HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Capstone project report

Online face recognition system

Submitted by

Doi Sy Thang - 20225528

Ngo Duy Dat - 20225480

Ta Ho Thanh Dat - 20225482

Nguyen Nhat Minh - 20225510

Nguyen Minh Quan - 20225520

Guided by

Assoc.Prof. Than Quang Khoat

Hanoi, December 2023

Abstract

This project intended to create a working system for face recognition. This technology is a branch of a more broaden research field namely computer vision, which has been developed for decades and has achieved many breakthroughs till now. In this project, we looked into some of the most simple concepts and architectures, by which, we hope to have the first insights of how to build a face recognition system step by step, as well as to grasp the relevant ideas and calculations. By inquiring the architecture of VGG-16, a highly accurate model, we managed to build a model with acceptable accuracy on a relative small scale. We built a dataset of 99 classes of labeled human faces with close to 3000 images in total. Considering the small data, our model was significantly simpler than its inspiration: it has 11 convolution layers compare to 16 of VGG-16. To verify the model, a face detection algorithm was applied to extract faces from images files or straight from live video captured by camera. Due to the deficiencies of the dataset and the non-optimized training model, the technical result was not as high as we expected.

Contents

Part I
Introduction

1 Motivation

Face recognition technology has become widespread, its commonality and impact on related computer vision tasks are apparent in our daily lives. From the tertiary industry to personal or even national security, this technology has proved its applications: access granting on personal smart devices, social administration by faces scanning in some countries. Noteworthy libraries like DeepFace and built-in functions like AppleID showcase their prevalence. Models such as VGG, FaceNet, OpenFace, and DLib contribute to the high accuracy of these systems, nearing human-level performance at 97 to 98%. Our hope was to grasp the main idea of face recognition by building a working system for classifying humans face. The motivation for this project goes beyond immediate objectives-technical proficiency; it represents a commitment to ongoing exploration and application of facial recognition technologies. This attempt laid the background for further delves into more tasks in computer vision in particular and machine learning in general.

2 Problem formulation

The project involves the application of computational algorithms to identify and authenticate individuals based on distinctive facial features. The methodology is quite simple: collect a sufficient number of images of some individuals' faces, put them in a model in which features on their faces are extracted and stored in feature maps. In fact, there are some challenges while carrying out this kind of research, many of those we have faced. The first and most obvious one is the high variability of human faces in term of shape, size, pose, illumination, makeup and so on. These factors make it hard for the algorithms to cope with different scenarios. A good diversely covered, non-bias dataset would enhance the overall performance, but collecting such well rounded datasets has never been easy. This project was carried out in a very small scale, we know that our dataset did not cover the least requirements, let alone all those factors to be a decent one. Another factor to be considered is the algorithms. On one hand, they should be accurate and strong enough to handle the diversity of faces and imperfection of dataset. On the other hand, they need to be efficient and adaptable to be executed on different devices and platforms. Researchers now are still finding more ways to cope with those obstacles.

3 Frameworks and libraries

Programming language

We used Python - a high level programming language for the code of this project. This is the most used language for data science according to DataCamp thanks to its friendly syntax and large number of built-in libraries, with some of them built exclusively for machine learning and deep learning tasks like Keras, or computer vision tasks in particular like OpenCV.

NumPy

NumPy allows fast numerical computing for large arrays and matrices, it also offers various functions to operate. It serves as a foundation for many computing libraries like Tensorflow in tasks like working with tensors.

Tensorflow

Developed by Google, this framework offers a comprehensive ecosystem of libraries and tools specialized for machine learning tasks. We used Tensorflow to import Keras, and initially indended to train model on GPU using this framework but did not manage to. Keras API was used by us to work with data manipulating and model deploying.

OpenCV

This library serves mainly for computer vision tasks, with many built-in tools for images and videos processing. Its is used for tasks like reading, displaying or filtering images, for example, we used it to convert images to grayscale in this project. OpenCV is also great working with cameras, this library was used to deploy our verification interface, which include some object detection access.

Part II

Overall Process

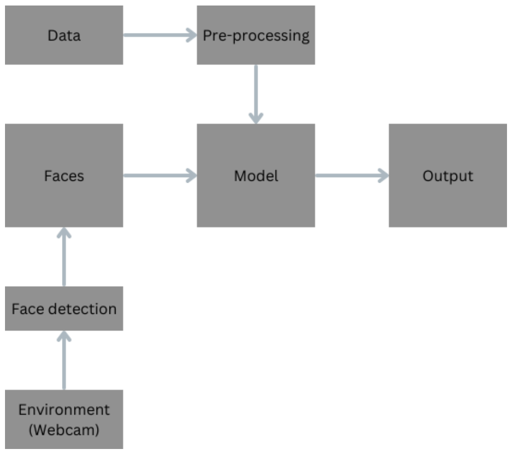


Figure 1 . The process of our system

1 Data Preparation

1.1 Data Acquisition and Annotation

The first step to build a face recognition model is definitely to gather a collection of people images. There are a variety of labeled datasets of human faces on internet, however we determined to make our own dataset to get the first understanding how a data scientist create data. Our group aimed to collect labeled data of 99 different people - a modest number but suitable to the scale of this project. We have used the web scraping tool bingdowloader to receive thousands of images of 95 celebrities and images of 4 of us were added later to the dataset. Each person images is stored in a folder named by that person.

1.2 Data Pre-processing

Data Cleaning: Although we have about 30-50 images dowloaded for each person in the dataset, many of them are noised data. They are:

1. Low resolution images: These images can significantly degrade model performance as it reduces the amount of information the model has to work with. Important details of people face can be lost.

2. Images in which there are multiple people: Putting these images into the model is harmful because the model has no idea which people faces it must learn.

3. Images with a face that has accessories(masks or glasses) : Because of the modest complexity of our model and the limitation of our data, these images also be considered noised data. We

must ensure that all images belong to a person are either faces with glasses or faces with no glasses.

Hence, we had to get rid of all kind of above images to ensure the performance of our model. After completing cleaning phase, some folders only had about 10-20 images remained which may leads to overfitting or underfitting. Thus, we manually download extra images to ensure each folder containing at least 30 pictures.

Face Extraction: After getting sufficient number of images, we had to extract human face from each picture. We located faces in by using face detection algorithm (the concept of the algorithm will be explained later in this report) to draw a square around the faces. Then we utilized cv2 to crop the detected faces from images.

Data Transformation and Data Reduction: All images we acquired until this point are digital RGB images. In RGB system, each pixel of a image has 3 channels: red, green, blue. Each color channel in a pixel is assigned a numerical value between 0 and 255 where 0 represents the absence of that color, while 255 represents its full intensity. The color of a pixel is determined by the combination of these values.

$$\begin{bmatrix} (100,100,50) & (101,112,3) & (131,20,80) \\ (150,210,130) & (10,120,130) & (111,120,130) \\ (10,260,30) & (200,20,30) & (100,20,3) \end{bmatrix}$$

Figure 2 . Matrix representation of 3 x 3 RGB image (source: Basic Deep Learning book)

A digital RGB images with size m x n can be represented as a m x n matrix in which each element is a vector (r_{ij},g_{ij},b_{ij}) that reflects the values of 3 channels of a pixel in the image. However, for better storage and processing, we have to split the above matrix to 3 matrices with the same size, each matrix stores only one channel.

Using large size RGB images would make the model train very slow and consume a lot of computer resources because there are too many parameters. Therefore we modified our data in two main steps to achieve higher computational efficiency.:

Data Transformation: all images were scaled down to the size of 224 x 224.
Data Reduction: all images were converted to grayscale. Each grayscale image can be represented by an one-channel matrix, instead of 3 one-channel matrix with the same size in RGB; thus reduce the number of parameters that the model must learn.

$$\begin{bmatrix} 0 & 215 & ... & 250 \\ 12 & 156 & ... & 1 \\ ... & ... & ... & ... \\ 244 & 255 & ... & 12 \end{bmatrix}$$

Figure 3 . Matrix representation of a grayscale image (source: Basic Deep Learning book)

1.3 Data Augmentation

Our number of images was relatively small, which can lead to overfitting, to increase the number of images, a technique called data augmentation was used. It is a method to generate more images based on the existing images. . In our model, 4 methods of augmentation are used: brightness adjusting, flipping, zooming and rotating. By those methods, our images are randomly brighten, flipped horizontally or vertically, rotated by different angles, zoomed in, zoomed out. All four methods are executed sequentially with each images creating the same number of images, but given that the variables of all methods are randomly generated in each epoch, the generated images which are the inputs for the model are different throughout the training process.

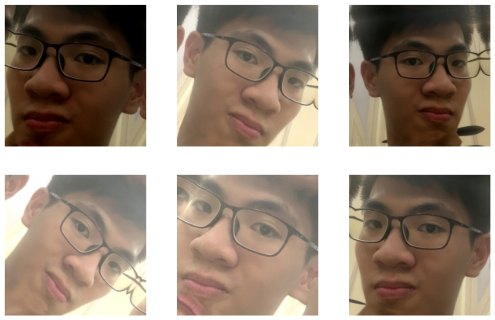


Figure 4 . Images generated from data augmentation

2 Training models

We used only the faces extracted from the images to put into the models. Those models follow the standard architecture including several convolutional blocks in which feature maps are generated from the images' features, transformed into non-linear form, downsampled, flattened, and connected. We modified the models several times in order to achieve the best accuracy. Each time, we systematically changed the learning rate or dropout, changed pool size, batch size, or stride and noted the changes in accuracy. Base on experiment, we slowly improved the ultimate accuracy, which reached 91%.

