IMAGE BASED RECOGNITION AND CLASSIFICATION

Mini Project - Report submitted by

KSHITIZ MANGAL

(4NM21CS082)

EMAD HABIBI

(4NM21CS058)

6th Semester B.E.

Under the Guidance of

DR PRADEEP KANCHAN

Assistant Professor Gd-III

DR SARIKA HEGDE

Professor

In partial fulfillment of the requirements for the award of

the Degree of

Bachelor of Engineering in Computer Science and Engineering

from

Visvesvaraya Technological University, Belagavi

Department of Computer Science and Engineering

NMAM Institute of Technology, Nitte - 574110

(An Autonomous Institution affiliated to VTU, Belagavi)

MAY 2024

N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Mini project work entitled

“ ……………………………………………………………………………… ”

is a bonafide work carried out by

KSHITIZ MANGAL	**EMAD HABIBI**
(4NM21CS082)	(4NM21CS058)

of 6th Semester B.E. in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering
prescribed by ***Visvesvaraya Technological University, Belagavi***
during the year ***2023-2024.***

______________________ ______________________

Signature of the Guide ***Signature of the HOD***

Viva Voce Examination

Name of the Examiners	**Signature with Date**
1. ____________________________	____________________________
2. ____________________________	____________________________

ACKNOWLEDGEMENTS

I express my deepest gratitude to our guides, Dr Pradeep Kanchan and Dr Sarika Hegde, whose wisdom, patience, and support have been the cornerstone of this project. Their expertise and insights have guided us through challenges and paved paths to success of this project. We are grateful to our Institution for providing this opportunity to create this project.

My thanks to our beloved HOD, Dr. Jyothi Shetty, for extending support for carrying out this project in the department and providing us with all the necessary facilities.

My sincere thanks to our beloved principal Dr. Niranjan N Chiplunkar for permitting us to carry out this project at our college and providing us with all the necessary facilities.

Finally, thanks to the staff members of the department of Computer Science and Engineering and my friends for their honest opinions and suggestions throughout the course of my internship.

ABSTRACT

A hybrid model using deep and classical machine learning for detecting face masks will be presented. A dataset is used to build this face mask detector using Python, OpenCV, TensorFlow and Keras. While entering the place everyone should scan their face and then enter, ensuring they have a mask with them. If anyone is found to be without a face mask, a beep alert will be generated. As all the workplaces are opening. It detects objects using cameras and can alert in cases of dangerous situations.

TABLE OF CONTENTS

INTRODUCTION

GENERAL

In today's era of rapid technological advancement, computer vision powered by deep learning methodologies has emerged as a cornerstone of innovation. Within this realm, the project focuses on two pivotal tasks: image-based recognition of cats or dogs, and recognition of individuals with or without face masks. Leveraging deep learning techniques, particularly Convolutional Neural Networks (CNNs), this project aims to develop robust models capable of discerning between different categories in images with high accuracy.

AIM OF PRESENT STUDY

- Provide an in-depth understanding of the underlying technology behind image processing, such as computer vision and machine learning.
- Investigate wide range of applications.
- Address the challenges such as improving accuracy etc.
- Identify the future scope of image recognition and detection.

OBJECTIVES

- Improve safety by detecting various objects which can be dangerous.
- Ensure accuracy and reliability on recognizing objects or face.
- The users can use it for wide application by changing the dataset.
- **Cats vs. Dogs Classification**: The primary objective of this study is to develop a CNN model capable of accurately classifying images as either cats or dogs. This involves collecting a dataset containing labeled images of cats and dogs, preprocessing the data to enhance model performance, designing an optimal CNN architecture, and training the model to achieve high accuracy in differentiating between the two classes.
- **Mask Detection**: The secondary objective is to create a CNN model for detecting whether individuals in images are wearing masks or not. This task requires assembling a dataset comprising images of individuals with and without masks, preprocessing the data to extract relevant features, designing an appropriate CNN

architecture for mask detection, and training the model to accurately identify mask-wearing behavior

LITERATURE REVIEW

- **Deep Residual Learning for Image Recognition:** Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun introduced the ResNet architecture, addressing the vanishing gradient problem in deep neural networks with skip connections, achieving state-of-the-art performance in various image recognition tasks.
- **ImageNet Classification with Deep Convolutional Neural Networks:** Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton proposed AlexNet, a deep convolutional neural network architecture that achieved significant performance improvements in image classification tasks, marking a breakthrough in deep learning for computer vision.
- **Support-Vector Networks:** Corinna Cortes and Vladimir Vapnik's work on Support Vector Machines (SVM) provided a powerful machine learning algorithm for classification and regression tasks, widely adopted in image classification due to its ability to handle high-dimensional feature spaces and nonlinear decision boundaries.
- **Scale-Invariant Feature Transform (SIFT):** David G. Lowe proposed SIFT, which introduced a robust method for feature extraction invariant to scale and rotation, enabling tasks like object recognition and image stitching.

