2022/2023

FACE MASK DETECTION

MINI PROJECT AI

Prepared by :
KAMAL EL MAGOURI
SAMIHA TALJAOUI

Contents

Contents 2
List of Figures 4
Chapter 1: Face Detection 5
1.1. Definition 5
1.2. Face Detection Methods 6
1.2.1. Knowledge-based 7
1.2.2. Template matching 8
1.2.3. Feature-based 8
1.2.4. Appearance-based 8
1.3. Viola-Jones Algorithm 9
1.3.1. Haar-like Features 9
1.3.2. Integral Images 10
1.3.3. The AdaBoost Algorithm 11
1.3.4. The Cascade Classifier 11
1.4. Existing Software-Based Solutions 12
1.4.1. Proprietary face detection software 12
1.4.1.2. Face^{++} Detect 13
1.4.1.3. Microsoft Azure Face Api 13
1.4.1.4. Kairos 14
1.4.1.5. Paravision 14
1.4.2. Open-source face detection solutions 14
Chapter 2: ML Models for Face Detection 15
2.1. Artificial Intelligence 15
2.2. Machine Learning 15
2.2.1. Supervised Learning 16
2.2.2. Unsupervised Learning 16
2.2.3. Reinforcement Learning 17
2.2.4. Artificial Neural Network 17
2.3.Deep Learning 18
2.4.Conclusion 19
Chapter 3: COVID-19 Face Mask Detector 20
3.1. Collecting Dataset 21
3.2. Classification Models 22
3.3. Conclusion 23

3.4. Detect and predict mask 24
3.1. Test Results of our Face Mask Training model 27
General Conclusion 29

List of Figures

Figure 1.1– Face detection.....5
Figure 1.2– Types of face detection methods. 6
Figure 1.3– Knowledge-based face detection method.....7
Figure 1.4– Template matching face detection method.....8
Figure 1.5– The different stages of Viola-Jones algorithm. 9
Figure 1.6– Haar-like features for face detection 10
Figure 1.7– Conversion of original image to integral image.....10
Figure 1.8– The AdaBoost algorithm - extracting the best features from n features.....11
Figure 1.9– The Cascade Classifier 12
Figure 1.10– Amazon Rekognition detects faces and analyses it.....12
Figure 1.11– Azure Face API detects 27 landmarks for each face.....13
Figure 2.1– A schematic overview of supervised learning.....17
Figure 2.2– A schematic overview of unsupervised learning.....17
Figure 2.3– The RL cycle 18
Figure 2.4– A typical architecture of an ANN.....18
Figure 2.5– Illustration of difference between ML and DL.....18
Figure 3.1– Samples from the dataset of faces with masks 21
Figure 3.2– Samples from the dataset of faces without masks 21
Figure 3.3– Live Test of Our Face Mask Detector (CNN Model).....27
Figure 3.4– Other Live Test Results covering the three cases.....28

Chapter 1: Face Detection

The ministry has urged all residents to wear a face mask in public areas since the start of the COVID-19 pandemic to protect themselves and sever the outbreak chain. Unfortunately, in large cities, it has been difficult for shop owners and office guards to keep track of whether or not every person entering the building is wearing a mask, The model can detect human faces and determine whether they are wearing a face mask or not.

In this chapter, we will look at how computers view things and the many computerized face detection algorithms that can be used.

1.1. Definition

Face detection, often known as *facial detection*, is a technology that uses artificial intelligence (AI) to determine whether or not there are any human face in digital images or videos .. As shown in Figure 1.1, the detection result provides face location parameters such as a bounding box covering the middle region of the face, or landmarks such as eyes, nose and mouth corners, eyebrows, and so on.

Figure 1.1– Face detection

Detecting the locations in photos where faces are present is obviously a first step in any face processing system. In the next sub-section, we will discuss the different methods used to detect human faces.

1.2. Face Detection Methods

Face detection technology can be applied to various fields, including security, biometrics, law enforcement, entertainment, and personal safety, to offer real-time surveillance and tracking of people. First, the model examines a photo or a video image, attempting to differentiate faces from other items in the background.

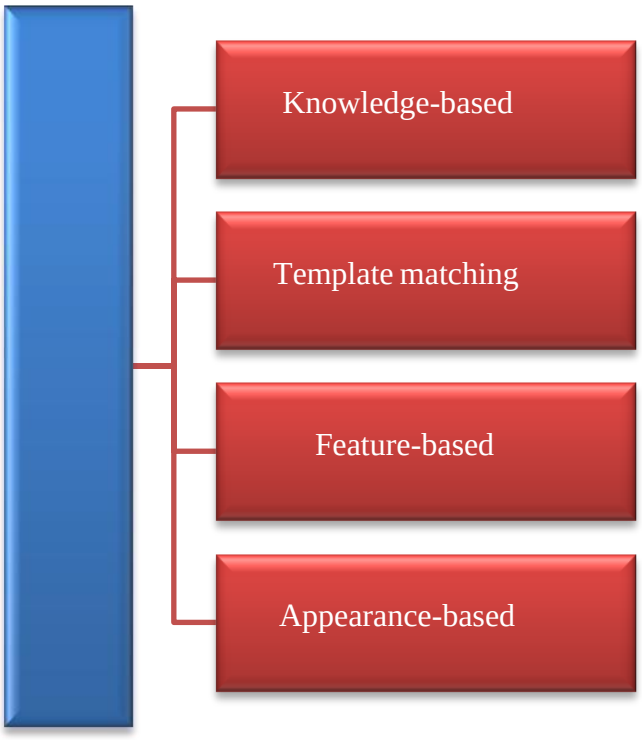


Figure 1.2– Types of face detection methods

1.2.1. Knowledge-based

This method is based on a set of rules that represent human knowledge of what a normal face looks like (we know that a face must have a nose, eyes, and mouth within certain distances and positions with each other.).

