Final year project

Visual recognition of human postures inside MTR station control room for anti-robbery

Supervisor: Dr. S.H. Choi

Student: Wong Mun Hin (Joseph)

University Number: 30350 73484

Year: April 2017

Abstract

This study aims to develop a visual recognition program for human and posture detection for the MTR company. In this dissertation, the idea of using machine learning for recognition problem would be discussed. The study used a deep learning approach, which is convolutional neural network, to train the human recognition and posture recognition algorithm. The data was obtained from an online open-sourced dataset called INRIA Person Dataset, as well as by manually collection. The convolutional neural network was constructed with Keras, a high-level Python library. Combining the recognition algorithm with a sliding window algorithm, the program successfully detected human from different images. The result was satisfying, and further study was suggested to improve the accuracy and performance of the program.

Acknowledgements

This study was made possible with the help of my supervisor, Dr. S.H.Choi. Dr. Choi provided a lot of insight and aspiration to me when I faced difficulties in the research. With his expertise knowledge in e-commerce and programming, he also provided me with a lot of ideas to improve my algorithm. I would like to thanks Dr. Choi for providing me a lot of support during the project.

I would also like to thanks Sampson, representative of the MTR company. Sampson provided me the direction and goal of this project, and also patiently explained a lot of details in the current MTR system. Without Sampson, the study would have a much difficulty start.

Content Page

Chapter 1: Project Introduction

This project started with the MTR company “Rail Gen 2.0” initiative. In this section, I would introduce the project background, identify the problem, define project objectives and give an outline of methodology used in the project.

1.1 Project Background

With the ever-increasing demand in railway service and changing technology, the MTR company is enhancing its existing railway network to the “Rail Gen 2.0”. Rail Gen 2.0 is a project that brings superior connectivity, better facilities and enhanced services to the public. [1] In addition to the extension of network and upgraded infrastructure, the Rail Gen 2.0 is also a great opportunity for MTR to revisit the technology they are using for now. With the advancement in technology, MTR felt the need to research on what could be applied for better operational efficiency and customer experience in the new Rail Gen 2.0. Therefore, the department of Industrial and Manufacturing Systems Engineering of HKU offered help for MTR to investigate this topic. Three students, me included, were engaged in different research projects for their Final year project, in the hope of helping MTR to investigate the use of new technology in their service, and complete our Final year project with the same topic.

After discussion with the representatives from MTR, I had selected the topic of Artificial Intelligent (AI) and investigated in which area AI could be best applied for better performance. With further meeting with one of the MTR representatives, Sampson, it was suggested that a posture recognition program could be designed for visual recognition.

The MTR station control room is a small office for station staff to monitor the station situation and handling different tasks. Currently, there are surveillance cameras installed inside the rooms. Whenever there are emergency situations, such as robbery or fighting inside the room, the station staff would trigger the panic button, which will start the recording function of the camera, as well as posting the record to the MTR intranet system. The video of that station room would be then available to office staff for taking the corresponding actions. Being manually operated, the undue reaction time and sometimes human errors often delay the required actions. Also, in case of robbery, the station staffs may not be able to trigger the panic button at all. These will result in a slow respond to the event and losing important data from recording. It is therefore desirable to develop a system that incorporates Artificial Intelligence for automatic visual recognition of human postures. In event whether certain postures are detected, the panic button function would be trigger automatically.

As the use of Artificial Intelligent is crucial for this topic, this project requires heavy knowledge on both machine learning as well as programming. Some of the basic concept of machine

[1] “RAIL GEN 2.0 NEXT GENERATION RAIL”, MTR Corporation Limited website, http://www.mtr.com.hk/en/customer/main/railgen_2.html

learning, such as simple linear regression and logistics regression were covered in IMSE 3108, Operational Research I. While the basic of programming concepts were covered in ENGG 1112, Computer Programming I (With C++ as language). In addition, I had previously learned Python programming and completed the Machine Learning online course by Stanford University at Coursea. Therefore, I believed I am capable to complete this project and it would be a good challenge for me.

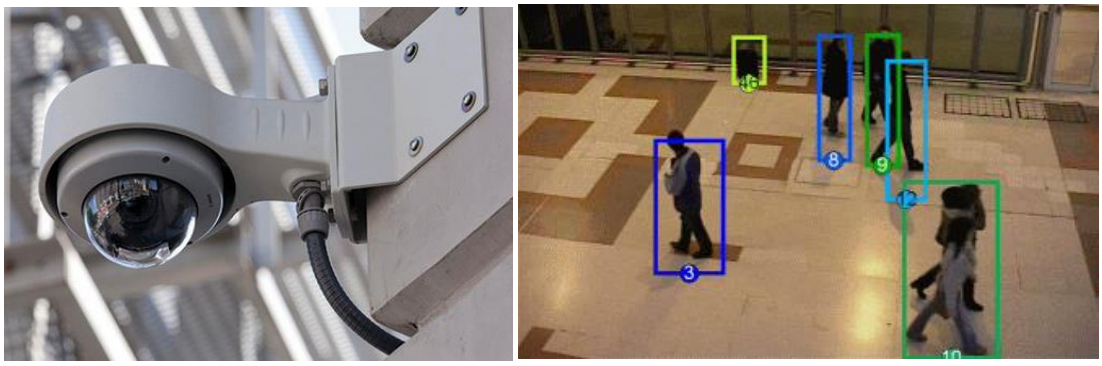


Fig 1. Surveillance camera (Left) and Artificial Intelligent human detection (Right)

1.2 Problem Identification

The problem described above seem straight forward, but is in fact very complicated and involves a lot of sub-problems and objectives. In this section, I will identify the steps and challenges of the problem and state which part I will be focusing on for my final year project.

The general process of the entire visual recognition program is as follow. The program first retrieves real time image data from the camera, then runs the human recognition algorithm to identify the human in the image. Then it runs the posture recognition algorithm to classify their postures. As the main purpose for the algorithm is anti-robbery, the postures to be recognized are weapon holding and surrendering. The algorithm should return positive if recognizing these two postures. After receiving the positive feedback from the algorithm, the panic button should be triggered to post record and start recording.

First of all, the program involves fetching real time data from the MTR surveillance camera and effectively process the frames. As the AI program is supposed to run on real time, a program is required to effectively fetch data from the surveillance camera. To develop this program, the MTR surveillance camera information, such as resolution, data format and communication channel are needed. However, due to company policy of MTR, the listed information is not obtainable. With a discussion with Sampson, it was agreed that the project should pay less attention on this part, and focus on other more important area.

Secondly, the main AI of the program should be developed. There are a lot of different ways to recognize human postures, which I would discuss in detail in the later chapters. After some research, it is believed that one of the effective method is to first recognize human from the

image, and then run another AI program on the human image to deter certain postures. With this approach, two separate AIs should be trained, one specializes in human recognition, and another one specializes in postures recognition. The human recognition algorithm should be able to identify human in any given image, crop them out, and feed them into the postures recognition program for classification. Due to resources and time constraint, it was agreed that the focus of the project should be the development of these two AI algorithms.

Lastly, after the completion of the two AI programs, everything should be put together for a fully functional program that fetch data from camera, classify it and return the result of classification. In order to trigger the panic function, the current system need to be studied. The outputting data format should be compatible with the MTR current system, the program should also work seamlessly with the system. Again, due to time and resources limitation, it is agreed that this part will not be the main focus of the project.

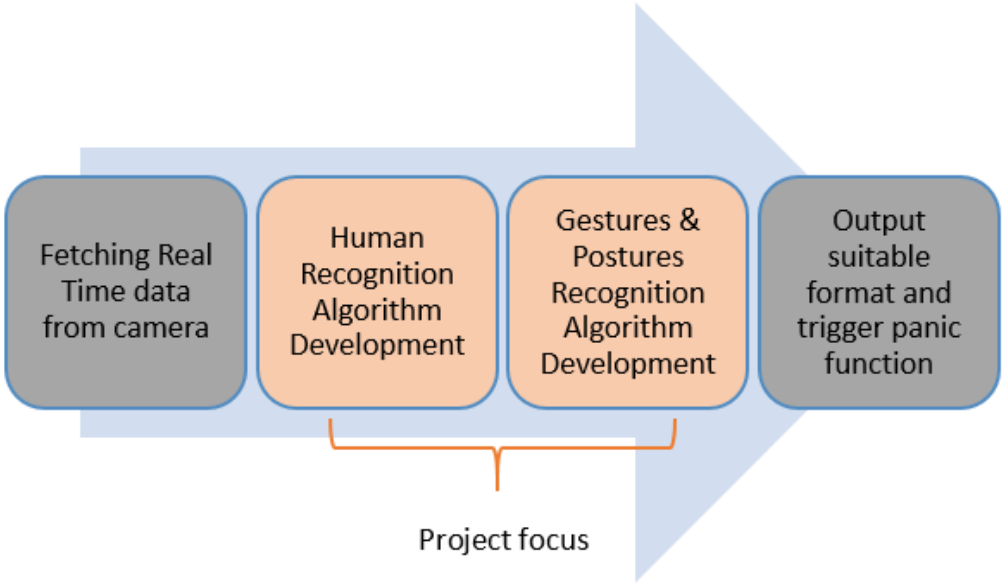


Fig 2. The general tasks of the project and tasks to be focused on

The development of the two recognition algorithms involve two separate process, the algorithm training and the actual program development. A machine learning algorithm should first be trained to provide accurate prediction, the algorithm training process is an important part of the project. After the algorithm is trained and tested, it could be used for the actual recognition program, where the algorithm would predict the type of the input image by previous training experience. The challenge for these process is to figure out an effective way to train the algorithm and the accuracy of the algorithm in real-life situation.

1.3 Project Objectives

Project Objectives are the breakdown of the high-level problem into smaller, more tangible project statements. These statements are works to be done, product to be made or service to

be delivered. A good project objective should be clear, easy to understand and measurable. As mentioned in the previous section, this project would focus on the development of two recognition algorithms, and the two goals are training the algorithm and apply the algorithm in the program. As these two goals are very different in their nature, I would discuss them one by one in the following.

1.3.1 Algorithm Training Objectives

Algorithm training is the most important part of an Artificial Intelligent system. When we talk about training an algorithm, we are referring to the parameters inside the algorithm being trained to fit our requirement. The training itself is done by machine learning, the use of training data, machine learning algorithm, minimizer and training setting would all affect the performance of the algorithms. Therefore, there are a lot of decision to be made before the training. In addition, algorithm training is an iterative process. The trained algorithm and parameters will be tested with different data for performance checking. If the performance is not as expected, the algorithm and parameter may need to be trained again with different setting. The objectives of this phrase include:

1. To obtain and process training data

Training data collection is the most important part of machine learning. The performance of the algorithm highly depends on the number and variety of data. Generally, a simply algorithm with larger training dataset outperforms a sophisticated algorithm with less data. Therefore, it is of most importance to obtain a large training dataset for your problem to train algorithm with high accuracy. The more similar the training data and the actual data the better. Therefore, the data may need to be processed first before feeding in the machine learning algorithm. For my two algorithms, I would need to obtain human image data for the human recognition algorithm, and human with different postures image for the postures recognition algorithm.

2. To choose machine learning algorithm for classification

After collecting the training data, the machine learning algorithm need to be determined. Different machine learning algorithms have different characteristics and are suitable for different situations. Some algorithms work best with a simply problem and some work best with more complicated problem. The machine learning algorithm required for this problem is classification algorithm, where the algorithm try to determine the class of certain input features. For our human recognition algorithm, the class would be human and non-human. While for the posture recognition algorithm, the classes would be surrendering, weapon-holding and no special posture or posture.

3. To choose the program or software to train the machine learning algorithm

Different programming language and software could be used to train the machine learning algorithm. However, some of the programming language and software do have better support

on running machine learning. Therefore, choosing these technologies for the training could effectively reduce the trouble in constructing the algorithm and messing with different setting.

4. To implement and test the machine learning algorithm

After deciding everything, the machine learning algorithm need to be constructed and ran. Depending on the programming language, this process could be very different. After successfully constructing the algorithm, the algorithm need to be trained with the training data. The result should be tested before moving on to the recognition program.

And as mentioned before, the whole machine learning process is iterative, which means that if problems are found in any stages, the whole process should be revisited and looked for room of improvement. There are no correct solutions for machine learning, therefore a lot of experiments are needed.

1.3.2 Recognition Program Objectivew

After completing the machine learning phrase, the actual recognition program use the trained machine learning algorithm and perform classification. The two recognition programs are a bit different in nature. As the first human recognition algorithm need to be able to recognize any human from the input image, and output the result to the postures recognition algorithm for further process. Therefore, it is more complicated. For now, I will not go into the detail of the algorithm, and only go through the basic objectives.

1. To decide on the language or software to use

The language and software used for the recognition highly depends on what technologies have been chosen in the machine learning phrase. As different programming languages may not be fully compatible, and it may cause a lot trouble trying to combine languages and software not supposed to work together. Therefore, choosing the languages or software for the recognition program is a crucial task.

2. To construct the recognition program

After the programming language or software is chosen, the recognition program should be constructed. The program should be able to receive input, run the recognition algorithm and return output. Other consideration, such as how the users interact with the program, whether the program could handle errors and what data format to output are also crucial for the performance of the program.

3. To convert the program into window executable application

The last step of the recognition program is to convert the entire program into window executable application. As for most programming language, such as MATLAB or Python, a complier is required to compile and run the code. However, the customer device may not have the specific complier, or the library required for the code. Therefore, the program should be

converted into a window executable application, where the application itself contains everything it needed to compile and execute.

1.3.3 Other Objectives

In addition to the above objectives, this project also serves as a research on how machine learning algorithm could be applied in MTR. Therefore, it is important to analysis the overall performance of the program and possible further improvement, so that MTR could better apply machine learning in their business.

1. To evaluate the performance of the program

The program performance, in term of accuracy, speed, user friendliness and other factors, should be evaluated. If the program performance is not as expected, method of improvement should be proposed for later study purpose.

2. To discuss further study on the topic

As mentioned previously, the algorithm designed was only a prototype to test the using of machine learning on recognition. The MTR company would need further work and study to make it feasible. Therefore, the further study and methodology should be purposed.

1.4 Project Outline

In this section, I will briefly explain the outline of methodology used in this project. The problems descripted, both human recognition and posture recognition are pattern recognition problems. A pattern recognition refers to the identification of the class that the given object belongs to (Mao, 1991). According to Mao, a pattern recognition system composes of three stages, they are:

1. Instantiation

   The training data of the algorithm, in other words the input patterns are read in a particular format, that are computable by the algorithm. The input data may be massive and require special skill to process.

2. Feature Extraction

   The feature of the data is extracted and converted into effective representation for the algorithm. For example, the image data could be converted into matrix of pixel intensity

3. Recognition

   A recognizer for identifying the class of the input feature pattern belongs. The recognizer need to be trained, tested and evaluated before accurate usage.

(Mao, 1991)

The challenges lay behind a pattern recognition problem is getting relevant training data, as well as selection of the recognizer. The general methodology for constructing a pattern recognition algorithm could be represented as follow.

Fig 3. The general process flow of pattern recognition problem

In the instantiation stage, the training data should be collected and converted into format that are computable for the recognizer. Some of the software and library provide functions that help converting the data. After converting the data, the features of the input data need to be extracted and feed into the recognizer. Next, the setting of the recognizer should be configured, and the actual training could be started. Finally, the recognizer should be tested, it performance should also be evaluated. If the performance is not as expected, the whole process, or part of them, should be revisited.

Chapter 2: Literature Review

In this section, I would summarize the literature reviews on the topic in machine learning and different algorithm for pattern recognition. I would analyse what methodology was used in those research, the results or findings and comparing them with my project.

2.1 Review on Machine Learning

What exactly is machine learning? According to Arthur Samuel, one of the pioneers of machine learning, Machine Learning is *"Field of Study that gives computers the ability to learn without being explicitly programmed."* (Arthur, 1959). This definition is still true in the current era. Comparing to traditional programming approach, where a new program was introduced for solving a new problem, machine learning is a concept where the algorithm itself "learn" how to solve new problem by training. A more modern definition of machine learning provided by Tom Mitcell, Professor of the Machine Learning Department in E. Fredkin University, said that *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."* (Tom, 2006) In other words, if the experience and performance of a program are positively proportional, we could say that it is a machine learning algorithm.

For end user, the machine learning algorithm should be a black box that generate output based on input, even without any understanding on machine learning or programming, the user should be able to effectively use the machine learning algorithm for its purpose.

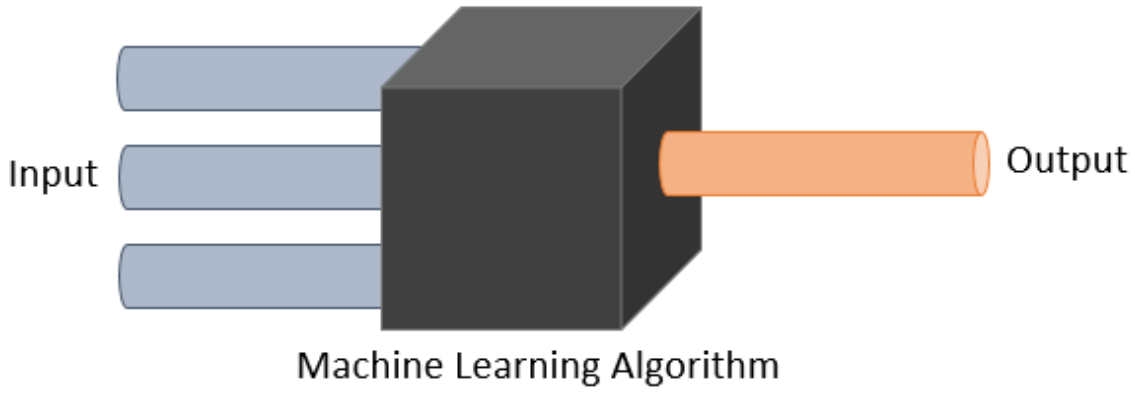


Fig 4. A machine learning concept diagram

Generally, a machine learning algorithm could be classified as either supervised or unsupervised. Each type of algorithms serve different purposes and are used in different situation.

2.1.1 Supervised Machine Learning

Supervised machine learning is one type of machine learning algorithm. In a supervised learning problem, the algorithm is presented with a set of labelled dataset, where the label is the correct output of that data. Therefore we have an idea of the relationship between the input and output to begin with. According to Andrew Ng, professor from Stanford University in the field of machine learning, Supervised problems could be further categorized into “regression” and “classification” problem. (Andrew, 2016) In the regression problem, the output of the problem is continuous. For example, cost, ages and area. To predict continuous result, a function that output continuous result is required. Whereas for classification problems, we are trying to predict results in discrete output. For example, types, patterns and classes. Therefore, a function outputting discrete values is required.

The simplest machine learning algorithm is linear regression. Linear regression is a famous tool used in many areas, including statistics, business and computer science. Linear regression summarizes the relationships between two or more variables, allowing us to observe pattern and make prediction. For a 2-dimension linear regression, where only two variables are concerned, we could easily plot a graph to show their relationship. The linear function best fit the graph gives us an estimation on the output value Y for any input value X. The process of finding this linear function is a process of machine learning. With more data and examples, we could obtain a better estimation. The same goes with 3-dimension linear regression, where we could obtain an estimation of output based on two variables.

