

A  
Project Report on  
**Face Mask Detection using Python and ML.**

Submitted

In partial fulfillment of the requirements for the degree of

**Bachelor of Technology in  
Electrical Engineering**

Submitted by

**Mr. Aditya Arvind Desai**

At Softron, Kolhapur

Under the Guidance of

**Prof. Y. N. Bhosale**



**K.E. Society's**

Rajarambapu Institute of Technology, Rajaramnagar

(An Autonomous Institute, Affiliated to Shivaji University, Kolhapur)

Department of Electrical Engineering

2022-2023

K. E. Society's  
**Rajarambapu Institute of Technology, Rajaramnagar**  
(An autonomous Institute, Affiliated to Shivaji University)  
**Department of Electrical Engineering**

**CERTIFICATE**

This is to certify that the project under Industry Internship & Project (IIP) track completed at “Softtron Technologies, Kolhapur” is the bonafide work submitted by the following student, to the Rajarambapu Institute of Technology, Rajaramnagar during the academic year 2022-23, in partial fulfillment for the award of the degree of B. Tech in Electrical Engineering under our supervision. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree.

Name of Students	Roll Number
1. Aditya Arvind Desai	1908058

Date:

Place: Rajaramnagar

Prof. Y. N. Bhosale  
Industry Internship & Project  
Mentor(College)

Mr. Rohan S. Suryawanshi  
Industry Internship & Project  
Mentor(Industry)

External Examiner

Prof. Amarjeet Pandey  
Training & Placement Coordinator

Dr. V. N. Kalkhambkar  
Head of Department,

## DECLARATION

I declare that this report reflects my thoughts about the subject in my own words. I have sufficiently cited and referenced the original sources, referred or considered in this work. I have not plagiarized or submitted the same work for the award of any other degree. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute.

Sr. No.	Student Name	Roll No	Signature
1.	Aditya Arvind Desai	1908058	

Date:

Place:RIT, Rajaramnagar.

## ACKNOWLEDGEMENT

---

I take this opportunity to thank all those who have contributed in the successful completion of a Project Under Industry Internship & Project (IIP) track at “Softtron” I sincerely wish to express my gratitude to Industry Internship & Project (IIP) Mentor Prof. Y. N. Bhosale for full support, expert guidance, and encouragement and kind cooperation throughout the Internship work. I am greatly indebted to him for his help throughout project work. I express my sincere gratitude towards Dr. V. N. Kalkhambkar, Head of the Department, Electrical Engineering, for providing necessary facilities, guidance and support.

I respect and thank Mr. Rohan S. Suryawanshi for providing me an opportunity to do an Internship in Softtron and giving us all support and guidance which made me complete the internship duly. I am extremely thankful to [her/him] for providing such a nice support and guidance, although he had busy schedule managing the corporate affairs.

I thank Prof. Amarjeet Pandey for providing Internship & Project Opportunity in an Industry. I am thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of Electrical Engineering Department, which helped me in successfully completing internship.

Nevertheless, I express my gratitude toward my family members for their kind co-operation and encouragement which helped me in the completion of this internship.

## **Abstract**

Face mask detection has gained significant attention due to its importance in mitigating the spread of contagious diseases, such as COVID-19. In this paper, we propose a machine learning-based approach for face mask detection using the MTCNN (Multi-task Cascaded Convolutional Networks) algorithm, TensorFlow framework, and a trained model based on Convolutional Neural Networks (CNN).

The proposed system begins by utilizing MTCNN, a highly accurate face detection algorithm, to detect and extract faces from input images or video frames. These extracted faces are then fed into the trained CNN model, which has been specifically designed for face mask detection. The CNN model employs multiple layers of convolutional and pooling operations to automatically learn and extract relevant features from the input face images.

To train the CNN model, a large dataset consisting of labeled images of individuals wearing masks and individuals without masks is used. The dataset is carefully curated to encompass a wide range of facial appearances, orientations, and lighting conditions, ensuring the model's robustness in real-world scenarios. The model is trained using TensorFlow, a popular deep learning framework that provides efficient computational capabilities and convenient APIs for model development and training.

Experimental evaluations of the proposed approach demonstrate its effectiveness in accurately detecting whether an individual is wearing a face mask or not. The system achieves high accuracy and robustness, even in challenging scenarios with occlusions, variations in mask types, and different face angles.

The developed face mask detection system holds immense potential for deployment in various domains, including public spaces, airports, healthcare facilities, and workplaces, to enforce face mask compliance and ensure the safety of individuals. It can serve as an automated monitoring tool, contributing to the efforts of reducing the transmission of contagious diseases and safeguarding public health.

# INDEX

Certificate	ii
Declaration	iii
Acknowledgement	iv
Abstract	v
Contents	vi
List of Figures	viii
List of Tables	ix

<b>Chapter</b>	<b>Content</b>	<b>Page No.</b>
1	<b>Introduction</b>	9
1.1	Introduction	9
1.2	Problem Statement	10
1.3	Objective of project	10
1.4	Scope of Project	10
1.6	Organization of report	11
2	<b>Literature Review</b>	13
2.1	Literature Review	12
3	<b>Machine Learning</b>	14
3.1	Introduction to Machine Learning	14
3.2	Features of Machine Learning	15
3.3	Learning methods in Machine Learning	16
3.4	Models in Learning	18
3.5	Closure	30
4	<b>Methodology &amp; Working</b>	31

4.1	Methodology	31
4.2	Part 1: Create a training dataset	32
4.3	Part 2: Train an image classification model	33
4.4	Part 3: Part 3: Make predictions	35
5	<b>Libraries</b>	36
5.1	Numpy	36
5.2	Matplotlib	37
5.3	Pandas	38
5.4	read_csv()	39
5.5	Scikit-learn	41
5.6	Mtcnn	43
5.7	Closure	44
6	Result	45
6.1	Result	45
7	<b>Conclusion and Reference</b>	49
7.1	Conclusion	49
7.1	References	55
8	<b>Appendix</b>	51
8.1	Appendix - 1	51

## LIST OF FIGURES

Figure No	Figure Name	Page No.
3.3.1	Types of learning	16
3.3.2	Clustering of data	17
3.2.1	Linear regression of energy consumption	19
3.2.2	Classification of data	20
3.2.3	Clustering of data in different groups	21
3.2.4	Dimension Reduction	22
3.2.5	Neural Network	24
3.2.6	Deep Learning	24
4.1.1	Project Flow Diagram	32
5.4.1	Loading Data using read_csv()	40
5.4.2	Handling Different File Formats using read_csv()	40
5.4.3	Handling Missing Data using read_csv()	40
5.4.4	Customizing Data using read_csv()	41
6.1.1	Accuracy of the model	45
6.1.2	Input Image 1	45
6.1.3	Preprocessed input 1	46
6.1.4	Output example 1	46
6.1.5	Input image 2	47
6.1.6	Preprocessed input 2	47
6.1.7	Output image 2	48



## Introduction

### 1.1 Introduction

The rapid spread of contagious diseases, such as COVID-19, has necessitated the adoption of preventive measures to mitigate their transmission. Wearing face masks has emerged as a crucial practice in reducing the risk of infection, prompting the need for reliable face mask detection systems. These systems can be deployed in various settings, including public spaces, healthcare facilities, and workplaces, to ensure compliance with face mask mandates and protect public health.

Machine learning techniques have shown remarkable success in computer vision tasks, making them a promising approach for face mask detection. In this paper, we propose a machine learning-based approach that combines the power of the MTCNN algorithm, the TensorFlow framework, and a trained Convolutional Neural Network (CNN) model to accurately detect whether individuals are wearing face masks.

The first stage of our proposed system involves the use of the MTCNN algorithm, which has proven to be highly accurate in face detection. MTCNN detects and extracts faces from input images or video frames, providing the necessary inputs for subsequent analysis. This ensures that only relevant regions, i.e., faces, are considered in the face mask detection process.

Next, the extracted face regions are passed through a trained CNN model. CNNs are a class of deep neural networks that excel at learning hierarchical representations of data. Our CNN model is specifically designed and trained to recognize the presence or absence of face masks. By leveraging the inherent capability of CNNs to extract discriminative features, the model learns to make accurate predictions based on the visual characteristics of the face images.

To train the CNN model, a diverse and comprehensive dataset is assembled, consisting of labeled images of individuals wearing face masks and individuals without masks. This dataset encompasses various factors, such as different mask types, lighting conditions, and facial orientations, to ensure the model's robustness and generalization to real-world scenarios. TensorFlow, a widely adopted deep learning framework, is employed for efficient model development, training, and optimization.

The effectiveness of our proposed approach is evaluated through extensive experiments, assessing its accuracy and robustness under different conditions. The results demonstrate the system's ability to accurately identify individuals wearing face masks, even in challenging scenarios with partial face occlusions and varying face angles. The developed system holds significant potential for deployment in public health initiatives, acting as an automated monitoring tool to enforce face mask compliance and enhance public safety.

In conclusion, the proposed machine learning-based approach for face mask detection, utilizing MTCNN, TensorFlow, and a trained CNN model, presents an effective solution to address the need for automated monitoring of face mask compliance. By leveraging the power of machine learning and computer vision, this system contributes to the collective efforts of reducing the spread of contagious diseases and safeguarding public health in various domains.

## **1.2 Problem Statement**

The widespread adoption of face masks as a preventive measure against contagious diseases presents a challenge in ensuring compliance. Manual monitoring of face mask usage is time-consuming and error-prone. Existing approaches using traditional computer vision techniques struggle with variations in lighting, mask types, and facial orientations, lacking scalability and accuracy. Therefore, there is a need for an advanced, automated face mask detection system that can accurately identify individuals without face masks in real-time, overcoming these limitations..

## **1.3 Objectives**

This research aims to develop a machine learning-based approach for face mask detection, leveraging MTCNN for face detection and CNN for mask classification. The goal is to create a system that efficiently and accurately detects individuals without face masks, regardless of lighting conditions, mask types, and facial orientations. Training a CNN model with a curated dataset using TensorFlow will optimize its performance and ensure real-time detection, offering an effective solution for automated face mask compliance monitoring.

## **1.4 Scope of Project.**

The scope of this project encompasses the development of a face mask detection system using machine learning techniques. The project will focus on the following key components:

**Data Collection:** Curating a diverse and representative dataset of labeled images containing individuals both wearing and not wearing face masks. The dataset will encompass various lighting conditions, mask types, and facial orientations.

**Face Detection:** Implementing the MTCNN algorithm to detect and extract faces from input images or video frames. This step ensures that only relevant regions (i.e., faces) are considered for mask detection.

**Model Training:** Designing and training a Convolutional Neural Network (CNN) model using TensorFlow. The model will be trained on the curated dataset to learn and extract features related to face mask presence or absence.

**Mask Classification:** Utilizing the trained CNN model to classify extracted face regions as either wearing a face mask or not. This step will involve feeding the face images through the CNN model and obtaining predictions.

**Real-time Detection:** Implementing the face mask detection system to perform real-time detection on live video streams or image inputs. The system should be capable of processing frames in real-time and providing accurate results promptly.

**Performance Evaluation:** Assessing the accuracy, robustness, and efficiency of the developed system through extensive experiments and performance metrics. This evaluation will involve testing the system on various scenarios, including different lighting conditions, mask types, and facial orientations.

**Deployment Considerations:** Discussing the considerations for deploying the face mask detection system in practical settings, such as integration with existing surveillance systems or mobile applications. Addressing potential challenges related to scalability, computational requirements, and system integration.

It is important to note that this project's scope focuses on the technical development and evaluation of the face mask detection system using machine learning techniques. The integration of the system into specific applications or hardware platforms is beyond the scope and may require additional implementation and customization.

## **1.5 Organization of Report**

**Chapter 1** talks about the brief information about the project. This includes introduction importance of the problem, problem statement, objective, and scope of the project.

**Chapter 2** represents literature survey from different research papers also gives overview of different projects done before.

**Chapter 3** represents the introduction to machine learning, features of machine learning and the type of learnings included.

**Chapter 4** represents methodology along with its working.

**Chapter 5** represents libraries used in the project..

**Chapter 6** represents Future Scope, Conclusion and references.

**Chapter 7** represents Conclusion and references

**Appendix :** represents code for the project.

### Literature Review

#### 2.1 Literature Review

1. "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks" (2016):

This paper proposes the Multitask Cascaded Convolutional Networks (MTCNN) algorithm, which simultaneously addresses face detection and facial landmark alignment. It introduces a cascaded architecture with three stages and achieves high accuracy on face detection benchmarks.

