Vu Thuy Vy Tran

REAL-TIME EMOTION DETECTION USING PYTHON

Thesis

CENTRIA UNIVERSITY OF APPLIED SCIENCES

Information Technology

2023

ABSTRACT

Centria University of Applied Sciences	Date	Author
	May 2023	Vu Thuy Vy Tran
Degree programme Bachelor of Engineering – Information Technology		
Name of thesis REAL-TIME EMOTION DETECTION USING PYTHON		
Centria supervisor Jari Isohanni		**Pages** 34 + 3
Instructor representing commissioning institution or company Jari Isohanni		

The thesis' intention was to understand more about the application of Deep Learning in how to develop real-time emotion recognition detection. The study covered the basic of Python programming language, with fundamental information about CNN (Convolutional Neural Network), TensorFlow, Keras and OpenCV.

The thesis consist of three parts, which included theoretical sections, practical sections and evaluation. The theoretical has an introduction about Deep Learning and imagine processing. The practical part will showcase toward a program that can detect human faces and recognize their facial emotion through cameras in real time with Deep Learning methods for face recognition and Python programming language. After the successful test running, the project of Emotion Detection will using webcam to determine user's facial expression with accuracy rate and emotion labels.

Key words

Convolutional Neural Network, Deep Learning, facial recognition, Machine Mearning, Python.

CONCEPT DEFINITIONS

AI

Artifact Intelligence

ANN

Artifact Neural Network

BSD

Berkeley Software Distribution. A low restriction type of license for open-source software that does not put requirements on redistribution

CNN

Convolutional Neural Network

DL

Deep Learning

FACS

Face Action Coding System

FER

Facial Expression Recognition

Haar Cascade

A model class that is used for face detection (Haar Cascade face detection)

OpenCV

Open Computer Vision. A set of software tools for real-time image processing, video, analytics, and machine learning

Wavelet Transforms

Mathematical tools for analysing data where features vary over different scales.

WISARD

WIlkie, Stoneham and Aleksander, Recognition Device

ABSTRACT

CONCEPT DEFINITIONS

CONTENTS

1 INTRODUCTION....1

2 THEORETICAL BASICS....2
2.1 Facial Expression Recognition....2
2.1.1 Knowledge-Based Methods....4
2.1.2 Template Matching....6
2.1.3 Appearance-Based Method....8
2.2 Deep Learning....9
2.2.1 Artificial Neural Network....9
2.2.2 Deep Learning Method....12
2.2.3 Real Life Application....13
2.3 Convolutional Neural Networks....14
2.3.1 CNN Architect....14
2.3.2 Typical CNN Models....17

3 RELATED TECHNOLOGY....22
3.1 Python Programming Language....22
3.2 TensorFlow....23
3.3 Keras....24
3.4 OpenCV....24
3.5 NumPy and SciPy....25

4 IMPLETION OF THE PROJECT....26
4.1 Preparing the dataset....26
4.2 Importing Libraries....27
4.3 Data Visualisation....28
4.4 Pre-processing Data....29
4.5 Convolutional Neural Network (CNN) Model....30
4.6 Test Model and Validation....31
4.7 Emotion Detection Process....33
4.8 Final Result....37

5 CONCLUSION....38

REFERENCES....39

FIGURES

FIGURE 1: Basic steps involved in automated FER 3
FIGURE 2: Detecting Faces in Images Examples 5
FIGURE 3: A typical face kind of knowledge-based method in facial analysis research 6
FIGURE 4: Projection Method 6
FIGURE 5: Face template 8
FIGURE 6: Structure of a biological neuron 10
FIGURE 7: The structure of the artificial neural network 11
FIGURE 8: The processing of a neuron in an ANN 12
FIGURE 9: Model of basic layers of CNN 15
FIGURE 10: Illustrate convolution on an image matrix 16
FIGURE 11: Average Pooling and Max Pooling Methods 17
FIGURE 12: LeNet-5 Architecture 18
FIGURE 13: AlexNet Architecture 19
FIGURE 14: ZFNet Architecture 19
FIGURE 15: VGG-16 Architecture 20
FIGURE 16: InceptionV1 Architecture 21
FIGURE 17: Tensor Architecture 23
FIGURE 18: Implementation model 26
FIGURE 19: Some sample images from FER2013 that could be confused for the machine 27
FIGURE 20: Imported libraries for the training and testing process. 28
FIGURE 21: The distribution of emotions in the dataset 28
FIGURE 22: Image pre-processing 29
FIGURE 23: A CNN model with multiple convolutional layers, pooling layers, and fully connected layers 30
FIGURE 24: Class probabilities and configuration of the optimizer and compilation of the model 31
FIGURE 25: The training of a Convolutional Neural Network (CNN) for Emotion Detection 32
FIGURE 26: Visualization of the performance of the trained model during the training process 33
FIGURE 27: Utilizing the imported model and setup the face detection classifier 34
FIGURE 28: Initialising webcam with OpenCV library 34
FIGURE 29: Font and label properties with frames capturing functions. 35
FIGURE 30: Emotion predictions for the input (resized) image 35

FIGURE 31: The resulting frame.....36
FIGURE 32: Closing and released the webcam’s resources.....36
FIGURE 33: Displayed result and performance. 37

1 INTRODUCTION

The problem of facial emotion detection has a long history of research. Since 1964, Bledsoe was the first to build an automatic face recognition program combined with a computer system, by classifying faces on the basis of manually entered benchmarks. The parameters for classification are the standard distance, the ratio between points such as angle, eyes, mouth, tip of nose and tip of chin. Later, Bell Labs developed a vector-based technique with 21 face attributes detected using the standard pattern classification technique. The selected attributes are mainly evaluated: hair colour, length of ears, thickness of lips. In 1986, the neural network-based WISARD system was able to recognize facial expressions and statuses in a limited way. (Bledsoe 1964.)

Facial emotion detection is the next evolution of face detection, but there are many views in the definition of emotion, which is very unclear, which let Matsumoto divides facial emotions into 7 main groups of expressions: Happy, Surprised, Satisfied, Sad, Angry, Wrath and Fear. (Matsumoto & Hwang 1991). The Radboud Faces Database divides facial emotions into eight categories: Anger, Indignation, Fear, Happiness, Sadness, Surprise, Contempt, and Neutrality. However, Mase's group suggest that only four emotions are clearly expressed: Happiness, Surprise, Anger and Indignation; Other types of emotions are often ambiguous and are highly dependent on the experience of the observer, which means it cannot be quantified accurately. (Mase 1991). Dataset Kaggle FER-F2013 has only 7 emotions: Anger, Anger, Fear, Happiness, Sadness, Surprise, and Neutrality. (Kaggle 2020.)

Deep learning networks are widely applied to facial emotion detection, especially those suitable for processing image data such as Convolutional Neural Network (CNN), Deep Belief Network (DBN), Deep Autoencoder (DAE). The study of neural networks as well as CNNs and the use of CNNs model in image processing is an attractive and applicable problem to solve many real-world problems. The aim for this thesis is to discover what is Deep Learning purpose in Emotion Recognition model development, and by using different datasets of imagine represents basic facial emotions in humans being, the data obtained from the webcam will be identified by the Haar Cascade method from the OpenCV library, then the data is transferred to the Deep Learning network with the output probability (softmax), returning the probabilities of 7 types of emotions calculated by the system. (Janiesch & Zschech & Heinrich 2021.)

2 THEORETICAL BASICS

Face recognition is a synthetic problem. It needs important modules such as face positioning, feature extraction and classification. From there we can determine the emotion of the person in the image. Image processing has been researched and developed at a rapid pace by research centers, universities and academies. Among them, image recognition and classification is one of the areas that are actively pursued. The core idea takes from image recognition and classification is to analyse images from data obtained by image sensors such as cameras and webcams. The application of classification technology is currently developing very strongly in many fields such as: academia, business, security, health and in subjects such as social researchers, politics government and other non-profit organizations. These organizations own a large amount of unstructured data and data processing becomes much easier if this data is normalized by topics/labels. The technology foundation to perform the classification problem is Artificial Intelligence (AI) and Deep Learning (DL). (Zhang, Lipton, Li & Smola 2021.)

Along with other forms of recognition such as speech recognition, handwriting, fingerprint, retina, the problem of emotion recognition on human faces is attracting the attention of the most curious population of scientists. By knowing better about facial emotion, people can distinguish between negative and positive feedback, example in academia background, where teachers/professors can detect whether their attendees understand the on-going subject, and health care, where psychologists is able to give a quick therapy session to patients with extreme anxiety and not fond of direct eye con-tact. Emotional analysis is optimally exploited in online applications from customers' point of view as a way to collect opinions from different studies. (Kasar & Bhattacharyya & Kim 2016.)

2.1 Facial Expression Recognition

With the ubiquity of security cameras in airports, workplaces, and academics, as well as ATMs and banks, Facial Expression Recognition (FER) plays a significant role in human-machine interaction. FER can also be used in behavioural psychology research, customer care or in image-based recommendation systems. Facial expression expressed an individual's mood or emotional state at a particular time such as sad, happy, or angry. Paul Ekman shows six common emotions as Sad, Happy, Angry, Scared (Fear), Disgust and Surprised. Face detection is the first step of facial emotion recognition,

where the face is identified from the input image and other objects (if any), are removed. After the human face has been identified, the next step is to perform feature extraction and representation of those features. (Bhardwaj & Dixit 2016). With the obtained features, the final job is to classify the features into one of the six common emotions, as demonstrated in FIGURE 1. Many research have been carried out to improve the accuracy of the FER problem.

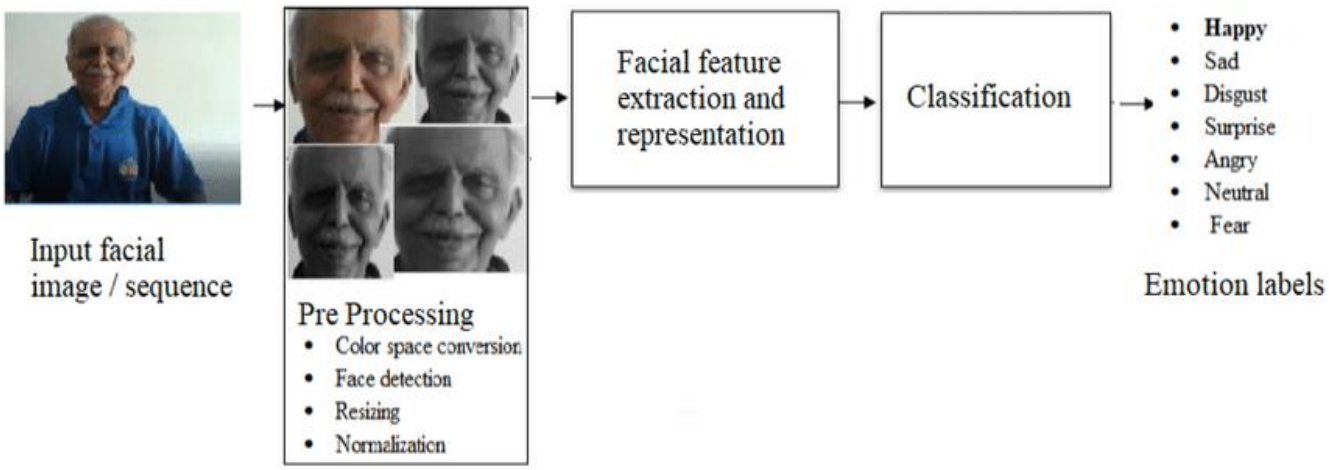

FIGURE 1: Basic steps involved in automated FER (Review of automated emotion-based quantification of facial expression in Parkinson's patients 2021)

FER has two main approaches: based on the appearance and based on the geometry. (Mistry & Goyani 2013.) The face-based approach considers the information obtained from the intensity values of pixels or the entire image by applying transformations, filters or machine learning methods, statistics, etc. While in the approach based on geometry, shape, distance, position of changes of facial components such as facial muscles, eyes, mouth, forehead... will be considered. (Kumari, Rajesh, Pooja 2015.)

Psychologist and researcher, Dr. Paul Ekman, has made substantial contributions to the field of emotion and facial expression research. He is well known for developing the Facial Action Coding System (FACS), a device for monitoring and standardizing facial emotions, in 1978. FACS was developed by analysing the relationships between muscle contractions and the changes in facial appearance caused by them. FACS is used to code facial expressions in terms of individual muscle movements, or Action Unit (AU), which can then be combined to form more complex expressions. (Rathi & Shah 2016.)