3 System verification

The highest accurate model was verified by testing its either with image files or in live respond. In the former way, using cv2, library image files were loaded, read and faces-extracted before altered to gray scale and resized to 224x224. The processed images were then classified using the trained model. With live respond, using VideoCapture(), a class in cv2 library, images are captured by device's camera as input for a face detection algorithm. This algorithm locates the faces and separates them from the background. The separated faces, if recognized, are later classified into labeled classes using the previous trained model. Results are shown on an interface, in which the recognized faces are presented alongside their corresponding labels. The images are captured under various conditions, angles, and due to the fact that our dataset has weaknesses that were mentioned above, the testing accuracy may not be as decent as expected.

Part III
Concept

1 Face detection Algorithm

1.1 Introduction

Face detection is one of the most important steps in computer vision and machine learning problems such as human interaction, facial expression detection and etc. In the digitalizing era, it turns into explosive applications as the advancement of mobile phones, computer systems, and the popularity of digital security, personality confirmation. Although the technology grows, the problem remains challenging in real-world applications due to the large facial variations of view angle, expression, and makeup. Many efficient algorithms and models have been researched successfully for the problem such as Histogram of Oriented Gradients (HOG), Convolutional Neural Network (CNN), Support Vector Machine-Based face detection, Successive Mean Quantization Transform (SMQT) features, and Sparse Network of Winnows (SNOW) Classifier. In this report, the algorithm proposed is Viola and Jones because of its simplicity and clearness for implementation.

Viola-Jones presents as a multi-stage algorithm with the improvement of speeding, and relatively high accuracy by combining Haar-Like Features, Integral Images, Adaptive Boosting, and Cascade Classifier to create a fast model in detecting faces.

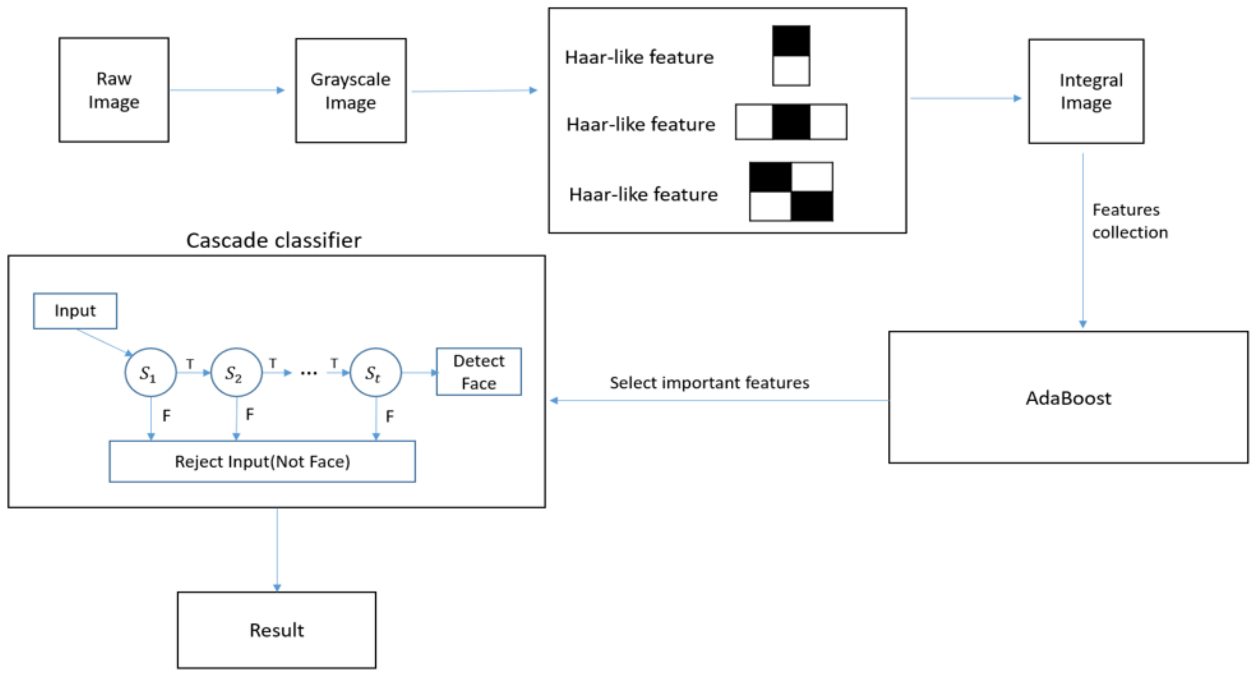


Figure 5 . Working process of Viola Jones for Face detection.

Face detection [7] is determined using the Viola Jones algorithm which refers to Figure 5 above, where the grayscale image transferred from the raw image would be split into sub-windows to scan whether or not the face exists. AdaBoost is used to select certain critical features from the extremely large number of features (far larger than number of pixels) to ensure fast classification by evaluating the performance of each feature in classifying between face and non-face, which is speeded up in calculations by invention of integral image. Face detection can be considered as a problem of classification with 2 classes, namely “face” and “not face”. The next step is to combine complex classifier generated from the AdaBoost in a multilevel structure; by focusing on the area of sub-window that has a chance, the result can be produced very fast.

1.2 Haar-like Features

Haar-Like Features[5] are inspired by the Haar wavelet transform which is a mathematical concept in signal processing and image analysis, proposed by Alfred Haar in 1909. Paul Viola and Michael Jones later adapted the idea of using Haar wavelets and developed the so-called Haar-like features. They are the digital image features or rectangle filter-like convolutional kernels used in computer vision to classify the intensity of pixels in regions of an image. There are various types of Haar Features but the most common ones used are two-rectangle features are responsible for detecting edges in horizontal or vertical directions, three-rectangle features are responsible for detecting lines, and four-rectangle features are responsible for detecting change in pixel intensities across diagonal.

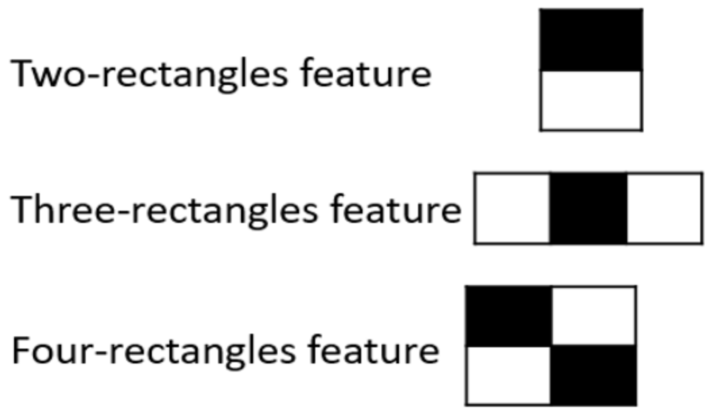


Figure 6 . Haar-like features

Haar-like features are transverse from the top left to the bottom right of the sub-window to extract particular features. It is commonly observed that the human face always has features related to roughness and edge rather than other objects, which play a role as the distinctive filters to classify effectively. For example, the region of eyes is darker than the bridge of nose.

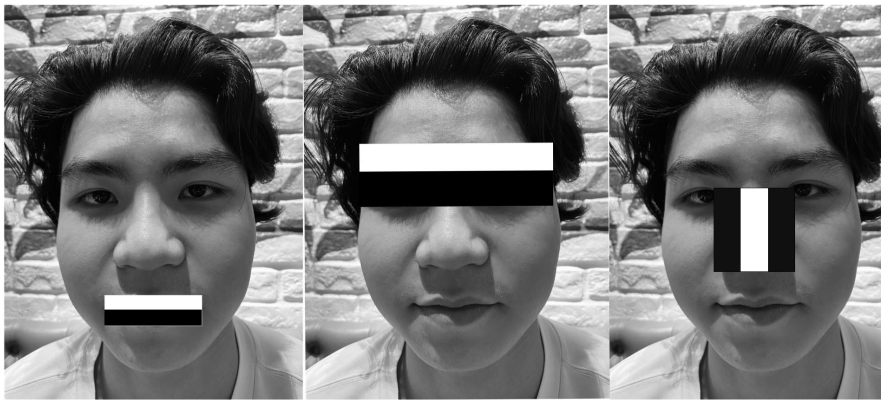
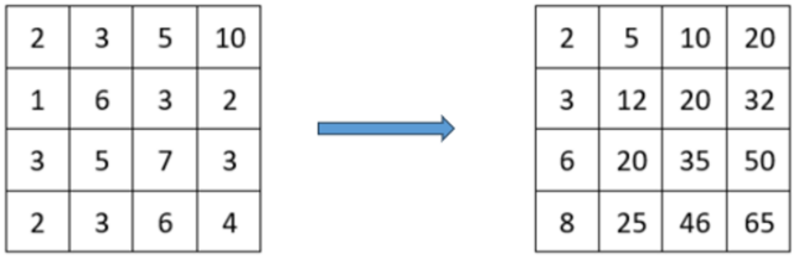


Figure 7

Moreover, the Haar-like feature has its scalar value which is calculated by the difference in the sum of pixel intensities., it is equal to the sum of pixel values in the black area and subtracts the sum of pixel values in the white area. This computation needs to iterate all the pixels in a particular feature, which costs a lot of time and money when facing a very large number of features. Hence the integral image method is proposed to reduce the computational time from O(H*W) to the O(1).

