

Fireball Event Detection using a Machine Learning Approach

Jean Deshayes

B.Eng. Mechatronic Engineering
Project Report
Department of Mechanical Engineering
Curtin University

2015

8a Keane Street,
KEWDALE,
WA 6105

5th of November 2015

The Head
Department of Mechanical Engineering,
Curtin University,
Kent Street,
Bentley,
WA 6102

Dear Sir

I submit this report entitled “Fireball Event Detection using a Machine Learning Approach”, based on Mechanical Project 491/493, undertaken by me as part-requirement for the degree of B.Eng. in Mechatronic Engineering.

Yours faithfully

Jean Deshayes

Acknowledgements

The author wishes to thank Jonathan Paxman as project supervisor for the valuable opportunity to participate in the DFN project and his appreciated help and guidance provided throughout the project. The author would also like to thank Martin Towner, Dr Martin Cupak, Professor Phil Bland, Ellie Sansom and every other DFN member for their support and involvement in the project.

The author also wishes to thank Dane Black, his brother in law, for his proofreading; and also his girlfriend Marion Monteil for her everyday support and assistance.

Preface

The sections 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.2 and 2.3 of this report contain material previously presented in the progress report (Deshayes, 2015) and the abstract contain material presented in the extended abstract (Deshayes, 2015). Figures 1, 2, 3, 4 and 8 were also previously used in the progress report for this project.

Abstract

As part of the Desert Fireball Network (DFN), this project was proposed to test the performance of a machine learning algorithm at detecting fireball trails in images of the night sky. The goal of the DFN project is to build a network of digital cameras that will image most of the Australian sky to track meteor orbits in the solar system and estimate their landing zone on Earth. Hence the importance of implementing a computer program to automate the fireball event detection. Due to the complexity of the data, an artificial neural network algorithm was chosen to perform the meteor detection, as its integration into multiple classification problems has shown remarkably good performance. However to perform as well as a human brain would, the artificial neural network requires meticulous tuning. Throughout the project an image pre-processing program as well as a neural network algorithm have been implemented and compared with an explicitly programmed computer code. The image filtering process emphasized any differences between two images which considerably narrowed down the potential meteor images. The neural network developed on Python successfully classified 92% of the tiles having meteor streaks which was an improvement from the original detection method. On a selected set of fireballs containing several faint meteor streaks, the neural network performed with 86% accuracy compared to 44% for the code currently in operation.

Nomenclature

| Variables | Definitions |
|--|--|
| X | Inputs |
| Y | Outputs |
| θ | Parameters of the model (weights) |
| $h\theta(X)$ | Sigmoid activation function |
| $a_i^{(j)}$ | Activation of unit i in layer j |
| $\Theta^{(j)}$ | Matrix of weights controlling function mapping from layer j to layer $j+1$ |
| g | Sigmoid function |
| m | Total number of training examples |
| n | Total number of features |
| L | Total number of layers in the network |
| s_l | Number of units in layer l (not including bias unit) |
| Lambda | Regularisation parameter |
| Accuracy | Sum of true positives and true negatives over sum of all data |
| Precision | Sum of true positives over sum of true positives and false positives |
| Recall | Sum of true positives over sum true positives and false negatives |
| J | Cost function |
| $J_{\text{train}}(\theta)$ | Cost function of the training set |
| $J_{\text{cv}}(\theta)$ | Cost function of the cross validation set |
| $J_{\text{train}}(\theta)$ | Cost function of the test set |

Table of Contents

| | |
|---|-----|
| Acknowledgements | V |
| Preface..... | VII |
| Abstract | IX |
| Nomenclature | XI |
| Chapter 1: Introduction | 1 |
| 1.1. Overall problem definition | 1 |
| 1.2. Desert Fireball Network Aim | 1 |
| 1.3. Project Background | 2 |
| 1.4. Motivation/Interest | 3 |
| 1.5. Requirements..... | 3 |
| 1.5.1. Requirement of data inputs..... | 3 |
| 1.5.2. Requirement of software | 4 |
| 1.5.3. Requirement on the performance | 4 |
| 1.5.4. Requirement of output..... | 4 |
| 1.6. Scope | 5 |
| 1.6.1. Machine learning algorithm | 5 |
| 1.6.2. Meteor properties | 5 |
| 1.6.3. Meteorite recovery | 5 |
| Chapter 2: Background | 7 |
| 2.1. Literature review | 7 |
| 2.1.1. Meteor properties | 7 |
| 2.1.2. Existing detection techniques | 8 |
| 2.1.3. Previous meteor detection work using Machine learning algorithms | 9 |
| 2.1.4. Overview of Current detection process | 9 |
| 2.2. Machine Learning..... | 9 |
| 2.2.1. Overall view | 9 |
| 2.2.2. Supervised learning algorithms | 10 |
| 2.2.3. Unsupervised learning algorithms | 12 |
| 2.2.4. Reinforcement learning algorithms | 12 |

| | | |
|------------|--|----|
| 2.2.5. | Machine learning algorithm selection | 12 |
| 2.3. | Neural Network | 14 |
| 2.3.1. | Neural Network origin..... | 14 |
| 2.3.2. | Applications of Neural Networks..... | 16 |
| 2.3.3. | Neural network terminologies | 16 |
| 2.3.4. | Activation function..... | 16 |
| 2.3.5. | Feed forward Propagation | 18 |
| 2.3.6. | Learning algorithms | 18 |
| 2.3.7. | Gradient Descent | 20 |
| 2.3.8. | Back Propagation | 21 |
| 2.3.9. | Testing | 22 |
| 2.3.10. | Performance improvements | 24 |
| 2.4. | Python..... | 28 |
| Chapter 3: | Image pre-processing | 29 |
| 3.1. | Purpose of Filtering raw images..... | 29 |
| 3.2. | Image Differentiating | 30 |
| 3.2.1. | Channel selection | 30 |
| 3.2.2. | Masking | 31 |
| 3.2.3. | Background removal | 31 |
| 3.3. | Down sampling..... | 34 |
| 3.4. | Slicing..... | 35 |
| 3.5. | Filter steps representation..... | 35 |
| Chapter 4: | Design | 37 |
| 4.1. | Neural Network Architecture | 37 |
| 4.2. | Datasets | 38 |
| 4.3. | Development of back propagation | 39 |
| 4.3.1. | Hyperparameters | 41 |
| a. | Number of examples | 41 |
| b. | Regularisation parameter | 42 |
| 4.4. | Program design..... | 43 |
| 4.4.1. | Input data..... | 43 |
| 4.4.2. | Structure | 44 |

| | |
|---|-----------|
| 4.4.3. Output | 44 |
| Chapter 5: Performance evaluation..... | 45 |
| 5.1. Testing set..... | 45 |
| 5.2. Comparison with current detection process | 45 |
| 5.2.1. ROC..... | 45 |
| 5.2.2. Confusion matrix | 46 |
| a. Tile basis | 46 |
| b. Image basis | 47 |
| 5.2.3. Speed | 48 |
| Chapter 6: Conclusion and Future Work | 49 |
| 6.1. Pre-processing improvements | 49 |
| 6.2. Neural network improvements | 50 |
| 6.2.1. Deep neural network | 50 |
| 6.2.2. Learning methods | 50 |
| 6.3. Hardware improvement | 50 |
| 6.4. Next step..... | 51 |
| References | 53 |
| <i>Appendix A: Performance results of NN and MT codes</i> | <i>54</i> |
| <i>Appendix B: System Summary.....</i> | <i>54</i> |
| <i>Appendix C: Description of the contingent matrix</i> | <i>54</i> |

Table of Figures

| | |
|---|----|
| Figure 1: Neuron and perceptron illustration (Motor Neuron 2015) | 15 |
| Figure 2: Artificial intelligence neural network nodes (Tadiou. K, Mamadou, 2015) ... | 15 |
| Figure 3: Graphical representation of (a) perceptron and (b) sigmoid activation function (Perceptron 2012)..... | 17 |
| Figure 4: Cost function with two variables (Ng, Andrew 2015)..... | 20 |
| Figure 5: Confusion matrices | 23 |
| Figure 6: Two typical ROC curves and an expected performance level of a random guessing system (Woods, 1997)..... | 24 |
| Figure 7: Under- and Over-fitting examples (Ng, 2015) | 26 |
| Figure 8: Learning curves examples (Ng, 2015)..... | 27 |
| Figure 9: Orion's belt classified as a meteor by the current program | 30 |
| Figure 10: RGB channels | 31 |
| Figure 11: Threshold examples | 32 |
| Figure 12: Graph of the threshold and brightness relationship | 33 |
| Figure 13: Slicing examples | 35 |
| Figure 14: Filtering steps | 36 |
| Figure 15: Subsets of positive and negative samples | 39 |
| Figure 16: Gradient calculation on Python | 40 |
| Figure 17: Plot of cost against number of training examples | 41 |
| Figure 18: 8 different orientation for one data sample..... | 41 |
| Figure 19: Plot of error against regularisation parameter | 42 |
| Figure 20: User interface to browse directory..... | 43 |
| Figure 21: ROC curves | 45 |
| Figure 22: Confusion matrices | 46 |
| Figure 23: Filtered cloudy image | 47 |

