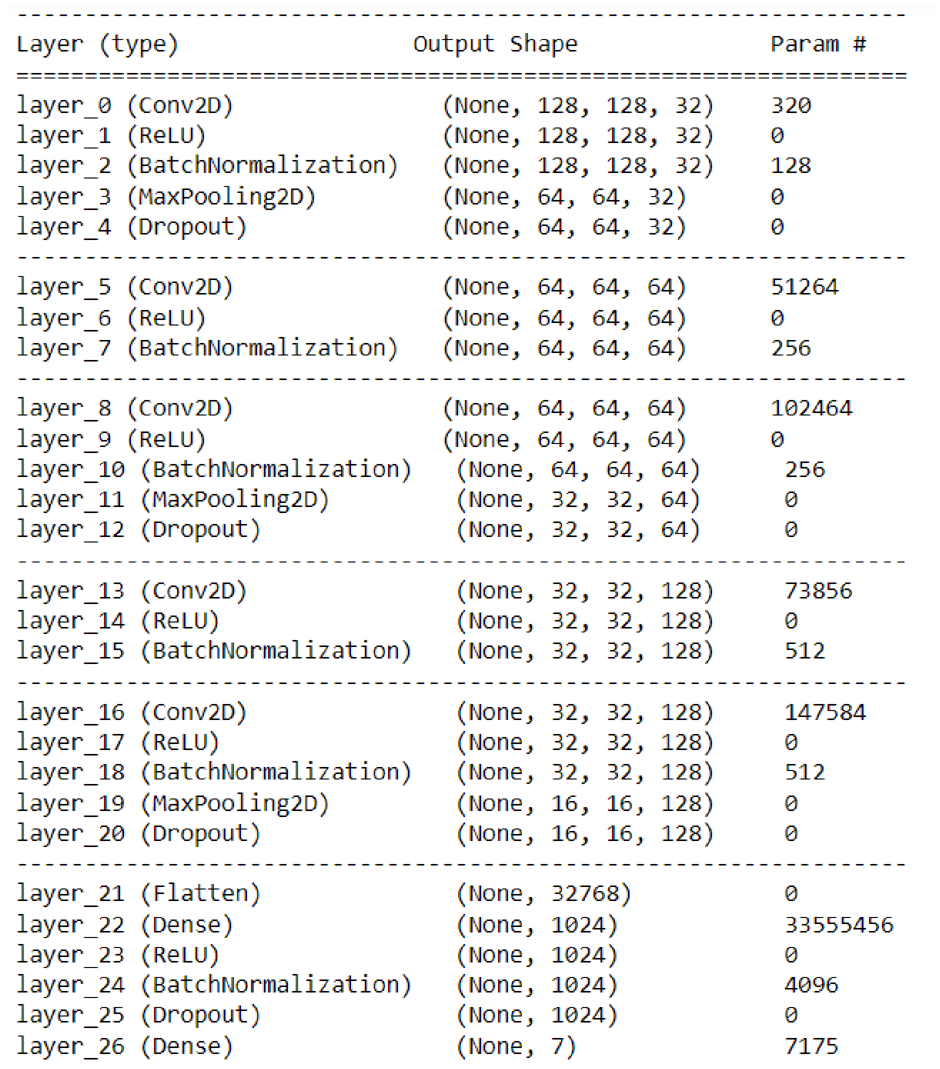
By applying those techniques, Dataset expanded to 234,400 images.
And the image been cleaned of images that don’t have human face by using Haar Cascade Algorithm

CNN Architecture:

5 Convolution Block were chosen for feature extraction, each block consists of Convolution layer with RelU activation Function, Batch Normalization Layer, Max pooling layer and Dropout layer.

Classification Block with Flaten, Dense, RelU, Batch Normalization, dropout and SoftMax layer with 7 units were chosen.

Batch Normalization, Max Pooling and Dropout Layers helped with reducing overfitting in each block.

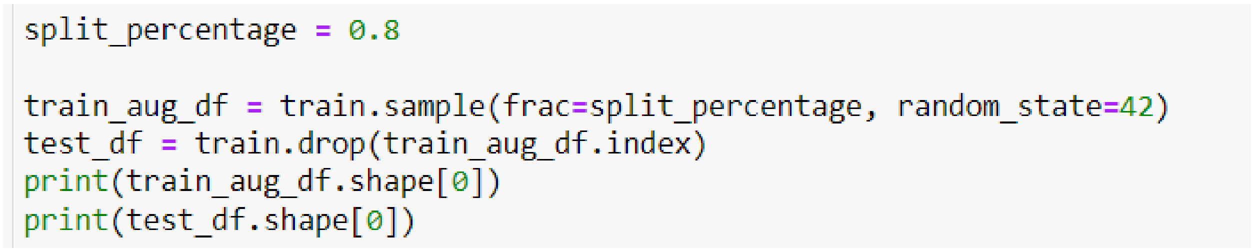


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_0 (Conv2D)               (None, 128, 128, 32)    320
layer_1 (ReLU)                 (None, 128, 128, 32)    0
layer_2 (BatchNormalization)   (None, 128, 128, 32)    128
layer_3 (MaxPooling2D)         (None, 64, 64, 32)      0
layer_4 (Dropout)              (None, 64, 64, 32)      0
_________________________________________________________________
layer_5 (Conv2D)               (None, 64, 64, 64)      51264
layer_6 (ReLU)                 (None, 64, 64, 64)      0
layer_7 (BatchNormalization)   (None, 64, 64, 64)      256
_________________________________________________________________
layer_8 (Conv2D)               (None, 64, 64, 64)      102464
layer_9 (ReLU)                 (None, 64, 64, 64)      0
layer_10 (BatchNormalization)   (None, 64, 64, 64)      256
layer_11 (MaxPooling2D)         (None, 32, 32, 64)      0
layer_12 (Dropout)              (None, 32, 32, 64)      0
_________________________________________________________________
layer_13 (Conv2D)               (None, 32, 32, 128)     73856
layer_14 (ReLU)                 (None, 32, 32, 128)     0
layer_15 (BatchNormalization)   (None, 32, 32, 128)     512
_________________________________________________________________
layer_16 (Conv2D)               (None, 32, 32, 128)     147584
layer_17 (ReLU)                 (None, 32, 32, 128)     0
layer_18 (BatchNormalization)   (None, 32, 32, 128)     512
layer_19 (MaxPooling2D)         (None, 16, 16, 128)     0
layer_20 (Dropout)              (None, 16, 16, 128)     0
_________________________________________________________________
layer_21 (Flatten)              (None, 32768)           0
layer_22 (Dense)                (None, 1024)            33555456
layer_23 (ReLU)                 (None, 1024)            0
layer_24 (BatchNormalization)   (None, 1024)            4096
layer_25 (Dropout)              (None, 1024)            0
layer_26 (Dense)                (None, 7)               7175
```

Figure 28 Age Model CNN Architecture.

Implementation:

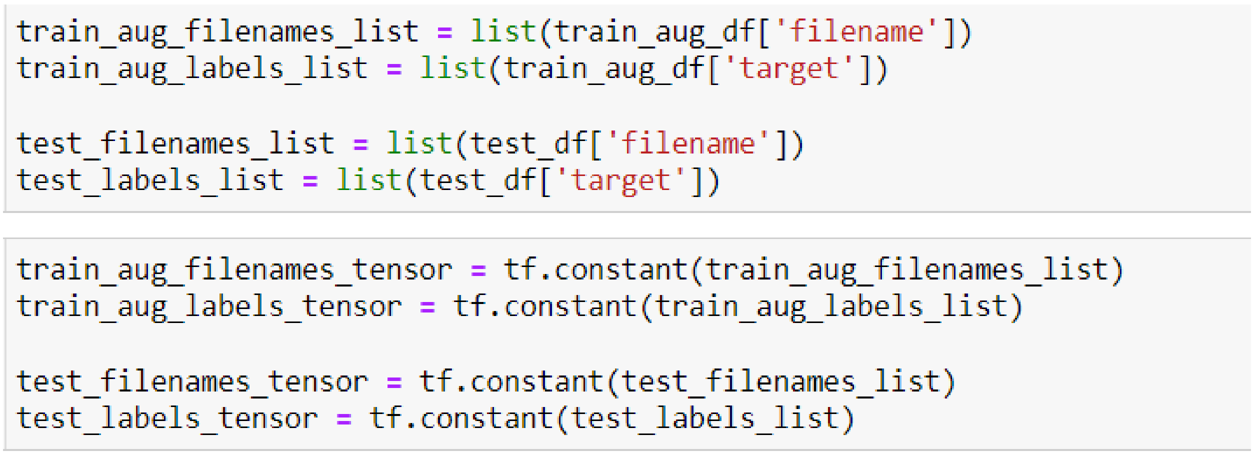
TensorFlow was used in the implementation of this model.
A csv containing all images labels and path created.
Dataset split to 80% train and 20% validation and testing.



```
split_percentage = 0.8

train_aug_df = train.sample(frac=split_percentage, random_state=42)
test_df = train.drop(train_aug_df.index)
print(train_aug_df.shape[0])
print(test_df.shape[0])
```

Figure 29: Split the dataset in the age model.

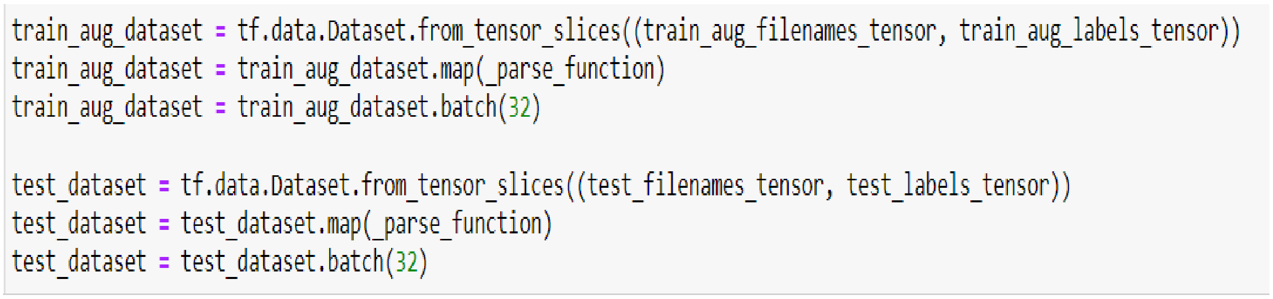
A tensor data loader was created to load the dataset as batches from the csv file.
A batch of 32 images was chosen due to the limited memory.



```
train_aug_filenames_list = list(train_aug_df['filename'])
train_aug_labels_list = list(train_aug_df['target'])

test_filenames_list = list(test_df['filename'])
test_labels_list = list(test_df['target'])
```

```
train_aug_filenames_tensor = tf.constant(train_aug_filenames_list)
train_aug_labels_tensor = tf.constant(train_aug_labels_list)

test_filenames_tensor = tf.constant(test_filenames_list)
test_labels_tensor = tf.constant(test_labels_list)
```

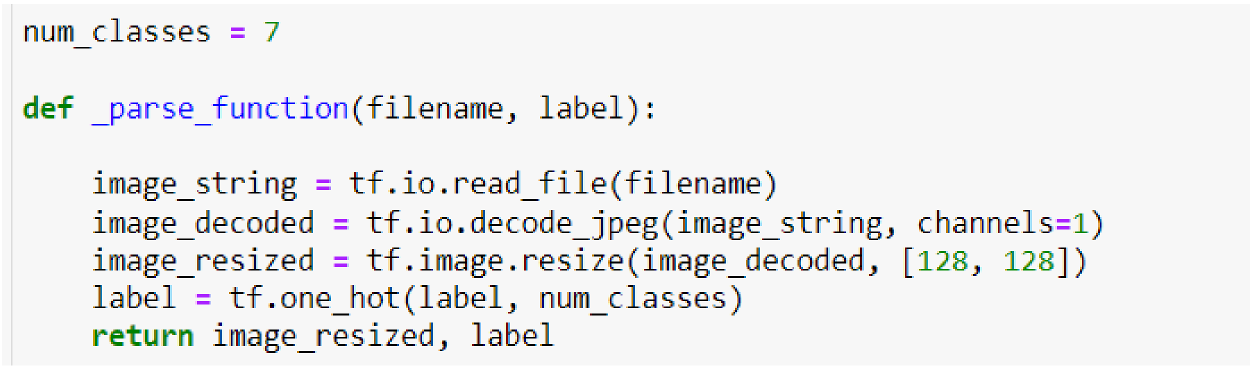
Figure 30: creating tensors from age dataset images and labels.



```
train_aug_dataset = tf.data.Dataset.from_tensor_slices((train_aug_filenames_tensor, train_aug_labels_tensor))
train_aug_dataset = train_aug_dataset.map(_parse_function)
train_aug_dataset = train_aug_dataset.batch(32)

test_dataset = tf.data.Dataset.from_tensor_slices((test_filenames_tensor, test_labels_tensor))
test_dataset = test_dataset.map(_parse_function)
test_dataset = test_dataset.batch(32)
```

Figure 31:Apply preprocessing on tensors and specify the batch size to the data loader in age model.

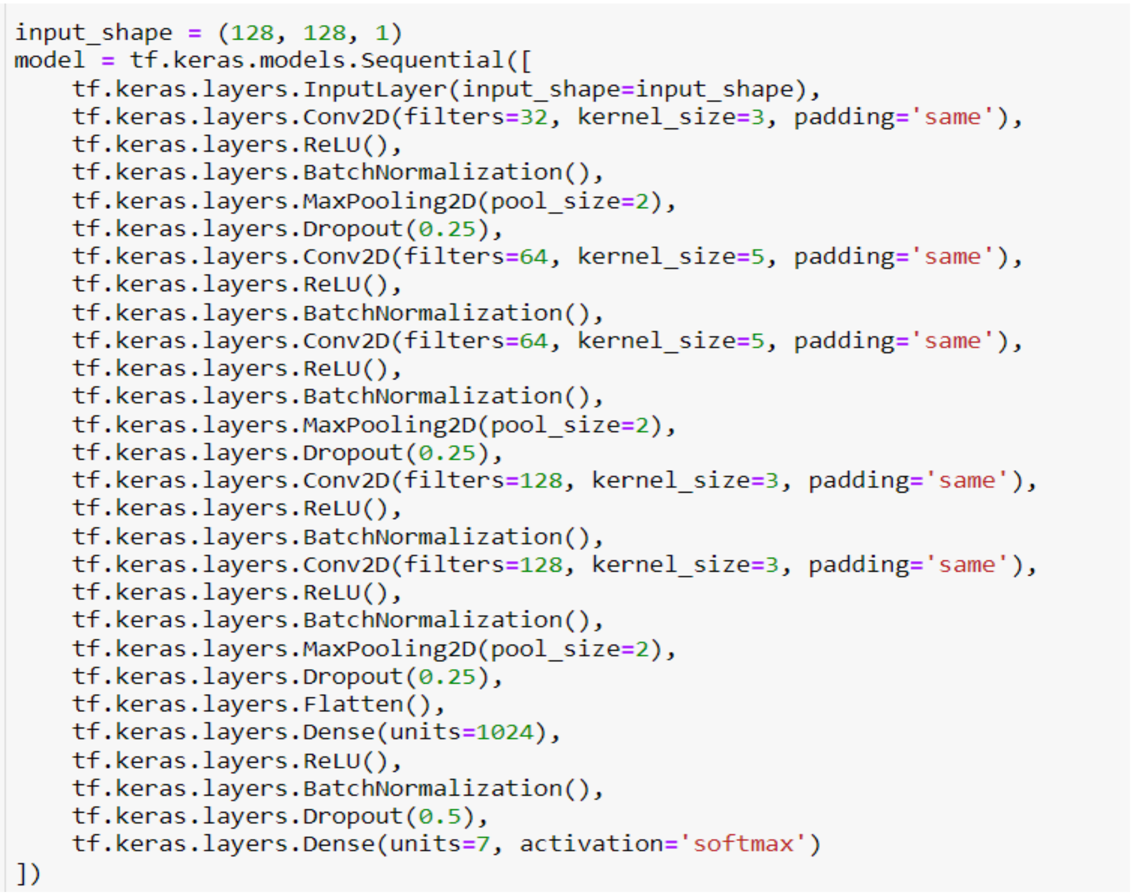
Preprocessing consists of transforming images to grayscale images with one channel since the RGB channels are not needed in this model and to reduce competition power.
All images resized to 128x128 images.
Labels were encoded using one hot encoding.



```
num_classes = 7

def _parse_function(filename, label):

    image_string = tf.io.read_file(filename)
    image_decoded = tf.io.decode_jpeg(image_string, channels=1)
    image_resized = tf.image.resize(image_decoded, [128, 128])
    label = tf.one_hot(label, num_classes)
    return image_resized, label
```

Figure 32: Age Model Preprocessing Function.

The CNN architecture implemented using TensorFlow sequential method.



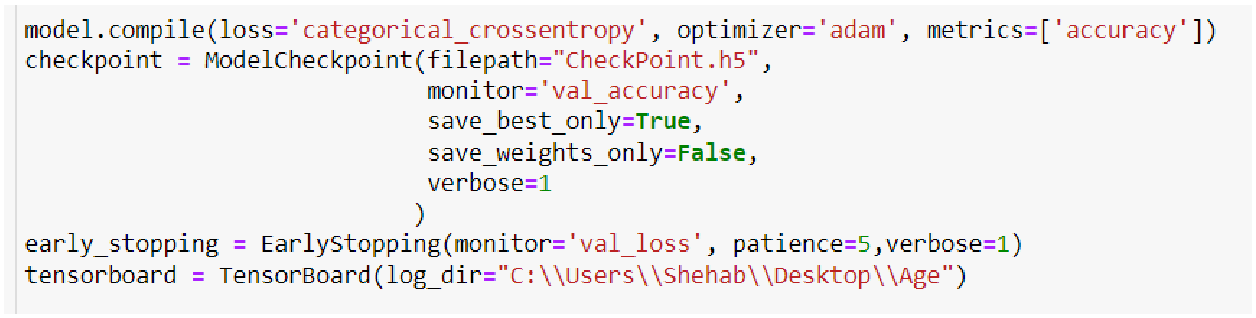
```
input_shape = (128, 128, 1)
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(input_shape=input_shape),
    tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(filters=64, kernel_size=5, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=64, kernel_size=5, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=1024),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=7, activation='softmax')
])
```

Figure 33: Age Model Architecture Sequential Implementation.

Total number of parameters were 33,943,879 with 33,940,99 trainable parameters.

Adam was used as optimizer instead of SGD because of its effectiveness with 0.001 learning rate.
Categorical cross entropy was chosen as loss function.
Early stopping, checkpoints and tensor board classes chosen as callback after each epoch to monitor the validation loss and stop the model if no improvement in validation loss happened after number of epochs call patience which set to 5 epochs and save the best model in terms of low validation loss using the check points.

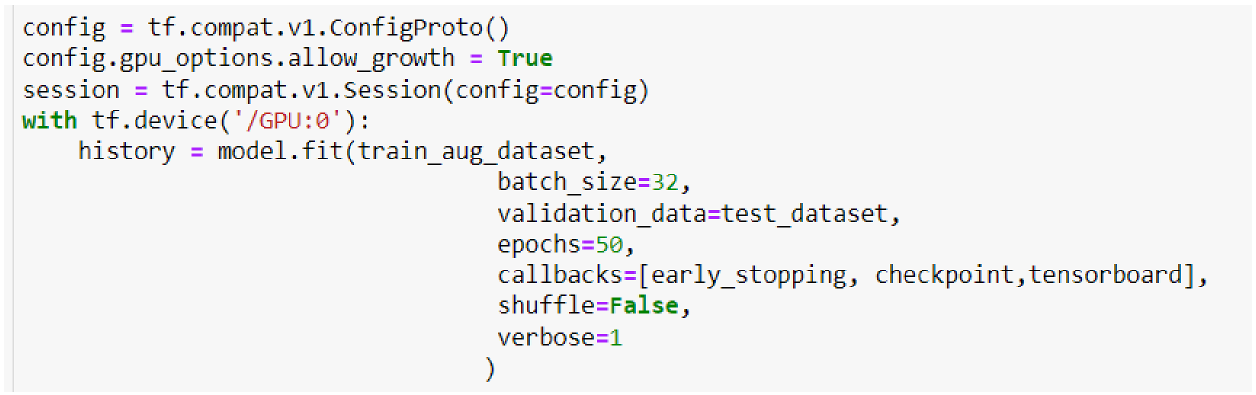
The default learning rate of 0.001 of Adam Optimizer chosen.



```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
checkpoint = ModelCheckpoint(filepath="CheckPoint.h5",
                             monitor='val_accuracy',
                             save_best_only=True,
                             save_weights_only=False,
                             verbose=1
                            )
early_stopping = EarlyStopping(monitor='val_loss', patience=5,verbose=1)
tensorboard = TensorBoard(log_dir="C:\\Users\\Shehab\\Desktop\\Age")
```

Figure 34: initialize the compile method and callbacks for age model.

Model fitted with 100 epochs.