1.3 Integral Image

Integral image is a strong method to solve the excessive calculation problem from the rectangle feature by initializing the value as the sum of all pixels above and to the left. Particularly, the integral image at location x, y has the formula: $ii(x, y) = \sum_{x'x,y'y} i(x', y')$

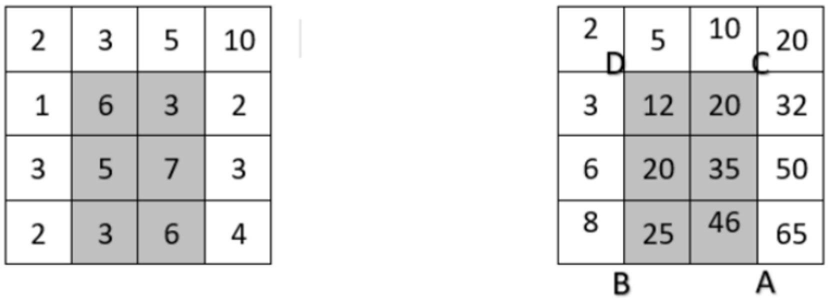


2	3	5	10
1	6	3	2
3	5	7	3
2	3	6	4

2	5	10	20
3	12	20	32
6	20	35	50
8	25	46	65

Figure 8

By integral image, it is easy to make calculation for the rectangular sum of pixels by using four values that exist in each corner of this rectangle. For example, to calculate the sum of grey rectangular in Haar Feature, we only need to use the formula A-B-C+D in the integral image, particularly in Figure 5 below, i.e.,46-10-8+2=30.



2	3	5	10
1	6	3	2
3	5	7	3
2	3	6	4

2	5	10	20
3	12	20	32
6	20	35	50
8	25	46	65

Figure 9

Indeed, the integral image can be computed from an image using a few operations per pixel, and once computed, any one of these Haar-like features can be computed at any scale or location in constant time. Particularly, we use the formula: A=B+C-D+ v, v is the value of pixel in the original image at the corresponding position to integral one.

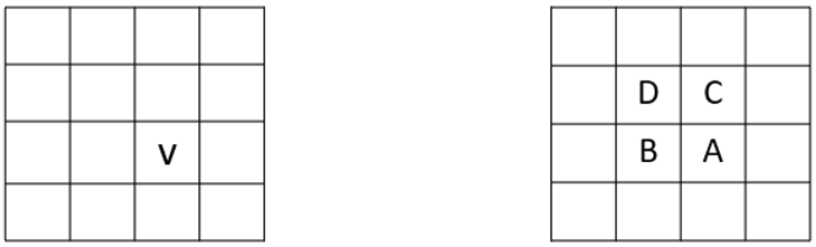


Figure 10

1.4 AdaBoost (Adaptive Boosting)

1.4.1 Ensemble learning

Ensemble learning [8] has emerged as a powerful paradigm that uses multiple so-called base learners to tackle the same problem in contrast to a mere learner in standard machine learning techniques to enhance predictive performance. This learning is appealing since it is efficient to boost the synergistic group of base learners being marginally better than a random guess to the strong learner who has the ability to make exceptionally accurate predictions by weighted voting. Thus, the base learner is also referred to as a weak learner. Moreover, the base learner is usually generated from training data by base learning algorithms that can be decision trees, neural networks, or other kinds of machine learning models. There are many ensemble methods such as Bagging, Stacking and etc.; however, we only concentrate on Boosting, i.e., AdaBoost (aka Adaptive Boosting) due to its presentation in Viola and Jones algorithm for Face Detection problem.

1.4.2 AdaBoost

Initially, Boosting is a family of algorithms that concentrate on sequentially training weak learners and assigning weights to data points, i.e., higher weights for misclassified instances. Boosting also has its roots in the theoretical framework in machine learning called PAC model because the birth of boosting algorithms originated from the answer to an interesting problem in 1989 which is whether a weak learner which just performs slightly better than a random guess in PAC model can be boosted into an arbitrarily accurate strong leaner. This question is of fundamental importance because if the answer is positive then any weak learner is potentially able to be boosted to a very accurate learner. Schapire proved that the answer is positive, and the proof is construction, i.e., boosting.

Among Boosting methods such as Gradient Boosting, XG Boost, etc., the most widely used boosting algorithm is AdaBoost, it gets its name by “adaptive” nature in the sense that subsequent weak learners are tweaked in favor of data points misclassified by previous classifiers.

AdaBoost is extremely simple to use and implement (far simpler than SVMs), and often gives very effective results. At each iteration, AdaBoost determines a new weak classifier relative to the lowest value in the weight distribution in the training set. The principal advantage of AdaBoost is that the training error converges exponentially towards zero and the generalization performance grows at each iteration when the null training error is reached by the algorithm. The training of this algorithm can be presented in pseudo-code and mathematical models as below.

Input

Data set : $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, $y_i \in \{-1, 1\} \; \forall \, i \in \{1, 2, \ldots, n\}$

Sample distribution of data points: $D_t(i) \; \forall \, i \in \{1, 2, \ldots, n\}$

h_t : weak leaners at t , $h : x_i \rightarrow \{-1, 1\} \forall \, i \in \{1, 2, \ldots, n\}$

H_t: strong learner at t.

Process

Initialize the sample distribution: $D_1(i) = \frac{1}{n} \; \forall \, i \in \{1, 2, \ldots, n\}$

For t=1,2,...,T do:

1. Train weak learners to minimize the error function

$$\epsilon_t = \frac{\sum_{y_i \neq h_t(x_i)} D_t(i)}{\sum_{i=1}^{n} D_t(i)} = \sum_{y_i \neq h_t(x_i)} D_t(i) \quad (since \sum_{i=1}^{n} D_t(i) = 1)$$

2. Determine the weight of the weak learner $h_t(x)$

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

3.Update the strong learner $H_t(n)$

$$H_t(n) = H_{t-1}(n) + \alpha_t h_t(n)$$

4. Update the distribution of data points

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{\sum_{i=1}^{n} D_t(i)} \begin{cases} e^{-\alpha_t} & \text{if } h_t(x) = y_i \\ \\ e^{\alpha_t} & \text{if } h_t(x) \neq y_i \end{cases} \\ &= \frac{D_t(i)}{\sum_{i=1}^{n} D_t(i)} e^{-y_i h_t(x_i) \alpha_t} \end{aligned}$$

Output:

$$H(n) = sign(\sum_{t=1}^{T} \alpha_t h_t(n))$$

From the process of AdaBoost above, a problem has been considered:

Why assign the weight of weak learner $h_t(n)$ equal to $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

The exponential loss would be used since it gives a simple and elegant update formula, and it is consistent with the goal of minimizing classification error.

Put L as the sum of exponential loss of $H_t(x)$ at t in each data point.Then:

$$L = \sum_{i=1}^{n} e^{-y_i H_t(x_i)} = \sum_{i=1}^{n} e^{-y_i H_t(x_i) + \alpha_t h_t(x_i)} = \sum_{i=1}^{n} e^{-y_i H_t(x_i)} e^{-y_i h_t(x_i)\alpha_t}$$
$$= \sum_{i=1}^{n} D_t(i) e^{-\alpha_t y_i h_t(x_i)} = \sum_{y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} + \sum_{y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t}$$

When the exponential loss is minimized by α_t, then partial derivative of the exponential loss for α_t is 0,$\frac{\delta L}{\delta \alpha_t} = 0$, solve this equation we can get:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{\sum_{y_i = h_t(x_i)} D_t(i)}{\sum_{y_i \neq h_t(x_i)} D_t(i)} \right) = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{1}$$

1.5 Cascade classifier

The overall form of the detection process is an organized set of features in multilevel classification, where a positive result from the first classifier prompts the evaluation of the second and a positive result from the second classifier subsequently prompts the third. The algorithm runs until the face is found, if any classifier returns a negative result, the whole sub-window is immediately rejected. A false negative occurs when the classifier is unable to detect the actual face from the image and

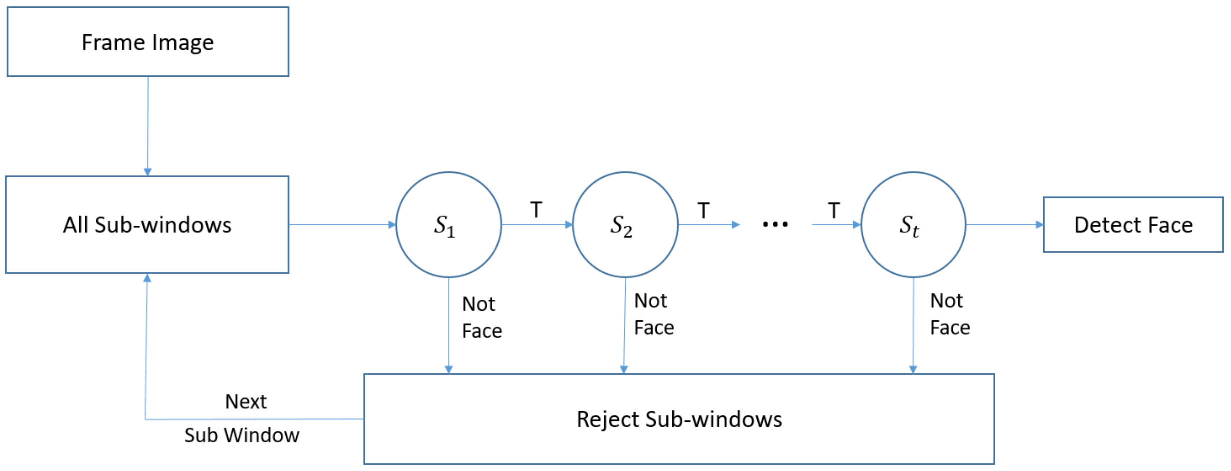


Figure 11 . Cascade classifier flow diagram

a true negative means that a non-face was correctly classified as not being the face in question. To work well, each stage of the cascade must have a low false negative rate, because if the actual object is classified as a non-face, then the classification of that branch stops, with no way to correct the mistake made. Therefore ,stages in the cascade are constructed by training classifiers using AdaBoost and then adjusting the threshold to minimize false negatives.