PROBLEM STATEMENT

The problem addressed in this study revolves around the development of robust convolutional neural network (CNN) models for two distinct yet interconnected tasks: image classification of cats and dogs, and detection of mask-wearing behavior in individuals.

Firstly, the challenge lies in accurately classifying images containing cats and dogs, which requires the CNN model to learn discriminative features to distinguish between these two classes despite their inherent visual similarities. The objective is to achieve high classification accuracy on unseen images, enabling applications such as pet monitoring and wildlife conservation.

Secondly, the study aims to address the pressing need for automated mask detection systems, particularly in the context of the ongoing COVID-19 pandemic. Detecting whether individuals in images are wearing masks or not is crucial for public health surveillance and adherence to safety protocols. The CNN model must effectively identify facial regions and discern the presence or absence of masks, even amidst varying facial expressions and environmental conditions.

By tackling these challenges, the study aims to contribute to the advancement of computer vision technologies, with practical implications in diverse domains including healthcare, public safety, and beyond.

DATASET DESCRIPTION

The dataset is taken from Kaggle.

Image Classification (Cats vs. Dogs)

- **Training Dataset:** Consists of 4000 images of cats and 4000 images of dogs for training the image classification model.
- **Test Dataset:** Comprises 1000 images of cats and 1000 images of dogs for evaluating the performance of the trained model.

Image Classification (Mask vs. No Mask)

- **Training Dataset:** Includes 5000 images with individuals wearing masks and 5000 images without masks for training the mask detection model.
- **Test Dataset:** Contains 483 images of individuals wearing masks and 509 images of individuals without masks for evaluating the performance of the trained model.

Real-Time Mask Detection

- **Dataset:** Consists of 5000 images, evenly distributed between individuals wearing masks and individuals without masks.
- **Testing Set:** Comprises 20% of the entire dataset, with 1000 images randomly selected from the combined mask and no mask images for evaluating the real-time mask detection model.
- **Training Set:** Includes 4000 images of individuals wearing masks and 4000 images of individuals without masks for training the real-time mask detection model.

METHODOLOGY

Collection of Data:

Dogs and cat:

- Data is collected from the directory where all the images are stored.
- ImageDataGenerator class allows you to randomly rotate images through any degree between 0 and 360 by providing an integer value in the rotation_range argument.
- The flow_from_directory() method takes a path of a directory and generates batches of augmented data. The directory structure is very important when you are using flow_from_directory() method. The flow_from_directory() assumes: The root directory contains at least two folders, one for train and one for the test.

Face mask detection:

- We iterated over each category in the dataset using the `**os**` module.
- For each category, we traversed through the images within that category's directory.
- Each image was loaded using `**load_img**` from the `Keras` preprocessing module, resized to a target size of 224x224 pixels, and converted to a NumPy array using `**img_to_array**`.
- We applied preprocessing using `**preprocess_input**` to prepare the image for model input.
- Finally, the preprocessed image data and its corresponding label (category) were appended to the `**data**` and `**labels**` lists, respectively.
- This methodology ensured that our dataset was properly loaded, preprocessed, and organized for subsequent analysis and model training.

Training of the model using Collected Data:

Dogs and cats:

- **filters**: We specified the number of filters in the convolutional layer as 32. Filters are essentially the number of output channels produced by the convolution operation. Increasing the number of filters allows the network to learn more complex patterns and features from the input data.