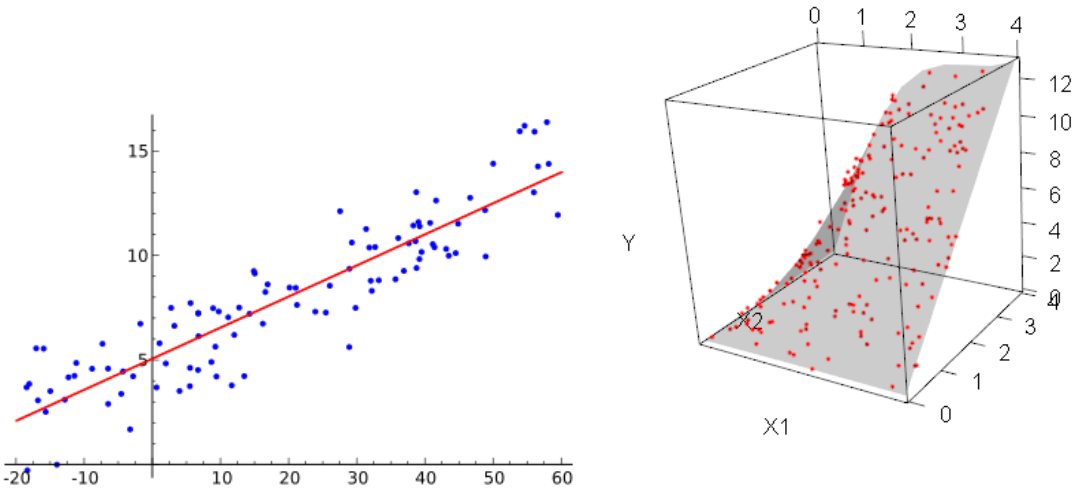


Fig 5. 2-Dimension (Left) and 3-Dimension (Right) linear regression graphic representation

For a 2-Dimension linear regression, the function has two parameters. The two parameters were initially randomly generated, and then learnt by the algorithm. The parameters are used to calculate the output prediction for a certain set of input features, x. For example, if the parameters are θ_0,θ_1, the function of x is:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

This is a simple linear equation, where the value of $h_\theta(x)$ depends on the value of x, as well as the two parameters θ_0, θ_1. To get the optimal value of the parameters, we need to determine whether the function is performing well in terms of prediction accuracy.

The performance of linear regression, or most of the machine learning algorithms, is determined by the cost function. The cost function measures the "cost" of the algorithm, which is the difference between prediction and actual data. The cost function of a linear regression is the sum of square difference of the prediction value and actual value. The cost function J, regarding two parameters θ_0, θ_1 could be written in:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Where m refers to the total number of training data, $h_\theta(x)$ refers to the regression function of x and y refers to the actual output value of x. Generally, a lower cost function means the algorithm and parameter describe the training data better. At the beginning of the algorithm training, the value of the parameters is random. The program will then evaluate the performance of the parameters by calculating the cost function. In order to find optimal value of the parameters, the program needs an optimizer. An optimizer is function that minimize or maximize the value of the objective function by updating the parameters. In the example of simple linear regression, the optimizer minimize the value of $J(\theta_0, \theta_1)$ by updating the value of θ_0, θ_1. The most commonly used optimizer is gradient descent.

Gradient descent is a way to minimize the objective function by updating the parameters in the opposite direction of the gradient of the objective function with respect to the parameter. (Sebastian, 2016) For a 2-Dimension linear regression problem, when plotting the objective function against the parameter value, the objective value is lowest at the optimum amount of parameter. The objective value goes up when the parameter get further away from the optimum value, as shown in figure 6. In a 2-Dimension problem, the gradient is the slope of the function at the specific parameter. The gradient is zero at the local minimum, positive at value smaller than local minimum and negative at value greater than local minimum. Therefore, this tell the algorithm what changes in parameter is required to reach the local minimum. Gradient descent slowly increase or decrease the value of the parameter according to the gradient, until the local minimum is reached.

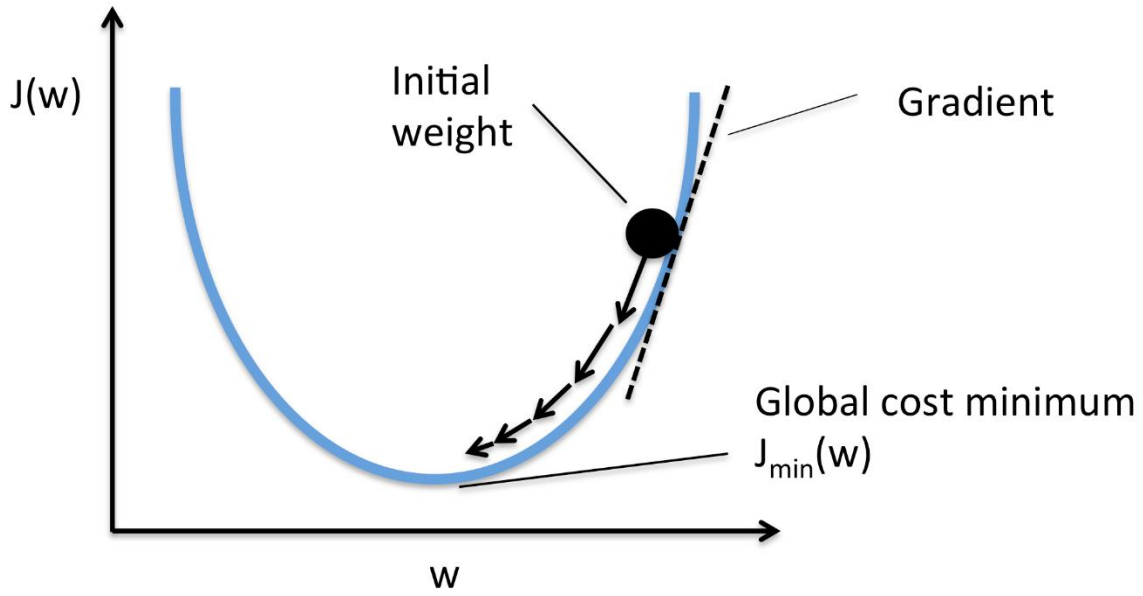


Fig 6. Graphical representation of the gradient descent algorithm

Gradient descent could be classified in batch gradient descent, stochastic gradient descent and mini-batch gradient descent (Sebastian, 2016). The different between them are the data used to calculate the cost function in each iteration. For batch gradient descent, the whole batch of training data is used to calculate the cost function. Stochastic gradient descent use one training example at a time when updating the cost function. Comparing to batch gradient descent, it cut a lot of redundant calculation in each iteration, but the variance in each iteration is higher as the training example is randomly selected. Last but not least, mini-batch gradient descent use a small batch of training example when updating the cost function each time. This method provides some of the benefit of the two above methods. Some of the consideration when using gradient descent are:

- The training rate

  The training rate determines the quantity of parameter value change in each iteration. In other word, how fast the parameter moves towards local minimum. A larger step size means that the algorithm would reach local minimum faster, but have a greater chance of overshooting and never reach local minimum. While a smaller step size results in slower convergence, but a less chance of overshooting.

- The batch of training example to use in each iteration

  The batch of training example to use, which is the type of gradient descent, need to be considered. The type of gradient descent greatly affect the time needed to run the algorithm and the accuracy. Choosing the right gradient descent depends on the problem.

To solve some of the problem of gradient descent, some other optimizers, which are more complicated, were developed. For example, optimizers that could determine the training rate automatically based on gradient, and use lower training rate when approaching convengence. In

practice, these optimizers are preferred as there are library in different languages for these optimizers. A more detail explanation would be given in the methodology section.

Another type of supervised machine learning is the classification problem. In oppose to regression problem, classification problem output discrete value. Classification is used to define class, true or false value of pattern. The two recognition problems in this project are classification problem. The concept of classification problem is very similar with the regression problem. Except that instead of the continuous prediction function, the algorithm use a sigmoid function for prediction. The sigmoid function is defined as:

$$h_\theta(x) = g(\theta^T x) \quad z = \theta^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$ (Andrew, 2010)

Where x is the input features of the algorithm, and θ is the parameter of the algorithm. θ^T is the transpose of θ. So $\theta^T x$ is the sum of all multiplication between θ and x. The final value of g(z) is always between 1 and 0 for any value of x. The relationship between g(z) and x could be represented by the following graph.

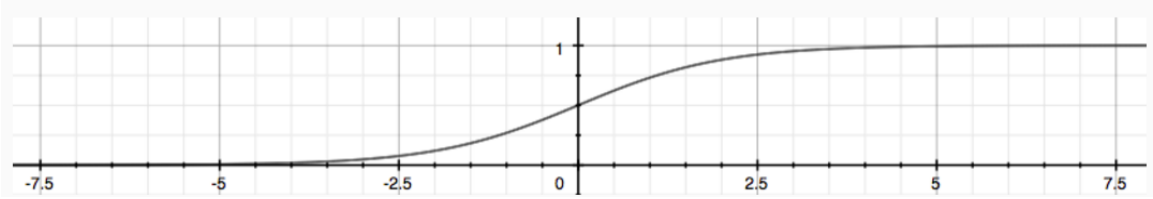


Fig 7. Sigmoid function graph (Andrew, 2010)

For a class with only two classes, we label one class as the positive by giving it value 1 and the other as the negative by giving it value 0. To classify the class of the input training data, we first need to set a threshold value. The threshold value should between 0 and 1. For example, if the threshold is set T, if the output of the sigmoid function is greater than T, we classify the class as the positive class. Else, the input data would be classified as the negative class. The cost function of classification problem is also a little different from the regression problem. As the hypothesis function always output between 0 and 1, if we accumulate all hypothesis function value, the cost function would be very small. In addition, we want to punish the algorithm for wrong prediction by giving it higher cost when the prediction get worse. Therefore, logarithm function is used to transform the hypothesis function. To get positive result for both classes, we have:

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$ (Andrew, 2010)

The whole cost function is then similar to regression problem expect the logarithm version of hypothesis is used.

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$

The process of a supervised machine learning problem training could be generalized as follow.

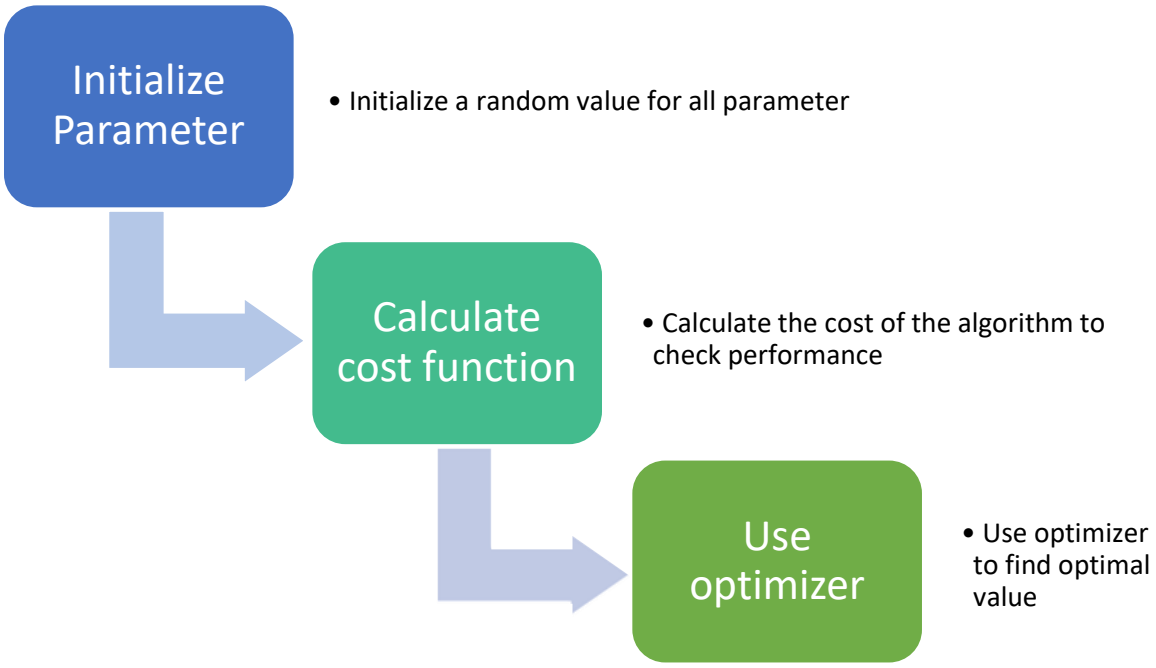


Fig 8. Supervised machine learning problem training

2.1.2 Unsupervised Machine Learning

In oppose to supervised learning, the correct output of the training data is unknown, and the goal of the training is to figure out the correct output of the training data. Unsupervised machine learning was considered when deciding the algorithm for the recognition, however later it is decided to use supervised machine learning instead for better performance. In this section, I would explain the principle and usage of unsupervised machine learning, and also how unsupervised machine learning was used in some similar research.

One of the most famous unsupervised machine learning algorithm is the K-means clustering. K-means clustering is used to classify training data into different clusters. Given a set of data and desirable number of clusters, the algorithm first initialize the position of cluster centers randomly, and then assign each data to its closest center. The algorithm then calculates the mean value of each clusters and move the center to that value, and reassign every data to the

new center called the “K-mean”. The process continues until the distance sum of each data to its “K-mean” is minimum. The following figures illustrate how the algorithm works.

The red and blue cross represent the center of each cluster, and the dot with red and blue color represent data assigned to the cluster respectively. As show in the figures, in each iteration, the total distance between data and cluster center decrease. Therefore, similar data are grouped together.

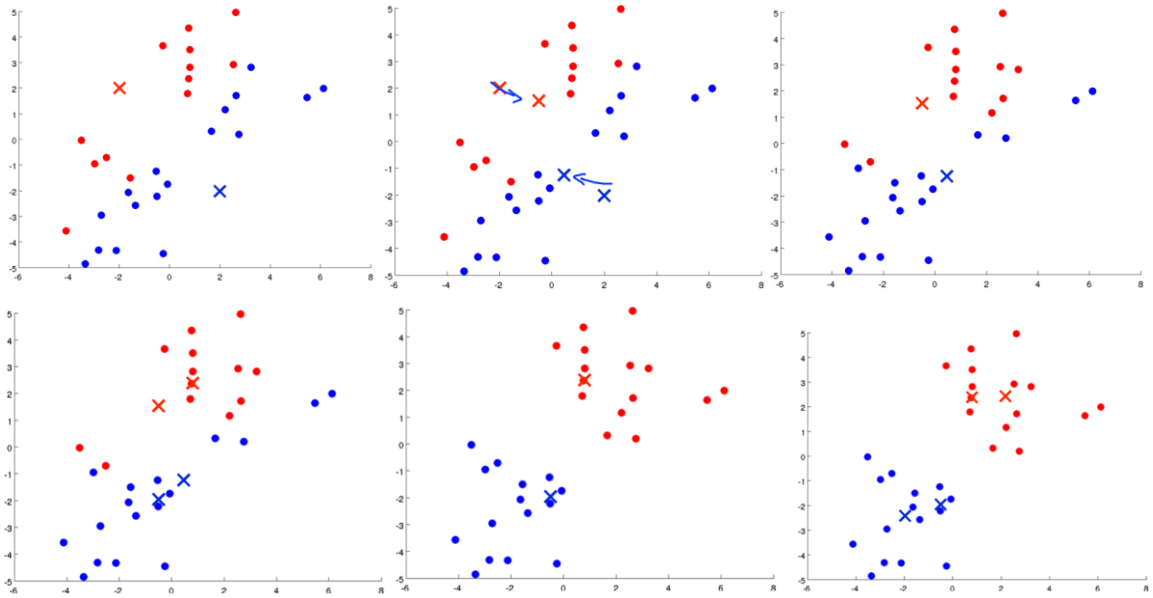


Fig 9. K-means clustering graphic illustration (Andrew, 2010)

Some of the application of K-mean clustering includes data segmentation, image compression and pattern finding. In the research by Siddheswar Ray and Rose H. Turi in their paper *“Determination of Number of Clusters in K-Means Clustering and Application in Colour Image Segmentation”*, K-means clustering was used in colour image segmentation. The two proposed a way to determine the best number of clusters by calculating the intra-cluster and inter-cluster distance, and apply the method to synthetic images and natural images segmentation. The images are simply with different color regions, the goal of the K-mean clustering is to classify these color regions into segments. According to the two, “… *Although there is a tendency to select smaller cluster numbers for natural images, … however, our validity measure performed more consistently for all of the natural images, producing good segmentation results.”* (Siddheswar & Rose, 1999). This research provided a ground of using K-mean algorithm for image segmentation and processing.

Image	Global Minimum	Modified Minimum
balls	2	10
Lenna	2	11
molecule	8	8
teapot	3	8
ant	3	6
blond	2	5
jet	2	5
mandrill	4	10
peppers	2	6
mouse	2	6
rose	3	6

Fig 10. The cluster sizes for different images in the research. (Siddheswar & Rose, 1999)

In additional to image segmentation, K-mean algorithm was also used to perform background separation a lot. An example is Chris Stauffer and W.E.L Grimson in their paper *"Adaptive background mixture models for real-time tracking"*. Their approach to the problem is by modeling each pixel as a miture of Gaussians and using an on-line approximation to update the model. (Chris & Eric, 2014) The "on-line approximation" mentioned is K-mean algorithm. Their approach is to first model the pixel as a mixture of Gaussian, and based on the persistence and the variance, the K-mean algorithm determine which is background. Object color has a larger variance than the background, therefore they could be separated.

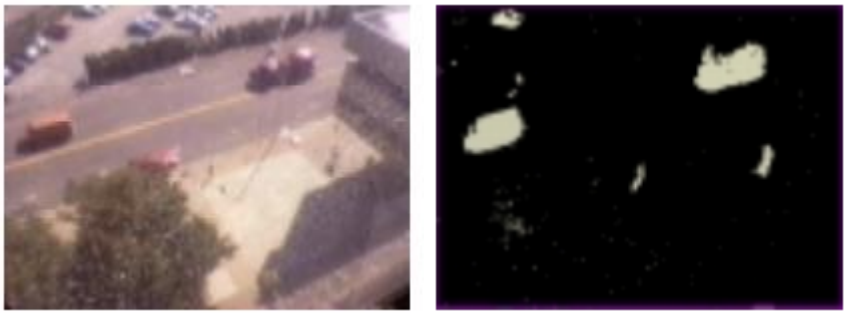


Fig 11. Image used for the algorithm (left) and the pixels separated from the background (right) (Chris & Eric, 2014)

The result was promising, the algorithm was a stable, real-time outdoor object tracker that could reliably deals with lighting changes, repetitive motions and long-term scene changes. However, this method was not used in the project for the following reasons.

First of all, the aim of our two recognitions are to recognize human and their posture. The proposed method in above did well in tracking moving object, however, a human could be relatively still. Our algorithm should be able to recognize both moving and still human. In addition, every single moving object would be tracked. So there is no way we know that the detected pixels are indeed human without manually checking.