Faces can be divided Upper Face AU and Lower Face AU. The top portion of the face, encompassing the forehead, brows, and eyes, is where the Upper Face AU are situated. The Lower Face AU, on the other hand, are a group of facial muscle movements that are restricted to the area of the face below the

eyes, including the mouth, lips, and chin. When recognizing facial expressions, both the upper and lower face AUs are crucial, and a variety of expressions may be produced by combining various AUs. There were 46 AUs representing changes in facial expression and 12 AUs related to eye and head orientation. AUs are highly descriptive of facial movements, however they do not provide any representative information. AUs are labeled with descriptions of actions. The task of emotion analysis using FACS is based on the decomposition of observed expressions into a set of AUs, after which emotions are identified. (Rathi & Shah 2016.)

There have been many studies looking for methods to identify human faces, from grayscale images to colour images, which qualified necessary requirements for the need in image processing in today's modern world. (Yang, Kriegman, Ahuja 2002). Based on the properties of methods to identify human faces on images, we can divide these methods into these main approaches:

2.1.1 Knowledge-Based Methods

This is a top-down approach. It is easy to construct basic rules for describing face features and corresponding relationships. For example, a face usually has two eyes that are symmetrical about the vertical axis in the middle of the face and has a nose and a mouth. The relationships of features can be described as distance and position relationships. Usually, the authors will extract the features of the face first to get the candidates, then these candidates will be identified through rules to know which candidates are faces and which are not. A rather complicated problem using this approach is how to efficiently translate from human knowledge to laws. If these rules are too detailed, then when determining, it is possible to identify missing faces in the image, because these faces cannot satisfy all the given rules. (Yang, Kriegman, Ahuja 2002, 3-4). But the rules are too general, we may misidentify an area that is not a face but still identified it as a face. And it is also difficult to expand the requirements of the problem to identify faces with many different poses.

FIGURE 2: Original image (a) n = 1, and corresponding low-resolution images (b) n = 4, (c) n = 8, (d) n = 16 (Yang, Kriegman and Ahuja 2002.)

Yang and Huang use a method that follows this approach to identify faces (Yang & Huang 1994). The system of these two authors consists of three levels of rules. At the highest level, it uses a scan window on the image and pass a set of rules to find possible candidates for faces. At the next level, they use a set of rules to describe the general shape of the face. At the last level, another set of rules is used to examine the facial features in detail. An ordered multi-resolution system is used to determine, as in FIGURE 2. Top-level rules for finding candidates such as: “the center of the face (the darker part in FIGURE 3) has four parts with one basic evenness”, “the upper periphery of a face (the lighter part in FIGURE 3) has some basic evenness”, and “the degree of difference between the mean grey values of the median the center and the upper envelope are significant”. At level two, the system examines the histograms of candidates to eliminate candidates that are not faces, and also detect edges around the candidate. At the final level, the remaining candidates will be examined for facial features in terms of eyes and mouth. They used a strategy of "coarse to fine" or "gradual clarification" to reduce the amount of computation in processing. Although the accuracy rate is not high, this is a premise for many future studies (Yang, Kriegman, Ahuja 2002, 3-4).

FIGURE 3: A typical face kind of knowledge-based method in facial analysis research (Yang, Kriegman and Ahuja 2002.)

Kotropoulos and Pitas provide a method for use on low resolution. They used projection method to identify facial features (Kotropoulos & Pitas 1997). Kanade has succeeded with the projection method to determine the contour of the face, the functions to project the image horizontally and vertically in FIGURE 4 (Kanade 1973). Based on the horizontal projection plot, there are two local minima that tell the position of the mouth, the tip of the nose, and the eyes. These features are sufficient to identify faces (Yang, Kriegman, Ahuja 2002, 4).

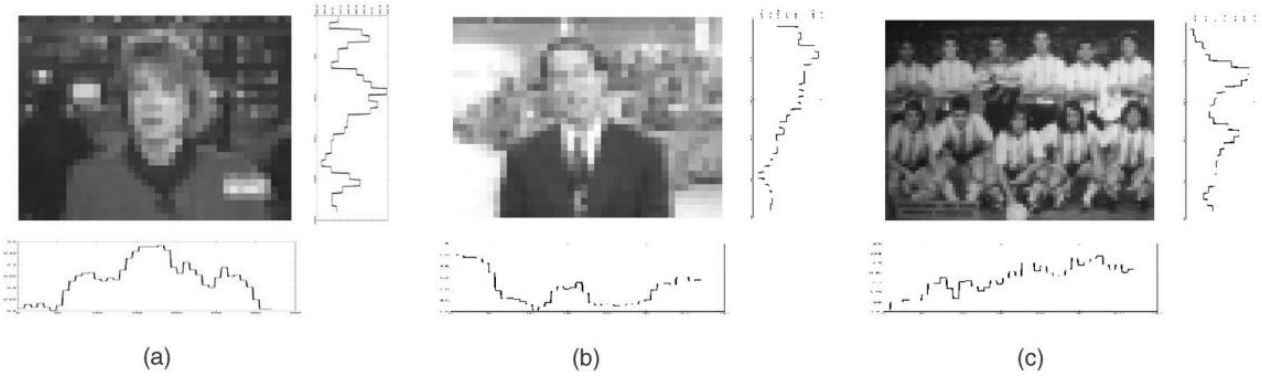

FIGURE 4: Projection Method: (a) The image has only one face and a simple background; (b) Photo with only 1 face and complex background; (c) Photos with many faces (Yang, Kriegman and Ahuja 2002.)

2.1.2 Template Matching

In Template Matching, standard samples of faces, usually straight-faced faces, are predefined or parameters defined through a function. From an input image, the system calculates correlation values against standard samples for facial contour, eyes, nose, and mouth. Through these correlation values, the authors decide whether or not there is a face in the image. This approach has the advantage of being very easy to set up, but does not work when there are changes in proportions, postures and shapes (Yang, Kriegman, Ahuja 2002).

Sinha uses a small set of image invariants in the image space to describe the space of image samples. His main idea is based on the variation of brightness levels of different areas of the face, such as eyes, cheeks, and forehead, the relationship of brightness levels of the remaining areas does not change significantly (Sinha et al. 2006). Determining the ratio pairs of the brightness levels of some region (a region darker or lighter) gives a fairly efficient amount of invariance. Areas of uniform luminance are considered as a proportional sample but rather a rough sample in the image space of a face with less suitable fit for selection such as major face features such as eyes, cheeks, and forehead. The system stores the brightness changes of facial regions in an appropriate set with lighter-darker relationship pairs between sub-regions. A face is determined when an image matches all the lighter – darker pairs. (Yang, Kriegman, Ahuja 2002, 4.)

This idea comes from the difference of intensity between local adjacencies, which is later extended on the basis of Wavelet Transform to represent pedestrian identification, car identification, and face detection (Papageorgiou & Poggio 2000). FIGURE 5 shows the prominent pattern in the 23 defined relationships. Using these relations for classification, there are 11 essential relations (black arrows) and 12 valid relations (gray arrows). Each arrow is a relation. A relationship that satisfies the face pattern when the ratio between two regions exceeds a threshold and these relationships exceed the threshold and is considered to identify a face. The ordinal pattern matching method for identifying human faces is presented by Miao. In the first stage, the image will be rotated from -20^0 to 2^0 in steps of 5^0 and in order. (Miao, Yin, Wang, Shen & Chen 1999). Build a multi-resolution image, then use Laplace math to determine the edges. An edge face pattern depicts six components: two eyebrows, two eyes, a nose, and a mouth.

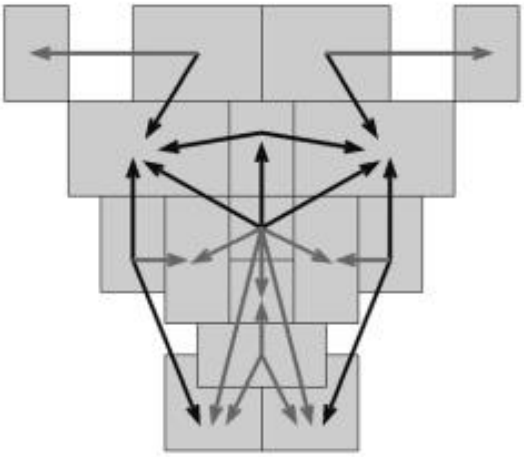

FIGURE 5: Face template, with 16 regions and 23 relationships (arrows), based on Sinha method (Yang, Kriegman and Ahuja 2002.)

2.1.3 Appearance-Based Method

The patterns in this method are learnt from example photos, as opposed to approaches that match patterns with predetermined patterns by experts. These methods often use statistical, probabilistic, and machine learning methods to identify pertinent aspects in both faces and non-faces. These traits may be used to recognize human faces using learnt features such as discriminant functions or distribution models. The issue of lowering the number of dimensions is frequently motivated by the need to improve both the computational and decision-making efficiency. (Yang, Kriegman, Ahuja 2002, 9-10.)

An image or a feature vector derived from an image is treated as a random variable x, and the random variable is characterized as a face or a non-face by the formula calculated according to the conditional classification density functions: p(x | face) and p(x | not face). Bayesian or maximum likelihood classifiers can be used to classify a candidate as a face or a non-face. It is not possible to directly implement the Bayes classifier because the dimensionality of x is quite high, because p(x | faces) and p(x | not faces) are polynomials and cannot be understood by constructing parameter forms or digitizing naturally for p(x | faces) and p(x | not faces). There are quite a few studies in this direction that are interested in parametric or non-parametric approximations for p(x | faces) and p(x | not faces). (Yang, Kriegman, Ahuja 2002, 3-4.)

Another approach in the face-based approach is to find a discriminant function, such as decision plane, hyperplane for data separation, threshold function, to distinguish two classes of data: faces and non-face templates. Image samples are projected into a lower dimensional space, and then either a discriminant function, based on distance measures, is used to classify, or construct a nonlinear decision surface using a multilayer grating neural network. (Rowley et al. 1996.)

2.2 Deep Learning

Machine learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce sites, and it is increasingly present in consumer goods such as cameras and smartphones. Machine learning is used to identify objects in images, convert speech to text, match news items, posts, or products with user interests, and select relevant search results. (Fu & Menzies 2017.)

Deep learning is a branch of machine learning based on a set of algorithms that attempt to model highly abstracted data using multiple processing layers with complex structures, or otherwise including many nonlinear transformations. Deep learning is part of a broader family of machine learning methods based on the learning representation of data. An observation (for example, an image) can be represented in many ways as a vector of intensity values for each pixel, or more abstractly as a set of edges, specific shape areas. One of the promises of deep learning is to replace manual features with algorithms that are efficient for unsupervised or semi-supervised learning and hierarchical features. (Bengio 2009.)

Studies in this area attempt to make better representations and create models to learn these representations from large-scale unlabelled data. Some representations are inspired by advances in neuroscience and are based on interpretations of patterns of information processing and communication in a nervous system, such as neural coding to attempt in determining the relationships between different stimuli and related neural responses in the brain. (Bengio 2009.)

2.2.1 Artificial Neural Network

An Artificial Neural Network (ANN) is an information processing model that mimics the way biological neural systems process information. It is fabricated of a large number of elements (neurons) connected

to each other through links that work as a whole to solve a particular problem. ANN is like the human brain, learning by experience through training, has the ability to store knowledge experiences and use that knowledge in predicting unknown data. ANN was introduced in 1943 by neuroscientist Warren McCulloch and logician Walter Pits. (Dongare, Kharde, Kachare 2012.)