- `kernel_size`: The kernel size determines the spatial dimensions of the convolutional window. Here, we set the kernel size to 3x3, indicating that each filter will convolve over a 3x3 region of the input image. Adjusting the kernel size can affect the size of the receptive field and the level of detail captured by the filters.
- `activation`: We applied the ReLU (Rectified Linear Unit) activation function to introduce non-linearity into the network. ReLU activation helps the model learn complex patterns by introducing non-linear transformations to the input data.
- `input_shape`: We provided the input shape of the images expected by the network as `[64, 64, 3]`. This indicates that the input images are 64 pixels in height and width, with 3 color channels (RGB). Specifying the input shape helps TensorFlow infer the shape of subsequent layers in the network.
- `pool_size`: We specified the size of the pooling window as 2x2. During the pooling operation, the MaxPooling layer selects the maximum value from each non-overlapping 2x2 region of the input feature map. This downsampling process helps reduce the spatial dimensions of the feature maps, effectively summarizing the information learned by the preceding convolutional layers.
- `strides`: The strides parameter determines the step size or the amount by which the pooling window moves across the input feature map. Here, we set the strides to 2 in both dimensions, indicating that the pooling window moves by 2 pixels horizontally and vertically after each pooling operation. Increasing the stride value results in more aggressive downsampling and further reduction in the spatial dimensions of the feature maps
- **Flatten Layer**: The Flatten layer transforms the multi-dimensional feature maps produced by the convolutional layers into a one-dimensional vector. This process is essential for connecting the convolutional feature extraction layers to the fully connected classification layers. By flattening the feature maps, we ensure compatibility with the dense layers that follow, allowing for further processing and classification of the extracted features.
- **Dense Layers**: The Dense layers consist of densely connected neurons, allowing each neuron to receive input from every neuron in the previous layer. The first Dense layer consists of 128 neurons with ReLU activation, which introduces non-linearity to the network and helps capture complex patterns in the input data. The final Dense layer consists of a single neuron with a sigmoid activation function, which outputs a probability score indicating the likelihood of the input belonging to a particular class (binary classification).
- **Model Compilation**: We compiled the CNN model using the Adam optimizer, binary cross-entropy loss function (suitable for binary classification), and accuracy as the

evaluation metric. Compiling the model prepares it for training by configuring the optimization algorithm and loss calculation method.

- **Model Training**: We trained the compiled CNN model using the **`fit`** method, providing the training and validation datasets (**`training_set`** and **`testing_set`**, respectively) along with the specified number of epochs (30 in this case). During training, the model adjusts its weights based on the optimization algorithm and updates its performance metrics based on the provided datasets.
- **Training Accuracy**: The training accuracy of 93.41% indicates the proportion of correctly classified examples in the training dataset. A high training accuracy suggests that the model has learned to classify the training examples accurately.
- **Training Loss**: The training loss of 0.1769 represents the average loss value calculated during the training process. The loss is a measure of the model's performance, with lower values indicating better performance. In this case, the low training loss suggests that the model's predictions are close to the actual labels for the training examples.
- **Validation Accuracy**: The validation accuracy of 78.25% indicates the proportion of correctly classified examples in the validation dataset. The validation accuracy provides an estimate of how well the model generalizes to unseen data. In this case, the validation accuracy is lower than the training accuracy, suggesting that the model may be overfitting to some extent.
- **Validation Loss**: The validation loss of 0.6419 represents the average loss value calculated on the validation dataset. Similar to training loss, lower validation loss values indicate better performance. The higher validation loss compared to the training loss suggests that the model may not perform as well on unseen data, possibly due to overfitting.

Face mask detection:

- **Label Binarization**:
  - You're using the **`LabelBinarizer`** from scikit-learn to convert categorical labels into binary vectors.
  - This step is essential for training models that require binary labels, such as neural networks for binary classification.
- **One-Hot Encoding**:
  - After binarizing the labels, you're further transforming them into one-hot encoded vectors using the **`to_categorical`** function from Keras.
  - One-hot encoding is necessary when dealing with categorical variables in machine learning models, particularly in neural networks.

- **Data Conversion**:
    - You're converting your data and labels into NumPy arrays with the appropriate data types (**`float32`** for data and the default data type for labels).
    - NumPy arrays are commonly used for efficient numerical computations in Python.

- **Train-Test Split**:
    - You're splitting your data and labels into training and testing sets using the **`train_test_split`** function from scikit-learn.
    - This step is crucial for evaluating the performance of your machine learning model on unseen data.
    - the MobileNetV2 model with pre-trained ImageNet weights, excluding the top layers, and defining the input shape for the model. This base model will serve as the feature extractor in your transfer learning setup, with additional layers added on top for classification or regression tasks specific to your application.
    - Max pooling, Flatten layer and Dense layers will be implemented on augmented data.