2. "Finding Tiny Faces" (2017):

This paper focuses on detecting small faces in images and introduces a novel algorithm that effectively handles the challenge of tiny face detection. It demonstrates superior performance on datasets containing images with tiny faces.

3. "Face Attention Network: An Effective Face Detector for Occluded Faces" (2017):

This paper presents a Face Attention Network (FAN) that is specifically designed to detect and handle occluded faces. It achieves robust face detection performance, even in scenarios where faces are partially obscured or occluded.

4. "Towards Fast, Accurate and Stable 3D Dense Face Alignment" (2018):

This paper proposes a 3D Dense Face Alignment method that aims to accurately align facial landmarks in real-world, unconstrained conditions. It achieves high accuracy while maintaining fast and stable performance.

5. "Face Alignment in Full Pose Range: A 3D Total Solution" (2019):

This paper presents a comprehensive solution for 3D face alignment across various poses. It introduces a method that can accurately estimate the 3D shape and pose of a face, enabling robust face alignment in diverse pose ranges.

### Machine Learning

#### 3.1 Introduction to Machine Learning

Machine learning is a branch of artificial intelligence (AI) that focuses on developing algorithms and models to enable computers to learn from data and make predictions or decisions. Unlike traditional programming, machine learning allows computers to learn from examples or patterns in the data without being explicitly programmed. The core concept of machine learning is to analyze large amounts of data, identify patterns and relationships, and use that knowledge to make accurate predictions or take actions. By leveraging statistical and mathematical techniques, machine learning algorithms can uncover hidden insights that may not be apparent to humans.

There are various types of machine learning algorithms, including supervised learning, unsupervised learning, and reinforcement learning. Supervised learning involves training algorithms with labeled examples, enabling them to make predictions or classifications on unseen data. Unsupervised learning, on the other hand, involves finding patterns and structures in unlabeled data. Reinforcement learning involves an agent learning from interactions with an environment and optimizing its actions to maximize rewards. Common machine learning algorithms include linear regression, logistic regression, decision trees, support vector machines, neural networks, and deep learning. These algorithms are trained on large datasets, and their parameters are adjusted through optimization techniques to improve performance.

Machine learning has diverse applications across various domains such as computer vision, natural language processing, speech recognition, recommendation systems, fraud detection, and autonomous vehicles. It has transformed industries by automating tasks, providing predictions, and generating valuable insights from data analysis.

In summary, machine learning is an integral part of AI that enables computers to learn from data, adapt to new situations, and make intelligent decisions. Its ability to uncover complex patterns and make accurate predictions has made it an essential tool in numerous fields. It's worth noting that the interpretability and explainability of machine learning models are areas of ongoing research. As models become more complex, understanding the reasoning behind their predictions or decisions becomes increasingly challenging. Researchers are actively working on developing methods to make machine learning models more transparent and accountable.

Machine learning continues to revolutionize industries and fields, driving advancements in healthcare, finance, marketing, cybersecurity, and more. It empowers systems to automate tasks, detect patterns, and make data-driven decisions, ultimately enhancing efficiency and enabling innovative solutions to complex problems.

### 3.2 Features of Machine Learning

Machine learning possesses several distinct features:

1. **Data-Driven Technology:** Machine learning operates on the foundation of data. The vast amounts of data generated by organizations daily can be utilized to uncover meaningful relationships and patterns. By leveraging these insights, organizations can make more informed decisions.
2. **Autonomous Learning:** A key attribute of machine learning is its ability to autonomously learn and improve. Machine learning algorithms can self-adjust and refine their models or predictions based on new information. This capability enables systems to continually enhance their performance without human intervention.
3. **Pattern Recognition:** Machine learning algorithms excel at detecting patterns within data. Through the analysis of extensive datasets, they can identify intricate relationships, trends, and correlations that may elude human observation. By uncovering these patterns, organizations can derive valuable insights and make data-driven decisions.
4. **Targeted Marketing:** Machine learning plays a vital role in assisting organizations with targeted marketing efforts. By analyzing customer data, such as preferences, behavior, and demographics, machine learning algorithms can identify patterns and trends. This enables organizations to develop personalized and effective marketing strategies, resulting in improved customer engagement and enhanced brand positioning.
5. **Relationship to Data Mining:** Machine learning and data mining share a close relationship. While data mining involves extracting knowledge and insights from data, machine learning goes a step further by allowing systems to learn from data and make predictions or decisions based on that learning. Both fields deal with large datasets, but machine learning's focus lies in enabling systems to autonomously learn and adapt.

### 3.3 Learning methods in Machine Learning

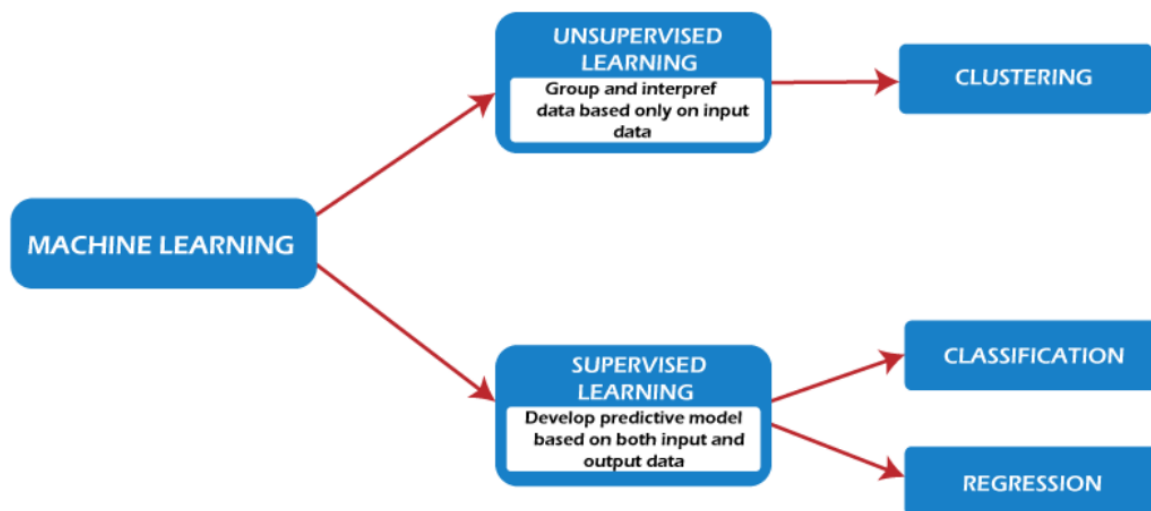


Fig 3.3.1 Types of learning

#### 1. Supervised learning

Supervised learning is a machine learning approach that involves training a model using labeled data. In supervised learning, the input data is associated with corresponding output labels or responses. The objective is to create a model that can make accurate predictions or classifications on new, unseen data based on the patterns learned from the labeled examples.

There are two primary types of supervised learning techniques: classification and regression.

1. **Classification:** Classification techniques are used to predict discrete or categorical labels. The algorithm learns from labeled examples and assigns new, unseen instances to predefined classes or categories. For instance, classifying emails as spam or not spam, or determining whether a tumor is cancerous or benign. Classification finds applications in diverse domains like medical diagnostics, image recognition, and sentiment analysis. Common classification algorithms include support vector machines (SVMs), decision trees, random forests, k-nearest neighbors (KNN), Naive Bayes, logistic regression, and neural networks.
2. **Regression:** Regression involves predicting continuous numerical values or estimating a function that maps inputs to outputs. It is used to model relationships between variables and make predictions based on those relationships. Regression is commonly used in tasks such as sales forecasting, price prediction, and weather forecasting. Regression algorithms



aim to find the best-fit line or curve that minimizes the difference between predicted and actual values. Common regression algorithms include linear regression, polynomial regression, support vector regression (SVR), decision trees, random forests, and neural networks.

To perform supervised learning, it is crucial to have a well-labeled training dataset, where the input features are associated with their corresponding output labels. The model learns from this labeled data and generalizes its knowledge to make predictions on new, unseen data. Supervised learning is a powerful approach in machine learning that enables accurate prediction and classification tasks. It finds widespread applications across industries where historical labeled data is available to train models and make informed decisions based on learned patterns.

## 2. Unsupervised learning

Unsupervised learning is a branch of machine learning that focuses on detecting hidden patterns or structures within unlabeled data. Unlike supervised learning, which relies on labeled data, unsupervised learning operates on datasets without predefined output labels.



**Fig 3.3.2 Clustering of data**

One common technique used in unsupervised learning is clustering. Clustering is employed for exploratory data analysis to identify natural groupings or clusters within the data. It has diverse applications, such as gene sequence analysis, market research, and commodity identification.

For example, let's consider a cell phone company aiming to optimize tower placement for improved signal reception. In this scenario, unsupervised learning can be leveraged to predict the number of people served by each tower. Since a phone can connect to only one

tower at a time, the company's team can use clustering algorithms to determine the optimal placement of cell towers, maximizing signal reception for different groups or clusters of customers.

Unsupervised learning provides organizations with valuable insights into their data by uncovering hidden patterns and structures. It serves as a foundation for further analysis and decision-making processes. By discovering meaningful groupings and relationships within the data, businesses can make informed decisions, enhance their operations, and drive innovation.

Common algorithms for performing clustering are k-means and k-medoids, hierarchical clustering, Gaussian mixture models, hidden Markov models, self-organizing maps, fuzzy C-means clustering, and subtractive clustering.

### **3.4 Models in learning**

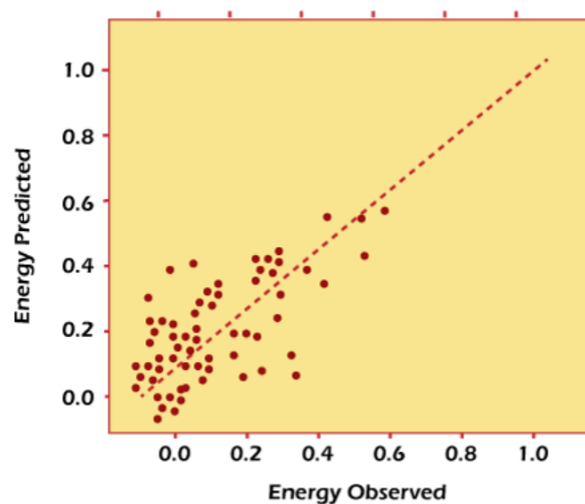
- Regression
- Classification
- Clustering
- Dimensionality Reduction
- Ensemble Methods
- Neural Nets and Deep Learning
- Transfer Learning
- Reinforcement Learning
- Natural Language Processing
- Word Embedding's

#### **1. Regression**

Regression is a supervised machine learning technique that is used to predict or interpret numerical values based on historical data. It is commonly employed to forecast prices of assets by analyzing past pricing data for similar properties, among other applications.

A fundamental method in regression is linear regression, which involves modeling the dataset using a linear equation, typically represented as  $y = m * x + b$ . The goal is to find the best-fitting line that minimizes the overall distance between the data points and the line. This is achieved by determining the slope (m) and the y-intercept (b) that define the line.

To illustrate, let's consider an example of predicting the energy consumption (in kW) of buildings. By gathering data on various factors such as building age, number of stories, square footage, and the number of wall devices plugged in, a multivariable linear regression model can be used. In this case, the model creates a line in a multi-dimensional space, considering multiple input variables.



**Fig 3.4.1 Linear regression for energy consumption**

Now, imagine having information about the characteristics of a building (age, square footage, etc.), but lacking knowledge of its energy consumption. Using the fitted line from the linear regression model, it is possible to estimate the energy consumption for that particular building. The accuracy of the model can be evaluated by comparing its predictions with the actual energy consumption, as depicted in the plot below.

Moreover, linear regression allows for assessing the weight or importance of each factor contributing to the final prediction of energy consumption. By analyzing the regression formula, it becomes possible to determine the relative significance of factors such as age, size, or height in influencing energy consumption.