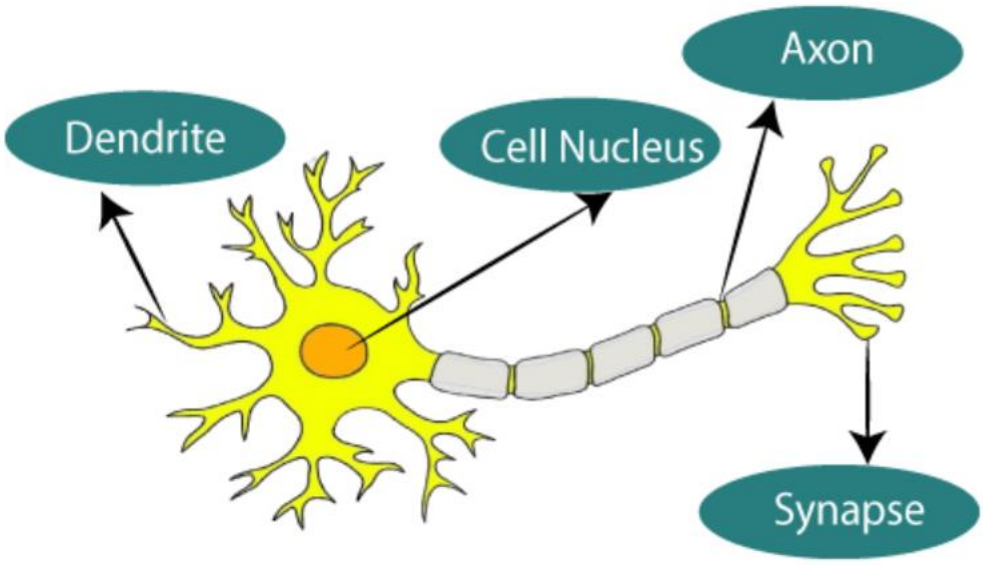

FIGURE 6: Structure of a biological neuron (JavaTpoint 2023.)

An ANN consists of three input layers, hidden layers and finally an output layer as shown in FIGURE 7, in which, the hidden layer consists of many neurons that receive input data from the previous layers to process and convert this data for the next layers. An ANN can have multiple hidden layers or no hidden layers. Each node in the network is called a neuron. Each neuron that receives the input data processes them and returns a unique result. The output of one neuron can be the input of other neurons. Weights are assigned once the data in the input layer has been specified. These weights assist in determining the significance of any given variable, with bigger variables contributing more heavily to the output than smaller variables. All inputs are multiplied by their weights and then added together. If the output reaches a specific threshold, the node is activated and the data is sent to the next tier of the network. As a result, one node's output becomes the next node's input. Most artificial neural networks are feedforward, meaning they flow from input to output in just one way. They may, however, be taught for backpropagation, such as moving from the output to the input. (Kasar, Bhattacharyya & Kim 2016.)

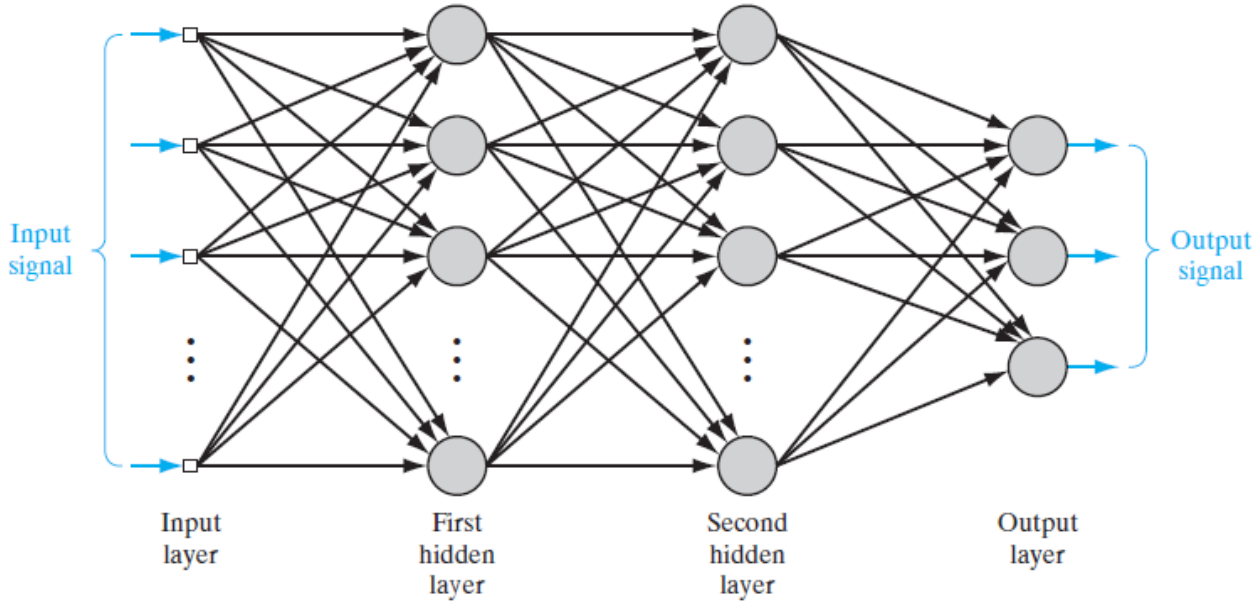

FIGURE 7: The structure of the artificial neural network. (Artificial Neural Network 2019)

Similar to biological neurons, each input to an artificial neuron has a different effect on the neuron's output. This is settled by the coefficients assigned to each input-w_i: the weight of the i-th input. The value of w_i can be positive or negative similar to having two types of coupling in a biological neural network. If w_i is positive, it is equivalent to excitatory coupling and if w_i is negative, it is equivalent to inhibitory coupling, as can be seen from FIGURE 8. The neuron body will be responsible for synthesizing input signals for processing to give an output signal of the neuron. This processing and calculation process will be discussed in detail in the following section. The output of the artificial neuron is similar to the axon of the biological neuron. The output signal can also be split into many branches in a tree structure to feed to the input of other neurons. (Agarwal 2020.)

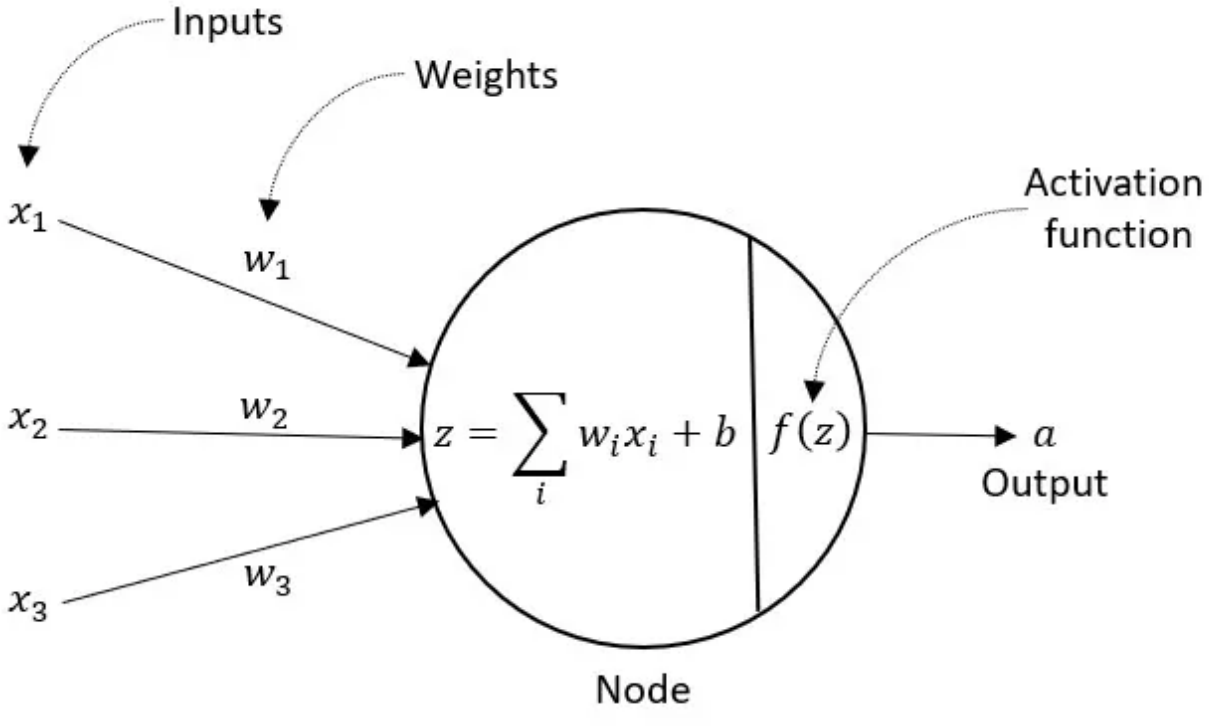

FIGURE 8: The processing of a neuron in an ANN (Agarwal 2020.)

2.2.2 Deep Learning Method

How the Deep Learning algorithm works is as following; the information flows will be passed through many layers until the last layer and takes the human learning process as a concrete example. The first grades will focus on learning more concrete concepts while the deeper layers will use the learned information for further research and analysis in abstract concepts. The process of building this representation of data is called feature extraction. The complex architecture of deep learning is fed from a deep neural network with the ability to perform automatic feature extraction. (Zhang, Lipton, Mu, Smola 2021.)

Deep Learning algorithms identify dogs in a manner akin to a child. Each algorithm in the order of operation modifies its input in a nonlinear way before producing a statistical model as its output. Until the output is accurate enough to be relied upon, iteration continues. The deep text's inspiration came from the several levels of processing that the data must have been through. In conventional machine learning, the learning process is supervised, and the programmer must be very explicit when instructing the computer what sort of item it has to search for. In order to determine if a picture contains a dog or not, the computer's success rate completely depends on the programmer's ability to identify the right feature set for the dog during this time-consuming procedure known as feature extraction. Deep Learning

has the benefit of allowing the feature builder software to be built up independently and without supervision (Zhang, Lip-ton, Mu, and Smola, 2021). Unsupervised learning is not only more rapid, but it is also more precise.

Initially, training data, a collection of photographs for which humans have assigned meta tags to each dog and non-dog-related image, might be provided to the computer program. The application develops a feature set for the dog and a prediction model using the data from the training set. In this instance, the computer-generated model was able to predict, for the first time that anything in the image with four legs and a tail would be classified as a dog. Naturally, the computer does not recognize labels for quarupeds or tails. It merely scans digital data for pixel patterns. The prediction model grows increasingly sophisticated and precise with each iteration. A computer software utilizing deep learning techniques may be provided a set of training and, after sorting through millions of photos, decide precisely which one has a dog. This is in contrast to infants, who need weeks or even months to acquire the notion of dogs. (Zhang, Lipton, Mu, Smola, 2021.)

To achieve an acceptable level of accuracy, Deep Learning programs require access to a significant amounts of training data and processing power, both of which are not easily available to coming-of-age programmers to the modern world of big data and cloud computing. Because Deep Learning programming can generate complex statistical models directly from its own iterative output, it can generate accurate predictive models from large amounts of data unlabelled and unstructured. This is important as the Internet of Things (IoT) continues to become more pervasive as most of the data that humans and machines generate is unstructured and unlabelled. (Zhang, Lipton, Mu, Smola 2021.)

2.2.3 Real Life Application

Because Deep Learning models process information in similar ways to the human brain, they can be applied to many tasks that humans perform. Deep learning is currently being used in most popular image recognition engines, Natural Language Processing (NLP) and speech recognition software. These tools are starting to appear in applications as diverse as self-driving cars and language translation services. Today's use cases for deep learning include all kinds of big data analytics applications, especially those focused on NLP, language translation, medical diagnostics, market trading signals securities, network security, and image recognition. (Zhang, Lipton, Mu, Smola 2021.)

The most popular application of Deep Learning today is the virtual assistant from Alexa to Siri, Google Assistant. Each interaction with these assistants provides them with the opportunity to learn more about users' voice and intonation, thereby providing user with a second-person, interactive experience. Virtual assistants use Deep Learning to learn more about their topics, from dinner preferences to most visited spots or favourite songs. They learn to understand basic commands by evaluating natural human language to carry them out. With deep learning applications like text generation and document summaries, virtual assistants can assist users in creating or sending the right email copy. (Zhang, Lipton, Mu, Smola 2021.)

2.3 Convolutional Neural Networks

Convolutional Neural Network (CNN) is one of the advanced deep learning models that helps researchers build intelligent systems with high accuracy today. This model has been developed and is being applied to large image processing systems of Facebook, Google or Amazon, for various purposes such as automatic tagging algorithms, image search or product recommendations for consumers. Convolution was first used in digital Signal Processing. Thanks to the principle of information transformation, scientists have applied this technique to digital image and video processing. (Szegedy et al. 2015.)

2.3.1 CNN Architect

The neural network model was born and has been widely applied to recognition problems. However, for image data, the feedforward neural network does not perform very well. The image data is quite large, a 32x32 pixel grey image will produce a feature vector of 1024 dimensions, for a colour image of the same size there will be 3072 dimensions. (Simonyan & Zisserman 2014.)