Chapter 1: Introduction

1.1. Overall problem definition

The formation of the solar system remains one of the biggest questions in planetary science that mankind has, yet, unable to explain due to a lack of evidence. Studying the composition of asteroids and comets, which are the first elements that form planets, will shed light on the fundamental question of how planets came into being and how dust and gas produced a planet capable of supporting life – The Earth (Bland and Tingay, 2013). The current way to gather this information is by collecting limited samples of asteroids directly from their orbit using robotic spacecraft. However this process is extremely costly and requires several years to complete. Therefore, planetary scientists are looking for easier methods to obtain this information such as the investigation of meteorites that have landed on Earth. A meteorite is the name given to the debris of meteoroid after surviving the trip through the atmosphere and has survived the impact on the Earth's surface (Oxford English Dictionary, 2015). A meteoroid is defined as a part of an asteroid or comet that is orbiting the sun at incredibly high speed. Studying the internal structure of a meteorite can provide valuable information on its relative age and its origin in the solar system which, in turn, can determine to what asteroid they belong to. Nevertheless, although it is possible to observe a scattering of falling stars in the night sky, only a tiny minority can be recovered. In fact most of them, particularly the smallest ones, get destroyed as they enter the atmosphere while the rest of them will just never be retrieved. When a meteoroid enters the atmosphere the air in front of it compresses extremely rapidly which engenders a quick rise in temperature. The meteor heats up so much that it starts to burn until it is completely reduced to dust or it has sufficiently dropped in speed. This is the reason why fireballs are visible from the Earth's surface.

1.2. Desert Fireball Network Aim

This project, in collaboration with the Desert Fireball Network (DFN) project, aimed at developing a fireball event detection program using a machine learning approach with the objective of improving the current detection method and thus enhancing the chances of recovering meteorites. The DFN project, founded by a group of planetary scientists,

geologists and engineers, is a network of digital cameras that image the whole night sky to capture fireball falls. The network is currently constituted of about 33 cameras spread across the Australian outback, each fitted with a processor, a storage system and a satellite communication device. Each observatory turns on after sunset to capture 30 second exposures throughout the night that are stored in real time on a 6TB hard drive. Before sunrise, the camera turns off and switches to the detection mode. The integrated processor, running a modified version of LINUX, executes a fireball detection script implemented by Dr Martin Towner to identify potential fireballs in the freshly captured images. This program was designed to be executed on a low performance processor which involved compromising the performance for speed. The potential meteor samples and their relative information are then transmitted to the DFN server for cross validation and continuation of the recovery process. The next step of the DFN project consists of combining the networked observations of the fireball along with sightings reported by the users of 'Fireball in the Sky' (a smartphone application developed by the DFN team), to estimate the fall position on Earth. The triangulation of the landing zone is evaluated from the calculated speed, weight and direction from the different observations (Fireballs in the sky, 2015). Once the searching zone is narrowed down, the ground search starts in the hope of addressing some of the biggest questions in planetary science.

1.3. Project Background

This project was proposed in order to evaluate the performance of a neural network at detecting meteors compared to an explicitly programmed approach. A machine learning approach will give the computer the ability to learn how to identify meteors without being explicitly programmed. This program, intended to run on an average computer or super computer, will verify that all pictures containing meteors have been classified by the currently operating program. Over 900 high resolution exposures, are captured and saved on the internal hard drive. This corresponds to about 10TB of photos per year per camera and 500TB since the beginning of the DFN project which explains why an efficient computer program was essential to automate the detection of the meteors. The landing zone is estimated from the triangulation of the identified meteor images captured by multiple cameras. The more observations from different point of views will result in a

higher accuracy of the landing zone. Therefore the following steps of the meteorite recovery process relies upon the meteor detection, hence the need to improve the current system.

Machine learning is a growing field that has brought incredible performance to a wide range of applications. The use of machine learning was not only a way to improve the accuracy of the current detection but also to increase the processing speed. Indeed, once the computer is trained, the prediction algorithm is a simple succession of matrix multiplications which requires little amount of memory and CPU processing.

1.4. Motivation/Interest

The author's motivation to undertake this project was to participate in a world class project that will shed light on the mystery of the composition of the Earth and the surrounding planets in the solar system. Although, the accomplishment of this project was just a small step in the recovery of meteorites and hence an even smaller step to solving the mystery of the solar system, it is still a crucial step forward to enhancing the chance of meteorite recovery.

In addition, this project is entirely focused on programming, which is one of the strengths of the author. Nowadays, the application of machine learning is taking part in a wide number of classification projects due to its efficient and powerful way to make predictions. Thus the opportunity to learn Machine Learning algorithms was highly sought after by the author. Therefore, this project was seen as a great opportunity to learn and apply Machine Learning techniques to real world problems that have never been completed before.

1.5. Requirements

1.5.1. Requirement of data inputs

Images provided by the DFN are obtained directly from the observatory hard drives. These pictures are labelled in a detailed manner, which gives information on the date, time and the picture ID. The ID refers to the order in which the photos were captured. As discussed in Chapter 3:, the images are pre-processed to eliminate identical features present on two successive images. However, when the image is too bright which either

means the sunlight is still visible or some clouds are present, the camera stops recording. Therefore any data preparation based on the difference between two images must take into consideration the time difference and ID number of both. Doing so prevents misclassification resulting from the difference of two pictures of more than 1 minute interval.

1.5.2. Requirement of software

The software program was required to be implemented on Python to use the same high-level programming language as the DFN team, which will facilitate an eventual incorporation to the observatory processing unit. To comply with the DFN standards, the code should be written on Python 2.7 or a compatible version and must be well documented, that is, every function should be explained, any constant should be defined in a separate section and any instruction on how to use the code must be provided. Thus, any user is able to easily execute the code or make modifications as desired.

1.5.3. Requirement on the performance

Although the primary goal of this project was to determine the practicability of using a machine learning algorithm to identify meteors, it was implied that the design should perform better than the current code. In fact, the final design should be able to detect most of the meteors that the current code has missed. Of course it was also defined that regular images with no fireball events had to be ignored. However, if it happened that aircraft or satellite streaks are present on an image, the classifier may misclassify them as those cases will be disregarded later on with the triangulation process.

Given the issues encountered by the current program, it was important that the new process would overcome those issues. For example, any partially visible meteor streak, even if partially hidden by trees, should be identified as they are important for accurately estimating the landing zone.

1.5.4. Requirement of output

The final program should process raw images from a folder inputted by the user. The program must then load the images in their order of creation, perform any filtering or transformation prior to feeding the data through the machine learning algorithm and finally determine whether a meteor is present on the image. The positive identification photo must be saved in a separate folder and clearly labelled to indicate the approximate

location of the predicted streak. Therefore the images names must match the inputted names to comply with the naming convention: “date_time_DSC_IDnumber.jpg”. As a result, the analysis or comparison between the different images at diverse times or from different cameras is facilitated. It is also necessary that a log of the code should be saved as the code executes to ease debugging and provide feedbacks.

1.6. Scope

1.6.1. Machine learning algorithm

The objective of this project was restrained to the design of a new meteor detection program based on machine learning algorithms but it was not intended to compare different algorithms. In fact the artificial neural network algorithm was suggested by the project supervisor for its powerful potential in pattern recognition.

1.6.2. Meteor properties

This project was limited to the identification of meteors in images of the night sky and did not involve studying any properties of the meteor. Any properties such as its speed, deceleration, constitution, weight, size and direction is to be estimated by the DFN members.

1.6.3. Meteorite recovery

The recovery process is also not included in the scope of this project and is done at a later stage once the landing zone has been sufficiently narrowed down. Currently, the recovery is completed by the DFN members who travel to the predicted location to intensively look for meteorites. Autonomous robots are being designed to fly over the estimated landing zone and search for rocks resembling meteorites. An autonomous ground vehicle which will be placed in the searching area is also being designed to drive to the meteorite sightings to have a closer look.

Chapter 2: Background

2.1. Literature review

2.1.1. Meteor properties

As a result of the high velocity of the meteoroids when penetrating the atmosphere, the air compresses and quickly heats up causing the asteroid debris to glow. This bright streak, observable by the naked eye when looking at the sky, is more often called a shooting star. The colour of the fireballs are often reported as eye catching and vivid because of the intense brightness. In fact, the colour of the fireball can already provide a few details about its internal composition. When burnt, sodium produces a yellow colour, nickel a green colour, and magnesium a blue-white colour (Richardson, 2013). The kinetic energy of the meteoroid also greatly affects the brightness of the fireball. It has been reported that fainter meteors are either slow and therefore produce a red or orange colour, or fast and so emit a blue colour.

By contrast, fireballs may not always reach the Earth's surface. It was observed that only the fireballs that were emitting a light of brightness magnitude smaller than -8 had a potential to produce a meteorite. In addition the meteoroid must be composed of solid materials to survive ablation and the impact on the ground. Therefore only meteors of asteroidal origin, of decent size and relatively slow speed will have a chance to hit the ground (Richardson, 2013).