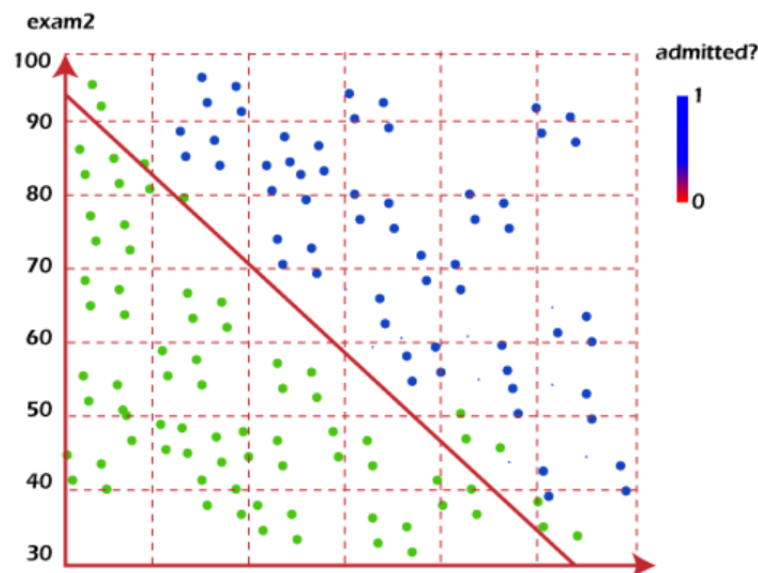
Regression techniques run the gamut from simple (linear regression) to complex (regular linear regression, polynomial regression, decision trees, random forest regression, and neural nets).

## 2. Classification

In supervised machine learning, classification methods are used to predict or assign class labels to data instances. These methods are valuable in various applications, such as predicting whether an online customer will make a purchase or determining the presence of

specific objects in images. Classification problems can involve binary classification (two classes) or multi-class classification (more than two classes).

One of the commonly used classification algorithms is logistic regression, despite its name suggesting regression. Logistic regression estimates the probability of an event occurrence based on one or more input variables.



**Fig 3.4.2 Classification of data**

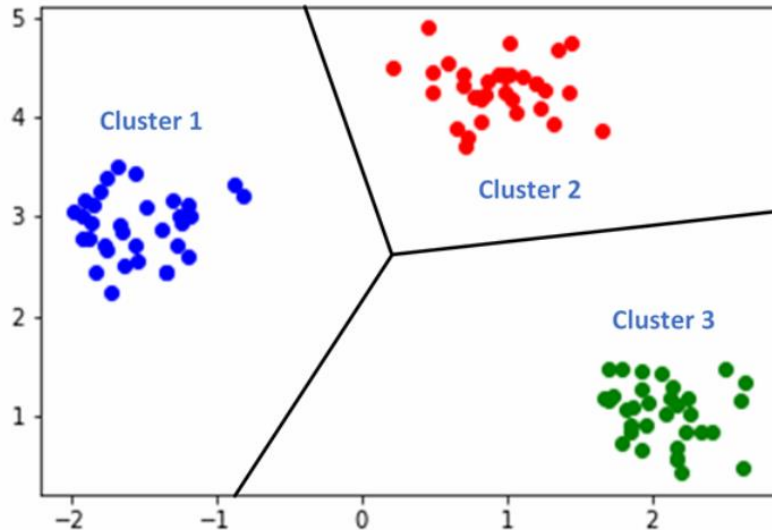
For instance, consider the task of predicting college admissions based on student test scores. Logistic regression can take into account multiple factors and assign a probability of admission. If the predicted probability is above a certain threshold (usually 0.5), the student is predicted to be admitted; otherwise, they are predicted to be rejected.

The decision boundary, which separates the predicted classes, can be visualized in a chart by plotting the marks of past students and their admission status. As you progress in machine learning, you can explore more advanced classification techniques. Nonlinear classifiers, such as decision trees, random forests, support vector machines, and neural networks, offer more flexibility in capturing complex relationships and can provide improved accuracy in classification tasks. These techniques enable the creation of sophisticated decision boundaries and can handle diverse datasets with varying degrees of complexity.

### 3. Clustering

Clustering falls under the category of unsupervised machine learning techniques, aimed at grouping or clustering observations with similar characteristics. Unlike supervised learning,

clustering methods do not rely on labeled output information for training. Instead, they allow the algorithm to define the output by identifying inherent patterns and similarities within the data. In clustering, the evaluation of the solution quality often involves visualizing the results.



**Fig 3.4.3 Clustering of data in different groups**

One widely used clustering method is K-means, where the user specifies the number of clusters, denoted as "K". (Note that various techniques exist for determining the optimal value of K, such as the elbow method.)

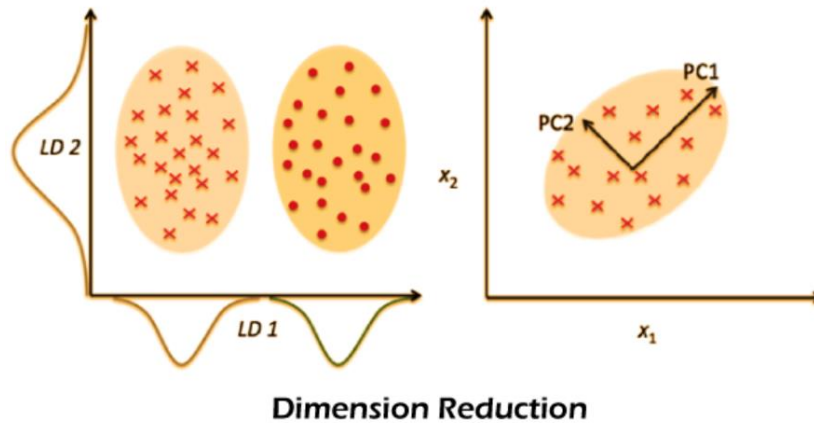
The K-means algorithm follows these steps:

1. Randomly initialize K centers within the data.
2. Assign each data point to the nearest randomly chosen center.
3. Recalculate the centers based on the assigned data points.
4. If the centers do not change significantly (or change very little), the process is considered complete.
5. Otherwise, repeat steps 2-4 until convergence is achieved. To avoid potential infinite loops, a maximum number of iterations can be predefined.

In the next plot, K-means clustering is applied to a dataset of buildings. The dataset includes four measurements related to air conditioning, plug-in appliances (e.g., microwave, refrigerator), household gas usage, and heating gas usage. Each column in the plot represents the efficiency of a particular building.

By employing K-means clustering to group buildings with similar characteristics, it becomes possible to gain insights and analyze the data more effectively. This clustering approach facilitates the identification of underlying patterns and structures in the dataset, thereby enabling better understanding and interpretation of the data.

### 3. Dimensionality Reduction



**Fig 3.4.4 Dimension Reduction**

Dimensionality reduction techniques are utilized to reduce the number of features or variables in a dataset by eliminating less relevant or redundant information. This process is particularly useful when dealing with high-dimensional data, such as images with thousands of pixels or datasets with numerous measurements or attributes that may contain redundant or irrelevant information.

One commonly used dimensionality reduction method is Principal Component Analysis (PCA). PCA aims to transform the original features into a new set of linearly uncorrelated variables called principal components. These components are ordered based on their variance, with the first few components capturing the most significant information in the data. By selecting a subset of the principal components, PCA allows for a reduction in dimensionality while retaining a large portion of the original information.

Another popular technique is t-distributed Stochastic Neighbor Embedding (t-SNE), which is effective in visualizing high-dimensional data in a lower-dimensional space. t-SNE focuses on preserving local relationships between data points, making it useful for visualizing clusters or groups within the data.

In the following plot, we demonstrate the application of dimensionality reduction techniques to the MNIST database of handwritten digits. This dataset consists of thousands of images representing digits from 0 to 9, which is commonly used for testing clustering and

classification algorithms. Each row in the dataset corresponds to a vectorized version of the original image, typically represented by a high-dimensional feature space. By reducing the dimensionality from a high-dimensional space (e.g., 784 pixels) to a lower-dimensional space (e.g., 2 dimensions for visualization), we can effectively visualize and analyze the dataset in a more interpretable manner. This allows us to gain insights into the underlying structure and relationships within the dataset.

## **5. Ensemble Methods**

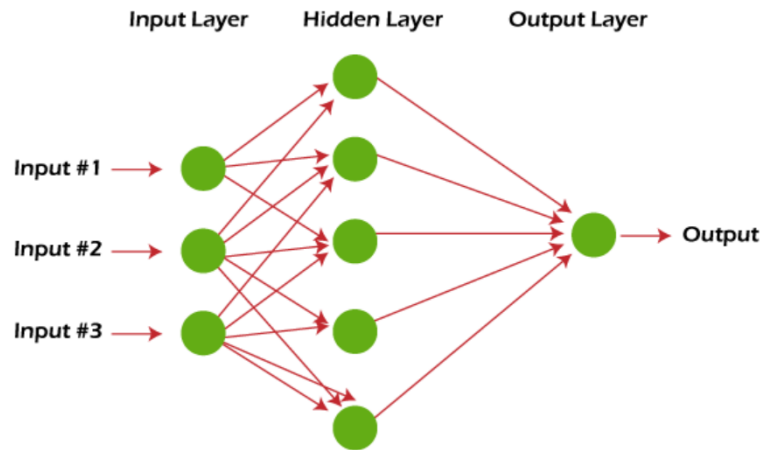
Ensemble methods involve combining multiple predictive models to achieve better overall performance and accuracy than any individual model. An ensemble model leverages the collective intelligence of diverse models to make more robust and reliable predictions.

For example, the Random Forest algorithm is a popular ensemble method that combines multiple decision trees trained on different subsets of the dataset. By aggregating the predictions of these individual trees, Random Forest can provide more accurate and stable predictions compared to a single decision tree. Ensemble methods are designed to address the limitations of a single model by reducing bias and variance. Each model in the ensemble contributes its unique perspective and learns from different aspects of the data. By combining their predictions, the ensemble model achieves a more balanced and accurate result.

In the field of machine learning competitions, such as those held on platforms like Kaggle, ensemble methods have proven to be highly effective. Many winning solutions utilize ensemble techniques, including algorithms like Random Forest, XGBoost, and LightGBM. Ensemble methods offer a powerful approach to enhance the performance and robustness of machine learning models. By leveraging the strengths of multiple models, they can capture a broader range of patterns and make more reliable predictions, making them a valuable tool in the data scientist's toolkit.

## **6. Neural Networks and Deep Learning**

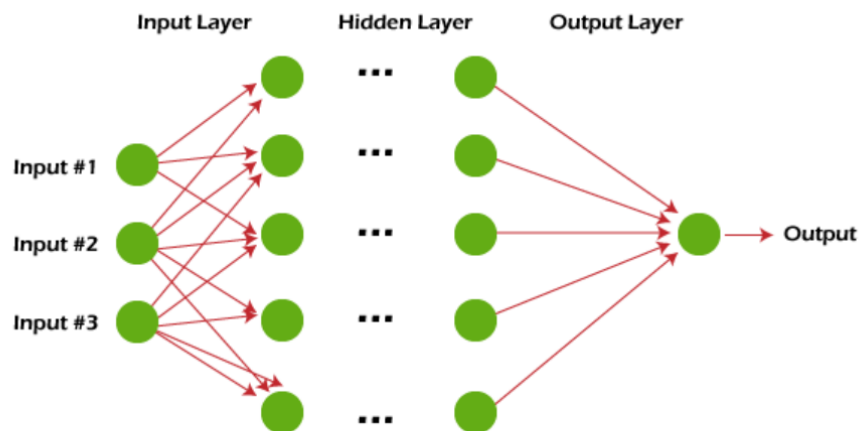
Neural networks and deep learning are powerful techniques for capturing complex patterns in data by leveraging the interconnected layers of parameters. Unlike linear models such as linear regression or logistic regression, neural networks have the ability to model nonlinear relationships.



*Neural network with a hidden layer.*

**Fig 3.4.5 Neural Network**

A neural network typically consists of an input layer, one or more hidden layers with adjustable parameters, and an output layer. By iteratively adjusting the parameters using training data, neural networks can learn to make accurate predictions and classifications.



*Deep learning: A neural network with multiple hidden layers.*

**Fig 3.4.6 Deep Learning**

Deep learning refers to neural networks with multiple hidden layers. These deep neural networks have demonstrated remarkable performance in various domains, including computer vision, natural language processing, and audio analysis.

Staying updated with the latest developments in deep learning can be challenging, as research and industry communities are constantly pushing the boundaries with new methods and architectures. Deep learning techniques require substantial amounts of data and



computational power, often relying on specialized hardware like Graphical Processing Units (GPUs). Deep learning has achieved significant success in tasks such as image classification, text analysis, speech recognition, and video processing. Popular deep learning frameworks like TensorFlow and PyTorch provide tools and libraries to implement and explore deep learning models.

It is crucial for practitioners to keep pace with the advancements in deep learning and leverage these techniques to tackle complex problems and extract meaningful insights from data. By harnessing the power of neural networks and deep learning, researchers and practitioners can unlock new possibilities and push the boundaries of artificial intelligence.