This also means that 3072 weights θ are needed between the input layer and a node in the next hidden layer. The number of weights will be more replicated if the number of nodes in the hidden layer increases and the number of hidden layers increases. Thus, with only a small 32x32 image, a rather massive feedforward neural network model is needed, which makes manipulating larger images quite difficult. Based on this idea, the convolutional neural network was born with a different structure from the

feedforward neural network. Instead of the entire image being directly connected to a node, only a local part of the image is connected to a node in the next layer. The original image data through the layers of the convolutional neural network model will learn the features to perform effective classification. (Simonyan & Zisserman 2014.)

Basically, CNN includes the following layers: convolutional layer, activation layer, pooling layer (also known as subsampling layer), and fully-connected layer. In FIGURE 9, the layers are linked together through a convolution mechanism. The next layer is the result of the convolution of the previous layer, so it is able to get local connections. Each neuron in the next layer is generated from the filters applied to a local image area of the neuron in the previous layer. Each such layer is subjected to different filters, usually several hundred to several thousand such filters. Other classes such as pooling/subsampling are used to extract more useful information. During training, the convolutional neural network automatically learns the parameters for the filters. As in image classification, CNN will try to find the optimal parameters for the corresponding filters in order of original pixel > edge > shape> face > featured high level. The last layer is often used to classify images. (Simonyan & Zisserman 2014.)

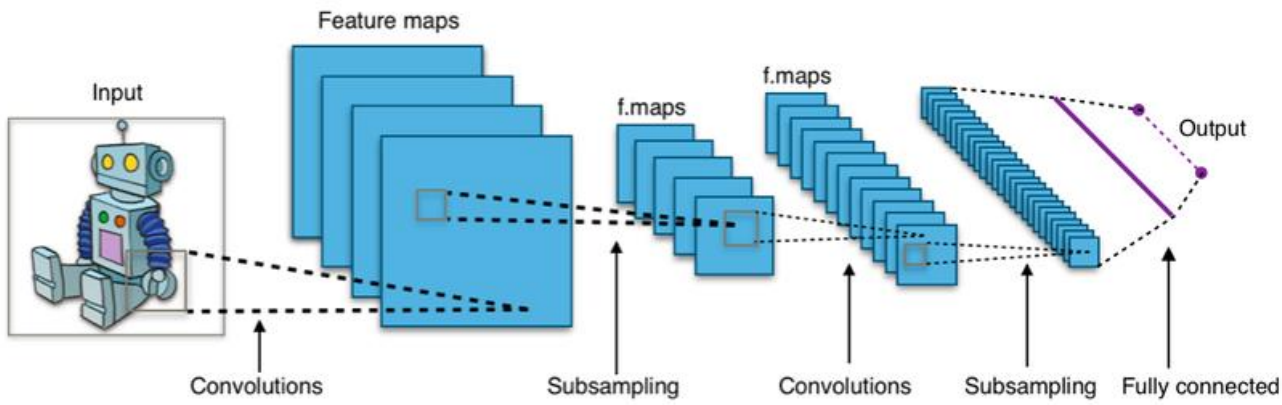

FIGURE 9: Model of basic layers of CNN.

Convolution is the most important component in the CNN network, which also represents the idea of building a local connection instead of connecting all the pixels. These local links are calculated by convolution between pixel values in a local image region with filters of small size. (Wu 2017.)

In the FIGURE 10, it can be seen that the filter used is a matrix of size 3x3. This filter is shifted through each image area in turn until it completes scanning the entire image, creating a new image that is less than or equal to the input size. Thus, after giving an input image to the convolution layer, the output will be a series of images corresponding to the filters used to perform the convolution. The weights of these filters are randomly initialized for the first time and will improve gradually throughout the training process. (Wu 2017.)

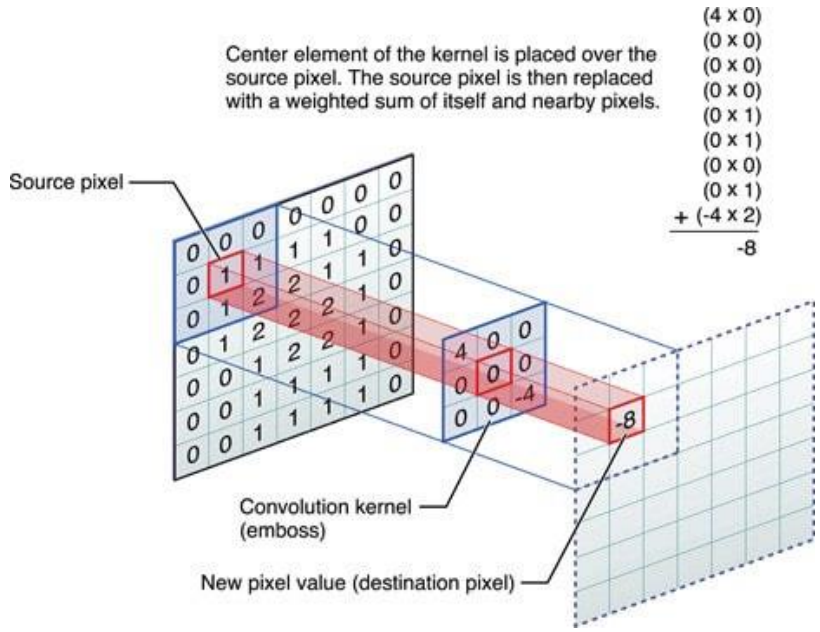

FIGURE 10: Illustration of convolution on an image matrix (MRIquestions 2023.)

The Rectified Linear Unit (ReLU) layer is built with the intention of ensuring the nonlinearity of the training model after performing a series of linear calculations through the convolution layers. Among the most popular activation functions such as tank, sigmoid. The ReLU function was chosen due to its simple installation, fast processing speed, and efficient computation. Specifically, the calculation of the ReLU function simply converts all negative values to zero. Typically, this layer is applied directly behind the convolution layer, with the output being a new sized image. Similar to the input image, the pixel values are exactly the same except the negative values have been removed. (Wu 2017.)

Another major computational component in a CNN network is pooling, which is usually placed after the convolution layer and the ReLU layer to reduce the size of the output image while preserving the important information of the input image. There are two popular sampling methods today, sampling the maximum pixel value (Max Pooling) and sampling the average value of pixels in the local image area

(Avarage Pooling). Thus, for each input image that is put through sampling, a corresponding output image will be obtained, whose size is significantly reduced but still retains the necessary features for later computation. (Wu 2017.)

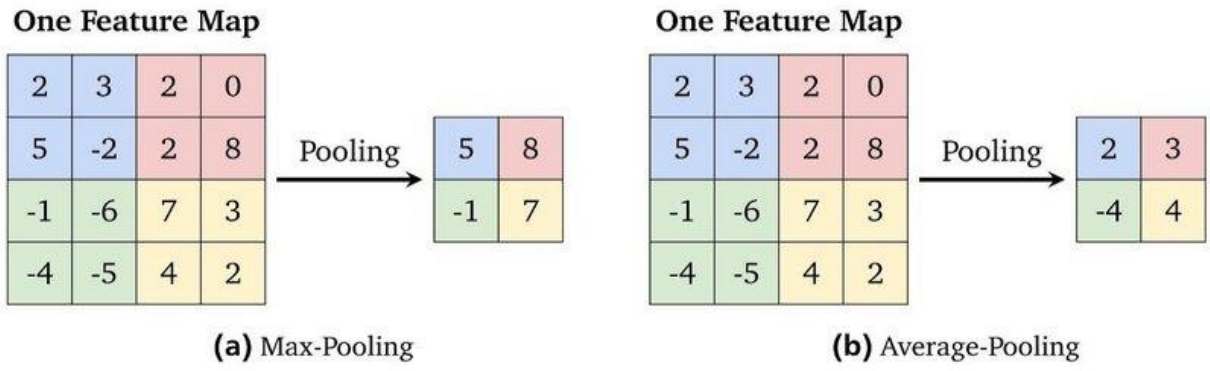

FIGURE 11: Average Pooling and Max Pooling Methods (Guissous 2019.)

This fully-connected layer is completely similar to the traditional neural network, that means all pixels are fully connected to the node in the next layer. Compared with traditional neural networks, the input images of this layer have been greatly reduced in size, while still ensuring important information for identification. Therefore, the recognition calculation using the feedforward model is no longer complicated and time consuming as in the traditional neural network. (Wu 2017.)

2.3.2 Typical CNN Models

The CNN network model has a long history. The original architecture of the CNN network model was introduced by a Japanese computer scientist Kunihiko Fukushima in 1980. (Fukushima 1980). Then, in 1998, Yan LeCun first trained the CNN model with the backpropagation algorithm for the handwriting recognition problem. (LeCun, Bottou, Bengio & Haffner 1998). In 2012, a Ukrainian computer scientist Alex Krizhevsky built a CNN (AlexNet) model and used GPUs to speed up the training process of deep nets to achieve the top 1 in the annual ImageNet Computer Vision competition with the highest accuracy. The top 5 classification error is reduced by more than 10% compared to the previous traditional models, which has created a strong wave of using the deep learning CNN model with GPU support to solve more and more problem in Computer Vision. (Krizhevsky, Sutskever, Hinton 2012.)

LeNet-5 is one of the most famous old CNN networks developed by Yann LeCun in the 1998s. The structure of LeNet-5 consists of 2 layers (convolution + maxpooling) and 2 layers fully connected layer and the output is softmax layer. Although simple, it has better results than other traditional machine learning algorithms in handwritten digit classification. (LeCun et al. 1998.)

In the first neural network architecture, to reduce the data dimensionality, Yan LeCun uses the Sub-Sampling Layer which is an Average-Pooling Layer, that meant layers aim to reduce the data dimensionality without changing the features we call it the Sub-Sampling Layer. This architecture is difficult to converge, so today they are replaced by Max Pooling. The input of the LeNet-5 network has a small size (only 32x32) and few layers, so its number of parameters is only about 60 thousand. (LeCun et al. 1998.)

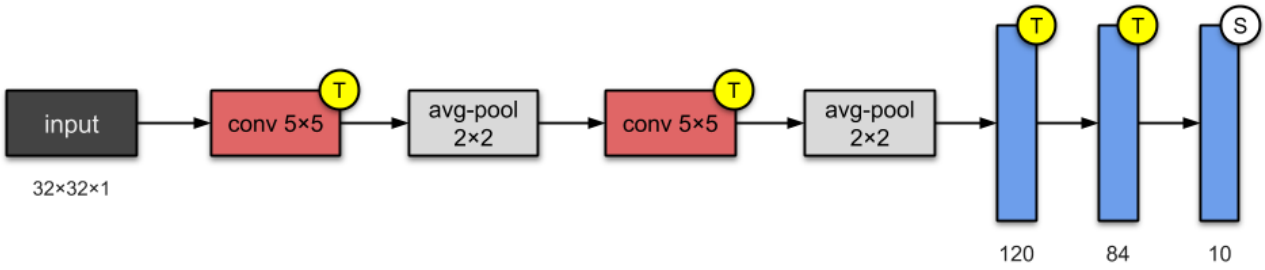

FIGURE 12: LeNet-5 Architecture (LeCun, Bottou, Bengio & Haffner 1998.)

The AlexNet Network is a CNN network that won the ImageNet LSVRC-2012 competition in 2012 developed by the team of authors Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. AlexNet is a training CNN with a very large number of parameters (60 million parameters) compared to LeNet. (Krizhevsky et al. 2012.)

AlexNet uses relu instead of sigmoid, or tanh, to deal with non-linearity, which increase calculation speed by 6 times. By using dropout as a new regularization method for CNNs, which not only helps the model avoid overfitting but also reduces the time to train the model. The model overlaps pooling to reduce the size of the network, which traditionally pooling regions do not overlap. By using local response normalization to normalize on each layer and use data augmentation technique to create more data training by translations orizontal reflections. AlexNet training with 90 epochs for 5 to 6 days with

2 GTX 580 GPUs and using SGD with learning rate 0.01, momentum 0.9 and weight decay 0.0005. (Krizhevsky et al. 2012.)

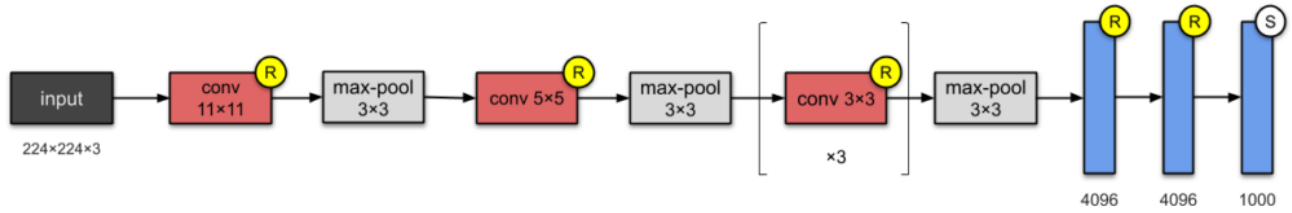

FIGURE 13: AlexNet Architecture (Krizhevsky, Sutskever, Hinton 2012,)

ZFNet (stands for Zeiler and Fergus) is a CNN network architecture model developed by Matthew Zeiler and Rob Fergus. The ZFNet network has an architecture that won the competition on ImageNet 2013. (ILSVRC 2013.) ZFNet has a very similar structure to AlexNet with 5 convolution layers, 2 fully connected layers and 1 output softmax layer. The difference is the kernel size in each Conv layer. Similar to AlexNet but with some minor adjustment, ZFNet training only has 1.3m images. ZFNet uses the 7x7 kernel on the first layer and the reason is to use a smaller kernel to retain more information on the image. The creators increase the number of filters more than AlexNet and training on GTX 580 GPU for 20 days. (Zeiler & Fergus 2013.)