2 Artificial neural network

Artificial Neural Networks (ANN)[1] simulate the biological neural network in the human brain. It is the subfield of machine learning and artificial intelligence, that can learn and generalize from data training. Neural networks apply in various fields, including face recognition tasks. In this section overall the concept of ANN is the foundation knowledge of the convolutional neural network.

2.1 Artificial Neurons

An artificial neuron is the most simple structure in the artificial neural network, also known as a node or a perceptron. Each neuron will receive the input signals and provide an output value. In Figure 9, it visualizes the structure of a neuron. Let be $X = x_0, x_1, ..., x_n$ are all input signals of the neuron from other nodes or input data, each signal x_i is associated with a weight w_i which represents the degree of connection to the previous neuron.

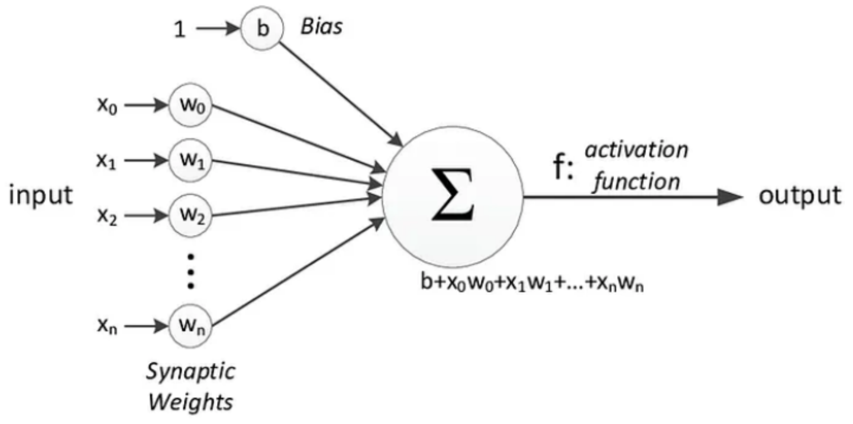


Figure 12 . The structure of a neuron (source: Medium)

The activation function determines the output of a neuron, which is then used as the input for the next layer in the network. Activation function is often non-linear allowing neural networks to learn more complex data. After taking all input, the total combination of input and weight denoted by $Net(W, X)$ will be added with bias value b and put on the activation function to compute the output of the neuron.

2.2 Neural Network Architecture

The general architecture of a neural network contains the number of layers and the number of neurons in each layer. And the number of connections for each neuron. The basic architecture of a neural network includes three layers, input layer, hidden layer, and output layer.

Input layer: containing neurons represent the input data, and the number of neurons in the layer typically corresponds to the dimensions of the input data. For example, in the classification image task, each neuron in the input layer may represent each pixel.
Hidden layers: the layer position between the input layer and the output layer and contains a

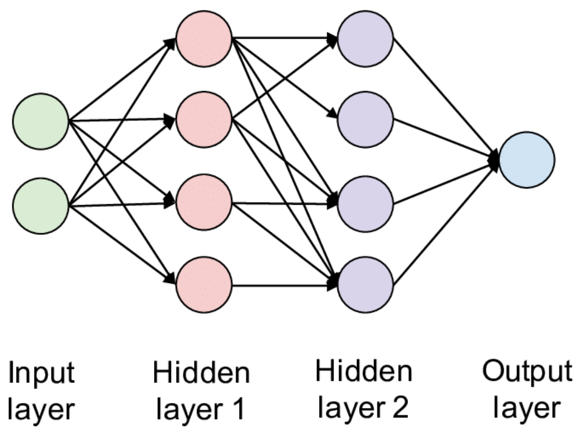


Figure 13 . Neural Network Architecture

variable number of hidden layers depending on the architecture of the neural network. Each hidden layer consists of multiple neurons, providing the ability to learn and extract complex features from the input data.

Output layers: containing neurons representing the result or prediction of the network. The number of neurons in this layer depends on the specific problem. For example, in the classification image task, each neuron in the output layer may represent a class of objects that the network is trying to classify.

Besides, the neural network can have several layers such as dropout, and batch normalization which is a technique the neural network learns more efficiently. We also utilize these layers in the proposed CNN model for the task of face recognition.

In the neural network, there are several basic types of neural networks such as the Feedforward network, Feedback network, and recurrent networks. Our model neural network is a feedforward network where neurons are connected to all neurons in the next and previous layer, without any backward connections. When input data is spread into the network, it passes through each layer and the output of the output of the preceding layer becomes the input for the subsequent layer, until at the output layer.

3 Convolutional neural network

3.1 Introduction

CNN [6] is a type of neural network architecture that is capable of learning by performing feature extraction through the use of convolution operations. CNN is designed to process grid-structured data like images and videos. Thus, it is highly suited for computer vision and applications involving image recognition, classification, and object detection. Inside the hidden layers lie three main types of layers of a CNN including convolution layers, pooling layers, and fully connected (or dense) layers.

3.2 Convolution layer (CONV)

The signature component that gives the model's name, convolution layers perform convolution operation on each spatial locality on the input volume with the use of sets of learnable filters called kernels.

Convolution is performed by "sliding" the kernel across each of the possible positions on the image input and then taking the dot product between its parameters and the matching spatial locality of the input to generate a feature map by the cross correlation function:

$$F(i,j) = (K * I)(i,j) := \sum_{m}\sum_{n} I(i-m, j-n)K(m,n)$$

with S being the pixel of the feature map generated at the valid position (i,j) by the convolution between the kernel K and the image input I within the valid range of m and n which are all the pixel locations where K overlaps I.

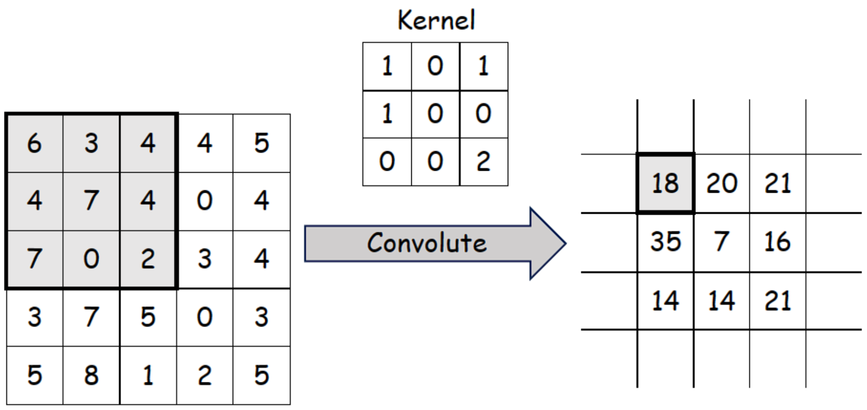
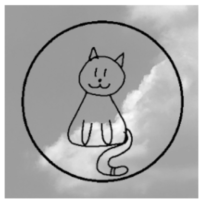
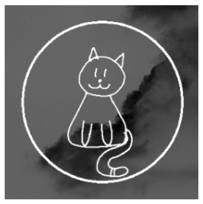
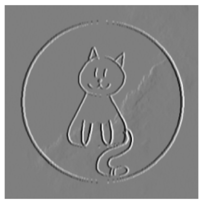


Figure 14 . Illustration of a convolution operation.

The use of convolution can be found in image processing with similar concepts. Depending on the parameter values, a kernel can cause a wide range of effects on the original image (see Table 1). Some kernels can highlight parts of the image (edge detection) while some blur out or inverse the image. Together, these make up the many "features" that can be infer from the image.

In CNN, multiple features are collected by using multiple kernels per layer the stack together to form the output. When using multiple CONV layers, convolution in the first layer will increase the receptive field of the later layers. Each feature in the next layer captures a larger spatial region in the input layer. For example, when performing two (3×3) convolutions consecutively, the captured region of the original input in the second layer increases from (3x3) to (5x5). This is a consequence of the fact that features in later layers capture complex characteristics of the image over larger regions by combining simpler features in previous layers.