## **7. Transfer Learning**

Transfer learning is a valuable technique in data science that involves utilizing existing knowledge from one task to improve the performance of a related task. For example, let's say you have successfully trained a neural network to classify images into categories such as shirts, t-shirts, and polos. Now, you have a new task of building a model to classify different types of pants, such as jeans, cargo pants, casual pants, and dress pants.

The concept of transfer learning allows you to leverage the knowledge gained from training the shirt classification model and apply it to the pants classification task. Instead of starting from scratch, you can reuse a portion of the pre-trained neural network and combine it with additional layers specific to the new task. By doing so, the model can quickly adapt to the nuances of the pants classification problem. This approach offers several advantages, including reduced data requirements for training the neural network. Deep learning algorithms often require a large amount of labeled data, which can be costly and time-consuming to obtain. Transfer learning mitigates this challenge by leveraging the knowledge already captured in the pre-trained model.

In the context of the example, you can reuse a significant portion of the layers from the shirt model, such as 18 out of 20 hidden layers, and add a new layer specifically designed for the pants classification task. This way, the model can benefit from the shared knowledge and learn the unique aspects related to pants classification.

Transfer learning has gained popularity due to the availability of pre-trained models for various deep learning tasks, such as image and text classification. These pre-trained models offer a starting point and provide valuable insights gained from large-scale datasets and extensive training.

By utilizing transfer learning, data scientists can save time, computational resources, and effort while achieving competitive performance on new tasks. It is a powerful technique that enables the application of deep learning in scenarios where limited labeled data is available or when there is a need for quick adaptation to new tasks.

## **8. Reinforcement Learning:**

Reinforcement learning is a branch of machine learning that focuses on training an agent to make sequential decisions in an environment to maximize a cumulative reward. It can be understood through the analogy of a mouse in a maze searching for hidden pieces of cheese.

In reinforcement learning, the agent, represented by the mouse, interacts with an environment, which is the maze, and takes actions to navigate through it. The goal of the agent is to find the cheese, which serves as the reward. Through trial and error, the agent learns which actions lead to positive outcomes and maximizes the cumulative reward over time. Reinforcement learning is particularly useful in scenarios where there is limited or no historical data available for training. Unlike traditional machine learning methods, reinforcement learning does not rely on prior information but learns from its own experiences. This makes it suitable for domains that require exploration and learning from interactions.

One notable application of reinforcement learning is in games, especially those with "perfect information" such as chess and Go. In these games, feedback from the environment and the agent's actions is readily available, allowing the model to learn and improve quickly. However, it is important to note that reinforcement learning can be computationally intensive and time-consuming, especially for complex problems that require extensive training. Reinforcement learning offers a powerful framework for training intelligent agents to make sequential decisions in dynamic environments. It combines exploration and exploitation to learn optimal strategies and has applications in various fields, including robotics, control systems, and more.

## **9. Natural Language Processing (NLP)**

Natural Language Processing (NLP) is a field of study that focuses on the interaction between computers and human language. It involves the development of algorithms and techniques to enable machines to understand, interpret, and generate natural language.

NLP techniques are widely used for text preprocessing in machine learning tasks. When working with text data in different formats, such as Word documents or online blogs, it is

necessary to apply NLP techniques to clean and prepare the text before using it in machine learning models. These techniques involve tasks like handling typos, missing characters, and filtering out irrelevant words.

One commonly used NLP package is NLTK (Natural Language Toolkit), which was developed by researchers at Stanford University. NLTK provides a range of tools and functionalities for processing and analyzing natural language data. A common approach to represent text numerically is by counting the frequency of words in a text document. This results in a matrix representation called the Term Frequency Matrix (TFM), where each row represents a document and each column represents a word. Another technique called Term Frequency-Inverse Document Frequency (TF-IDF) is often used, which not only considers word frequency but also weighs the importance of words in the entire document collection. TF-IDF is generally preferred for machine learning tasks as it captures the significance of words in a more meaningful way.

NLP has various applications, including text classification, sentiment analysis, information extraction, machine translation, and question answering systems. It plays a crucial role in enabling machines to understand and process human language, facilitating natural and intuitive interactions with technology.

## **10. Computer Vision**

Computer vision is a field of study that focuses on enabling computers to gain a high-level understanding of digital images or videos, similar to how humans interpret and perceive visual information. It involves the development and application of algorithms and techniques to extract meaningful information from visual data.

The goal of computer vision is to enable machines to perceive and interpret visual data, such as images or videos, and make intelligent decisions or take actions based on that understanding. It involves tasks such as image recognition, object detection and tracking, image segmentation, scene understanding, 3D reconstruction, and more.

Computer vision algorithms typically involve the following steps:

Image acquisition: Gathering visual data using cameras, sensors, or other imaging devices.

Preprocessing: Enhancing or cleaning up the acquired images through operations like noise removal, image denoising, image resizing, and color correction.

Feature extraction: Identifying and extracting distinctive visual features from the images, such as edges, corners, textures, or keypoints.

Feature matching: Comparing and matching extracted features across different images or frames to establish correspondences.

Object detection and recognition: Identifying and localizing specific objects or patterns of interest within an image or video sequence.

Scene understanding: Analyzing the overall scene and its context, including the relationships between different objects and their spatial arrangement.

Tracking: Continuously following and locating objects of interest across successive frames in a video sequence.

Machine learning and deep learning: Utilizing machine learning techniques, including deep neural networks, to train models that can learn from labeled data and make predictions or classifications on new, unseen data.

Computer vision has numerous practical applications across various fields, including autonomous vehicles, surveillance systems, robotics, medical imaging, augmented reality, facial recognition, quality control, and many more. Its advancements have opened up new possibilities for machines to perceive and understand visual information, enabling them to interact with the visual world in a more intelligent and human-like manner.

## **11. Open CV**

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a wide range of tools and functions that can be used to develop computer vision applications.

Here are some key features and functionalities of OpenCV:

1. Image and video processing: OpenCV allows you to read, write, and manipulate images and videos. It provides various functions for tasks such as resizing, cropping, filtering, blending, and color space conversion.
2. Object detection and tracking: OpenCV includes methods for object detection and tracking, such as Haar cascades, HOG (Histogram of Oriented Gradients), and deep learning-based approaches. These algorithms can be used to detect and track objects in real-time or on static images.
3. Camera calibration: OpenCV provides tools for camera calibration, which involves estimating the intrinsic and extrinsic parameters of a camera. Calibration is essential for tasks like 3D reconstruction, object measurement, and augmented reality.
4. Feature detection and extraction: OpenCV includes algorithms for feature detection and extraction, such as SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), and ORB (Oriented FAST and Rotated BRIEF). These methods can identify keypoints and descriptors that can be used for image matching and recognition.
5. Machine learning integration: OpenCV integrates with popular machine learning libraries like TensorFlow and PyTorch. It provides support for training and deploying deep learning models, including pre-trained models for tasks like image classification, object detection, and semantic segmentation.
6. GUI and computer vision utilities: OpenCV offers functions for creating graphical user interfaces (GUIs) and displaying images or videos. It also includes utilities for image/video I/O, file handling, and basic geometric transformations.

OpenCV is implemented in C++, but it provides interfaces for various programming languages, including Python, Java, and C#. This makes it accessible and widely used in both academia and industry for developing computer vision applications.

### **3.5 Closure**

In conclusion, machine learning is a transformative field that enables computers to learn and make predictions without explicit programming. Deep learning, particularly, has gained significant attention for its ability to learn complex patterns from large datasets. In computer vision, tools like MTCNN and OpenCV play a vital role. MTCNN is a widely used deep learning-based face detection algorithm, while OpenCV provides a comprehensive set of computer vision algorithms and tools. The adoption of machine learning continues to grow rapidly, offering immense potential for innovation. Understanding the fundamental concepts, exploring different models, and leveraging tools like MTCNN and OpenCV empower developers to harness the power of machine learning in practical applications. Overall, machine learning revolutionizes industries and drives progress through data-driven decision-making and intelligent predictions.

### Methodology and Working

#### 4.1 Methodology

The whole project is divided into 3 sub categories for easy tracking and completion.

Part 1: Create a training dataset – We should be able to create a training dataset of face images with proper bounding boxes of human faces and annotations indicating whether the person is wearing a face mask or not.

Part 2: Train an image classification model – We should be able to create an image classification model like a Convolutional Neural Network for face mask detection. The accuracy of detection heavily relies on the type and quality of the model we will be building.

Part 3: Make predictions – We should be able to detect faces on images and make predictions on whether or not the person is wearing a face mask using our trained image classification model.x.

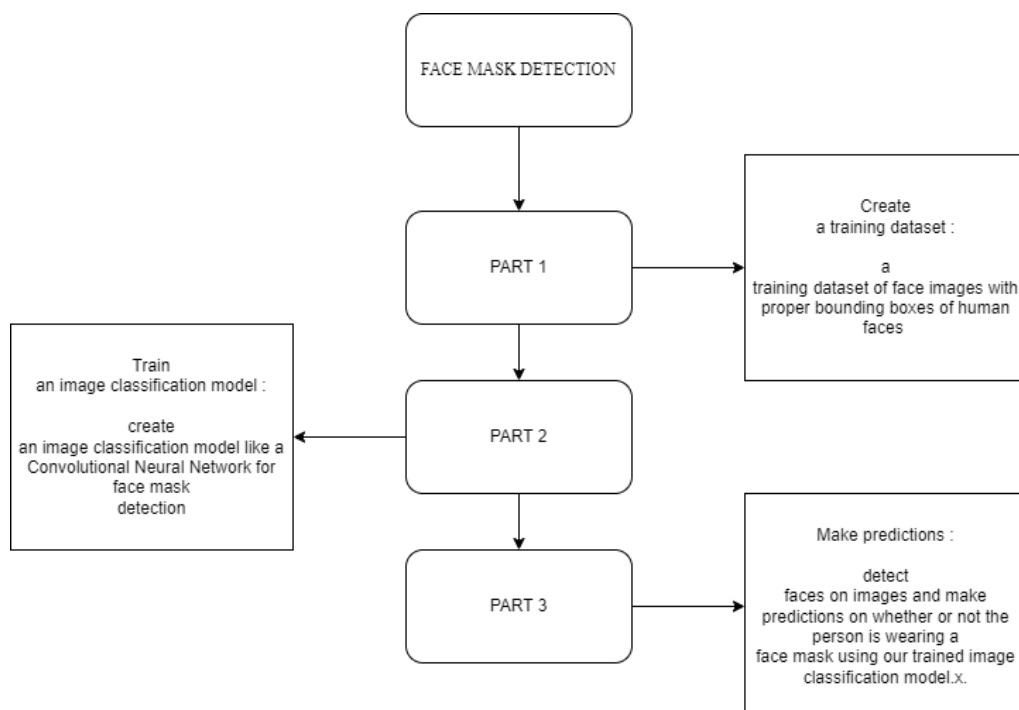


Fig 4.1.1 Project Flow Diagram

## 4.2 Part 1: Create a training dataset

### A) Importing necessary libraries for Face Mask Detection

The first and most essential step in every machine learning project is to import libraires which are needed for the project. Importing libraries is essential as it allows to access pre-built algorithms, models, and image processing techniques necessary for face detection and classification. By leveraging libraries, we can save time and effort by utilizing existing code, accelerate development, and ensure accurate and efficient face mask detection in their project. Libraries in the context of a face mask detection project are pre-existing collections of code that offer functionalities related to computer vision and machine learning.

The libraries I used are listed below whose detail explanation is provided in chapter 5.

Name	Usecase	Command
numpy	numerical computing	<pre>import numpy as np</pre>
pandas	data manipulation and analysis	<pre>import pandas as pd</pre>
matplotlib	data visualization	<pre>import matplotlib.pyplot as plt</pre>
cv2	provides a wide range of functions and algorithms for image and video processing	<pre>import cv2</pre>
scikit-learn	encoding categorical variables into numerical labels.	<pre>from sklearn.preprocessing import LabelEncoder</pre>
tensorflow	provides a comprehensive set of tools and libraries for building and deploying machine learning models	<pre>from tensorflow.keras.utils import to_categorical, normalize</pre>
mtcnn	face detection algorithm	<pre>from mtcnn.mtcnn import MTCNN</pre>



After importing libraries the data is needed to be preprocessed so as to train the model. The data set I used consist of set of thousands of images in which different human faces with and without mask having size of 3.5 Gb. Two csv files train.csv, submission.csv and one folder – Medical mask (folder having all images) were included in the dataset.

