

**Face mask detection**

**The Egyptian E-Learning University (EELU)**

**Faculty of Information Technology**

**Graduation Project**

**2023-2024**

**Presented by:**

Abdelrahman Galal Mohamed Kermina Nabil Henen

Fady Ehab Youssef Sara Fahmy Ahmed

Taher Abdelazim Fayez Moatasem Mohamed Sadek

Ahmed Mohamed Abdellatif

**Supervised by:**

**Dr. Hanaa Bauomy**

**Eng. Doaa Mahmoud Hassan**

**STUDENTS CONTRIBUTION**

|  |  |  |
| --- | --- | --- |
| **Chapters** | **Title** | **Contribution** |
| 1 | Introduction | -Moatasem Mohamed Sadek  -Kermina Nabil Henen |
| 2 | Background | -Sara Fahmy Ahmed |
| 3 | Related work | -Taher Abdelazim Fayez  -Ahmed Mohamed Abdellatif |
| 4 | Analyses and Design | -Sara Fahmy Ahmed  -Moatasem Mohamed Sadek |
| 5 | Implementation | -Abdelrahman Galal Mohamed -Fady Ehab Youssef |
| 6 | Conclusion and Future Work | -Ahmed Mohamed Abdellatif  -Taher Abdelazim Fayez |

**Table of Contents**

***Chapter 1: Introduction to Project ......................................................................***

***1.1*** [***Introduction***](#Introduction)***....................................................................................................***

***1.2*** [***Motivation***](#motivations)***…………………………………………………………………..***

***1.2.1*** [***Public Health and Safety***](#PublicHealth)***…………………………………………………***

***1.2.2*** [***Compliance Monitoring***](#ComplianceMonitoring)***…………………………………………….…….***

***1.2.3*** [***Automation and Efficiency***](#Automation)***……………………………………………….***

***1.2.4*** [***Integration with Access Control Systems***](#Integration)***………………………………...***

***1.2.5*** [***Technology Innovation***](#Technology)***………………………………………….....……..***

***1.3*** [***Benefits***](#be) ***..........................................................................................................***

***1.3.1*** [***Public Health Enhancement***](#PublicHealthEnhancement)***......................................................................***

***1.3.2*** [***Reduced Transmission of Airborne Pathogens***](#ReducedTransmission)***........................................***

***1.3.3*** [***Operational Efficiency***](#OperationalEfficiency)***...............................................................................***

***1.3.4*** [***Integration with Access Control Systems***](#IntegratioAccessontrolSystems)***…..............................................***

***Chapter 2: Background......................................................................................***

***2.1*** [***over view about Deep Learning***](#ov)***....................................................................***

***2.2.1*** [***over view about Machine Learning and Techniques***](#OverviMachineLearning)***……….…………..***

***2.2.2*** [***Support Vector Machine (SVM)***](#svm) ***..............................................................***

***2.2.3*** [***K-Nearest Neighbors(KNN)***](#knn)***......................................................................***

***2.2.4 A Convolutional Neural Network (CNN)……………………………….***

***2.3.1*** [***Tools are used***](#Tools)***...........................................................................................***

***2.3.2*** [***Libraries are used***](#Libraries)***......................................................................................***

***2.4*** [**Dataset**](#data)  ***.......................................................................................................***

***Chapter 3:*** [***Related work***](#related)***....................................................................................***

***3.1*** [***Related work system***](#related)***....................................................................................***

***3.2*** [***Related work summary***](#relatedsummary)***...............................................................................***

***Chapter 4: Implementation And methods.…………………………………..***

***4.1*** [***Data Selection***](#DataSe)***……………...…………………………………………….***

***4.2*** [***Data Preprocessing***](#Datape)***………………………………………………………***

***4.3*** [***Model Selection and Why***](#Why)***………………………………………………..***

***4.4*** [***Model Evaluation***](#Modelvaluation)***………………………………………………………...***

***4.5*** [***Model Testing***](#ModelTesting)***…………………………………………………………….***

***4.6*** [***Diagrams***](#Diagrams)***…………………………………………………………………***

***Chapter 5: Conclusion & Future Work ........................................................***

***6.1*** [***Conclusion***](#Conclusion)***……………………………………………………………….***

***6.2*** [***Future work***](#Futurework) ***..............................................................................................***

[***Code***](#Code)***…………………………………………………………………………..***

[***Reference***](#Reference)***…………………………………………………..…………………***

**Chapter 1: Introduction to Project**

***1.1*** **Introduction**

Face mask detection has emerged as a critical application in computer vision, specifically designed to ascertain whether individuals are wearing face masks. This technology holds particular relevance in the context of the ongoing COVID-19 pandemic, where the use of face masks has proven to be a crucial preventive measure. By employing advanced image processing and machine learning techniques, face mask detection systems analyse images or video feeds to determine whether a person's face is appropriately covered.

The significance of this technology becomes evident in its potential to contribute to public health and safety. During contagious disease outbreaks like the COVID-19 pandemic, the wearing of face masks has been strongly recommended to mitigate the transmission of respiratory droplets and reduce the risk of infection. Face mask detection systems serve as a technological response to enforce compliance with these guidelines in various settings, providing a scalable and efficient solution to monitor mask usage.

So, our project revolves around this topic, which is, in short, applying a specific model such as SVM or KNN to detect a face mask. This model analyses some facial data to detect whether the face is covered with a mask, without a mask, or wearing it incorrectly.

***1.2*** **Motivation**

***1.2.1*** **Public He****alth and Safety**

The primary motivation behind face mask detection is to contribute to public health and safety, particularly during contagious disease outbreaks such as the COVID-19 pandemic. Wearing face masks has been recommended as an effective measure to reduce the transmission of respiratory droplets and protect individuals from airborne infections.

***1.2.2*** **Compliance Monitoring**

Face mask detection systems can be used in various public spaces, such as airports, public transportation, healthcare facilities, and retail stores, to monitor and enforce compliance with face mask mandates. This helps ensure that individuals follow recommended guidelines and regulations to minimize the risk of infection.

***1.2.3*****Automation and Efficiency**

Automated face mask detection systems offer a more efficient way to monitor large crowds and public spaces. Compared to manual monitoring, computer vision algorithms can quickly analyse numerous faces in real-time, providing a rapid and accurate assessment of mask usage.

***1.2.4*****Integration with Access Control Systems**

Face mask detection can be integrated with access control systems to manage entry to certain premises or facilities. For example, individuals without proper face mask coverage may be denied access to specific areas, reducing the risk of virus transmission within confined spaces.

***1.2.5*** **Technology Innovation**

The development of face mask detection systems showcases the innovative application of machine learning technologies to address real-world challenges. It highlights the adaptability of AI in contributing to public health measures and demonstrates how technology can play a role in crisis management.

***1.3*** **Benefits**

***1.3.1*** **Public Health Enhancement:**

Face mask detection contributes to public health by encouraging and ensuring compliance with recommended safety measures. This, in turn, helps to curb the spread of contagious diseases, particularly respiratory infections like COVID-19.

***1.3.2*** **Reduced Transmission of Airborne Pathogens**

By accurately identifying individuals without face masks, the technology aids in minimizing the transmission of airborne pathogens, providing an additional layer of protection in crowded or enclosed spaces.

***1.3.3*** **Operational Efficiency**

Automation of mask detection processes enhances operational efficiency, especially in high-traffic areas such as airports, public transportation, and retail outlets. It allows for swift identification of non-compliance, reducing the workload on human personnel.

***1.3.4*** **Integration with Access Control Systems**

Integration with access control systems enhances security protocols. Individuals without proper face mask coverage may be restricted from entering specific areas, helping to create a more controlled and safer environment.

**- In the end, the most important benefit to us from our project is the model’s ability to recognize the face with or without a mask, with high accuracy and efficiency, by identifying some features from the images and then recognizing them.**

**Chapter 2:** **Background**

***2.1*** **Overview about Deep Learning**

Deep learning is a subset of machine learning that involves training artificial neural networks on large amounts of data to make predictions or decisions without explicit programming. The term "deep" refers to the use of multiple layers in the neural networks, allowing them to learn hierarchical representations of data.

**Here is an overview of key concepts and components of deep learning:**

1.Neural Networks

Basic Unit - Neuron: The fundamental building block of neural networks, mimicking the structure and function of biological neurons.

Layers: Neural networks are organized into layers, including an input layer, one or more hidden layers, and an output layer. Deep learning involves using multiple hidden layers, distinguishing it from shallow or traditional machine learning models.

2.Deep Learning Architectures

Feedforward Neural Networks (FNN): The simplest form of neural networks where information flows in one direction, from the input layer to the output layer.

Convolutional Neural Networks (CNN): Suited for image and video analysis, CNNs use convolutional layers to automatically and adaptively learn spatial hierarchies of features.

Recurrent Neural Networks (RNN): Designed for sequential data, RNNs have connections that create loops, allowing information persistence over time. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are variations that address the vanishing gradient problem.

3.Training Process

Backpropagation: The primary algorithm used to train neural networks. It involves adjusting the weights of the connections based on the difference between predicted and actual outputs, minimizing the error.

Activation Functions: Functions applied to the output of neurons to introduce non-linearity and enable the network to learn complex patterns.