```
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
with tf.device('/GPU:0'):
    history = model.fit(train_aug_dataset,
                                  batch_size=32,
                                  validation_data=test_dataset,
                                  epochs=50,
                                  callbacks=[early_stopping, checkpoint,tensorboard],
                                  shuffle=False,
                                  verbose=1
                                 )
```

Figure 35: Fitting the Age Model.

Results:

The early stopping callback triggered after 14 epochs.

The model showed a good convergence with around 84% train accuracy, 0.4176 train loss, 0.7554 validation loss and 72% validation accuracy.

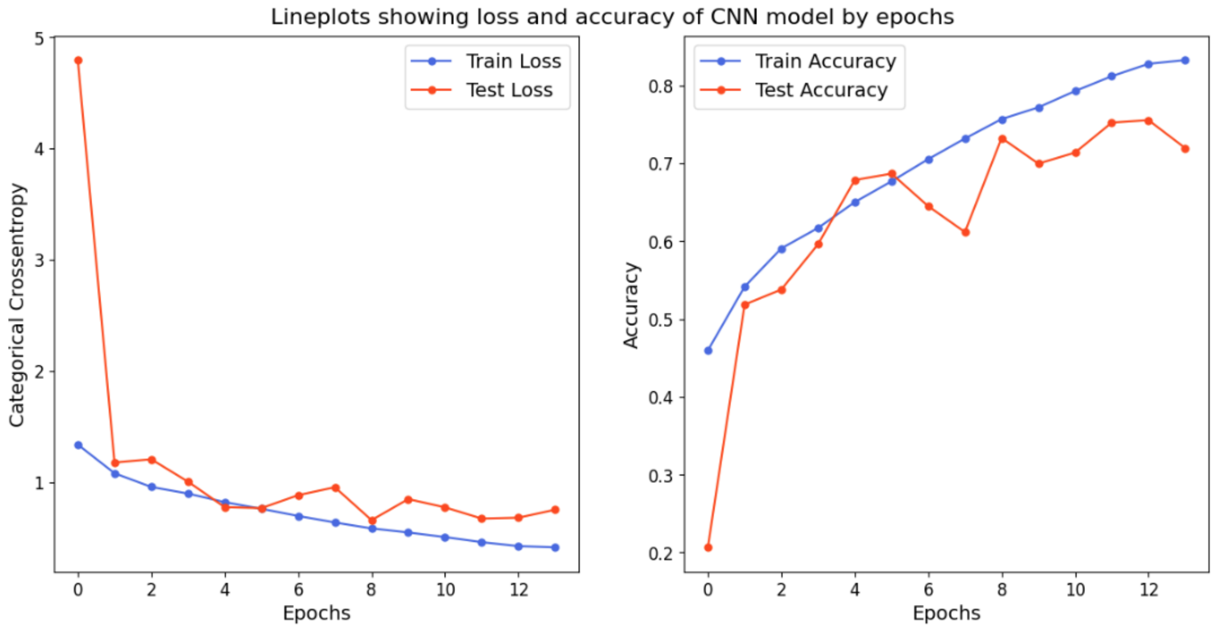
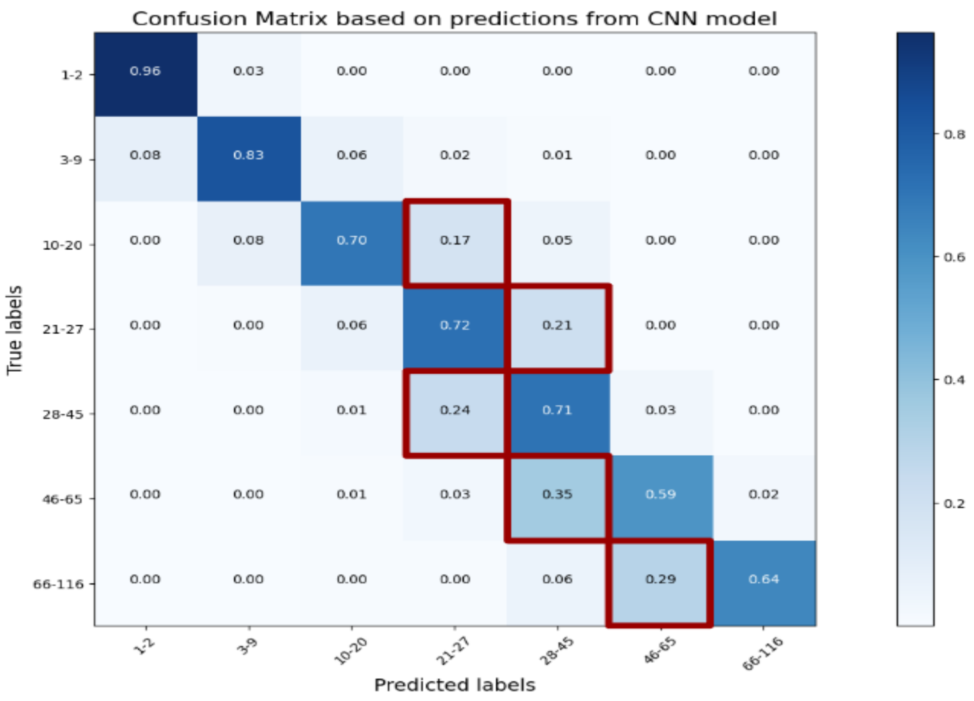


Figure 36 Age Model Convergence Plot

Model's Confusion Matrix showed some confusion in ages near the border of each class due to the increased similarities.



True \ Predicted	1-2	3-9	10-20	21-27	28-45	46-65	66-116
1-2	0.96	0.03	0.00	0.00	0.00	0.00	0.00
3-9	0.08	0.83	0.06	0.02	0.01	0.00	0.00
10-20	0.00	0.08	0.70	0.17	0.05	0.00	0.00
21-27	0.00	0.00	0.06	0.72	0.21	0.00	0.00
28-45	0.00	0.00	0.01	0.24	0.71	0.03	0.00
46-65	0.00	0.00	0.01	0.03	0.35	0.59	0.02
66-116	0.00	0.00	0.00	0.00	0.06	0.29	0.64

Figure 37 Age Model Confusion Matrix

The feature map for this model shows a good capturing of the model features.

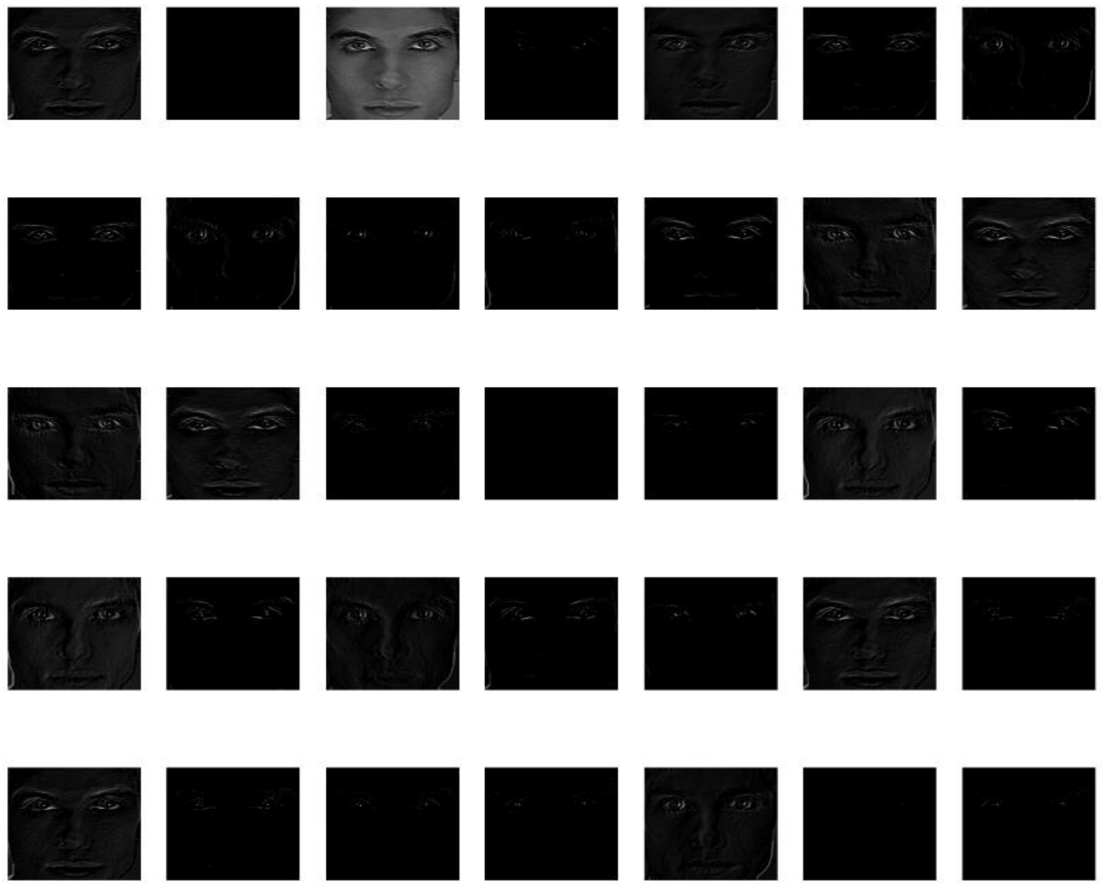
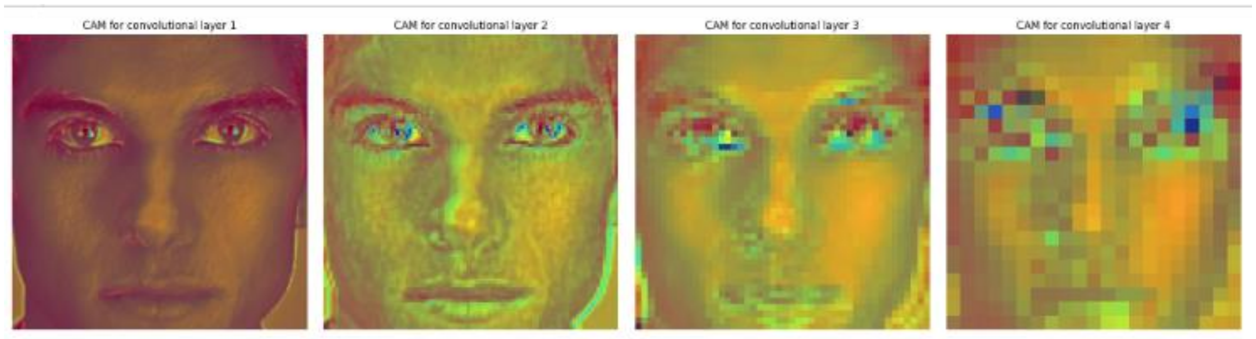


Figure 38:Age Model Feature Map.

The heat map for this model:


CAM for convolutional layer 1
CAM for convolutional layer 2
CAM for convolutional layer 3
CAM for convolutional layer 4
Figure 39 Age Model Heat Map

The Model Achieved 71% accuracy and 0.75 loss on the testing dataset.

3- Gender Prediction

The objective of this module is to develop a model capable of Predicting Human Gender from Facial images.

By utilizing the power of CNN models, we create a CNN model to predict the human gender.

Challenges of this model is the similarities between male and female genders in many features and the existence of male individuals with some female features like long hear and round face and the existence of female individuals with some male features like short hair and straight face etc.

Those challenges can be partially solved using a bigger dataset with a variety of races and features, but it is still hard to classify some individuals without human judgement.

The datasets used in the training of this model are UTK-Face dataset and B3FD dataset.

B3FD is a cleaned version of Both IMDB and WIKI datasets consisting of famous celebrities around the world and their age and gender.

Haarcascade algorithm was applied to all images in both datasets to remove any noisy or non-human images.

After preprocessing Both datasets combined consist of 366,603 images.

The distribution of the images in both genders are nearly equal so no class balancing was needed.

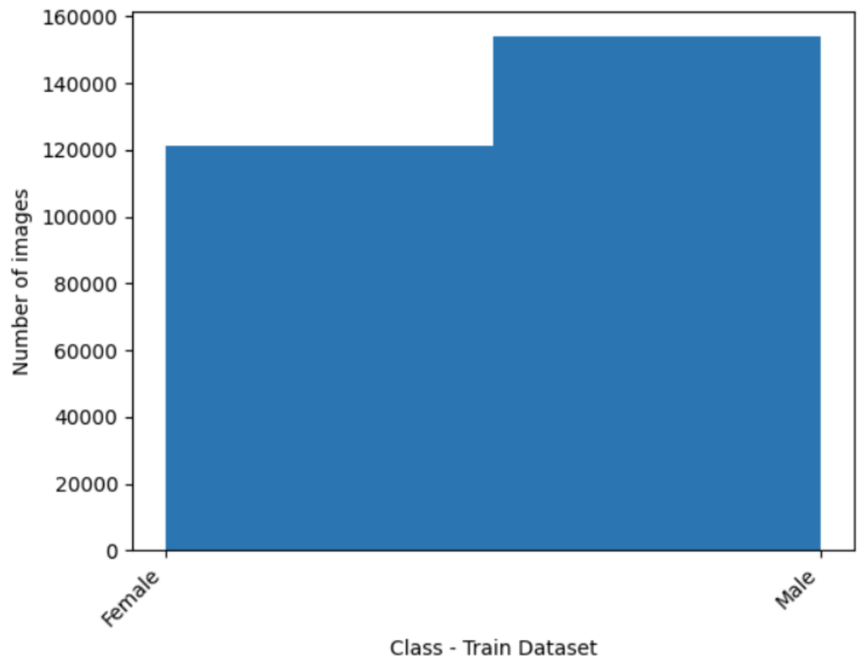


Figure 40: Gender Model Train Dataset Class Distribution.

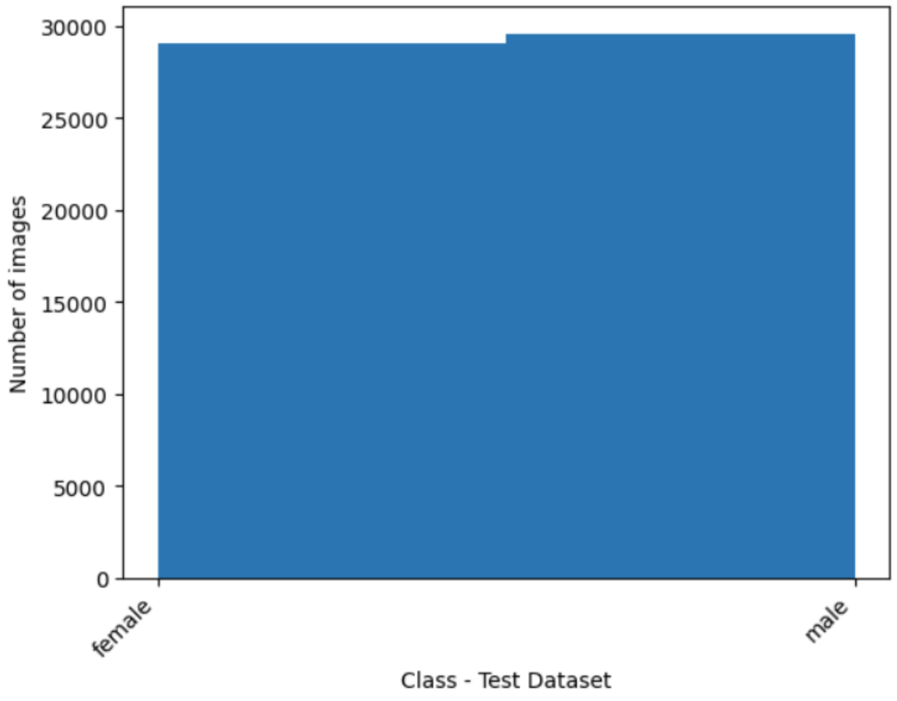


Figure 41: Gender Model Test and validate Dataset Class Distribution.

A sample from the combined datasets:

Figure 42: Samples of Gender Model Datasets.

To reduce the overfitting image augmentation applied during the training on the training dataset using TensorFlow.

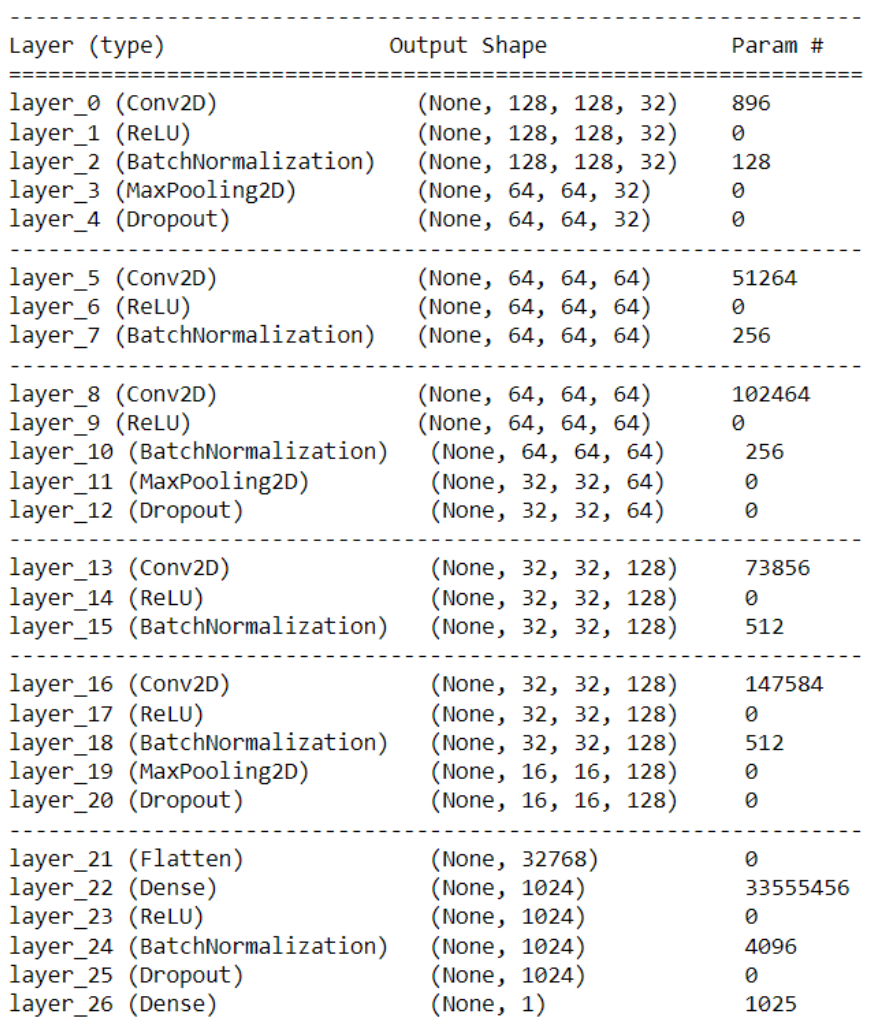
Training done using RGB and Grayscale and the RGB version gave slightly better results, so it was chosen during this model.

CNN architecture:

5 convolution blocks for feature extraction were used.
Each convolution block consists of a convolution layer, relU activation layer, Batch normalization layer, max pooling, and Dropout layers.

Not using Max pooling or dropout in blocks 2 and 4 gave a better result to prevent underfitting caused by the overuse of overfitting techniques.

A classification block with 1-unit sigmoid activation function was used for binary classification since we have only 2 classes.