Secondly, the authors pointed out that the tracking system has difficulty with scenes containing high occurences of objects that visually overlap. In out problem, the office environment often

has overlaying object, for example human behind desk. So other approach is needed to found the human out of the picture.

Finally, it is the logistics constraint that not allow me to use this method. As collecting a large amount of training video and processing them would take time longer than the duration of the project, it is not feasible to apply this method.

In conclusion, after anaylsing the characterisic of unsupervised machine learning and learning some limitation from previous done research, it was considered the supervised machine learning approach should be used for the two recognition algorithm.

2.2 Pattern Recognition

Our problems are pattern recognition from an image data. The first recognition requires classifying human and non-human pattern, while the second recognition requires classifying between surrender, weapon holding and no special posture patterns. To classify pattern for an image, we need to use to image features for training. The image features are the pixel of the image. The image feature could get massive for a small image. For example, a black and white image with a dimension of 30 x 60 pixel has a total of 1800 features. For RGB color image, each pixel is represented by three values, so we have 5400 features for one single training data. Therefore, it would be impossible to feed the training data into a simply structured linear regression model. In addition, the pixels are all interdependent. We are not only interested in a linear relationship, but a higher-level equation with exponential and multiplication between features. Therefore, the algorithm need to be able to formula high-level polynomial with the input features.

2.2.1 Artificial Neural Network

Artificial neural network, or often simply named neural network, is a type of supervised machine learning algorithm. A neural network is a mimic of the neurons' operation in the human brain. A human neuron is a computational unit that takes an input, computes it, and returns an output. Our brain performs sophisticated functions by linking numerous of neurons together, each neuron's output is fed to other neurons as an input, forming a huge neural network. An artificial neural network consists of many computational units, or nodes. A node is a sigmoid function, which is exactly the same as the sigmoid function of simple logistics regression. Therefore, a neural network with only one node is in fact a logistics regression. The following diagram illustrate how a neural network with one node works.

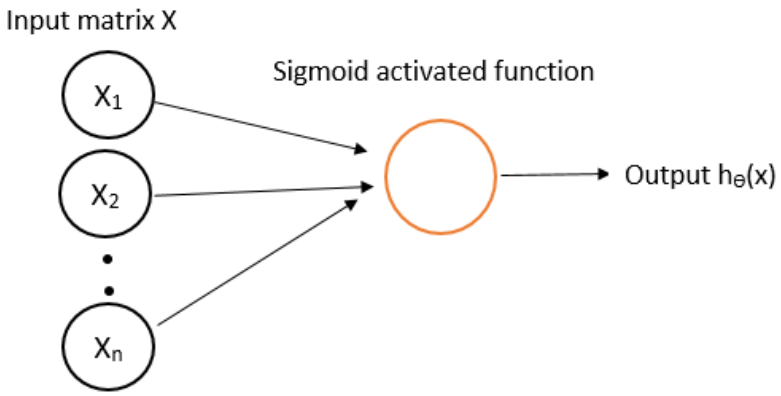


Fig 12. A neural network representation with one computational unit

The input matrix X represents the feature of the input training data. For example, in our image recognition, the input matrix X would be the pixels in matrix form of the training data. As illustrated in the above diagram, the sigmoid activated function take matrix X as input, and return the value of prediction $h_\Theta(x)$. As mentioned in logistics regression, the value of $h_\Theta(x)$ always lay between 0 and 1. The sigmoid function uses trained parameters to calculate the output. Again, the goal is to find the optimum parameters of the function that recognize the input data class best. For a neural network with multiply nodes, the nodes may form layers that take input from previous layer and feed output to the next layer. As the following diagram show.

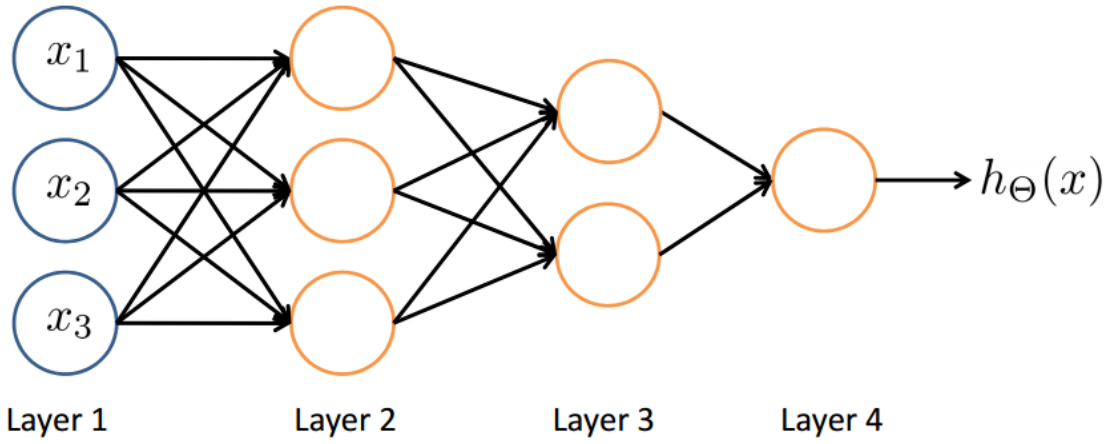


Fig 13. A neural network representation with multiply layers and computational units (Andrew, 2016)

The orange nodes in figure 13 are sigmoid functions. As shown in the diagram, every nodes in the previous layers are fully connected with the next layer. The layers in the middle are called the “hidden layer”, as their output values are not visible. Artificial neural network with multiply layers could create complex features’ map. In the first layer, which is the input features layer, each of the features are connected to each of the nodes in layer 2. In layer 2, each node contains a whole set of weighted features. The same goes to layer 3, where every nodes in layer 2 are connected to layer 3’s nodes. The result is we obtain a complicated feature map that contain exponential and multiplication between features.

The above example has only one output node, and it is a two-class classification problem. The output takes the value between 1 and 0, indicating the probability of the input data being the two classes respectively. While for multiply classes, multiply output nodes are needed. Each representing the probability of the input data to be a certain class.

Artificial neural network works very well with complicated input feature, such as image data. As artificial neural network is able to structure complex features' map equation, and also learn the weight of each nodes by training. To learn the weight of each nodes, we need to calculate the cost function first. The cost function of a neural network is the sum of cost of each output nodes. The equation is as follow:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

(Andrew, 2016)

Where K is the total number of classes for the neural network. The other part of the cost function is just the same as the logistics regression.

Neural Network is a commonly used algorithm for pattern recognition in different researches. In the research of Henry A. Rowley, Shumeet Baluja and Takeo Kanade, paper "Neural Network-Based Face Detection", a neural network-based upright frontal face detection system. The paper proposed a simply program for the recognition, a neural network that examined small windows of an image, and decided whether each window contains a face. The neural network itself was trained with positive and negative face examples. The features fed into the neural network were the pixel intensity. The image size of 20x20 pixels were used in the training. In order to obtain more data, fifteen face examples are generated for each original image, by rotating the image, scaling up and down, translating and mirroring. Similar approach was applied to the negative data to generate a massive amount of training data. One of the difficulty of face recognition is the algorithm often output false positive prediction. To reduce the number of false positives, two methods were used in this research. First of all, the threshold of face recognition was increased. So the neural network only classified the example as a face with high probability. (or in other word, higher certainty.) The second method was to apply multiple networks, and arbitrate between their output to produce to final decision. According to the authors, "*Each network is trained in a similar manner, but with random initial weights, random initial nonface images, and permutations of the order of presentation of the scenery images.*" (Rowley, Baluja and Kanade, 1998) The structure of the network is illustrated by the following diagram.

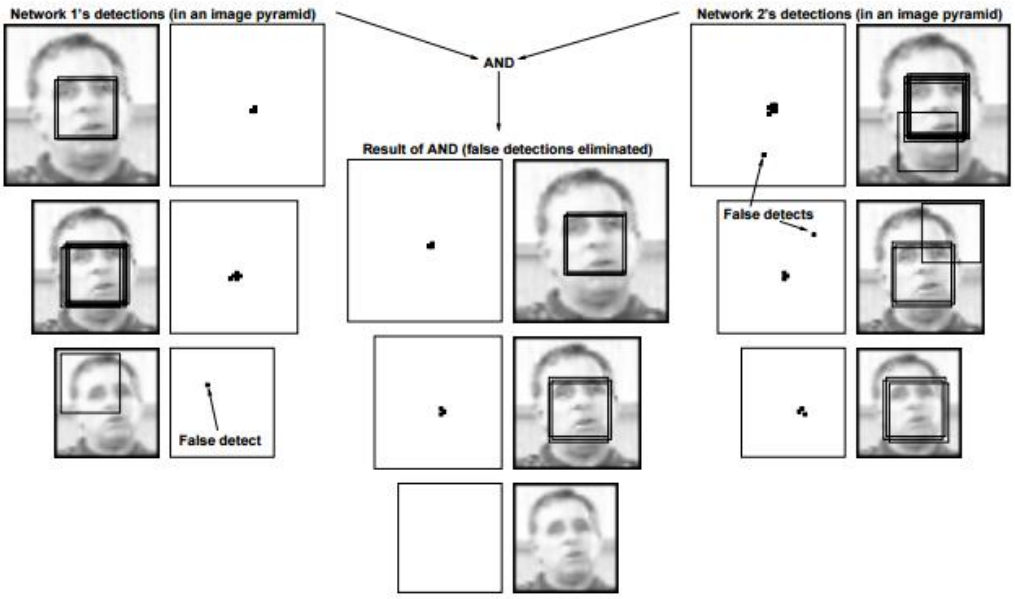


Figure 7: ANDing together the outputs from two networks over different positions and scales can improve detection accuracy.

Fig 14. Multiply neural network recognition (Rowley, Baluja and Kanade, 1998)

The resulting algorithm could detect between 77.9% to 90.3% of faces in a set of 130 test images, on a wide variety of faces and unconstrained background. The faster version of the system can process a 320x240 pixel image in 2 to 4 seconds on a 200MHz R4400 SGI Indigo2. This research shows that neural network is an effective algorithm for training a pattern recognition program.

Fig 15. Result from "Neural Network-Based Face Detection" (Rowley, Baluja and Kanade, 1998)

2.2.1.1 Convolutional Neural Network

A convolutional neural network is a kind of neural network. Similar to the neural network mentioned above, a convolutional neural network is made up of many neurons that have learnable weights and parameters, each neuron in the network receives input from some other neurons and output to some other neurons. Comparing with simply neural network, convolutional neural network assume that the input data are images, and have some special structures that cater image data. Therefore, convolutional neural network is often used for rapid image-based classification training.

One of the challenges of image classification is the large amount of training features. Use an image of 32 x 32 x 3 as an example, the size of input features is 3072. This scale up really quickly when the image size increase. Therefore, in some neural network image classifier, the input features are often compressed using different method. While convolutional neural network proposes another way to limit the number of parameters.

A general definition for convolutional neural network, given in the course CS231n Convolutional Neural Networks for Visual Recognition by Stanford University, is that *"A ConvNet is made up of Layers. Every Layer has a simple API: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.".* A convolutional neural network contain a lot of different layer that perform different function to the 3D volume input, but eventually converting them into another 3D volume output, that could be fed into another layer. The three main layers on a convolutional neural network are convolutional layer, pooling layer and fully-connected layer.

a. Convolutional layer

A convolutional layer is the core of the convolutional neural network. The convolutional layer take filters from the image. A filter is a small 3D volume from the training image, say 5 x 5 x 3, and slide the image to obtain all filter from any position of the image. The layer then compute a 2-dimensional activation map that gives the responses of that filter at every spatial position.

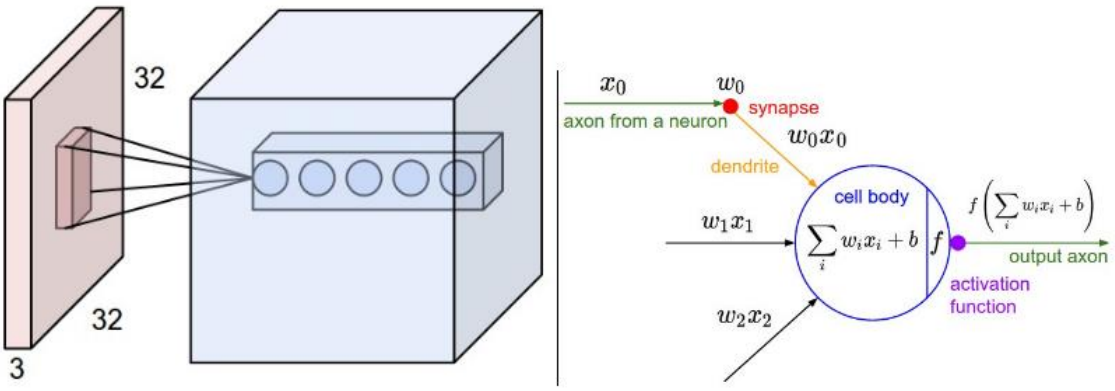


Fig 16. Visual representation of a convolutional layer (left) and its mathematical representation

(retrieved from CS231n Convolutional Neural Networks)

The convolutional layer take four hyperparameters, which refers to the parameters not from learning but are defined by us. The hyperparameters are the number of filters K, which

determine how many sub-images we want to obtain from the original image. The spatial extent F, which refers to the size of each filter. The stride S, which refers to the step to take between each concessive filters, and the amount of zero padding P, which define the number of zero to use for replacing empty space. With the above hyperparameter defined, the convolutional layer accepted a volume of size W_1 x H_1 x D_1, and return a volume of size W_2 x H_2 x D_2, where:

$W_2=(W_1-F+2P)/S+1$

$H_2=(H_1-F+2P)/S+1$

$D_2=K$

(CS231n Convolutional Neural Networks)

After the convolutional layer, we need an activation layer to calculate the dot product of the "convoluted" features. The activation layer contains an activation function, which is a function that convert the input data for some other format of output data for computational feasibility. For example, the use of sigmoid activated function convert any input data into output data with range of 0 and 1 for class prediction. The use of activated function is proved to improve the performance of the training. For the intermediary layer in a neural network, the rectified linear unit, or Relu in short, is used for the activation. The Relu function is

$$f(x) = \max(0, x)$$

Where x is the input feature from previous layer. The Relu layer simply convert all negative in the input feature to 0, and keep all the non-negative features. As what we are really looking for in training the neural network is the critical features that determine the class of the object, for example, edge, points or shape, we could ignore less important features by applying the Relu function.

b. Pooling layer

The pooling layer is a method to limit the feature size by compressing the input feature. The pooling layer generally take part of the input, and dispose the other to minimize the feature size. The pooling technique mostly used in convolutional neural network is the Max Pooling. Max pooling take a hyperparameter, the filter to perform max pooling. It would then select the largest value from that filter and dispose the rest. For example, if we select max pooling filter to be 2 x 2, the max pooling layer would take the largest value from each 2 x 2 volume from the input layer, and then return the largest value. Therefore, the output volume would be 4 times less than the input volume.

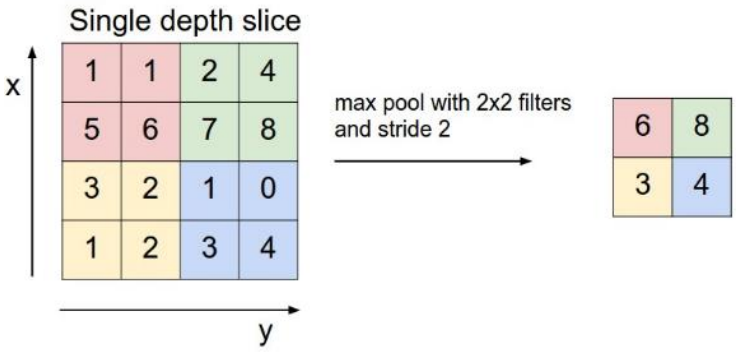


Fig 17. Visual representation of a max pooling for 2D-volume

(retrieved from CS231n Convolutional Neural Networks)

c. Fully Connected Layer

Finally, the fully connected layer performs the actual learning of the features by running the normal neural network where all nodes are connected. The fully connected layer take 2D input features. Therefore, the 3D volume output from the pooling layer should be first converted to 2D by a flatten layer. The flatten layer is simply algorithm that turn the whole 3D volume into 2D. Then we could train the algorithm just as we would in a normal neural network.

Generally, multiply layer of convolutional layer, pooling layer would be used before the fully connected layer. Some of the construction in convolutional layers are proven to be more effective by numerous of experiment. For example, the Canadian Institute for Advanced Research (CIFAR) has been working on image classification using convolutional neural network. In the paper of Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, the use of raw image for convolutional neural network training for 10 classes and 100 classes classification was proposed. The 10 classes recognition of CIFAR, also known as CIFAR-10, was developed using convolutional neural network and around 5000 images training data for each class. In the research, the training data was first processed with ZCA whitening to eliminate the lighting variation, then fed into the convolutional neural network. The network structure used by CIFAR-10 is as follow.

2 * (Convolution layer → Relu Activation layer → Convolution layer → Relu Activation layer → Max Pooling layer → Dropout layer) → Flatten layer → Fully-connected layer → Relu Activation layer → Dropout layer → Fully-connected layer → Softmax activation layer

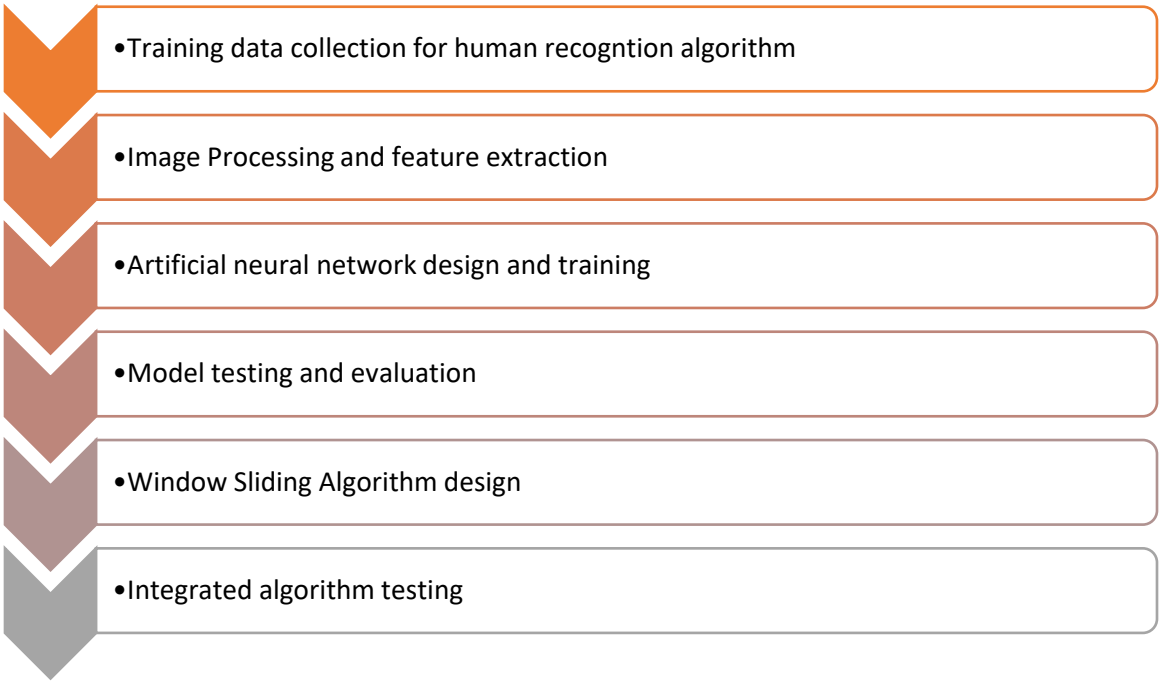
The whole structure consists of 8000 visible units and 20,000 hidden units. The dropout layer applied random drop out of data in each iteration, hence avoiding the chance of overfitting. This structure was proven to provide the best result for small image pattern recognition. Therefore, I would use this model as a reference when designing my convolutional neural network model.