- **Compiling the Model**:
    - The **`compile`** method is used to compile the model.
    - You're specifying the loss function as **`"binary_crossentropy"`** since it's a binary classification problem.
    - The optimizer is set to Adam with a specified learning rate (**`INIT_LR`**).
    - Metrics for evaluation during training are specified as accuracy.

Training the Model:

- The **`fit`** method is used to train the model.
- Training data (**`trainX`**, **`trainY`**) is passed using a data generator (**`aug.flow`**), which performs data augmentation on-the-fly.
- **`steps_per_epoch`** specifies the number of batches to yield from the generator in each epoch.
- Validation data (**`testX`**, **`testY`**) is provided directly.
- **`validation_steps`** specify the number of batches to yield from the validation data generator in each epoch.
- The number of epochs to train (**`EPOCHS`**) is specified.

Explanation:

- This code segment compiles the model with the specified optimizer, loss function, and evaluation metrics.

- Then, it trains the model on the training data while validating on the test data.
- During training, the data is augmented on-the-fly using a data generator (**`aug.flow`**), which helps improve the model's generalization ability.

Training Accuracy: The training accuracy of 99.25% indicates that 99.25% of the training examples were classified correctly by the model during training.

- **Training Loss**: The training loss of 0.0220 represents the average loss calculated on the training data during each epoch of training. Lower values indicate better performance, suggesting that the model's predictions are close to the actual labels for the training examples.
- **Validation Accuracy**: The validation accuracy of 99.20% indicates that 99.20% of the validation examples were classified correctly by the model during validation.
- **Validation Loss**: The validation loss of 0.0246 represents the average loss calculated on the validation data during each epoch of training. Similar to training loss, lower values indicate better performance on unseen data.
- You're evaluating the trained model on the test data using the **`predict`** method. The predictions (**`predIdxs`**) are obtained for the entire test set (**`testX`**) in batches (**`batch_size=BS`**).
- For each image in the testing set, you find the index of the class with the highest predicted probability. This is achieved using **`np.argmax`** along the appropriate axis (**`axis=1`**).
- You're generating a classification report using the **`classification_report`** function from scikit-learn.
- This report provides metrics such as precision, recall, and F1-score for each class, as well as overall accuracy.
- Finally, you're saving the trained model to disk using the **`save`** method.
- This serialized model can be loaded later for inference or further training.

Implementing the model:

Dogs and cats:

- **Load and Preprocess the Image**:
  - You're loading the image using **`image.load_img`** from Keras preprocessing module, specifying the target size as (64, 64) pixels.
  - Then, you're converting the image to a NumPy array using **`image.img_to_array`**.

    - Next, you're expanding the dimensions of the array using **`np.expand_dims`** to match the input shape expected by the model.
- **Make Prediction**:
    - You're using the trained model (**`cnn`**) to predict the class label for the test image.
    - The prediction is made using the **`predict`** method on the test image array.
- **Class Indices**:
    - You're accessing the class indices dictionary (**`training_set.class_indices`**) to interpret the model's prediction.
- Based on the prediction result, you're determining whether the image is classified as a cat or a dog.
- If the predicted result is 1, it's classified as a dog; if it's 0, it's classified as a cat.
- Overall, this code segment loads an image, preprocesses it, and uses the trained model to predict whether the image contains a cat or a dog. The prediction result is then interpreted based on the class indices defined during training.

Face mask detection:

- **Input Parameters**:
    - **`frame`**: The input frame (image) containing faces.
    - **`faceNet`**: A pre-trained deep learning model used for face detection.
    - **`maskNet`**: A pre-trained deep learning model used for face mask classification.

Face Detection:

- The function first resizes the input frame to (224, 224) pixels and preprocesses it to create a blob.
- The blob is passed through the face detection model (**`faceNet`**) to detect faces in the frame.
- Detected faces are stored along with their bounding box coordinates (**`locs`**).

Face Mask Classification:

- For each detected face, the function extracts the face region, converts it to RGB format, and resizes it to (224, 224) pixels.
- The face region is preprocessed and added to the **`faces`** list.
- If one or more faces are detected, the list of faces is converted to a NumPy array and passed to the face mask classification model (**`maskNet`**) for prediction.
- Predictions (**`preds`**) containing the probabilities of wearing a mask are obtained for each face.