```
-----------------------------------------------------------------
Layer (type)                 Output Shape              Param #
=================================================================
layer_0 (Conv2D)              (None, 128, 128, 32)    896
layer_1 (ReLU)                (None, 128, 128, 32)    0
layer_2 (BatchNormalization)  (None, 128, 128, 32)    128
layer_3 (MaxPooling2D)        (None, 64, 64, 32)      0
layer_4 (Dropout)             (None, 64, 64, 32)      0
-----------------------------------------------------------------
layer_5 (Conv2D)              (None, 64, 64, 64)      51264
layer_6 (ReLU)                (None, 64, 64, 64)      0
layer_7 (BatchNormalization)  (None, 64, 64, 64)      256
-----------------------------------------------------------------
layer_8 (Conv2D)              (None, 64, 64, 64)      102464
layer_9 (ReLU)                (None, 64, 64, 64)      0
layer_10 (BatchNormalization)  (None, 64, 64, 64)      256
layer_11 (MaxPooling2D)        (None, 32, 32, 64)      0
layer_12 (Dropout)             (None, 32, 32, 64)      0
-----------------------------------------------------------------
layer_13 (Conv2D)              (None, 32, 32, 128)     73856
layer_14 (ReLU)                (None, 32, 32, 128)     0
layer_15 (BatchNormalization)  (None, 32, 32, 128)     512
-----------------------------------------------------------------
layer_16 (Conv2D)              (None, 32, 32, 128)     147584
layer_17 (ReLU)                (None, 32, 32, 128)     0
layer_18 (BatchNormalization)  (None, 32, 32, 128)     512
layer_19 (MaxPooling2D)        (None, 16, 16, 128)     0
layer_20 (Dropout)             (None, 16, 16, 128)     0
-----------------------------------------------------------------
layer_21 (Flatten)             (None, 32768)           0
layer_22 (Dense)               (None, 1024)            33555456
layer_23 (ReLU)                (None, 1024)            0
layer_24 (BatchNormalization)  (None, 1024)            4096
layer_25 (Dropout)             (None, 1024)            0
layer_26 (Dense)               (None, 1)               1025
```

Figure 43: Gender Model CNN Architecture.

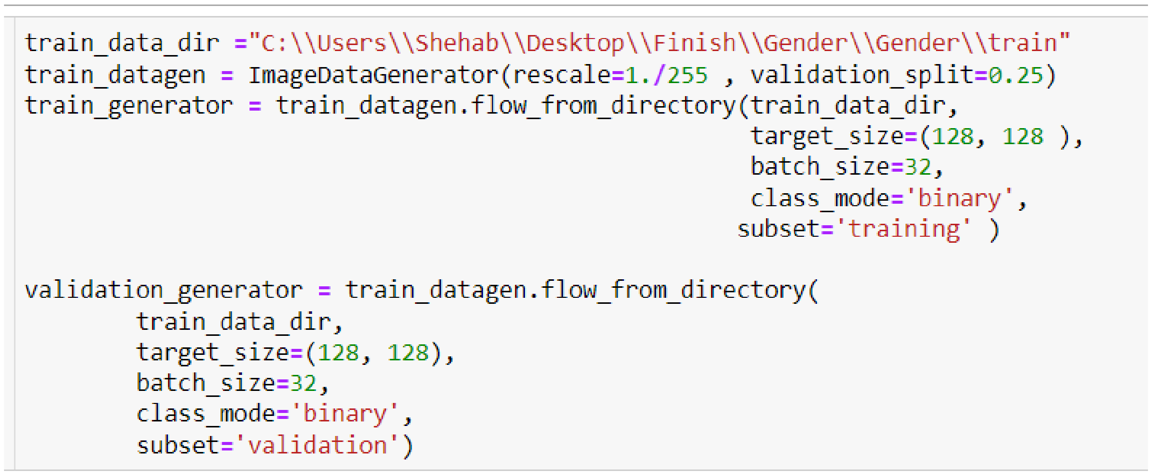
Implementation:

TensorFlow was used in this model.

TensorFlow image Data Generator used to apply normalization and augmentation on the images.

Dataset split to 75% for training and 25% for testing and validation.

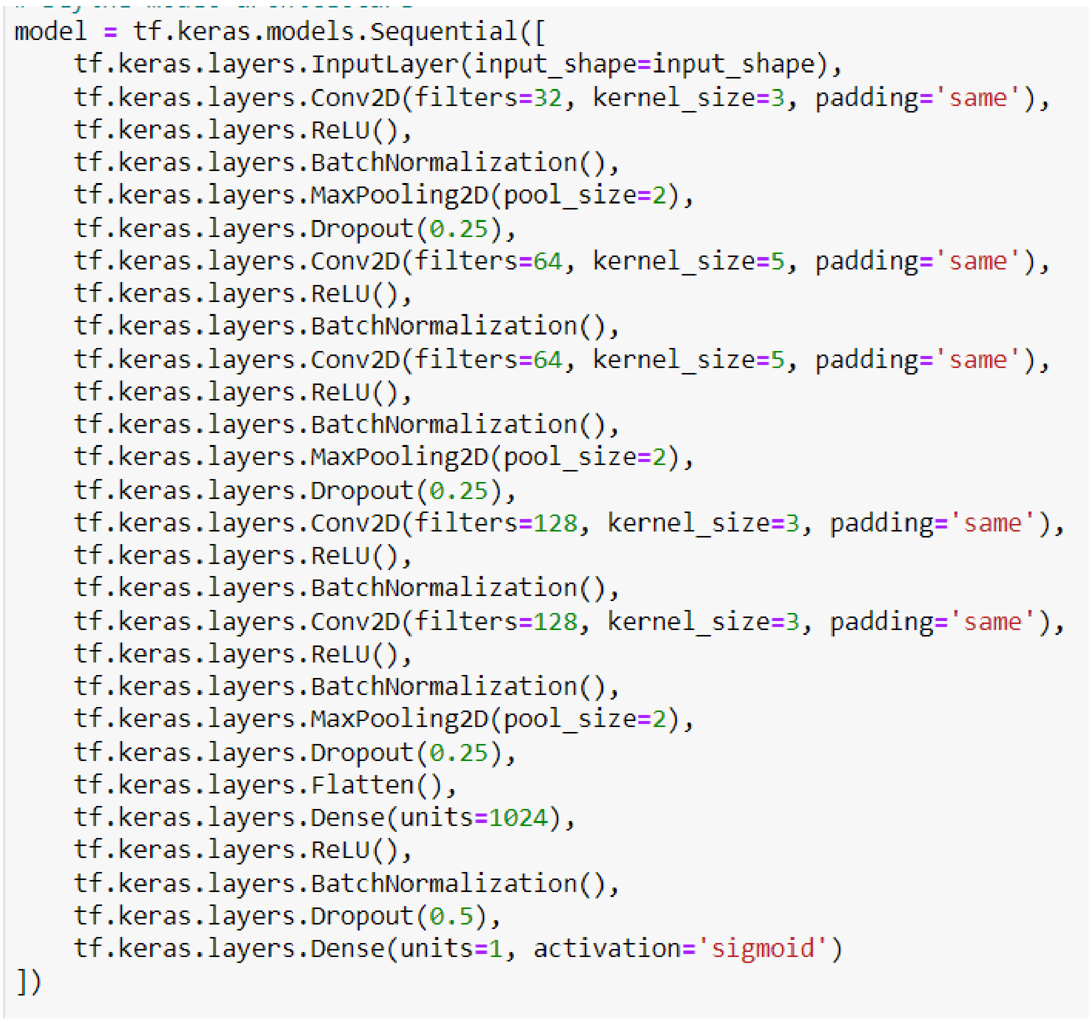
All images have been resized to 128x128x3.
A Batch size of 32 images was chosen.
The number of epochs chosen to be 100.



```
train_data_dir ="C:\\Users\\Shehab\\Desktop\\Finish\\Gender\\Gender\\train"
train_datagen = ImageDataGenerator(rescale=1./255 , validation_split=0.25)
train_generator = train_datagen.flow_from_directory(train_data_dir,
                                                    target_size=(128, 128 ),
                                                    batch_size=32,
                                                    class_mode='binary',
                                                   subset='training' )

validation_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(128, 128),
        batch_size=32,
        class_mode='binary',
        subset='validation')
```

Figure 44: Image Data Generator for Gender Model.

Model defined using TensorFlow sequential method.



```
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(input_shape=input_shape),
    tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(filters=64, kernel_size=5, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=64, kernel_size=5, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same'),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=1024),
    tf.keras.layers.ReLU(),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

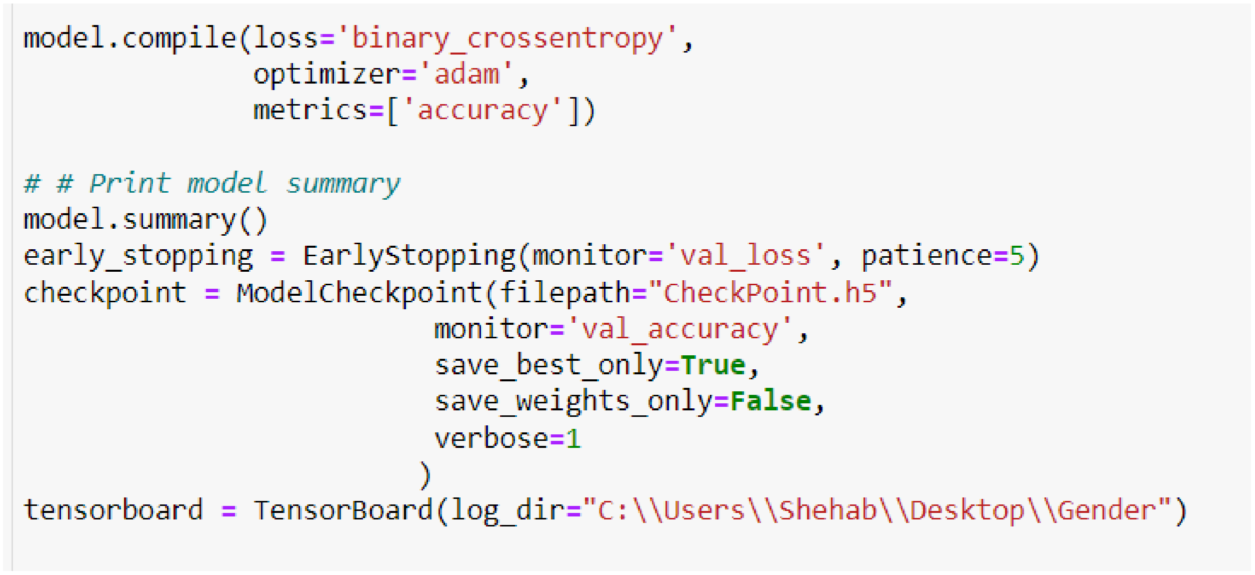
Figure 45:Gender Model Architecture in the Sequential Method.

A binary cross entropy loss chosen since its binary classification model.

Both Adam and SGD optimizers were tried and Adam Gave better convergence, so it was chosen.
Metrics is accuracy.

Callbacks initialized with Early stopping callback with 5 patience, model check points with 1 verbose to save the best model during training and tensor board callback to monitor the model performance during training.

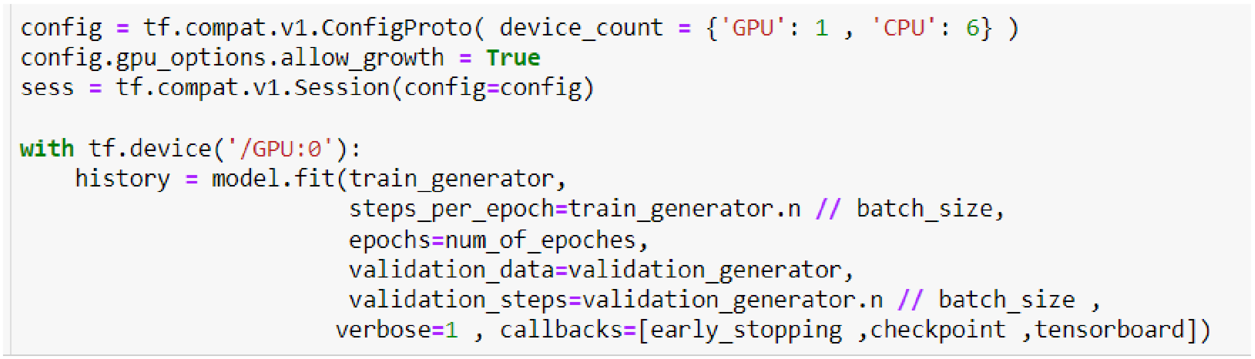
The default learning rate of 0.001 of Adam Optimizer chosen.



```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# # Print model summary
model.summary()
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
checkpoint = ModelCheckpoint(filepath="CheckPoint.h5",
                             monitor='val_accuracy',
                             save_best_only=True,
                             save_weights_only=False,
                             verbose=1
                            )
tensorboard = TensorBoard(log_dir="C:\\Users\\Shehab\\Desktop\\Gender")
```

Figure 46: Gender Model Compile and Callback Functions.

Model fitting:



```
config = tf.compat.v1.ConfigProto( device_count = {'GPU': 1 , 'CPU': 6} )
config.gpu_options.allow_growth = True
sess = tf.compat.v1.Session(config=config)

with tf.device('/GPU:0'):
    history = model.fit(train_generator,
                        steps_per_epoch=train_generator.n // batch_size,
                        epochs=num_of_epoches,
                        validation_data=validation_generator,
                        validation_steps=validation_generator.n // batch_size ,
                       verbose=1 , callbacks=[early_stopping ,checkpoint ,tensorboard])
```

Figure 47: Fitting the Gender Model.

Results:

Model early stopping callback triggered after 20 epochs.

The model achieved a training accuracy of 95%, training loss of 0.1285, validation accuracy of 89% and validation loss of 0.2881.

The model achieved a good convergence with some spikes in the validation which controlled with checkpoints to save the lowest point.

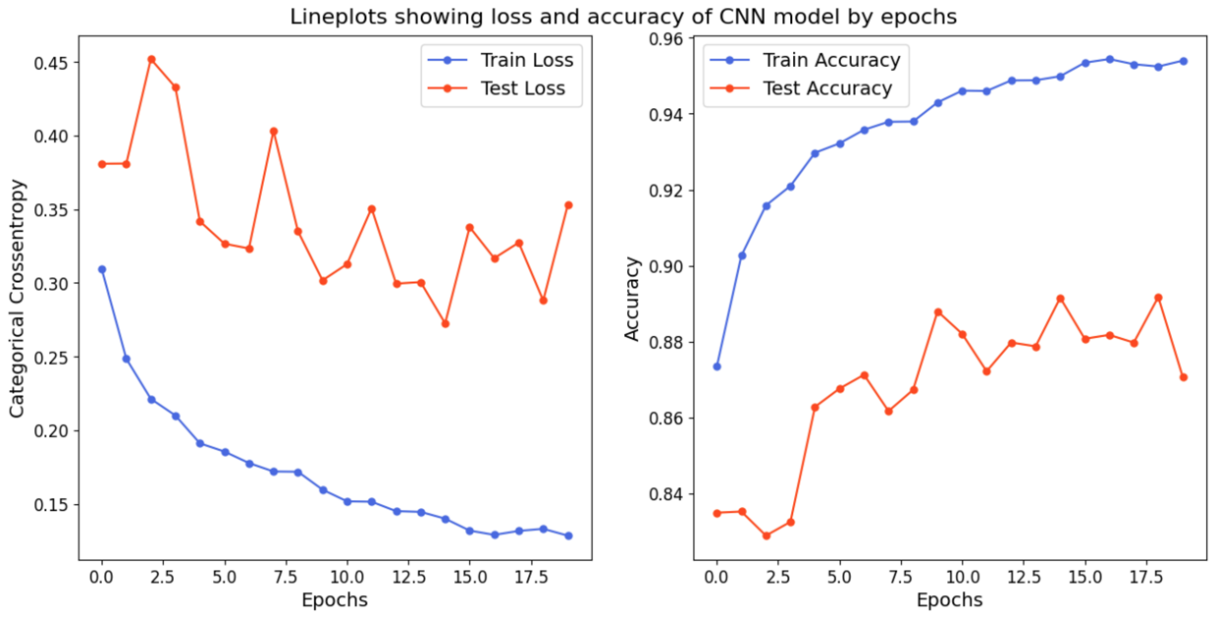
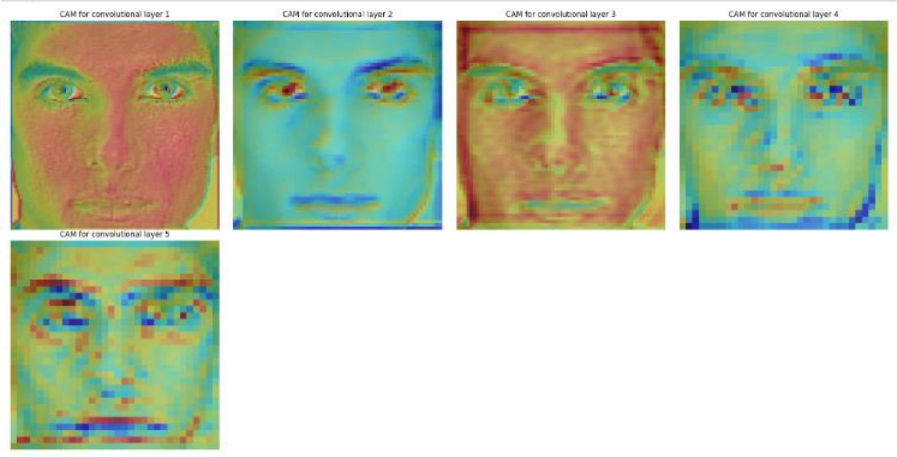


Figure 48: Gender Model Convergence Plot.

Heat Map shows a good capturing of features in the convolution blocks.



CAM for convolutional layer 1
CAM for convolutional layer 2
CAM for convolutional layer 3
CAM for convolutional layer 4
CAM for convolutional layer 5

Figure 49: Gender Model Heat Map.

Feature Map:

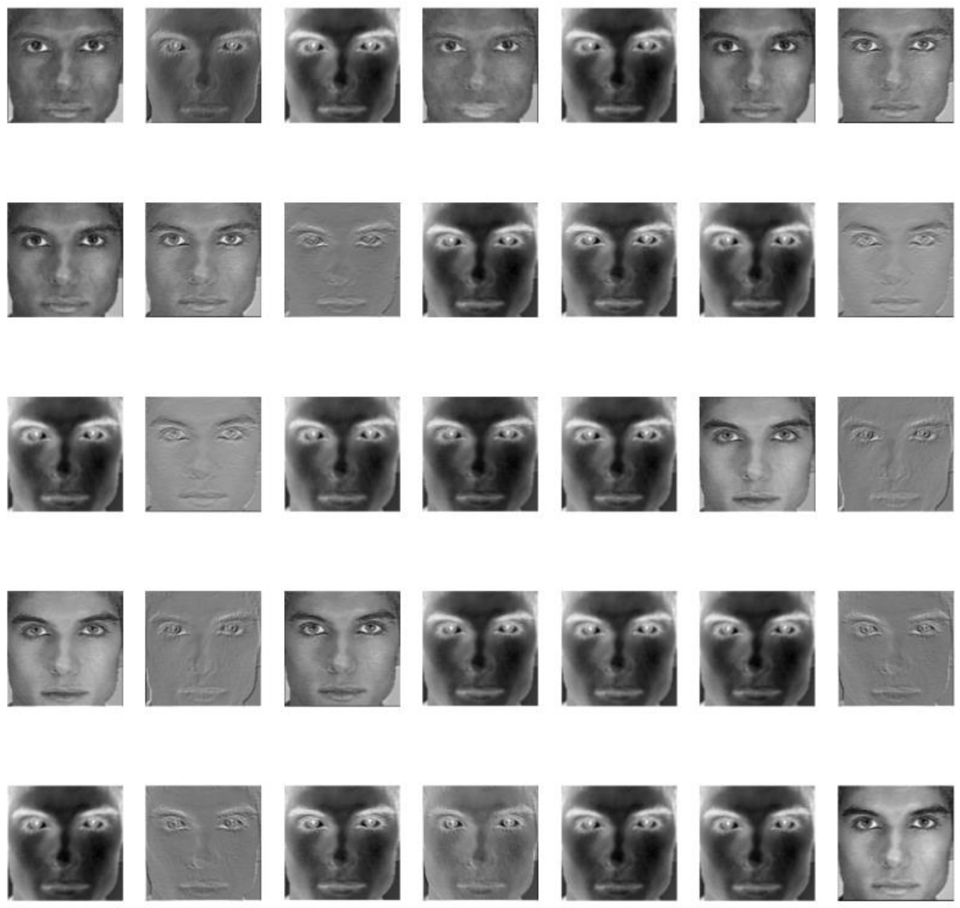


Figure 50: Gender Model Feature Map.

Model's Confusion matrix showed a good accuracy with male being the highest true positives.

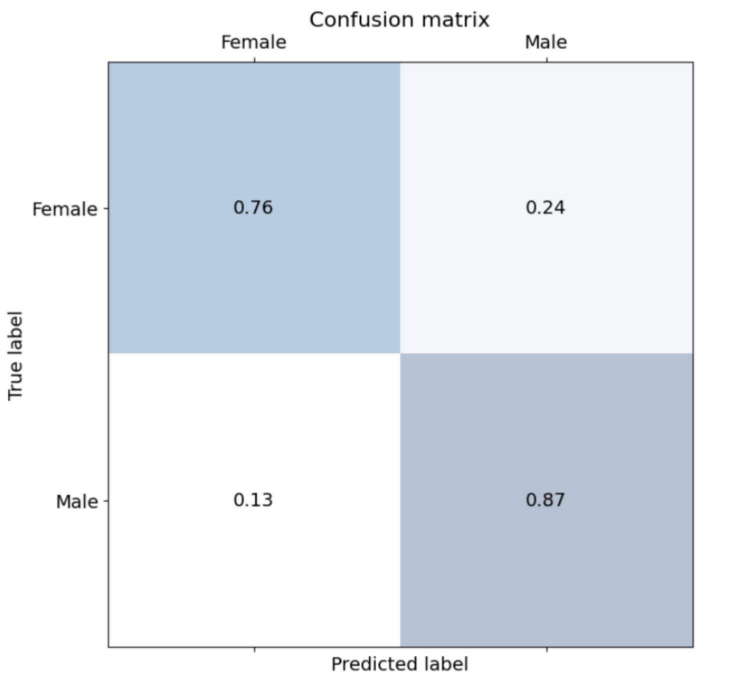


Figure 51:Gender Model Confusion Matrix.