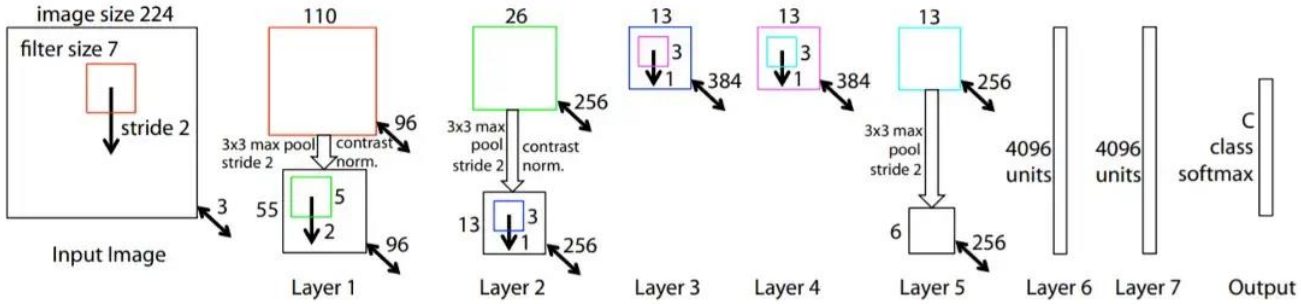

FIGURE 14: ZFNet Architecture (Verma 2020.)

VGGNet is the CNN network that won the second prize of the contest on ImageNet 2014 (ILSVRC 2014). The VGGNet network was researched and proposed by Karen Simonyan and Andrew Zisserman. VGGNet network is considered the best network in 2015, which consists of 16 convolutional lay-

ers and has a very unified architecture. The network uses 3x3 convolutional filters and 2x2 pooling filters, which are uniform in size from the top to bottom layers of the network. (Simonyan & Zisserman 2014.)

VGGNet network is deeper than AlexNet, and the number of parameters of the network is up to 138 million parameters. This is one of the networks that has the largest number of parameters. In addition, VGGNet has another version, VGG-19, which adds 3 layers of depth. Starting from VGGNet, a common pattern for CNN networks in supervised learning tasks in image processing has begun to take shape, which is that the networks become deeper and use blocks of the form [Conv2D*n + Max. Pooling]. (Simonyan & Zisserman 2014.)

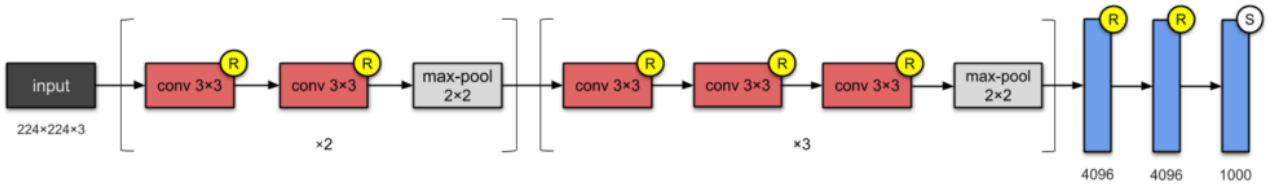

FIGURE 15: VGG-16 Architecture (Simonyan & Zisserman 2014.)

The GoogleNet network is rated as the best deep learning network in 2014, the network with the winning architecture of the contest on ImageNet 2014 (ILSVRC 2014). This network is proposed by the research group of C. Szegedy from Google and its architecture consists of 22 layers deep. The network reduces the number of parameters from 60 million of the AlexNet network to 4 million. (Szegedy et al. 2015.)

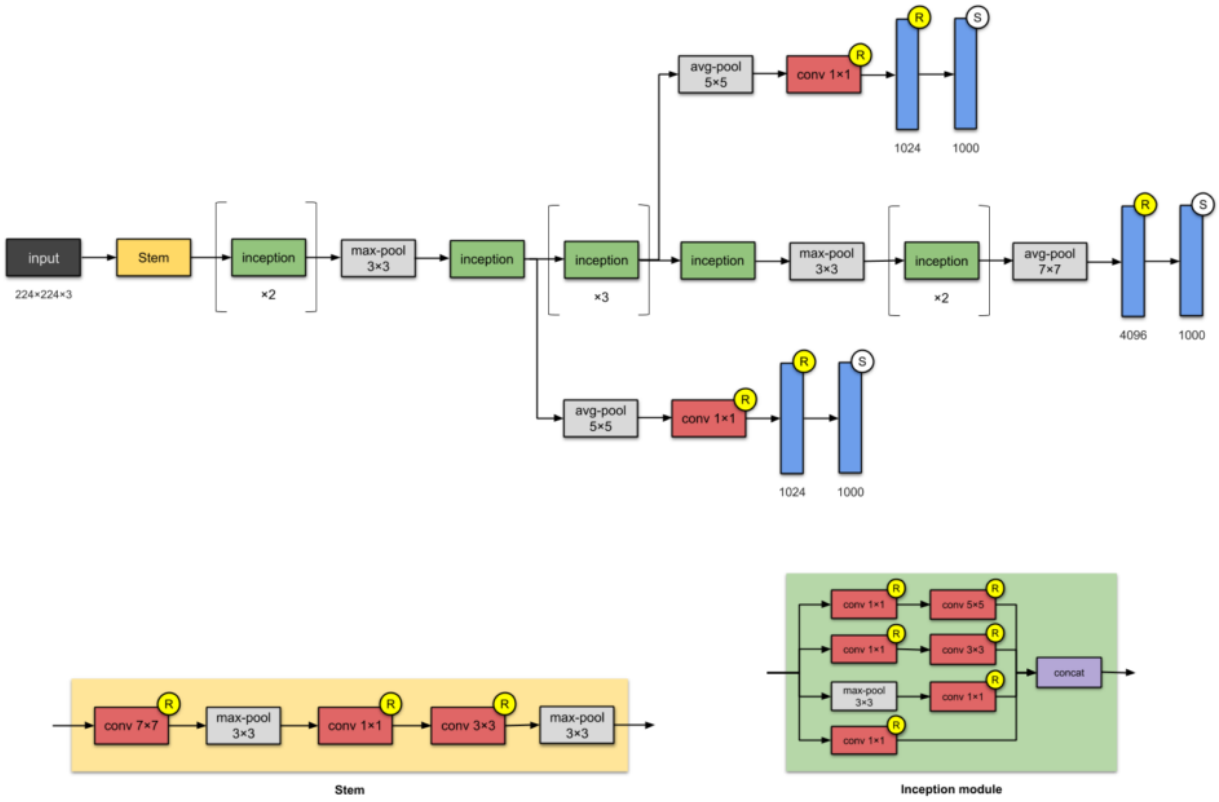

FIGURE 16: InceptionV1 Architecture (Szegedy et al. 2015.)

3 RELATED TECHNOLOGY

Deep Learning is a part of Machine Learning, which is based on the use of complex neural networks. Deep Learning uses this artificial neural network to analyse data, make predictions, and represent situations based on analysis and "experience" from learned patterns, in the same way that people perceive an object or phenomenon that they have experienced. (Janiesch, Zschech, Heinrich 2021). The Python programming language is known for its simplicity, easy-to-learn, easy-to-read code, logical and concise syntax, while Machine Learning involves extremely complex algorithms and multi-stage workflows. The concise and easy logic of Python plays an important role in saving time while to gain the advantage of offering better solutions. (Ketkar & Moolayil 2021.)

3.1 Python Programming Language

In 1980, at the Center for Mathematics - Informatics (Centrum Wiskunde & Informatica) in the Netherlands, Guido van Rossum first studied the Python programming language. After 9 years of research and preparation, in 1989, Python began to be deployed. Python is not named after the snake god Python in Greek mythology. Rossum was a fan of a late-1970s comedy series, and the name "Python" was taken from the title of a part in that series "Monty Python's Flying Circus". This language was originally developed on the basis of Unix. However, to keep up with the development of the times, the Python language has been promoted and expanded to many different operating systems such as MS Dos and Linux. (LearnPython.com 2022.)

Python is one of the top 10 programming languages in the world today. This language is trending strongly with a community of learners and workers. Google, Facebook, Instagram are currently using Python to program their large and small projects. Python has a very simple, clean syntax. It is much easier to read and write when compared to other programming languages like C++, Java, C#. Since Python is an open-source application, developers can not only use software and programs written in Python, but also change its source code. Anything that cannot be solved in other programming languages is easily solved in Python. (LearnPython.com 2022.)

3.2 TensorFlow

TensorFlow is an open-source end-to-end library created primarily for Machine Learning applications. It is a symbolic math library that uses discriminable programming and data streams to perform various tasks, with a focus on training and inferring deep neural networks. It allows developers to create machine learning applications using various tools, libraries, and community resources. TensorFlow was created and developed by a team of Google researchers (Google Brain) and officially licensed to operate on November 9, 2015 for the purpose of supporting research and application in effective production. (Yegulalp 2022.)

TensorFlow's name is directly derived from its core framework: Tensor. In TensorFlow, all calculations involve tensors. A tensor is a vector or matrix of n dimensions representing all data types. All values in a tensor hold the identical data type with a known, or partially known, shape. The shape of the data is the size of the matrix or array (Roman 2020). TensorFlow allows developers to build graphs and data flow structures that define how data moves through the graph by taking as input a multidimensional array, which allows users to build a diagram of the operations that can be performed on these inputs, going at one end and arriving at the other as outputs. (Yegulalp 2022.)

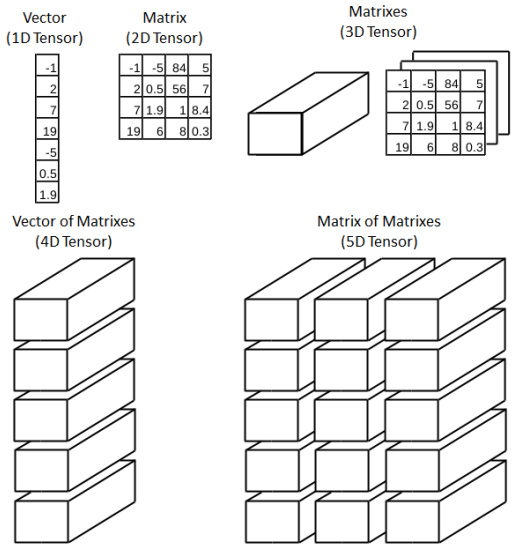

FIGURE 17: Tensor Architecture (Roman 2020.)

Every TensorFlow application is itself a Python application and is written in high performance C++ binaries. TensorFlow will rely on the Python language to provide all of the above for programmers because Python is easy to learn and helps understand how to make high-level abstractions composable. (Yegulalp 2022.)

3.3 Keras

Keras is an open-source for Neural Network written in Python language. It is a library developed in 2005 by Francois Chollet, a Deep Learning research engineer. Keras is a high-level API designed for Python to make implementing neural networks easier. Keras can run on libraries and frameworks like TensorFlow and Theano. They are both very powerful but also confusing libraries for creating neural networks. On the other hand, Keras is very beginner-friendly as its minimal structure provides a clean and easy way to generate deep learning models based on TensorFlow or Theano. (Chollet 2018.)

Keras has been adopted by TensorFlow as its official high-level API. When embedded in TensorFlow, it makes available modules for all neural network computations and thus can perform deep learning very quickly. TensorFlow is very flexible and the main benefit is distributed computing, so developers can be flexible and can control their application, execute the ideas in short time, using Keras, while computing involves tensors, compute graphs, and sessions, which can be customized using Tensorflow Core API (Chollet 2018).