A typical meteor burns up until its speed drops to around 2-4km/s. The ablation time depends upon many factors relative to the property of the meteor. Nonetheless, a fireball lasts on average less than 10 seconds which means it is visible on only one 30 second exposure. After the ablation process the speed continues to drop to about 0.1 km/s making it invisible to track during the final stage of the fall.

The population of meteoroids entering the atmosphere is estimated to be composed of 95% from cometary meteoroids, 5% from chondritic meteoroids and less than 1% from non-chondritic meteoroids. Hence only 5% of the total population of meteors has a high likelihood of reaching the ground which supports the idea that faint meteor streaks may be neglected as they are not of importance in the recovery.

2.1.2. Existing detection techniques

Earth is constantly bombarded by meteors but just over 50 000 meteorites have been retrieved in the world thus far (Wichman, 2010). For a long period, meteorites were recovered based on visual sightings and presumptions. Some large meteors have struck the Earth's surface causing craters that were easy to identify. However, meteorites are exposed to climatic risks and geological risks which highly contribute to the meteorite's demise hence the necessity to recover them as early as possible. Therefore to raise the chances of recovering fresh meteorites it is essential to track the meteor as they fall to the ground and act rapidly.

During the ablation process the air along the trail is ionised, reflecting radio waves. Therefore recent techniques consists of emitting radio waves into the sky using VHF radar to detect the ionisation trails from the reflected waves (Meteorscan, 2014). Although this method is fairly accurate and provides good information on the speed, deceleration and size of the meteor, it requires very expensive high powered VHF radars and does not reveal the fall position nor its direction of fall.

Another obvious method is to record images of the sky and look for fireballs. This is an inexpensive technique which has been used by amateurs for a long time. Nowadays, some weather radar imageries are available to the public on the internet and the UN World Meteorological Office reported 75 national weather radar network around the world. This means that a large number of meteors may be detected using these sources. In fact, in 2012, a planetary science institute helped in the recovery of four meteorites using radar imagery coupled with seismometers to estimate the position where the meteor struck the Earth's surface (Fries, Matson et al 2013). However seismometers are not present everywhere around the world nor are the weather radars. Therefore, for remote areas such as the Australian outback, the DFN project remains a better solution at a low cost.

A real time fireball event detection system, being designed by Jamie Rodriguez, is intended to track any change between two frames in video footage of the night sky. This will permit the ability to zoom in with the integrated camcorder to track the fireball during its fall. While this project remains a prototype, it has a great prospective for tracking long meteor trails but is restricted to the ability to process only one meteor at a time.

2.1.3. Previous meteor detection work using Machine learning algorithms

The meteor detection and landing zone estimation location remained human oriented and had not been automated for many years. Nowadays, with the help of numerous technologies, this process is being automated in different ways but the use of machine learning remains limited in this field. A graduate electrical engineer, Siming Zhao, applied Support Vector Machine (SVM) classification algorithms to give the computer the ability to learn how to cluster fireball event data, obtained from radar observations (Zhao, 2010). His project was successful in the identification of meteors with a trajectory perpendicular to the radar beam but had difficulties in the detection of other meteor trails due to the poor quality of data. Thus, his project has demonstrated once again the high potential of machine learning in meteor detection.

2.1.4. Overview of Current detection process

As briefly explained in section 1.3, an explicitly programmed script was developed by Dr. Martin Towner, a DFN member passionate in planetary sciences, to detect in real time the particular meteor appearing on the collected images. This program was designed to run directly on the observatory processor and hence was limited in processing time and memory. The detection process consists of a succession of complex filters to thoroughly examine the close dots or dashes that could be the result of a meteor trail. Due to the compromise made between the execution time and the performance, the program failed to identify some of the meteor streaks and misclassified other linear patterns such as the Orion's belt. On the other hand, it dismisses airplane and satellite streaks by checking and comparing their trajectory and location in the following images. Indeed, a satellite or aircraft moves so slowly that it is seen as a solid line on multiple exposures and hence easily identifiable.

2.2. Machine Learning

2.2.1. Overall view

Machine learning was defined by Arthur Samuel in 1959 as a “field of study that gives computers the ability to learn without being explicitly programmed”. In other words, machine learning algorithms provide computers the ability to develop their own intelligence based on their own experiences in the goal of accurately predicting or

classifying data. Machine learning has been implemented to a wide variety of application areas, from medicine to pedestrian detection including advertising (Brian et al, 2010), finance (Stephan, 2015), military (DesJardins, 1997), and astronomy (Bailer-Jones, 2009). The potential of machine learning is being developed at every level within the era of big data management. Machine learning algorithms are categorised in three main classes: Supervised learning, unsupervised learning and reinforcement learning algorithms.

2.2.2. Supervised learning algorithms

In supervised learning, the computer is presented with a set of sample data and its corresponding output, also called training set. The computer will then improve its prediction accuracy by reducing the error between the predicted value and the correctly labelled value, to reach a general model that best maps inputs to outputs (Bishop, 2006). Those algorithms have started to be integrated into diverse applications to automate the prediction or classification of data with an uncountable number of features. There are several supervised learning algorithms but each is suited to a particular problem. The canonical types of learning problems, in machine learning, are classified from the object they are trying to predict (Daume, 2014). The list below highlights the four primary supervised learning categories:

- Linear Regression: predicts a real value. For example, the price of a house may be approximated from the size of the land. Plotting the prices with respect to the surface area obtained from a few quoted houses may show that there is a relationship between those features. Therefore by fitting a trendline to the data set it is possible to extrapolate or evaluate the price of an unquoted house, based on its surface area.
- Binary classification: predicts a Boolean value either true or false. For example, the algorithm is required to identify whether an email is a scam or not based on its content.
- Multiclass classification: predicts what class the input belongs to. A common example is the hand written digit recognition problem. In this example, the hand written digits are required to be classified into 10 classes from 0 to 9.

- **Ranking:** predicts sorted outputs in order of relevance. For example, search engines sort the websites by order of relevance to users query.

However, each type of problem can often be solved by utilising several machine learning algorithms. For some cases, an algorithm may even be used for multiple types of problems such as Neural Networks which can perform binary classification and multiclass classification. Therefore the choice of an algorithm also depends on what the algorithm is designed for and the available data. The most common algorithms cited below represent half of the existing ‘supervised’ learning algorithms (Daume, 2014):

- *Decision trees* are a typical learning algorithm which is used to estimate the best decision based on the different alternatives and their resulting expected values. In other terms, the decision tree can be seen as a flowchart where the root is the start of the flowchart and the leaves are the possible outcomes, but only one outcome is predicted from the conditions of the branches.
- *k-Nearest Neighbours* estimates a decision from a decision made in a similar situation. Compared to decision trees, K-nearest neighbours algorithm uses all the features equally.
- *linear regression* are the simplest approach for modelling a function that maps the independent inputs to continuously dependant outputs. This algorithm is most often used to extrapolate the data.
- *Bayesian networks* are a complex algorithm based on Bayesian statistics to represent the conditional independencies of different variables in the model. The nodes of the network depict the variables while the link between them are their relationships.
- *Perceptron* is a simple algorithm for binary classification based on linear relationships and a threshold function to set the boundary of both classes.
- *Neural Networks* are a classification algorithm founded on multiple layers where the nodes of each layer are enabled by the linear relationships with the nodes of the previous layer. This algorithm provides the ability to deal with an indefinite number of features but may be limited to the hardware performance.

- *Logistic regression* is a regression technique to model input-output relationships where the output variable is discrete rather than continuous. This algorithm was designed for binary classification problems (Dreiseitl and Ohno-Machado, 2002).
- *Support Vector Machine (SVM)* is a classification algorithm that establishes a linear boundary also called ‘hyperplane’ between the classes. Compared to logistic regression, SVM finds the hyperplanes that maximizes the margin between two classes rather than minimizing the number of misclassifications. SVM can also be adapted to nonlinear relationships by transforming the input data using *kernel trick* to perform linear separation.

2.2.3. Unsupervised learning algorithms

Unsupervised learning algorithms, unlike supervised learning algorithms, do not learn from labelled data, leaving it on its own to find structure in its input (Ng, 2013). These algorithms, which are implemented for data mining, are founded on two main techniques: density estimation techniques and feature extraction techniques (Jordan and Bishop, 2004). Clustering techniques such as google news, gathers the publications into cohesive groups (Ng, 2013).

2.2.4. Reinforcement learning algorithms

Reinforcement learning is a reward based approach. Similar to unsupervised learning algorithms, reinforcement learning algorithms are not given any labelled data but learn from the consequences of their actions and an additional reward signal. When the action performed is the desired output, the algorithm is rewarded with a numerical value. The algorithm then learns how to select the parameters that maximize the accumulated reward over time (Szepesvari, 2009). Thus, these algorithms are more specific to adaptive control applications (Ng, 2013).

2.2.5. Machine learning algorithm selection

In the objective of recognising patterns in images such as fireball events, unsupervised learning algorithms and reinforcement learning algorithms were not appropriate due to their uncertain efficacy and limited use in such applications. Therefore a supervised learning method was the most appropriate.