The model achieved accuracy of 81% and loss of 0.8463 on the test data.

4- Ethnicity Prediction.

The objective of this module is to develop a model capable of predicting ethnicity with responsible accuracy.

In the development of this model both fair face and UTK-Face Datasets were tried but we got poor results because of the huge noise and misclassifications in the fair face dataset and the low size and variety of races in UTK-Face dataset.

In the search for a better dataset, we encountered a private dataset named "Ethnicity Aware" Dataset.

We sent an email to Beijing university in China which hold the license of the dataset to request the access to it.

After agreeing to the T&C we had access to the Ethnicity Aware dataset.

The dataset consists of 1,250,000 images labeled as Caucasian, African, North Asian, south Asian, and Indian.

The dataset doesn't have images for middle eastern or Arabs.

To solve this problem, we added new dataset called "Arab celebrity faces" which had 7453 images of famous Arab and middle eastern celebrities.

In addition to those two datasets, we added middle east and Arab images from Fair Face dataset after applying haar cascade algorithm.

Total images of middle eastern Arabs are 17,878 images which still too little compared to other classes which may cause the model to prefer other classes over this class.

To solve this problem, we applied class balancing techniques to the model.

Both North Asians and South Asians classed combined into Asian Class to avoid any misclassifications due to the huge similarity between the two classes.

The final Classes and its distribution:

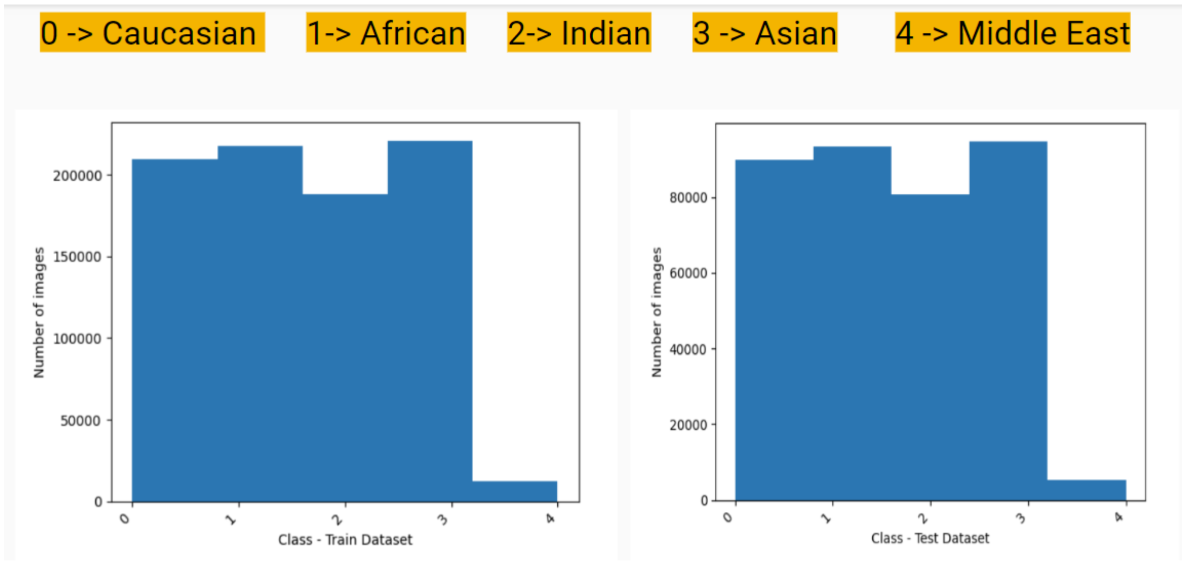


Figure 52: Ethnicity Model Classes Distribution.

A sample of the final Dataset:

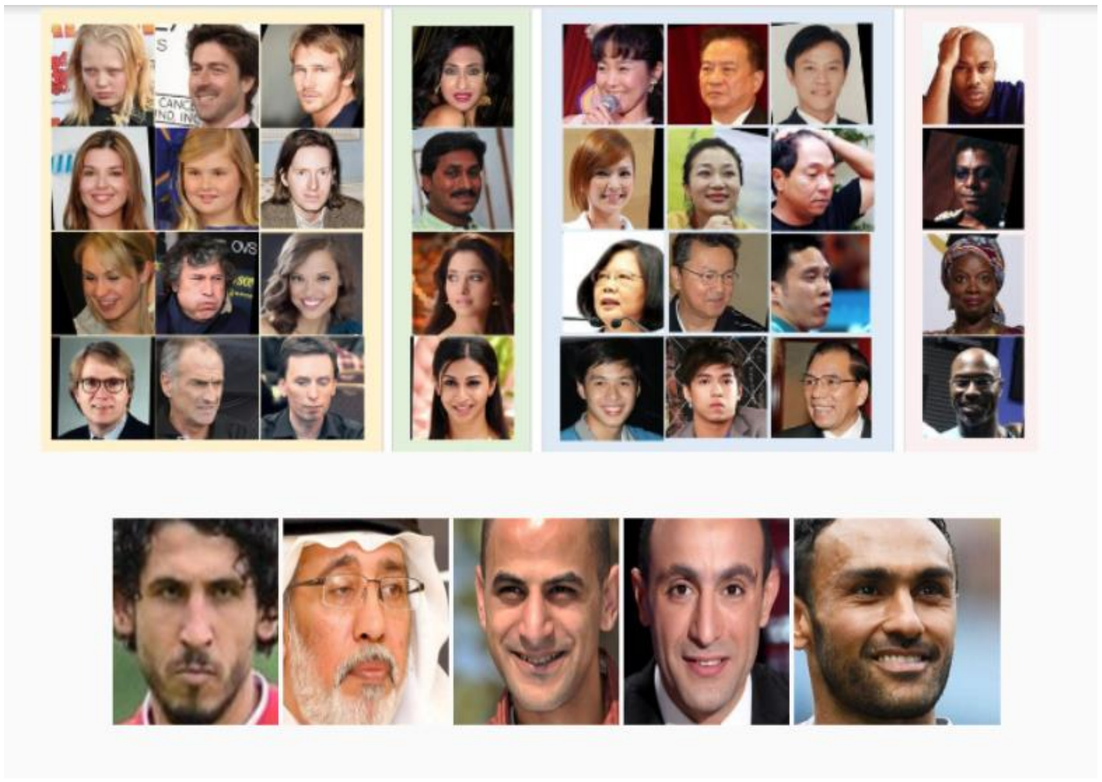


Figure 53: Ethnicity Model Dataset Samples.

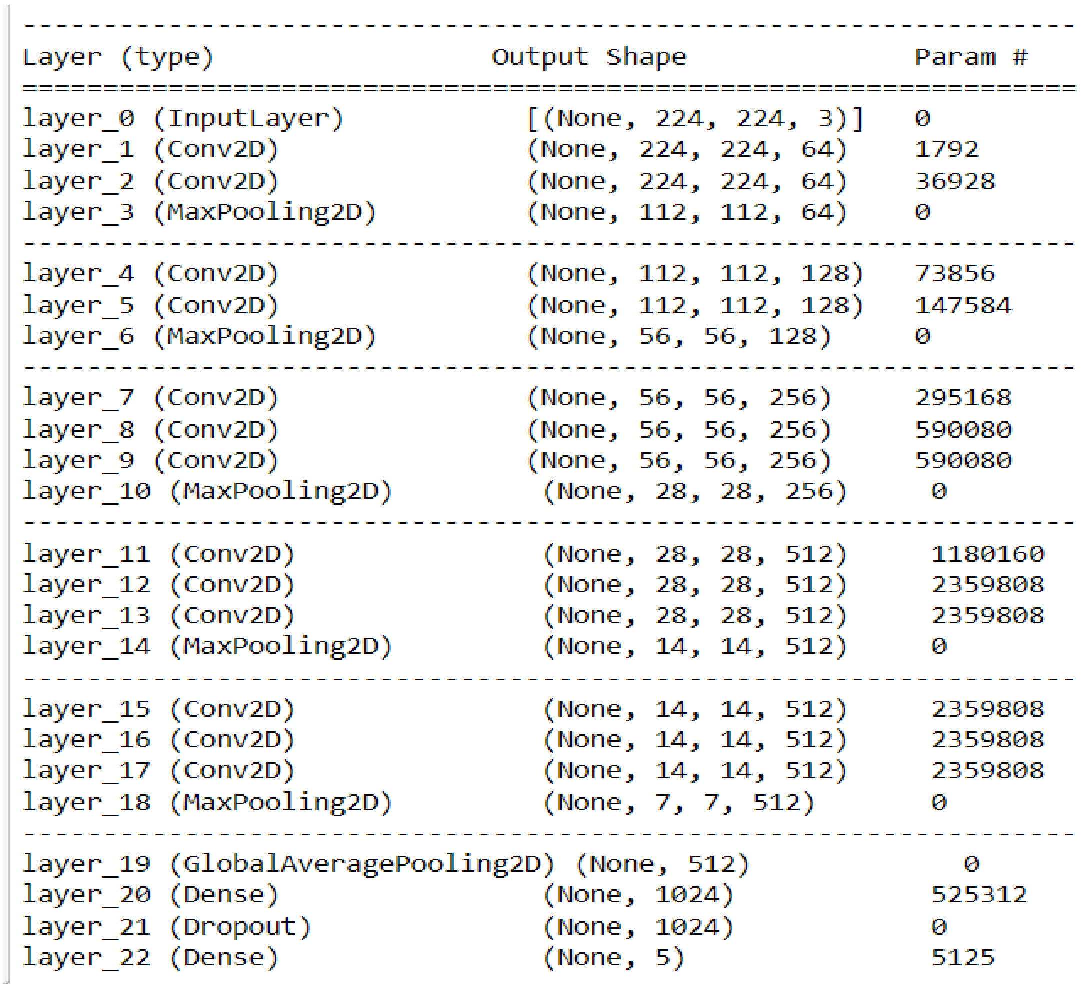
CNN architecture:

Using transfer learning, a VGG16 Model was chosen and fine-tuned.
To improve the feature extraction of the model we added the weights and parameters of VGG-Face Dataset to the VGG16 architecture.

A flattened layer was replaced with global average pool to improve model efficiency and reduce overfitting.

A Dense layer with 1024 units was added to help the model learn more complex features.

A dropout layer with 0.5 value was added to reduce overfitting.
A dense layer of 5 units with SoftMax activation function was used for classification.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_0 (InputLayer)          [(None, 224, 224, 3)]   0
layer_1 (Conv2D)              (None, 224, 224, 64)    1792
layer_2 (Conv2D)              (None, 224, 224, 64)    36928
layer_3 (MaxPooling2D)        (None, 112, 112, 64)    0
_________________________________________________________________
layer_4 (Conv2D)              (None, 112, 112, 128)   73856
layer_5 (Conv2D)              (None, 112, 112, 128)   147584
layer_6 (MaxPooling2D)        (None, 56, 56, 128)     0
_________________________________________________________________
layer_7 (Conv2D)              (None, 56, 56, 256)     295168
layer_8 (Conv2D)              (None, 56, 56, 256)     590080
layer_9 (Conv2D)              (None, 56, 56, 256)     590080
layer_10 (MaxPooling2D)        (None, 28, 28, 256)     0
_________________________________________________________________
layer_11 (Conv2D)              (None, 28, 28, 512)     1180160
layer_12 (Conv2D)              (None, 28, 28, 512)     2359808
layer_13 (Conv2D)              (None, 28, 28, 512)     2359808
layer_14 (MaxPooling2D)        (None, 14, 14, 512)     0
_________________________________________________________________
layer_15 (Conv2D)              (None, 14, 14, 512)     2359808
layer_16 (Conv2D)              (None, 14, 14, 512)     2359808
layer_17 (Conv2D)              (None, 14, 14, 512)     2359808
layer_18 (MaxPooling2D)        (None, 7, 7, 512)       0
_________________________________________________________________
layer_19 (GlobalAveragePooling2D) (None, 512)            0
layer_20 (Dense)               (None, 1024)            525312
layer_21 (Dropout)             (None, 1024)            0
layer_22 (Dense)               (None, 5)               5125
```

Figure 54: Ethnicity Model CNN Architecture.

Implementation:

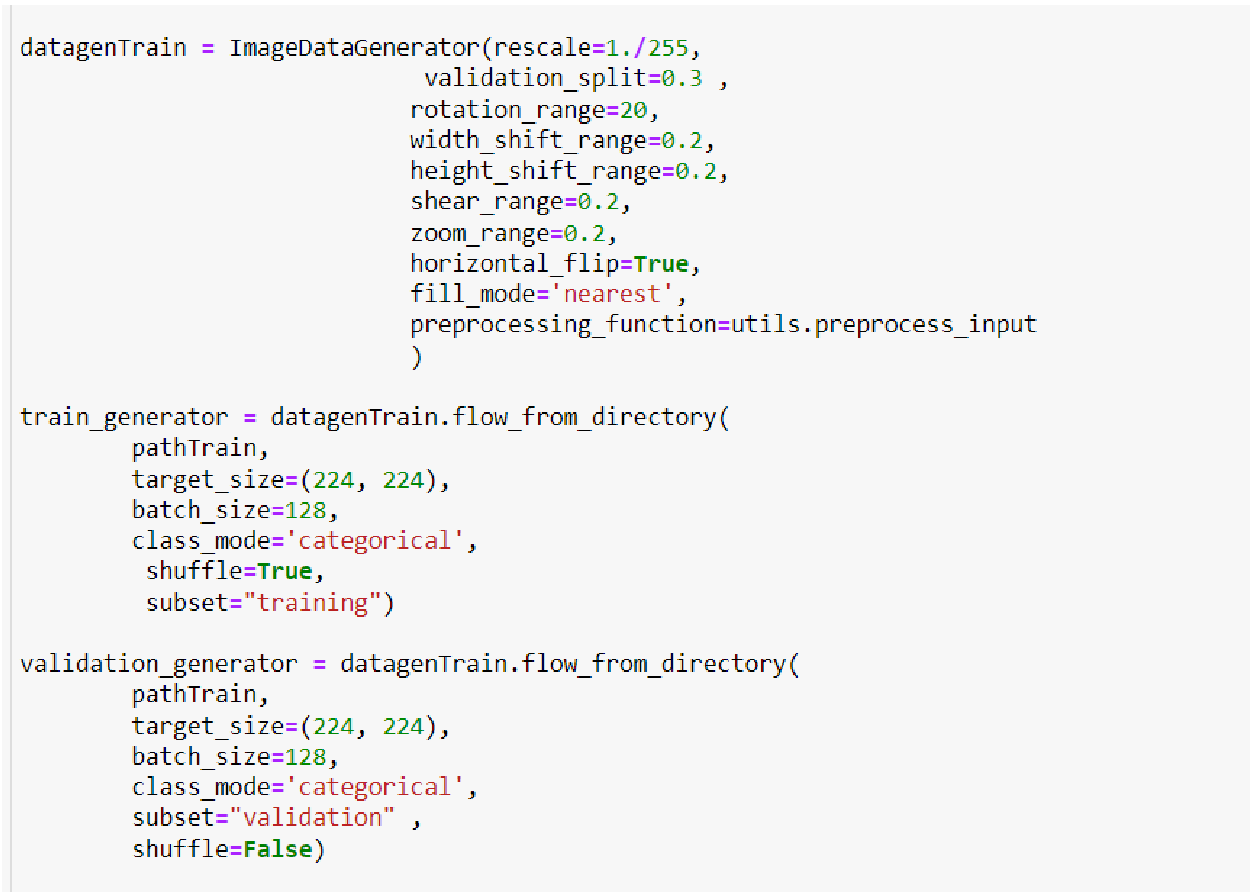
Native Keras Used in the implementation of this model because of its support for VGG-Face Dataset Parameters.

Image Augmentation and Normalization applied to all images.

VGG-FACE preprocessing techniques applied to all images, those techniques consist of Mean Subtraction and the Convert from RGB to BGR to meet the requirements of VGG-Face Dataset Parameters that will be applied to VGG16 architecture.

Dataset split to 70% training and 30% validation and testing.

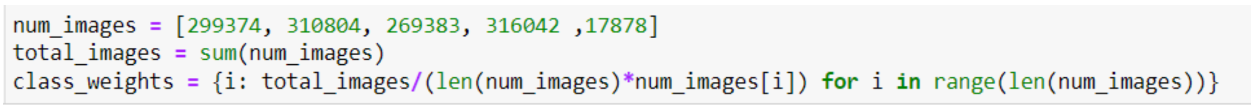
All images resized to 224x224x3.



```
datagenTrain = ImageDataGenerator(rescale=1./255,
                              validation_split=0.3 ,
                             rotation_range=20,
                             width_shift_range=0.2,
                             height_shift_range=0.2,
                             shear_range=0.2,
                             zoom_range=0.2,
                             horizontal_flip=True,
                             fill_mode='nearest',
                             preprocessing_function=utils.preprocess_input
                             )

train_generator = datagenTrain.flow_from_directory(
        pathTrain,
        target_size=(224, 224),
        batch_size=128,
        class_mode='categorical',
         shuffle=True,
         subset="training")

validation_generator = datagenTrain.flow_from_directory(
        pathTrain,
        target_size=(224, 224),
        batch_size=128,
        class_mode='categorical',
        subset="validation" ,
        shuffle=False)
```

Figure 55: Ethnicity Model Image Data Generator.

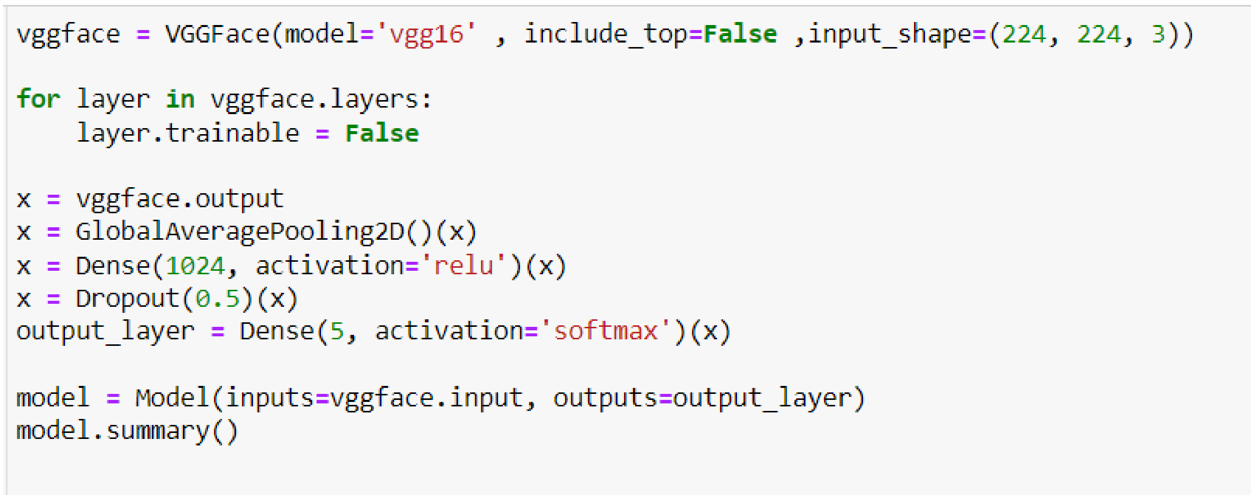
Class weights Balancing applied to all classes.