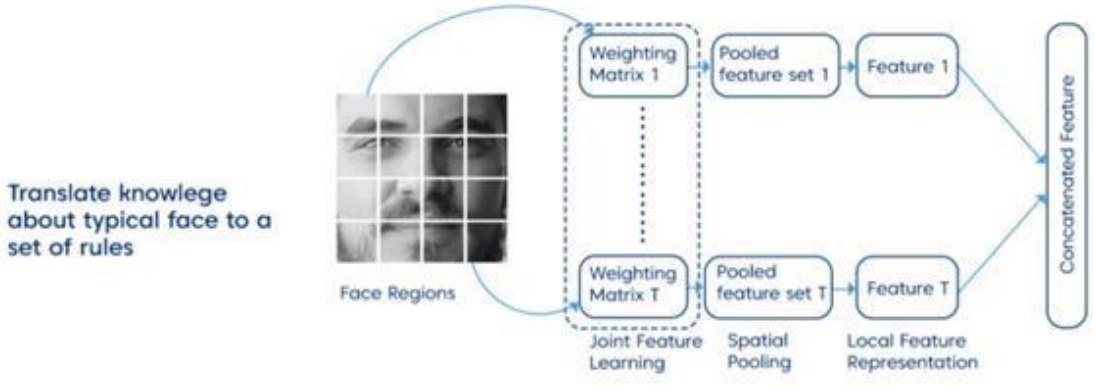


Figure 1.3– Knowledge-based face detection method

It is primarily intended for face localisation, but one obstacle is constructing an acceptable set of rules, as there may be many false positives if the rules were too generic or too detailed. Moreover, because of its reliance on lighting conditions and inability to locate many faces in multiple photos, employing this method alone is insufficient.

1.2.2. Template matching

In this method, several standard patterns of a face are stored to describe the face as a whole or the facial features separately. The correlations between an input image and the face templates are then calculated in order to locate or detect the faces. For example, a human face can be divided into eyes, face contour, nose, and mouth.

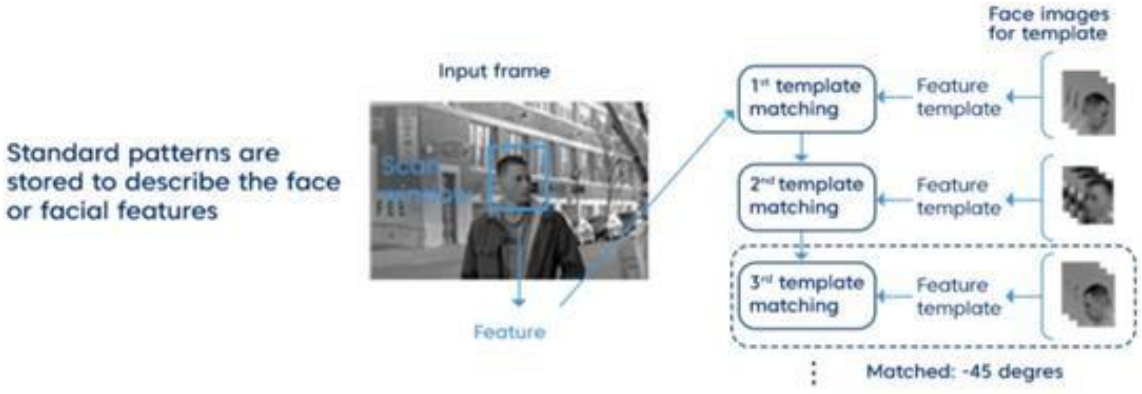


Figure 1.4– Template matching face detection method

This method is simple to implement, yet it is insufficient for face detection. However, deformable templates have been presented as a solution to these issues.

1.2.3. Feature-based

Objects are typically identified by their distinguishing characteristics. A human face has several distinguishing features that allow it to be distinguished from many different objects. A feature-based method detects faces by extracting structural features such as eyes, nose, and mouth and then using them to detect a face. It is often trained as a classifier that distinguishes between facial and non-facial regions. Features can be extracted either using *Haar feature selection* or h*istogram of oriented gradients*, among others. One of the popular algorithms that use a feature-based approach is the *Viola-Jones* algorithm

1.2.4. Appearance-based

The appearance- or image-based method depends on a set of delegate training face images to find out face resnet

The learned characteristics are in the form of distribution models or discriminant functions that is consequently used for face detection.

In general, the appearance-based method relies on techniques from statistical analysis and machine learning (ML) to find the relevant characteristics of face and non-face images. Among these methods, we cite: *distribution-based* algorithms like principal component analysis and Fisher‘s Discriminant. Besides, other ML models such as *artificial neural networks*, *support vector machines* (SVM), and *naive bayes classifiers*;

1.3. Viola-Jones Algorithm

Viola-Jones algorithm is named after two computer vision researchers who proposed the method in 2001, *Paul Viola* and *Michael Jones* in their article . The algorithm had proven to be efficient in real-time face detection, despite the fact being slow in training. The algorithm combines the four following stages to create a system for object detection that is fast and accurate.

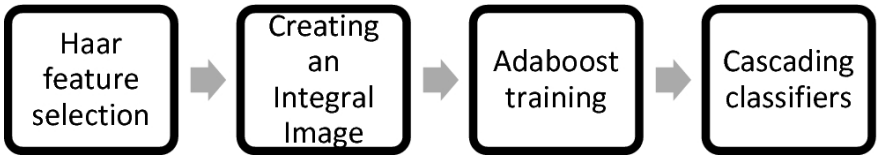


Figure 1.5– The different stages of Viola-Jones algorithm.