Moreover, the selection of a machine learning algorithm among the extensive list described previously turns out to be narrowed down by the problem and the available resources. In the presented scenario, the images needed to be classified as either positive (containing fireball events) or negative (otherwise), hence the need for a binary classification algorithm. From the quick overview of each algorithm in section 2.2.2, the logistic regression, SVM, neural network and Bayesian network were the most appropriate for a binary classification using multiple features. However, the neural network was found to more suitable for the project goal.

Despite the logistic regression being a very powerful algorithm that can be methodically tuned to efficiently separate some data, it is extremely computationally expensive due to the calculation of each derivative to find the global minimum of the cost function (Ng, 2013). Moreover, the greater the number of features, the bigger the degree of polynomials, resulting in an even more complex model. Therefore this algorithm was not suitable for the intended classification problem.

SVM and Bayesian networks have also shown promising performance in pattern recognition applications however these algorithms are often coupled with other algorithms or methods to produce the desired outcome. For example, Bayesian networks have been added to Hidden Markov Models (HMMs) in several pattern recognition problems to extend the capabilities of the original HMMs. However, due to their unreliable results as well as the time constrain to learn each algorithm and implement them on Python, the choice was narrowed down to the Neural Network algorithm.

2.3. Neural Network

2.3.1. Neural Network origin

Artificial neural networks are a refinement of the perceptron algorithm to design non-linear models inspired by human neural networks. In the Era of big data, the importance of neural network is fundamental as it allows to deal with massive amounts of data and models non-linear relationships to classify them. In many cases, artificial neural networks have outperformed a human brain as it has the capacity to take a large amount of features and make accurate predictions. This algorithm being a succession of matrix operations perform faster than a human brain and will keep improving with the improvement of computer hardware.

The human brain has been proven through scientific experiments to have the capacity to be retrained to learn irrational things. For instance, an experiment have demonstrated that when rerouting the auditory cortex with an optic nerve, the auditory cortex would learn how to see. Similar experiences have proven the powerful ability of the human neural network to learn and make prediction hence the development of a mathematical model to mimic how the human brain learns and responds to unseen stimulation. A human neural network is constituted of hundreds of billions of neurons, interconnected with each other, that are responsible for the communication between the sensory organs and the muscles. Neurons are electrically excited. The electrical signal travels from the axon of one neuron to the synapse of another until the activation of the correct neuron to respond adequately to the stimuli. Each neuron will enable the transmission of the signal through the axon, depending on the signal they receive from the synapses. However, some of the synapses may exert more influence than others on the performance of the neuron which raises the notion of synaptic weights. Illustrated, by the following diagrams (Figure 21), is the comparison between a neuron structure and a perceptron model. Each synapse link is weighted to produce different output based on its importance. Unlike a biological neuron, a perceptron computes the sum of the products of each input by their weights and compares it to a threshold to send an activation signal.

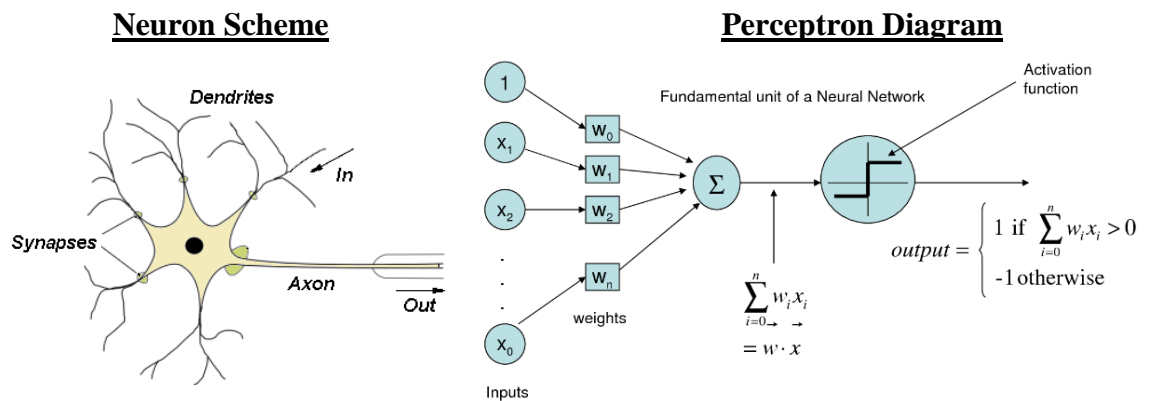


Figure 1: Neuron and perceptron illustration (Motor Neuron 2015)

Although in a biological neural network, neurons are connected to each other in an indeterminate way, the computer representation is formed with layers of neurons where each neuron of a layer is connected to every other neuron of the previous and next layers, as illustrated on Figure 2.

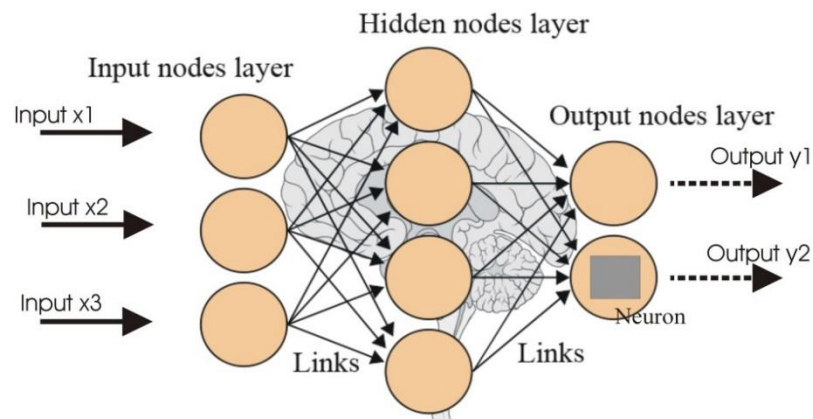


Figure 2: Artificial intelligence neural network nodes (Tadiou. K, Mamadou, 2015)

The illustration above represent a 3 layer network with three inputs (features) fed into 4 hidden units. These hidden units are then fed into two outputs. One extra node, also referred to as bias, is present in each layer but rarely represented on the diagrams as it always outputs a constant value, generally 1. The bias nodes are necessary for the neuron computation function. Just like the brain's neuron network, information flows in, is processed by the neurons, and flows out. The layers between the input and output layers are called hidden layers due to the fact that they are not directly seen. Increasing the

number of hidden layers tends to raise the complexity of the relationship between the inputs and outputs which could either improve the algorithm performance or, on the contrary, reduce its overall performance on unseen inputs.

2.3.2. Applications of Neural Networks

Artificial neural networks become prevalent in classification problems. The main limit of neural network resides in the limit of the hardware and data available. That is, as computers keep improving in performance, speed and memory, the use of neural networks to solve complex problems keeps growing. The most common use of neural networks is to perform multi-class classification where the system is given more than one category to distinguish. In fact, neural network algorithms have reached such good performances that they are used by banks to process cheques and by post offices to read addresses (Michael A. 2015). The most important aspect of this algorithm is its powerful ability to recognize patterns within an image.

2.3.3. Neural network terminologies

Some rudimentary terminology, specific to the mathematical model, must be first defined for a better insight. The inputs of the model will be symbolized by the matrix X where the number of training examples is denoted by m and the number of features is defined by n . The output of the model is expressed by the matrix Y which associates the input to its corresponding category. The hidden units are expressed by $a_i^{(j)}$ which corresponds to the activation unit i in layer j . Capital theta, Θ , represents the matrix of the weights controlling function, mapping from layer j to the next layer $j+1$. Generally if the network has s_j units in layer j , s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$. The sum of the inputs, scaled by their corresponding weights θ of a layer j , will also be represented by a matrix $Z^{(j+1)}$. The neuron function is defined by the function g . And finally the resulting expression for the computation of the output with respect to the inputs, also called the hypothesis, is denoted by $h_{\Theta}(X)$.

2.3.4. Activation function

The activation function is the function used to propagate the signal from one neuron to the next neuron, based upon the input values. A simple activation function is the sign function, which outputs a binary value determined by the sum of the input links and a

threshold value. However, for the network to learn how to make the right decision, the algorithm must learn from the impact on the output, resulting from small changes in the weights. Thus, with sign a function, a small change in the weights or bias of any single unit in the network, can either make no change or completely change its binary output. While this change might be beneficial for the prediction of a specific output it is likely to affect the prediction of the other outputs in an unsuitable way. Therefore a hyperbolic tangent function, also called s sigmoid function, was introduced to overcome this problem.

Sigmoid functions are similar to sign functions, that is, they transmit signals to the next set of neurons depending of its own inputs, but they are designed so that small changes in their weights and biases makes a small change in their output. By doing so the network can learn how to gradually modify the weights and biases so that the network gets closer to the desired behaviour. Sigmoid nodes are mathematically represented by the sigmoid activation function (Eqn (1))

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}} \quad \text{Eqn (1)}$$

The figure below compares both activation functions. The graph on the left illustrates the sign function which a simple step function that returns a binary value. On the other hand, the sigmoid function, as shown on the right, returns a real decimal number that always varies as the input varies.