Chapter 3: Research Methodology

In this chapter, I would discuss the methodology used in this project in details, which consists of collection of data, use of programming language, approach to construct the neural network and window sliding algorithm. The reason behind using the method and approach would be explained.

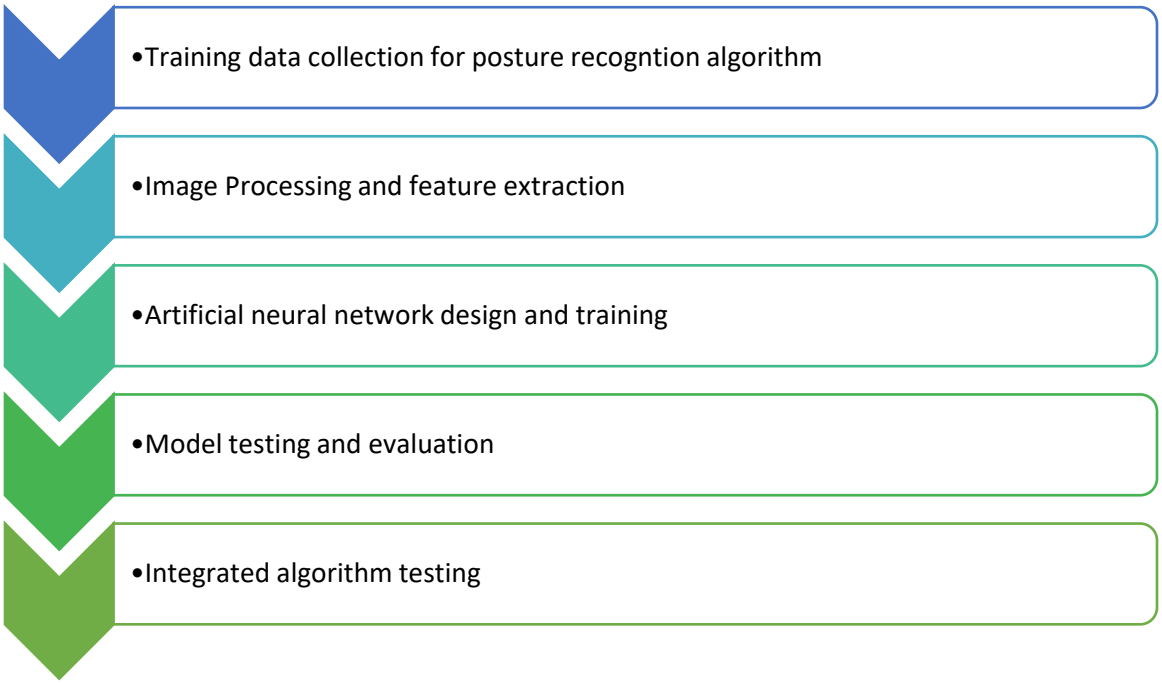
3.1 Project Approach

This project has two different parts, the development of human recognition algorithm, and the development of posture recognition algorithm. These two algorithms have similar approach, and the methods used in one could be applied in the other. The general approach for the posture recognition algorithm could be summarize with the following diagram.



The first step of the training is to collect training data for the algorithm. This is the most important part to build an accurate classification algorithm, as the performance of the algorithm largely depends on the training data fed into the algorithm. The next step is to process the image data and extract the feature from the image. This step depends on the format of the original image, and what goal we want to achieve at the end of the training. And next, the artificial neural network should be designed and trained with the training data. Then we would evaluate the trained model using different methods and parameter. Then, the window sliding algorithm that is used in pair with the human recognition algorithm should be developed. Finally, the integrated algorithm should be tested.

For the posture recognition algorithm, the general approach is basically the same. As it is planned to use artificial neural network for both recognition, so most of the codes and models developed in the human recognition part could be reused. The approach could be summarized in the following diagram.



The only differences between the two algorithms are the training data used and whether the window sliding algorithm exist. As mentioned in previous chapter, the human recognition algorithm is required to spot human from a whole image, therefore, the window sliding algorithm is required to move from frame to frame. For the posture recognition algorithm, the input data is fetched from the human recognition algorithm. Therefore, they are already in format that could be processed.

As the two recognitions are similar in nature, the experience and lessons learnt from the development of the first algorithm, could be effectively used in the second one to avoid making the same mistakes and more effective development.

3.2 Data Collection

There are two mean to collect training data for the algorithm. Either collect the data from existing dataset of open-sourced project, or to manually collect training data by the mean of taking photo, using search engine and so on. The data collection phrase, in most of the machine learning research, is the most time-consuming task and require a lot of human resources. As there is time constraint in this final year project and there was only me to collect the data, the preferable way for data collection is by using existing dataset online. However, as some training data may not be available in those dataset, manual collection is still necessary.

3.2.1 INRIA Person Dataset

One of the most popular dataset for human recognition is the INRIA Person Dataset. The INRIA stands for French Institute for Research in Computer Science and Automation. The dataset, as its name implied, contains up-straight human image and non-human image. The dataset was collected as part of research work for detection of upright people in images and video. The purpose of this dataset is very similar with my recognition algorithm. Therefore, it is a good dataset to use. The sources of the dataset, as mentioned in the INRIA Person dataset website, are:

- *Images from GRAZ 01 dataset, though annotation files are completely new.*
- *Images from personal digital image collections taken over a long time period. Usually the original positive images were of very high resolution (approx. 2592x1944 pixels), so we have cropped these images to highlight persons. Many people are bystanders taken from the backgrounds of these input photos, so ideally there is no particular bias in their pose.*
- *Few of images are taken from the web using google images.*

(Retrieved from INRIA Person dataset website)
The dataset is open sources to public and aims to help the development of other similar machine learning algorithm. The dataset contains upright persons with person height > 100 pixels in each image. In addition to the positive image data, the dataset also contains negative training data, which is street view, room view and non-human object. This INRIA dataset was used for their research in CVPR 2005 paper “Histograms of Oriented Gradients for Human Detection” and “Finding People in Images and Videos”.

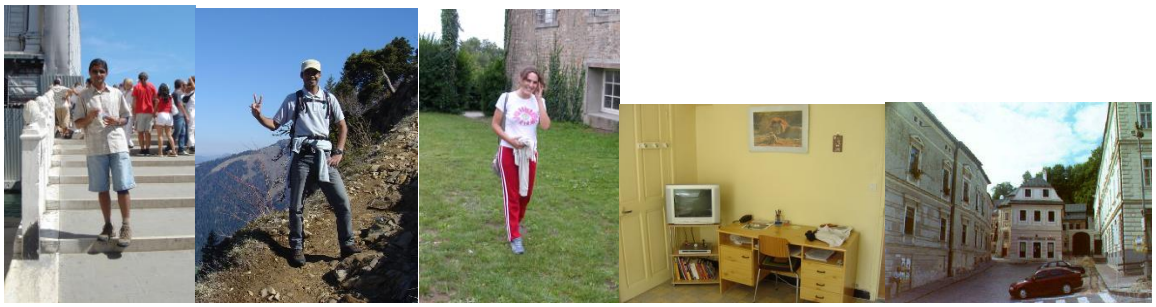


Fig 18. Positive training example (Left 3) and negative training example (Right 2) (INRIA Person Dataset)

The data inside the dataset was stored in PNG format. In addition, the data was pre-separated into training set and testing set. A training set is the set of data that the algorithm will learn from, and the testing set is used to test whether the algorithm perform well for new data. A more detail explanation of training set and testing set would be given in below section. In total, the INRIA Person Dataset give an amount of 3542 positive data, and 1213 negative data. The negative data has smaller amount as the actual negative data use in the algorithm should be the window inside the original negative image, not the whole image. Depending on how many windows we want, we could generate different number of negative data from that 1213 original image.

3.2.2 Manual Collection

While the INRIA Person Dataset provides a good amount of training data for human recognition, we still need to collect data for the posture recognition algorithm. As the interested postures are weapon holding and surrender, we need to collect positive image examples for weapon holding and surrender, and also negative image examples for non-weapon holding or surrender. Sadly, these two type of images are uncommon on the internet. There was no previously available dataset for similar problem. Most of the posture recognition dataset offer common posture, such as saying hello, shaking hand and walking. However, most of these datasets only provided video as data. Processing these videos and extract the data would be a lot of work. Therefore, it was decided that the posture training data should be collected manual with camera. After the getting the original image, the data should also be cropped into similar format with the human recognition data.

3.3 Programming Language

The programming language to use in the project is also crucial. Although the entire artificial neural network could be written in very low level language such as C and C++, it is better to use higher level language or software to write them because of the complexity in constructing the network with low level language. Two languages are selected due to their ability to process large amount of data and extensive library support. Those two languages are MATLAB/Octave and Python.

3.3.1 MATLAB/Octave

MATLAB is the language of technical computing. Being able to represent and compute number in matrix, MATLAB speed up a lot of engineering and scientific computation. The additional feature of MATLAB, such as built-in graphic, mathematical toolboxes and more also facilitate the computation. While Octave is a similar programming language that are designed just like MATLAB, but a free distribution. Octave has high compatibility with Matlb, such as similar syntax, compatible file and so on. However, being a free distribution, Octave does not offer similar toolbox feature MATLAB does.

Fig 19. MATLAB (left) and Octave (right) logos

3.3.2 Python

Python, in contrast, is a high-level programming language for more general-purpose. It was first released at 20 February 1991, and was designed to be a cross-platform interpreted language. Since development, Python has gain a lot of popularity from the programming community. New versions of Python are being introduced frequently and different open-source library was wrote for supplementary function. The version of Python used in this final year project was Python 3.6. Python itself provide some official libraries that help programmer to facilitate development. Some of these libraries are used in the code. In addition, a lot of open-sourced Python libraries were used for the program and were the program dependencies. These libraries are in MIT license. According to the MIT license, it is stated that:

> *"Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:*
>
> *The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software."*[2]

Therefore, this statement would also be included in my final program. In following, I would briefly introduce these libraries and their purpose in this project.

3.3.2.1 Pillow

Pillow is a Python Imaging Library (PIL). According to the author of Pillow, the goal is "to foster and support active development of PIL". Pillow provides a lot of handy function for image handling in Python, adds image processing capabilities. Some of the most common usage of Pillow include image archival, image batch processing and so on. In my project, Pillow was mainly used for image batch processing. For example, format conversion, image cropping, image resizing and so on.

3.3.2.2 NumPy/SciPy

SciPy is a Python-based ecosystem of open-source software for mathematics, science and engineering. [3] It introduces a lot of useful Python libraries that could accelerate calculation with matrix and big number. NumPy and SciPy are two of these libraries. NumPy performs N-dimensional array computation and manipulation, makes it an essential package in many

[2] Retrieved from https://opensource.org/licenses/MIT
[3] Retrieved from https://www.scipy.org/

scientific Python functions or libraries. While SciPy offers a lot of fundamental and efficient numerical routines, such as integration and optimization. NumPy and SciPy were the essential libraries in this project, as they help to convert image object into N-dimensional array, and allow effective storage and computation of these array.

3.3.2.3 Keras

Keras is a deep learning library of Python that provides a lot of high level neural network API. This is the most essential libraries in the whole project. Keras was written in Python and was designed to run on top of TensorFlow or Theano, which are two very popular machine learning libraries in Python. Keras even add another layer of simplicity in building neural network by providing high level function that interact with the TensorFlow or Theano backend. In this project, the TensorFlow backend was chosen. However, all codes were written with Keras API.

3.3.2.4 OpenCV-Python

OpenCV-Python is another powerful computer vision and image library for Python. OpenCV-Python provides a lot of API with interacting with image, such as image processing, face recognition and so on. However, as the aim of the project is to develop our own recognition algorithm with neural network, the role of OpenCV-Python was only to fetch image from web camera attached to the computer.

3.3.2.5 CX-Freeze

Finally, CX-Freeze is a library that freeze Python scripts into executables that run in different platform. The aim of using CX-Freeze is to create an executable that could run without Python and all libraries. As for a Python program, all dependencies (the libraries) should be installed in the local device in order for the program to run. CX-Freeze create a file that contain all the dependencies of the program, and also the program itself. This allow non-Python user to run the Python program without Python or any libraries installed in their computer.

3.4 Neural Network Construction Trial

In this chapter, I would discuss on the methodology in constructing a neural network. There are many different ways to construct a neural network, but the choice of programming language and software greatly affect the time of development. For example, a neural network could be constructed using low level programming level such as C++, but it will cost a lot of time and

resources. In chapter, I would discuss on some of the method I tried to construct a neural network and the reason I select one from the others.

3.4.1 Using Octave

My first trial in constructing a neural network was with Octave. As discussed before, Octave was a programming language for scientific and mathematic computation similar with MATLAB. As a language designed for mathematic computation, Octave support matrix computation. Making it suitable for neural network as there are lots of matrix computation involved. In addition, Octave is an object-oriented program, where functions could be treated as object and called in there function. To make the program more readable and maintainable, it is a common practice to use different files to store different functions, and call them when needed in the main function. In a neural network, the function involved are:

1. Sigmoid activated function
2. Cost function
3. Gradient function
4. Minimizer function

These functions are reusable and should be separated from the main program, and they should output correctly given the input value. For the sigmoid activated function, it is very simple as Octave already support matrix-based calculation. We could express the function as:

```
function g = sigmoid(z)
%SIGMOID Compute sigmoid functoon
%   J = SIGMOID(z) computes the sigmoid of z.

g = 1.0 ./ (1.0 + exp(-z));
end
```

The syntax of Octave is read like this, the function return the value g with the input parameter z. This simple one line function return the sigmoid activated value of z. As Octave support matrix-based calculation, we could simply plugin z into the function directly. Octave would detect it is a matrix, and perform item-wise calculation. At the end, it would return g, a matrix with same dimension with z. This sigmoid function could be called anytime from the main function and should return the correct value.

While for the cost function, we need to write a program that could calculate the cost of our network. Generally, it is a good idea to decide a cost function that could fit in any layout of network. However, as we are not planning to reuse the cost function, so we could focus on our own neural network and design a cost function that only calculate it cost. The development of such function is much easier as we could define variable as it in the cost function. Similar with the sigmoid activated function, the cost function return the cost from the input parameters. To

further simplify the development process, it was designed to integrate the gradient checking function into the cost function program. As these two require similar input parameters. Therefore, the final cost function program was:

```
function [J grad] = cost_function(all_theta,X,Y,
                                  input_layer_size,hidden_layer_size,
                                  num_labels,lambda)
```

Where J and grad represents the cost and gradient of the function respectively, and the input parameters are the thing that we need to fed into the program. “All_theta” is the value of all parameters in a single row array for the neural network, X and Y are the input training example features and classes, “input_layer_size” and “hidden_layer_size” are numbers that store the size of the input layer and hidden layer respectively, “num_labels” is a number that store the number of output classes, and finally lambda is a constant used in regularization.

For the cost calculating part of the function, the program simply preforms a forward propagation, feeding the features X layer by layer and calculate the prediction class. From that, it then uses the output class Y to calculate the cost at the current setting.

```
%% Unrolling theta
theta1 = reshape(all_theta(1:hidden_layer_size * (input_layer_size + 1)),
                           hidden_layer_size, (input_layer_size + 1));
theta2 = reshape(all_theta((1 + (hidden_layer_size * (input_layer_size + 1)))
                           :end), num_labels, (hidden_layer_size + 1));

%% Define variable
m = size(X, 1);
theta1_grad = zeros(size(theta1));
theta2_grad = zeros(size(theta2));

%% Forward prop
A1 = [ones(m,1),X];
Z2 = A1*theta1';
A2 = [ones(m,1),sigmoid(Z2)];
Z3 = A2*theta2';
A3 = sigmoid(Z3);

%% Cost function without regularization
J = 0;
J = J + sum(-Y .* log(A3) -(1-Y).*log(1-A3));
J = 1/m * J;

%% Cost function with regularization
reg = 0;
reg = reg + lambda/(2*m) *  (sum(sum(theta1(:,2:size(theta1,2)).^2))
      + sum(sum(theta2(:,2:size(theta2,2)).^2)));
J = J + reg;
```

The first two lines of the program unroll the theta into their correct shape. As they are compressed as a single array when feeding into the function. Then, after defining the variables, the cost function performs forward propagation using input feature X. Also worth mentioning that the X here is the matrix of all training example features. Therefore Octave perform the forward propagation of all examples by once. After getting the hypothesis value (prediction) of

all training examples, it then uses Y, the correct input example classes to calculate the cost function J. Finally, a regulation term to prevent over-fitting is added to the cost function for the parameter theta.

While the gradient part of the function use the technic of backward propagation to find the gradient of the cost function. As it is out of the scope of this discussion, I will not go through how the backward propagation work. Basically, by computing the error of each layer, we could then calculate the gradient of the cost function with respect to theta using those errors. The program for the backward propagation is:

```
%Back prop
err3 = A3 -  Y;
temp = err3*theta2;
err2 = temp(:,2:end).*sigmoidGradient(Z2);
theta1_grad = 1/m * (theta1_grad + err2' * A1);
theta2_grad = 1/m  * (theta2_grad + err3' * A2);

% theta1_num_grad = computeNumericalGradient(J, theta1)
% theta2_num_grad = computeNumericalGradient(J, theta2)

grad = [theta1_grad(:) ; theta2_grad(:)];

end
```

This few lines of code give us the gradient of the cost function. Now, with both the cost function and gradient, we could use minimizing function to train the neural network.

There are a lot of open-sourced libraries in Octave that provide optimizing function. One of them called the “fcmincg”. Fcmincg is an optimizing algorithm that work similar to gradient descent, but instead of choosing the step size, which is the amount of change in parameter manually, Fcmincg automatically determine the step size according to the size of the gradient and how close is the parameter to optimal. The step size is large at the beginning of the training, and gradually decreases when the parameter convergent to the optimal value. Therefore, the function could reach convergence faster.

After combining everything together, the algorithm was trained with 2240 positive example (image with human) and 2202 negative example (image without human) in a three layers’ neural network, with a hidden layer of 100 nodes. Fmincg was used for minimizing the cost function. After 50 iterations, the algorithm achieved an accuracy of 89% and 87.8% for recognizing the training data and testing data correctly.

```
Command Window
The percentage of correctness for training data is
 89.004
The percentage of correctness for testing data is
 87.815
>>
```

Fig 20. Training result of the Octave neural network

This seems to be a good result on paper. However, after testing with some real-life image, it is found that the algorithm perform poorly. One of the reason is that when converting the image into black and white, we are losing almost 75% of the image details. Therefore the algorithm can't effectively detect human. The reason that it performs well in the training data is due to over fitting and the images in the training examples are similar. To develop a more accurate algorithm, convolutional neural network should be used. However, it is very difficult to construct a convolutional neural network inside Octave, as it contains a lot of layers and functions. Instead, a higher-level library in Python was used for the final development of the convolutional neural network.