1.3.1. Haar-like Features

Rather than using intensities directly, features are frequently extracted from input images in Computer Vision. One example is *Haar-like features*, which consist of dark and light regions. There are numerous sorts of Haar-like features that allow us to extract meaningful information from an image, such as edges, straight lines, and diagonal lines that can be used to identify an object.

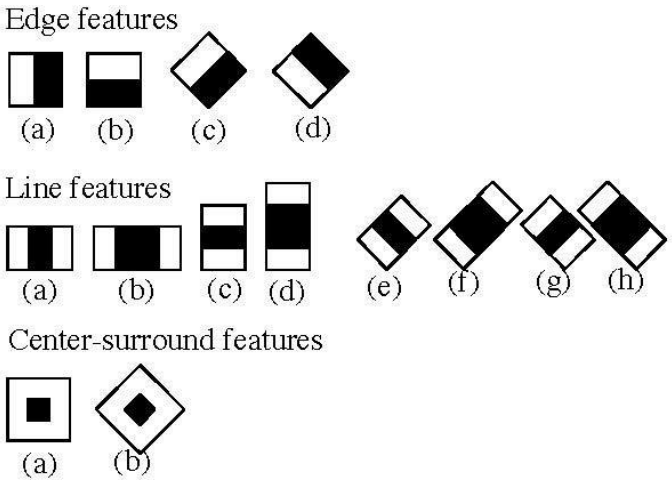


Figure 1.6– Haar-like features for face detection

Because an image can have numerous faces of varying sizes, the algorithm must verify many alternative places and scales. This would be too computationally expensive to execute in real time, so the concept of integral images was invented to tackle the problem.

1.3.2. Integral Images

An integral image is an intermediate representation of an image where the value for location (*x*, y) on the integral image equals the sum of the pixels above and to the left (inclusive) of the (*x*, *y*) location on the original image

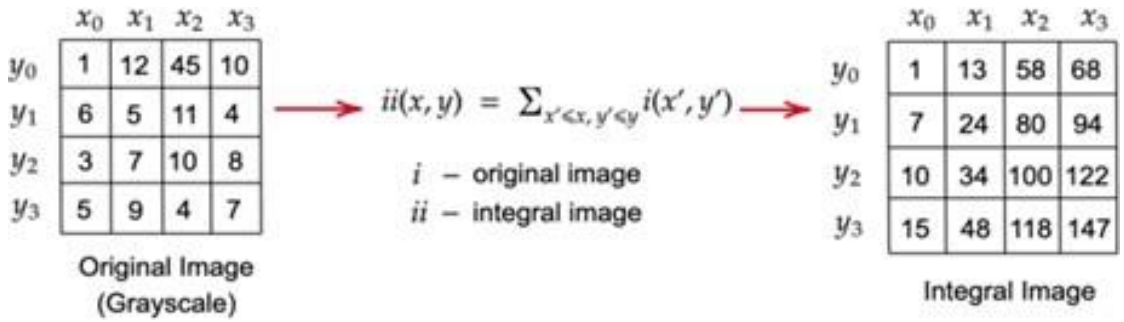


Figure 1.7– Conversion of original image to integral image

Whatever the size of the image, this intermediate representation reduces the calculations of a given pixel to an operation involving only four pixels.

1.3.3. The AdaBoost Algorithm

The AdaBoost (*Adaptive Boosting*) algorithm is a ML algorithm for selecting the best subset of features among all available features. The output of the algorithm is a classifier called a *"strong classifier"*.

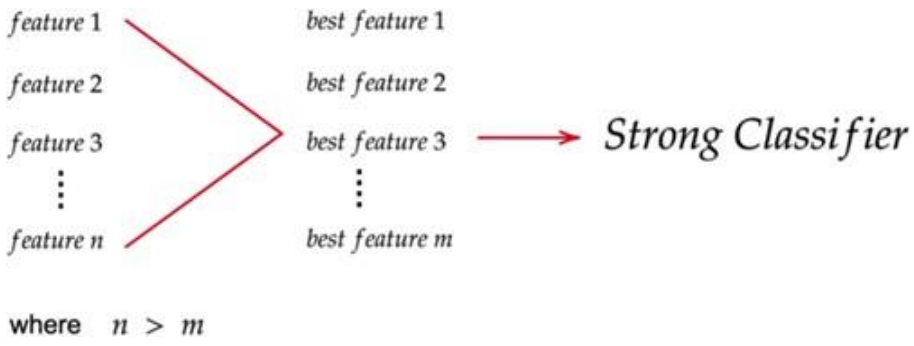


Figure 1.8– The AdaBoost algorithm - extracting the best features from n features

The classifier, as illustrated above, is composed of linear combinations of best features. The algorithm iterates for *m* iterations in order to find these features (*m* is the number of features to find). In each iteration, the algorithm calculates the error rate for all features and then chooses the feature with the lowest error rate for that iteration.

1.3.4. The Cascade Classifier

A *cascade classifier* is a multi-stage classifier that detects objects rapidly and reliably. Each stage consists of a strong classifier generated using the AdaBoost Algorithm. The number of best features in a strong classifier grows from stage to stage. If a classifier for a certain step returns a negative result, the input is immediately rejected. If the result is positive, the input is passed on to the next stage.