```
num_images = [299374, 310804, 269383, 316042 ,17878]
total_images = sum(num_images)
class_weights = {i: total_images/(len(num_images)*num_images[i]) for i in range(len(num_images))}
```

Figure 56: Ethnicity Model Class weight Balancing.

VGG-FACE weights applied to vgg16 architecture, and the architecture layers and we freeze the architecture layers to prevent the weight from changing during learning by setting it trainability to False and fine tune the architecture.

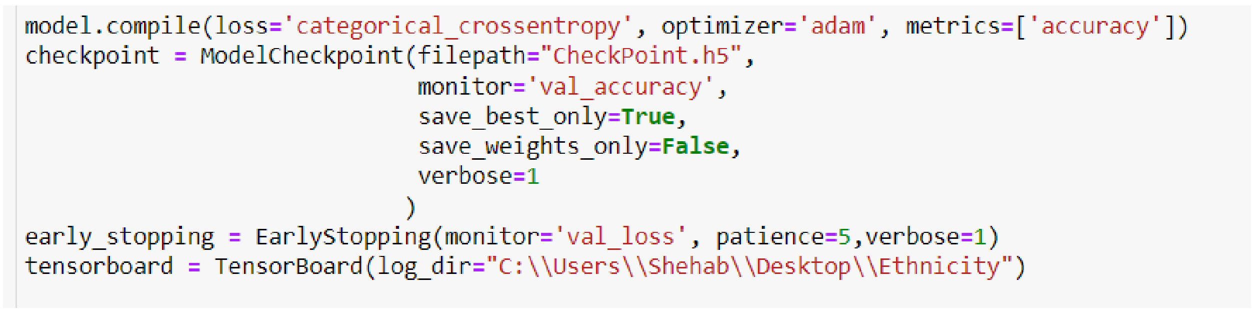


```
vggface = VGGFace(model='vgg16' , include_top=False ,input_shape=(224, 224, 3))

for layer in vggface.layers:
    layer.trainable = False

x = vggface.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
output_layer = Dense(5, activation='softmax')(x)

model = Model(inputs=vggface.input, outputs=output_layer)
model.summary()
```

Figure 57: Fine Tuning VGG16 in Ethnicity Model.

A Cross Entropy loss, Adam optimizer and accuracy metric were chosen. Checkpoint 1 verbose, early stopping with 5 patience and tensor board callbacks initialized.

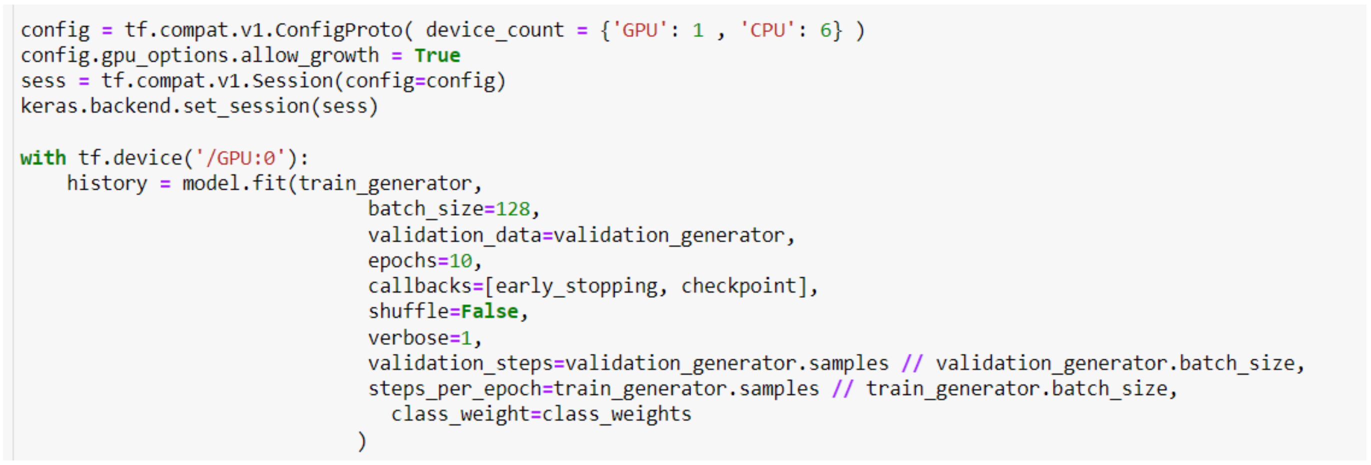
The default learning rate of 0.001 of Adam Optimizer chosen.



```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
checkpoint = ModelCheckpoint(filepath="CheckPoint.h5",
                             monitor='val_accuracy',
                             save_best_only=True,
                             save_weights_only=False,
                             verbose=1
                            )
early_stopping = EarlyStopping(monitor='val_loss', patience=5,verbose=1)
tensorboard = TensorBoard(log_dir="C:\\Users\\Shehab\\Desktop\\Ethnicity")
```

Figure 58: Compile and Call backs Functions in Ethnicity Model.

Model Fitted with 100 epochs and 128 batch size.



```
config = tf.compat.v1.ConfigProto( device_count = {'GPU': 1 , 'CPU': 6} )
config.gpu_options.allow_growth = True
sess = tf.compat.v1.Session(config=config)
keras.backend.set_session(sess)

with tf.device('/GPU:0'):
    history = model.fit(train_generator,
                              batch_size=128,
                              validation_data=validation_generator,
                              epochs=10,
                              callbacks=[early_stopping, checkpoint],
                              shuffle=False,
                              verbose=1,
                              validation_steps=validation_generator.samples // validation_generator.batch_size,
                              steps_per_epoch=train_generator.samples // train_generator.batch_size,
                                class_weight=class_weights
                             )
```

Figure 59: Fitting Ethnicity Model.

Results:

The early stopping method hasn't been triggered and model completed 10 epochs. The model hasn't been trained further since it started to show signs of overfitting on the tensor board at the beginning of epoch 11 when fitted for 20 epochs.

The model achieved 87% training accuracy, 0.3655 loss, 87% validation accuracy and 0.3692 validation loss.

Model Convergence:

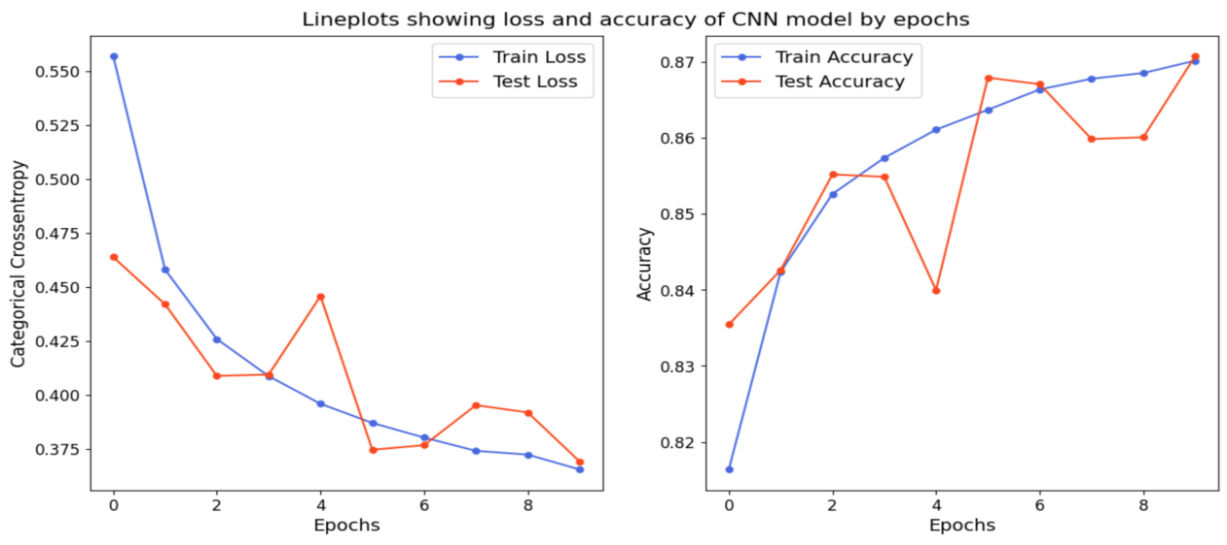


Figure 60: Ethnicity Model Convergence Plot.

Features captured in every convolution layer:

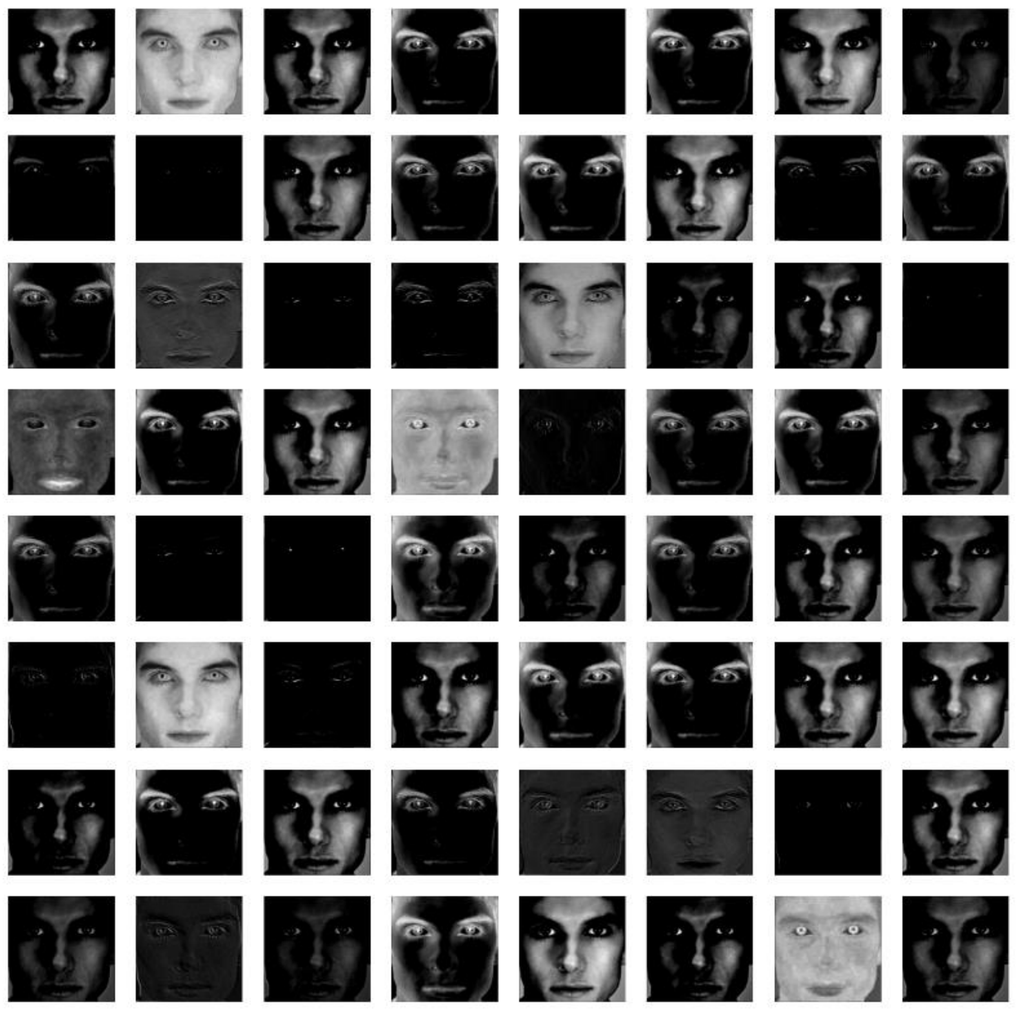


Figure 61: Ethnicity Model Feature Map.

Heat Map of each Convolution layer:

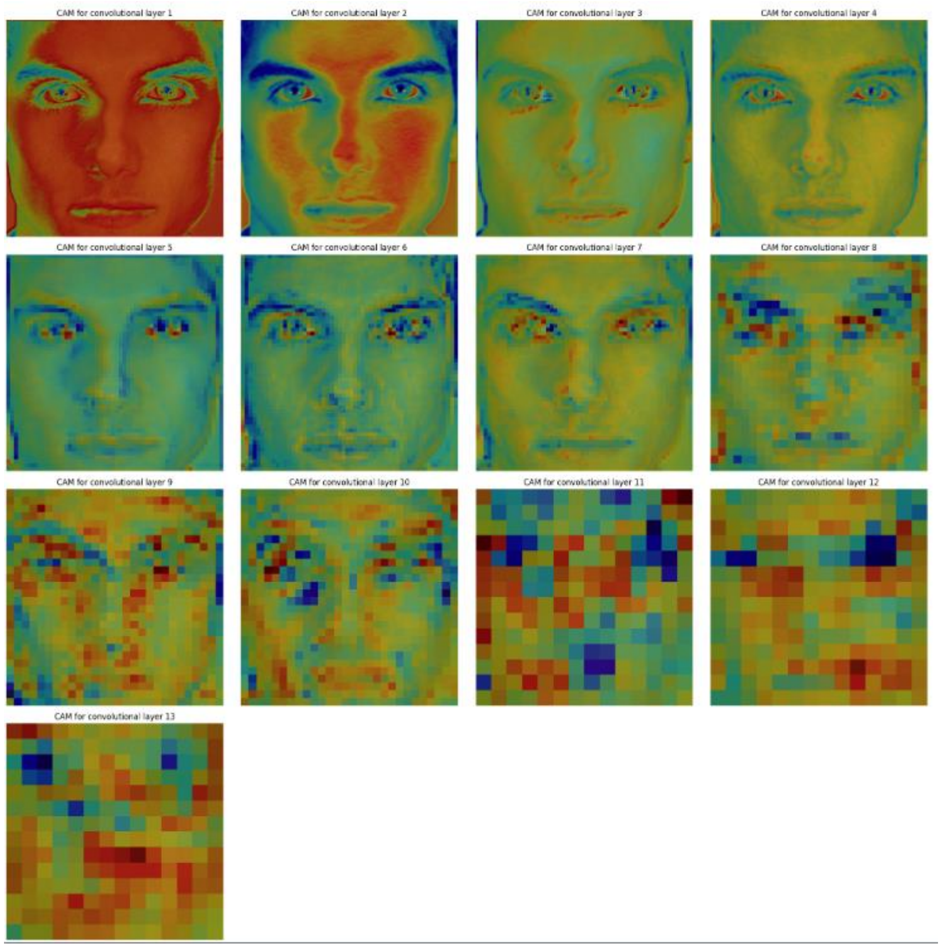
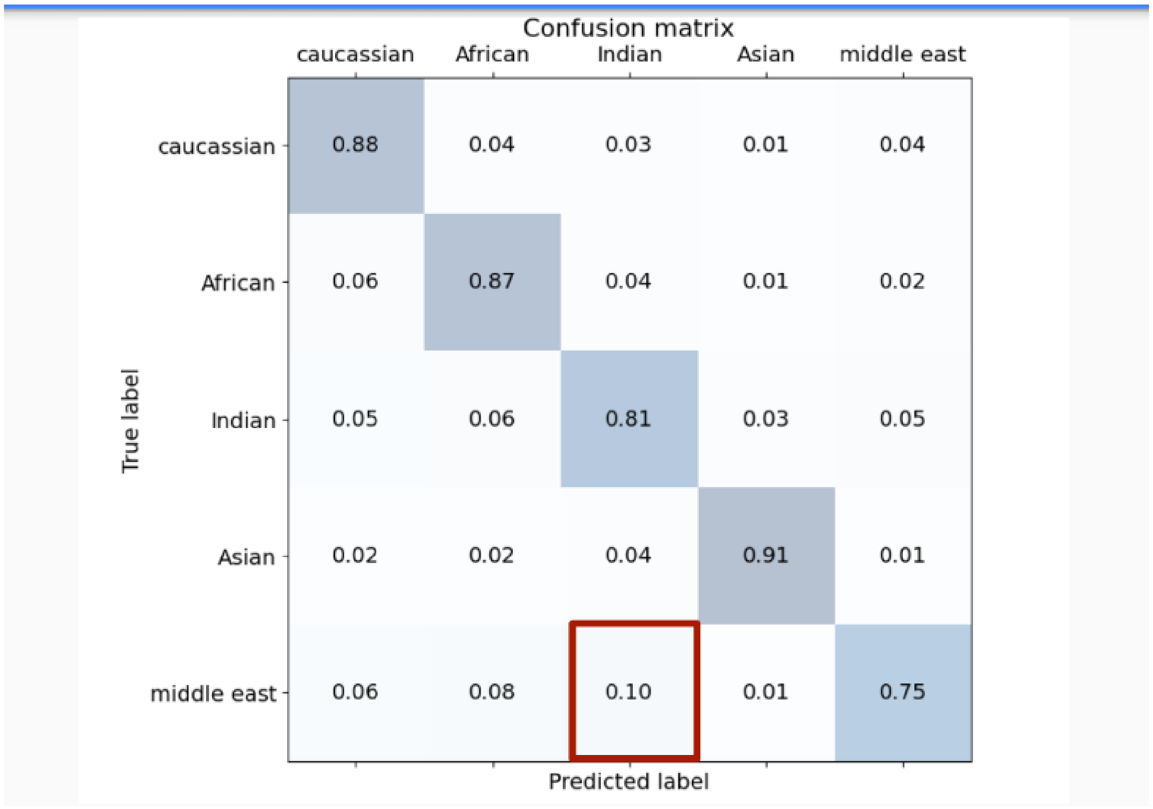

CAM for convolutional layer 1
CAM for convolutional layer 2
CAM for convolutional layer 3
CAM for convolutional layer 4
CAM for convolutional layer 5
CAM for convolutional layer 6
CAM for convolutional layer 7
CAM for convolutional layer 8
CAM for convolutional layer 9
CAM for convolutional layer 10
CAM for convolutional layer 11
CAM for convolutional layer 12
CAM for convolutional layer 13

Figure 62:Ethnicity Model Heat Map.

Model’s Confusion matrix shows some confusion between middle eastern and Indian which is normal since both ethnicity groups share many features and similarities that is sometime harder to classify even under human judgement.



Confusion matrix

True label \ Predicted label	caucassian	African	Indian	Asian	middle east
caucassian	0.88	0.04	0.03	0.01	0.04
African	0.06	0.87	0.04	0.01	0.02
Indian	0.05	0.06	0.81	0.03	0.05
Asian	0.02	0.02	0.04	0.91	0.01
middle east	0.06	0.08	0.10	0.01	0.75

Figure 63: Ethnicity Model Confusion Matrix.

The model achieved accuracy of 86% and loss of 0.3744 on the test data.

5- Emotions Prediction

The objective of this module is to build a model capable of predicting human emotions from an image.

The Dataset used in this model is FER-2013 Dataset which consists of 35887 Gray Scale images split to 7 Classes Represent the Main Human Emotions which are anger, disgust, fear, happiness, sadness, surprises and Neutral.

The Haar cascade algorithm was used to clean the dataset.

Due to the low number of images, Data Augmentation was used on the Dataset. All images resized to 48x48x1.

Classes Distributions in the train and Test Datasets:

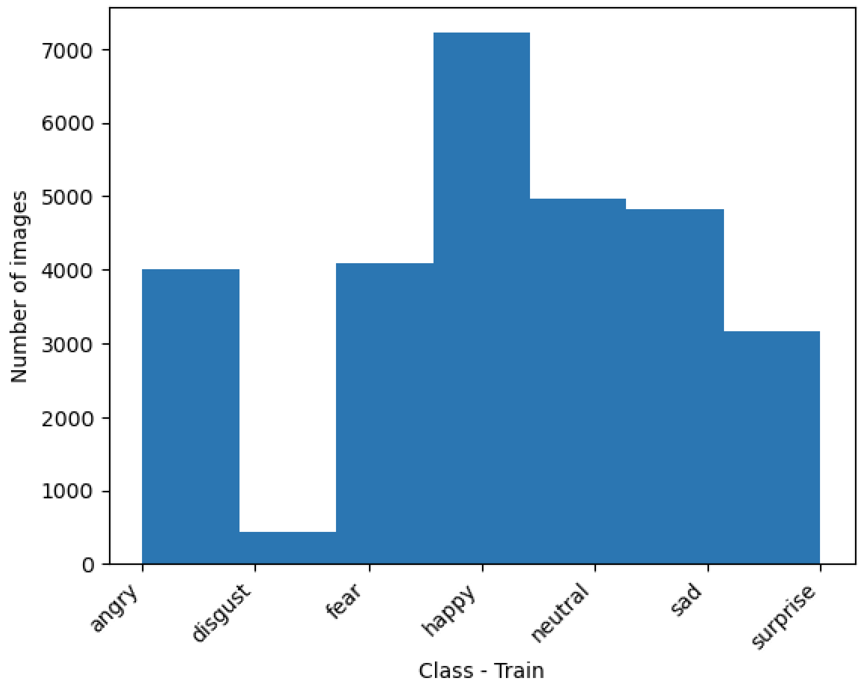


Figure 64: Train Dataset Class Distribution in Emotion Model.

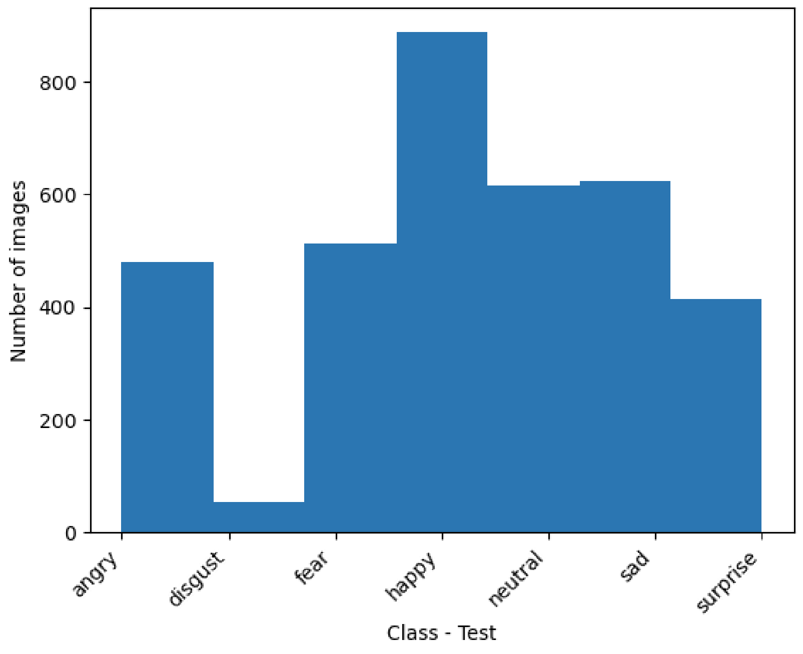


Figure 65: Test Dataset Class Distribution in Emotion Model.

Samples of the Dataset Used:

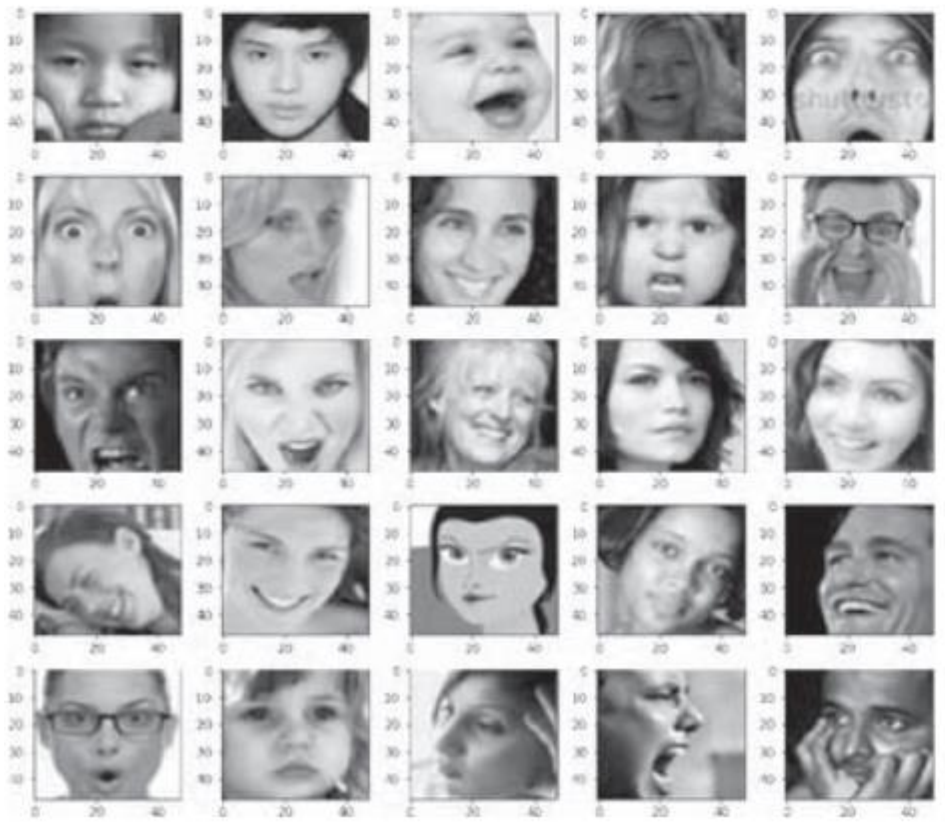


Figure 66: Sample of Dataset used in Emotion Model.

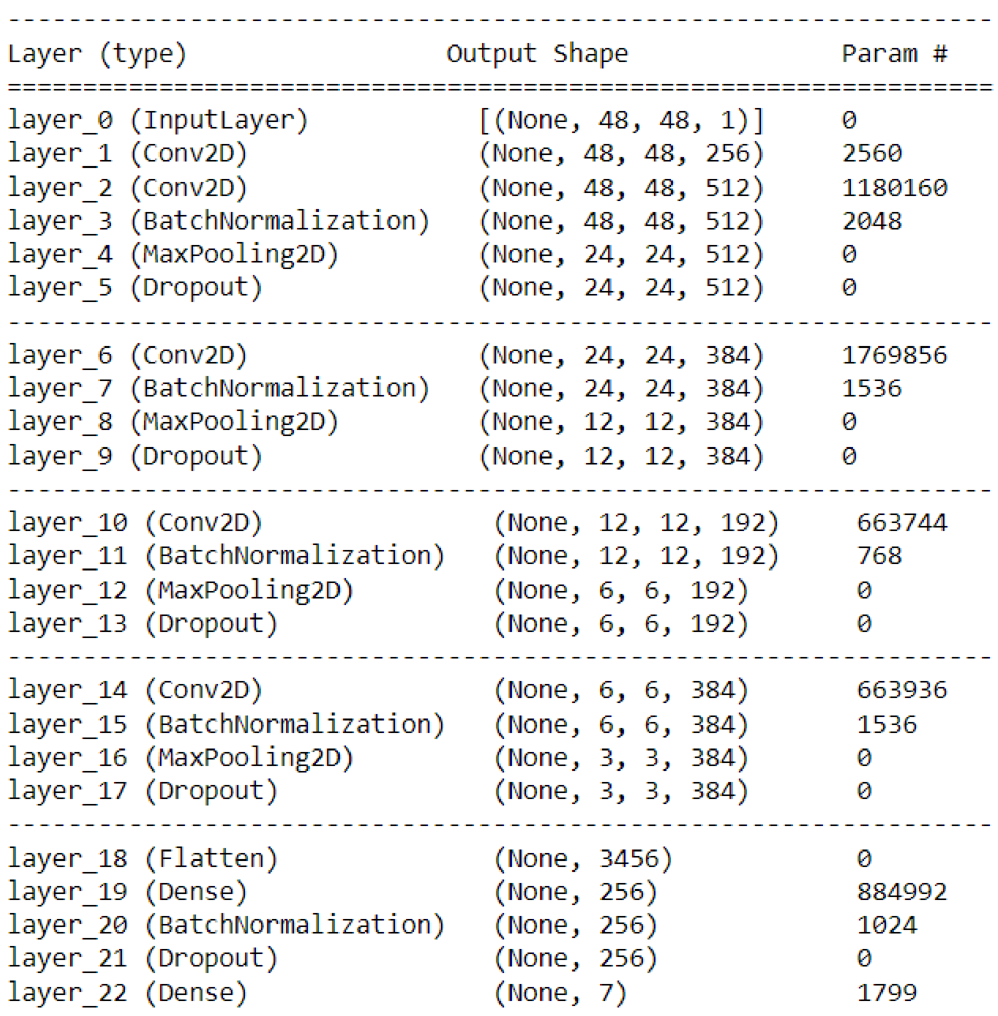
CNN architecture:

Different CNN architectures were tested for this model.

Four Convolution Blocks were chosen for features extraction, each block has convolution layer with RelU activation, Batch Normalization layer, Max pool layer and Dropout layer.

Classification block with dense, batch normalize, drop out and SoftMax layers were chosen to classify between 7 classes.

Having batch normalization, max pool and drop out layers in each block increased model efficiency.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_0 (InputLayer)          [(None, 48, 48, 1)]      0
layer_1 (Conv2D)              (None, 48, 48, 256)      2560
layer_2 (Conv2D)              (None, 48, 48, 512)      1180160
layer_3 (BatchNormalization)  (None, 48, 48, 512)      2048
layer_4 (MaxPooling2D)        (None, 24, 24, 512)      0
layer_5 (Dropout)             (None, 24, 24, 512)      0
_________________________________________________________________
layer_6 (Conv2D)              (None, 24, 24, 384)      1769856
layer_7 (BatchNormalization)  (None, 24, 24, 384)      1536
layer_8 (MaxPooling2D)        (None, 12, 12, 384)      0
layer_9 (Dropout)             (None, 12, 12, 384)      0
_________________________________________________________________
layer_10 (Conv2D)              (None, 12, 12, 192)      663744
layer_11 (BatchNormalization)  (None, 12, 12, 192)      768
layer_12 (MaxPooling2D)        (None, 6, 6, 192)        0
layer_13 (Dropout)             (None, 6, 6, 192)        0
_________________________________________________________________
layer_14 (Conv2D)              (None, 6, 6, 384)        663936
layer_15 (BatchNormalization)  (None, 6, 6, 384)        1536
layer_16 (MaxPooling2D)        (None, 3, 3, 384)        0
layer_17 (Dropout)             (None, 3, 3, 384)        0
_________________________________________________________________
layer_18 (Flatten)             (None, 3456)             0
layer_19 (Dense)               (None, 256)              884992
layer_20 (BatchNormalization)  (None, 256)              1024
layer_21 (Dropout)             (None, 256)              0
layer_22 (Dense)               (None, 7)                1799
```