4.Optimization Techniques

Gradient Descent: An optimization algorithm used to minimize the error during training by adjusting the weights in the direction that reduces the error.

Stochastic Gradient Descent (SGD): A variant of gradient descent that updates weights after processing each training example.

Adam, RMSprop, etc.: Advanced optimization algorithms that enhance the convergence speed and stability of training.

***2.2.1* Overview about Machine Learning and Techniques**

Machine Learning (ML) is a subfield of artificial intelligence (AI) that focuses on developing algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. ML techniques leverage patterns and information from data to improve their performance over time.

**Here's an overview of key concepts and techniques in machine learning:**

1.Supervised Learning

Definition: In supervised learning, the model is trained on a labeled dataset, where each input is associated with a corresponding output.

Examples: Classification (assigning labels to inputs) and regression (predicting numeric values).

2.Unsupervised Learning

Definition: Unsupervised learning deals with unlabeled data, and the algorithm tries to find patterns or structures within the data without predefined categories.

Examples: Clustering (grouping similar data points) and dimensionality reduction.

3.Support Vector Machines (SVM)

Definition: A supervised learning algorithm that finds a hyperplane to separate data into different classes while maximizing the margin between classes.

Applications: Classification and regression tasks.

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It is particularly effective in scenarios where the data has clear margins of separation between different classes or groups. SVM aims to find the optimal hyperplane that maximally separates data points of different classes while minimizing the classification error.

***2.2.2*** **Support Vector Machine (SVM)**

**Here's an overview of how the Support Vector Machine algorithm works:**

1. Hyperplane

In a two-dimensional space, a hyperplane is a line that separates data points of different classes. In higher dimensions, it becomes a hyperplane.

2. Margins

The margin is the distance between the hyperplane and the nearest data point of either class. SVM aims to find the hyperplane with the maximum margin, as it is considered more robust and less prone to overfitting.

3. Support Vectors

Support vectors are the data points that lie closest to the decision boundary (hyperplane). These are crucial in defining the optimal hyperplane and maximizing the margin.

4. Kernel Trick

SVM can be extended to handle non-linear decision boundaries by using the kernel trick. This involves transforming the original feature space into a higher-dimensional space, where a linear separation becomes possible. Common kernel functions include the linear kernel, polynomial kernel, and radial basis function (RBF) kernel.

***2.2.3* K-Near****est Neighbors (KNN)**

The k-Nearest Neighbors (KNN) algorithm is a simple and intuitive supervised machine learning algorithm used for classification and regression tasks. It is a type of instance-based learning, where the model memorizes the entire training dataset and makes predictions based on the similarity between new data points and existing examples.

**Here's an overview of how the k-Nearest Neighbors algorithm works:**

1. Basic Idea

The main idea behind **KNN** is that similar data points tend to have similar labels or values. If you have a new data point, the algorithm looks at its k nearest neighbors in the training dataset and assigns the majority class (for classification) or averages the values (for regression) of those neighbors to make predictions.

2. Key Parameters

k (Number of Neighbors) The algorithm considers the "k" nearest neighbors when making predictions. The choice of "k" is a crucial parameter that influences the model's performance. A smaller "k" makes the model more sensitive to noise, while a larger "k" makes it more robust but potentially less flexible.

3. Distance Metric

The distance between data points is used to measure their similarity. Common distance metrics include Euclidean distance, Manhattan distance, Minkowski distance, etc. The choice of distance metric depends on the nature of the data and the problem at hand.

4. Training Phase

KNN doesn't have a conventional training phase. Instead, it stores the entire training dataset in memory.

5. Prediction Phase

When a new data point is provided, the algorithm calculates its distance from all data points in the training set.

It then identifies the k-nearest neighbors based on the calculated distances.

For **classification**, the algorithm assigns the most frequent class among the k-nearest neighbors to the new data point.

For regression, the algorithm averages the values of the k-nearest neighbors to predict the target value for the new data point.

6. Decision Boundaries

In the case of classification, KNN's decision boundaries are formed by the regions where the majority class changes. These boundaries can be complex and irregular, making KNN suitable for non-linear relationships in the data.

7. Pros and Cons

Pros:

* Simple and easy to understand.
* No training phase, which makes it suitable for dynamic datasets.
* Can handle both classification and regression tasks.

Cons:

* Computationally expensive, especially for large datasets.
* Sensitivity to irrelevant or redundant features.
* The choice of the distance metric and k can impact performance.

8. Use Cases

**KNN** is often used in recommendation systems, pattern recognition, and scenarios where the decision boundaries are complex and not easily defined by a simple mathematical formula.

***2.2.3* A Convolutional Neural Network (CNN)**

A Convolutional Neural Network (CNN) is a type of deep learning model that's particularly well-suited for analyzing visual imagery

CNNs are widely used in various applications such as image classification, object detection, image segmentation, and more. They have shown remarkable performance in tasks involving visual data and have become a cornerstone in the field of computer vision.

We use this model because we have a lot of images to deal with in our project

It's inspired by the structure and function of the human visual cortex. Here's a brief overview:

1. Convolutional Layers: These are the core building blocks of CNNs. They consist of filters (also called kernels) that slide over the input data, typically images, to extract features. Each filter detects specific patterns or features, such as edges or textures.

2. • Pooling Layers: After convolution, pooling layers are often used to reduce the dimensionality of the feature maps, making computation more efficient. Max pooling and average pooling are common techniques used in CNNs.

3. • Activation Functions: Activation functions, such as ReLU (Rectified Linear Activation) are applied to introduce non-linearity into the network, allowing it to learn complex patterns in the data.

4. • Fully Connected Layers: These layers connect every neuron in one layer to every neuron in the next layer, allowing the network to learn high-level features and make predictions based on the features extracted by earlier layers.

5. • Output Layer: The final layer of the CNN where the network's predictions are produced. Depending on the task, the output layer may have one or more neurons, each representing a class label in classification tasks, or a continuous value in regression tasks

***2.3.1*** **Tools are used**

**Anaconda and Jupyter** are tools commonly used in the field of data science and programming and we used it in our project.

* Anaconda

Description: Anaconda is a distribution of open-source software for data science, machine learning, and scientific computing. It includes a package manager (Conda), which simplifies the process of installing, managing, and updating various libraries and packages commonly used in these domains.

Key Features:

* Comes with popular programming languages such as Python and R.
* Simplifies the setup and management of environments for different projects.
* Provides a wide range of pre-installed libraries and tools used in data science and machine learning.
* Jupyter

Description: Jupyter is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It supports various programming languages, but it is most commonly associated with Python.

Key Features:

* Provides an interactive computing environment with support for code execution, data visualization, and text documentation.
* Breaks down code into cells, making it easy to run and test code in segments.
* Supports the creation of interactive and dynamic notebooks that combine code, text, and multimedia elements.
* Android Studio

Android Studio is the official integrated development environment (IDE) for Android app development. It provides tools and features specifically designed to streamline the process of creating, testing, and deploying Android applications.

Here are some key features of Android Studio:

User Interface Designer: Android Studio includes a powerful visual layout editor that allows developers to design their app's user interface by dragging and dropping elements onto a canvas. Developers can preview how their app will look on different Android devices and screen sizes.

Code Editor: Android Studio comes with a rich code editor that supports code completion, syntax highlighting, and refactoring tools. It also provides integration with version control systems like Git.

Build System: Android Studio uses Gradle as its build system, which allows developers to customize and automate the build process. This includes managing dependencies, compiling code, and packaging the app for distribution.

Emulator: Android Studio includes an emulator that allows developers to test their apps on virtual Android devices. Developers can simulate different device configurations, screen sizes, and Android versions to ensure their app works correctly across a wide range of devices.

***2.3.2*** **Libraries are used**

**1- Pandas**

is an open-source data manipulation and analysis library for Python. It provides data structures for efficiently storing and manipulating large datasets and offers a wide range of functions for data cleaning, exploration, and analysis. Pandas is a fundamental tool in the toolkit of data scientists, analysts, and researchers working with tabular and structured data.

**Key features and components of Pandas include:**

* Data Frame

The central data structure in Pandas is the Data Frame, a two-dimensional table with rows and columns. It allows you to store and manipulate structured data in a way similar to a spreadsheet or a SQL table.

* Series

A one-dimensional labelled array capable of holding any data type. A Data Frame is essentially a collection of Series.

Data Cleaning and Preprocessing:

Pandas provides a variety of functions for handling missing data, removing duplicates, and reshaping data.

**2- NumPy**

is an open-source numerical computing library for Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy is a fundamental library in the Python ecosystem for scientific computing and data analysis.

**Key features and components of NumPy include:**

* Arrays

NumPy provides a powerful ND array (n-dimensional array) object, which is the cornerstone of the library. These arrays are homogeneous and can contain elements of the same data type.

* Mathematical Functions

NumPy includes a wide range of mathematical functions for basic arithmetic, linear algebra, statistics, trigonometry, and more. These functions operate efficiently on NumPy arrays.

Random Number Generation:

NumPy provides functions for generating random numbers and random samples, which is useful for various applications, including simulations and statistical analysis.