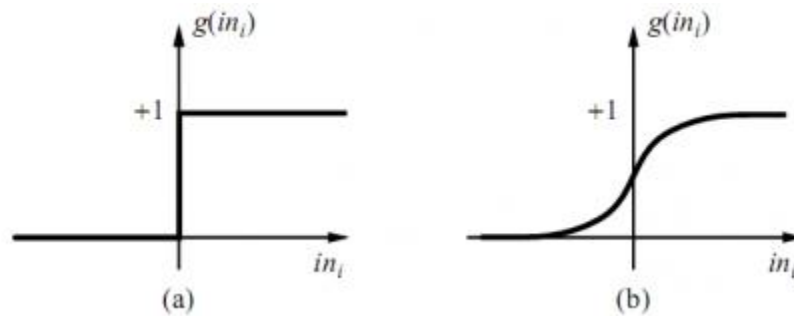


Figure 3: Graphical representation of (a) perceptron and (b) sigmoid activation function (Perceptron 2012)

2.3.5. Feed forward Propagation

Forward propagation comes from the fact that the inputs of the first layer are propagated through the network to calculate the activation of each node of each layer sequentially. This process is vectorized for efficiency. Hence the activation of each node, illustrated in Figure 2 is calculated as follow:

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g(Z_1^{(2)}) \quad \text{Eqn (2)}$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(2)}x_0 + \Theta_{21}^{(2)}x_1 + \Theta_{22}^{(2)}x_2 + \Theta_{23}^{(2)}x_3\right) = g(Z_2^{(2)}) \quad \text{Eqn (3)}$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(2)}x_0 + \Theta_{21}^{(2)}x_1 + \Theta_{22}^{(2)}x_2 + \Theta_{23}^{(2)}x_3\right) = g(Z_2^{(2)}) \quad \text{Eqn (4)}$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(3)}x_0 + \Theta_{31}^{(3)}x_1 + \Theta_{32}^{(3)}x_2 + \Theta_{33}^{(3)}x_3\right) = g(Z_3^{(2)}) \quad \text{Eqn (5)}$$

$$a_4^{(2)} = g\left(\Theta_{40}^{(4)}x_0 + \Theta_{41}^{(4)}x_1 + \Theta_{42}^{(4)}x_2 + \Theta_{43}^{(4)}x_3\right) = g(Z_4^{(2)}) \quad \text{Eqn (6)}$$

$$h_{\Theta_1}(x) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)} + \Theta_{14}^{(2)}a_4^{(2)}\right) \quad \text{Eqn (7)}$$

$$h_{\Theta_2}(x) = g\left(\Theta_{20}^{(2)}a_0^{(2)} + \Theta_{21}^{(2)}a_1^{(2)} + \Theta_{22}^{(2)}a_2^{(2)} + \Theta_{23}^{(2)}a_3^{(2)} + \Theta_{24}^{(2)}a_4^{(2)}\right) \quad \text{Eqn (8)}$$

Which can be further simplified, using matrices and vectors, to:

$$Z^{(2)} = \Theta^{(1)}X = \Theta^{(1)}\alpha^{(1)} \quad \text{Eqn (9)}$$

$$\alpha^{(2)} = g(Z^{(2)}) \quad \text{Eqn (10)}$$

When adding the bias unit in the hidden layer $\alpha_0^{(2)} = 1$, the hypothesis is expressed as:

$$\therefore Z^{(3)} = \Theta^{(2)}\alpha^{(2)} \quad \text{Eqn (11)}$$

$$h_{\Theta}(x) = \alpha^{(3)} = g(Z^{(3)}) \quad \text{Eqn (12)}$$

The process for computing the hypothesis function is called forward propagation because inputs of the first layer are propagated to the next layer which proceed to the activation of the hidden layer to then feed the next layer. The exact same steps for a single layer are simply repeated for each layer of the neural network up to the output layer.

2.3.6. Learning algorithms

The next step when implementing a neural network is obviously the learning stage. After a random initialisation of the weights, the labelled inputs of the training set are propagated once to evaluate how bad the network is doing. The training stage is then the process of

optimizing the weights of the network to minimise the error between the predictions and the correct values. This is done through iterations to evaluate the weights that require more adjustment and to ensure a constant reduction of the error. Assuming $(h_{\theta}(x))_i$ is the hypothesis function of the output i_{th} where $h_{\theta}(x) \in \mathbb{R}^K$, the cost function (squared error cost function) of the neural network is to be expressed as:

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \times \log(h_{\theta}(x^{(i)})) \times k + (1 - y_k^{(i)}) \times \log(1 - (h_{\theta}(x^{(i)})) \times k) \right] \quad \text{Eqn (13)}$$

Where $y \in \mathbb{R}^K$ is the matrix of labelled output values and k is the number of classes. Therefore for a binary classification the cost function reduces down to:

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \times \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \times \log(1 - h_{\theta}(x^{(i)})) \right] \quad \text{Eqn (14)}$$

To put it another way, this equation determines how close the prediction $h_{\theta}(x^{(i)})$ is compared to the correct label $y^{(i)}$. Thus, minimising the cost function $J(\Theta)$ implies finding the best parameters Θ that produce the best prediction of the training data.

Nevertheless, plotting the function to visually find the global minimum is not feasible due to the tremendously large number of parameters. For the same reason it is not possible to compute the global minimum from the second derivative. Fortunately minimising the cost function can be done automatically with the help of optimisation algorithms. A common optimisation algorithm called gradient descent has been chosen for its simplicity and efficacy. However, more advanced techniques based on the motion of a ball rolling down a hill, produce higher performance at the expense of being more computationally costly and harder to tune.

2.3.7. Gradient Descent

As always, to understand how things work it is essential to break them down into smaller parts. In this case, to understand the principle of gradient descent it is easier to reduce the cost function to a rather simple one with only a few parameters. Suppose Figure 4 illustrates a cost function with only two variables where θ_0 and θ_1 are the weights of the two input links (Michael, 2015). This function can be imagined as being a landscape with

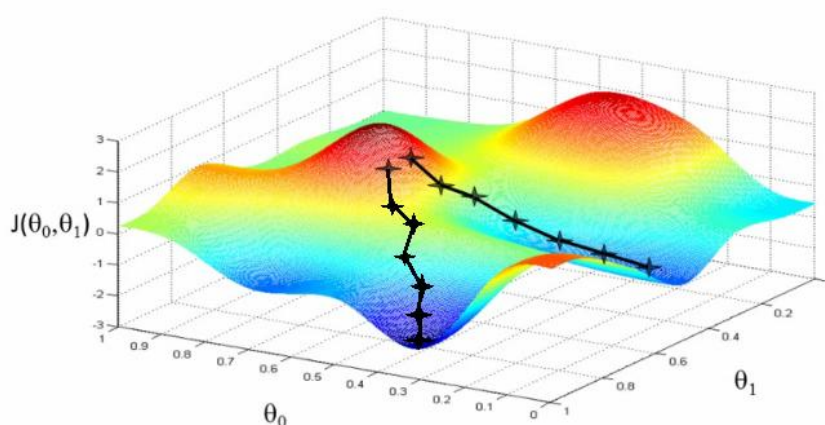


Figure 4: Cost function with two variables (Ng, Andrew 2015)

hills and valleys. From this perception and applying the law of physics, if a ball were placed at any location on the landscape it will roll down to the local minimum. Therefore a similar approach, ignoring the law of physics for the motion of the ball, is followed by the gradient descent algorithm. To clarify without entering into too much detail and mathematical proof, the gradient descent starts at any point defined by randomly initialised parameters and then estimates the steepest descending slope based on the derivative of a small change in θ_0 and θ_1 . Once the slope is found the algorithm steps down at a rate determined by what is called the learning rate. The process is repeated until the ‘ball’ has eventually reached the bottom of the valley. It can be observed in Figure 4 that two starting points relatively close from each other can lead to two different local minimums. The Gradient Descent algorithm is efficient as it just requires a few iterations to reach a local minima. Nevertheless, this method might fail for diverse reasons such as setting the learning rate too high. Indeed, this would result in a change of position towards the bottom of the valley but on the opposite hill at a higher altitude. Therefore the learning rate and the number of iterations are also important parameters, which is discussed in section 2.3.10. On a side note, Gradient Descent permits to minimise the cost function

but it rarely reaches the global minimum of the function, hence the need for better and complex optimisation algorithms.

2.3.8. Back Propagation

Back propagation is among the most effective approaches to machine learning when the data is large. Although it has been discussed in the previous section how Gradient Descent helps optimizing the model, it has not been explained yet how this algorithm helps to achieve a better prediction. Unlike forward propagation, back propagation, as its name suggests, starts from the output of the network and back propagates the error layer by layer up to the first hidden layer. It is therefore obvious now that to estimate the error between the prediction and the known output value the algorithm must start with an initial prediction with randomly initialised parameters. Then the back propagation algorithm computes the partial derivatives $\frac{\partial J}{\partial \theta_j}$ of the cost function J with respect to any weights or bias θ_j in the network. This parameter is also referred as $\delta_j^{(l)}$ representing the error of the activation unit $a_j^{(l)}$ in unit j in the layer l . Explicitly this derivative term demonstrates how quickly the cost changes when applying a small change in the weights or biases (Ng, 2015). For example, if the output does not depend much on an activation output, then a change in the parameter of this unit will result in a little change in the output.