3.4.2 Using Keras

After the trial with Octave, it was designed to seek another way to construct a neural network. I found that there were different high level Python libraries that provide neural network API. For example, TensorFlow and Theano. These two are the common libraries that allow user to construct a neural network quickly. They both make use of the Multi-dimensional arrays computation with NumPy and SciPy. Both libraries were designed for neural network deep learning and therefore offer a lot of high level deep learning API, such as layer configuration. Taking one step further, a library built on these two libraries, Keras, was used to construct the convolutional neural network.

As mentioned in the previous chapter, Keras offers a lot of neural network API and functions. The basic of Keras is the Model class. The model is the backbone of the neural network, and different functions could be called on the model. For example, adding training layer, train the model, testing and so on. After a few trials with Keras, it is decided that the neural network should be constructed using Keras. The detail will be discussed in the implementation chapter.

3.4.3 Result Evaluation

Another important part of the project is to evaluate the performance of the algorithm. As our goal is to develop a real-time recognition algorithm for anti-robbery detection, the accuracy of the algorithm is one of the critical factor for success. Generally, the performance of the algorithm could be evaluated using training set and testing set error, and by evaluation the bias and variance related with the algorithm.

3.4.3.1 Training Set and Testing Set error

In an algorithm training, two different datasets should be used, the training set and testing set. The training set contains data that would be used to train the algorithm, while the testing set contains data that are not exposed to the algorithm during training. The reason of using a

testing set is to prevent overfitting. The training set error tell us how accuracy the algorithm performs on the training data, or in other word how close is the prediction. A low accuracy indicates that the algorithm fail to fit the training set. We call this underfitting. However, a high accuracy in training set not necessarily mean that the algorithm is performing well, as the parameters could have just overfit the training data. Overfitting refers to the situation where the algorithm is too fit for the training data, and fail to perform well on data outsides of the training set. Use a two-dimension classification as an example, an underfitting algorithm may be a linear function that miss some of the classes. While an overfitting one may be a complex polynomial that try to fit all examples.

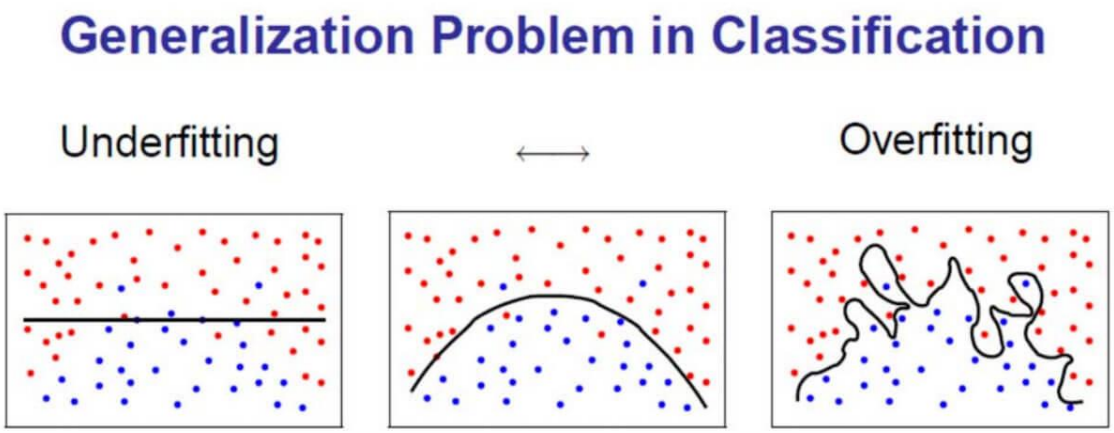


Fig 21. Example of overfitting and underfitting [4]

In the case of neural network, it is very common to see an overfitting algorithm. As a neural network produce very high degree polynomial, to achieve the lowest cost, the algorithm could simply train the parameters to overfit the training set. For a small training set, it is even possible to achieve 100% accuracy due to overfitting. Therefore, to know the true performance of the algorithm, we introduce another dataset, the testing set. The testing set was not seen by the algorithm in the training, and its accuracy represent how well the algorithm could generalize the parameters. For a higher dimension machine learning problem, we can't plot the graph to see whether the algorithm has overfit or underfit the data. Instead, we would try to find the bias and variance of the algorithm.

3.4.3.2 Bias and Variance

Bias and variance are used to measure the performance of a machine learning algorithm. If an algorithm is said to have high bias, it means it is underfitting the training data. While if an algorithm is said to have high variance, it means it is overfitting the training data. We could

[4] Retrieved from https://tomrobertshaw.net/2015/12/introduction-to-machine-learning-with-naive-bayes/

determine whether the algorithm suffers from these two problems from plotting the cost of it against the degree of polynomial, as shown in the following figure.

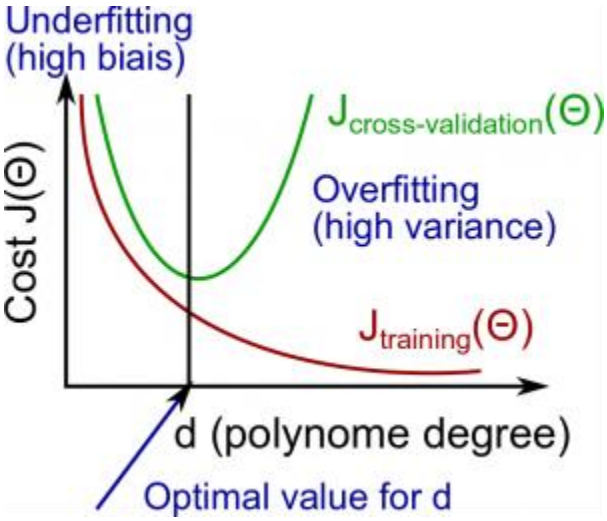


Fig 22. Graph of algorithm cost against polynomial degree (Andrew, 2016)

When the degree of polynomial is low, both the training error and cross-validation (test set) errors are high. This shows the algorithm is in bias. While when the degree of polynomial is very high, the training set error is low, but the cross-validation error is high. This represents a high variance. By knowing whether the algorithm is in bias or variance, we could take corresponding actions to improve its performance.

Algorithm in bias:

- Increase the number of training data
- Increase the degree of polynomial
- Decrease regulation
- Increase the number of features
- Increase the number of iteration

Algorithm in variance:

- Decrease the number of feature
- Decrease the degree of polynomial
- Increase regulation

3.4.4 Sliding Window

Another important part of the whole project is the sliding window algorithm. The sliding window algorithm is not a machine learning algorithm, but simply a program that slide the image frame by frame and run the trained classifier to identify human image. As Keras is chosen to be the

library, the sliding window program should also be written in Python for compatibility. Basically, the idea is very simply. The program first read an image file from the folder or web camera, then it “slide” the image frame by frame on predefined-size and run the classifier. If not human is found, the program will proceed to the next slide. Otherwise, it will save the frame contain human and proceed.

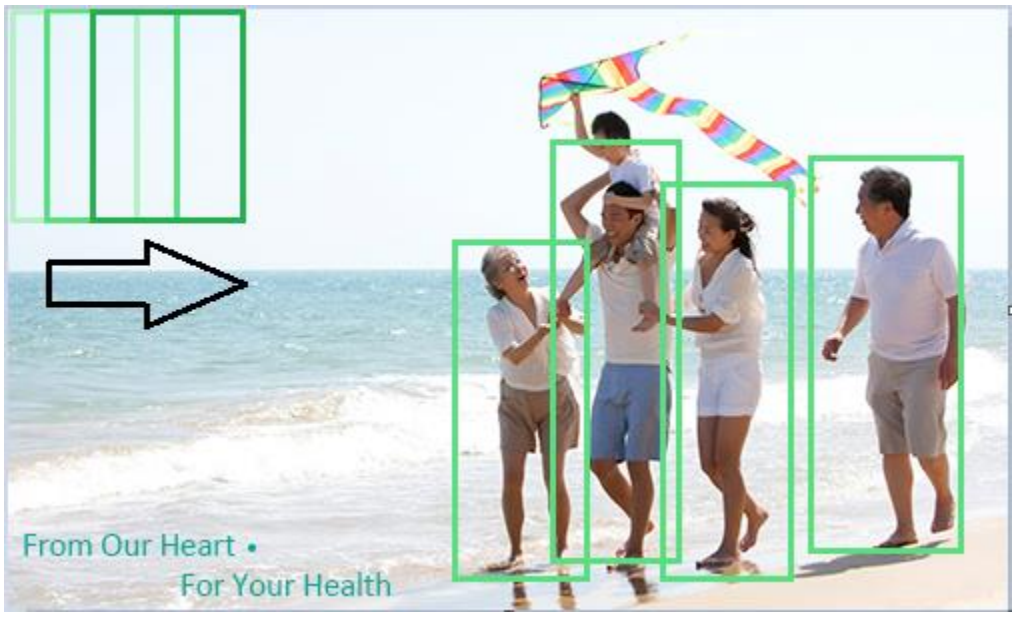


Fig 23. Using sliding window to find human

The window sliding program has a lot of predefined parameters that affect its performance. For example, the size of thee frame used, the step size, the ratio of the frame and so on. In order to fit different image, user should be able to define their own window sliding setting. A more detail discussion about the window sliding program would be given in chapter 4 Implementation.

Chapter 4: Project Implementation

In this chapter, I will discuss in detail the technical implementation of the whole project. In previous chapter, I had discussed the concept of machine learning and methodology used in this project. This chapter will mainly focus on the detail of the implementation, such as formatting training data, writing Python code, technical concern and difficulties and so on. As the two recognition algorithms, the human recognition algorithm and posture recognition algorithm, were developed separately, and for the posture recognition algorithm, most of the technic and configuration were same as the human recognition algorithm. Therefore I will mainly focus on the human recognition algorithm.

4.1 Human Recognition

The human recognition algorithm is the first part of the project. The human recognition algorithm identify human from an image. As discussed in the methodology, the human recognition algorithm would be a convolutional neural network trained with positive and negative data of human image. The features of the image data would be the pixel of the image. There are three main phrases in the human recognition, the data collection phrase, convolutional neural network construction phrase and sliding window program phrase. The success of the whole algorithm highly depends on whether these three phrases are done correctly.

4.1.1 Data Collection

The data used in the human recognition algorithm was sourced from the INRIA Person Dataset. As discussed in previous chapters, this dataset features upright human image and non-human image. This images, however, were in different dimension and need to be first proceeded. Therefore, Python is used to standardize these images first. After that, the image data is converted into matrix format for calculation. In this chapter, the Python code for standardization and conversion of images would be explained in details. In addition, some of the training data is not suitable for my human recognition algorithm. As my algorithm aims to provide input for the posture recognition algorithm, the output image for the human recognition should be upright human inside a box. And therefore, the training data of the human recognition algorithm should also be upright human inside a box. Manually selection is required for choosing the right image.

4.1.1.1 Data Processing

The first step of the data standardization process is to manually select the suitable training data for the human recognition algorithm. As the INRIA Person Dataset was created for human

detection in an image, it contains human in different position, overlapping, or multiply human inside one image. The dataset could effectively train the algorithm to recognize human-like object in different circumstance. Those training example, however, are not suitable for our human recognition algorithm as it will cause the algorithm to produce a lot of false output. The only output we want is a frame containing only a human, if there are multiply human inside a frame, it would be difficult to train a posture recognition algorithm to recognize it. Moreover, the dataset also features image with different sizes and only show in parts. This is also not suitable for our program due to the same reason. Therefore, before actually standardizing the image, the dataset was gone through manually.

Fig 24. Suitable positive training images (left 2) and not suitable positive training images (right 2)

To take one step further, as we want to prevent our algorithm from classifying the overlapping or multiply human in a single frame, those images in the positive example could be regarded as the negative example in the training. To do so, I opened two new folders, and started selecting my positive and negative example from the original positive example of the INRIA dataset. The selection process was very simple. Every time I saw a training example that didn't meet my standard for positive example, I move it to my negative example file. After went through every example, I copied the remaining images into my positive example file. After deleting some image with odd dimension, these resulted in 2,299 positive images and 1,141 negative images.

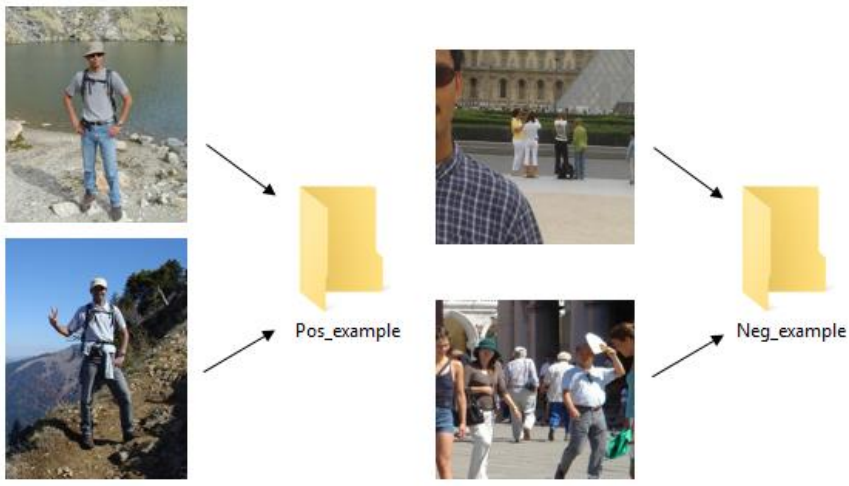


Fig 25. Constructing positive and negative example folders using INRIA dataset

After manually selected the positive example, further processing was required. The positive training data in the original INRIA dataset was in two different dimension, 70 x 134 and 96 x 160 (width x height). We need to convert all these different dimension images into standard size for our training algorithm. The final size of the images should be 30 x 60, as the trial of constructing neural network with Octave showed that larger image could be computational infeasible. 30 x 60 resolution provides enough features (pixels) for the neural network to recognize. However, before the actual resize, most of the image should be cropped first. As most of the positive human image have large padding of background around it, those are irrelevant pixels that may affect the performance of the algorithm. Therefore, the actual human inside the image should be cropped out before further process. The INRIA dataset provide annotations for the images, specifying different details of the images, including the bounding box for the human in the image. The annotations are in PASCAL Challenge format, a non-standardized format commonly used for annotation in machine learning database. The bounding box provide the top-left pixel and bottom-right pixel of the bounding box, allowing future users to crop out the human image using this annotation easily.

Fig 26. Example with 96 x 160 (left) and 70 x 134 (right) with bounding boxes

However, since I messed up the order of the training dataset by manually selecting my own positive and negative example, it was troublesome to go through the dataset again to match up the annotation with the images. An easier but effective method was used instead. As the images are all taken with similar human to background padding ratio, in other word, the “unless” background padding for all images of the same resolution are nearly the same. So once we determined how much padding from the top, left, right and bottom should be cropped away from one image, we could just applied the same rule for every other images of the same resolution. Python was used to implement this idea. The main library used in this cropping program is Pillow, which perform the cropping operation of the image. The Python libraries glob and os are also used to retrieve data from a file and get the file name. The complete program, named “image_crop.py” is shown below.

```
from PIL import Image
import glob
import os

left = 23
upper = 30
right = 73
lower = 130
box = (left, upper, right, lower)

for infile in glob.glob("*.png"):
    file, ext = os.path.splitext(infile)
    im = Image.open(infile)
    crop_area = im.crop(box)
    crop_area.save(file + "_crop.png", "png")

```

Fig 27. Screen shot from image_crop.py

The first three lines of the program call the required libraries, then it defined the size of the "cropping box". The size of the cropping box was defined by the resolution of the image. For this example, the 96 x 160 resolution was used. The cropping box was defined so it started from 23 pixels from the left, 30 pixels from the top, and went 73 pixels to the right and 130 pixels to the bottom. This result in cropping out a 50 x 100 box out of the image. The cropped image was renamed into "original image name" plus "crop.png" for identification. If the resolution changed, the value of parameters left, upper, right and lower should also be changed to locate the human in the image. The program used a for loop to find all image of PNG format in the folder, cropping the human out, and save it as a new file. This simple program performed pretty good in practice. The output images correctly cropped out the human in most examples.

Fig 28. Cropping the human pixels out of the original image

After that, the image is resized to 30 x 60 with Pillow. The program is similar, using the same libraries with that of the cropping program. The program simply do a for loop for all PNG format

image, resize them to 30 x 60, and save a new copy with a file name of “original file name” plus “_30x60.png”. The program was named resize.py.

```
1  from PIL import Image
2  import glob
3  import os
4
5  size = (30,60)
6
7  for infile in glob.glob("*.png"):
8      file, ext = os.path.splitext(infile)
9      im = Image.open(infile)
10     out = im.resize(size)
11     out.save(file + "_30x60.png", "png")
```

Fig 29. Screen shot from resize.py

After finish processing the positive training data, we need to process the negative training data. The previously selected negative training data from the original positive dataset didn’t require cropping. While for the original negative dataset, the images are full resolution image with different view, object or building. These images should not be fed into the algorithm directly. As the input for the algorithm is supposed to be small frame image, the full size negative image couldn’t help the algorithm to build up effective parameters for classifying negative window. Therefore, instead of using the full images, I wrote another program in Python to randomly select 30 x 60 windows from the negative training examples, and saving them into my negative training folder. The program, in addition to using Pillow, glob and os, also used the randint library from Python which generate random number. The logic of the program is simply, firstly, the program asks for user input for the number of windows for one single image, which will later determine the number of loop for cropping random images out of the original one.

```
6  numberOfCrop = input("number of window to crop from a image: ")
7  numberOfCrop = int(numberOfCrop)
```

Then, we need to set the limit for the random initiation. The number of the width and height of the top left corner of the cropped image should not exceed “Original image width - 30” and “Original image height - 60”. The reason is if the top left corner is larger than this limit, the resulting crop will go outside of the image, and the output image would not have a dimension of 30 x 60. Therefore, we need to first define the limit. After defining the limit, we could again perform a for loop for all PNG format image in the folder, inside the for loop, run a while loop based on the number of window required, randomly initiate the crop starting position and save the resulting images.

```
im = Image.open(infile)
width,height = im.size

width = width - 30
height = height - 60
```

Fig 30. Visual representation of the image cropping limit (left) and code to set the limit (right)

Fig 31. Getting smaller negative window frame from a negative training example

Using this method, we could potentially produce a very large amount of negative training data. However, as I don't have many positive example, I only produced 2,000 negative examples for consistency.

This random cropping program has another application. As for my human recognition program, the positive class should be an entire up-straight human only, we don't want the human recognition algorithm to recognize part of human as human. As the sliding window algorithm will scan the photo and run the recognition algorithm, if the algorithm return human output every time it see something resemble a human, we would end up will a lot of different human outputs. To prevent this, we need to train the algorithm to classify between full human and part human. To do so, I run the random crop program in the original positive image training data from INRIA dataset, and generate around 2,000 human part images.