- The function returns two lists: **`locs`** containing the bounding box coordinates of the detected faces and **`preds`** containing the mask-wearing probability predictions for each detected face.

Defining Model Paths:

- **`prototxtPath`**: Path to the prototxt file containing the network architecture.
- **`weightsPath`**: Path to the Caffe model file containing the pre-trained weights.
- loading a pre-trained model named "mask_detector.keras" using Keras' **`load_model`** function.

Loading the Face Detection Model:

- The **`cv2.dnn.readNetFromCaffe()`** function is used to load the face detection model from the specified prototxt file and weights file

.

Starting Video Stream:

- It initializes the video stream using **`cv2.VideoCapture(0, cv2.CAP_DSHOW)`**. The **`0`** argument indicates the default webcam device, and **`cv2.CAP_DSHOW`** specifies the DirectShow API for capturing frames.

Frame Processing Loop:

- It enters a **`while`** loop to continuously read frames from the video stream.
- Each frame is resized to a width of 400 pixels using **`imutils.resize`**.

Face Mask Detection:

- The **`detect_and_predict_mask`** function is called to detect faces in the frame and predict whether they are wearing masks
- The bounding box coordinates of detected faces (**`locs`**) and mask-wearing predictions (**`preds`**) are obtained.

Drawing Predictions:

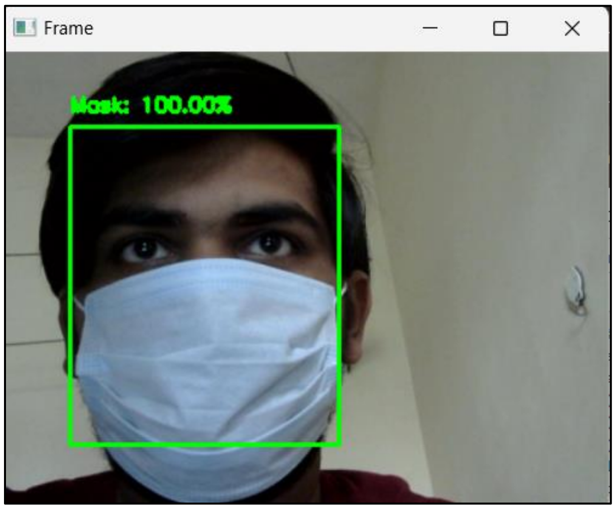
- For each detected face, a bounding box is drawn around it, and a label indicating whether a mask is present or not is overlaid.
- The color of the bounding box and label text is determined based on the prediction result.

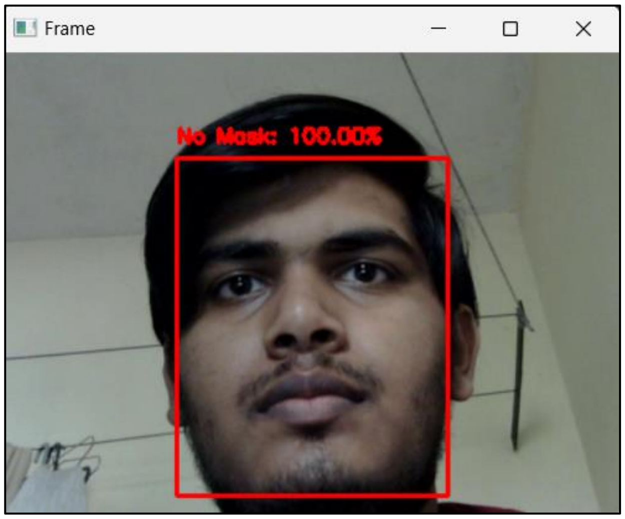
Displaying Frames:

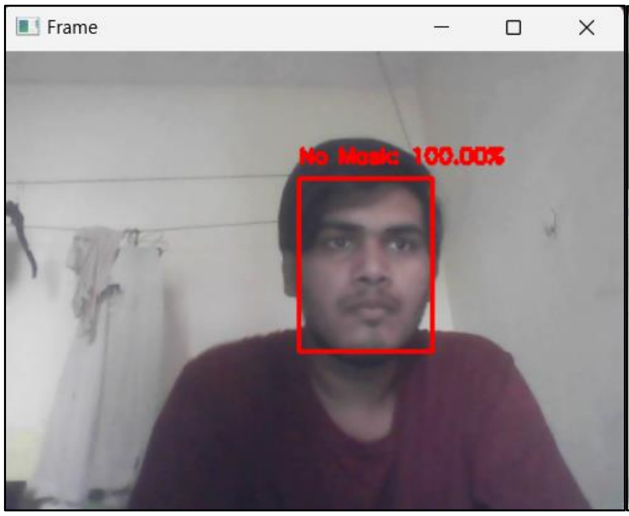
- The processed frame with bounding boxes and labels is displayed using **`cv2.imshow`**.