For each node the error $\delta_j^{(l)}$ is computed to estimate how the algorithm might have done better. The following equations shows how the error is propagated and expressed in terms of the weights. These equations hold for a network of three layers (one hidden, on input and one output layer). Nevertheless, the same principle can be followed for any number of layers.

$$\delta_j^{(3)} = a_j^{(3)} - y_i \quad \text{Eqn (15)}$$

$$\delta_j^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(Z^{(2)}) \quad \text{Eqn (16)}$$

Where $g'(Z)$ is the gradient of the activation function and is calculated as follow:

$$g'(Z^{(i)}) = \frac{\partial}{\partial Z} g(Z^{(i)}) = g(Z)(1 - g(Z)) \quad \text{Eqn (17)}$$

This is where the gradient descent algorithm comes in handy. Indeed, the weights of the network can be updated using gradient descent, for each layer, based on Eqn (16). While this explanation is a quick overview of the back propagation, it gives a good insight on the way it computes the error.

2.3.9. Testing

Before assuming the artificial neural network has successfully learnt and is ready to predict any situation for which it has been trained, it is essential to test its ability on another set of data labelled the test set. For example, it can be stated that a student has fully learnt the material given during a course when they completely understand the supplied material. The only way this can be known is by assessing the knowledge of that student, with an appropriate test different from the examples provided in the course but related to what has been communicated. This test will evaluate the ability of the student to use their knowledge to answer questions that they have never seen before. From this stems the concept of generalization, which is the fundamental goal in machine learning (Hal, 2014). A similar method is undertaken to assess the capability of the artificial neural network to generalise the data. In this case an appropriate test set would be smaller but similar to the training set. Therefore a typical way is to first shuffle and then split the labelled data into two sets proportioned 70% for the training set and 30 % for the test set.

Then, the marking criteria to measure the classifier performance depends on what the goal of the neural network is. Typical performance measures include assessment on accuracy, precision, recall, confusion matrix and ROC plots (Powers, 2007).

Confusion matrix, or contingency table, is the measure of True Positives (TP), True Negatives (TN), False Negatives (FN) and False Positives (FN) of the algorithm based on the data set. A confusion matrix for a binary classification is illustrated on the next page.

| | | Predicted Class | |
|--------------|---|-------------------------|-------------------------|
| | | 1 | 0 |
| Actual Class | 1 | True Positives (TP) | False Negatives (FN) |
| | 0 | False Positives (FP) | True Negatives (TN) |

Figure 5: Confusion matrices

Accuracy is the proportion of correctly classified data against the total number of labelled examples in the set. Thus a high accuracy means the system performs well on the specific set but does not necessarily imply better performance on target task. This emanates the concept of ‘overfitting’ the training set which will be discussed later on.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \quad \text{Eqn (18)}$$

On the other hand, Precision is defined as the proportion of true positives against all false positive predictions and is expressed as:

$$Precision = \frac{TP}{TP + FP} \quad \text{Eqn (19)}$$

Recal, also called sensitivity, defines the proportion of true positives against all the positive samples and is calculated as follow:

$$Recall = Sensitivity = TPR = \frac{TP}{TP + FN} \quad \text{Eqn (20)}$$

The specificity or true negative rate is the proportion of true negatives against all the negative samples as expressed below:

$$Specificity = TNR = \frac{TN}{TN + FP} \quad \text{Eqn (21)}$$

Therefore the false positive rate is the opposite of the specificity or 1-TNR.

A typical method to illustrate the performance of a binary classifier is to plot the Receiver Operating Characteristics (ROC) curves which encompass all the measures explained.

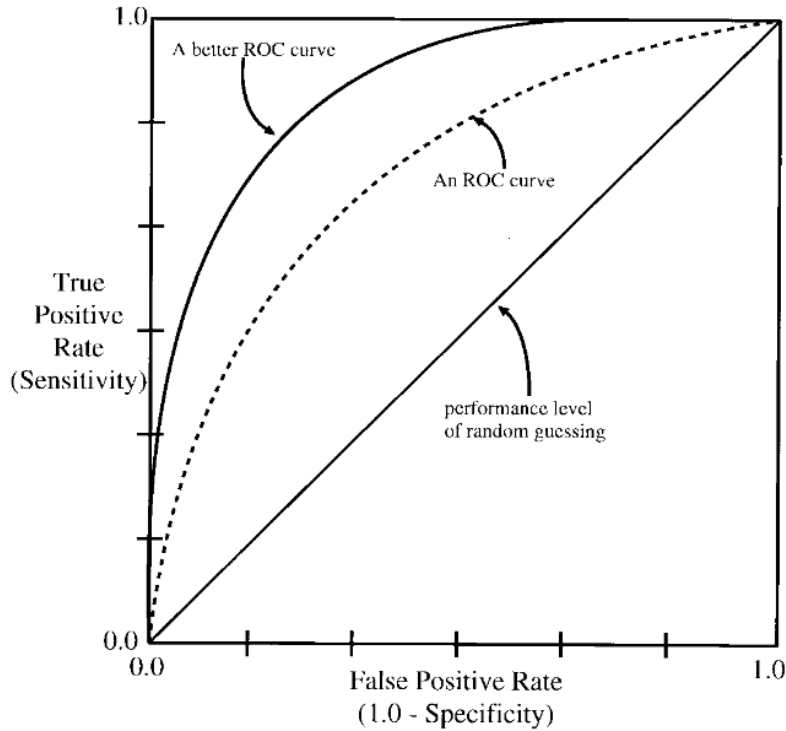


Figure 6: Two typical ROC curves and an expected performance level of a random guessing system (Woods, 1997).

This graph presents the True Positive Rate (TPR) against False Positive Rate (FPR) as the threshold of the output node is varied (Woods, 1997).

The performance of the classifier is then determined by calculating or visualising the area under the ROC curve. The greater the area the greater the power of the classifier.

F-measure can also be used as a single evaluation of how well the system behaves but does not tell the whole story behind it. In fact, these measures are only here to help interpret the data but should not be used independently.

$$F_{measure} = \frac{2 \times precision \times recall}{precision + recall} \quad \text{Eqn (22)}$$

2.3.10. Performance improvements

Although a neural network might be able to be trained to its optimal performance, two problems may arise. Either the network still performs poorly on the training set even though the global minimum of the cost function has been reached. Or, the model performs

extremely well on the training set but poorly on ‘unseen’ data. In technical terms, the model may underfit the data and hence suffer from ‘high bias’, or overfit the data and suffer from ‘high variance’. To diagnose what the model is suffering from and to determine how to improve it, it is important to take into consideration the performance measures and to have a deep understanding of the algorithm.

Improving the algorithm means modifying the parameters of the network or the training set in order to perform what it was designed to do on unseen data. In general, the accuracy achieved by the algorithm on the training and testing set is highly regarded. However, depending of the purpose, the recall rate and precision might be considered more important. For instance, if an artificial neural network was to be implemented to diagnose whether a patient has a cancer or not, it is safer to allow some false positives than to allow some false negatives. Indeed, if a patient has been misdiagnosed with cancer they will go through additional examination but they are not in a critical situation. On the other hand, if the patient has cancer but was misdiagnosed by the algorithm, it is a grave situation.

Another important factor in the evaluation of the performance of the classifier is about the quality and quantity of data. For instance, if a training set is made out of 90 cases without cancer and 10 with cancer, and the algorithm has misclassified the 10 positive samples, then the accuracy is 90% but in reality the system performs very poorly. Therefore a typical start is to have the same number of examples for each class.

Overfitting happens when the algorithm fails to generalise, or in other words when it performs extremely well on the training set but performs poorly on the test set. Although this may seem normal due to the fact that the algorithm has only learnt what it has ‘seen’, it is possible to train it in a way that it does just as well on ‘seen’ and ‘unseen’ data.

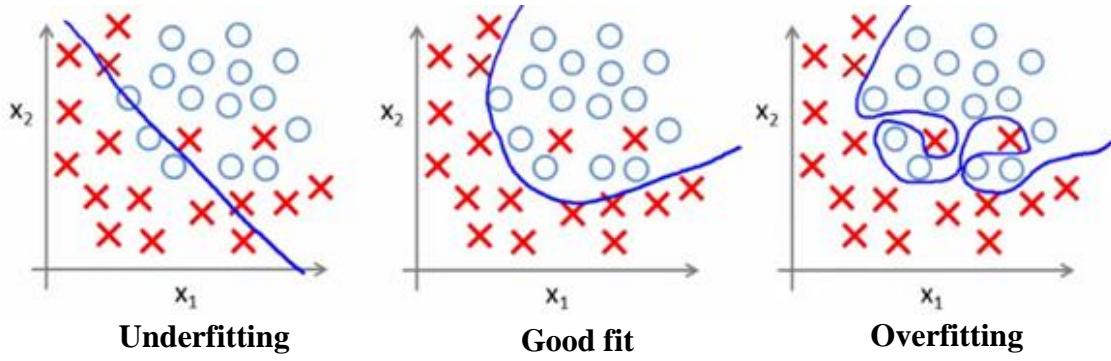


Figure 7: Under- and Over-fitting examples (Ng, 2015)

Underfitting is the opposite of overfitting which means the model is too simple with regards to the data it is trying to model. An example of underfitting and overfitting in logistic regression is illustrated in Figure 7 but the idea is the same for the artificial neural network, although the visualisation is much more complex.