Table 1 . Effects of some kernels on an image

Name	Kernel	Figure
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Inverse	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Vertical edge	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	

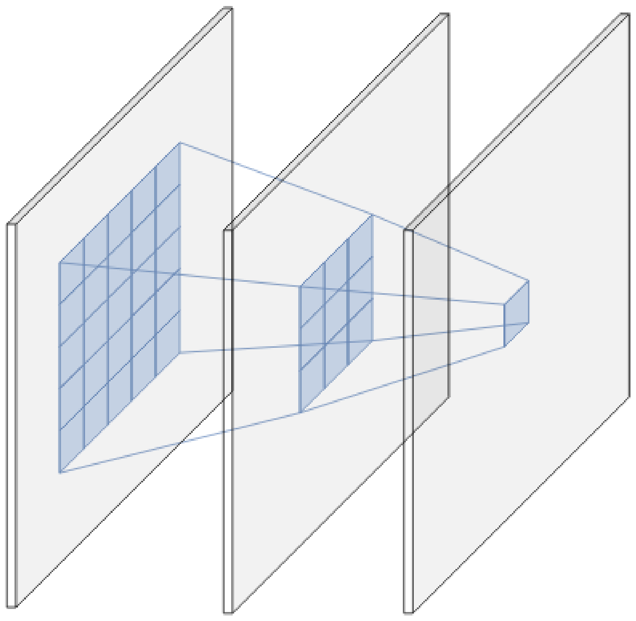
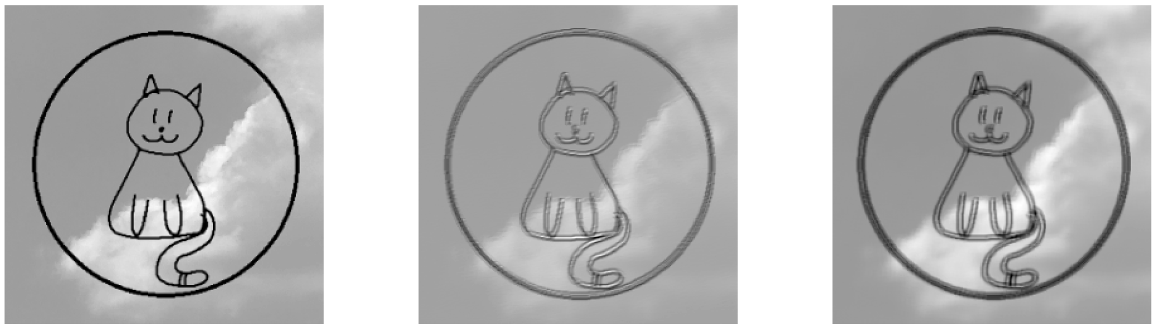


Figure 15 . Propogation of the receptive field over multiple CONV layers.

3.2.1 Kernel size

Kernel size is one of the three main hyperparameters of the kernel and is much smaller than the input size, usually set to (3x3) or (5x5) but some model can opt for larger kernel size such as (7x7) or (9x9) depend on the type of input image. The size is small in order to capture smaller features before combining to make bigger and more complex features later on. The size should also be of odd integers to reduce the distortion (aliasing) between layers as the previous layer pixels would be symmetrical around the output pixel.



(a) Original (b) Kernel (4x4) (c) Kernel (5x5)

Figure 16 . Effects of using larger kernels.

3.2.2 Padding

When performing convolution, the pixel at the edge of the image input is ignored since the input doesn't overlap with the kernel, resulting in the output having a smaller size compared to the input. This reduction is not desirable in most cases since more information at the edge will be lost deeper into the model. Padding is a method introduced to solve this by adding pixels around the border of the image data to ensure the output size being the same as the input. When there is no padding, we refer to it as "valid" padding and is usually not recommended due to the loss of marginal data mentioned before. "Same" padding is when zeros are added to the border of the input before convolution so that the output will have the same size as the input (this is only true when stride is equal to 1, we will talk about stride in the next part). The number of padding is approximately half of the kernel's width so it is also called "half" padding.

3.3 Maxpooling

Maxpooling is a layer that reduces the size of the feature map by sweeping through the feature map with a small-sized matrix and taking the maximum pixel value in the region that the matrix passes through. The max-pooling layers help reduce the parameters of the model, enabling the model focus on the specific features of the image.

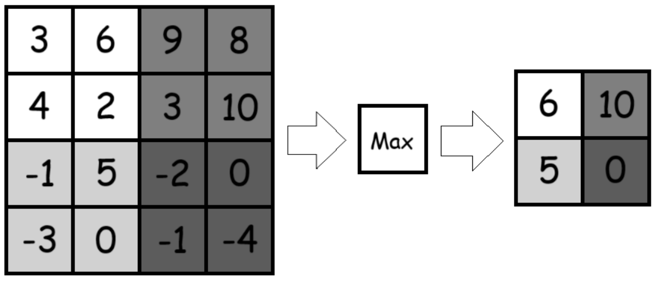


Figure 17 . Max pooling with pool size (2x2), stride 2

3.4 Fully connected layer (FC)

After the input data passes through the convolutional layer and the pooling layer, the input will be flattened by a flatten layer before reaching the fully connected layer. In this layer, each input node (or neuron) of the previous layer will be connected to all of the layer's neurons, hence the name. This layer functions in exactly the same way as forward propagation in traditional artificial neural networks with the features of the image being extracted and then tallied up to make a prediction in the last dense layer.

3.5 ReLU activation

An activation is a function that maps the output of a layer by setting an activation threshold for the next layer. The main purpose of an activation function is to introduce non-linearity to the neural network, this is important for image processing models where an image itself is not linear but a complex set of features that the models need to learn. ReLU, shorted for Rectified Linear Unit, is a simple activation but has become essential in today's image model. ReLU will only take the positive values of the input and deactivate the negative values by setting them to 0, which can be represented mathematically as a max function:

$$f_{ReLU}(x) = max(0, x)$$

Compared to other activation functions, ReLU is much more computationally efficient

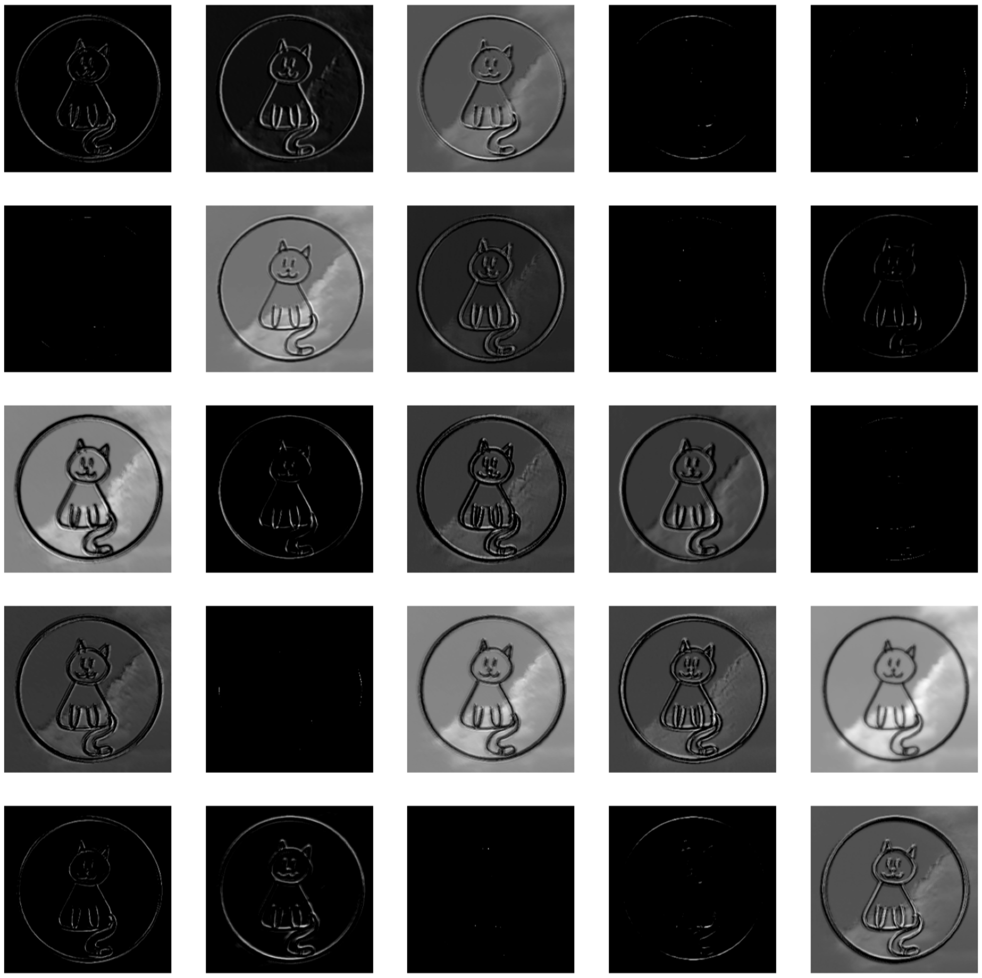


Figure 18 . Some feature maps after 5 convolutions with ReLU, fully blackout images are considered ”dead”.

3.6 Dropout

Dropout is a technique where randomly selected neurons are ignored during training and its weights are not updated during backpropagation. As a neural network learns, neuron weights settle into their context within the network. The weights of neurons are tuned for specific features. Neighboring neurons come to rely on this specialization, which when taken too far, can result in a model too specialized for the training data, a major cause of overfitting. By dropping some nodes, other nodes will be used to handle the representation of the image data to make predictions. This can result in multiple independent internal representations being learned by the network and the network can better generalize the features of an image.

The amount of neurons to be dropped out can depend on many factors and are tuned accordingly depending on the model performance. Dropout can be vital when the size of the training data is small where there is a rather small variance in how each feature is perceived by the model.