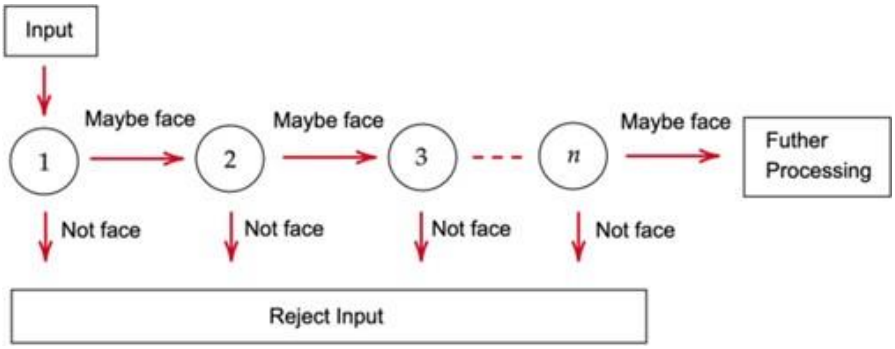


Figure 1.9– The Cascade Classifier

1.4. Existing Software-Based Solutions

There are dozens of face detection technologies available, with functionality ranging from simple face detection through emotion detection and lip reading. Some were developed by IT businesses, while others are open-source.

1.4.1. Proprietary face detection software

1.4.1.1. Amazon Rekognition

Amazon Rekognition is a deep learning-based technology that is fully integrated into the *Amazon Web Services* environment. It can detect not just human faces in images and videos, but also objects, people, text, settings, and activities, as well as detect any inappropriate content. Meanwhile, it offers the possibility of custom labels, which may be used to identify items and scenes in images that are relevant to the user's business needs.

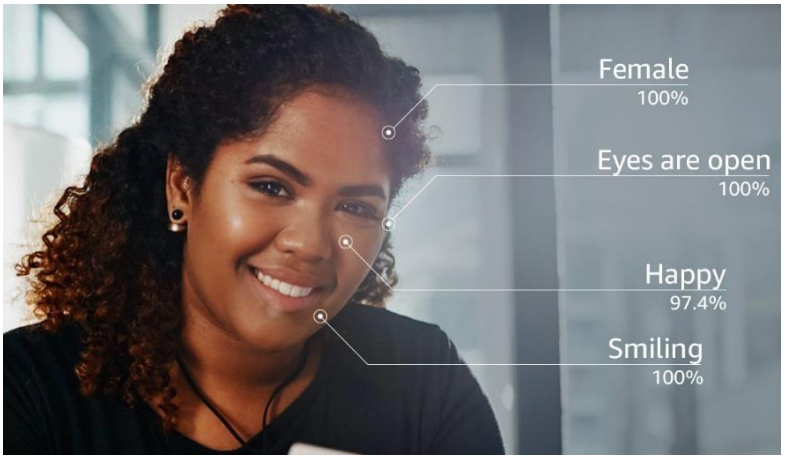


Figure 1.10– Amazon Rekognition detects faces and analyses it

1.4.1.2. Face^{++} Detect

Face^{++} *detect* enables the detection of human faces in images and delivers high-precision face bounding boxes. It is a cloud service with offline SDKs for iOS and Android. Its primary market is China, and it is well-known for its incorporation in Lenovo products. This API can be used for free, with the possibility for upgrade to paid service by Pay-As-You-Go service or SDK licensing options.

1.4.1.3. Microsoft Azure Face API

Face API is one of Microsoft Azure Cognitive services that allows face detection that perceives facial features and attributes in an image, as well as age estimation, gender and emotion recognition, and landmark detection. It enables making 30000 requests per month, at a rate of 20 requests per minute for free. The price for paid requests is determined by the number of recognitions received per month, and it begins at $1 for 1000 recognitions.

Figure 1.11– Azure Face API detects 27 landmarks for each face.

1.4.1.4. Kairos

Kairos provides a number of image recognition solutions. Their API endpoints include photo and video identification of gender, age, facial recognition, and emotional depth. They provide SDKs for PHP, JS, .Net, and Python and provide a 14-day free trial with a maximum limit of 10000 requests.

1.4.1.5. Paravision

Paravision is a face recognition company for enterprises providing self-hosted solutions. Face and activity recognition and COVID-19 solutions are among their services. The company has SDKs for C++ and Python.

1.4.2. Open-source face detection solutions

1.4.2.1. OpenCV

OpenCV (*Open-source Computer Vision*) is a library with over 3,000 optimized algorithms for real-time computer vision. It was originally developed by Intel for use under the open-source *Apache 2* license. It offers many options for developers, including *Eigenfacerecognizer* and *LBPHFacerecognizer* face recognition modules.

Chapter 2: ML Models for Face Detection

In this chapter, we will give more details about computer-based technologies like machine learning, deep learning, and computer vision that are broadly used for face detection.

2.1. Artificial Intelligence

The term *artificial intelligence* (AI) commonly refers to the ability of a computer or machine to mimic the capabilities of the human mind, such as reasoning, learning, decision-making, and problem-solving

Another definition provided by John McCarthy is as follows:

> *AI is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.*

2.2. Machine Learning

Machine learning (ML) involves the development of algorithms, by which computers may learn from data and perform predictions without previous specific programming. To do so, the algorithms analyse the data and its properties using probabilistic and statistical toolsto determine the actions as well as computers modify or adapt these actions to improve their accuracy, which is measured by how well the chosen actions reflect the correct ones .

ML techniques can be broadly classified into three paradigms: *supervised*, *unsupervised*, and *reinforcement learning*.

2.2.1. Supervised Learning

Supervised learning is the search for algorithms that have been trained on explicit data sets that have been labelled by experts in order to produce general hypotheses and then make predictions about future instances. Examples of such algorithms include *K-nearest-neighbours* (KNN) *linear* and *logistic regression*, *random forests*, and *support vector machines* (SVM).