Underfitting may be due to the fact that the network was not trained for a long enough period resulting in the cost function not being sufficiently minimised, or the number of hidden units is not enough to represent the desired mapping.

In most cases, overfitting remains the biggest issue. To prevent overfitting, several options can be undertaken such as (Ng, 2015):

- Reducing the number of hidden units or hidden layers
- Increasing the number of training samples
- Increasing the set of features
- Stopping the training before the cost function is extremely minimised.
- Adding a regularisation term to the cost function which penalises the weights of the network and encourages smoother network mapping.

In the last option, the cost function of the neural network becomes:

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \times \log(h_{\Theta}(x^{(i)})) \times k + (1 - y_k^{(i)}) \times \log(1 - (h_{\Theta}(x^{(i)})) \times k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2 \quad \text{Eqn (23)}$$

Or, for binary outputs:

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \times \log(h_{\Theta}(x^{(i)})) + (1 - y^{(i)}) \times \log(1 - h_{\Theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \Theta_j^2 \quad \text{Eqn (24)}$$

This is where the cross validation set comes in handy. Plotting the learning curves which are a plot of the cross validation cost function and the training cost function against the parameter being tested, is an efficient way to select the parameters values. The constant lambda governing the regularisation term can hence be determined by plotting the learning curves. By varying the regularisation parameter as illustrated by the left graph in Figure 8 it is possible to find the most suitable value that will fit the data reasonably well fit. From the graph, the best hyperparameter lambda that generalises the predictions is in the orange region. Similarly the correct number of examples is when both curves converge to a relatively close point while keeping the training examples as low as possible as shown in the right graph below.

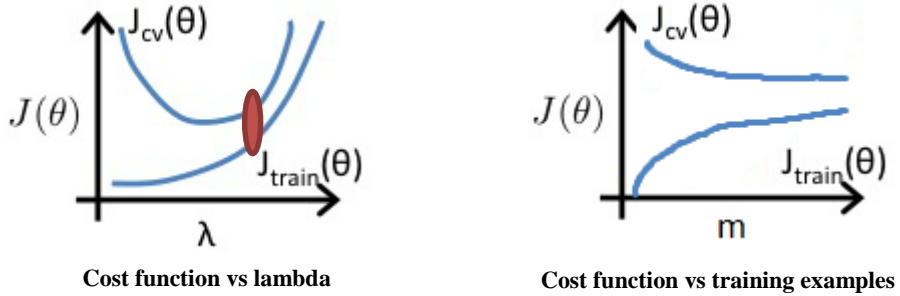


Figure 8: Learning curves examples (Ng, 2015)

On a side note, hyperparameter refers as the variables that are set before optimizing the network's parameters (weights). These parameters are generally set up by hand by visual inspection of the learning curves or with the help of some search algorithms. Hyperparameters can be classified in two main categories: 1) where the model hyperparameters encompass the number of hidden layer, number of hidden units, number of features and weight initialisation; and 2) where the learning algorithm hyperparameters include the learning rate, number of training iterations and other variables specific to other learning algorithms.

2.4. Python

Python is a, script-based, high-level programming language designed by Guido Van Rossum to improve readability and simplicity of other programming languages such as C, C++, C#, Java and many others. Python allows for object-oriented programming and structured programming while considerably reducing the cost of program maintenance thanks to its efficient syntax. Thus, making it easy to learn and use in any application. In addition, Python includes a large standard library optimised for speed and efficiency but also allows for the development of additional modules and packages which encourages program modularity (Rossum, 1997). On a downside, its execution speed is generally slower than most other programming language, but on the developing side, it is much quicker to progress. Therefore for the reasons outlined above, Python was selected by the DFN team as a primary high-level language and hence was also used to implement the detection algorithm.

Chapter 3: Image pre-processing

3.1. Purpose of Filtering raw images

Data preparation is the most important step when dealing with tremendous amounts of features such as the number of pixels in high resolution images. The dimensions of the images captured by the DFN cameras are 7360x4912 pixels which corresponds to 36,152,320 pixels for each channel. Having such a huge vector inputted straight into the neural network would result in a number of links between the input and hidden layer equal to the number of features multiplied by the number of hidden units. This is unconceivable due to the limited performance of the computer available but also because it would require an enormous number of examples in order to learn what to look for. Therefore down sampling was required.

There are two ways to down sample an image: either by splitting the image into smaller tiles or by reducing the resolution. However, the main challenge was to keep enough information in the image while minimising its size.

Meaningful and clear data is the key for a high ‘prediction’ performance. For a human to be able to learn from a set of data (in this case a set of pictures) and identify the features that belong to the different classes, they must be given reasonably clear data that is relatively easy to distinguish. The same process is followed for a computer neural network. The computer must be fed with pre-filtered data to prevent confusion and false predictions. For example if the algorithm is fed with the pixels of an image representing the digit ‘9’, it will learn what features are required to estimate it is a ‘9’ and not a ‘8’. A neural network must be given enough information on the class to successfully distinguish between the different classes. The same applies for a binary classification problem. The neural network must be given sufficient information to distinguish between a meteor and non-meteor image. Therefore a large number of steps was required to convert an untouched image containing stars, surrounding objects, the moon, aircrafts, satellites, the Milky Way and other undesired patterns into a reduced matrix with two groups: meteors and not meteors.

Another benefit of the image filter was to address a few issues encountered by the current detection code. Indeed, it was noticed that the Orion's belt often came up as a positive identification, due to the close distance between the three stars and the perfect alignment, which might be interpreted as 3 dashes of a fast meteor streak.

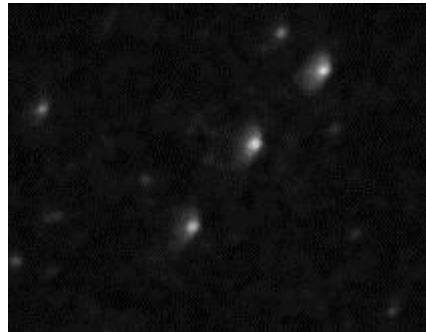


Figure 9: Orion's belt classified as a meteor by the current program

However, it is important to note that to take full advantage of the neural network algorithm such as its fast processing speed, the pre-filter should not overload the process or complete the whole work. The pre-processing code was implemented for efficiency but did not achieve a perfect result.

3.2. Image Differentiating

3.2.1. Channel selection

An image is generally composed of 3 channels; Red, Green and blue (RGB), that are superposed to create all ranges of colour visible to a human eye. However it is not essential to conserve the colour of the meteor for identification, thus the matrix of the image could be reduced to only one only channel which considerably diminishes the amount of features. Grayscale is the representation of an RGB colourmap to a range of shades of gray by eliminating the hue and saturation information while retaining the luminance. It was noticed that on most fireball images, this scale seemed to reveal the meteor trail the best. However, the blue and red channels were the worst to clarify the meteor streak. The green channel in other hand was very similar to the grayscale but had the advantage of avoiding an operation for the conversion. The channels of a tile containing a meteor streak have been split and shown in the figure below. The streak is barely discernable on the blue channel. The red channel nevertheless also reveals the trail

but the background looks noisier than the green channel. Therefore, the green channel's brightness was given a scale from 0 to 255 and then used as data point.

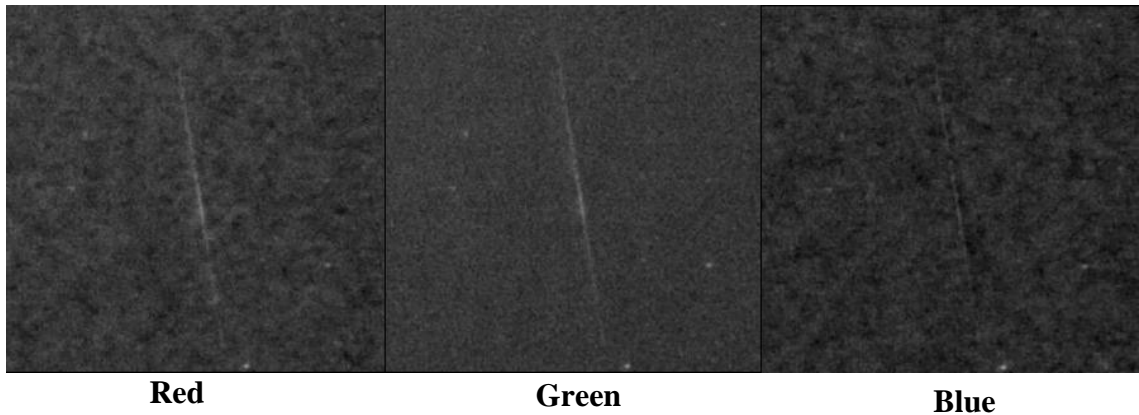


Figure 10: RGB channels

3.2.2. Masking

Due to the vignetting of the fish-eye lens (darkening around the edges of the images), it was difficult to conduct calculations based on the overall brightness of the image. Therefore a mask of diameter 1000 pixels was applied around every image being processed to eliminate this effect.