**2- Tenserflow**

TensorFlow is an open-source library developed by the Google Brain team for machine learning and deep learning applications. It provides a comprehensive of tools, libraries, and community resources that allow researchers and developers to build and deploy machine learning models effectively. Here’s an overview of the

**key features and components of TensorFlow:**

1- Scalability: TensorFlow can run on multiple CPUs and GPUs, and it can be distributed across clusters of machines. This makes it suitable for large-scale machine learning tasks.

2- **Keras Integration**: TensorFlow integrates seamlessly with Keras, a high-level API that allows for easy and fast prototyping, making it accessible to beginners and experts alike.

3- **Visualization**: TensorFlow includes TensorBoard, a suite of visualization tools to understand, debug, and optimize machine learning models.

***2.4*** **Dataset**

Face Mask Detection by vijay Kumar it has 3000 images in RGB form with png, and jpg format with size 488 MG

**The dataset is splitted as follow:**

* With Mask : 1000 images
* Without mask : 1000 images
* Incorrect mask : 1000 images

**Chapter 3:** **Related work**

***3.1* Related work System**

CNN for Real-Time Face Mask Detection Based on Deep Learning Techniques.

**More than one person worked on this paper:**

Ismail Nasri, Mohammed Karrueche, Hajar Snoussi, Abdelhafid Messaoudi & Kamal Kassmi.

the extracted model can achieve an accuracy of more than 98% and the proposed method is CNN model.

A real time DNN-based face mask detection system using single shot multibook detector and MobileNetV2.

**More than one person worked on this paper:**

Preeti Nagrath, Rachna Jain, Agam Madan, Rohan Arora, Piyush Kataria, Jude Hemanth. This model can be used for safety purposes since it is very resource efficient to deploy. The proposed method is CNN model and accuracy of model is 93%.

A real time face mask detection system using convolutional neural network.

**More than one person worked on this paper:**

Hiten Goyal, Karanveer Sidana, Charanjeet Singh, Abhilasha Jain & Swati Jindal. and attained a performance accuracy rate of 98%. The proposed model is CNN model.

Face mask recogniser using image processing and computer vision approach.

**More than one person worked on this paper:**

A.K. Sharadhi, Vibhuti Gururaj, Sahana P. Shankar, M.S. Supriya, Neha Sanjay Choogle. The goal of this study is to create a new and improved real time face mask recogniser using image processing and computer vision approach. For the purpose of this study a pre-trained convolution neural network Mobile Net V2 was used. The proposed method is CNN model and accuracy of models is 98%.

COVID-19 Face Mask Detection Using CNN and Transfer Learning.

**More than one person worked on this paper:**

Cecilia Ajowho Adenusi, Olufunke Rebecca Vincent, Jesufunbi Abodunrin & Bukola Taibat Adebiyi. The goal of this research is to create an automated system to assist organizations, individuals, and government officials in monitoring people who wear or do not wear masks in public places.

The proposed model is CNN model, and his accuracy is 91,48%.

Real-Time Face Mask Detection Method Based on YOLOv3.

**More than one person worked on this paper:**

Xinbei Jiang, Tianhan Gao, Zichen Zhu, Yukang Zhao. The proposed method is SE-YOLOv3 model and accuracy of model is 99,64%. The goal of this model is to detect a person wearing a correct mask, without a mask, or wearing an incorrect mask.

Deep Neural Architecture for Face mask Detection on Simulated Masked Face Dataset against Covid-19 Pandemic.

**More than one person worked on this paper:**

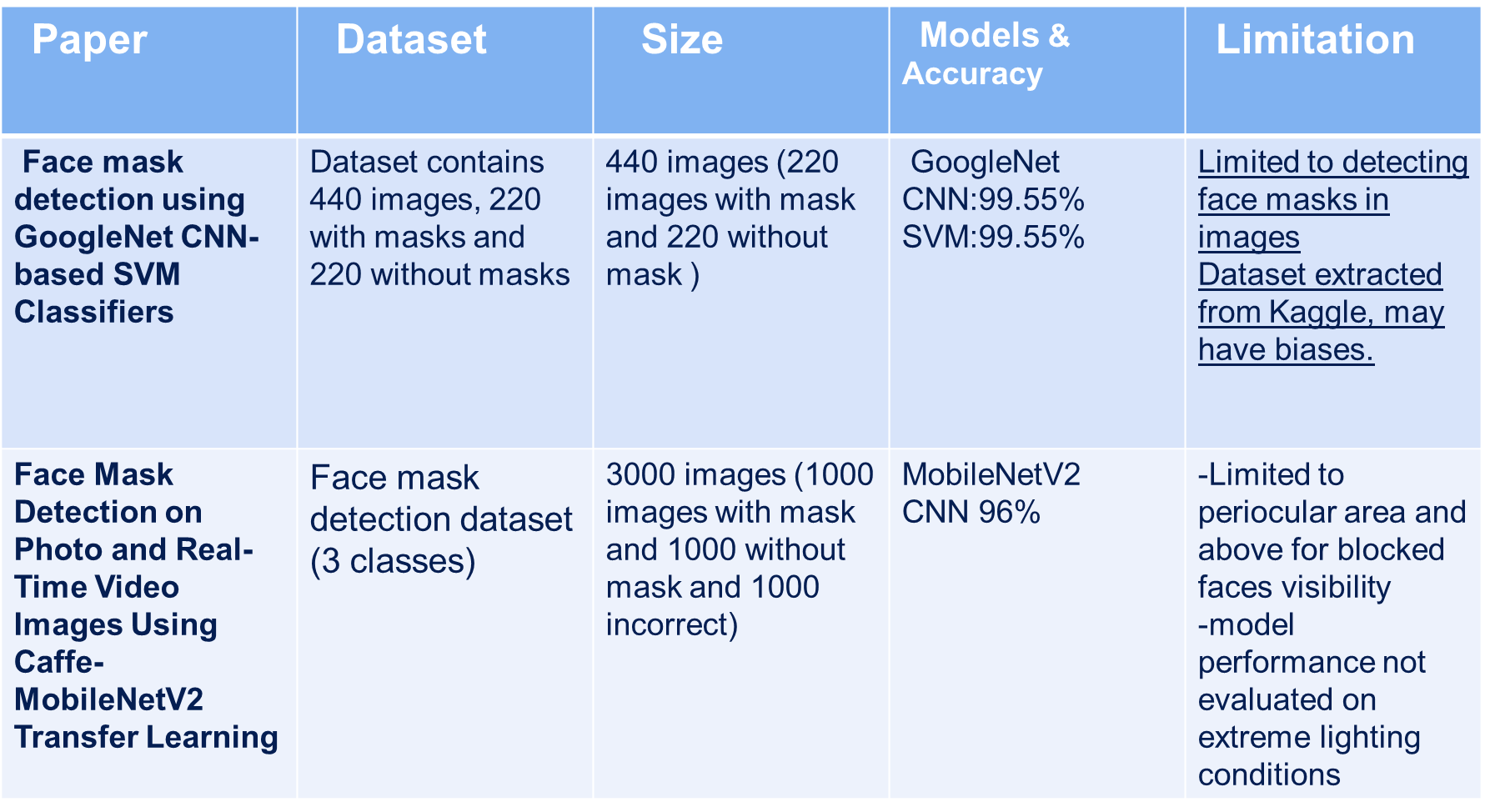
Alok Negi, Krishan Kumar, Prachi Chauhan, R. S. Rajput. This model is.

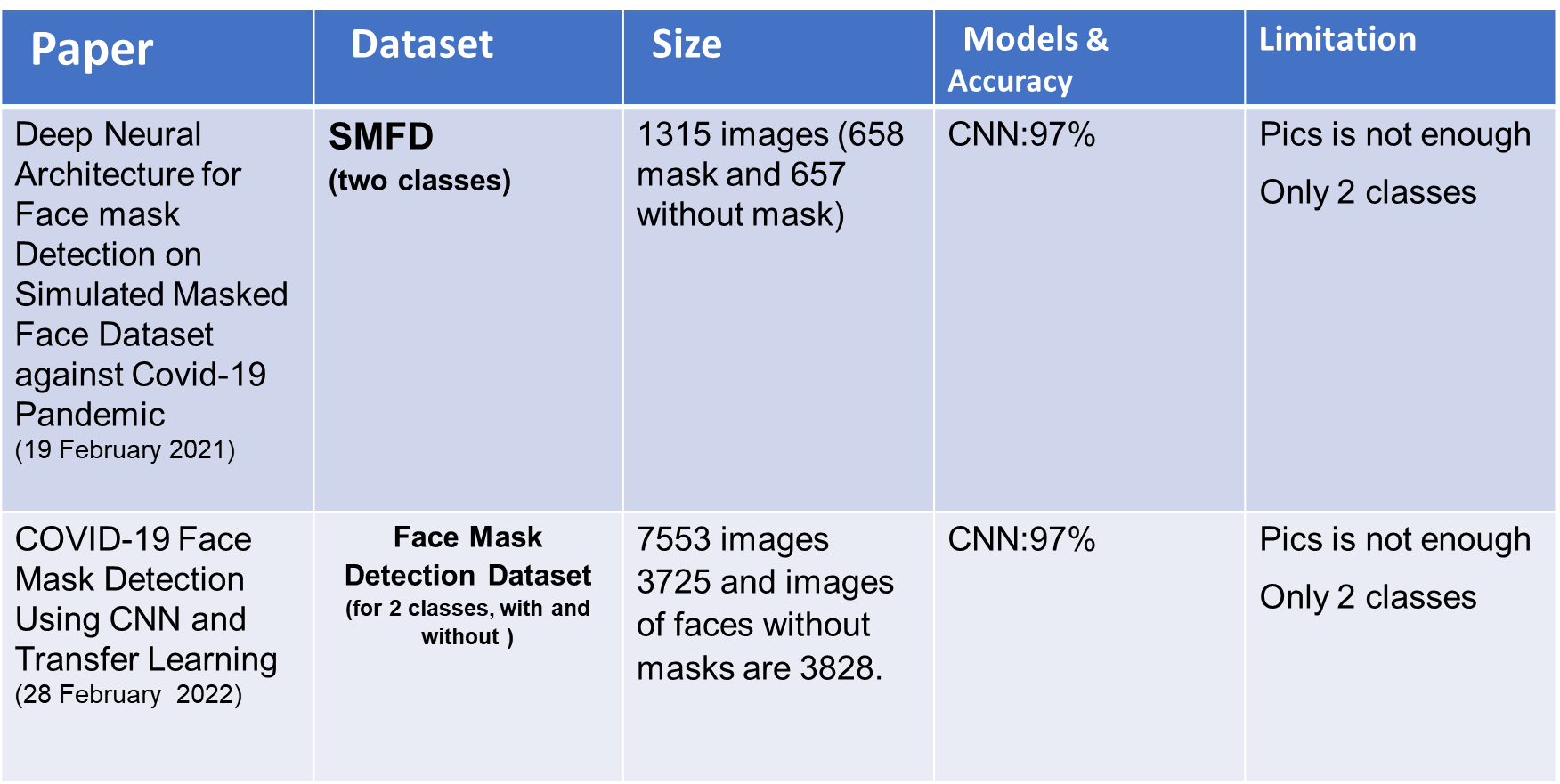
capable of recognizing masked and unmasked faces to help