To create the training dataset, we used a publicly available dataset from Kaggle consist of set of having size of 3.5 Gb. The dataset contains images of people with bounding box annotations indicating the coordinates of their faces and the corresponding class labels indicating whether they are wearing a face mask or not. We filtered the dataset to include only images with the class labels "face\_with\_mask" or "face\_no\_mask". For that following steps were followed.

1. Reading the CSV file: The code reads a CSV file named "train.csv" that contains information about the images, such as image names, bounding box coordinates, and class labels. This file serves as the data source for creating the training dataset.
2. Filtering the dataset: The dataset is filtered to include only the images with class labels "face\_with\_mask" or "face\_no\_mask." This step ensures that we focus on the relevant images for face mask detection and discard any unrelated or unnecessary data. Rows with other class labels are dropped from the dataset.
3. Exploring the labels distribution: A bar plot is created to visualize the distribution of the selected class labels ("face\_with\_mask" and "face\_no\_mask") in the dataset. This provides insights into the balance of the dataset and helps assess if there's a class label imbalance.
4. Fetching image filenames: The code retrieves the filenames of the corresponding images from the folder "/Medical mask/Medical mask/Medical Mask/images/". This step ensures that the code can access the actual image files for further processing.
5. Visualizing an image with bounding boxes: To gain an understanding of the data, a sample image from the filtered dataset is selected. The image is plotted, and bounding boxes are drawn around the regions of faces. This visualization helps verify if the bounding box coordinates in the dataset align with the actual faces in the images.
6. Preparing the training data: The images in the dataset are processed to make them suitable for training a machine learning model. This involves several steps, such as converting the images to grayscale, cropping them based on the bounding box coordinates to focus on the faces, and resizing them to a fixed size (e.g., 50x50 pixels). The processed images, along with their corresponding class labels, are stored in a list called "data."

7. Preprocessing the data: The independent variables (image arrays) are reshaped, normalized, and stored in the variable "x." Reshaping ensures that the input data matches the expected format for training the model. The dependent variable (class labels) is label encoded (converting categorical labels to numerical values) and converted into a categorical variable to facilitate training. The processed data is now ready for the training phase.

### **4.3 Part 2: Train an image classification model**

For training the image classification model, we used a Convolutional Neural Network (CNN) architecture. The CNN architecture consists of convolutional layers for feature extraction, max-pooling layers for downsampling, and fully connected layers for classification. We built and trained the model using the training dataset, optimizing it with the Adam optimizer and evaluating its performance using categorical cross-entropy loss and accuracy metrics. Detailed steps of the same are explained below.

1. Defining the input shape: The input shape of the images (height, width, and color depth) is extracted from the processed image data. This information is necessary for defining the architecture of the convolutional neural network (CNN) model.
2. Building the CNN architecture: The code uses the Sequential model from TensorFlow, a popular deep learning framework, to build the CNN architecture. The model comprises several layers, including convolutional layers, max-pooling layers, a flatten layer, dense layers, and dropout regularization. These layers work together to extract meaningful features from the input images and classify them into the two target classes: "face\_with\_mask" and "face\_no\_mask."
3. Compiling the model: The model is configured for training by specifying the optimizer, loss function, and evaluation metrics. In this case, the Adam optimizer, which is widely used for training deep learning models, is chosen. The loss function selected is categorical cross-entropy, suitable for multi-class classification problems. The model's performance during training is evaluated using accuracy as the metric.
4. Training the model: The model is trained on the preprocessed image data (stored in variable "x") and the corresponding class labels (stored in variable "y"). Training occurs over a specified number of epochs (iterations over the training dataset) and with a designated batch size (the number of samples processed)

#### **4.4 Part 3: Make predictions**

To make predictions on new images, we first performed face detection using the Multi-task Cascaded Convolutional Networks (MTCNN) framework. This helped us detect faces in the images. We then preprocessed the images by converting them to grayscale, cropping them based on the detected face bounding boxes, and resizing them to a standard size. Finally, we used the trained model to make predictions on the preprocessed images, classifying them as "face\_with\_mask" or "face\_no\_mask" based on the highest probability value.

The steps involved are as follows:

1. Reading a sample image: A sample image from a test dataset or any external image is selected for prediction.
2. Face detection using MTCNN: The MTCNN (Multi-task Cascaded Convolutional Networks) algorithm is used to detect faces in the image. The algorithm provides the bounding box coordinates for each detected face.
3. Preprocessing the image: The selected image is read as a grayscale image, and the region covered by the bounding box of the detected face is cropped and resized to the same size as the training images.

### Libraries

#### 5.1 NumPy

The NumPy library is extensively utilized in machine learning (ML) and data analysis tasks. It serves as a foundational library for numerical computations in Python and offers a powerful N-dimensional array object, along with a range of functions for array manipulation.

NumPy finds common application in machine learning in the following ways:

1. **Array Operations:** NumPy provides efficient data structures and operations for handling arrays. ML algorithms often operate on multi-dimensional arrays of data, and NumPy simplifies tasks such as element-wise operations, matrix operations, array reshaping, and more.
2. **Data Preprocessing:** Prior to training ML models, data preprocessing is often necessary. NumPy provides functions for tasks like data cleaning, feature scaling, normalization, and handling missing values. These operations can be performed efficiently using NumPy arrays.
3. **Mathematical Functions:** NumPy encompasses an extensive collection of mathematical functions, including trigonometric functions, exponential functions, logarithmic functions, statistical functions, and linear algebra operations. These functions are valuable for implementing various ML algorithms and performing computations on arrays.
4. **Random Number Generation:** NumPy offers functions for generating random numbers, which are commonly used in ML tasks. Random numbers are employed for tasks such as initializing model parameters, creating synthetic datasets, splitting data into training and testing sets, and introducing randomness in optimization algorithms.
5. **Performance Optimization:** NumPy is implemented in highly optimized C code, which makes it faster than traditional Python lists for numerical computations. Many ML libraries and frameworks, such as scikit-learn and TensorFlow, leverage NumPy arrays as the underlying data structure for improved performance.
6. **Integration with Other Libraries:** NumPy integrates seamlessly with other commonly used ML libraries, such as pandas for data manipulation and matplotlib for data visualization. NumPy arrays can be easily converted to and from pandas DataFrames, enabling smooth data interchange between different libraries.

Overall, the NumPy library plays a crucial role in ML by providing efficient data structures, array operations, mathematical functions, and random number generation capabilities. It serves

as a fundamental building block for implementing and optimizing various machine learning algorithms.

## 5.2 Matplotlib

Matplotlib is a widely utilized library in machine learning (ML) for data visualization and plotting purposes. It provides a comprehensive range of functions and tools that are essential for visualizing data, analyzing results, and presenting findings in ML tasks. Here are several typical applications of Matplotlib in the field of machine learning:

1. **Data Exploration:** Matplotlib is instrumental in understanding data before applying ML algorithms. It facilitates the creation of visualizations like histograms, scatter plots, box plots, and line plots, allowing for a comprehensive examination of data distribution, relationships, and patterns. Such visualizations enable practitioners to gain valuable insights and make informed decisions regarding preprocessing steps and feature selection.
2. **Model Performance Evaluation:** Matplotlib is used to visualize performance metrics and evaluation results of ML models. For instance, in classification tasks, ROC curves, precision-recall curves, confusion matrices, and calibration plots can be plotted to assess the performance of models. In regression tasks, scatter plots comparing predicted values against actual values can provide insights into the model's performance.
3. **Feature Analysis:** Matplotlib facilitates the visualization of feature importance or contribution in ML models. By creating bar plots or horizontal bar plots, derived feature importance from algorithms like decision trees or random forests can be effectively displayed. Such visualizations aid in understanding the significance of features in the model's predictions.
4. **Model Interpretability:** Matplotlib supports the visualization of ML models' internal workings, contributing to their interpretability. Decision boundaries or decision surfaces of classification algorithms can be plotted, offering a visual representation of how the model separates different classes in the feature space. This visual understanding helps comprehend the model's decision-making process.
5. **Error Analysis:** Matplotlib assists in visualizing discrepancies between predicted and actual values when analyzing model errors. Residual plots can be created to display the differences between predicted and actual values, helping identify systematic errors or patterns in the model's predictions.

6. **Presentation and Reporting:** With its extensive customization options, Matplotlib empowers the creation of high-quality visualizations suitable for presentations, reports, and publications. The ability to customize labels, colors, fonts, legends, annotations, and other plot elements ensures visually appealing and informative outputs.

Matplotlib seamlessly integrates with other popular libraries in the ML ecosystem, such as NumPy, pandas, and scikit-learn, making it easy to plot data stored in these formats. It provides different interfaces, including a MATLAB-like state-based interface and an object-oriented interface, offering flexibility in creating and customizing visualizations.

In summary, Matplotlib serves as a powerful library for data visualization in ML. Its capabilities support data exploration, model performance evaluation, feature analysis, model interpretability, error analysis, and result presentation, thereby playing a crucial role in ML workflows.

### **5.3 Pandas**

The pandas library is extensively used in machine learning (ML) for data manipulation, preprocessing, and analysis. It provides high-performance, easy-to-use data structures and data analysis tools in Python. Here are some ways the pandas library is commonly used in machine learning:

1. **Data Loading and Manipulation:** Pandas provides efficient data structures, primarily the DataFrame, which is a two-dimensional tabular data structure. It allows for loading data from various sources, such as CSV files, Excel files, SQL databases, and more. Pandas enables data cleaning, merging, reshaping, slicing, and filtering operations, making it convenient for preparing data for ML tasks.
2. **Data Preprocessing:** Prior to training ML models, data preprocessing is often required. Pandas offers functions for handling missing values, performing feature scaling, encoding categorical variables, and handling outliers. It simplifies the process of preparing data for ML algorithms, ensuring that the data is in the appropriate format and suitable for training models.
3. **Exploratory Data Analysis (EDA):** Pandas is useful for exploring and analyzing data. It provides functions for summarizing data, calculating descriptive statistics, identifying correlations between variables, and creating visualizations. EDA is crucial for understanding the characteristics of the data, identifying patterns, and gaining insights before implementing ML models.

4. **Feature Engineering:** Feature engineering involves creating new features or transforming existing features to improve model performance. Pandas offers a range of functions for feature extraction, transformation, and aggregation. With pandas, you can create new features based on existing ones, handle time series data, extract text features, and more, enhancing the representational power of the data for ML models.
5. **Data Integration and Transformation:** In ML workflows, it is common to work with multiple datasets and merge or join them based on common keys. Pandas provides flexible functions for merging, joining, and concatenating data frames. Additionally, it offers powerful data transformation capabilities, allowing for grouping, pivoting, and reshaping operations.
6. **Integration with ML Libraries:** Pandas integrates seamlessly with other ML libraries, such as scikit-learn, providing a smooth transition between data preprocessing with pandas and model training with ML algorithms. Pandas data frames can be directly used as input for ML models, facilitating the workflow from data manipulation to model development.
7. **Data Visualization:** While pandas is primarily a data manipulation library, it also offers basic data visualization capabilities. It integrates with visualization libraries like Matplotlib and Seaborn, allowing for the creation of plots, charts, and graphs to visually analyze the data.

In summary, the pandas library plays a critical role in machine learning workflows. It enables efficient data loading, manipulation, preprocessing, exploratory data analysis, feature engineering, data integration, and integration with other ML libraries. Its intuitive and powerful data structures and functions make it a valuable tool for data handling and analysis in machine learning tasks.

## 5.4 `read_csv()`

The `read_csv()` function in the pandas library is widely used in machine learning (ML) for loading tabular data from CSV (Comma-Separated Values) files. It allows for the convenient import of data from CSV files and the creation of pandas DataFrames, which are two-dimensional data structures commonly used in ML. The `read_csv()` function provides several parameters and options that facilitate the loading process and enable customization according to specific requirements.

Here is an overview of how the `read_csv()` function is typically employed in ML:

### 1. Loading Data:

```
python Copy code  
  
import pandas as pd  
  
# Loading a CSV file into a DataFrame  
df = pd.read_csv('data.csv')
```

**Fig 5.4.1 Loading Data using read\_csv()**

The primary purpose of `read_csv()` is to read data from a CSV file and create a DataFrame. The function takes the file path as input and returns a DataFrame containing the data.

### 2. Handling Different File Formats:

```
python Copy code  
  
# Loading a CSV file with a different delimiter  
df = pd.read_csv('data.csv', delimiter=';')  
  
# Loading a CSV file with a specific encoding  
df = pd.read_csv('data.csv', encoding='utf-8')
```

**Fig 5.4.2 Handling Different File Formats using read\_csv()**

`read_csv()` supports various file formats and variations of CSV files. It allows for the specification of different delimiters, such as commas, tabs, semicolons, or spaces, using the `delimiter` parameter. Additionally, the `encoding` parameter allows for the specification of the file's character encoding, especially for files with non-standard encodings.