3.4 OpenCV

OpenCV is an abbreviation of the phrase “Open-Source Computer Vision Library” – an open-source library for Machine Learning and Computer Vision. OpenCV is created for real-time analysis and machine learning as well as image and video processing. Currently, the OpenCV toolkit is also added with real-time GPU acceleration. The OpenCV project was born in 1999, created by Gary Bradski from Intel. In 2000, the first version of OpenCV was released. OpenCV is released under the BSD (Berkeley Software Distribution) license, so it is completely free for both academic and commercial use. It has C++, C, Python, Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV is designed for efficient computing and with a heavy focus on real-time applications. The advantage of

OpenCV is that it provides over 2,500 classical and modern algorithms, all optimized for machine learning and computer vision. And because it was written with optimized C/C++, the library can take advantage of multi-core processing. The main object of OpenCV is real-time applications. (Bradski & Kaehler 2008.)

3.5 NumPy and SciPy

NumPy stands for Numerical Python. NumPy is a math library used to work with Matrices and Arrays and commonly used to deal with large arrays and matrix data. NumPy was created in 2005 by Travis Oliphant and its processing speed is many times faster than using regular Python arrays and lists. NumPy provides objects and methods for working with multidimensional arrays and linear algebra operations. In NumPy, the dimension of the array is called axes while the dimension is called rank. The main library in NumPy is array objects. Arrays are similar to lists in Python, provided that every element in the array must have the same data type. Arrays can work with large amounts of numeric data, usually floats or ints, and are much more efficient on lists. The commonly used class in NumPy is Ndarray (N-dimentional array). (The NumPy Community 2022.)

SciPy stands for Science Python and is a free, open-source Python science library for math, science, and engineering. The SciPy library is built on top of the NumPy library, providing convenient and fast N-dimensional array manipulation. SciPy includes submodules for linear algebra, optimization, integration, and statistics. SciPy is the base library and it is built on top of the NumPy extension. SciPy is compatible with NumPy's N-dimensional array object. SciPy includes code for the operation of NumPy functions. SciPy and NumPy together are the best choice for scientific operations. (The SciPy Community 2013.)

4 IMPLETION OF THE PROJECT

By using Convolutional Neural Network (CNN) Recognition model, the model of Facial Emotion Recognition has been developed to distinguish human's basic emotions and made this idea the main subject of this project. Dataset FER2013 from Kaggle is necessary and the system is building with Python programming language and OpenCV libraries, and with the use of computer's camera to determine the desired result based on the dataset. Facial Emotion Recognition folder was created inside and the project was completely executed in Jupyter Notebook. As for dataset folder for emotion recognition training, seven imagine datasets have been created to be used in the model, which can be seen from FIGURE 18.

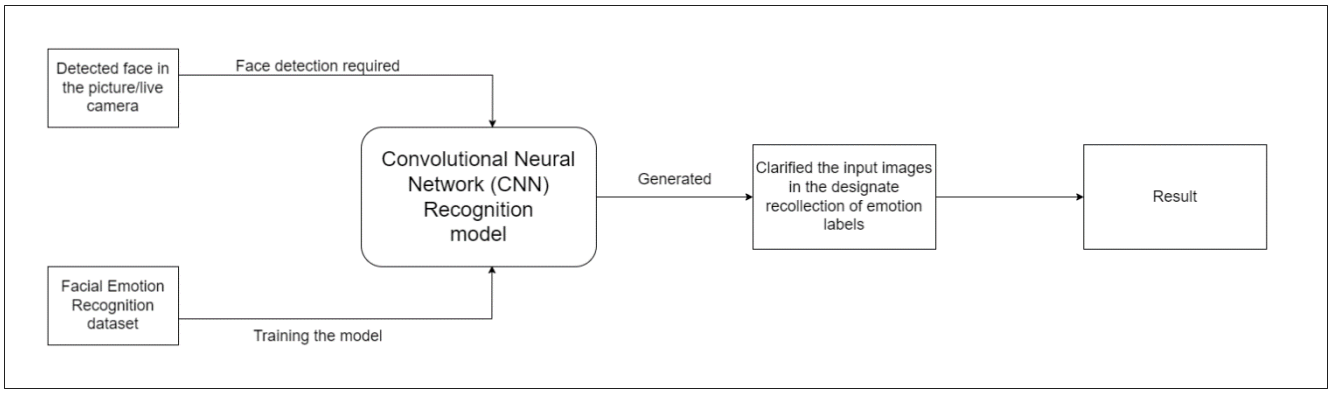

FIGURE 18: Implementation model

4.1 Preparing the dataset

FER2013 is a database of facial emotions provided by the Kaggle website for the teams participating in the facial emotion recognition challenge competition. Image data is a collection of grayscale images with size 48x48 pixels. The images in FER2013 are also labelled each face based on the emotion expressed in facial expressions in 1 of 7 emotion categories: 0 = Anger, 1 = Disgust, 2 = Fear, Happiness, 4 = Deal, 5 = Surprise, 6 = Neutral/Normal (Kaggle 2023).

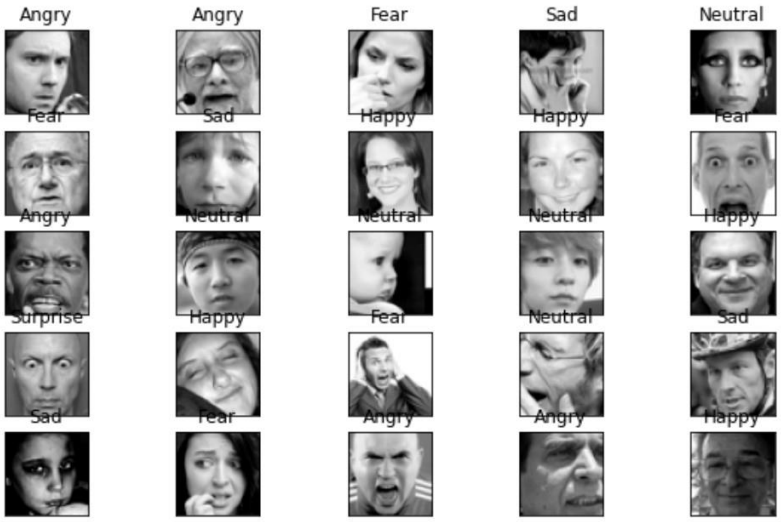

FIGURE 19: Some sample images from FER2013 that could be confused for the (Kaggle 2023.)

The face in the photo is not completely straight and in the center of the image, but carefully designed to increase the difficulty of the recognition results (Kaggle 2023). Another obstacle of the FER2013 photo album is that the facial images do not show clear emotions but are sometimes obscured by other expressions such as hands covering the mouth, forehead, wearing a hat, scarf, glasses, which can be seen in FIGURE 19.

4.2 Importing Libraries

Python is a popular computer language for recognizing and detecting faces. In the project path, it is also an important language for creating real-time face expression recognition because of its adaptability and simplicity. Using libraries like OpenCV, availability of strong Machine Learning and Deep Learning tools like TensorFlow and NumPy as necessary libraries for the project to have a smooth training and testing process. FIGURE 20 shows the imported Python libraries and tool, along with serval modules and functions for both of FER_train.ipynb and EmotiojnDetection.ipynb files from Jupyter Notebook.

In [1]:
```
import pandas as pd  # For reading and manipulating data
import numpy as np # For numerical operations
import os # For interacting with the operating system
import matplotlib.pyplot as plt # For data visualization
import seaborn as sns # For data visualization
%matplotlib inline
```

In [19]:
```
from tensorflow.keras.layers import Input, Dense, Dropout, Flatten, Conv2D, BatchNormalization, Convolution2D, Activation, MaxPooling2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
```

In [1]:
```
import os
import cv2
import numpy as np
from tensorflow.keras.models import load_model, model_from_json
from tensorflow.keras.preprocessing.image import load_img,img_to_array
from tensorflow.keras.preprocessing import image
```

FIGURE 20: Imported libraries for the training and testing process.

4.3 Data Visualisation

In FIGURE 21, after building a dictionary that converts numeric labels into textual labels for emotions, the code extracts the count of each individual value in the ‘emotion’ column of the ‘data’ dataframe and stores the results in a new dataframe. Column names in the new dataframe are now ‘emotion’ and ‘number’ respectively. The ‘emotion’ column of the new dataframe's numeric labels are converted to the assigned corresponding textual labels using the previously created dictionary. The updated dataframe is subsequently generated, displaying the quantity of images and each emotion's corresponding string name.

In [5]:
```
emotion_map = {0: 'Angry', 1: 'Digust', 2: 'Fear', 3: 'Happy', 4: 'Sad', 5: 'Surprise', 6: 'Neutral'}
emotion_counts = data['emotion'].value_counts(sort=False).reset_index()
emotion_counts.columns = ['emotion', 'number']
emotion_counts['emotion'] = emotion_counts['emotion'].map(emotion_map)
emotion_counts
```

Out[5]:

	emotion	number
0	Angry	4953
1	Fear	5121
2	Sad	6077
3	Neutral	6198
4	Happy	8989
5	Surprise	4002
6	Digust	547

FIGURE 21: The distribution of emotions in the dataset.

The application reads data from “fer2013.csv” file that contains labels and pixel values for facial image data. After processing the pixel values, the 2D array of picture data is plotted as a 3x3 grid of grayscale images using “matplotlib”, confirmed that the data is being received and processed correctly. The code set up a loop separated the string of pixel values into separate values and turns them into integers, goes back and forth between each row of pixel values and the row of the photo array is updated with the pixel values.

4.4 Pre-processing Data

The code from FIGURE 22 is used to build a new array called “reshaped_images” with the dimensions (num_images, 48, 48, 1) by reshaping the original pictures array to only include one channel. The code changed the images by dividing each pixel value by 255.0, which ensured that all pixel values fall between the range of 0 and 1. The code improved the model's accuracy and showed a sample picture that has been standardized and reshaped to have the dimensions (48, 48) from the “norm_images” array. FIGURE 22 visualizes one of the pre-processed images, which is 7th imagine in “norm_images”.

```
In [13]:
reshaped_images = np.zeros((images.shape[0], 48, 48,1))
i = 0
for image in images:
  reshaped_images[i] = image.reshape((48,48,1))
  i+=1

In [14]:
reshaped_images.shape

Out[14]: (35887, 48, 48, 1)

In [15]:
norm_images = reshaped_images / 255.0

In [16]:
plt.imshow(norm_images[6].reshape((48,48)), cmap="gray")
plt.show()
```

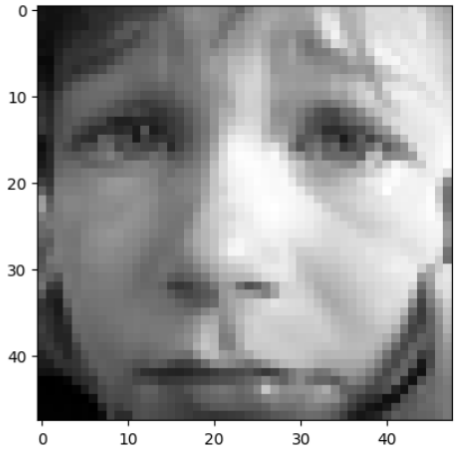

FIGURE 22: Image pre-processing.

4.5 Convolutional Neural Network (CNN) Model

As seen in FIGURE 23, by creating Convolutional Neural Network (CNN) model using Keras API for image and real-time categorization, “Sequential()” created a blank sequential model which allowed adding layers to it in a sequential order. “Conv2D()” function with 64 filters, 3x3 kernel size, and ReLU activation function is used to add the first convolutional layer and the training data's shape is used to define the input shape. The same settings are applied to a second Convolutional layer. The function “MaxPooling2D()” chooses the highest value inside each area of the pool size (2x2) in order to reduce the sample size of the feature maps. By randomly eliminating some of the input units, Dropout() is added to stop overfitting and the second Convolutional layer follows the same structure, with ‘max pooling’ and ‘dropout’ after that. The third Convolutional layer is then added, with 128 filters, a 3x3 kernel, and a ReLU activation function.