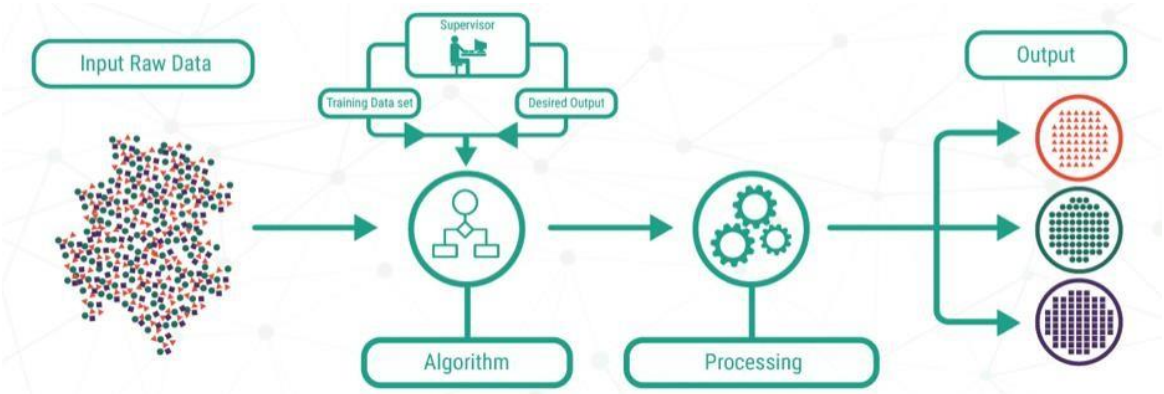


Figure 2.1– A schematic overview of supervised learning.

2.2.2. Unsupervised Learning

In *unsupervised learning*, ML algorithms use a dataset without labels and try to discover hidden patterns or data groupings without the need for human intervention. Such algorithms are utilised for three main tasks: *clustering*, *association rules*, and *dimensionality reduction*

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition..

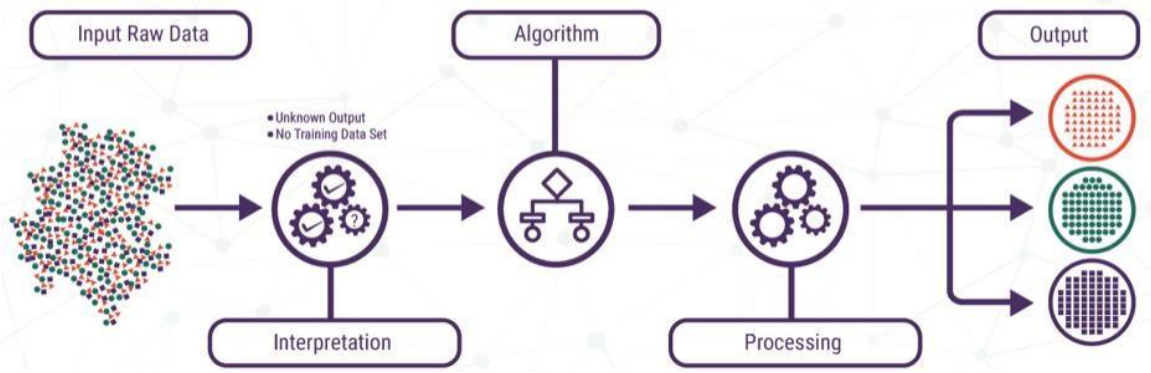


Figure 2.2– A schematic overview of unsupervised learning .

2.2.3. Reinforcement Learning

Reinforcement learning (RL) is an approach through which intelligent programs, known as *agents*, acts in an environment to constantly adapt and predict the features at a future step on the basis of past and present features. On the basis of the prediction, the feedback might be positive, also known as *rewards*, or negative, also called *punishments*. The agent eventually learns a policy for choosing the action to take at each stage in order to maximise the expected return, which is usually the sum of predicted future rewards.

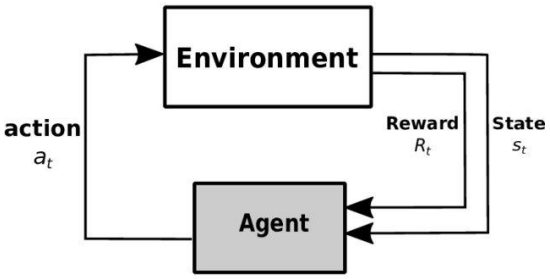


Figure 2.3– The RL cycle

2.2.4. Artificial Neural Network

Artificial neural networks (ANNs) are well-suited to solving classification problems, but they can also be used for regression or clustering. Due to the fact that they are inspired by the sophisticated functionality of human brains where hundreds of billions of interconnected neurons process information in parallel, ANNs are very suited to solve problems that people are good at but computers are not . These problems include pattern recognition and forecasting (which requires the recognition of trends in data). An ANN consists of three layers: an *input* layer of neurons (or nodes, units), one or two (or even three) *hidden* layers of neurons, and a final layer of *output* neurons .

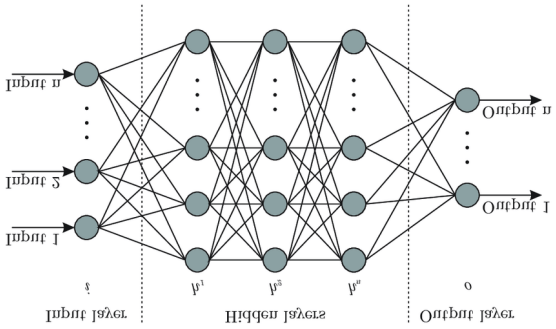


Figure 2.4– A typical architecture of an ANN

2.3.Deep Learning

The depth and width of the network determine its complexity and ‗learning potential' .*Deep neural networks* are made up of multiple layers of interconnected nodes, each building upon the previous layer to improve and optimise the prediction or categorisation . This progression of computations through the network is called *forward propagation*. The input is presented at the *visible* layer, which is so named because it contains the variables that are observable. Then a series of *hidden* layers extracts increasingly abstract features from the image. These layers are referred to as ‗hidden' because their values are not given in the data; instead, the model must determine which concepts are useful for explaining the relationships in the observed data. Consequently, even though the training time is longer, these networks outperform simple ANN. Fortunately, methods such as transfer learning, GPU computing can be used to reduce the training time.