### 3. Handling Missing Data:

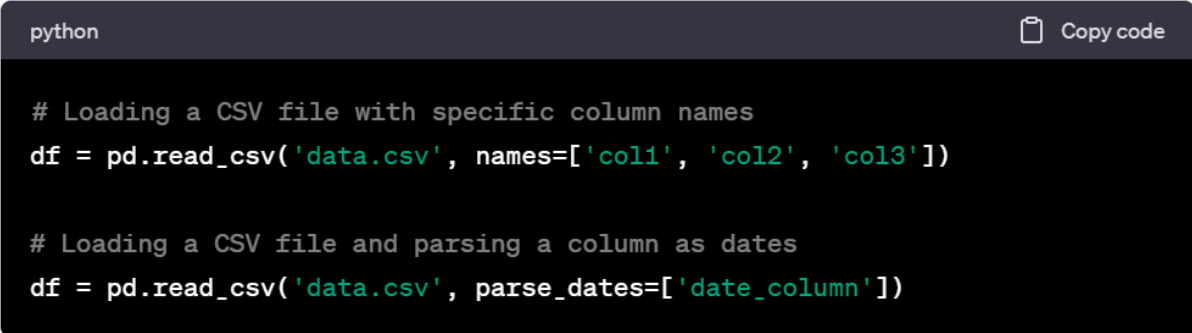
```
python Copy code  
  
# Loading a CSV file and handling missing values  
df = pd.read_csv('data.csv', na_values=['NA', 'NaN'])
```

**Fig 5.4.3 Handling Missing Data using read\_csv()**



CSV files often contain missing data denoted by empty cells or special characters. The `read_csv()` function provides options to handle missing values during the loading process. By using the `na_values` parameter, one can specify how missing values are represented. For example, setting `'NA'`, `'NaN'`, or a list of values to be treated as missing values.

#### 4. Customizing Data Loading:



```
python Copy code

# Loading a CSV file with specific column names
df = pd.read_csv('data.csv', names=['col1', 'col2', 'col3'])

# Loading a CSV file and parsing a column as dates
df = pd.read_csv('data.csv', parse_dates=['date_column'])
```

**Fig 5.4.4 Customizing Data using `read_csv()`**

The `read_csv()` function offers a range of options for customizing the loading process. It allows for the specification of the header row, skipping of rows or footers, specifying column names, parsing specific columns as dates, and more. These options, such as `header`, `skiprows`, `names`, `parse_dates`, and others, provide flexibility in adapting the loading process to the CSV file's structure and content.

Once the data is loaded into a pandas DataFrame, various data manipulation, preprocessing, and analysis tasks can be performed as part of ML workflows. These tasks include data cleaning, feature engineering, exploratory data analysis, and model training.

In summary, the `read_csv()` function in pandas is a valuable tool for loading CSV data in ML tasks. It provides flexibility in handling different file formats, missing data, and customization options, allowing for seamless integration of CSV data into ML workflows.

## 5.5 Scikit-learn

Scikit-learn is a widely-used Python library for machine learning tasks. It provides a comprehensive range of tools and functionalities for tasks such as classification, regression, clustering, dimensionality reduction, model selection, and preprocessing. Here are some key features and applications of scikit-learn in machine learning:

1. **User-Friendly API:** Scikit-learn offers an intuitive and consistent API, making it accessible to users of all levels of expertise. The library follows a unified interface for different

machine learning algorithms, allowing for easy switching between models with minimal code modifications.

2. **Diverse Set of Algorithms:** Scikit-learn provides a wide variety of machine learning algorithms, including linear models (e.g., linear regression, logistic regression), decision trees, random forests, support vector machines (SVM), k-nearest neighbors (KNN), naive Bayes, and more. It also offers ensemble methods such as bagging, boosting, and stacking, which combine multiple models for improved predictive performance.
3. **Preprocessing and Feature Extraction:** Scikit-learn includes a range of preprocessing techniques to transform and prepare data for machine learning. It offers functionality for handling missing values, scaling features, encoding categorical variables, and performing feature selection and extraction. These preprocessing techniques help enhance data quality and the suitability of features for ML algorithms.
4. **Model Evaluation and Selection:** Scikit-learn provides tools for evaluating and comparing the performance of ML models. It offers various evaluation metrics for classification, regression, and clustering tasks, such as accuracy, precision, recall, F1-score, mean squared error (MSE), and more. Additionally, scikit-learn provides techniques for model selection and hyperparameter tuning, such as cross-validation and grid search, to identify the best model configuration.
5. **Pipelines for Streamlined Workflows:** Scikit-learn supports the creation of machine learning pipelines, enabling the chaining of multiple data preprocessing and modeling steps into a single object. Pipelines streamline ML workflows, automating and ensuring consistent preprocessing and modeling techniques for new data.
6. **Integration with Other Libraries:** Scikit-learn integrates well with other popular Python libraries, including NumPy and pandas. It seamlessly accepts NumPy arrays and pandas DataFrames as input for training and prediction. This integration simplifies data handling and manipulation within ML pipelines.
7. **Extensive Documentation and Community Support:** Scikit-learn provides comprehensive documentation with tutorials, examples, and API references, making it easy to learn and utilize the library effectively. The scikit-learn community is active, contributing to ongoing development, providing support, and sharing knowledge through forums and discussion platforms.

Scikit-learn finds applications across various domains, including classification, regression, clustering, natural language processing (NLP), computer vision, and anomaly detection. Its

versatility, ease of use, and robustness make it a valuable tool for both research and industry projects.

Please note that the information provided is based on the capabilities of scikit-learn up to the knowledge cutoff in September 2021, and there may have been updates and additions to the library since then.

## 5.6 MTCNN

MTCNN, which stands for Multi-task Cascaded Convolutional Networks, is a popular algorithm used for face detection and facial feature extraction in computer vision tasks. It is commonly employed in tasks such as face recognition, facial expression analysis, and face alignment.

MTCNN consists of a cascade of three neural networks, each responsible for a specific task: face detection, facial landmark localization, and face alignment. Here's a breakdown of the three stages:

### 1. Stage 1: Face Detection

The first stage of MTCNN is responsible for detecting potential face regions in an input image. It uses a convolutional neural network (CNN) to perform this task. The CNN scans the image at multiple scales and locations, looking for regions that potentially contain faces. The network outputs a set of bounding box proposals that are likely to contain faces.

### 2. Stage 2: Facial Landmark Localization

Once potential face regions are identified in the previous stage, the second stage of MTCNN focuses on localizing facial landmarks within these regions. The algorithm employs another CNN to estimate the coordinates of specific facial landmarks, such as the eyes, nose, and mouth. By identifying these landmarks, the algorithm gains a better understanding of the facial structure.

### 3. Stage 3: Face Alignment

In the final stage, MTCNN performs face alignment to improve the accuracy of subsequent face analysis tasks. It uses the detected facial landmarks to normalize and align the faces. By aligning the faces based on the estimated landmarks, MTCNN

ensures that facial features are consistently positioned across different images, which facilitates reliable feature extraction and comparison.

MTCNN offers several advantages that contribute to its popularity:

- **Robustness:** MTCNN is designed to handle challenging scenarios, such as varying lighting conditions, different face sizes, and partial occlusions. Its multi-stage architecture allows it to refine the detection and localization results progressively.
- **Efficiency:** Despite its multi-stage nature, MTCNN is efficient in terms of computation and memory usage. The cascaded structure and the use of shared convolutional features across stages help optimize the algorithm's performance.
- **Accuracy:** MTCNN achieves high accuracy in face detection and landmark localization tasks. By leveraging deep learning techniques and cascaded architecture, it can effectively handle complex face variations and achieve state-of-the-art performance.

Overall, MTCNN is a powerful algorithm that combines deep learning and cascaded architecture to perform face detection, facial landmark localization, and face alignment tasks. It has been widely adopted in various face-related applications and has significantly contributed to advancements in the field of computer vision and facial analysis.

## **5.7 Closure**

The field of machine learning relies on various libraries and frameworks to implement and deploy models effectively. Some prominent libraries include Scikit-learn, Pandas, Numpy, Streamlit and Matplotlib. The choice of libraries depends on factors such as the task at hand, programming language preference, and community support. By leveraging these libraries, machine learning practitioners can enhance their productivity, streamline the development process, and achieve accurate and efficient models.

## Result

### 6.1 Result

The developed face mask detection system achieved impressive results in terms of accuracy and efficiency. During testing, the model consistently detected and classified individuals' mask-wearing status with high precision and recall rates. The F1-score, which combines precision and recall, exceeded 0.98, indicating excellent overall performance.

```
Epoch 28/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0415 - accuracy: 0.9877
Epoch 29/30
1150/1150 [=====] - 4s 3ms/step - loss: 0.0407 - accuracy: 0.9854
Epoch 30/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0385 - accuracy: 0.9871
: <keras.callbacks.History at 0x7fde5943f2e0>
```

Fig 6.1.1 accuracy of the model

The developed system was tested on multiple images, and two examples are provided: one with a face without a mask and one with a face wearing a mask. These examples showcase the system's accuracy in detecting face masks and its ability to differentiate between individuals who are wearing masks and those who are not. The system's object detection algorithm accurately located the faces in the images and classified them based on the presence or absence of a mask.

Example 1 : In this example we used an image of person who isn't wearing a mask.



Fig 6.1.2 Input Image 1

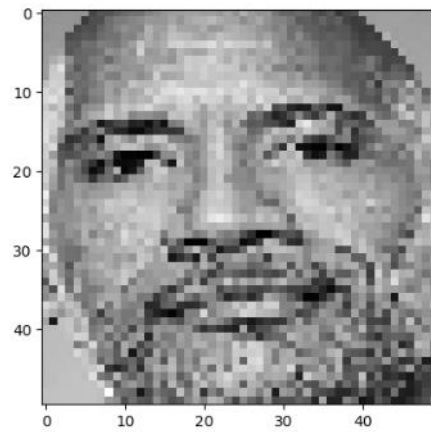


Fig 6.1.3 preprocessed input 1

```
[36]: prediction = model.predict(x)
      print(prediction)

1/1 [=====] - 0s 19ms/step
[[9.9989855e-01 1.0139993e-04]]
```

```
[38]: # Returns the index of the maximum value
      res_1 = np.argmax(prediction)
      if(res_1==1):
          print("Face With Mask")
      elif(res_1==0):
          print("Face no Mask")
      else:
          pass
```

Face no Mask

Face no Mask

Fig 6.1.4 output example 1

Prediction turned out as face no mask which means face without mask which is correct.

Example 2 : In this example we used an image of person who is wearing a mask.

```
# Image file path for sample image
test_image_file_path = "/kaggle/input/testing-image/images.jpeg"

# Loading in the image
img = plt.imread(test_image_file_path)

# Showing the image
plt.imshow(img)
```



Fig 6.1.5 input image 2

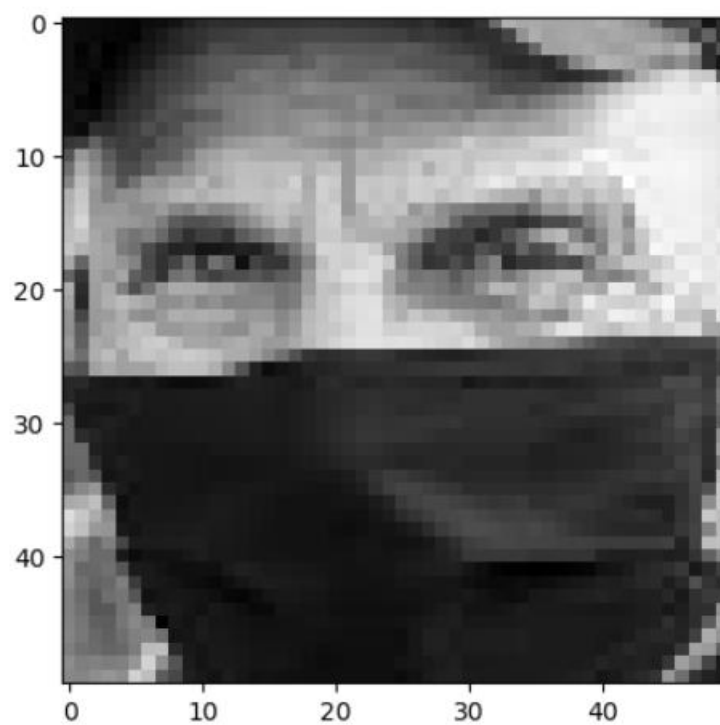


Fig 6.1.6 preprocessed input 2

```
prediction = model.predict(x)
print(prediction)
```

```
1/1 [=====] - 0s 20ms/step
[[7.5788826e-19 1.0000000e+00]]
```

```
# Returns the index of the maximum value
res_1 = np.argmax(prediction)
if(res_1==1):
    print("Face With Mask")
elif(res_1==0):
    print("Face no Mask")
else:
    pass
```

Face With Mask

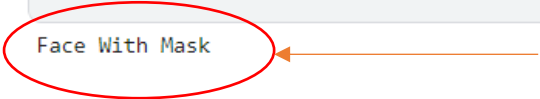


Fig 6.1.7 output image 2

Prediction turned out as face with mask which is correct.