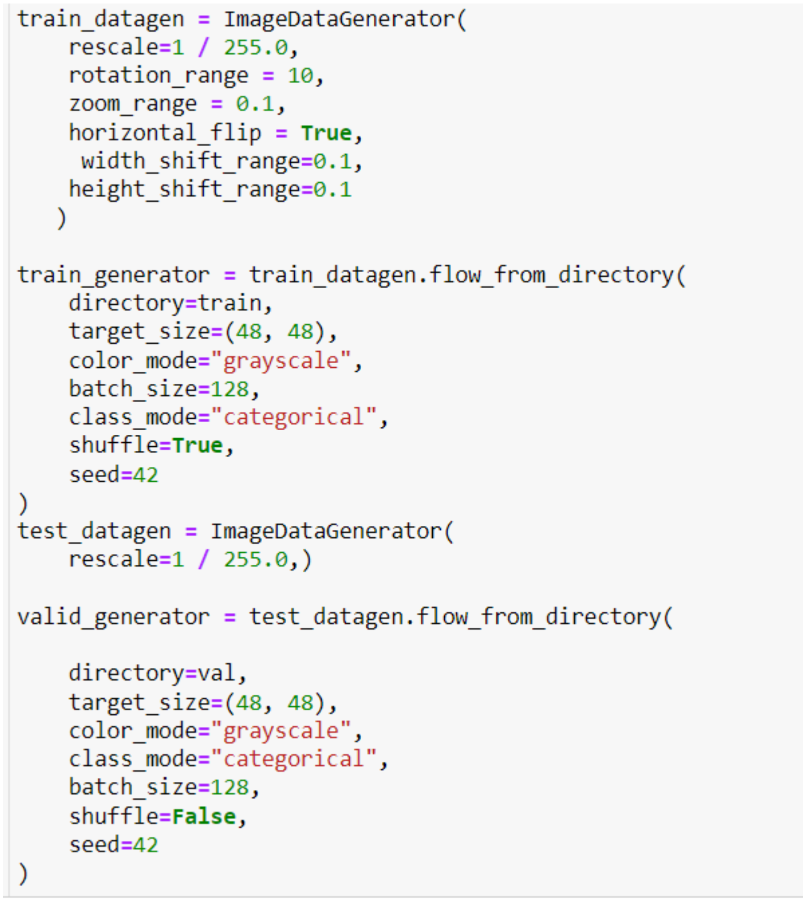
Figure 67: Emotion Model CNN Architecture.

Implementation:

Using TensorFlow we implemented image data generator and applied normalization and augmentation.

Images resized to 48x48x1, and dataset split to 80% for training and 20% for validation and testing.

A batch size of 128 was chosen.



```
train_datagen = ImageDataGenerator(
    rescale=1 / 255.0,
    rotation_range = 10,
    zoom_range = 0.1,
    horizontal_flip = True,
     width_shift_range=0.1,
    height_shift_range=0.1
   )

train_generator = train_datagen.flow_from_directory(
    directory=train,
    target_size=(48, 48),
    color_mode="grayscale",
    batch_size=128,
    class_mode="categorical",
    shuffle=True,
    seed=42
)
test_datagen = ImageDataGenerator(
    rescale=1 / 255.0,)

valid_generator = test_datagen.flow_from_directory(

    directory=val,
    target_size=(48, 48),
    color_mode="grayscale",
    class_mode="categorical",
    batch_size=128,
    shuffle=False,
    seed=42
)
```

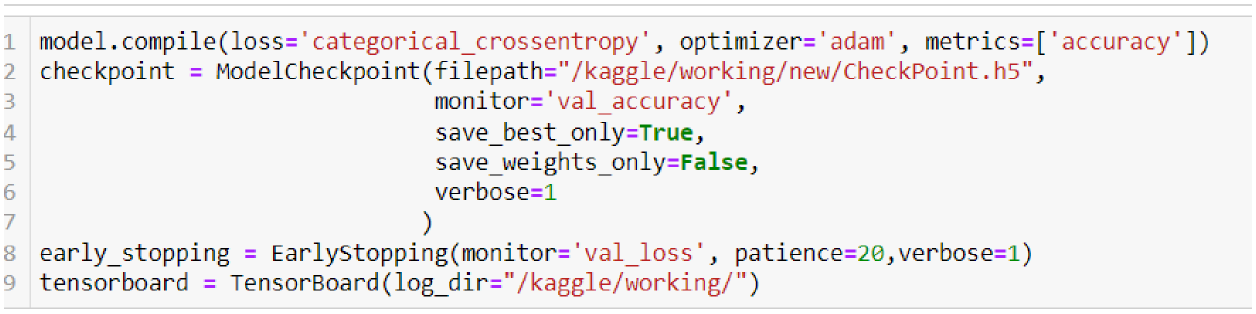
Figure 68: Image Data Generator in Emotion Model.

Class weight balancing was applied because of the unbalanced distribution in the data between classes.

Cross entropy loss, Adam optimizer and accuracy metric were chosen.

Learn rate of 0.001 was chosen.

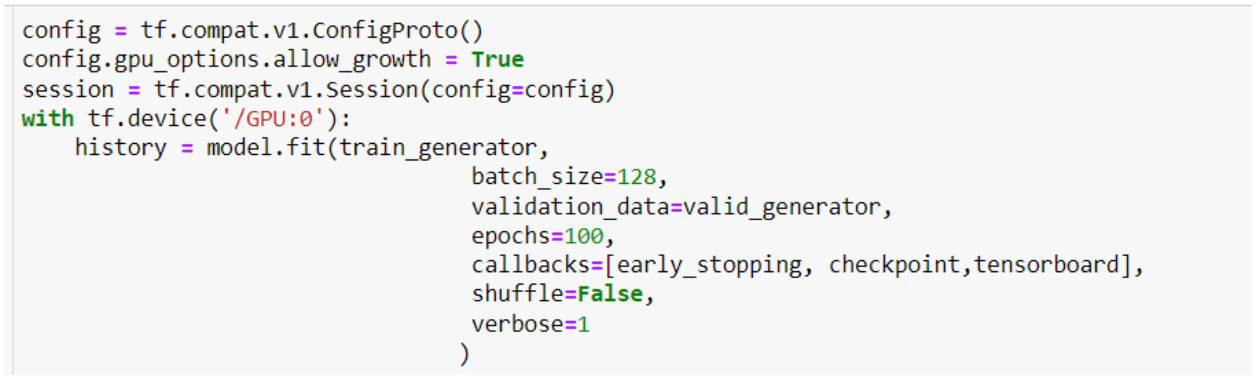
Callbacks of checkpoint with 1 verbose, early stop with 20 patience and tensor board were initialized.



```python
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
checkpoint = ModelCheckpoint(filepath="/kaggle/working/new/CheckPoint.h5",
                             monitor='val_accuracy',
                             save_best_only=True,
                             save_weights_only=False,
                             verbose=1
                            )
early_stopping = EarlyStopping(monitor='val_loss', patience=20,verbose=1)
tensorboard = TensorBoard(log_dir="/kaggle/working/")
```

Figure 69: Compile and Call Backs Functions in Emotion Model.

Model Fitted with 128 batch size and 100 epochs.



```python
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
with tf.device('/GPU:0'):
    history = model.fit(train_generator,
                              batch_size=128,
                              validation_data=valid_generator,
                              epochs=100,
                              callbacks=[early_stopping, checkpoint,tensorboard],
                              shuffle=False,
                              verbose=1
                             )
```

Figure 70: Fitting Emotion Model.

Results:

Early Stopping has been triggered after 69 epochs.

Model achieved 0.7309 train loss, 72% train accuracy, 0.9411 validation loss and 67% validation accuracy.

The model showed stable convergence and checkpoints callback used to save the best weights.

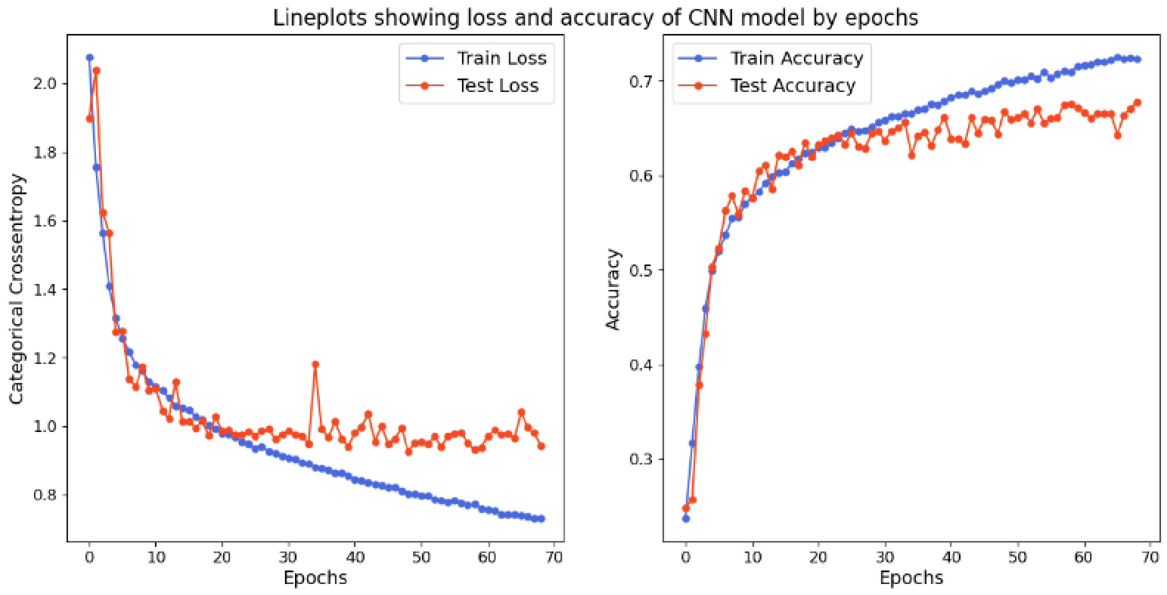
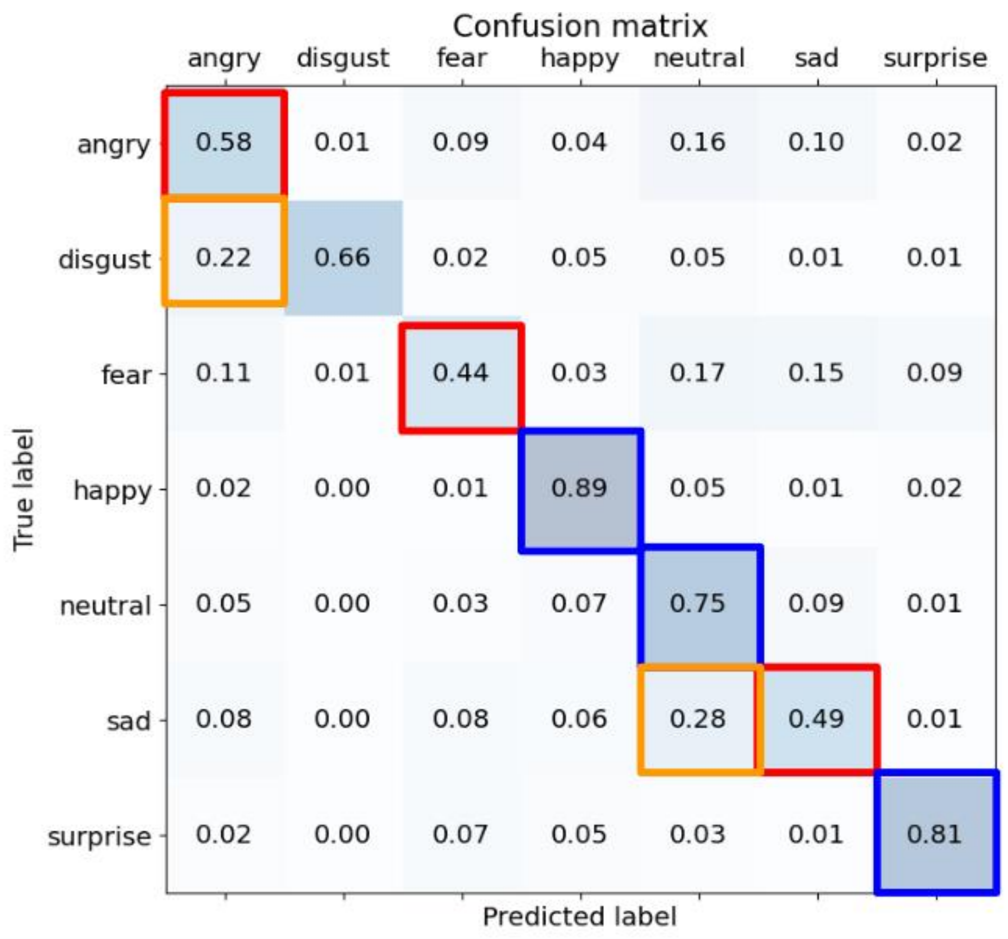


Figure 71: Emotion Model Convergence Plot.