3.7 Batch Normalization

Batch Normalization [2] is a technique used in neural networks to make the training process more stable and faster. The main idea is to normalize the activation vectors from the hidden layer before passing them as the input of the next hidden layer by using the mean and variance of the current batch. Batch Norm find means μ and variance σ^2 of activation value of the mini-batch:

$$\mu = \frac{1}{n}\sum_i z_i$$

,

$$\sigma^2 = \frac{1}{n}\sum_i (z_i - \mu)^2$$

Then it normalizes the activation across the vector Z(i) to get the distribution with means close to zero and the standard deviation close to 1.

$z_i(norm) = \dfrac{z_i(norm) - \mu}{\sqrt{\sigma^2 - \epsilon}}$, ϵ: a small constant used for numerical stability(avoid division by zero)

Scale (γ) and shift(β): these are learnable parameters that allow the model to choose the optimal distribution of normalized inputs for each hidden layer. In particular, γ allows to adjust the standard deviation by scattering or gathering the distribution, while β allows to adjust the bias by shifting the curve on the right or on the left side.

$$z = \gamma . z_{norm} + \beta$$

Indeed, different from the training process, during inference, the activations of a layer are normalized using the mean and variance of the activations calculated during training, rather than using the mean and variance of the mini-batch.

Accoring to [4],[3], Batch Norm reduces the internal covariate shift of the network. The covariate shift is a change in input distribution of an internal layer of Neural Network. Applying Batch Norm ensures that mean and standard deviation of layer input will always remain the same.

Moreover, Batch Norm also has slight regularization effect since it is done in mini-batches instead of the entire dataset. The model ‘s data distribution sees each time has some noise, which is regarded as material for regularization to improve the overfitting problem.

3.8 The proposed model

In this section, we will mention the proposed convolutional neural network for face recognition tasks that are based on the VGG model and more our experience in the process of the training model. In Figure 16, it is the structure of our proposed model.

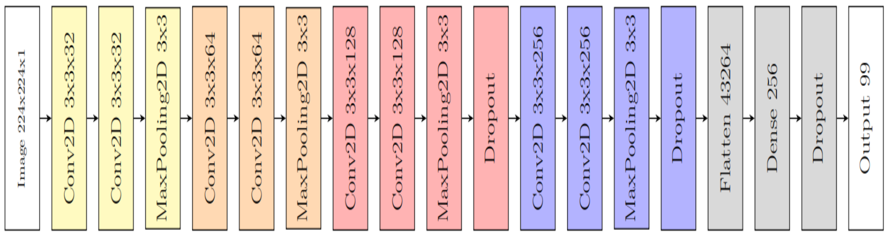


Figure 19 . Model CNN propose for Face Recognition task

The input of the model is a grayscale image with a size **224x224x1**. During the training process, we observed the performance of the model when learning from grayscale images and color images is similar. However, grayscale images support the model learning faster due to their less storage space. This can be explained that our image training only consists of faces, therefore the image data has few color features.

The architecture proposed convolutional neural network divided into four blocks. Each block includes two convolution layers and a max pooling layer, after each convolutional layer, the feature maps are normalized by using batch normalization, which helps stabilize the training process of the model. The feature maps through the max pooling layers at the end of the block reduce the size of feature maps, assist in reducing the parameters for the model, and help the model focus on crucial features.

The number of nodes in each convolutional layer of each block increases evenly. This helps the model avoid learning too much noisy data from the initial features and enables it to extract more specific features in the subsequent layers. However, When the model learns too many features from the convolutional layers led to overfitting, causing the model to not be able to generalize features on the test set. Therefore, in block 3 and block 4, dropout layers are applied after the convolutional layers, and the dropout rates for each node at the last layer of block 3 and block 4 are set to 0.05 and 0.1, respectively. The final block includes 256 feature maps with a size of 13x13. These feature maps then flattened through a flatten layer, converting them from a 2D shape to a 1D shape with 43264 nodes. Then through a dense layer with 256 nodes, and each node with a probability dropout equal to 0.65. The proposed model trains on data with 99 labels, at the output layer all values predict all labels transfer to probability by the sortmax function. And each the uncertainty of input image is the value of entropy of output distribution.

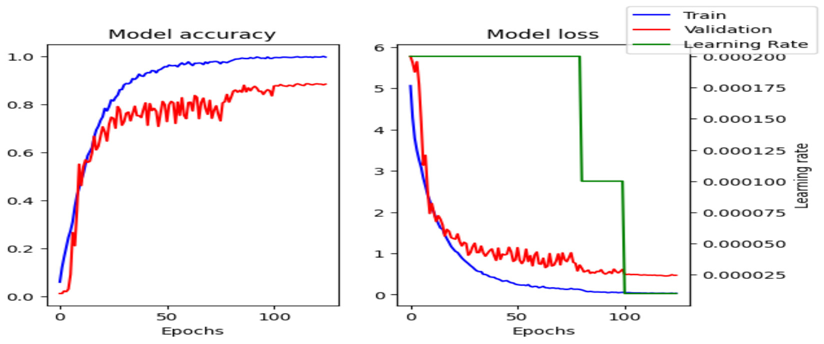


Figure 20 . The accuracy and loss function of proposed model

The model achieved a 91 percent accuracy on the test set, and its performance gradually stabilized towards the end of testing. The learning rate of the model was initialized at a small value of 0.0002, and this parameter was experimented with extensively during the training process, proving to stabilize the model's learning rate effectively.

4 Model Training Process

In this section, we will discuss about ideal training model. Training a convolutional neural network is learning the weights and biases of the neural network from the data training, and the goal of the learning process is to optimize the network performance. Weights are the parameters of the network that are adjusted to minimize the error between the distribution of predicted output and actual values. The process often uses the backpropagation algorithm.

4.1 Loss Function

In the learning process, the parameters of neural networks are subject to minimizing a certain objective function, which calculates different errors in the prediction of neural networks. It is called the cost function or loss function. The Optimization of neural network performance is also to minimize the loss function. In the following, we shall introduce some popular loss functions and choose a suitable loss function for the model.

4.1.1 Mean Square Error

Let W = [W_1,W_2,...,W_n] be matrix weight, and b = [b_1,b_2,...,b_n] be the vector bias and f(x) is an activation function (often softmax for classification image task) at the end of the output layer. X and Y are data training, and L(W,b, x_i,y_i) is a mean square error for each data point in (X, Y). The training error is represented by the Mean Square Error function follow as.

$$L(W, b, x_i, y_i) = \sum_{j=1}^{n} E\left[(\hat{y_j} - y_i)^2\right]$$

where E is the expectation operator. And $\hat{y_j}$ is output at node j for data point (x_i,y_j).

$$y_j = f(W_j^T x_i + b_j)$$

Total error for data training X, Y

$$J(W, b) = \sum_{x_i \in X} L(W, b, x_i, y_i)$$

Mean Square Error is extremely basic and just focuses on comparing vector outputs and vector labels. It will be not suitable for classification image problems if there is noisy data in the images, squaring the errors may increase the error magnitude and the model can tend to minimize the error without considering the overall structure of the images.

MSE (Mean Square Error) can impact the learning velocity of the model. In process learning, we must optimize the loss function by calculating the gradient of the loss function. As for MSE, When the predicted value is almost near to the growth truth, the gradient of the loss function approaches zero so that the acceleration of the learning process tends to slow down. In conclusion, the MSE function is not efficient for image classification problems. In the next section, we will discuss about cross-entropy loss function, considered effective in image classification.

4.1.2 Cross Entropy

Let p and q be two distributions on R. The cross-entropy of the distribution q relative to distribution p defined that.

$$H(p, q) = E_p[-\log(q)]$$

Where E is the expected value operator with respect to the distribution p.With p and q are discrete distributions, formulated by.

$$H(p, q) = -\sum_i p_i \log(q_i)$$

Cross-entropy H(p,q) of distribution q relative p more least than the two distributions are more similar, the minimum of cross-entropy occurs for q = p. The purpose of the cross entropy loss function measure the degree of error between predicting the distribution and growth truth distribution. Consider the Cross-Entropy loss function for the weight of the model.

$$L(W, b, x_i, y_i) = -\sum_{j=1}^{n} y_i \log(\hat{yj})$$

Total error for data training X, Y

$$J(W, b) = \sum_{x_i \in X} L(W, b, x_i, y_i)$$

In three examples(figure 20), the minimum value of two loss functions approach at p = q. However, the cross-entropy function takes on larger values as p and a q diverge, explicitly illustrating the dissimilarity between the two distributions. In contrast, the mean squared error (MSE) function lacks this capacity, as it is only oriented towards error minimization. This observation provides insight into the statement made in section 4.1.1 regarding MSE inability to effectively capture the overall structural differences in the image.