***3.2*** **Related work summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accuracy | Model | Size | Paper | Dataset Name |
| MaskNet: 99%  SSDMNV2  93% | CNN (masknet) | 48 MB  1395 Images | ●CNN for Real-Time Face Mask Detection Based on Deep Learning Techniques● A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2 | The Prajna Bhandary dataset |
| CNN: 98% | CNN  Model | 133 MB  4000 Images | A real time face mask detection system using convolutional neural network | Omkar Gaurav dataset |
| CNN:98%  CNN:91.49% | CNN  Model | 171 MB  7553 pics | ● Face mask recognizer using image processing and computer vision approach● COVID-19 Face Mask Detection Using CNN and Transfer Learning | Face Mask Detection Dataset |
| Final accuracy: 99.64% | SE- YOLOv3 and more version YOLO | 1GB  Pics26062 | Real-Time Face Mask Detection Method Based on YOLOv3 | Properly-Wearing-Masked-Detect-Dataset |
| 95 percent accuracy | CNN model and VGG16 | 89MB  1315 images (658 mask and 657 without mask) | Deep Neural Architecture for Face mask Detection on Simulated Masked Face Dataset against Covid-19 pandemic | SMFD |
| 93% in general | SSDMNV2 | 165MB  632 Images | A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2 | YOLO medical mask dataset |

**Second Phase :-**





**Chapter 4****: Implementation And methods**

***4.1*** **Data Selection**

Lets talk about data we use data with the name Face Mask Detection By Vijay Kumar The number of images in this dataset are 3000 images with and the size of them is 488 MG and split into 3 categories with mask , without , incorrect the format of images include jpg and png and we make the resolution of this data 64 × 64 and the images in RGB format

Data Challenges The images is little little and some of the data need more preprocess to make the model identify it

***4.2*** **Data Preprocessing**

Image Loading:

First, we use library us we create an array for images that are in drive called images give it folder path.

Convert it to grayscale and resize it:

we convert it to grayscale, and we resize the image to 28\*28 to make it easier for training and spends little time compared to RGB.

Labelling:

After that give it label 1 for category that contain 3 types of pics and 0 for non-category

Normalization:

Every pixel that in the image have value between 0 to 255 because we use grayscale 8-byte the division on 255.0 for each pixel make them from range [0,1] original value of pixel [0,255] after normalization [0/255,1/255,] we make it because it helps in preventing one feature from dominating the others and can make optimization algorithms more stable.

Data splitting:

After that we split our data to test and train and the train 0.8 and the 0.2

**- Second Phase**

**Data Pre-processing: AI**

In PreProecessing step we use IMAGEDATAGENERATOR library

this library we use it for preprocess out data and make data augmentation for preventing the data to be overfitting by improving

our generalization next we will discuss our preprocess steps

- normalize our data to make our data rescaling between [1,0]

- rotate our data with 40 degree

- shift images horizontily up to 20% in width

- shift our images vertically up to 20% in height

- shear our images to 20% and thats means we stretch our images and this helps model to learn features

- applying zoom in and out to images

- horizontal flipping help the model to learn features more

- we apply after that fill mode :- nearest this mean that after we apply some transformations in our dataset

this will lead to find empty pixels fill mode nearest fill these empty pixels with the color of nearest pixel before tansformation

.

**GUI App**

**This file relation with (resultOverlayView)**

**This file takes an image that is uploaded from the device files**

**A screenshot of a computer

Description automatically generated**

**This interface is used to operate the camera to activate the model**

**A screenshot of a computer

Description automatically generated**

**Integration**

val modelFile = FileUtil.loadMappedFile(this, "model.tflite")  
val model = Interpreter(modelFile, Interpreter.Options())   
val labels = FileUtil.loadLabels(this, "labels.txt")

Load model.tflite from assets directorty and initialize a interpreter Tf lite after this load file label that also in assets that interpret the output of model

val cropSize = kotlin.math.min(input.width, input.height)  
val imageProcessor = ImageProcessor.Builder()  
 .add(ResizeWithCropOrPadOp(cropSize, cropSize))   
 .add(ResizeOp(inputShape[1], inputShape[2], ResizeOp.ResizeMethod.NEAREST\_NEIGHBOR))   
 .add(NormalizeOp(127.5f, 127.5f))   
 .build()

// load image  
inputImageBuffer.load(input)   
inputImageBuffer = imageProcessor.process(inputImageBuffer)

Make normalization and reshape image to fit the model and making it into square and and load input image into tensor image and apply preprocessing on image

model.run(inputImageBuffer.buffer, outputBuffer.buffer.rewind())

Execute tenserflow model with preprocessed image and save the results in output buffer

val labelOutput = TensorLabel(labels, outputBuffer)   
  
val label = labelOutput.mapWithFloatValue  
return label

Making tensor label and associate it with outputbuffer with provided label and convert output values into labels names and float values

Camera Integration

App use device camera to capture frame and detect face with ‘Face detector’ and app crop the face from the frame the cropped image is processed by tenseflow lite and detect if it wears mask or not after that abounding box is drawn around each face and detect if it wearing mask or not with score top the boundry and overlay view is responsible to appear it

***4.3* Model Selection and** **Why**

Support Vector Machine:

Why? because there is non-linearity relationship in data image SVM can actually deal with that with using kernel that can deal with it that if the face wearing mask or not and SVM so good for images that has pixel value.

K-nearest Neighbor:

Why? Knn is a simple algorithm that provide simplicity just depend on the number of k neighbors that are nearer to the point to identify it what is it from three categories so it’s easy in implementation addition to that it’s easy to change between dataset as it is a non-parametric algorithm and it handle non-linear relations

Convolution Neural Network:

we use our CNN model for these reasons :-

1- feature extraction

this can identify if there is mask or not and its position and our project depend on position of mask to ensure if its correct or not in wearing

2 - local connectivity

this depend on detecting edges and regions and this can help us to find the mask presents in any point in image

3 - translation invariance

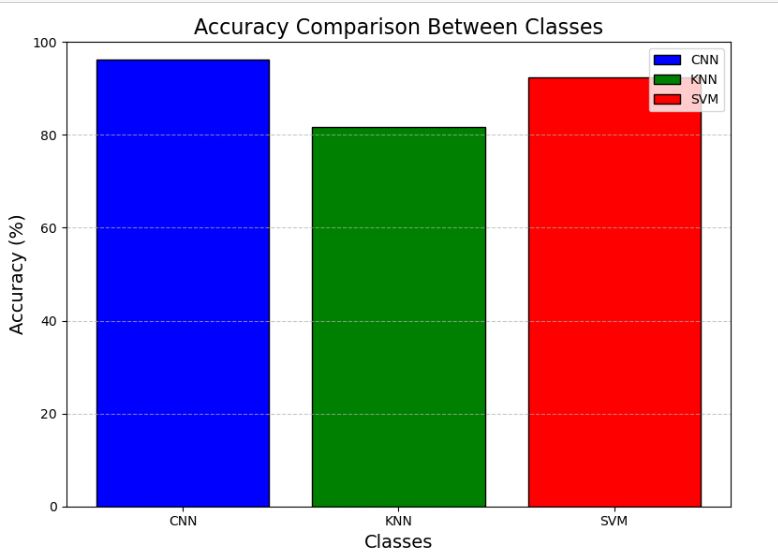
this feature determine the object place if they moved or scaled in image

4 - effiency

this can process high resolution images which is usefull for real-time mask detection

***4.4*** **Model Evaluation**

**In model Evaluation we use accuracy to find out the comprasion between 3 models**



***4.5*** **Model Testing**

In model testing we make this step to find out our project works or not

If you to try and test an image you just need to put path of picture on the code like that

new\_image\_path = '/content/drive/My Drive/without/002477.jpg'  # Path to the new image

But before that to know we reshape our image to (-1,1) to suit the model.