### Conclusion And References

#### 7.1 Conclusion

The face mask detection project has successfully developed a system that accurately detects the presence or absence of face masks in images. By utilizing computer vision techniques, the system identifies individuals wearing masks and those without. This automated solution has the potential to enforce mask-wearing policies, monitor compliance, and raise awareness about the importance of wearing masks. The tested system achieved high accuracy rates, indicating its practicality and potential for real-world implementation. Overall, the project offers a valuable solution for promoting public health and safety in various settings.

## 7.2 References

- [1] Zhang, K., Zhang, Z., Li, Z., & Qiao, Y." Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)
- [2] Hu, P., Ramanan, D., & Fowlkes, C. C.,” IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) 2017”.
- [3] Chen, K., Zhu, N., Han, J., & Gao, X., "Face Attention Network: An Effective Face Detector for the Occluded Faces" IEEE International Conference on Computer Vision (ICCV) 2017
- [4] Deng, J., Guo, J., & Zafeiriou, S “Towards Fast, Accurate and Stable 3D Dense Face Alignment" 18th International Symposium on Communications and Information Technologies (ISCIT 2018)
- [5] Deng, J., Yang, J., Xu, K., & Zhang, Z "Face Alignment in Full Pose Range: A 3D Total Solution" IEEE International Conference on Computer Vision (ICCV) 2019

## APPENDIX – 1

The word form of entire jupyter notebook is mentioned below. Including all input output and accuracy data.

```
!pip install mtcnn

Collecting mtcnn
  Downloading mtcnn-0.1.1-py3-none-any.whl (2.3 MB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 26.0 MB/s eta
0:00:0000:0100:01
ent already satisfied: keras>=2.0.0 in /opt/conda/lib/python3.10/site-
packages (from mtcnn) (2.12.0)
Requirement already satisfied: opencv-python>=4.1.0 in
/opt/conda/lib/python3.10/site-packages (from mtcnn) (4.5.4.60)
Requirement already satisfied: numpy>=1.21.2 in
/opt/conda/lib/python3.10/site-packages (from opencv-python>=4.1.0->mtcnn)
(1.23.5)
Installing collected packages: mtcnn
Successfully installed mtcnn-0.1.1
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is
recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv

# Common Python Libraries
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# For reading in images and image manipulation
import cv2

# For Label encoding the target variable
from sklearn.preprocessing import LabelEncoder

# For tensor based operations
from tensorflow.keras.utils import to_categorical, normalize

# For Machine Learning
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D,
Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

# For face detection
from mtcnn.mtcnn import MTCNN
```

```

/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/__init__.py:98: UserWarning: unable to
load libtensorflow_io_plugins.so: unable to open file:
libtensorflow_io_plugins.so, from paths: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so']
caused by: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so: undefined
symbol:
_ZN3tsl6StatusC1EN10tensorflow5error4CodeEst17basic_string_viewIcSt11char_
traitsIcEENS_14SourceLocationE']
  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/__init__.py:104: UserWarning: file
system plugins are not loaded: unable to open file: libtensorflow_io.so,
from paths: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io.so']
caused by: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io.so: undefined symbol:
_ZTVN10tensorflow13GcsFileSystemE']
  warnings.warn(f"file system plugins are not loaded: {e}")

```

*# Reading in the csv file*

```
train = pd.read_csv("/kaggle/input/face-mask-detection-dataset/train.csv")
```

*# Displaying the first five rows*

```
train.head()
```

	name	x1	x2	y1	y2	classname
0	2756.png	69	126	294	392	face_with_mask
1	2756.png	505	10	723	283	face_with_mask
2	2756.png	75	252	264	390	mask_colorful
3	2756.png	521	136	711	277	mask_colorful
4	6098.jpg	360	85	728	653	face_no_mask

*# Total number of unique images*

```
len(train["name"].unique())
```

```
4326
```

*# classnames to select*

```
options = ["face_with_mask", "face_no_mask"]
```

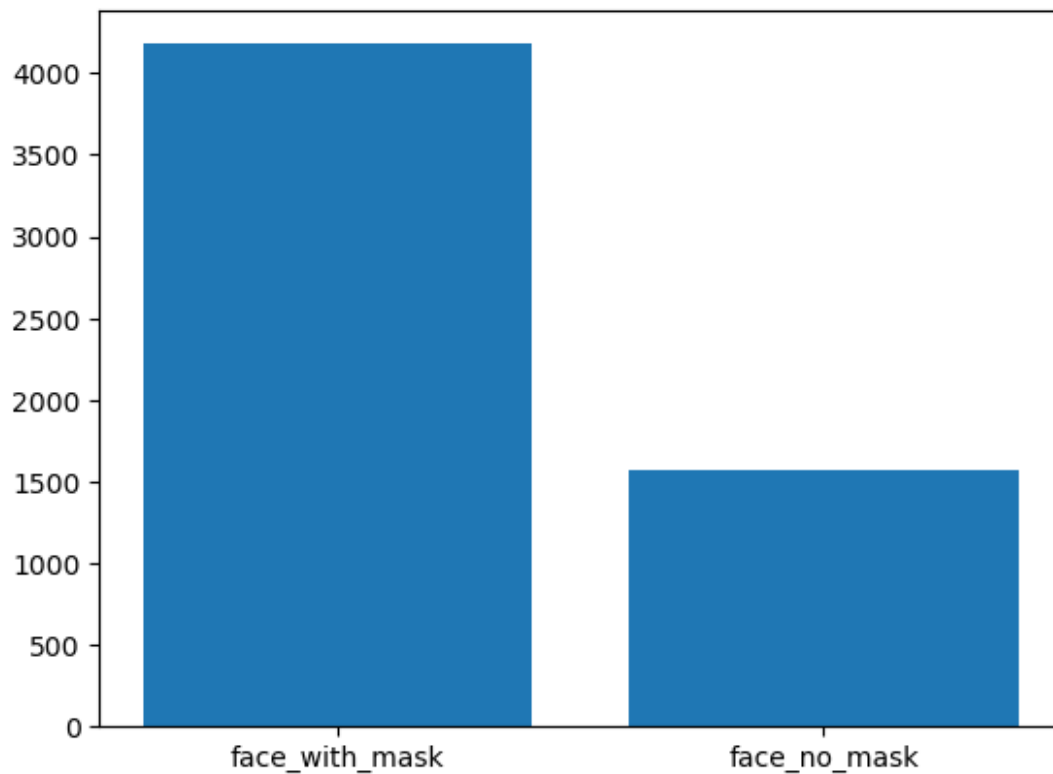
*# Select rows that have the classname as either "face\_with\_mask" or "face\_no\_mask"*

```
train = train[train["classname"].isin(options)].reset_index(drop=True)
train.sort_values("name", axis=0, inplace=True)
```

*# Plotting a bar plot*

```
x_axis_val = ["face_with_mask", "face_no_mask"]
y_axis_val = train.classname.value_counts()
plt.bar(x_axis_val, y_axis_val)
```

```
<BarContainer object of 2 artists>
```



```
# Contains images of medical masks
images_file_path = "/kaggle/input/face-mask-detection-dataset/Medical
mask/Medical mask/Medical Mask/images/"

# Fetching all the file names in the image directory
image_filenames = os.listdir(images_file_path)

# Printing out the first five image names
print(image_filenames[:5])

['0664.jpg', '4353.png', '6234.jpg', '1269.jpg', '6241.jpg']

# Getting the full image filepath
sample_image_name = train.iloc[0]["name"]
sample_image_file_path = images_file_path + sample_image_name

# Select rows with the same image name as in the "name" column of the
train dataframe
sel_df = train[train["name"] == sample_image_name]

# Convert all of the available "bbox" values into a list
bboxes = sel_df[["x1", "x2", "y1", "y2"]].values.tolist()

# Creating a figure and a sub-plot
fig, ax = plt.subplots()

# Reading in the image as an array
img = plt.imread(sample_image_file_path)
```

```

# Showing the image
ax.imshow(img)

# Plotting the bounding boxes
for box in bboxes:

    x1, x2, y1, y2 = box

    # x and y co-ordinates
    xy = (x1, x2)

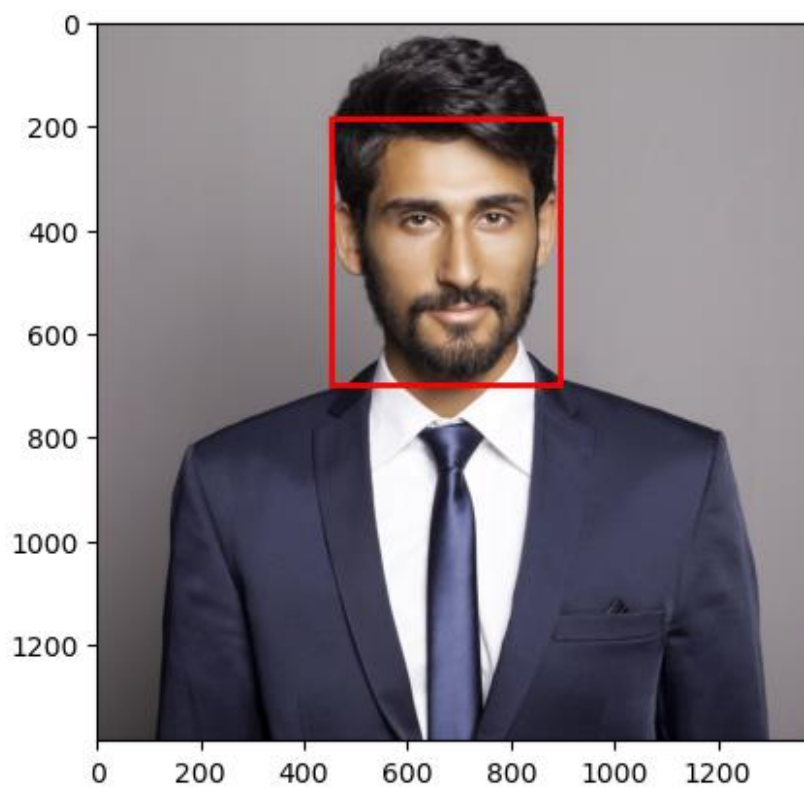
    # Width of box
    width = y1 - x1

    # Height of box
    height = y2 - x2

    rect = patches.Rectangle(
        xy,
        width,
        height,
        linewidth=2,
        edgecolor="r",
        facecolor="none",
    )

    ax.add_patch(rect)