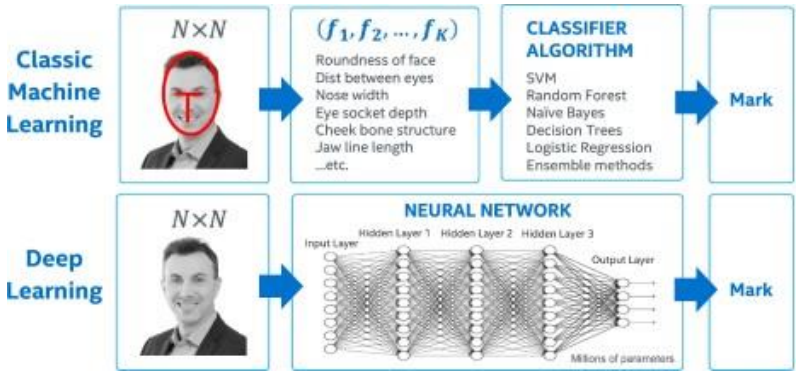


Figure 2.5– Illustration of difference between ML and DL .

DL methods have shown promising results in health care, in which they have been successfully applied to image recognition diagnostic, clinical decision-making, and cancer detection. To address specific problems or datasets, different types of neural deep networks are available .For instance, *convolutional neural networks* are primarily used in image classification applications, while *recurrent neural networks* are used in speech recognition applications.

2.4.Conclusion

Based on the preceding, ML algorithms appeared to be a promising solution that should be used for face detection. As a result, new solutions have been emerged for face detection in general, and COVID-19 pandemic in particular. In the next chapter, we will present the results obtained after applying three ML models for face mask detection.

Chapter 3: COVID-19 Face Mask Detector

Now we will go through the specifics of our suggested face mask detector. This chapter will begin by providing an outline of the research method. The latter consists of three main stages, which are *collecting datasets*, *building classification models*, and eventually *training and testing*.

3.1. Collecting Dataset

Our dataset contains 3833 face images, in which 1915 images of people wearing face masks, 1918 images of people who do not wear face masks,

The dataset was separated into two different folders:

–*WithMask*

Figure 3.1– Samples from the dataset of faces with masks

-*Without Mask*

Figure 3.2– Samples from the dataset of faces without mask

3.1. Classification Models

Using python libraries such as *Tensorflow*, *Keras*, and *Imutils* numerous classification models were created and many parameters were configured in order to discover which parameters are the most appropriate in each classifier. The algorithm was fine-tuned using hyper parameters such as the *number of epochs*, *learning rate*, and *batch size*.

The table below summarise all parameters and presents the layer sequence.

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_9 (InputLayer)         [(None, 224, 224, 3       0           []
                             )]

Conv1 (Conv2D)               (None, 112, 112, 32       864         ['input_9[0][0]']
                             )

bn_Conv1 (BatchNormalization)  (None, 112, 112, 32     128         ['Conv1[0][0]']
                             )

Conv1_relu (ReLU)            (None, 112, 112, 32       0           ['bn_Conv1[0][0]']
                             )

expanded_conv_depthwise (Depth  (None, 112, 112, 32    288         ['Conv1_relu[0][0]']
wiseConv2D)                  )

expanded_conv_depthwise_BN (Ba  (None, 112, 112, 32    128         ['expanded_conv_depthwise[0][0]']
tchNormalization)            )

expanded_conv_depthwise_relu (  (None, 112, 112, 32    0           ['expanded_conv_depthwise_BN[0][0
ReLU)                        )                                     ]']
```

Training the head of the network, the model reaches an accuracy level of 99%. As presented below,