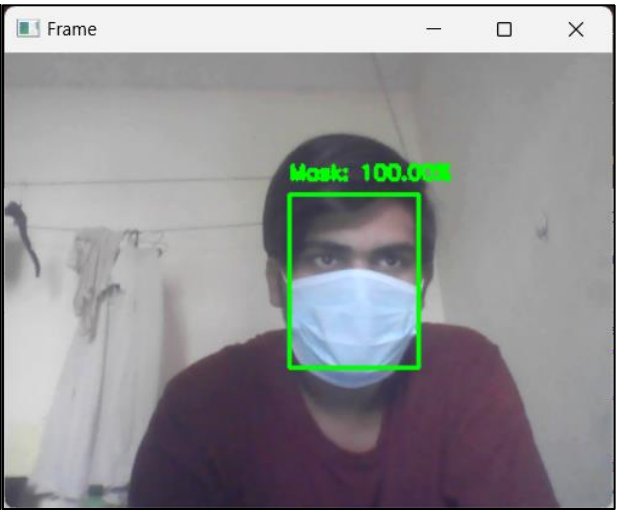
Exiting the Program:

- Pressing the "q" key terminates the video stream and closes the OpenCV windows.
- The **`frame.release()`** method releases the video stream


Frame
Mask: 100.00%


Frame
No Mask: 100.00%


Frame
No Mask: 100.00%


Frame
Mask: 100.0

FEATURES

- Image preprocessing
- Feature extraction
- Mask region detection
- CNN model development
- Cats vs. dogs' classification
- Mask detection task
- Integration of OpenCV
- Image preprocessing
- Feature extraction

APPLICATIONS

- Wildlife conservation and monitoring: By accurately recognizing and classifying images of cats and dogs, the project can assist in wildlife conservation efforts by tracking and monitoring populations of these animals in their natural habitats.
- Public health and safety: The mask detection aspect of the project contributes to public health initiatives by enabling the automated detection of individuals wearing face masks, aiding in enforcing safety protocols during pandemics and in crowded spaces.
- Security surveillance: The project's capabilities in image-based recognition, combined with mask detection, have applications in security surveillance systems, enhancing the ability to identify individuals and ensuring compliance with security measures.

CONCLUSIONS and FUTURE SCOPE

∉ In conclusion, our project showcases the effectiveness of utilizing convolutional neural networks and deep learning techniques for image classification and mask detection tasks. With high accuracy rates of 93.4% for cats and dogs classification and 98.67% for image-based mask detection, alongside a remarkable accuracy of 99.25% in real-time mask detection, our methodology demonstrates robust performance. These results underscore the practical utility of modern computational techniques in solving real-world problems. As we continue to refine and optimize our models, we contribute to the advancement of computer vision applications in various domains, from wildlife conservation to public health and security. Expansion to multi-species classification: Extend the project to recognize a broader range of animal species, enhancing its utility in biodiversity monitoring and conservation efforts.

- Integration of real-world data augmentation: Incorporate real-world data augmentation techniques to improve model generalization and adaptability to diverse environmental conditions and scenarios.

- Integration into attendance systems: Extend the project's applicability by integrating the developed models into attendance systems, enabling automated tracking and verification of individuals in various settings such as schools, workplaces, and events.

- Integration of advanced deep learning techniques: Investigate state-of-the-art methodologies, such as attention mechanisms or reinforcement learning, to further improve model performance and robustness in image recognition and mask detection tasks.

REFERENCES

- **OpenCV** : https://www.geeksforgeeks.org/opencv-python-tutorial/
- **TensorFlow :** https://www.tensorflow.org/learn
- **Scikit-learn (Sklearn):** https://scikit-learn.org/stable/index.html
- **Keras:** https://keras.io/
- **Kaggle:** https://www.kaggle.com/

∉ **NumPy:** https://numpy.org/
∉ Imutils : https://github.com/jrosebr1/imutils
∉ Pandas : https://pandas.pydata.org/