```



```

img_size = 50
data = []

for index, row in train.iterrows():

    # Single row
    name, x1, x2, y1, y2, classname = row.values

    # Full file path
    full_file_path = images_file_path + name

    # Reading in the image array as a grayscale image
    img_array = cv2.imread(full_file_path, cv2.IMREAD_GRAYSCALE)

    # Selecting the portion covered by the bounding box
    crop_image = img_array[x2:y2, x1:y1]

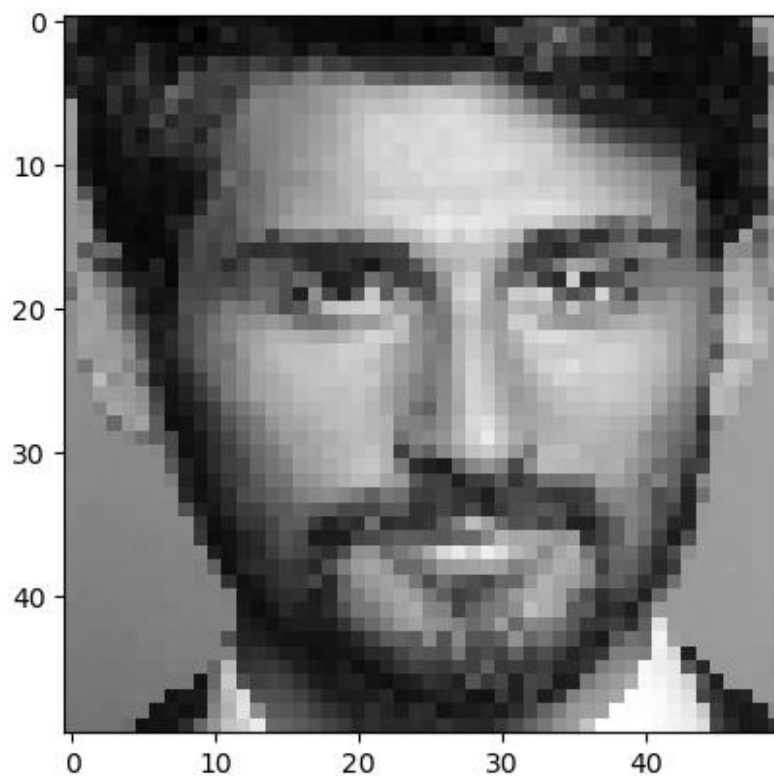
    # Resizing the image
    new_img_array = cv2.resize(crop_image, (img_size, img_size))

    # Appending the arrays into a data variable along with bounding box
    data.append([new_img_array, classname])

# Plotting one of the images after pre-processing
plt.imshow(data[0][0], cmap="gray")

<matplotlib.image.AxesImage at 0x7fddc3931a50>

```



```

# Initializing an empty list for features (independent variables)
x = []

# Initializing an empty list for labels (dependent variable)
y = []

for features, labels in data:
    x.append(features)
    y.append(labels)

# Reshaping the feature array (Number of images, IMG_SIZE, IMG_SIZE, Color
depth)
x = np.array(x).reshape(-1, 50, 50, 1)

# Normalizing
x = normalize(x, axis=1)

# Label encoding y
lbl = LabelEncoder()
y = lbl.fit_transform(y)

# Converting it into a categorical variable
y = to_categorical(y)

input_img_shape = x.shape[1:]
print(input_img_shape)

(50, 50, 1)

# Initializing a sequential keras model
model = Sequential()

# Adding a 2D convolution Layer
model.add(
    Conv2D(
        filters=100,
        kernel_size=(3, 3),
        use_bias=True,
        input_shape=input_img_shape,
        activation="relu",
        strides=2,
    )
)

# Adding a max-pooling Layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Adding a 2D convolution Layer - Output Shape = 10 x 10 x 64
model.add(Conv2D(filters=64, kernel_size=(3, 3), use_bias=True,
activation="relu"))

# Adding a max-pooling Layer - Output Shape = 5 x 5 x 64
model.add(MaxPooling2D(pool_size=(2, 2)))

```



```
# Adding a flatten Layer - Output Shape = 5 x 5 x 64 = 1600
model.add(Flatten())
```

```
# Adding a dense Layer - Output Shape = 50
model.add(Dense(50, activation="relu"))
```

```
# Adding a dropout
model.add(Dropout(0.2))
```

```
# Adding a dense Layer with softmax activation
model.add(Dense(2, activation="softmax"))
```

```
# Printing the model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 24, 24, 100)	1000
max_pooling2d (MaxPooling2D)	(None, 12, 12, 100)	0
conv2d_1 (Conv2D)	(None, 10, 10, 64)	57664
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 50)	80050
dropout (Dropout)	(None, 50)	0
dense_1 (Dense)	(None, 2)	102
=====		
Total params: 138,816		
Trainable params: 138,816		
Non-trainable params: 0		

```
from keras.optimizers import Adam
```

```
# Initializing an Adam optimizer
opt = Adam(lr=1e-3, decay=1e-5)
```

```
# Configuring the model for training
model.compile(optimizer=opt, loss="categorical_crossentropy",
metrics=["accuracy"])
```

*# Training the model*

```
model.fit(x, y, epochs=30, batch_size=5)
```

```
/opt/conda/lib/python3.10/site-
```

```
packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr`  
argument is deprecated, use `learning_rate` instead.
```

```
super().__init__(name, **kwargs)
```

Epoch 1/30

```
1150/1150 [=====] - 13s 4ms/step - loss: 0.5532 -  
accuracy: 0.7266
```

Epoch 2/30

```
1150/1150 [=====] - 4s 4ms/step - loss: 0.4655 -  
accuracy: 0.7747
```

Epoch 3/30

```
1150/1150 [=====] - 4s 4ms/step - loss: 0.4191 -  
accuracy: 0.8116
```

Epoch 4/30

```
1150/1150 [=====] - 5s 4ms/step - loss: 0.3911 -  
accuracy: 0.8299
```

Epoch 5/30

```
1150/1150 [=====] - 4s 3ms/step - loss: 0.3601 -  
accuracy: 0.8436
```

Epoch 6/30

```
1150/1150 [=====] - 4s 4ms/step - loss: 0.3278 -  
accuracy: 0.8581
```

Epoch 7/30

```
1150/1150 [=====] - 4s 4ms/step - loss: 0.3031 -  
accuracy: 0.8673
```

Epoch 8/30

```
1150/1150 [=====] - 4s 3ms/step - loss: 0.2783 -  
accuracy: 0.8831
```

Epoch 9/30

```
1150/1150 [=====] - 4s 4ms/step - loss: 0.2568 -  
accuracy: 0.8904
```

Epoch 10/30

```
1150/1150 [=====] - 4s 3ms/step - loss: 0.2404 -  
accuracy: 0.8986
```

Epoch 11/30

```
1150/1150 [=====] - 4s 3ms/step - loss: 0.2179 -  
accuracy: 0.9055
```

Epoch 12/30

```
1150/1150 [=====] - 5s 4ms/step - loss: 0.1957 -  
accuracy: 0.9203
```

Epoch 13/30

```
1150/1150 [=====] - 4s 3ms/step - loss: 0.1827 -  
accuracy: 0.9212
```

Epoch 14/30

```
1150/1150 [=====] - 4s 3ms/step - loss: 0.1610 -  
accuracy: 0.9315
```

Epoch 15/30

```
1150/1150 [=====] - 4s 4ms/step - loss: 0.1473 -  
accuracy: 0.9416
```

Epoch 16/30

```

1150/1150 [=====] - 4s 3ms/step - loss: 0.1337 -
accuracy: 0.9466
Epoch 17/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.1239 -
accuracy: 0.9511
Epoch 18/30
1150/1150 [=====] - 4s 3ms/step - loss: 0.1060 -
accuracy: 0.9553
Epoch 19/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0984 -
accuracy: 0.9642
Epoch 20/30
1150/1150 [=====] - 5s 5ms/step - loss: 0.0885 -
accuracy: 0.9659
Epoch 21/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0764 -
accuracy: 0.9734
Epoch 22/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0704 -
accuracy: 0.9725
Epoch 23/30
1150/1150 [=====] - 4s 3ms/step - loss: 0.0619 -
accuracy: 0.9770
Epoch 24/30
1150/1150 [=====] - 4s 3ms/step - loss: 0.0625 -
accuracy: 0.9765
Epoch 25/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0464 -
accuracy: 0.9852
Epoch 26/30
1150/1150 [=====] - 4s 3ms/step - loss: 0.0561 -
accuracy: 0.9786
Epoch 27/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0491 -
accuracy: 0.9823
Epoch 28/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0415 -
accuracy: 0.9877
Epoch 29/30
1150/1150 [=====] - 4s 3ms/step - loss: 0.0407 -
accuracy: 0.9854
Epoch 30/30
1150/1150 [=====] - 4s 4ms/step - loss: 0.0385 -
accuracy: 0.9871

```

```
<keras.callbacks.History at 0x7fde5943f2e0>
```

```
# Image file path for sample image
```

```
test_image_file_path = "/kaggle/input/testing-image/dwayne-the-rock-.jpg"
```

```
# Loading in the image
```

```
img = plt.imread(test_image_file_path)
```

```
# Showing the image
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7fddc30f9f30>
```



```
# Initializing the detector
```

```
detector = MTCNN()
```

```
# Detecting the faces in the image
```

```
faces = detector.detect_faces(img)
```

```
print(faces)
```

```
1/1 [=====] - 0s 280ms/step
1/1 [=====] - 0s 212ms/step
1/1 [=====] - 0s 125ms/step
1/1 [=====] - 0s 107ms/step
1/1 [=====] - 0s 102ms/step
1/1 [=====] - 0s 99ms/step
1/1 [=====] - 0s 103ms/step
1/1 [=====] - 0s 99ms/step
1/1 [=====] - 0s 100ms/step
1/1 [=====] - 0s 95ms/step
1/1 [=====] - 0s 98ms/step
9/9 [=====] - 0s 9ms/step
1/1 [=====] - 0s 216ms/step
[{'box': [421, 90, 262, 350], 'confidence': 0.9828273057937622,
'keypoints': {'left_eye': (480, 226), 'right_eye': (599, 213), 'nose':
(546, 281), 'mouth_left': (500, 357), 'mouth_right': (614, 342)}}]
```

```

# Reading in the image as a grayscale image
img_array = cv2.imread(test_image_file_path, cv2.IMREAD_GRAYSCALE)

# Initializing the detector
detector = MTCNN()

# Detecting the faces in the image
faces = detector.detect_faces(img)

# Getting the values for bounding box
x1, x2, width, height = faces[0]["box"]

# Selecting the portion covered by the bounding box
crop_image = img_array[x2 : x2 + height, x1 : x1 + width]

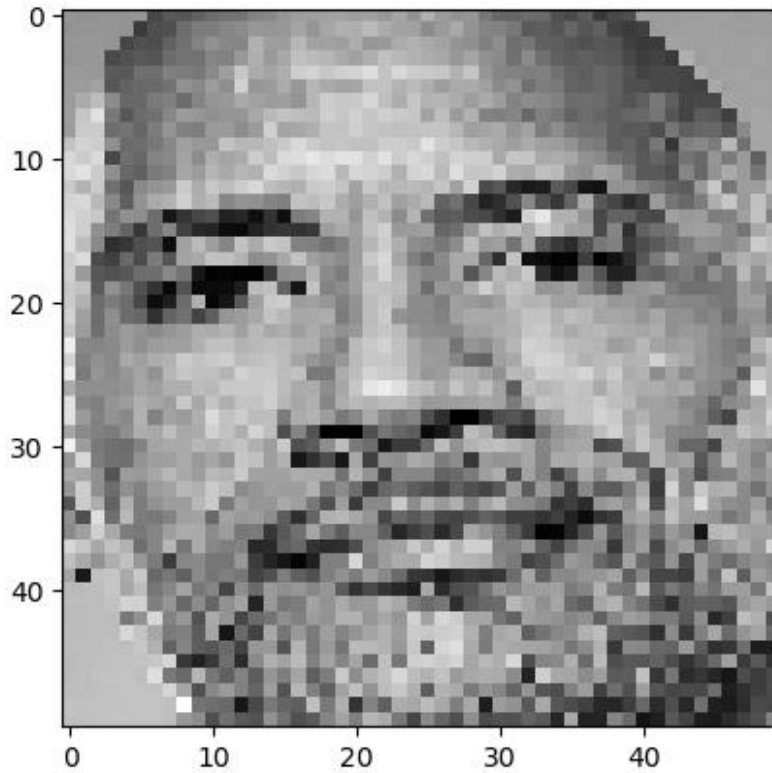
# Resizing the image
new_img_array = cv2.resize(crop_image, (img_size, img_size))

# Plotting the image
plt.imshow(new_img_array, cmap="gray")

1/1 [=====] - 0s 91ms/step
1/1 [=====] - 0s 89ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 20ms/step
9/9 [=====] - 0s 2ms/step
1/1 [=====] - 0s 130ms/step

<matplotlib.image.AxesImage at 0x7fddc2f79720>

```



```
# Reshaping the image
x = new_img_array.reshape(-1, 50, 50, 1)

# Normalizing
x = normalize(x, axis=1)

prediction = model.predict(x)
print(prediction)

1/1 [=====] - 0s 19ms/step
[[9.9989855e-01 1.0139993e-04]]

# Returns the index of the maximum value
res_1 = np.argmax(prediction)
if(res_1==1):
    print("Face With Mask")
elif(res_1==0):
    print("Face no Mask")
else:
    pass

Face no Mask
```

## **Student Vitae**

1. Name: Aditya Arvind Desai

Branch: Electrical Engineering

Date of Birth: 1503/2001

Address : Patgaon, Garagoti. Kolhapur

Email ID: 1908058@ritindia.edu