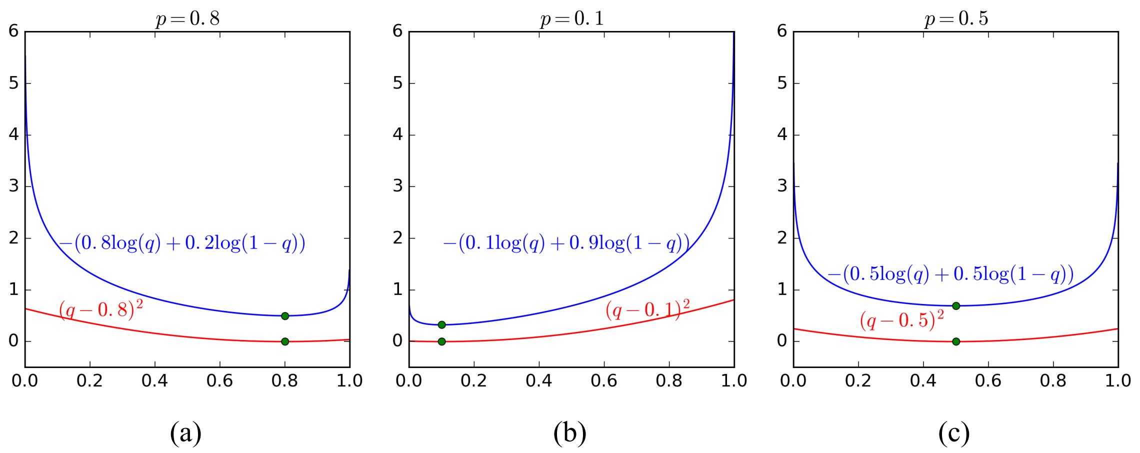


Figure 21 . Compare between Cross entropy and MSE (blog machine learning basic)

Another advantage of the cross-entropy loss is that computing gradients does not give the issue of vanishing gradients (when the gradient of the loss function for the weights of the model becomes extremely small, approaching zero, it gives an issue to update weight that will slow down significantly in some cases) as observed in MSE. Based on all analyses, our proposed model uses cross-entropy for loss function.

4.2 Optimization Algorithm

The learning process of the model involves the optimization of the loss function to learn optimal parameters for the given problem. One method to find the minimum of loss function identifies all local minimum points by the derivative of a function and chooses which point is optimal for the function (global optimization is often impractical, especially in non-convex optimization problems). In some cases, computing derivatives can be particularly challenging, especially when dealing with high-dimensional data. In response to this challenge, the Gradient Descent algorithm and its various adaptations are commonly used to solve. This section reviews basic concepts of algorithms and identifies optimization algorithms for classification image tasks.

4.2.1 Gradient Descent

The gradient Descent algorithm is a procedure of finding the minimum of a function by navigating the direction of maximum cost decrease. The fundamental idea of the algorithm is initiated from a point and subsequently proceeds toward the optimal point through an iterative operation.

Consider function f(x), assuming x_t is the point obtained after the t-th iteration. The purpose of gradient descent is to take x_t closer to x* (optimal point). At each iteration, x_t must move by an amount Δt.

$$x_{t+1} = x_t + \Delta t$$

Where Δt is a quantity opposite in sign to f'(x_t)

$$\Delta t = -\eta f'(x_t)$$

η is the learning rate of the model, the negative sign signifies moving in the opposite direction to the derivative.

Consider the loss function of the model. Let J(W,b) be the loss function with W = [$W1$,$W2$,...,Wn] be vector weight and b vector bias. The formula here only considers an update to vector weight at the output layer with n neurons.

$$\nabla J(W,b) = [\frac{\partial J(W,b)}{\partial W_1}, \frac{\partial J(W,b)}{\partial W_2}, ..., \frac{\partial J(W,b)}{\partial W_n}]^T$$
$$W_{t+1} = W_t - \eta \nabla J(W,b)$$

Where $\nabla J(W,b)$ vector gradient of loss function $J(W,b)$ determines the direction of the loss function.

4.2.2 Gradient Descent Momentum

The learning speed of GD can fluctuate greatly with η large that can be moved on global optima, and with η small the learning speed of GD can be slow affecting the process of learning. The gradient descent momentum is a solution to reduce fluctuation and accelerate the learning speed of the model. The main idea of GDM (gradient descent momentum) is to add a momentum component, which represents information about the velocity from the previous step. The formula update of gradient descent momentum for weight in process training.

$$W_{t+1} = W_t + \Delta W_t$$

$$\Delta W_t = -\eta \nabla J(W,b) + \alpha \Delta W_{t-1}$$

where α is a momentum parameter (α normally assign with 0.9) and note that (η+α) ~ 1

4.2.3 RMSprop (Root Mean Square Propagate)

As mentioned before the learning speed and the learning rate have a significant impact on the effectiveness of the learning process. To control this, the RMSprop algorithm automatically adjusts the learning speed for the process and selects suitable learning rates for each parameter in the model.

The concept of RMSprop is that divide the learning rate by an exponentially decreasing moving average of the squared gradient. The weight update formula of the algorithm is implemented as follows.

$$W_{t+1} = W_t - \frac{n}{\sqrt{E[g^2]_t + e}} \cdot \nabla J(W_t)$$

$$E[g^2]_t = \beta \cdot E[g^2]_{t-1} + (1-\beta) \cdot (\nabla J(W_t))^2$$

Where $E[g^2]_t$ is the mean square gradient of the loss function and e is the small epsilon term to avoid division by zero,β is typically between 0.9 and 0.999.

The RMSprop algorithm can avoid vanishing gradients by adjusting the learning rate based on the gradient values. In the updated formula of RMSprop, if the mean square gradient $E[g^2]_t$ increases leading to a reduction in the learning rate. Otherwise, when the value $E[g^2]_t$ decreases allowing the learning rate to increase.

4.2.4 Adam (Adaptive Moment Estimation)

The Adam optimization algorithm takes ideas from the RMSprop algorithm and the momentum method. Adam combines both the moment of the gradient and the moment of the squared gradient, supporting to enhanced efficiency and flexibility of the training model. The process update weight of the Adam algorithm follows as.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla J(w_t)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)[\nabla J(W_t)]^2$$

where m_t is the moment of gradient, and v_t is the moment of squared gradient, note that β_1 and β_2 belong to [0,1). To enhance the stability and accuracy of estimating the moment gradient and squared moment, m_t and v_t are added as bias correction terms.

$$\hat{m_t} = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v_t} = \frac{v_t}{1-\beta_1^t}$$

$$W_t = W_{t-1} - \frac{\eta \cdot m_t}{\sqrt{v^t} + e}$$

Adam typically performs well in preventing overfitting, especially when dealing with large data or models with a large number of parameters. In conclusion, Adam will use an optimization algorithm for training the model of the convolution neural network.

4.3 Backpropagation Algorithm

The learning process of a convolutional neural network (CNN) is based on a backpropagation algorithm, aiming to find the optimal weights for the model. This algorithm is divided into two stages, the first stage is forward propagation (propagation from the input layer to the output layer), and the second stage is backward propagation (propagation from the output layer back to the input layer).The backward propagation process is repeated across epochs (each epoch is the period learning all data training) to allow the model to learn more complex features, improving the predictive performance of the model. In our proposed convolutional neural network model, the model shall learn by mini-batch learning technique. Especially, in each epoch, the training dataset is divided into batch-size, and update weights after each batch. The technique learning is proven more efficient than batch learning.

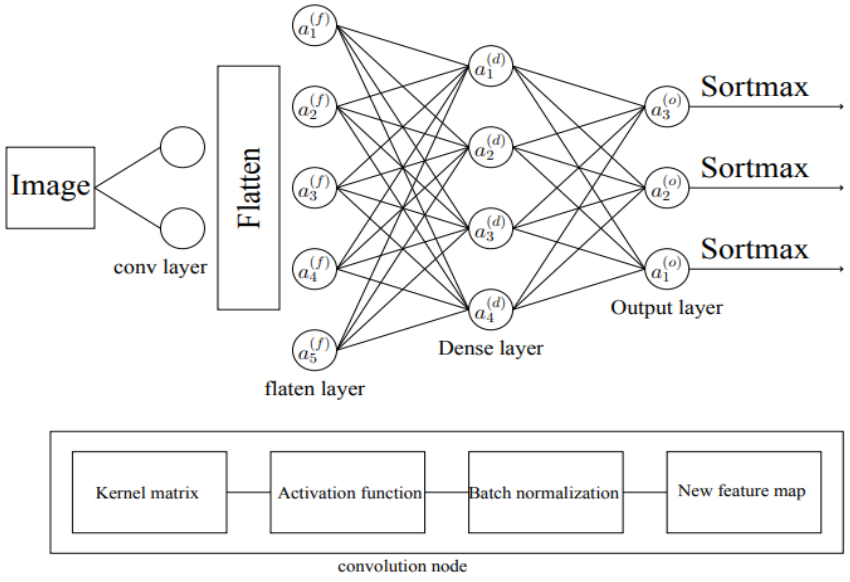


Figure 22 . Basic structure of proposed model CNN

In the following, We shall discuss the process of the propagation on the basic structure proposed model (CNN) in Figure 22.
1. Feedforward propagation
Firstly, The initial model weights are usually randomly valued, with weights typically initialized to small values following a Gaussian or uniform distribution. This supports the model to avoid issues such as vanishing gradients and weights converging to similar values.