After all these data processing, the positive and negative dataset end up having 2,299 and 5,260 images. These images should be separated into training set and testing set as mentioned. 10% of all images are selected into the test set. To randomly select the images from training set into test set, another Python library shutil was introduced. This library allows moving or copying of files from different folders. The program first get a list of total files in the folder, then use the

random number generator to select random files from the list, finally moving them to the new folder. With this random file program, I could effectively generate training set and test set for different problems.

4.1.1.2 Features Extraction

After processing the image data, the next step was to extract the features from the image. The features to use in the convolutional neural network were the pixels of the image. For a 30 x 60 RGB-coloured image, the number of features is 5,400. As each pixel is represented by three values, the red, green and blue intensity. For effective calculation and process of the features, they should be stored in the form of matrix. As during the convolutional neural network training, there would be a lot of matrix-wised computation, the features should be stored in matrix form to allow the computation. Each image was converted into a 30 x 60 x 3 matrix, the elements of the matrix are number between 0 to 255, a larger number represents a high Red/Green/Blue intensity at that pixel. For example, a (255,0,0) represents a completely red pixel, a (0,0,0) represents a black pixel and a (255,255,255) represents a white pixel. After converting all training examples, we have a matrix of size (training example size, 30, 60, 3).

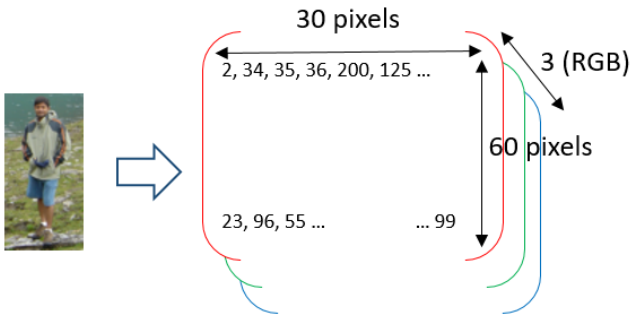


Fig 32. Converting one training example into a 30 x 60 x 3 matrix

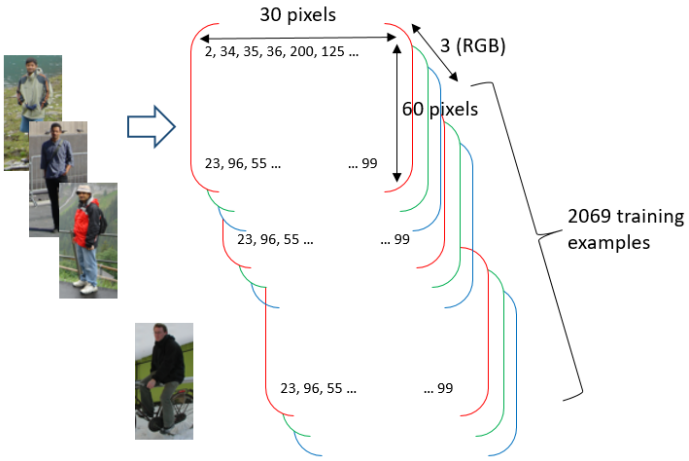


Fig 33. Converting all positive training example into a 2069 x 30 x 60 x 3 matrix

The conversion was done by the Python libraries NumPy and SciPy. As mentioned before, these two libraries offer the capability to compute and manipulate matrix. For reusability, the program was designed to convert different number of training data. The program first get two user input, the number of images in the dataset to convert, and the name of the output file. It then initializes a matrix of the size of (inputted number, 60, 30, 3). After that, the program perform a for loop for all training data with PNG format, and use a SciPy function called imread to get the pixel of the image in matrix form, and store it at the output matrix. The output was then stored in NPY format, a format used by NumPy and SciPy to store matrix. Using this format, the storage space required was minimized, and the matrix data inside could be effectively read by SciPy and NumPy. This provided us the training features as well as testing features to use in the convolutional neural network.

4.1.2 Convolutional Neural Network

After finishing the image processing and the features extraction, the convolutional neural network was constructed and trained. In this section, I will talk about the technical details on constructing the convolutional neural network, network training and testing.

4.1.2.1 Model Construction

For model construction, there are a lot of different ways to construct a convolutional neural network. A convolutional neural network consists of three main components, the convolutional layer, pooling layer and fully connected layer. There is no standard on how many layers to use, it depends on the input features number, complexity of the problem, number of output classes and so on. There is no best convolutional neural network, to find the most suitable convolutional neural network layout for the problem, a lot of trial and testing are required. Luckily, there are numerous of deep learning research using convolutional neural network, and some of the convolutional neural network layer was proven to perform better on certain problem. For my problem, I applied the layout used in CIFAR-10, and modify it based on my requirement.

The CIFAR-10 image classifier was designed for classifying images within 10 different classes, the classifier use 32 x 32 images to train the convolutional neural network. Compare to the CIFAR-100, it has a simpler structure. The CIFAR-10 structure, as mentioned in Chapter 2, was:

2 * (Convolution layer → Relu Activation layer → Convolution layer → Relu Activation layer → Max Pooling layer → Dropout layer) → Flatten layer → Fully-connected layer → Relu Activation layer → Dropout layer → Fully-connected layer → Softmax activation layer

This structure, although simpler than the CIFAR-100, is still complicated to construct manually. Luckily, Keras provides the CIFAR-10 convolutional neural network example written using Keras.

This example was modified to train my human recognition problem. In the following, I would explain the code line by line and explain how different Keras functions work.

```
11  from keras.datasets import cifar10
12  from keras.preprocessing.image import ImageDataGenerator
13  from keras.models import Sequential
14  from keras.layers import Dense, Dropout, Activation, Flatten
15  from keras.layers import Convolution2D, MaxPooling2D
16  from keras.utils import np_utils
```

Like all other Python program, the necessary libraries should be imported before using. Keras was developed in a modular structure, user should choose what libraries of Keras to import. In line 11, the cifar10 dataset was imported using Keras dataset. This library was not required for the human recognition as I used my own data. While for line 12, a Keras library for image processing was imported. The image data generator creates different alternative version of an image by using transition, whitening, rotating and other effect. I will talk about this more in the next section. While line 13 to 16 import the essential libraries for a convolutional neural network. Sequential is the core component of any neural network, it creates a platform for other layers. The dense, dropout, activation, flatten, convolution2D and maxpooling2D are the layers from the network. Dense represents the fully connected layer, convolution2D and maxpooling2D represents convolutional layer and max pooling layer for image input. Finally, the np_utils library was used to convert the output class into binary representation. If there are 10 output classes, the np_utils function converts any class to a 10 x 1 vector with 1 on the corresponding class position and 0 on the other. For example, an output with class 4 is converted into [0,0,0,1,0,0,0,0,0,0].

To construct a neural network in Keras, the model should first be defined. The model is a Python class that contains different function on deep learning. After initialized the model, the function could be called to add layer to the neural network. Initialize model in Keras is simple, with only one line of code:

```
44  model = Sequential()
```

The Sequential method create a new model class named “model”. We could then call different function on the model class to add layers on it. The first layer of the convolutional neural network is the convolutional layer. The convolutional layer, again, could be called with one single line.

```
46  model.add(Convolution2D(32, 3, 3, border_mode='same',
47                          input_shape=X_train.shape[1:]))
```

The add function add the layer to the model, in the convolutional layer, the convolution2D method was called. Inside the method, there are 5 parameters, the first 4 were the hyperparameters of the convolutional layer, specifying the output size, filter size and zero padding used respectively. While the last one specify the shape of input features to the layer. In later layers, this is not required as Keras automatically calculate it based on previous layer. The layer follow by it is the activation layer.

```
48  model.add(Activation('relu'))
```

The activation function provides keywords for different activation function. “relu” stands for rectified linear unit. Other keywords include “sigmoid” and “softmax” which stands for sigmoid and softmax function respectively. The activation layer take input from the previous layer and return output to next layer.

```
51  model.add(MaxPooling2D(pool_size=(2, 2)))
```

The max pooling 2D function creates a max pooling layer, the pool size defines the filter size of the max pooling (i.e. How many pixels to perform the max pooling on each time). The pool size take a tuple input, which should contain two number. As this max pooling function is for 2D image. Similarly, the dropout layer was defined by one single line.

```
52  model.add(Dropout(0.25))
```

The dropout layer randomly ignores some of the node in each of the training iteration. Dropout layer is an effective tool to avoid overfitting, as it prevents the node from memorizing one set of the image features and never change it parameters. The number in the dropout layer is the percentage of nodes to ignore in each iteration. For example, 0.25 represents that 25% of the nodes are ignored in each iteration. Finally, the flatten layer and fully connected layer are defined.

```
61  model.add(Flatten())
62  model.add(Dense(512))
```

The flatten layer converts the 3D output of the convolutional layer into 2D, and the dense layer represents a fully connected layer with 512 nodes. After defining all the layers for the convolutional neural network, we need to define how to train the model and what optimizer to use in the training. Keras provides support for most of the popular training model and optimizer. The setting used in the CIFAR10 classifier is:

```
69  model.compile(loss='categorical_crossentropy',
70                optimizer='rmsprop',
71                metrics=['accuracy'])
```

The loss is the cost function to use in the training. Categorical cross entropy is a cost function used for multiply classes classification, the RMSProp optimizer is a more advance optimizer comparing with gradient descent, and is frequently used in neural network optimization. And the metrics judges the performance of the model, in the CIFAR10, the accuracy was used as the parameter.

Different layouts of the convolutional neural network were tried and tested. For example, fewer layers, higher or lower dropout rate, using different optimizers and so on. It was found that the original CIFAR10 layout provided the best output for our human classifier as well. Therefore, the

layout of the convolutional neural network is the same with that of the CIFAR10. The whole network looks like this:

```
32  model = Sequential()
33
34  model.add(Convolution2D(32, 3, 3,
35                          input_shape=X_train.shape[1:]))
36  model.add(Activation('relu'))
37  model.add(Convolution2D(32, 3, 3))
38  model.add(Activation('relu'))
39  model.add(MaxPooling2D(pool_size=(2, 2)))
40  model.add(Dropout(0.5))
41
42  model.add(Convolution2D(64, 3, 3))
43  model.add(Activation('relu'))
44  model.add(Convolution2D(64, 3, 3))
45  model.add(Activation('relu'))
46  model.add(MaxPooling2D(pool_size=(2, 2)))
47  model.add(Dropout(0.25))
48
49  model.add(Flatten())
50  model.add(Dense(512))
51  model.add(Activation('relu'))
52  model.add(Dropout(0.5))
53  model.add(Dense(nb_classes))
54  model.add(Activation('sigmoid'))
```

Fig 34. Screen shot from the convolutional neural network

There are four convolutional layers in total, two max pooling and three dropout layers. For the fully connected hidden layer, 512 nodes were used. While for the training setting, the binary cross entropy was used instead of the categorical cross entropy, as we only have two classes in the human recognition, either is or not a human.

```
56  model.compile(loss='binary_crossentropy',
57                optimizer='rmsprop',
58                metrics=['accuracy'])
```

4.1.2.2 Image Data Generator

One of the function Keras offers is the image data generator. For most of the machine learning problem, a large variety of training data often help the algorithm perform better. To obtain a large number of training examples, alternative version of an image was generated using different image processing method. The keras function image data generator help to generate different image data based on the original image. After defining the processing to use in the function, the data generator was used in the algorithm training. Some of the attributes for the image data generator are:

- Featurewise center

  Set the input mean to 0 over the dataset, based on the feature

- Samplewise center

  Set the sample mean to 0

- ZCA whitening

  Apply ZCA whitening to all images

- Rotation

  Create different rotated version of images

- Zoom

  Create different zoomed in and out version of images

- Horizontal/vertical flip

  Create horizontally or vertically flipped version of images

- Width/height shift

  Create horizontally or vertically shifted version of images

The image generator was tried in the human recognition algorithm. The result would be discussed in later chapter.

4.1.2.3 Model Training

Similar with other Keras functions, we could set up the model for training with one line of code. The fit function of the model get two parameters, the input training feature “X_train” and input training classes “Y_train”. These two parameters are essential to start the training. The batch size determines how many training data to feed into the neural network during the iteration. The advantages and disadvantage of using different batch size was discussed in different type of gradient descent. The epoch is the total number of iteration to run. Generally, a larger number

of epoch result in the parameters being more close to convergence, given that the neural network was configurated correctly. While the validation data is the testing data used to check the neural network performance. This together with the training error allow us to determine whether the model suffers from bias or variance. Finally, shuffle means the training data should be shuffled in every iteration, and prevent the model from not changing it weights in different iteration.

```
model.fit(X_train, Y_train,
                batch_size=batch_size,
                nb_epoch=nb_epoch,
                validation_data=(X_test, Y_test),
                shuffle=True)
```

The parameters used in the training were 68 from batch size and 100 for epoch. The batch size was set to be 1% of the total number of training data, while the epoch is set to be 100 because no significant improvement was made after around 60 – 70 iterations. So 100 iterations were enough for the algorithm to reach convergence.

```
batch_size = 68
nb_epoch = 100
```

When running the program, it outputs the training process with this format:

```
Train on 6803 samples, validate on 756 samples
Epoch 1/100
6803/6803 [==============================] - 94s - loss: 0.5213 - acc: 0.7563 - val_loss: 0.3039 - val_acc: 0.8690
Epoch 2/100
6803/6803 [==============================] - 94s - loss: 0.3004 - acc: 0.8783 - val_loss: 0.1923 - val_acc: 0.9259
Epoch 3/100
6803/6803 [==============================] - 93s - loss: 0.1961 - acc: 0.9253 - val_loss: 0.3820 - val_acc: 0.8598
Epoch 4/100
4352/6803 [==================>...........] - ETA: 32s - loss: 0.1751 - acc: 0.9341
```

The epoch 4/100 means that it is the first iteration out of 100 iteration. While the 4352/6803 was the number of training data processed so far. The first number updated by 68 each time as the batch size was defined as 68. The ETA was the expected time of accomplishment. The loss and acc are the cost function and accuracy of the training data. These two values updated when the training proceeded. If the neural network was constructed correctly, the loss should decrease and accuracy should increase when epoch increase, as shown in the figure above. While the val_loss and val_acc are the cost function and accuracy of the testing data. As the model was not trained for the accuracy of the testing data, these two value may fluctuate when training proceed. But normally, they will slowly increase across many epochs. The epoch kept adding until 100, and the training was finished. The trained model was saved in h5 format, which is a format used by Keras to effectively rebuilt model in other programs. Training the model with the Intel i5-4210U processer, the 100 epochs took around 2.5 hours to finish.

4.1.2.4 Model Testing

The last step of the model was to test the model using training data and testing data. The trained model was stored in folder, and Keras load model function was used to load the model for testing.

```
model = load_model('human_reg_model1.h5')

loss_and_metrics_train = model.evaluate(X_train, Y_train, batch_size=32)
loss_and_metrics_test = model.evaluate(X_test, Y_test, batch_size=32)
```

The load model found the model with the name “human_reg_model1.h5” in the same folder, and construct the model based on the model setting stored in that file. The evaluate function of the model compute the cost function error and accuracy of the model. The training loss and accuracy of model 1 were 0.0107 and 99.9% respectively and the testing loss and accuracy were 0.089 and 99.34%.

```
6803/6803 [==============================] - 30s
756/756 [==============================] - 3s
train:  [0.010769995958864449, 0.9992650301337645]
test:  [0.08937027302270785, 0.9933862433862433]
```

4.1.3 Sliding Window Algorithm

After completing the model training, the next task was to build the sliding window algorithm. The sliding window algorithm scan the input image frame by frame and run the human recognition algorithm. In this section, I will go through the general logic of the program, program parameters, window sliding details and the output of the program.

4.1.3.1 General program Logic

The program logic could be separated into two parts, the general program configuration and the window sliding function. The first part load the Keras model, fetch image to process from either web camera or folder, and define the setting to use in the window sliding. While the second part performs the actual window sliding on the image, run the Keras model on each frame, and saving the frame if human is found. The whole program was decided as an infinite loop until user input the keyword ‘end’ at the first part of the program. As the Keras model loading took about 30 seconds, it is more convenience to load the model once, and keep using it by a loop.

The logic flow diagram of the general program configuration is as follow:

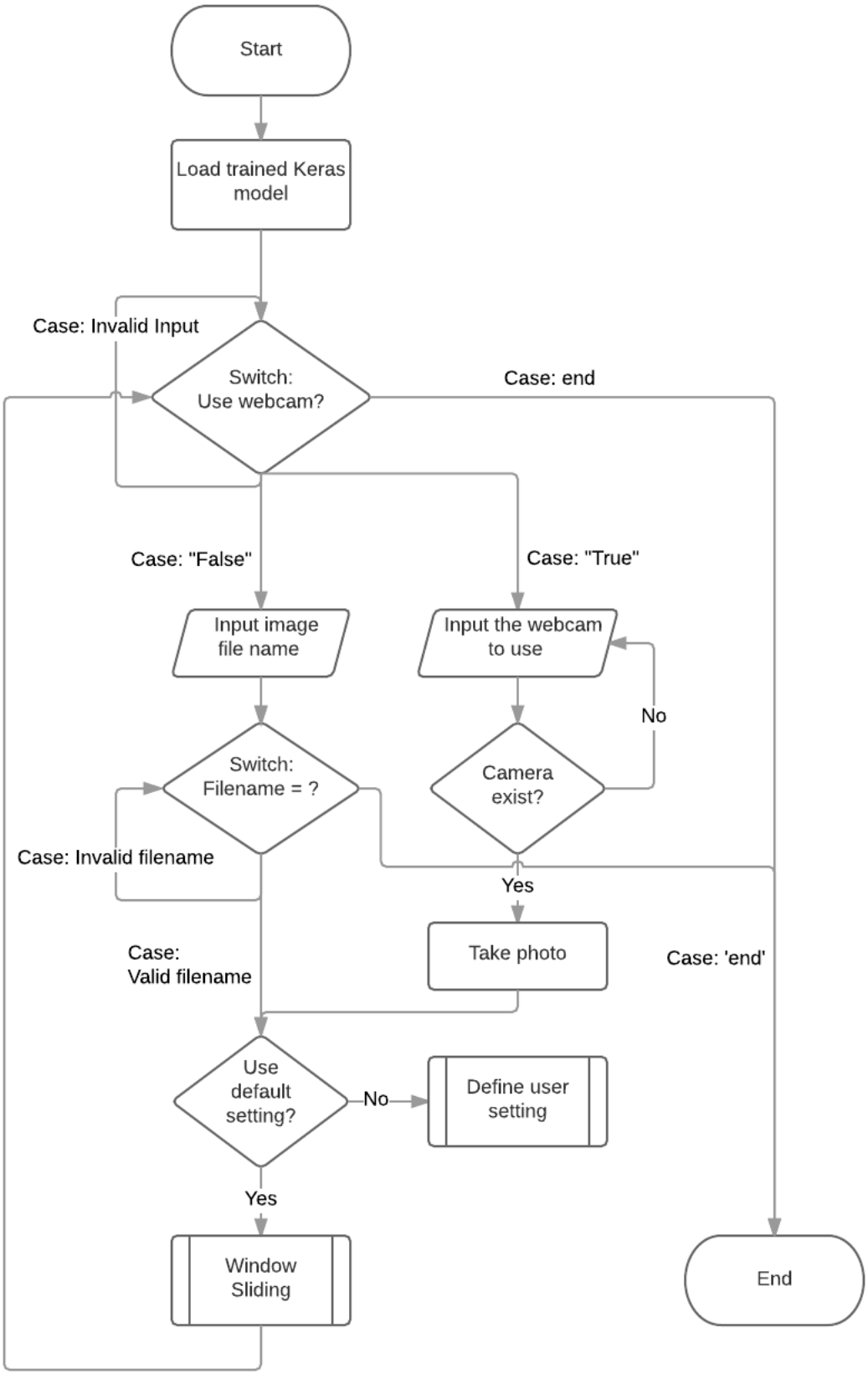


Fig 34. Logic flow chart for the general program configuration

The program folder structure is as follow:

>Sliding_window

>Input

>Output

>human_reg_model1.h5

>sliding_window_v1.py

The input and output are two folders containing the input images and output images of the algorithm, human_reg_model1.h5 is the Keras model used in the program, while the sliding_window_v1.py is the Python program for the sliding window.