**K-Nearest neighbor, Support Vector machine**

The main difference in the result is that u should in the KNN define model categories 3 neighbor of like that: -

Pre-processed Image Shape:

Loaded Models:

with mask: KNeighborsClassifier(n\_neighbors=3)

without: KNeighborsClassifier(n\_neighbors=3)

withincorrt: KNeighborsClassifier(n\_neighbors=3)

The image most likely belongs to: without

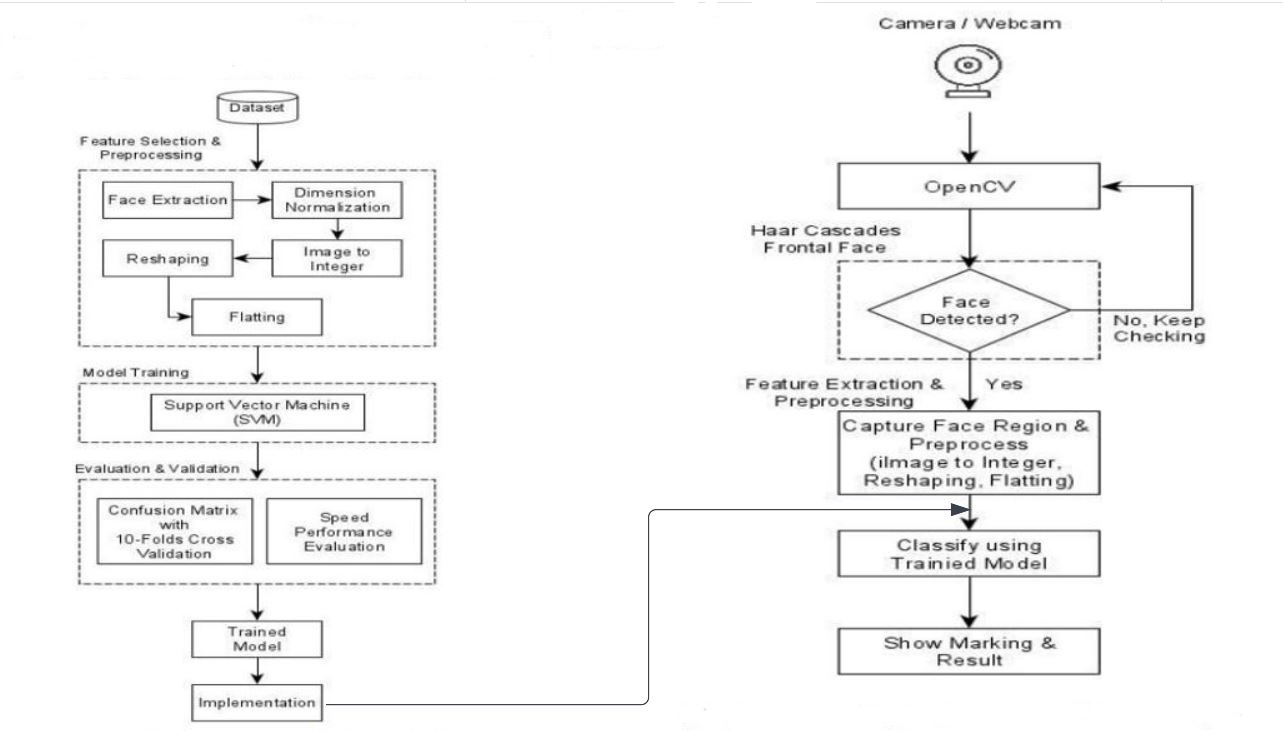
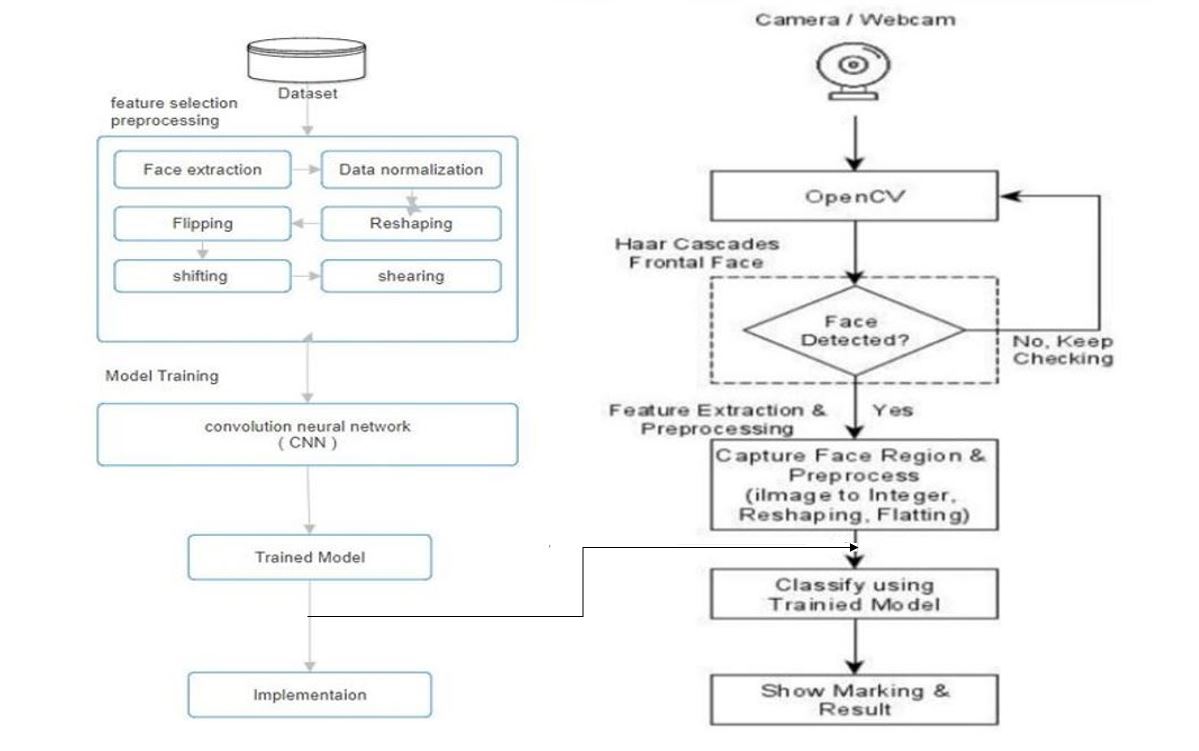
* **Second Phase**

In model testing we make this step to find out our project works or not

If you to try and test an image you just need to put path of picture on the code like that

img\_path = '/content/drive/MyDrive/New dataset/New dataset/With\_mask/simple1000.jpg'

***4.******6* Diagrams**

* ******
* 

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Training** | **predict** | **accuracy** |
| **CNN** | **This model takes the most time in training** | **The fastest one in predicting** | **96.3%** |
| **KNN** | **This model takes less time in training** | **This model is the slowest one in predicting** | **81.78%** |
| **SVM** | **Take a lot of time in training** | **This model is faster than KNN & slower than CNN** | **92.41%** |

**Chapter 6: Conclusion & Future work**

***6.1*** **Conclusion**

This pandemic has wreaked havoc on the entire world, and the world is now facing a massive health crisis as a result of the deadly disease that was declared by WHO in 2020, and governments and organizations struggled to control it before WHO issued preventive measures such as hand sanitizer, hand washing, and social distance. According to COVID-19 statistics, the transmission of the disease is higher in crowded areas, and many researchers have determined that wearing a mask in public spaces will immediately minimize the disease’s spread. As a result, governments in various countries have mandated the wearing of masks in public and crowded areas, making it extremely difficult to monitor people’s compliance with government rules, particularly in crowded areas. This is what prompted this research work, and we proposed a deep learning model that detects whether a person is wearing a mask or not. This model was created using CNN and image augmentation techniques were employed to improve the model’s performance in this study. The proposed method uses the SVM as an image classification algorithm that can detect whether an image contains a picture of people wearing or not wearing a face mask. Based on the experiments that have been conducted, the proposed method succeeded in obtaining % of accuracy, % of precision, % of recall and % of F-measure. In addition, the proposed method also shows fast computational performance.

***6.2*** **Future work**

* **combine between models** :Instead of relying on just one model (CNN, KNN, or SVM), you could combine their predictions using ensemble methods. Imagine your app asks each model if someone is wearing a mask, then takes a majority vote (most models agree) for the final answer. This can potentially make your app more accurate and reliable.
* **Robustness and Generalization**: CNNs can be sensitive to small variations in input data, leading to reduced performance in real-world scenarios. Future research focuses on developing techniques to improve the robustness and generalization of CNNs, making them more reliable across different domains and datasets.
* **Domain-Specific Applications**: CNNs have shown remarkable success in various domains, including computer vision, natural language processing, and healthcare. Future research will continue to explore domain-specific applications of CNNs, tailoring architectures and techniques to address specific challenges and requirements in different fields.