```
[INFO] training head...
Epoch 1/10
95/95 [==============================] - 33s 348ms/step - loss: 0.0391 - accuracy: 0.9885 - val_loss: 0.0395 - val_accuracy: 0.9883
Epoch 2/10
95/95 [==============================] - 33s 348ms/step - loss: 0.0358 - accuracy: 0.9888 - val_loss: 0.0394 - val_accuracy: 0.9896
Epoch 3/10
95/95 [==============================] - 33s 351ms/step - loss: 0.0294 - accuracy: 0.9918 - val_loss: 0.0387 - val_accuracy: 0.9896
Epoch 4/10
95/95 [==============================] - 34s 359ms/step - loss: 0.0306 - accuracy: 0.9921 - val_loss: 0.0335 - val_accuracy: 0.9896
Epoch 5/10
95/95 [==============================] - 35s 372ms/step - loss: 0.0320 - accuracy: 0.9914 - val_loss: 0.0305 - val_accuracy: 0.9922
Epoch 6/10
95/95 [==============================] - 33s 348ms/step - loss: 0.0324 - accuracy: 0.9895 - val_loss: 0.0337 - val_accuracy: 0.9883
Epoch 7/10
95/95 [==============================] - 33s 352ms/step - loss: 0.0288 - accuracy: 0.9918 - val_loss: 0.0319 - val_accuracy: 0.9896
Epoch 8/10
95/95 [==============================] - 33s 350ms/step - loss: 0.0323 - accuracy: 0.9891 - val_loss: 0.0313 - val_accuracy: 0.9922
Epoch 9/10
95/95 [==============================] - 33s 347ms/step - loss: 0.0242 - accuracy: 0.9927 - val_loss: 0.0320 - val_accuracy: 0.9896
Epoch 10/10
95/95 [==============================] - 34s 363ms/step - loss: 0.0274 - accuracy: 0.9924 - val_loss: 0.0328 - val_accuracy: 0.9896
```

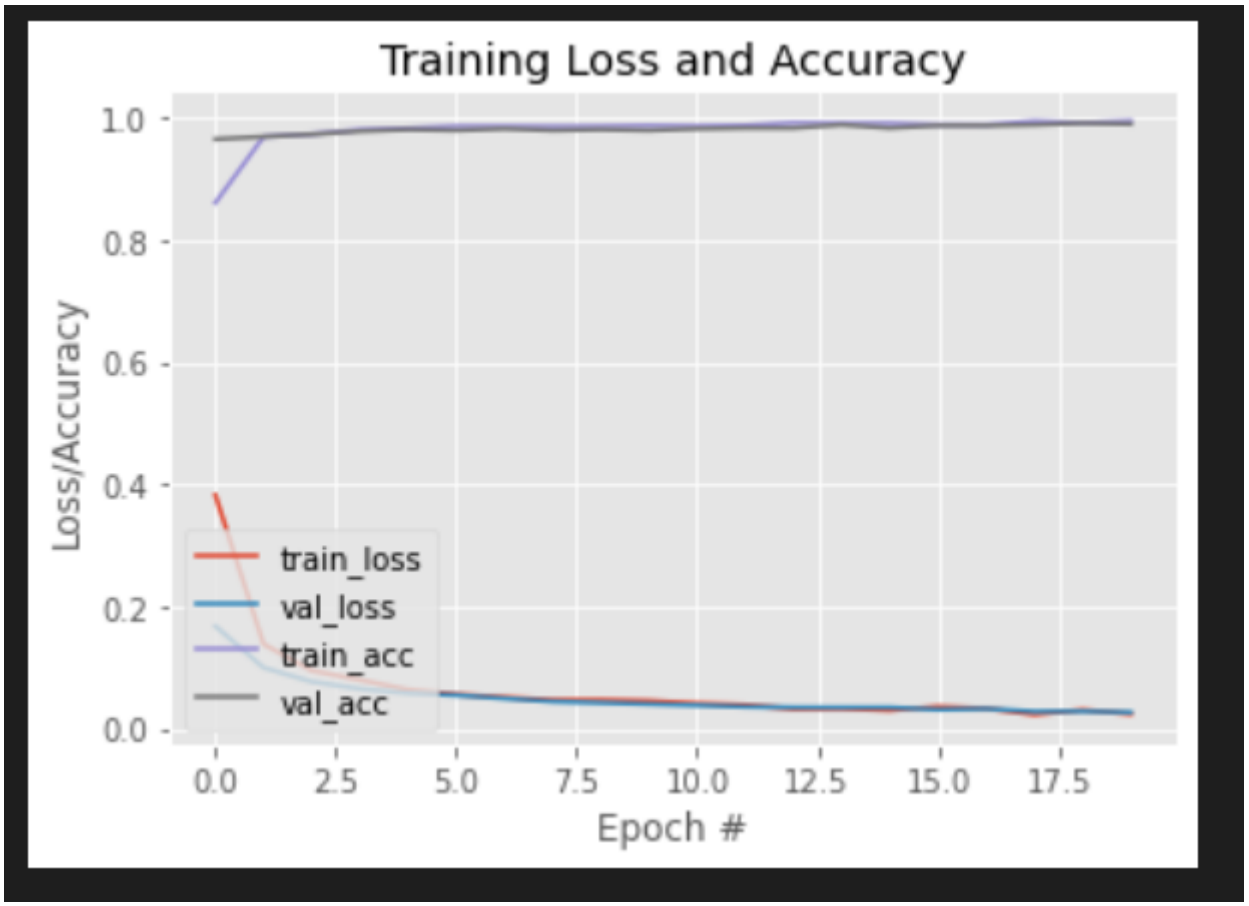
Below we have the classification report:

```
...                  precision    recall  f1-score   support

       with_mask       0.99      0.99      0.99       383
    without_mask       0.99      0.99      0.99       384

        accuracy                           0.99       767
       macro avg       0.99      0.99      0.99       767
    weighted avg       0.99      0.99      0.99       767
```

Plot the training loss and accuracy:



3.2. Conclusion

Finally, with a testing accuracy of 99 percent, we decided that model performed very well

3.3. Detect and predict mask

```
def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob
    # from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
        (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    faceNet.setInput(blob)
    detections = faceNet.forward()
    print(detections.shape)

    # initialize our list of faces, their corresponding locations,
    # and the list of predictions from our face mask network
    faces = []
    locs = []
    preds = []

    # loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the detection
        confidence = detections[0, 0, i, 2]
```

THIS CODE DEFINES A FUNCTION CALLED detect_and predict_mask that takes in two arguments : a frame and two neutral wetworks, faceNet and MaskNet

```
# load the face mask detector model from disk
maskNet = load_model("/content/drive/MyDrive/AIC/TP/Project AI Face Mask Detection/Model/mask_detector.model")
```

this code is used to load the pre-trained model for detecting masks in image or video frames.

Once the model has been loaded, it can be used to predict whether a face in an imageor video frame is wearing a mask or not. The model will output a probability for each face, indicating the likelihood that the face is wearing a mask

```
# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
  """
  Params:
          js_reply: JavaScript object containing image from webcam
  Returns:
          img: OpenCV BGR image
  """
  # decode base64 image
  image_bytes = b64decode(js_reply.split(',')[1])
  # convert bytes to numpy array
  jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
  # decode numpy array into OpenCV BGR image
  img = cv2.imdecode(jpg_as_np, flags=1)

  return img

# function to convert OpenCV Rectangle bounding box image into base64 byte string to be overlayed on video stream
def bbox_to_bytes(bbox_array):
  """
  Params:
          bbox_array: Numpy array (pixels) containing rectangle to overlay on video stream.
  Returns:
```

the js_to image function takes in a JavaScript object containing an image from a webcam as an argument and return an image in OpenCV's RGB format

the bbox_to_bytes function takes in a numpy array containing a rectangle to overlay on a video stram and return a base64 string