The entire program started with loading the Keras model, this was done by the Keras function load model. And then it asked the user whether to use web camera to fetch the image. If the user input “end”, the program would shut down.

```
Loading Complete!
>> Use webcam to fetch image (T/F)? default False  (To end program, type 'end'):
```

If the user inputted F/t, which means true, the program would then ask for which web camera to use. If the device is only connected with one web camera, the device would be 0, which is also the default setting. Otherwise, the user could select which web camera to use (Second camera connect would be 1, third would be 2 and so on). If the web camera was not found, the user would be prompt and input a valid camera number until the program could proceed.

```
>> ERROR: The input is incorrect. Either the device does not exist or a non number is inputted
>> Select the webcam to use. Default 0:
```

After the web camera was found, the user could then take a photo by pressing any key. The image would be saved in the input folder. In another case, if the user select False, the program would ask the user to input the filename instead. The file should be in image format, and was searched in the input file. If the image format is not valid or the file is not found in the input file, the program would prompt error and ask for input again. If the user input end, the program will shut down.

```
>> Input file name, default 'window1.png' (To end program, type 'end'): someimage.png
>> ERROR: The value you inputted is not a valid file name or not exist in file,try again
>> Input file name, default 'window1.png' (To end program, type 'end'):
```

After getting the image to use for the program, the program would ask for whether to use the default setting. The default setting predefined the parameters to use in the window sliding. Otherwise, the user will need to define their only parameters. A more detail discussion on the parameters would be given in the next chapter. Finally, the window sliding algorithm would run based on the setting.

4.1.3.2 Program parameters

The program took different parameters for the window sliding algorithm. In this section, I will explain them in details, what are those parameters and their impact on the window sliding.

a. Frame size

The frame size is the size of each frame to look at in each iteration in the window sliding algorithm. To effectively scan the image and look for different sizes of human, a list of frame size is used, and the window sliding algorithm will scan through the original image use different frame size. For example, if the frame sizes of (50,100), (100,200), (150,300) are used, then the algorithm will first scan the image using 50 x 100 frame, follow by 100 x 200 frame, and lastly 150 x 300 frame. Generally, a longer list of frame size increase the running time of the program, and a smaller frame size also increase the running time of the program as there are more frame to scan through.

Fig 35. Using different frame size to scan the image

Instead of using fixed pixels to define the frame size, the frame size was defined by a percentage of the original image. For example, a frame size of (0.1,0.2,0.3,0.4) represents the algorithm will first scan through the image with frame width equal 10% of the image width, then by 20% of it, 30% and finally 40%. Comparing this to the fixed pixel, the user could set the frame size easier without needing to know the image size. The defaulted frame size was (0.1,0.2,0.3,0.4). User could define their own frame size but not using the default setting and input valid sizes.

```
>> Input the list of frame sizes, all sizes should be less than 1 (default .1,.2,.3,.4):
```

b. Frame ratio

The frame ratio is the frame height to width ratio. As human height tends to be greater than it width in a normal up-straight image, the height was set to be n times the width. The default setting of the frame ratio was 4, so it the frame width was 100 pixels after calculation, the frame height was 400 pixels. The frame ratio is essential as the algorithm sometime fail to detect human if the frame ratio is too far from the actual, as the frame can't capture the whole human in it. User could set their own ratio to fit their image.

```
>> Input the human height:width ratio (default 4):
```

c. Moving step

The moving step is how many pixels the frame move each time. A smaller moving step produce a larger number of total frame, and hence has a lower chance of missing the human. However, a smaller moving step also increase the total running time of the program, as more frames should be processed. The default moving step was 2. User could define their own moving step.

```
>> Input the moving step in pixel (default 2):
```

d. Detection delay

The detection delay is used to delay the actual saving of the image after human is found. For example, if the detection delay is 1, the window sliding algorithm would save the frame one moving step right and down. As the frame scan from top left to bottom right, the algorithm tends to recognize a human when it detected enough feature. Using figure 36 as an example, the image at the left shows the top left part of the human, but not all of it. However, for the algorithm, it contains enough features to be classified as a human. What we really want should be the image at the right, which is a little bit right and below the frame on the left. Therefore, using the delay, we obtain better human frame.

Fig 36. Possible image output without delay (left) and with delay (right)

The delay could be determined by a few trials on the same image. If it is found that the output images are always too far left and top from the actual human, the delay could be increase, and vice versa.

e. White filter

The white filter is a function that could be turned on or off. The white filter function filters all frame with more than 50% white pixels. The reason behind the white filter is because in the window sliding algorithm, when a human I found, to prevent the same human being found again with slightly different frames, the pixels where the human was should be replaced with white pixels. The white filter function prevents the classifier running on useless frame, as running the classifier cost a lot of time. The logic of the white filter is simply, I used a function called isWhite to calculate the number of white pixel, and then calculate the total number of pixel in the frame. If the number of white frame is more than 50% of the total, then that image is classified as white frame. If the white filter is turned on, the frame would be ignored. The reason to make this a user modifiable variable is that in some of the case, the human in the images wear white clothes, and they may be filtered away due to the white filter.

f. Display mode

The display mode, if turned on, will output the human found during the algorithm in a window. The display mode allows better demonstration of the algorithm as it shows the human found in real time. Turning on the display mode will cause extra running time so the default setting is False.

All the above parameters, if not using the default setting, require user to input a correct value. If the user fail to do so, the program would prompt error message accordingly. Moreover, for efficiency, the default setting is used if the user just press enter without inputting anything. This allow users to skip through the setting they don't want to change.

4.1.3.3 Window Sliding logic

Moving on to the actual window sliding algorithm, the program logic is illustrated in figure 36. The window sliding algorithm consists of several loops. The first loop is the frame size loop, the window sliding algorithm would perform all functions for different frame size. Inside that, we have the height and width loop. The function get the image width and height, and calculate the limit of the frame movement both horizontally and vertically. The frame first slide horizontally, until reach the limit. Then, it returns to the zero position in width, and slide down vertically for moving step pixels, and then slide horizontally again. These two loop together allow the sliding window to scan the image from left to right, from top to bottom.

For each frame, the function check whether the white filter is on, if so, it determines whether the frame is a white frame. After that, the frame will be fed into the Keras model for classification. If the frame is a human, the program will check for display mode, if on, it would output the image as a window. Then the program save the original frame in the Output folder, changing the corresponding frame to white, and continue the loop.

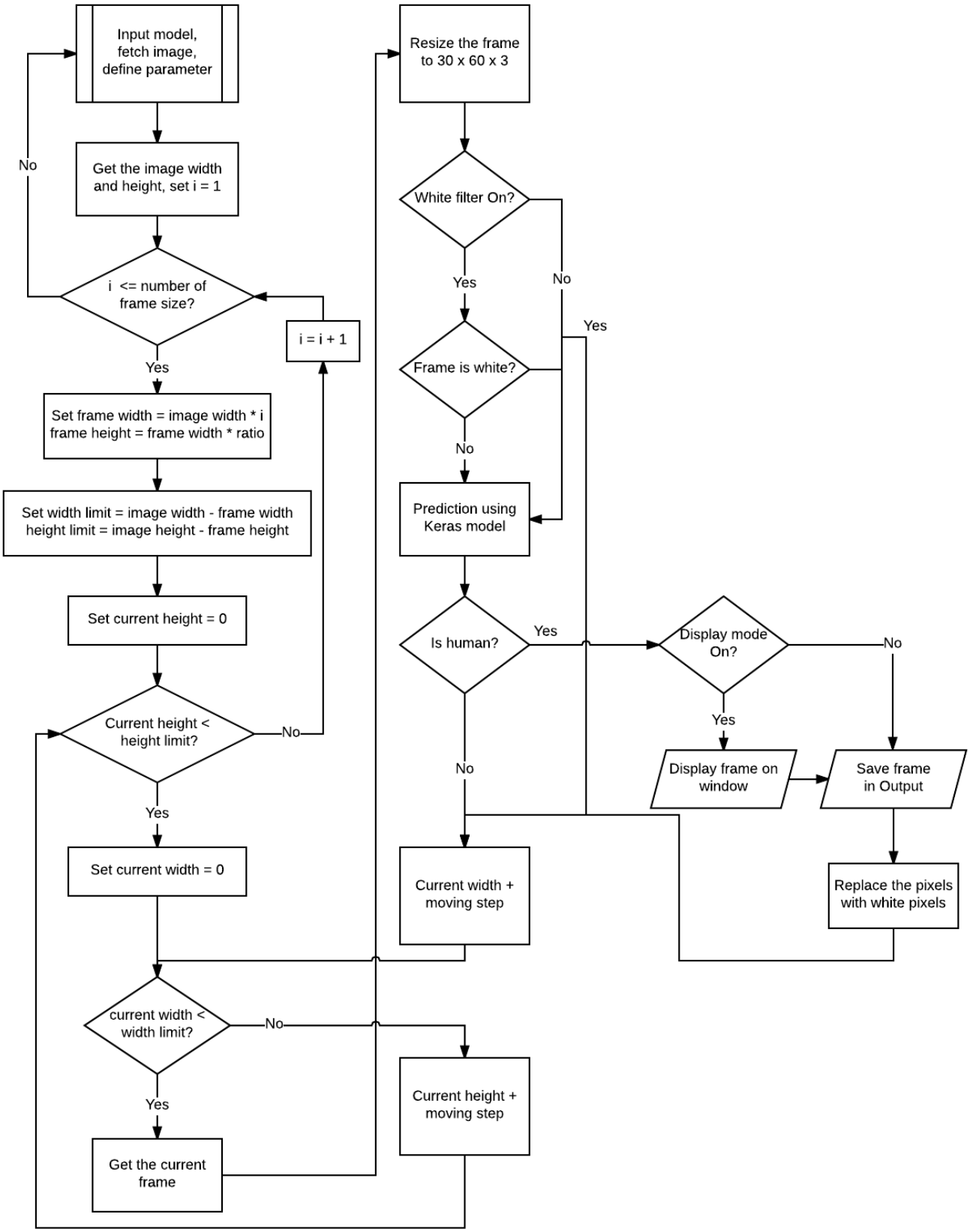


Fig 36. Window sliding algorithm logic flow chart

Different libraries were applied in the program for different purpose. SciPy and NumPy were used to convert the image into matrix form and for most of the computation. Kera was used to load the model and run the prediction, matplotlib was used for outputting the human in real time, and OpenCV Python was used to fetch the image from the web camera. In addition, other Python libraries such as os, sys and time are used for fetching file from folder, getting user input and time stamping.

4.2 Posture Recognition

For the posture recognition, the approach was very similar. First of all, the data was collected and processed. Then the convolutional neural network was trained using the data. Finally, the result was test. As the methodology and implementation on the convolutional neural network has already been explained in previous chapter, this section will only focus on the difference between this algorithm and the previous one.

4.2.1 Data Collection Phrase

For data collection, I first searched up the internet for human posture dataset. Unfortunately, those postures are some general human posture in daily life such as hand shaking and saying hello. The postures I required were the weapon holding and surrender postures, which are very rare. Therefore, I collect my own training data, with the help of some of my friends and classmates. The device I used was an Iphone SE, and the photo was taken in different environment, different background and different lighting condition to improve the variance of the data. Two equipment, a toy knife and toy gun shown in figure 37 were used for the weapon holding posture.

Fig 37. Toy knife and gun used in weapon holding posture

In total, 10 different subjects were invited to take the weapon holding and surrender images. Out of the 10 subjects, 6 of them are boys and 4 of them are girls. These subject varies in height, body shape and clothing. They were told to hold the weapon and prevent to rob a MTR station, and to prevent surrender when there was robbery. Some of the result are shown in figure 38.

Fig 38. Some of the training images obtained

For the normal posture (no weapon holding or surrender posture), I use the positive training data from the INRIA Person dataset. As the people in the dataset has a large variety and with different normal posture, such as walking, talking and standing. The dataset could provide the data I need to train my posture recognition algorithm.

4.2.1.1 Data Processing

The data was manually filtered and processed before converting into matrix. Unlike the INRIA dataset, these set of data don't have constant human to background ratio, and the size and ratio of the human inside a photo could be different. For example, a weapon holding and point to the sky image would have very high height to width ratio, but a surrender posture on knee has a low height to width ratio. It is difficult to use a general program to crop out the human image. Therefore, the image was investigated one by one and the human region was cropped out using Microsoft Paint.

After completing the cropping, I have in total 384 weapon holding and around 200 surrender images. As I would like to have the same number of training image for each class, 184 of the surrender images were selected for flipped to create another copies. This resulted in same number of images in the weapon holding the surrender classes. To match up the number, I used 400 images from the INRIA dataset for the non-weapon holding nor surrendering posture.

Finally, the using the same program I used in the human recognition algorithm, 38 out of the 384 data from both weapon holding and surrendering were randomly selected out to form the testing dataset. Similarly, 40 out of 400 data from the normal posture were selected to be the testing set. Both the training dataset and testing dataset were then converted into matrix using SciPy and NumPy. The process was exactly the same with the human recognition, just adding one more dataset.

4.2.2 Convolutional Neural Network

For the convolutional neural network, the setting was a little different. One of the different of this problem with the human recognition was that we had three classes instead of two, so we could no longer use the binary classifier. In addition, to train a multiply classes classification problem, different cost function and configuration of data is used. Luckily, Keras provides different functions and libraries to solve these problems at ease. In the following section, I would talk about how the convolutional neural network setting was different from the previous one.

4.2.2.1 Model Construction

For the model construction, the same model was used for the posture recognition algorithm, but with some different. The first difference between them was we had three classes instead of two. In two classes situation, we could label the data with only 1 and 0, with 1 representing the positive example, and 0 representing the negative example. With three classes, we need an additional number to label the classes first. The three classes, surrendering, weapon holding and normal were labelled with 1,2 and 0 respectively.

```
31  class_surrender = np.zeros((surrender_train.shape[0],1))
32  class_surrender.fill(1)
33  class_weapon = np.zeros((weapon_train.shape[0],1))
34  class_weapon.fill(2)
35  class_neg = np.zeros((neg_train.shape[0],1))
36  class_neg.fill(0)
```

Using NumPy, arraies filled with zeros with elements number equal to the training set size were created. Then, the correct numbers were filled in to the three different classes.

For the neural network training, the classes should be in binary format instead of number. As the calculation is computed in matrix form. Therefore, the correct representation of the three classes, 0,1 and 2, should be [1,0,0], [0,1,0] and [0,0,1] respectively. Keras provides utility functions for easy conversion between decimal numbering label and binary representation. With the Keras library np_utils, the output classes were generated with two lines of code.

```
57  # Convert class vectors to binary class matrices.
58  Y_train = np_utils.to_categorical(y_train, nb_classes)
59  Y_test = np_utils.to_categorical(y_test, nb_classes)
```

For the algorithm layout, as we only have around 400 training data for each class, a simpler layout was used to prevent overfitting. Instead of using four convolutional layers and two max pooling layers, only two convolutional layers and one max pooling layer were used. The final layout of the network was:

Convolution layer → Relu Activation layer → Convolution layer → Relu Activation layer → Max Pooling layer → Dropout layer → Flatten layer → Fully-connected layer → Relu Activation layer → Dropout layer → Fully-connected layer → Softmax activation layer

In addition to the simplified convolutional neural network structure, the posture recognition algorithm used the softmax activation layer instead of the sigmoid, as sigmoid activated is for binary classification only. And finally, the training loss was categorical cross entropy.

```
88  model.add(Activation('softmax'))
89
90  # Let's train the model using RMSprop
91  model.compile(loss='categorical_crossentropy',
92                optimizer='rmsprop',
93                metrics=['accuracy'])
94
```

The algorithm would output three value, [a,b,c], which are the respective percentage of the image being class 0, 1 and 2. The sum of them is always 1.

4.2.2.2 Model Training

Similarly, after all the network configuration, the network was trained with Keras. With less amount of training data, the training process was faster. On average, each epoch took 18 seconds to train. To train 100 epochs, the total time was 30 minutes. Similarly, each epoch showed the loss and accuracy of the model. When training went on, the loss should gradually decrease and the accuracy should gradually increase.

```
Train on 1092 samples, validate on 116 samples
Epoch 1/50
1092/1092 [==============================] - 18s - loss: 1.0543 - acc: 0.4918 - val_loss: 0.7868 - val_acc: 0.6121
Epoch 2/50
1092/1092 [==============================] - 17s - loss: 0.6579 - acc: 0.7408 - val_loss: 0.6905 - val_acc: 0.7414
Epoch 3/50
 224/1092 [=====>........................] - ETA: 12s - loss: 0.5585 - acc: 0.7857
```

4.2.2.3 Model Testing

The model testing was done using the same testing program. The loss and accuracy of the model was as follow:

```
train:  [4.4506529150320549e-05, 1.0]
test:  [0.78411946261044718, 0.91379310344827591]
```

The training loss was 4 x 10^{-5} and the accuracy was 100%. This may seem really good, but was accurately an indicator of overfitting. Comparing to the testing dataset, the testing dataset has a loss of 0.78 and accuracy of 91.3%. To fight overfitting, different setting of the neural network

was tried out. However, the network either suffer from underfitting or overfitting no matter what. It was concluded that the main problem of this algorithm was insufficient training data. More would be mentioned in result analysis and discussion.

4.3 Program Delivery

The last step of the program was to convert everything into a window executable file. The original program, being written in Python, required Python and the Python library as dependency to be installed in the local device. However, for most end user of the program, they don't have Python nor the dependency installed. So the program won't run on their device. One way to solve this problem is to use low level language such as C++ or JAVA to rewrite the entire program and the neural network, however it would require much more resources and time for this project. Therefore, to convert the program, a Python library named CX-Freeze was used.