Let be $W^{(d)}$, $W^{(o)}$ are matrix weight at dense layer and output layer that show the connections between layers. And $a^{(f)}$,$a^{(d)}$,$a^{(o)}$ are vector output values at the flatten layer, Dense layer, and output layer, respectively. For each pair of training data (x,y), where x is an image and y is the label of the image encoded into a vector with a dimension corresponding to the number of labels to be predicted, in Figure 14 that value is equal to 3. The image x passes through the convolutional layer and undergoes the convolutional operation (operator to feature extraction from the image) with each kernel matrix at the convolutional node. After that, the feature map shall through the flattening step, aiming to transform the feature map from 2D to 1D. Subsequently, the feed forword propagation to the remaining layers will be represented as follows.

$$a_i^{(d)} = f(W_i^{(d)T} a^{(f)} + b_i^{(d)}), i = \overline{1,4}$$

$$a_i^{(o)} = f(W_i^{(o)T} a^{(d)} + b_i^{(o)}), i = \overline{1,3}$$

2. Backforward propagation

In the backforward propagation, the output results will be used to compute the error value with the actual output, and the errors will be propagated back to the previous layer to update weights.

$$\hat{y}_i = \frac{exp(a_i^{(o)})}{\sum_{j=1}^{3} exp(a_j^{(o)})} = softmax(a_i^{(o)})$$

Where $\hat{y}_i$ value predict

$$J(W^{(o)}, b) = -\sum_{i=1}^{3} y_i \log \hat{y}_i = -\sum_{i=1}^{3} y_i \log \frac{exp(a_i^{(o)})}{\sum_{j=1}^{3} exp(a_j^{(o)})}$$
$$= -\sum_{i=1}^{3} y_i a_i^{(o)} + \sum_{i=1}^{3} y_i \log \sum_{j=1}^{3} exp(a_j^{(o)}) = -\sum_{i=1}^{3} y_i a_i^{(o)} + \log \sum_{j=1}^{3} exp(a_j^{(o)})$$

Consider y is a one-hot vector then $\sum_{i=1}^{3} y_i = 1$.

$$J(W^{(o)}, b) = -\sum_{i=1}^{3} y_i f(W_i^{(o)T} a^{(d)} + b_i^{(o)}) + \log [\sum_{i=1}^{3} exp(f(W_j^{(o)T} a^{(d)} + b_i^{(o)}))]$$

Let be $\delta^{(o)}$, $\delta^{(d)}$ are signal error back ward to layer output, layer dense.

$$\delta_i^{(o)} = \frac{\partial J(W^{(o)}, b)}{\partial a_i^{(o)}} = a^{(o)\prime}{}_i(\hat{y}_i - y_i)$$

$$\nabla J(W_i^{(o)}) = \frac{\partial J(W^{(o)}, b)}{\partial W_i^{(o)}}$$

$$= -y_i a^{(d)} f'(W_i^{(o)T} a^{(d)} + b_i^{(o)}) + a^{(d)} f'(W_i^{(o)T} a^{(d)} + b_i^{(o)}) \frac{\exp(f(W_i^{(o)T} a^{(d)} + b_i^{(o)}))}{\sum_{j=1}^{3} \exp(f(W_j^{(o)T} a^{(d)} + b_j^{(o)}))}$$

$$= a^{(d)} f'(W_i^{(o)T} a^{(d)} + b_i^{(o)})(\hat{y}_i - y_i) = a^{(d)} \delta_i^{(o)}$$

similarly, as for the dense layer

$$\delta_i^{(d)} = a^{(d)\prime}{}_i \sum_{i=1}^{3} \delta_i^{(o)} W_i^{(o)}$$

After calculating the signal error, the weight vector at each node is updated by the Adam algorithm that was mentioned before. The signal error from fully connected layers continues to propagate the convolutional layer and update weight for kernel matrix.

Part IV
VGG-16

Transfer learning

Transfer learning is a technique in machine learning in which a model trained on a specific task can be reused and adapted to a related task to boost performance. Indeed, in the era of digitalization, many real-world applications use Machine Learning models to optimize their problems with limited data resources, data gathering also becomes unaffordable investing since they are expensive and inaccessible. Compared with self-built models with a specific set of data, which is affected badly by an imbalanced dataset, non-presentative dataset or extremely small dataset, transfer learning performs surprisingly well by allowing leverage the knowledge from the pre-trained model to solve problem. Moreover, by using pre-trained features as a starting point, models often generalize better to new tasks at the beginning.

Generally, transfer learning process including two main steps. The first step is to choose the model close to the task, then extract the base network from the model without the fully connected layer. The second is that the fully connected layers considered as a MLP network are linked to the base network to reduce the dimensionality and calculate the probability distribution of the output. The process of continual training of weights based on task-specific data is fine-tuning process.

Other model for face recognition VGG-16

VGG stands for Visual Geometry Group which is the department in the university of authors, and 16 refer the 16 number of convolution layers.VGG-16 which was one of the most popular models submitted to ILSVRC-2014 when achieving almost 92.7% top-5 test accuracy in ImageNet. Indeed, VGG-16 consists of 13 convolution layers and 3 fully connected layers, especially, the differnet features of VGG from the previous is that the architecture that use successive Convolution layer and use a Max Pooling layer afterwards,ie.((Conv).m+Pooling).n

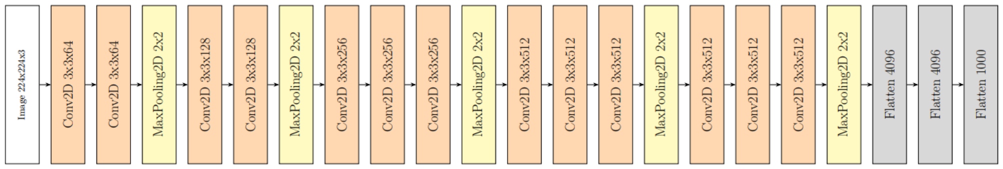


Figure 23 . VGG-16 Architecture

Compared with previous network at that time such as Lenet-5, AlexNet, VGG-16 uses multiple kernel with size 3x3 in consecutive order instead of using kernel 5x5, 11x11 in AlexNet which make calculation more complex and resources wasting, to reduce the number of parameters. Moreover, multiple convolution layers with ReLU activation function can extract face features better.

Result

The image below shows the result of training the VGG-16 on our dataset:

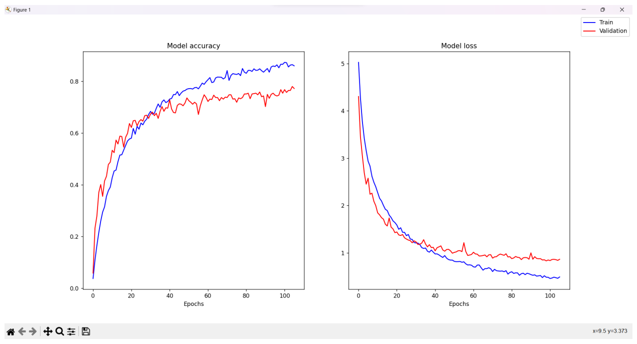


Figure 24 . VGG-16 performance

It is noticeable that model using VGG-16 as a base model to train the dataset with 99 labels can not reach high in training accuracy in the early epochs since the base model is freexe in training, and the model only learn in the MLP we added afterwards.

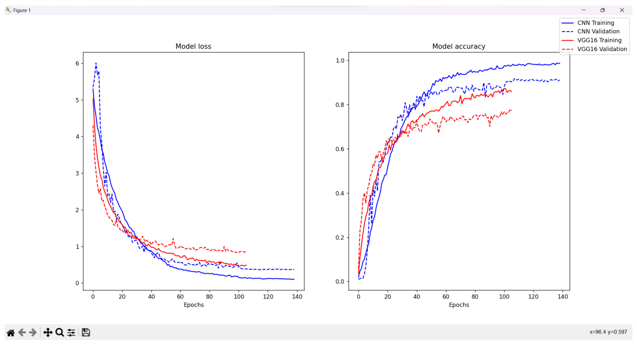


Figure 25 . Comparison between two models performances

The accuracy of VGG-16 started better than that of our model, it converged faster since it was pre-trained. The under-performance of the VGG-16 may be due to the deficiency of our dataset: the dataset on which this model reached very accuracy had millions of images, while ours had thousands. Our model was trained several times to best-fit the dataset we built, that may be the reason why it outperformed VGG-16.

Part V
Conclusion

We managed to implement a face recognition system using some of the most simple structures and methods, following the process: data preparing and processing, model building, optimizing and finally verifying. Managing to achieve an acceptable performance of the model, we know that everything need to be improved. Nevertheless, keeping in mind that the most important thing we achieved was knowledge at the end of the day, this project should be seen as the preliminary step. After finishing the project, we now know what are the criteria for a sufficient dataset, what is the general structure for a model used for classifying or identifying objects in general and all the relevant calculations or methods included.

Researchers are now still trying to find ways to improve facial recognition algorithms, with the burst of data, developments of modern hardware, especially with the advent of generative AI in visual recognition, this technology is becoming more and more accessible and reliable. Although more developments means more uprising problem, like how the 3D printing technology puts new challenges on the security access, we believe that face recognition will still find its way to be a vital technology in the future.

We want to thank professor Than Quang Khoat for being our instructor in this semester. We appreciate all the assistance and suggestions from our tutor Doan The Vinh and seniors in writing the report. Thank you professor Khoat for reading our report,,this is our first time trying to write one, were there any mistakes or misunderstandings, please inform us so we can improve ourselves in upcoming projects.

References

[1] Charu C. Aggarwal. An introduction to neural networks. In *Neural Networks and Deep Learning*. Springer International Publishing AG, part of Springer Nature 2018.

[2] Johann Buber. Batch normalization in 3 levels of understanding - Medium.com. **URL: https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338l**.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[4] Ketan. Batch Norm Explained Visually — How it works, and why neural networks need it - Medium.com. **URL: https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338l**.

[5] Takeshi Mita, Toshimitsu Kaneko, and Osamu Hori. Joint haar-like features for face detection. volume 2, pages 1619 – 1626 Vol. 2, 11 2005.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[7] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. volume 1, pages I–511, 2 2001.

[8] Zhi-Hua Zhou. *Ensemble Methods - Foundations and Algorithms*. Taylor & Francis Group,, Cambrige, UK, 2012.