In [21]:

```
model = Sequential()

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1:])))
model.add(Conv2D(64,kernel_size= (3, 3), activation='relu'))
# model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

#2nd convolution layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
# model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

#3rd convolution layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
# model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Flatten())
```

FIGURE 23: A CNN model with multiple convolutional layers, pooling layers, and fully connected layers

The 2D output of the preceding layer is then changed to a 1D vector using the “Flatten()” function. Two completely connected layers with 1024 neurons each and ReLU activation function are added to minimize overfitting and Dropout is implemented after each fully connected layer. In order to acquire class probabilities, a dense output layer with softmax activation function is added, which has a number of neurons equal to the output classes. The initial setting of the Adam optimizer's learning rate, which determines how much the weights are updated each iteration, is 0.0005. The model is constructed to use the Adam optimizer, with categorical cross-entropy acting as the loss function (typically used for multi-

class classification problems) and accuracy serving as the performance metric to assess the model's efficacy during training. The optimizer, loss function, and metrics are configured as part of the model compilation process via the “compile()” method. Using the “fit()” function, the model is able to be coached after it has been compiled, which can be seen in FIGURE 24.

```
#fully connected neural networks
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(num_labels, activation='softmax'))
```

In [22]:

```
optimizer = Adam(learning_rate = 0.0005)

model.compile(
    optimizer = Adam(learning_rate=0.0005),
    loss='categorical_crossentropy',
    metrics=['accuracy']
  )
```

FIGURE 24: Class probabilities and configuration of the optimizer and compilation of the model.

The Keras function call “model.summary()” is also used to produce a list of the layers and parameters of the model’s neural network. Each model layer's type, output shape, and number of trainable parameters are listed in a row for each layer in the summary. Each layer's output shape is presented as a tuple in which the first value denotes the batch size and the remaining values denote the size of the output tensor.

4.6 Test Model and Validation

In order to allow the model and variable to be used in the future variable to make predictions on new data without having to retrain it, it is necessary to make a trained model and loaded it under a .h5 file. In FIGURE 25, the model is plotted to train the model for identifying facial expressions for 15 epochs using the “fer2013” dataset. The dataset name is specified in the "datasets" list and set up three callback functions: ModelCheckpoint, ReduceLROnPlateau, and EarlyStopping, which will be used during training to save the best model, reduce the learning rate if the validation loss does not improve, and stop the training early if the validation loss does not improve after a certain number of epochs. The callbacks are combined into a list called "my_callbacks" and the training and validation data (X_train, y_train, X_test, y_test) are passed to the model's "fit" method along with the number of epochs, verbose mode, and the list of callbacks. Training involved updating the model weights based on training data, and computing

accuracy and loss using both training and validation data. The ‘history’ variable kept track of the previous occurrences of these values.

In [25]:
```
datasets = ['fer2013']
num_epochs = 15 # Number of epochs to train the model.
base_path="/FERProject"
for dataset_name in datasets:
    print('Training dataset:', dataset_name)

    #callbacks
    model_checkpoint = ModelCheckpoint('fer_model.h5', monitor='val_loss', verbose=1,mode='max')
    reduce_lr = ReduceLROnPlateau('val_loss', factor=0.1,patience=2, verbose=1, mode='auto')
    early_stop = EarlyStopping('val_loss', patience=patience, verbose=1)

    my_callbacks = [model_checkpoint, early_stop, reduce_lr]

    # loading dataset
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=num_epochs, verbose=1,callbacks=my_callbacks)
```

FIGURE 25: The training of a Convolutional Neural Network (CNN) for Emotion Detection.

As shown in FIGURE 26, the model’s performance is recorded and displayed in the Notebook with the “%matplotlib inline” command. The history object that was given back after the model was trained is used to retrieve the training and validation loss and accuracy metrics, and plotted the training loss, validation loss, and training accuracy on separate graphs, along with the validation accuracy and to produce a fresh figure for each plot using the "plt.figure()" function.

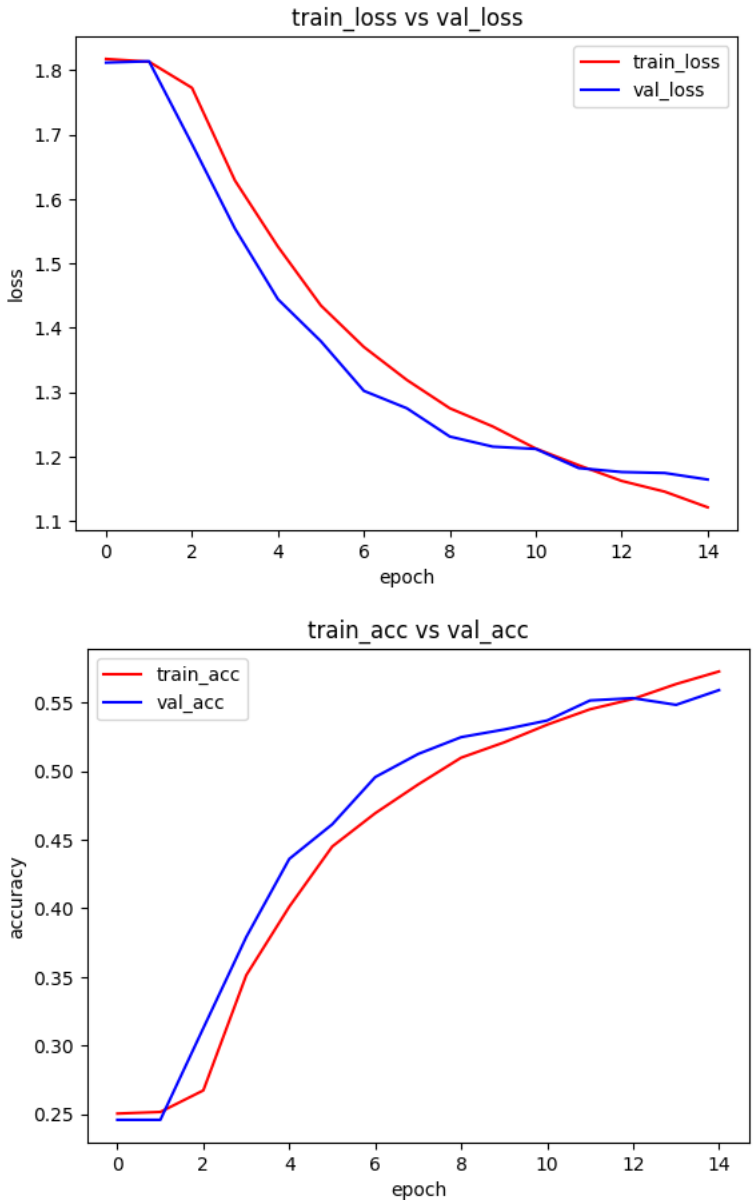

FIGURE 26: Visualization of the performance of the trained model during the training process.

4.7 Emotion Detection Process

The Emotion Detection classifier must be initialized in order to recognize and analysed imagines and real-time movement from webcam and also in order to use the loaded model to make predictions. “Model_Weights.h5” is the file that is saved during training using the “model.save_weights()” method and is loaded the stored model's weights from this file, which can be seen in FIGURE 27. The weights stand in for the learnt parameters that were gained by the model during training to produce precise predictions. Using the Haar Cascade technique, the classifier is initialized in the following line to find faces.

The “load()” function loaded the pre-trained XML file for the Haar cascade classifier that is placed in the OpenCV data directory after creating a classifier object with “cv2.CascadeClassifier()”.

```
In [2]: model = model_from_json(open("./model.json", "r").read()) #reading the saved model from training
        model.load_weights('./Model_Weights.h5') #loading the weights from saved file after training

In [13]: face_cascade = cv2.CascadeClassifier() #initializing classifier to detect faces using haar_cascade
         face_cascade.load(cv2.data.haarcascades + './haarcascade_frontalface_default.xml')
```

FIGURE 27: Utilizing the imported model and setup the face detection classifier.

To take the facial image through webcam, the model created the "cap" camera object using the OpenCV library. The function cv2 received the parameter "0" as input. The number 0 specifies the default camera on the system, “VideoCapture()” represented the index of the webcam to be used. Frames from the camera are continuously received by the while loop, which then examines them to identify faces and deduce emotions, as shown in FIGURE 28, and the height, width, and channel of the ‘frame’ are returned by the frame's shape attribute, which is then used to extract the frame's measurements. The 'res' variable is given as the outcome of applying a weighted sum of the 'sub_img' and 'rect' arrays using the cv2.addWeighted() function. The generated ‘sub-image’ is blended with a black rectangle to add contrast and enhance the visibility of the text labels that will be presented afterwards.

```
In [7]: cap = cv2.VideoCapture(0) #initializing camera using cv2 to capture face
        while cap.isOpened():
            #setting up the camera frame and UI
            res, frame = cap.read()
            height, width , channel = frame.shape
            sub_img = frame[0:int(height / 9),0:int(width)]
            rect = np.ones(sub_img.shape, dtype = np.uint8) * 0
            res = cv2.addWeighted(sub_img, 0.77, rect, 0.23, 0)
```

FIGURE 28: Initialising webcam with OpenCV library.

Front, label and position properties are necessary when the webcam feed is on. The label's dimensions are determined by the program using the “cv2.getTextSize()” method, which accepts the label, font, scale, and thickness as inputs and returns the label's measurements. The ‘textX’ and ‘textY’ variables are determined using the label's measurements and the “res” image's form, also displayed the label using “cv2.putText” function. The model uses the “cap.read()” function to take frames from webcam and used

“cv2.cvtColor()” to convert the taken frames to grayscale. The output of the “face_cascade.detectMultiScale()” method, which located face in the grayscale picture, is set as the value of the faces variable, which is shown in FIGURE 29.

```
#styling font and label properties
FONT = cv2.FONT_HERSHEY_TRIPLEX
FONT_SCALE = 0
FONT_THICKNESS = 0
label_color = (64, 255, 0)
label = ""

#positioning
label_dimension = cv2.getTextSize(label, FONT, FONT_SCALE, FONT_THICKNESS)[0]
textX = int((res.shape[1] - label_dimension[0]) / 2)
textY = int((res.shape[0] + label_dimension[1]) / 2)
cv2.putText(res, label, (textX, textY), FONT, FONT_SCALE, (255, 255, 255), FONT_THICKNESS)

#capturing faces and converting the images to grayscale
gray_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray_image)
```

FIGURE 29: Font and label properties with frames capturing functions.

In FIGURE 30, the model analysed the faces found by the face detection algorithm, which also using as a base for identity through face detection, and applied a trained model to estimate the emotion that was expressing. By using the “for” loop, the face in the webcam frame will be recognized by the “face_cascade” object and with the “cv2.rectangle()” method, it drawn a region of interest (ROI) rectangle around each face. The ROI is chosen from the grayscale version of the frame and resized to 48x48 pixels, which is what the emotion detection model anticipates.

```
try:
    for (x, y, w, h) in faces:
        #selecting Region of Interest(ROI) from camera
        cv2.rectangle(frame, pt1 = (x, y), pt2 = (x+w, y+h), color = (64, 255, 0),thickness =  2)
        roi_gray = gray_image[y-5 : y+h+5, x-5 : x+w+5]
        roi_gray = cv2.resize(roi_gray, (48,48))
        image_pixels = img_to_array(roi_gray) #converting image to array for prediction of emotions
        image_pixels = np.expand_dims(image_pixels, axis = 0)
        image_pixels /= 255
        predictions = model.predict(image_pixels) #using the model to predict emotions
        max_index = np.argmax(predictions[0]) #storing max prediction value to max_index
        emotions = ('Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral')
        emotion_prediction = emotions[max_index] #selecting matching prediction from emotions tuple
```

FIGURE 30: Emotion predictions for the input (resized) image.

Using the “img_to_array()” method, the resized image is normalized by dividing by 255 and turned to an array. The “model.predict()” method used this array to feed the input image to the emotion detection model, which returns a collection of predicted facial expressions. The emotion with the highest probability is selected by using the “np.argmax()” function, which to get the index of the maximum probability value, and the relevant string label for the expected emotion is found by looking it up in a tuple named ‘emotions’.

The “cv2.putText()” method is used to show the predicted emotion and the prediction's accuracy as labels on the user interface and the “cv2.imshow()” method helped to display the resultant frame. In order to prevent any mistakes that may appeared during the face detection and emotion prediction processes, the whole code block is enclosed in a “try-except” block. If an error occurs, the block is skipped, and the for loop continued to loop through once again, like what is shown in FIGURE 31. "Emotion Detector" is shown on the frame as the title, with the bonus tag indicated the first date that this project was made.

```
            #displaying prediction and accuracy as labels in UI
            cv2.putText(res, "Emotion: {}".format(emotion_prediction), (40, textY+5+5), FONT, 0.7, label_color, 1)
            label_violation = 'Accuracy: {}'.format(str(np.round(np.max(predictions[0]) * 100, 1)) + "%")
            cv2.putText(res, label_violation, (400, textY+5+5), FONT, 0.7, label_color, 1)
    except:
        pass

    frame[0:int(height / 9), 0:int(width)] = res
    cv2.imshow('Emotion Detector (Create on: 16/02/2023)', frame) #frame (window) title
```

FIGURE 31: The resulting frame.