**Code**

SVM model

# prompt: data from drive

from google.colab import drive

drive.mount('/content/drive')

import os

import numpy as np

from PIL import Image

from sklearn import svm

from sklearn.model\_selection import train\_test\_split

#from utils import load\_and\_preprocess\_images

def load\_and\_preprocess\_images(folder\_path, is\_category):

    images = []

    labels = []

    for filename in os.listdir(folder\_path):

        if filename.endswith(".jpg") or filename.endswith(".png"):

            img\_path = os.path.join(folder\_path, filename)

            img = Image.open(img\_path).convert('L')  # Convert to grayscale

            img = img.resize((28, 28))  # Resize to 28x28 pixels

            img\_array = np.array(img)

            img\_array = img\_array.flatten()  # Flatten the 2D image to 1D for SVM

            images.append(img\_array)

            labels.append(int(is\_category))  # 1 for category, 0 for not-category

    return np.array(images), np.array(labels)

def train\_svm\_for\_category(category\_path, other\_paths):

    # Load images for the category

    cat\_images, cat\_labels = load\_and\_preprocess\_images(category\_path, True)

    # Load images from other categories as negative examples

    for path in other\_paths:

        other\_images, other\_labels = load\_and\_preprocess\_images(path, False)

        cat\_images = np.vstack((cat\_images, other\_images))

        cat\_labels = np.concatenate((cat\_labels, other\_labels))

    # Normalize images

    cat\_images = cat\_images / 255.0

    # Split data into training and testing sets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(cat\_images, cat\_labels, test\_size=0.2, random\_state=42)

    # Create an SVM classifier

    clf = svm.SVC(gamma='scale')

    # Train the SVM classifier

    clf.fit(X\_train, y\_train)

    # Evaluate the model

    accuracy = clf.score(X\_test, y\_test)

    print(f"Test Accuracy for {os.path.basename(category\_path)}: {accuracy \* 100:.2f}%")

    return clf

# prompt: Paths to your category folders from drive

categories = [r'/content/drive/My Drive/withmask', r'/content/drive/My Drive/without', r'/content/drive/My Drive/withincorrt']  # etc.

models = {}

for i, cat in enumerate(categories):

  # cat= categories

    other\_cats = categories[:i] + categories[i+1:]

    models[os.path.basename(cat)] = train\_svm\_for\_category(cat, other\_cats)

import numpy as np

from PIL import Image

def preprocess\_image(image\_path):

    img = Image.open(image\_path).convert('L')  # Convert to grayscale

    img = img.resize((28, 28))  # Resize to match training images

    img\_array = np.array(img)

    img\_array = img\_array.flatten()  # Flatten the image

    img\_array = img\_array / 255.0  # Normalize the image

    return img\_array

def predict\_category(image\_path, models):

    preprocessed\_image = preprocess\_image(image\_path)

    preprocessed\_image = preprocessed\_image.reshape(1, -1)  # Reshape for the model

    predictions = {category: model.predict(preprocessed\_image)[0] for category, model in models.items()}

 # Determine the most likely category

    most\_likely\_category = max(predictions, key=predictions.get)

    if predictions[most\_likely\_category] == 0:

        return "None of the categories"

    return most\_likely\_category

 # Use the trained models for prediction

new\_image\_path = '/content/drive/My Drive/without/002477.jpg'  # Path to the new image

predicted\_category = predict\_category(new\_image\_path, models)

print(f"The image most likely belongs to: {predicted\_category}")

Knn model

# prompt: data from drive

from google.colab import drive

drive.mount('/content/drive')

import os

import numpy as np

from PIL import Image

from sklearn.model\_selection import GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

#from utils import load\_and\_preprocess\_images

def load\_and\_preprocess\_images(folder\_path, is\_category):

    images = []

    labels = []

    for filename in os.listdir(folder\_path):

        if filename.endswith(".jpg") or filename.endswith(".png"):

            img\_path = os.path.join(folder\_path, filename)

            img = Image.open(img\_path).convert('L')  # Convert to grayscale

            img = img.resize((28, 28))  # Resize to 28x28 pixels

            img\_array = np.array(img)

            img\_array = img\_array.flatten()

            images.append(img\_array)

            labels.append(int(is\_category))  # 1 for category, 0 for not-category

    return np.array(images), np.array(labels)

def train\_knn\_for\_category(category\_path, other\_paths):

    # Load images for the category

    cat\_images, cat\_labels = load\_and\_preprocess\_images(category\_path, True)

    # Load images from other categories as negative examples

    for path in other\_paths:

        other\_images, other\_labels = load\_and\_preprocess\_images(path, False)

        cat\_images = np.vstack((cat\_images, other\_images))

        cat\_labels = np.concatenate((cat\_labels, other\_labels))

    # Normalize images

    cat\_images = cat\_images / 255.0

    # Split data into training and testing sets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(cat\_images, cat\_labels, test\_size=0.2, random\_state=42)

    #Creat Knn classifier

    knn\_classifier = KNeighborsClassifier(n\_neighbors=3)

    knn\_classifier.fit(X\_train, y\_train)

    y\_pred = knn\_classifier.predict(X\_test)

    accuracy = knn\_classifier.score(X\_test, y\_test)

    print(f"Test Accuracy for {os.path.basename(category\_path)}: {accuracy \* 100:.2f}%")

    return knn\_classifier

 # prompt: Paths to your category folders from drive

categories = [r'/content/drive/My Drive/withmask', r'/content/drive/My Drive/without', r'/content/drive/My Drive/withincorrt']

models = {}

for i, cat in enumerate(categories):

  # cat= categories

    other\_cats = categories[:i] + categories[i+1:]

    models[os.path.basename(cat)] = train\_knn\_for\_category(cat, other\_cats)

import numpy as np

from PIL import Image

def preprocess\_image(image\_path):

    img = Image.open(image\_path).convert('L')  # Convert to grayscale

    img = img.resize((28, 28))  # Resize to match training images

    img\_array = np.array(img)

    img\_array = img\_array.flatten()  # Flatten the image

    img\_array = img\_array / 255.0  # Normalize the image

    return img\_array

def predict\_category(image\_path, models):

    preprocessed\_image = preprocess\_image(image\_path)

    preprocessed\_image = preprocessed\_image.reshape(1, -1)  # Reshape for the model

    print(f"Preprocessed Image Shape: {preprocessed\_image.shape}")

    print("Loaded Models:")

    for category, model in models.items():

        print(f"{category}: {model}")

    predictions = {category: model.predict(preprocessed\_image)[0] for category, model in models.items()}

    # Determine the most likely category

    most\_likely\_category = max(predictions, key=predictions.get)

    if predictions[most\_likely\_category] == 0:

        return "None of the categories"

    return most\_likely\_category

# Use the trained models for prediction

new\_image\_path = '/content/drive/My Drive/without/002477.jpg'  # Path to the new image

predicted\_category = predict\_category(new\_image\_path, models)

print(f"The image most likely belongs to: {predicted\_category}")

CNN model

# Define constants

IMG\_HEIGHT, IMG\_WIDTH = 150, 150

BATCH\_SIZE = 32

NUM\_CLASSES = 3

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train\_dir = r'/content/drive/MyDrive/New dataset/New dataset'

val\_dir = r'/content/drive/MyDrive/New dataset/New dataset'

test\_dir = r'/content/drive/MyDrive/New dataset/New dataset'

# Data preprocessing and augmentation

train\_datagen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=40,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    fill\_mode='nearest'

)

val\_datagen = ImageDataGenerator(rescale=1./255)

test\_datagen = ImageDataGenerator(rescale=1./255)

val\_dataset = val\_datagen.flow\_from\_directory(

    val\_dir,

    target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

    batch\_size=BATCH\_SIZE,

    class\_mode='categorical'

)

test\_dataset = test\_datagen.flow\_from\_directory(

    test\_dir,

    target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

    batch\_size=BATCH\_SIZE,

    class\_mode='categorical'

)

# Define the CNN model architecture

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(IMG\_HEIGHT, IMG\_WIDTH, 3)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dropout(0.5),

    layers.Dense(512, activation='relu'),

    layers.Dense(NUM\_CLASSES, activation='softmax')  # 3 output classes: mask correctly worn, mask incorrectly worn, no mask

])

train\_dataset = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

    batch\_size=BATCH\_SIZE,

    class\_mode='categorical'

)

model.compile(optimizer='adam',

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

history = model.fit(

    train\_dataset,

    epochs=10,

    validation\_data=val\_dataset

)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(test\_dataset)

print('Test accuracy:', test\_acc)

# Save the model

model.save('facce\_mask\_classifier.keras')

import numpy as np

from tensorflow.keras.preprocessing import image

# Load the saved model

loaded\_model = tf.keras.models.load\_model('facce\_mask\_classifier.keras')

# Load and preprocess a single image for testing

img\_path = '/content/drive/MyDrive/New dataset/New dataset/With\_mask/simple1000.jpg'

img = image.load\_img(img\_path, target\_size=(IMG\_HEIGHT, IMG\_WIDTH))

img\_array = image.img\_to\_array(img)

img\_array = np.expand\_dims(img\_array, axis=0)  # Add batch dimension

img\_array /= 255.0  # Normalize pixel values

prediction = loaded\_model.predict(img\_array)