The Model Showed some confusion between some classes like:

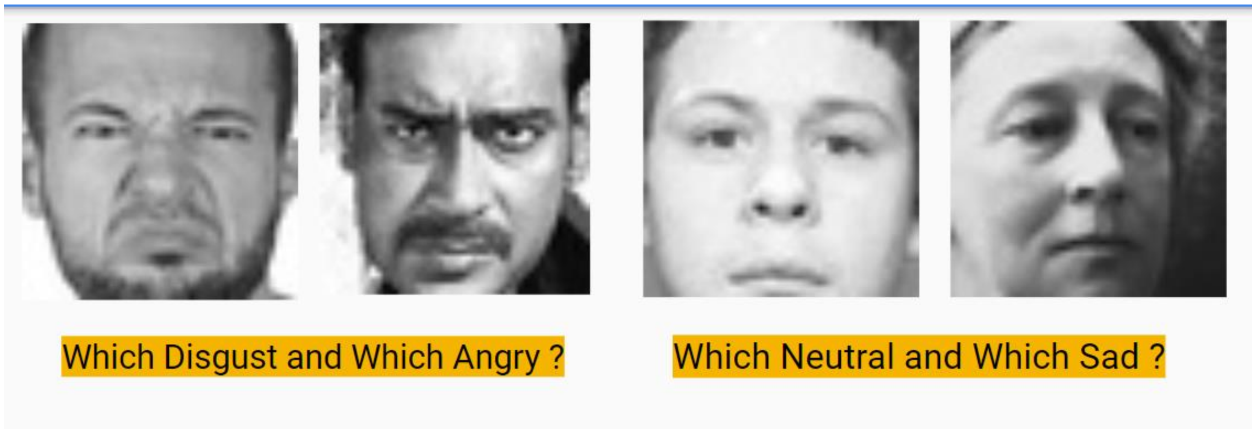
1- Disgust vs Anger
2- Sad vs Neutral
3- Fear vs Sad
4- Fear vs Neutral

Those classes are sometime hard for even humans to classify them and due to the limited data and the low number of images in some classes like disgust class, those confusions are expected.



	angry	disgust	fear	happy	neutral	sad	surprise
angry	0.58	0.01	0.09	0.04	0.16	0.10	0.02
disgust	0.22	0.66	0.02	0.05	0.05	0.01	0.01
fear	0.11	0.01	0.44	0.03	0.17	0.15	0.09
happy	0.02	0.00	0.01	0.89	0.05	0.01	0.02
neutral	0.05	0.00	0.03	0.07	0.75	0.09	0.01
sad	0.08	0.00	0.08	0.06	0.28	0.49	0.01
surprise	0.02	0.00	0.07	0.05	0.03	0.01	0.81

Figure 72: Emotion Model Confusion Matrix.



Which Disgust and Which Angry ?
Which Neutral and Which Sad ?

Figure 73: Emotion Model Confessions that are hard to classify.

The model Achieved 0.9411 loss and 67% accuracy on the test data.

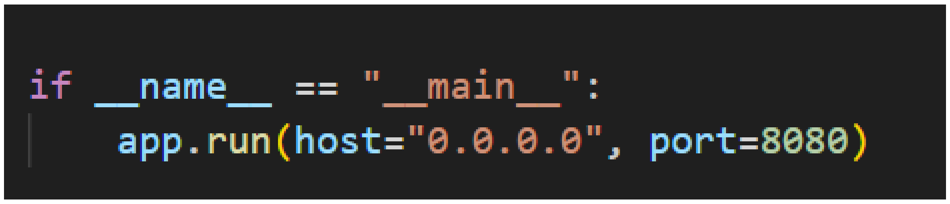
6- API Endpoint:

The objective of this module is to develop an API Endpoint capable of delivering the Project Features to other Systems easily.

Flask Framework was used to build the API Endpoint.

How it Work:

1- The API Application launches on port 8080.



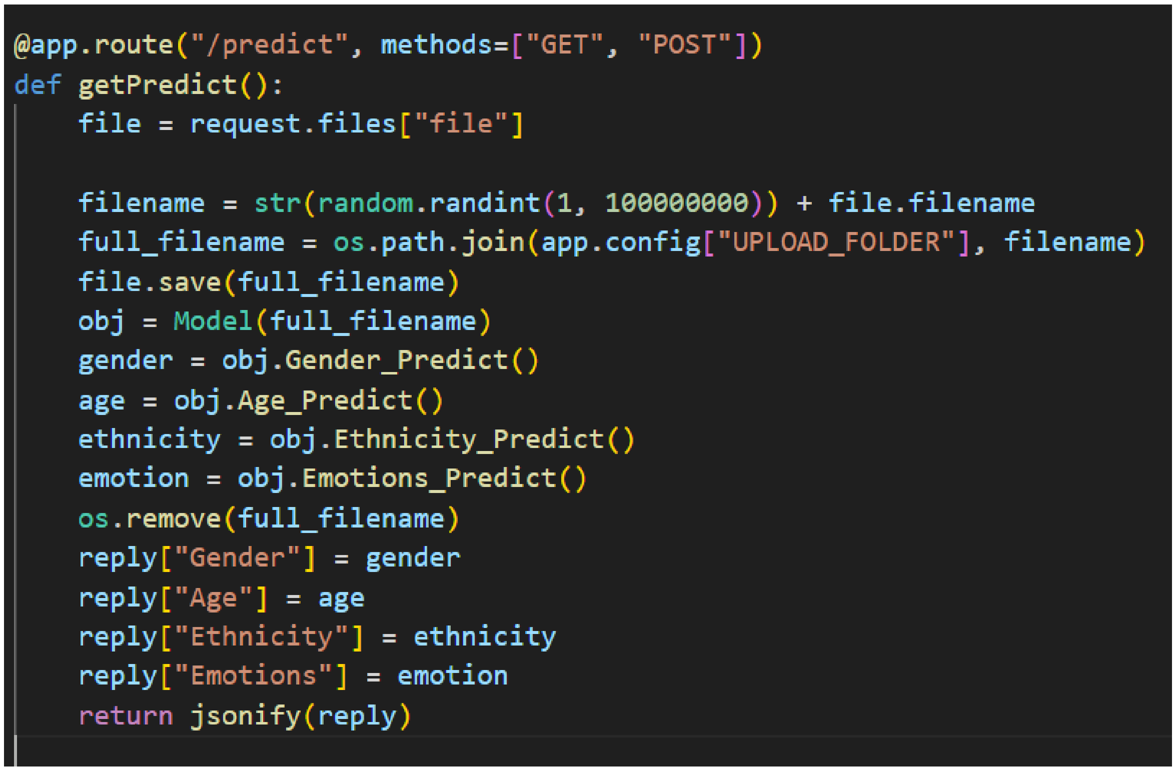
```python
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Figure 74: API Running on port 8080.

2- The API Application receives requests on "/predict" Endpoint.

The Application save the received image on static folder then creates an object from the class "Model" and pass the path of the image to it and start a method related to each model in the project to predict the corresponding task and return the predictions.

Predictions returned from each method saved in json format and sent as reply to the requester.

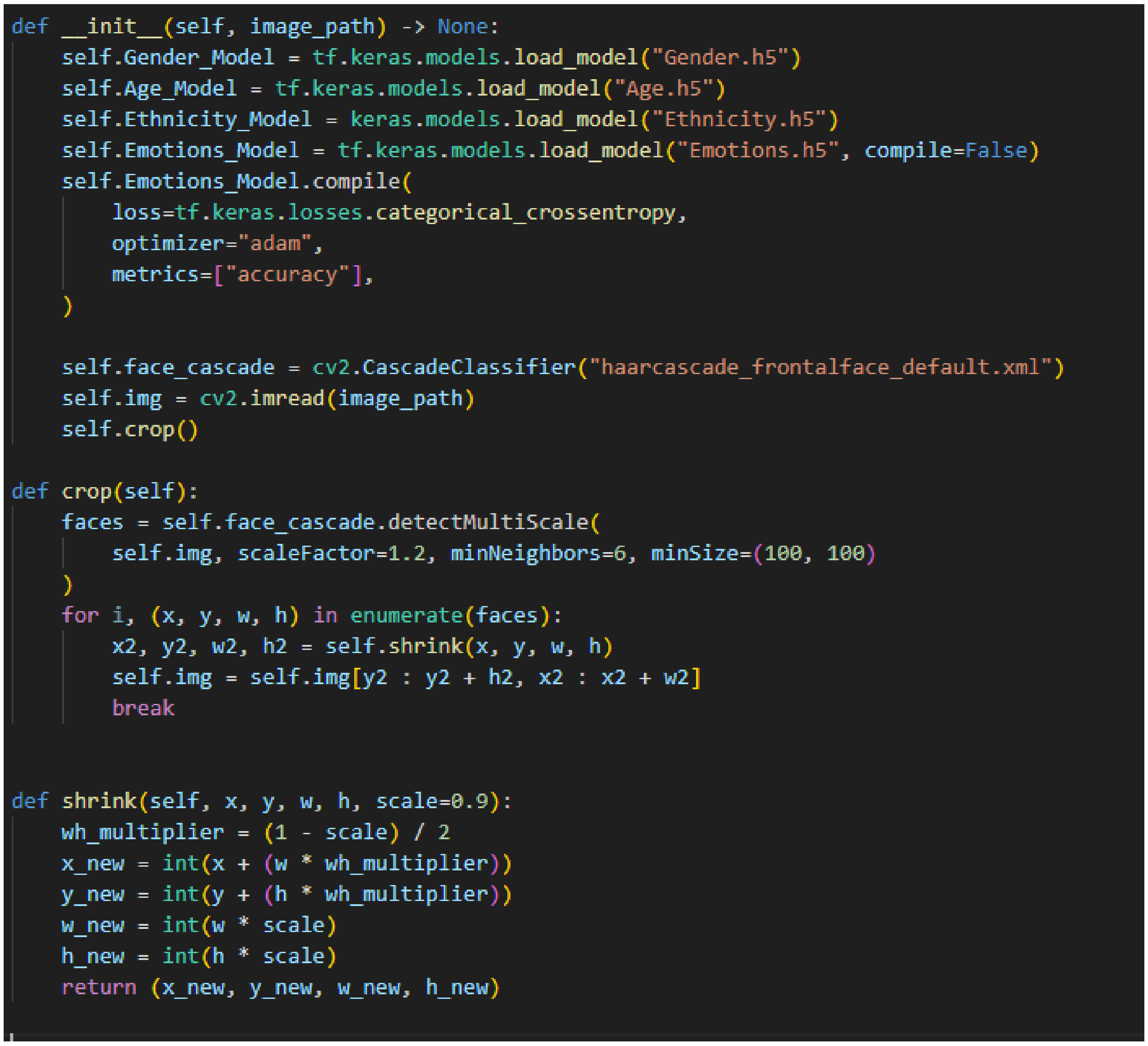


```
@app.route("/predict", methods=["GET", "POST"])
def getPredict():
    file = request.files["file"]

    filename = str(random.randint(1, 100000000)) + file.filename
    full_filename = os.path.join(app.config["UPLOAD_FOLDER"], filename)
    file.save(full_filename)
    obj = Model(full_filename)
    gender = obj.Gender_Predict()
    age = obj.Age_Predict()
    ethnicity = obj.Ethnicity_Predict()
    emotion = obj.Emotions_Predict()
    os.remove(full_filename)
    reply["Gender"] = gender
    reply["Age"] = age
    reply["Ethnicity"] = ethnicity
    reply["Emotions"] = emotion
    return jsonify(reply)
```

Figure 75:Method Called When API Endpoint Receives a Request Message.

3- When an object is created from the Class “Model” it loads the model weights for each model in H5 file format.

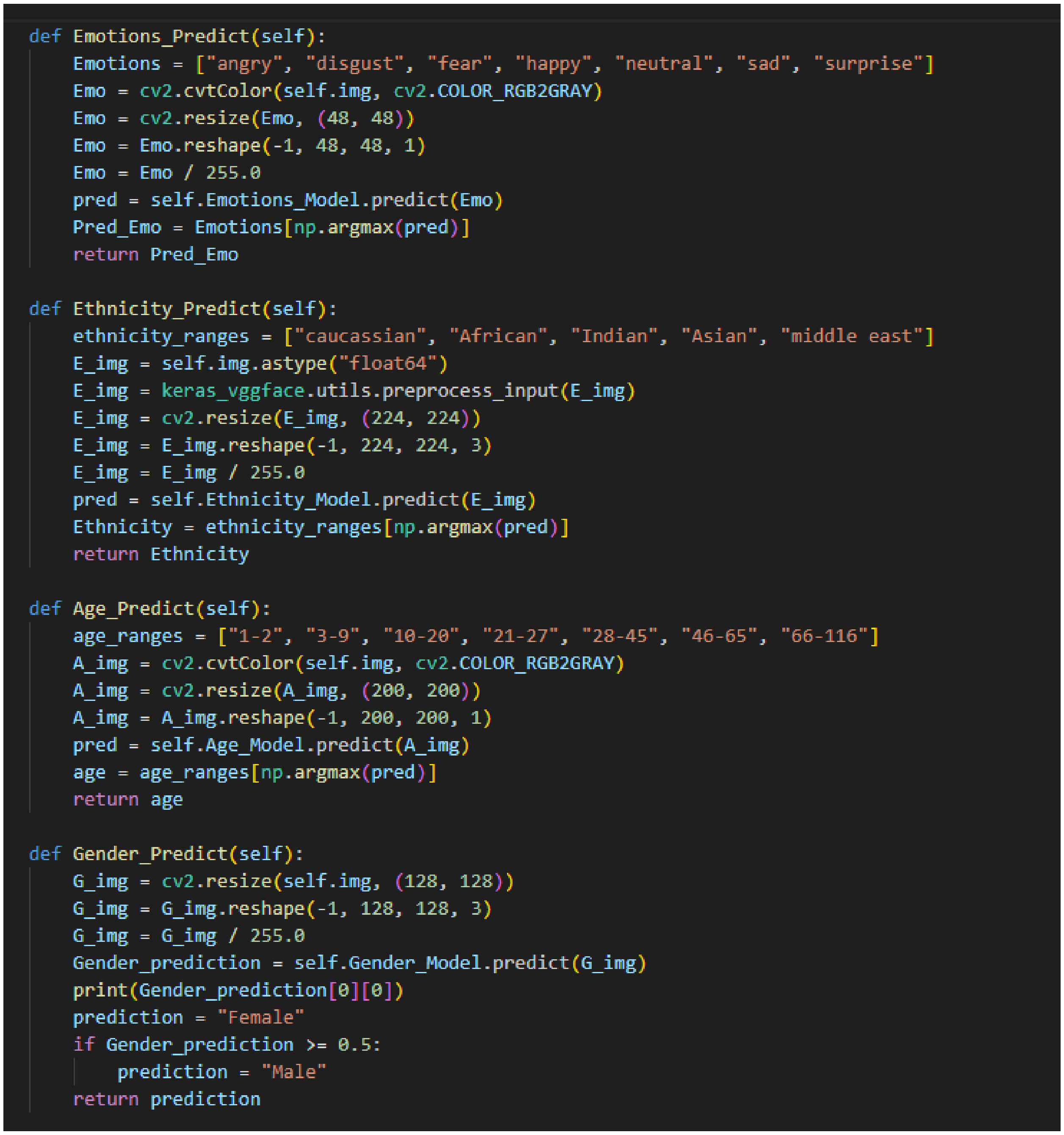
And initialize haarcascade algorithm and apply it on the image to extract the human face from the image and apply the crop method to crop it and the shrink method to shrink the output image.



```python
def __init__(self, image_path) -> None:
    self.Gender_Model = tf.keras.models.load_model("Gender.h5")
    self.Age_Model = tf.keras.models.load_model("Age.h5")
    self.Ethnicity_Model = keras.models.load_model("Ethnicity.h5")
    self.Emotions_Model = tf.keras.models.load_model("Emotions.h5", compile=False)
    self.Emotions_Model.compile(
        loss=tf.keras.losses.categorical_crossentropy,
        optimizer="adam",
        metrics=["accuracy"],
    )

    self.face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
    self.img = cv2.imread(image_path)
    self.crop()

def crop(self):
    faces = self.face_cascade.detectMultiScale(
        self.img, scaleFactor=1.2, minNeighbors=6, minSize=(100, 100)
    )
    for i, (x, y, w, h) in enumerate(faces):
        x2, y2, w2, h2 = self.shrink(x, y, w, h)
        self.img = self.img[y2 : y2 + h2, x2 : x2 + w2]
        break


def shrink(self, x, y, w, h, scale=0.9):
    wh_multiplier = (1 - scale) / 2
    x_new = int(x + (w * wh_multiplier))
    y_new = int(y + (h * wh_multiplier))
    w_new = int(w * scale)
    h_new = int(h * scale)
    return (x_new, y_new, w_new, h_new)
```

Figure 76: "Model" Class constructor Function when an object is created.

4- Each Model has its own method where the human face image is preprocessed according to the model requirements and applies the model weight on it to get the predictions and send it as a return to the main file where the methods are called.

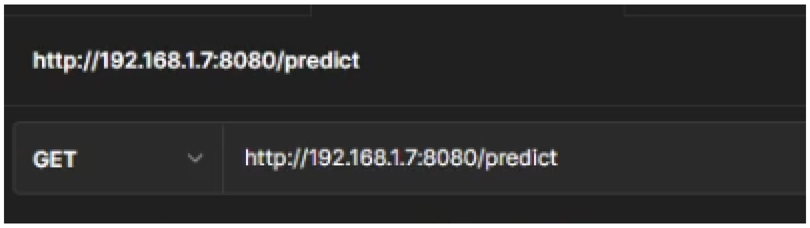


```python
def Emotions_Predict(self):
    Emotions = ["angry", "disgust", "fear", "happy", "neutral", "sad", "surprise"]
    Emo = cv2.cvtColor(self.img, cv2.COLOR_RGB2GRAY)
    Emo = cv2.resize(Emo, (48, 48))
    Emo = Emo.reshape(-1, 48, 48, 1)
    Emo = Emo / 255.0
    pred = self.Emotions_Model.predict(Emo)
    Pred_Emo = Emotions[np.argmax(pred)]
    return Pred_Emo

def Ethnicity_Predict(self):
    ethnicity_ranges = ["caucassian", "African", "Indian", "Asian", "middle east"]
    E_img = self.img.astype("float64")
    E_img = keras_vggface.utils.preprocess_input(E_img)
    E_img = cv2.resize(E_img, (224, 224))
    E_img = E_img.reshape(-1, 224, 224, 3)
    E_img = E_img / 255.0
    pred = self.Ethnicity_Model.predict(E_img)
    Ethnicity = ethnicity_ranges[np.argmax(pred)]
    return Ethnicity

def Age_Predict(self):
    age_ranges = ["1-2", "3-9", "10-20", "21-27", "28-45", "46-65", "66-116"]
    A_img = cv2.cvtColor(self.img, cv2.COLOR_RGB2GRAY)
    A_img = cv2.resize(A_img, (200, 200))
    A_img = A_img.reshape(-1, 200, 200, 1)
    pred = self.Age_Model.predict(A_img)
    age = age_ranges[np.argmax(pred)]
    return age

def Gender_Predict(self):
    G_img = cv2.resize(self.img, (128, 128))
    G_img = G_img.reshape(-1, 128, 128, 3)
    G_img = G_img / 255.0
    Gender_prediction = self.Gender_Model.predict(G_img)
    print(Gender_prediction[0][0])
    prediction = "Female"
    if Gender_prediction >= 0.5:
        prediction = "Male"
    return prediction
```

Figure 77: API predictions Methods.

5- After sending the predictions as reply to the requester the image is deleted from the static folder.

Example of sending a request message to the API Endpoint using Postman Application:



```
http://192.168.1.7:8080/predict

GET    http://192.168.1.7:8080/predict
```

Figure 78: Sending a request Message To the API Endpoint.

The Image sent by the postman:

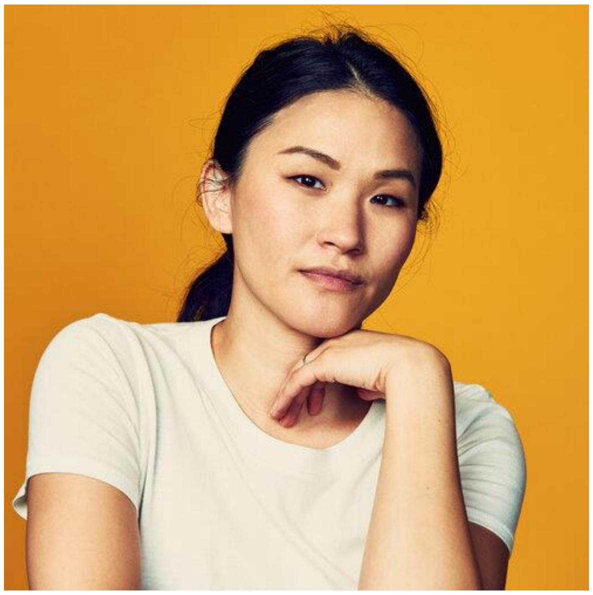
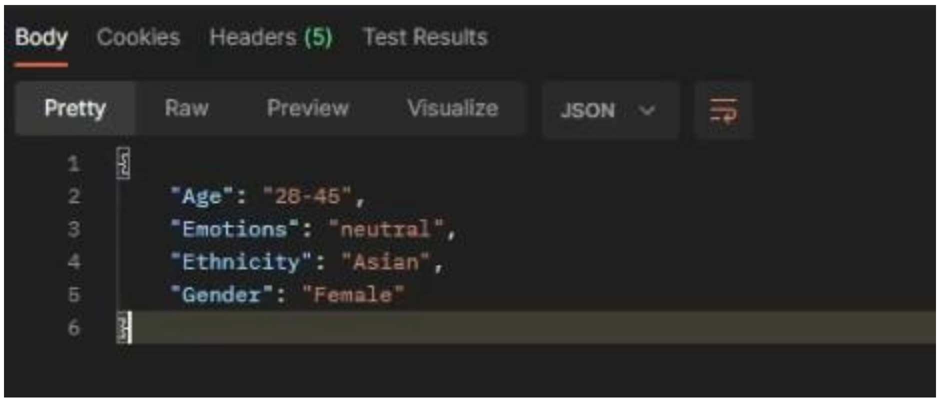


Figure 79: image sent by postman as request.

the replay from the API Endpoint:



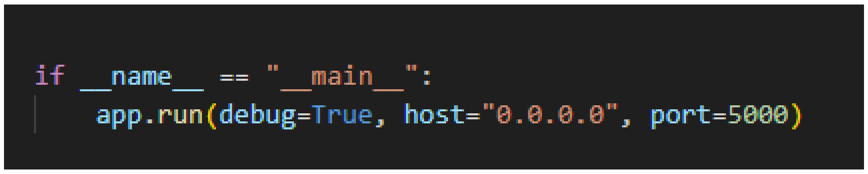
```
{
    "Age": "28-45",
    "Emotions": "neutral",
    "Ethnicity": "Asian",
    "Gender": "Female"
}
```

Figure 80: Reply from the API Endpoint to a Request

7- Web Application

The web application was developed using python Flask and Bootstrap.

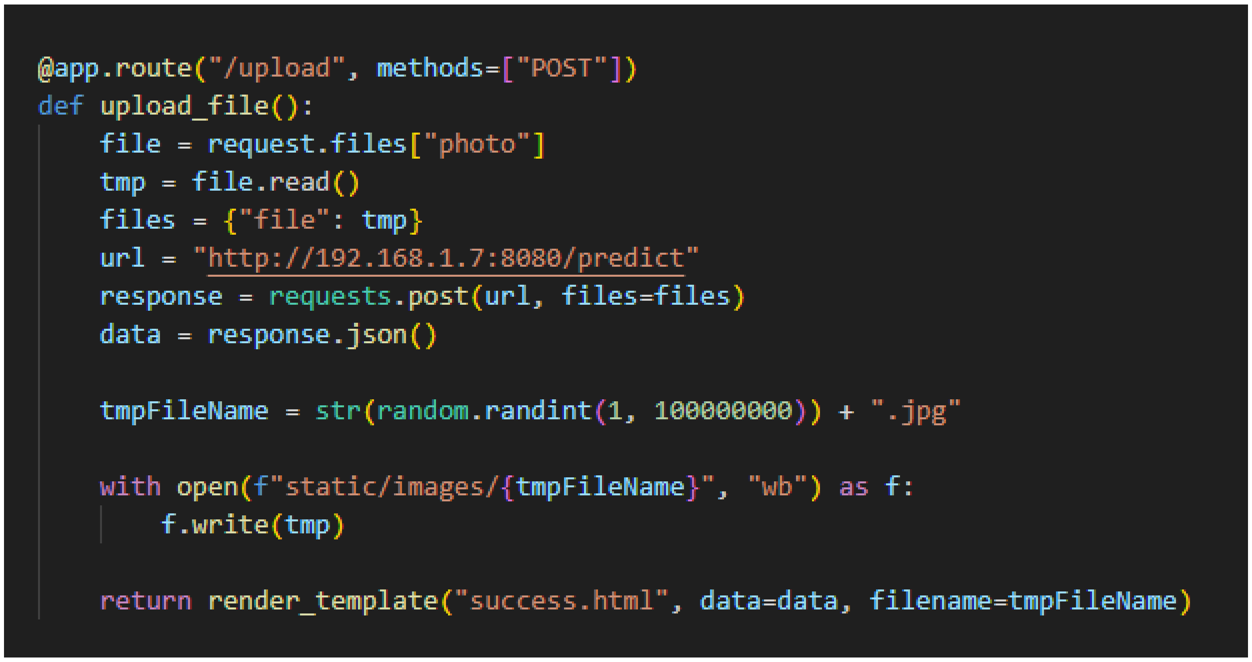
The Application Runs on Port 5000.



```python
if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=5000)
```

Figure 81: Web Application Port.

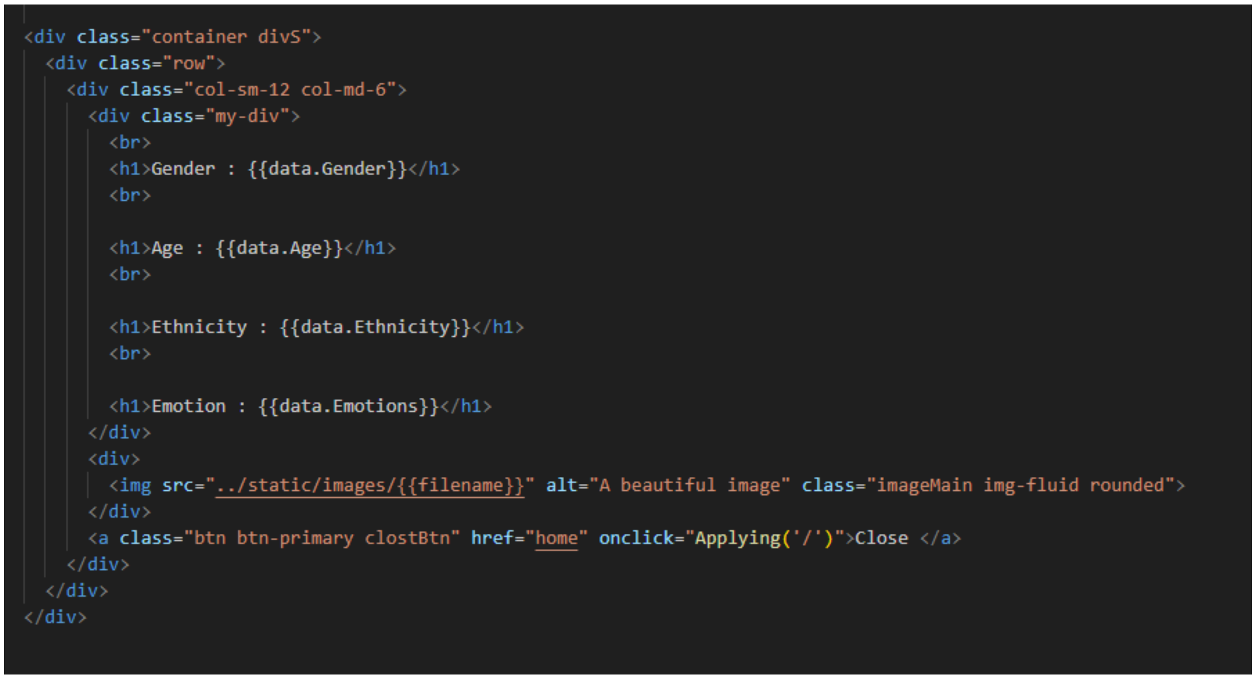
Through "/upload" Route user can upload an image and once uploaded, the application sends an API request to API application on "/Predict" Endpoint with the uploaded image.



```python
@app.route("/upload", methods=["POST"])
def upload_file():
    file = request.files["photo"]
    tmp = file.read()
    files = {"file": tmp}
    url = "http://192.168.1.7:8080/predict"
    response = requests.post(url, files=files)
    data = response.json()

    tmpFileName = str(random.randint(1, 100000000)) + ".jpg"

    with open(f"static/images/{tmpFileName}", "wb") as f:
        f.write(tmp)

    return render_template("success.html", data=data, filename=tmpFileName)
```

Figure 82: Web Application Upload Route.

Once the Web Application Gets the reply from API application, it loads a new html page with the received predictions.



```
<div class="container divS">
  <div class="row">
    <div class="col-sm-12 col-md-6">
      <div class="my-div">
        <br>
        <h1>Gender : {{data.Gender}}</h1>
        <br>

        <h1>Age : {{data.Age}}</h1>
        <br>

        <h1>Ethnicity : {{data.Ethnicity}}</h1>
        <br>

        <h1>Emotion : {{data.Emotions}}</h1>
      </div>
      <div>
        <img src="../static/images/{{filename}}" alt="A beautiful image" class="imageMain img-fluid rounded">
      </div>
      <a class="btn btn-primary clostBtn" href="home" onclick="Applying('/')">Close </a>
    </div>
  </div>
</div>
```

Figure 83: displaying predictions received in html page in the Web Application.

8- User Manual

The project is currently deployed in a local host environment which requires some extra steps and tools to use.
We intend to deploy it to the web in the future.

To use this project, you need to have those tools downloaded in your operating system:

1- Python Environment with TensorFlow, CV2, VGGFace, Flask, and Native Keras libraries.
2- Web Browser.
3- Models Weights in H5 Format.
4- Project Source Code.
5- IDE.

To start using the project simply open your IDE (in this example we use VSC). Navigate to where the project folder is located and click open Folder in VSC file tab.

After opening the project in VSC click "Ctrl+Shift+P" and type in the search "python: Select Interpreter" and choose your python environment where your libraries are installed.

After Selecting your Interpreter Go to Model.py file in VSC project navigation and Run the Code.

Now the API Application is ready to receive requests.

Open "Gray Scale" web application in VSC too.

Run the app.py File.

Now you have both the API application and Web Application Running in your local host.

Open your web browser and navigate to" http://192.168.1.9:5000".

You will see the web Application UI.

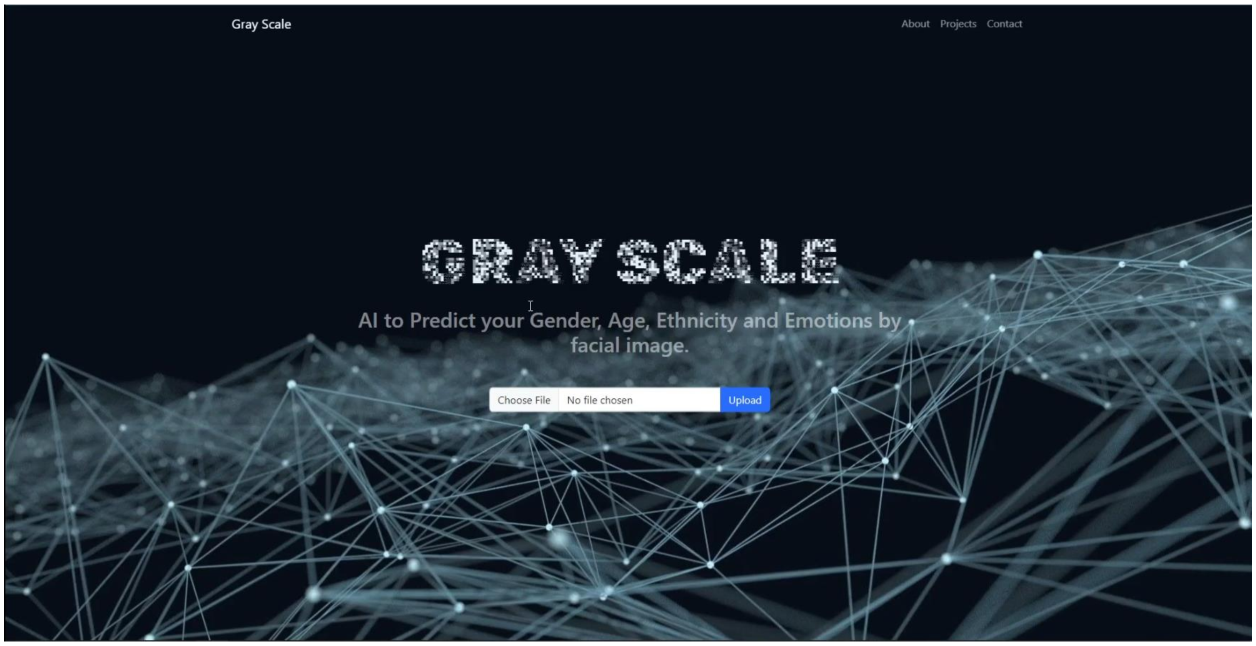
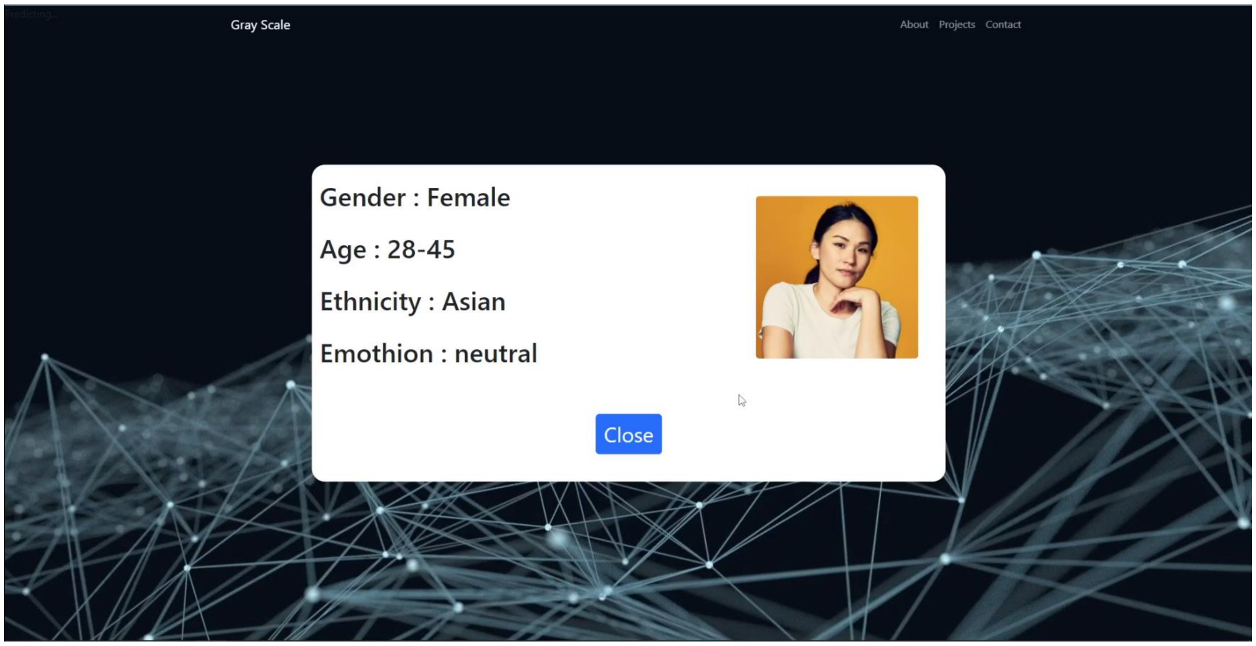

Gray Scale
About
Projects
Contact
GRAY SCALE
AI to Predict your Gender, Age, Ethnicity and Emotions by facial image.
Choose File
No file chosen
Upload

Figure 84: Web Application Main Page.

Click on Choose File and Choose any image in your disk with JPG, JPEG or PNG format and click Upload.

After the loading screen finishes, you can see the predictions.



Gender : Female

Age : 28-45

Ethnicity : Asian

Emothion : neutral

Figure 85: Predictions From the API.

9- Conclusion and Future Work

6.1 Conclusion

The Field of Computer Vision faces rabid changes right now with increasing number of promising research and an increasing availability of Big Datasets

The need for automated systems that detect and analyze human faces that are fast and accurate is increasing.
To address this increasing, we proposed our project “Human Face Detector and Analyzer.”

The “Human Face Detector and Analyzer” introduces models capable of detecting human face and predicting age, gender, ethnicity and emotions with responsible accuracy.

The “Human Face Detector and Analyzer” also provides an API Endpoint to integrate its services in other systems easily.

The “Human Face Detector and Analyzer” Age, Gender, Ethnicity and Emotion Models are implemented using TensorFlow and Keras

The Face Detection used is haar cascade algorithm from CV library.

The Web Application and the API Application are developed using Python Flask library.

Approaches used to build the models are CNN, Transfer Learning, and optimizations techniques like Data Augmentation and Adam Optimizer etc.

The models achieved the following results using the following datasets for training and testing:

Model	method	Dataset	Loss	Accuracy
Age Prediction	CNN	Facial Age and UTK-Face	0.75	71%
Gender Prediction	CNN	UTK-Face and B3FD	0.8463	81%
Emotions Prediction	CNN	FER-2013	0.9411	67%
Ethnicity Prediction	Transfer learning and CNN	Ethnicity Aware and Arab Celebrity Faces	0.3744	86%
Face Detection	State-of-Art pretrained Model	______	___	___

The datasets used are free datasets collected from various sources, so it has less quality and more misclassifications and noises than Private Datasets.

Providing good quality preprocessed datasets can increase the model's performance.

6.2 Future Work

In the future we are planning to increase our model performance and the project features by:

1- Try fine tuning more transfer learning models.
2- Try the Attention Models and Vision Transformers techniques.
3- Deploy the Project to the internet for global access using host providers.
4- Searching for more quality datasets to train the models.
5- Add more Models to predict more features like personality treats etc.
6- Enhance the Application UI
7- Add more security to the API Application.
8- Upload the project source code to the Internet and encourage more people to work to contribute to it as an open-source project.

References

[1] Nippon Datta Nippon , Juel Sikder Juel , "An Approach Based on Deep Learning for Recognizing Emotion, Gender and Age," 13 12 2022. [Online]. Available: https://assets.researchsquare.com/files/rs-2367553/v1_covered.pdf?c=1671159568.

[2] Simanjuntak, Frans,and George Azzopardi., "”Fusion of cnn-and cosfire-based features with application to gender recognition from face images.”," 2019. [Online].

[3] Wang, Xiaofeng, Azliza Mohd Ali, and Plamen Angelov, "Gender and age classification of human faces for automatic detection of anomalous human behaviour," 2017. [Online].

[4] Mollahosseini, Ali, David Chan, and Mohammad H. Mahoor., "going deeper in facial expression recognition using deep neural networks.," 2016. [Online].

[5] Agrawal, Abhinav, and Namita Mittal, "Using CNN for facial expression recognition: a study of the effect of kernal size and the number of filters on the accuracy.," 2020. [Online].

[6] Mohammad, A.S.,Al-Ani,J.A.,, "Convolutional neural network for ethnicity classification using ocular region in mobile environment.," 2018. [Online].

[7] Viola, Paul; Jones, Michael, "Rapid Object Detection using a Boosted Cascade of Simple Features," 2001. [Online].

[8] Alex Krizhevsky, Ilya Sutskever ,GeoffreyE.Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 2010. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[9] Awati, Rahul, "convolutional neural network (CNN)," 2021. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network.

[10] Karen Simonyan, Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 4 9 2014. [Online]. Available: https://arxiv.org/abs/1409.1556.

[11] Yaniv Taigman, Ming Yang ,Marc’ Aurelio Ranzato, Lior Wolf, "DeepFace:Closing the Gap to Human-Level Performance in FaceVerification," [Online]. Available: https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf.

[12] Florian Schroff, Dmitry Kalenichenko, James Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," 2015. [Online].

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, "Generative Adversarial Networks," 2014. [Online].

[14] Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980.

[15] Chollet, François, "Who is François Chollet.," [Online]. Available: https://en.wikipedia.org/wiki/Fran%C3%A7ois_Chollet.