```
# JavaScript to properly create our live video stream using our webcam as input
def video_stream():
  js = Javascript('''
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
    var labelElement;

    var pendingResolve = null;
    var shutdown = false;

    function removeDom() {
       stream.getVideoTracks()[0].stop();
       video.remove();
       div.remove();
       video = null;
       div = null;
       stream = null;
       imgElement = null;
       captureCanvas = null;
       labelElement = null;
    }
```

with viedo_stream function, we capture and display a video stream from a webcam in a web page

```
# start streaming video from webcam
video_stream()
# label for video
label_html = 'Capturing...'
# initialze bounding box to empty
bbox = ''
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    ##frame = vs.read()
    ##frame = imutils.resize(frame, width=400)


    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    frame = js_to_image(js_reply["img"])

    # detect faces in the frame and determine if they are wearing a
    # face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    # loop over the detected face locations and their corresponding
    # locations
```

Here we use the video_stream function defined earlier to capture a video stream from the user's webcam in a web page and display it in a loop

For each frame in the video stream, the code calls the detect_and predict_mask function to detect faces in the frame and predict whether the person is wearing a mask or not, the bounding boxes and labels for the deteced faces are then drawn on the frame using the cv2.putText and cv2.rectangle functions from **OpenCv** library,

3.1. Test Results of our Face Mask Training model

Here are some Live Test results of our Face Mask model :

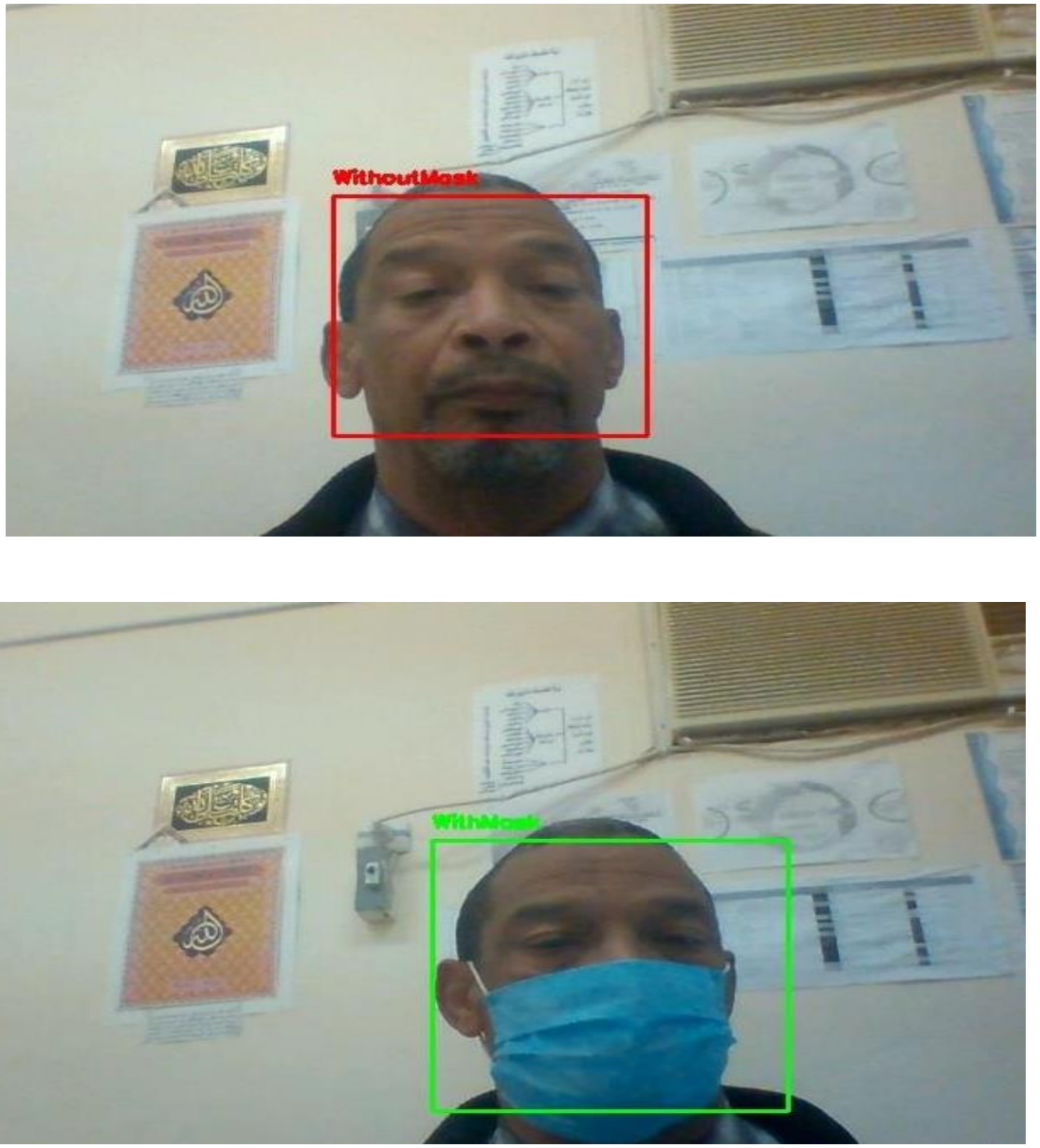


Figure 3.3 - Live Test of Our Face Mask Detector

Figure 3.4 Other Live Test results covering the three cases.

Therefore, we can see that our model gave good results and it can be used in real timescenarios.

General Conclusion

We believe that our model can detect faces and identify whether people are wearing a facemask or not. This model can perform the detection, with sufficient precision and accuracy, ones in test (webcam) videos as well as real life videos, which are entirely different,