To make sure that the model will exist the ongoing process of the Emotion Detector, “cv2.waitKey()” function waited one millisecond for a key event and the loop is broken and the sequence moved on to the following lines of code when the ‘v’ is pressed. After checking the keyboard input, the model will release the video capture resources and closed all the windows using the “cv2.destroyAllWindows()” function, as seen in FIGURE 32.

```
    #exit application on pressing 'v'
    if cv2.waitKey(1) & 0xFF == ord('v'):
        break

cap.release()
cv2.destroyAllWindows()
```

FIGURE 32: Closing and released the webcam’s resources.

4.8 Final Result

The author checked the result by running the cell on Jupyter Notebook and wait for the webcam light turned on to examine the outcome. The emotion in the feed and the real-time facial expression has been made to be comparison with the ones that the individual is actually displaying. The accuracy score of the emotion forecast has done a passable calculation which is also a good indication of the model's performance if it is consistently high. FIGURE 33 shown the author's emotion prediction result that contained multiple possible emotions.

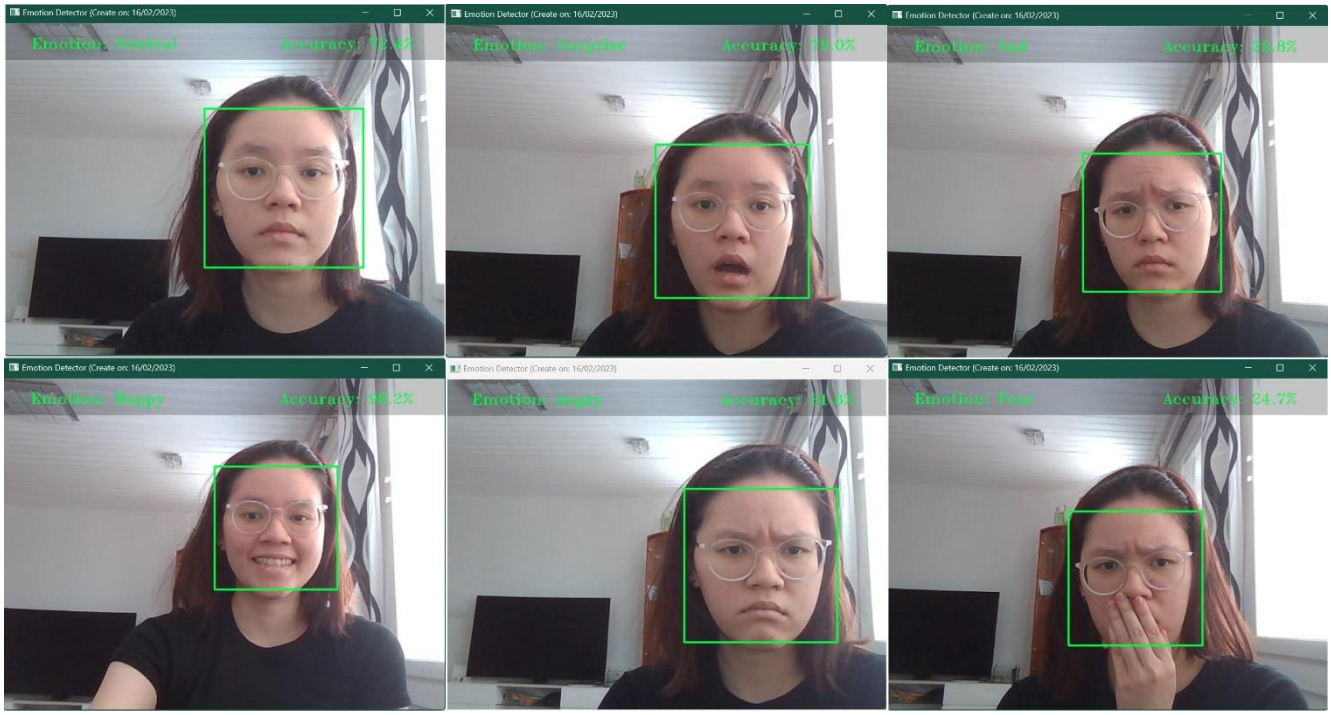

FIGURE 33: Displayed result and performance.

5 CONCLUSION

Face recognition is an increasingly popular and important application of computer vision technology, which also advanced in emotion detection and have been developed from the base of machine learning application and benefits in Artifact Intelligence. By using Deep Learning algorithms and Convolutional Neural Network, Real-time Emotion Detection systems are able classify emotions quickly in real-time video feeds. This thesis served the purpose of learning more about the world of machine learning and created a Real-time Emotion Detector using Python programming language as the achieved goal. By putting the concept of teaching a machine to learning the basic of face detection to expand to facial expressions of human being and made it a practice and cultured experiments using accessible high tech like OpenCV.

However, there are still some limitations and challenges to overcome. One major challenge is ensuring the accuracy and reliability of emotion detection algorithms upon different structures of human anatomy, as well as the possible hindrances like facial modification, along with dysmorphic, objects like glasses, hairs, and even in facial expression that most people is unable to express correctly. There is also a potential for bias or discrimination in the algorithms used that get to detect emotions based on how easily to capture on webcam skin tones. To ensure the equality in recognition of facial emotions, this application should has testing more with different individuals and with different structure of face, races, ethnicity, gender, age and possible craniofacial anomaly people, to learn more about how this detector could get some more cutting-edge development in modern world.

REFERENCES

Agarwal, D. 2020. *A Review of the Math Used in Training a Neural Network*. Available at: https://levelup.gitconnected.com/a-review-of-the-math-used-in-training-a-neural-network-9b9d5838f272. Accessed 12 January 2023.

Bengio, Y. 2009. *Learning Deep Architectures for AI*. Foundations and Trends in Machine Learning: Vol. 2: No.1, 1-127. Deft: Now Publishers Inc.

Bhardwaj, N, Dixit, M. 2016. *A Review: Facial Expression Detection with its Techniques and Application*. International Journal of Signal Processing, Image Processing and Pattern Recognition Vol. 9, No. 6/2016, 149 – 158.

Bradski, G & Kaehler, A. 2008. *Learning OpenCV*. California: O'Reilly Media.

Bledsoe, W.W. 1964. *The Model Method in Facial Recognition*. Technical Report, PRI 15. California: Panoramic Research, Inc.

Chollet, F. 2018. *Deep Learning with Python*. New York: Manning Publications.

Dongare, A.D, Kharde, R.R, Kachare, A. 2012. *Introduction to Artificial Neural Network*. International Journal of Engineering and Innovative Technology (IJEIT), Vol. 2, No. 1/2012.

Fu, W, Menzies, T. 2017. *Easy over Hard: A Case Study on Deep Learning.* In Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany.

Fukushima, K. 1980. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biological Cybernetics Vol. 36, No. 4/1980, 193–202.

Janiesch, C & Zschech, P & Heinrich, K. 2021. Machine learning and deep learning. Available at: https://arxiv.org/ftp/arxiv/papers/2104/2104.05314.pdf Accessed 10 February 2023.

Javapoint. Available at: https://www.javatpoint.com/artificial-neural-network. Accessed 1 February 2023.

Kaggle. Available at: https://www.kaggle.com/. Accessed 21 December 2022.

Kanade, T. 1973. *Picture processing system using a computer complex*. PhD Thesis. Kyoto: Kyoto University, Department of Information Science.

Kasar, M & Bhattacharyya, D & Kim, T. 2016. *Face Recognition Using Neural Network: A Review*. International Journal of Security and Its Applications, Vol. 10, No.3/2016, 81-100.

Ketkar, N, Moolayil, J. 2021. *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*. New York City: Apress.

Krizhevsky, A, Sutskever, I, Hinton, G. 2012. *ImageNet classification with deep convolutional neural networks*. NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems, Vol. 1, 1097–1105.

Kumari, J, Rajesh, R, Pooja, KM. 2015. *Facial expression recognition: A survey*. Second International Symposium on Computer Vision and the Internet, 486 – 491.

LearnPython. Available at: https://learnpython.com/blog/ Accessed 9 February 2023.

LeCun, Y, Bottou, L, Bengio, Y and Haffner, P.1998. *Gradient Based Learning Applied to Document Recognition*. Proceedings of IEEE, Vol. 86, No. 11/1998, 2278-2324.

Mase, K. 1991. *Recognition of facial expression from optical flow*. IEEE TRANSACTIONS on Information and Systems, Vol E74-D, No. 10/1991, 3474-3483.

Matsumoto, D & Hwang, H. 2011. *Reading facial expressions of emotion*. Psychological Science Agenda, Vol. 25, No. 5/2011, 10-18.

Miao, J, Yin, B, Wang, K, Shen, L, Chen, X. 1999. *A hierarchical multiscale and multiangle system for human face detection in a complex background using gravity-center template*. Pattern Recognition, Vol. 32, No. 7/1990, 1237-1248,

Mistry, V, Goyani, M. 2013. *A Literature Survey on Facial Expression Recognition using Global Features*. International Journal of Engineering and Advanced Technology Vol. 2, No. 4/2013, 1 – 5.

Papageorgiou, C & Poggio, T. 2000. *A Trainable System for Object Detection*. International Journal of Computer Vision Vol. 38, No.1/200, 15-33.

Rathi, A, Shah, B. 2016. *A Survey: Facial Expression Recognition*. International Research Journal of Engineering and Technology Vol. 3, No. 4/2016, 540 – 545.

Roman, V. 2020. *Deep Learning: Introduction to Tensors & TensorFlow*. Available at: https://towardsdatascience.com/deep-learning-introduction-to-tensors-tensorflow-36ce3663528f Accessed 9 February 2023.

Rowley, H, Baluja, S, Kanade, T, et al. 1996. *Neural Network-Based Face Detection*. Doctoral Thesis, Computer Science. Pittsburgh: Carnegie Mellon University.

Simonyan, K & Zisserman, A. 2014. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. The 3rd International Conference on Learning Representations. Available at: https://arxiv.org/pdf/1409.1556.pdf. Accessed 2 February 2023.

Sinha, P & Balas, B & Ostrovsky, Y & Russell, R. 2006. *Face Recognition by Humans: Nineteen Results All Computer Vision Researchers Should Know About*. Proceedings of the IEEE. 94. 1948 - 1962.

Szegedy, C, Liu, W, Jia, Y, Sermanet, P, Reed, S, Anguelov, D, Erhan, D, Vanhoucke, V, Rabinovich, A. 2015. *Going Deeper with Convolutions*. Available at: https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf. Accessed 7 February 2023.

The NumPy Community. 2022. *NumPy User Guide*. Available at https://numpy.org/doc/1.22/numpy-user.pdf Accessed 9 February 2023.

The SciPy Community. 2013. *SciPy Reference Guide*. Available at https://docs.scipy.org/doc/scipy-0.13.0/scipy-ref.pdf Accessed 9 February 2023.

Wu, J. 2017. *Introduction to Convolutional Neural Networks*. China: Nanjing University. Available at: https://cs.nju.edu.cn/wujx/paper/CNN.pdf Accessed 7 February 2023.

Yang, G, Huang, T. 1992. *Human face detection in a complex background*. Pattern Recognition, Vol. 27, No. 1/1994, 53-63.

Yang, M, Kriegman, D, Ahuja, N. 2002. *Detecting Faces in Images: A Survey*. IEEE Transaction on Pattern Analysis and Machine Intelligent, Vol. 24, No. 1/2002.

Yegulalp, S. 2022. *What is TensorFlow? The machine learning library explained*. Available at: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html Accessed 9 February 2023.

Zeiler, M, Fergus, R. 2013. *Visualizing and Understanding Convolutional Networks. European Conference on Computer Vision*. Available at: https://papers.baulab.info/Zeiler-2014.pdf. Accessed 9 February 2023.

Zhang, A, Lipton, Z, Li, M, Smola, A. 2021. *Dive into Deep Learning*. Available at: https://d2l.ai/d2l-en.pdf. Accessed 2 February 2023.