As discussed in previous chapter, CX-Freeze is an open-sourced Python library for easy conversion from Python program to window executable. According to the official document of CX-Freeze, it is cross platform and work on any platform Python itself works on. The CX-Freeze allows user to write a setup program, which would fetch the required libraries for the Python program specified and generate a build file.

```
import sys
from cx_Freeze import setup, Executable

base = 'Console'

# excludes = ['tkinter']
packages = ['tkinter']

methods = ['numpy.core._methods', 'numpy.lib.format','scipy.sparse.csgraph._validation',
           'matplotlib.backends.backend_tkagg']

files = ['human_reg_model1.h5']

setup(  name = "sliding_window_v1",
        version = "0.1",
        description = "Sliding window algorithm for human recognition",
        options = {"build_exe": {
                                 'include_files':files,
                                 'includes':methods,
                                 'packages':packages
                                 # 'excludes':excludes
                                 }},
        executables = [Executable("sliding_window_v1.py", base=base)]
    )
```

Fig 39. Screen capture of the CX-Freeze setup program

The CS-Freeze setup method allows user to define the setting of the build, for example, the name of the program, version, description, options and executables. The first three are self-explanatory, while the options define the necessary libraries and files for the program. For most of the libraries, CX-Freeze would automatically detect and include them in the final build.

However, for complicated program with different libraries, manual input is often required. For this example, I manually added the SciPy and NumPy core methods. The files to be included is the human recognition model trained with Keras. After the conversion, CX-Freeze created a folder, inside the folder, there was the executable program and all the dependency of it. As the folder looks really messy with all the files and libraries, to explain the folder structure better, those files would be hidden.

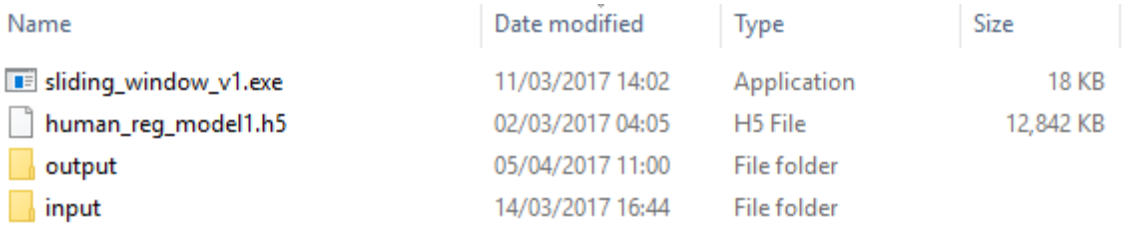


Fig 40. Folder layout for the window executable program

There are four main components for the program, the first one is the actual executable, the second one is the human recognition model used in the program, the third one is the output folder, and the final one is the input folder. The input folder store the images used in the algorithm, for web camera fetching mode, the image fetch from the camera is also stored in the input folder. While the output images of the program are stored in the output folder.

CX-Freeze didn't add running time on the original program, so the program run in a relatively same speed.

Chapter 5: Result Analysis and Discussion

This this section, I would talk about the result and analyse the performance of the two algorithms, the human recognition algorithm and the posture recognition algorithm. For the human recognition algorithm, the performance of the window sliding algorithm would also be included.

5.1 Human Recognition Algorithm Performance

Overall, the human recognition algorithm has a decent performance for many different images. The accuracy is good for detecting the human out of the images. As the images have different human ratio, the default setting not always provide the optimal result. By adjusting the window sliding algorithm setting, the program could detect human in different situation. The human recognition algorithm was tested with five different images. The results are shown and explained below.

Image example 1:

The first image is a family at the beach. This example has similar color for the background and the human, and it aims to test whether the program could detect the human correctly in such situation. Using the default setting, with white filter turning on, some of the human was filtered as they are in white shirt and have white sky as background. With the white filter turned off and no delay saving, the result was as follow.

Fig 41. Image 1: Original image (top) and output from algorithm (bottom)

It was surprised that the algorithm would recognition the first frame as a human. For most of the case, it is not a problem as the white filter should be turned on. This was a quite decent result as it recognized all human in the image with only one false output. The running time for the program was 45 seconds.

```
Program complete,in total, 5 found.
 Execution time:  45.00271725654602  seconds
```

Image example 2:

The second image was a woman sitting behind a table in a room with different object. This image tested the ability to detect sitting human with minor blocking. The default setting was used, and the algorithm successfully found the human and output one false detection. The result was still satisfying as the non-human object is most likely to be classified as no posture in the later posture recognition. The running time was 89 seconds.

```
Program complete,in total, 2 found.
 Execution time:  89.74300694465637  seconds
```

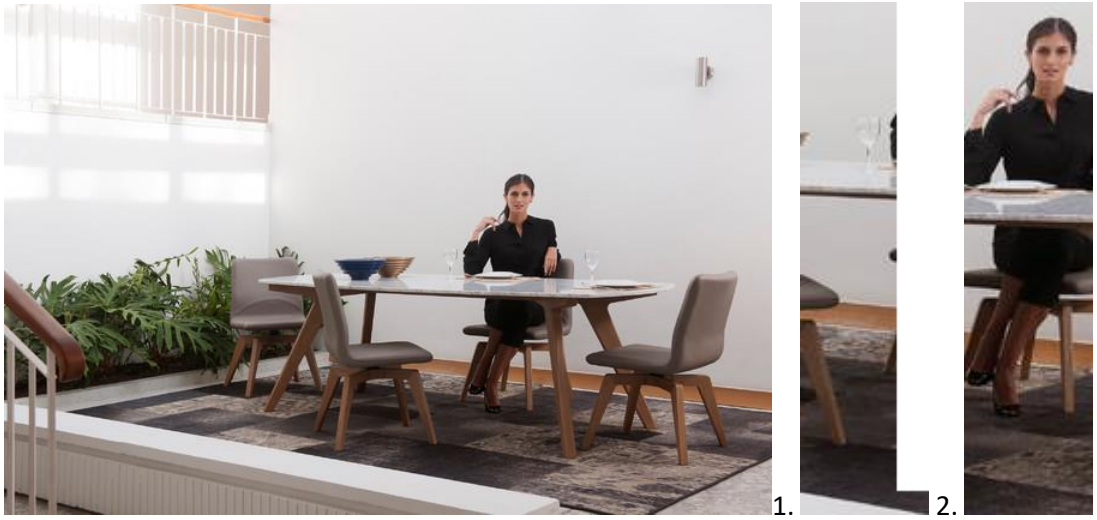
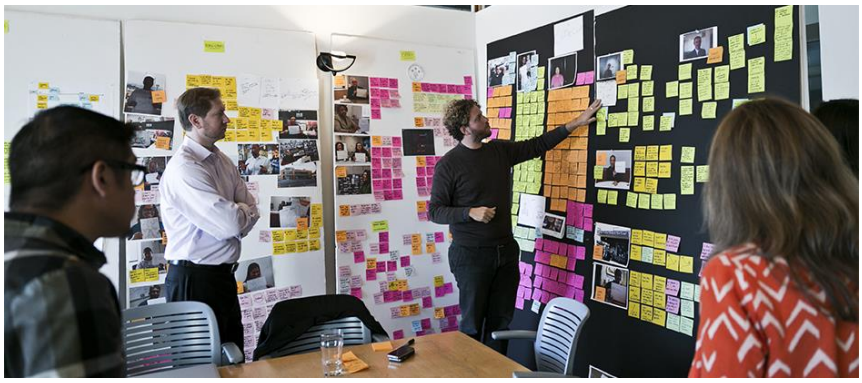


Fig 42. Image 2: Original image (left) and output from algorithm (right)

Image example 3:

The third image is 4 different people in a meeting room. Two of them had their back to the camera. This test experimented how the algorithm would perform in a complex environment with a lot of objects and colors. As for the two standing human, some of their body was blocked by the table, it is decided to use a 3:1 height to width ratio. Also, no delay was applied



1.  2.  3.  4.  5.  6.  7. 

Fig 43. Image 3: Original image (top) and output from algorithm (bottom)

Out of 6 images, 7 of them contain human or part of human. The algorithm successfully recognized the two standing human, and also regard the head of the woman on the right bottom as human. However, it also had 4 false output, two of them are part of human, the hand and part of the head, the other two are just colourful background. This may due to there are few colourful negative image in the training data. The total running time was 48 seconds.

```
Program complete,in total, 7 found.
 Execution time:  48.930269956588745  seconds
```

Image example 4:

This image is another people meeting in a room. The purpose was to test whether the algorithm could classify similar setting with image 3 but more simple background. The frame size used was [0.2,0.3] and the human ratio was 3. As the smallest human width is at least 20% of the whole image, there is no point running the 10% frame size.

1.  2.  3.  4. 

Fig 44. Image 4: Original image (top) and output from algorithm (bottom)

For some reason, the program missed the man at the right. One reason may be in image 3, half of that man was cropped and replaced with white frame, and the program failed to recognize it as human. The running time was 30 seconds.

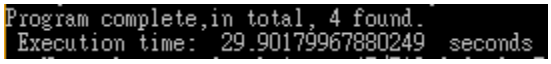


Image example 5

The final example was three people sitting in a room with simple background. This was aimed to study whether the algorithm could effectively output sitting human. As the training data are mostly standing human, it was interested whether the algorithm learned that sitting human is the same with standing human from the features.

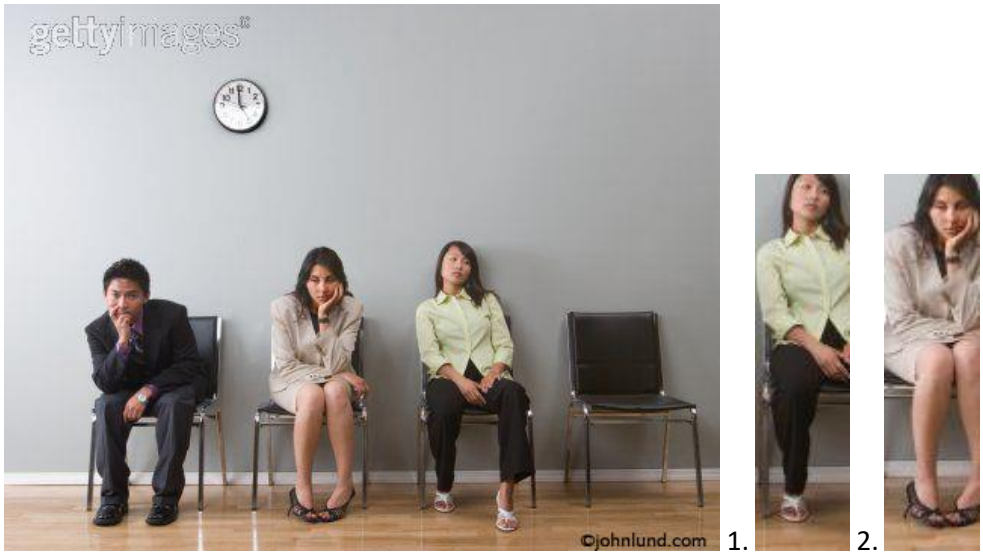


Fig 45. Image 5: Original image (left) and output from algorithm (right)

Two out of three human were detected. Turns out the algorithm can classify sitting human. However, the man in black suit was not detected. It is not sure whether it was due to him sitting posture, or the absence of black dress human in the training data. The program finished in 73 seconds.

```
Program complete,in total, 2 found.
 Execution time:  73.50383257865906  seconds
```

A short conclusion, the human recognition algorithm combining with the window sliding algorithm provide decent and satisfying result on accuracy. However, the running time of the program was too high for real time practice. Also, the algorithm has some difficulty in locating correctly for a sitting human. The human in the frame was usually not at the optimal position. In addition, the program had same trouble recognizing colourful background. Some of the further study and improvement to tackle these problems would be given in 5.3 future improvement.

5.2 Posture Recognition Algorithm

The posture recognition, with similar configuration, didn't have very high accuracy on real-life data. The algorithm output was very random, and tends to predict no posture or posture. For the 5 negative examples, 4 of them were classified correctly. For simplicity, I would use No to denote no posture, Weapon for weapon holding and Sur for surrender. The result was as follow:

1. No 2. No 3. Weapon 4. No 5. No

The reason for the high accuracy is that the algorithm always try to predict no posture. For surrender posture, Out of 6 examples, the program only got two correct.

1. No 2. No 3. No 4. Sur 5. Weapon 6. Sur

For image 4, the problem predicted 53% for surrender and 45% for weapon holding. Which showed that it struggled to tell the different between surrender and weapon holding. And finally for the weapon holding, out of 7 images, the program, surprisingly, got 5 of them correct.

1. Weapon 2. Weapon 3. No 4. Weapon 5. No

6. Weapon

7. Weapon

For the weapon class, the algorithm was pretty sure about it decision. For most of the image classified as weapon holding, the percentage of the weapon holding class exceed 99.9%. There is no ambiguous case. It is suspected that the weapon itself has easy to recognize features, making it easy for the algorithm to recognize the weapon holding class.

One of the reason that the posture recognition is not doing particularly well is that the training dataset is small. With only around 400 images to train for each class, the algorithm didn't learn much. Also, the same human posture could have unlimited variety, make it really difficult to train the algorithm to recognize all the possible posture. In addition, the testing images used in the posture recognition was perfectly cropped, while in the case while it was fed by the sliding window algorithm, it maybe even more difficult to recognize the image.

5.3 Future improvement

The result proving that convolutional neural network is a good approach for recognizing images. However, the current algorithm is not yet mature for a real-time application. As the algorithm design is only part of the whole study, this section will be too long to include all future work to be done, such as fetching image from surveillance camera, connecting with MTR system, security and so on. My discussion would only focus on the further study direction and possible improvement to the program itself.

5.3.1 Data collection

One of the critical reason that the algorithm is not preforming well in different situation is the low variety of the training data. The training data used in this study, was mainly up-straight human. As a result, the human recognition algorithm had difficulty recognizing sitting human. It is believed that, with a larger amount and variety of training data, the human recognition algorithm could output better result in classifying the human. The same applies to the posture recognition algorithm, due to limited time and resources for the final year project, it is impossible to collect a large amount of human posture data. It is also worth knowing that classifying a human posture is much more difficult than classifying the human itself. As the same posture could have completely different meaning in different situation. For example, a man

hands up could be surrendering, or just celebrating. Using more training data, it is possible to better recognize certain posture, but still there are limitation that the algorithm could not tell the different between a positive and negative meaning posture, given that them are the same.

A convolutional neural network is very powerful given enough amount of training data to learn from. To fit the real situation better, the MTR company could collect positive image training data in their station control room, using the surveillance camera. As the surveillance camera take image from the top, the human ratio maybe a bit different from my training set. Using the real data for algorithm training, the algorithm could perform much better. For negative training example, the MTR company could easily take different image from an empty control room, and use my random crop image program to obtain the desirable amount of negative data.

For the posture recognition algorithm, in additional to collect more training data, it is believed that the MTR company should limited the scope to only weapon holding. As mentioned, for a human posture, although could be recognized, is difficult to be interpreted by the algorithm. My study showed that the weapon held by the human contain features that are easily identifiable by the program. Therefore, I believe for further study, the focus should be identifying weapon out of the image.

5.3.2 Algorithm improvement

One of the main problem of the current algorithm is the long run time. For real-time application, the running time should be less than 1 second. One of the reason for the long running time is the CPU I was using is not powerful enough, another reason is that the sliding window function involve calling the Keras model for classification. Calling and using Keras model cause the long running time of the program. It is believed that if the entired program and network was constructed using low level language, such as C++ and JAVA, the running time should decrease.

In addition, the current sliding window algorithm using the looping method for the classification. Another way to use the sliding window is to first slide through the image and store all the frames, then run the recognition for all frame by one. This approach would take up a lot of memory, but could run in a faster time. However, using this method, the program may output multiply human for a single human image, as multiply frame would contain the same human with little difference in position. The current sliding window algorithm use white frame to replace the found human to replace this problem. If it is designed to change the program logic, this issue should be put in consideration.

5.3.3 Histogram of Oriented Gradient

The last thing that MTR company could be looking at is the histogram of oriented gradient (HOG). In additional to convolutional neural network approach, the HOG approach was also used for object classification, especially for human object. The HOG is an image features

extractor, which compute the gradient vector of the image. The gradient vector indicates how much each pixel change vertically and horizontally. The HOG was usually integrated with simpler machine learning algorithm, such as Support Vector Machine (SVM). Similar with the deep learning approach, the HOG + SVM approach works best with accurate training data. However, the running time of these two combination is faster than the deep learning and may prove to be more suitable for real time classification.

5.3.4 Conclusion

In conclude, this project studied the feasibility of using the deep learning approach, aka the convolutional neural network for human and posture classification. The accuracy of the algorithm highly depends on the training data and the algorithm configuration. A better algorithm could be trained with more close-to-real-life training data, and a larger variety of the training data would help the algorithm to classify in different situation. My study shows some success in recognizing human in images, and proving that the deep learning approach is appropriate for image recognition. For future study, it is recommended that the MTR company should continue to collect more tailor-made data, reconstruct the algorithm using low level language and better CPU, as well as considering using HOG in image feature extraction.

Reference

1. Brian D. Ripley (1996), “Pattern Recognition and Neural Networks”, published by Cambridge University Press.
2. CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.stanford.edu/
3. Christopher M. Bishop. (1995), “Neural Networks for Pattern Recognition”, Aston University, published by CLARENDON PRESS, OXFORD http://cs.du.edu/~mitchell/mario_books/Neural_Networks_for_Pattern_Recognition_-_Christopher_Bishop.pdf
4. Michael Nielsen (2016), “CHAPTER 1 Using neural nets to recognize handwritten digits”, “Deep Learning”, retrieved from http://neuralnetworksanddeeplearning.com/chap1.html
5. Sebastian Ruder (2016) “An overview of gradient descent optimization algorithms”, retrieved from http://sebastianruder.com/optimizing-gradient-descent/
6. Andrew Ng. (2016), “Week 4 – neural network representation”, from Coursera MOOC “Machine Learning”, Stanford University
7. Andrew Ng. (2016), “Week 5 – neural network: learning”, from Coursera MOOC “Machine Learning”, Stanford University
8. Machine Learning for Sequential Data: A Review , Thomas G. Dietterich
9. Navneet DALAL (2005) “INRIA Person Dataset”, retrieved from http://pascal.inrialpes.fr/data/human/
10. Navneet DALAL (2005) “Finding People in Images and Videos”, Grenoble Institute of Technology, retrieved from http://lear.inrialpes.fr/people/dalal/NavneetDalalThesis.pdf
11. Mao, Weidong (1991) “Neural network algorithms and architectures for pattern classification”, retrieved from HKUL electronic resource
12. Alex Krizhevsky (2009), “Learning Multple Layers of Features from Tiny Images”, published by Canadian Institute.
13. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton (2012), “ImageNet Classification with Deep Convolutional Neural Networks”
14. Histograms of Oriented Gradients for Human Detection http://vc.cs.nthu.edu.tw/home/paper/codfiles/hkchiu/201205170946/Histograms%20of%20Oriented%20Gradients%20for%20Human%20Detection.pdf
15. Tom M. Mitchell (2006), “The Discipline of Machine Learning”, published by School of Computer Science, Carnegie Mellon University
16. Siddheswar Ray and Rose H. Turi (1999), “Determination of Number of Clusters in K-Means Clustering and Application in Colour Image Segmentation”, published by School of Computer Science and Software Engineering, Monash University
17. Henry A. Rowley, Shumeet Baluja, and Takeo Kanade (1998), “Neural Network-Based Face Detection”,
18. Chris Stauffer, W.E.L Grimson (2014) “Adaptive background mixture models for real-time tracking” by The Artificial Intelligence Laboratory, Massachusetts Institute of Technology