# Get class labels

class\_labels = list(test\_dataset.class\_indices.keys())

# Convert prediction to class label

predicted\_label = class\_labels[np.argmax(prediction)]

print("Predicted Label:", predicted\_label)

import tensorflow as tf

# Load the existing model

model = tf.keras.models.load\_model('facce\_mask\_classifier.keras')

# Convert the model to the TensorFlow Lite format

converter = tf.lite.TFLiteConverter.from\_keras\_model(model)

tflite\_model = converter.convert()

# Save the TensorFlow Lite model to a file

with open('facce\_maask\_claassifier.tflite', 'wb') as f:

    f.write(tflite\_model)

**Code App**

mainActivity.kt

package com.example.aps1

import android.app.Activity

import android.content.Intent

import android.graphics.\*

import android.os.Bundle

import android.provider.MediaStore

import android.view.View

import android.widget.Toast

import androidx.appcompat.app.AlertDialog

import androidx.appcompat.app.AppCompatActivity

import com.google.android.gms.vision.Frame

import com.google.android.gms.vision.face.FaceDetector

import kotlinx.android.synthetic.main.activity\_main.\*

import org.tensorflow.lite.Interpreter

import org.tensorflow.lite.support.common.FileUtil

import org.tensorflow.lite.support.common.ops.NormalizeOp

import org.tensorflow.lite.support.image.ImageProcessor

import org.tensorflow.lite.support.image.TensorImage

import org.tensorflow.lite.support.image.ops.ResizeOp

import org.tensorflow.lite.support.image.ops.ResizeWithCropOrPadOp

import org.tensorflow.lite.support.label.TensorLabel

import org.tensorflow.lite.support.tensorbuffer.TensorBuffer

import java.io.ByteArrayOutputStream

import kotlin.math.min

class MainActivity : AppCompatActivity() {

private val IMAGE\_PICKER\_REQUEST\_CODE = 123

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

cameraView.setLifecycleOwner(this)

// Create a FaceDetector

val faceDetector = FaceDetector.Builder(this).setTrackingEnabled(true).build()

if (!faceDetector.isOperational) {

AlertDialog.Builder(this)

.setMessage("Could not set up the face detector!")

.show()

}

cameraView.addFrameProcessor { frame ->

val matrix = Matrix()

matrix.setRotate(frame.rotationToUser.toFloat())

if (frame.dataClass === ByteArray::class.java) {

val out = ByteArrayOutputStream()

val yuvImage = YuvImage(

frame.getData(),

ImageFormat.NV21,

frame.size.width,

frame.size.height,

null

)

yuvImage.compressToJpeg(

Rect(0, 0, frame.size.width, frame.size.height), 100, out

)

val imageBytes = out.toByteArray()

var bitmap = BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.size)

bitmap = bitmap.copy(Bitmap.Config.ARGB\_8888, true)

bitmap = Bitmap.createBitmap(bitmap, 0, 0, bitmap.width, bitmap.height, matrix, true)

bitmap = Bitmap.createScaledBitmap(bitmap, overlayView.width, overlayView.height, true)

overlayView.boundingBox = processBitmap(bitmap, faceDetector)

overlayView.invalidate()

} else {

Toast.makeText(this, "Camera Data not Supported", Toast.LENGTH\_LONG).show()

}

}

}

private fun processBitmap(bitmap: Bitmap, faceDetector: FaceDetector ): MutableList<Box> {

val boundingBoxList = mutableListOf<Box>()

// Detect the faces

val frame = Frame.Builder().setBitmap(bitmap).build()

val faces = faceDetector.detect(frame)

if(faces.size()==0){

boundingBoxList.add(Box(RectF( ( bitmap.width.toFloat()/12), ( bitmap.height.toFloat()/12), bitmap.width.toFloat() - ( bitmap.width.toFloat()/12), bitmap.height.toFloat()- ( bitmap.height.toFloat()/12)), "Incorrect (No Faces)", false))

}

// Mark out the identified face

for (i in 0 until faces.size()) {

val thisFace = faces.valueAt(i)

val left = thisFace.position.x

val top = thisFace.position.y

val right = left + thisFace.width

val bottom = top + thisFace.height

val bitmapCropped = Bitmap.createBitmap(

bitmap,

left.toInt(),

top.toInt(),

if (right.toInt() > bitmap.width) {

bitmap.width - left.toInt()

} else {

thisFace.width.toInt()

},

if (bottom.toInt() > bitmap.height) {

bitmap.height - top.toInt()

} else {

thisFace.height.toInt()

}

)

val label = predict(bitmapCropped)

val with = label["WithMask"] ?: 0F

val without = label["WithoutMask"] ?: 0F

var predictions = if (with > without) {

"With Mask : " + String.format("%.1f", with \* 100) + "%"

} else {

"Without Mask : " + String.format("%.1f", without \* 100) + "%"

}

boundingBoxList.add(Box(RectF(left, top, right, bottom), predictions, with > without))

}

return boundingBoxList

}

private fun predict(input: Bitmap): MutableMap<String, Float> {

// load model

val modelFile = FileUtil.loadMappedFile(this, "model.tflite")

val model = Interpreter(modelFile, Interpreter.Options())

val labels = FileUtil.loadLabels(this, "labels.txt")

// data type

val imageDataType = model.getInputTensor(0).dataType()

val inputShape = model.getInputTensor(0).shape()

val outputDataType = model.getOutputTensor(0).dataType()

val outputShape = model.getOutputTensor(0).shape()

var inputImageBuffer = TensorImage(imageDataType)

val outputBuffer = TensorBuffer.createFixedSize(outputShape, outputDataType)

// preprocess

val cropSize = min(input.width, input.height)

val imageProcessor = ImageProcessor.Builder()

.add(ResizeWithCropOrPadOp(cropSize, cropSize))

.add(ResizeOp(inputShape[1], inputShape[2], ResizeOp.ResizeMethod.NEAREST\_NEIGHBOR))

.add(NormalizeOp(127.5f, 127.5f))

.build()

// load image

inputImageBuffer.load(input)

inputImageBuffer = imageProcessor.process(inputImageBuffer)

// run model

model.run(inputImageBuffer.buffer, outputBuffer.buffer.rewind())

// get output

val labelOutput = TensorLabel(labels, outputBuffer)

val label = labelOutput.mapWithFloatValue

return label

}

fun fromDevice(view: View) {

val intent = Intent(Intent.ACTION\_PICK, MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI)

startActivityForResult(intent, IMAGE\_PICKER\_REQUEST\_CODE)

}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {

super.onActivityResult(requestCode, resultCode, data)

if (resultCode == Activity.RESULT\_OK && requestCode == IMAGE\_PICKER\_REQUEST\_CODE) {

val selectedImageUri = data?.data

if (selectedImageUri != null) {

// Start ResultActivity and pass the bitmap and bounding boxes

val intent = Intent(this, ResultActivity::class.java)

intent.putExtra("uri", selectedImageUri.toString())

startActivity(intent)

}

}

}

}

**Box.kt**

package com.example.aps1

import android.graphics.RectF

import android.os.Parcel

import android.os.Parcelable

data class Box(

val rect: RectF,

val label: String,

val withMask: Boolean

) : Parcelable {

constructor(parcel: Parcel) : this(

parcel.readParcelable(RectF::class.java.classLoader)!!,

parcel.readString()!!,

parcel.readByte() != 0.toByte()

)

override fun writeToParcel(parcel: Parcel, flags: Int) {

parcel.writeParcelable(rect, flags)

parcel.writeString(label)

parcel.writeByte(if (withMask) 1 else 0)

}

override fun describeContents(): Int {

return 0

}

companion object CREATOR : Parcelable.Creator<Box> {

override fun createFromParcel(parcel: Parcel): Box {

return Box(parcel)

}

override fun newArray(size: Int): Array<Box?> {

return arrayOfNulls(size)

}

}

}

**OverlayView.kt**

package com.example.aps1

import android.graphics.Canvas

import android.content.Context

import android.graphics.Color

import android.graphics.Paint

import android.util.AttributeSet

import android.view.View

class OverlayView @JvmOverloads constructor(

context: Context, attrs: AttributeSet? = null, defStyleAttr: Int = 0

) : View(context, attrs, defStyleAttr) {

var boundingBox: MutableList<Box> = mutableListOf()

var paint = Paint()

override fun draw(canvas: Canvas) {

super.draw(canvas)

paint.style = Paint.Style.STROKE

paint.strokeWidth = 3f

paint.strokeCap = Paint.Cap.ROUND

paint.strokeJoin = Paint.Join.ROUND

paint.strokeMiter = 100f

boundingBox.forEach { box ->

if (box.withMask){

paint.color = Color.GREEN

} else {

paint.color = Color.RED

}

paint.setTextAlign(Paint.Align.LEFT)

paint.textSize = 54f // Set the text size to 24 (adjust size as needed)

canvas.drawText(box.label, box.rect.left, box.rect.top-9F, paint)

canvas.drawRoundRect(box.rect, 2F, 2F, paint)

}

}

}

**ResultActivity.kt**

package com.example.aps1

import android.graphics.Bitmap

import android.graphics.BitmapFactory

import android.graphics.Canvas

import android.graphics.Color

import android.graphics.ImageDecoder

import android.graphics.Matrix

import android.graphics.RectF

import android.net.Uri

import android.os.Build

import android.os.Bundle

import android.provider.MediaStore

import android.view.ViewTreeObserver

import androidx.appcompat.app.AlertDialog

import androidx.appcompat.app.AppCompatActivity

import com.google.android.gms.vision.Frame

import com.google.android.gms.vision.face.FaceDetector

import kotlinx.android.synthetic.main.activity\_main.overlayView

import kotlinx.android.synthetic.main.activity\_result.resultImageView

import kotlinx.android.synthetic.main.activity\_result.resultOverlayView

import org.tensorflow.lite.Interpreter

import org.tensorflow.lite.support.common.FileUtil

import org.tensorflow.lite.support.common.ops.NormalizeOp

import org.tensorflow.lite.support.image.ImageProcessor

import org.tensorflow.lite.support.image.TensorImage

import org.tensorflow.lite.support.image.ops.ResizeOp

import org.tensorflow.lite.support.image.ops.ResizeWithCropOrPadOp

import org.tensorflow.lite.support.label.TensorLabel

import org.tensorflow.lite.support.tensorbuffer.TensorBuffer

import kotlin.math.min

class ResultActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_result)

val viewTreeObserver = resultOverlayView.viewTreeObserver

if (viewTreeObserver.isAlive) {

viewTreeObserver.addOnGlobalLayoutListener(object : ViewTreeObserver.OnGlobalLayoutListener {

override fun onGlobalLayout() {

resultOverlayView.viewTreeObserver.removeOnGlobalLayoutListener(this)

val overlayWidth = resultOverlayView.width

val overlayHeight = resultOverlayView.height

load(overlayWidth, overlayHeight)

}

})

}

}

fun load(overlayWidth: Int, overlayHeight: Int) {

val uri = intent.getStringExtra("uri")

val selectedImageUri = Uri.parse(uri)

val bitmap = if (Build.VERSION.SDK\_INT < 28) {

MediaStore.Images.Media.getBitmap(this.contentResolver, selectedImageUri)

} else {

val source = ImageDecoder.createSource(this.contentResolver, selectedImageUri)

ImageDecoder.decodeBitmap(source) { decoder, \_, \_ ->

decoder.setTargetSampleSize(1)

decoder.isMutableRequired = true

}

}

val faceDetector = FaceDetector.Builder(this).setTrackingEnabled(true).build()

if (!faceDetector.isOperational) {

AlertDialog.Builder(this)

.setMessage("Could not set up the face detector!")

.show()

return

}

val transformedBitmap = transformBitmap(bitmap, overlayWidth, overlayHeight)

val boundingBoxes = processBitmap(transformedBitmap, faceDetector)

resultImageView.setImageBitmap(transformedBitmap)

resultOverlayView.boundingBox = boundingBoxes

resultOverlayView.invalidate()

}

private fun transformBitmap(bitmap1: Bitmap, overlayWidth: Int, overlayHeight: Int): Bitmap {

val matrix = Matrix()

var bitmap = bitmap1

// Ensure bitmap is in ARGB\_8888 format

bitmap = bitmap.copy(Bitmap.Config.ARGB\_8888, true)

// Calculate the scale factor

val scaleWidth = overlayWidth.toFloat() / bitmap.width

val scaleHeight = overlayHeight.toFloat() / bitmap.height

val scaleFactor = minOf(scaleWidth, scaleHeight)

// Calculate new width and height based on the scale factor

val newWidth = (bitmap.width \* scaleFactor).toInt()

val newHeight = (bitmap.height \* scaleFactor).toInt()

// Create a scaled bitmap

bitmap = Bitmap.createScaledBitmap(bitmap, newWidth, newHeight, true)

// Create a new bitmap with the desired overlay size and black background

val finalBitmap = Bitmap.createBitmap(overlayWidth, overlayHeight, Bitmap.Config.ARGB\_8888)

val canvas = Canvas(finalBitmap)

canvas.drawColor(Color.BLACK) // Fill the background with black

// Calculate the position to center the scaled bitmap

val left = (overlayWidth - newWidth) / 2

val top = (overlayHeight - newHeight) / 2

// Draw the scaled bitmap onto the canvas

canvas.drawBitmap(bitmap, left.toFloat(), top.toFloat(), null)

return finalBitmap

}

private fun processBitmap(bitmap: Bitmap, faceDetector: FaceDetector): MutableList<Box> {

val boundingBoxList = mutableListOf<Box>()

val frame = Frame.Builder().setBitmap(bitmap).build()

val faces = faceDetector.detect(frame)

if(faces.size()==0){

boundingBoxList.add(Box(RectF( ( bitmap.width.toFloat()/12), ( bitmap.height.toFloat()/12), bitmap.width.toFloat() - ( bitmap.width.toFloat()/12), bitmap.height.toFloat()- ( bitmap.height.toFloat()/12)), "Incorrect (No Faces)", false))

}

for (i in 0 until faces.size()) {

val thisFace = faces.valueAt(i)

val left = thisFace.position.x

val top = thisFace.position.y

val right = left + thisFace.width

val bottom = top + thisFace.height

val croppedBitmap = Bitmap.createBitmap(bitmap, left.toInt(), top.toInt(), min(thisFace.width.toInt(), bitmap.width - left.toInt()), min(thisFace.height.toInt(), bitmap.height - top.toInt()))

val label = predict(croppedBitmap)

val withMaskProbability = label["WithMask"] ?: 0F

val withoutMaskProbability = label["WithoutMask"] ?: 0F

val predictions = if (withMaskProbability > withoutMaskProbability) {

"With Mask: ${String.format("%.1f", withMaskProbability \* 100)}%"

} else {

"Without Mask: ${String.format("%.1f", withoutMaskProbability \* 100)}%"

}

boundingBoxList.add(Box(RectF(left, top, right, bottom), predictions, withMaskProbability > withoutMaskProbability))

}

return boundingBoxList

}

private fun predict(input: Bitmap): MutableMap<String, Float> {

val modelFile = FileUtil.loadMappedFile(this, "model.tflite")

val model = Interpreter(modelFile, Interpreter.Options())

val labels = FileUtil.loadLabels(this, "labels.txt")

val imageDataType = model.getInputTensor(0).dataType()

val inputShape = model.getInputTensor(0).shape()

val outputDataType = model.getOutputTensor(0).dataType()

val outputShape = model.getOutputTensor(0).shape()

var inputImageBuffer = TensorImage(imageDataType)

val outputBuffer = TensorBuffer.createFixedSize(outputShape, outputDataType)

val cropSize = min(input.width, input.height)

val imageProcessor = ImageProcessor.Builder()

.add(ResizeWithCropOrPadOp(cropSize, cropSize))

.add(ResizeOp(inputShape[1], inputShape[2], ResizeOp.ResizeMethod.NEAREST\_NEIGHBOR))

.add(NormalizeOp(127.5f, 127.5f))

.build()

inputImageBuffer.load(input)

inputImageBuffer = imageProcessor.process(inputImageBuffer)

model.run(inputImageBuffer.buffer, outputBuffer.buffer.rewind())

val labelOutput = TensorLabel(labels, outputBuffer)

return labelOutput.mapWithFloatValue

}

**Reference**

|  |  |
| --- | --- |
| *How knn algorithm works* | <https://youtu.be/UqYde-LULfs?si=zPVy6TLMZQABt_5T> |
| *Support Vector Machine (SVM)* | <https://youtu.be/_YPScrckx28?si=qPqBrc8PiYoHHodK> |
| *Paper* | <https://www.mdpi.com/2079-9292/10/7/837> |
| *MaskNet: CNN for Real-Time Face Mask Detection Based on Deep Learning Techniques* | <https://link.springer.com/chapter/10.1007/978-3-030-73882-2_9> |
| *SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2* | <https://www.sciencedirect.com/science/article/pii/S2210670720309070> |
| *A real time face mask detection system using convolutional neural network* | <https://link.springer.com/article/10.1007/s11042-022-12166-x> |
| *[PDF] Deep Neural Architecture for Face mask Detection on Simulated Masked Face Dataset against Covid-19 Pandemic Semantic Scholar* | <https://ieeexplore.ieee.org/document/9397196> |

**Dataset**

|  |  |
| --- | --- |
| *YOLO medical mask dataset* | <https://www.kaggle.com/datasets/gooogr/yolo-medical-mask-dataset?fbclid=IwAR2sPezMahzxdeBSYQI0zTBjfwPhTF_j5KIIEnsb7bOtUMCO346uDD2K7EQ> |
| *Properly-Wearing-Masked Detect Dataset.*  *口罩人脸数据集* | <https://github.com/ethancvaa/Properly-Wearing-Masked-Detect-Dataset> |
| *Face Mask Detection Dataset* | <https://www.kaggle.com/datasets/omkargurav/face-mask-dataset> |
| *Face Mask Detection Data* | <https://www.kaggle.com/datasets/aneerbanchakraborty/face-mask-detection-data?select=with_mask> |
| *Face Mask Detection Dataset (MaskNet)* | <https://www.kaggle.com/datasets/ismailnasri20/face-mask-detection-dataset-masknet> |