3.2.3. Background removal

As illustrated by Figure 10, the background of the night sky does not appear completely black due to the light sources around the camera and the light emitted by the stars and moon. The main issue was that the brightness of the background varies with time and location. To enhance the appearance of meteors in the images a dynamic threshold was implemented. As one might think at first glance, the difference between two successive images is so negligible that subtracting one image from the other would eliminate the noisy background as well as all the fixed objects and stars. However, in practice the noise is never constant nor, the position of the stars. Indeed because of the rotation of the planet and the effect of the fish eye, stars are moving in a non-linear motion with varying speed and brightness intensity across the image. Therefore the best solution was to implement a dynamic threshold based on the image properties in order to design a versatile process applicable to any camera.

Multiple solutions have been tested to reduce the noise in the background. First a constant threshold was applied across the entire image but due to the inner disk being generally darker than the outer ring, it turned out that the threshold was either too big that most meteor streaks near the centre of the image would disappear or too low that the outer ring remained very bright. Although it is not very clear on the images below, after applying a threshold similar to the brightness of the background at the middle, the meteor streak near the centre is clarified but the outer ring remains too bright that a meteor of similar brightness would not be clearly identifiable. On the other hand, applying a high threshold that allows the meteors around the horizon to be detected, would considerably reduce the visibility of the meteor trail at the centre.

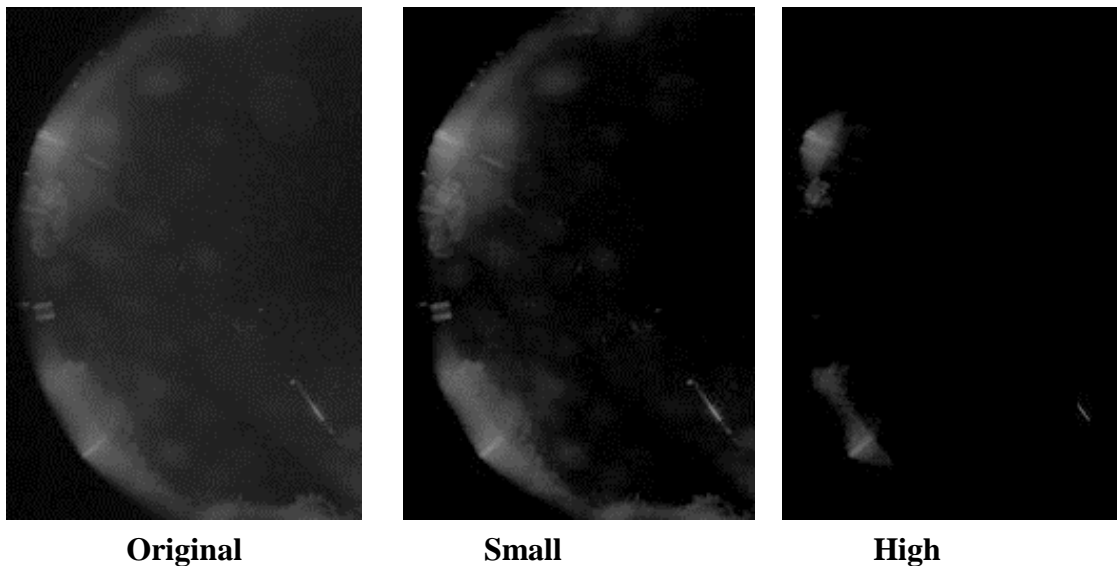


Figure 11: Threshold examples

Hence rather than applying a different threshold for multiple ring diameters, the previous image was used as a base as the lights are more or less at the same place from one image to the next one. Therefore the first image (previous image), after being considerably blurred to smoothen the noise, was subtracted from the next one. However, the brightness drops significantly in the blurring process which does not completely erase the background as it should have. In addition, it was noticed that when the camera's start recording, the brightness of the sky continues to drop. Therefore a test was carried on 20 consecutive photos, captured at the beginning of the night, to find a model that predict the ideal threshold, based on the properties of the images. The graph shown in Figure 12

illustrates the difference between the mean brightness of the picture being processed, and the mean brightness of the previous picture against the ideal threshold value.

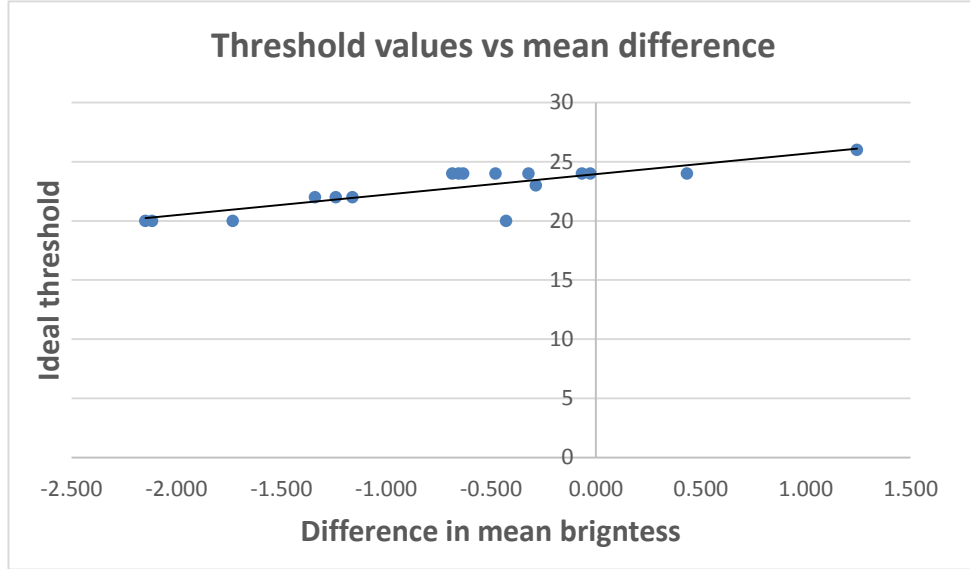


Figure 12: Graph of the threshold and brightness relationship

Fitting a line to the data results in the following expression that best model the threshold relationship.

$$\text{threshold} = 1.73 * (\text{mean}(\text{img2}) - \text{mean}(\text{img1})) + 24 \quad \text{Eqn (25)}$$

Another issue arose from the presence of stars which impacted the accuracy of the prediction. Indeed, a high false positive rate was predicted by a human eye which means that an artificial neural network would likely perform worse. Therefore, the blurring process was reciprocally applied to the previous image, that is, the second image was also blurred, slightly scaled up and subtracted from the previous one. In this case, it did not really matter to increase the brightness of the blurred image by a well-defined model before subtracting from the first image, because stars are generally much brighter than the overall brightness of the background.

Once the stars were isolated, a binary dilation was performed to increase the size of the dots in order to overlap with the dots present on the second image. By doing so, it was possible to entirely erase the stars as well as any fixed objects that have not been removed in the previous steps, leaving the second photo with meteor, airplanes or satellite strikes.

Another more accurate method could have been implemented to calculate the exact position of the stars based on their previous positions, the time elapsed between the exposures and a lens adjustment. But this would have been too much time consuming and computationally costly.

3.3. Down sampling

Having a neat meteor trail was one challenge but passing the right amount of information to the neural network was another major challenge. To avoid having the training stage take a long time to complete its sequence a reasonable number of features for it to calculate would be below 1000 input units. Therefore it was decided upon the completion of a hand digit recognition laboratory from the neural network course taught by Andrew Ng that a sample image of size 25x25 pixels was reasonable for rapid training of the network. However to encompass the most important features in such a small area, resizing was essential. Thus, the size of the resulting image was reduced by half the original size using 'bilinear' interpolation which computes the weighted average of the closest 2 x 2 neighbourhood pixels. Consequently, a neat line was produced for several hashed meteor streaks, while the thinnest streaks were discarded in the down sampling process. Although it is important to note that first, it is uncertain that those thin lines are indeed meteor trails, and second, they are likely to be too small to ever reach the Earth's surface and hence unusable for the triangulation stage. On the other hand, being able to observe those events would permit the opportunity for statistics gathering and a census of meteors to study eventual meteor showers. However, this was not the main aim of this project and was considered as a bonus if it did classify those meteors successfully.

3.4. Slicing

The resulting image was then sliced into tiles of 25x25 pixels to be fed forward through the neural network and be classified as either positive or negative accordingly. Increasing the size of the tile allows for a larger section of the meteor streak to be passed to the neural network and hence provide more information, outside the original box, to the network. This would be somewhat beneficial for distinguishing a cloud edge to a meteor streak with wide gaps between the dashes, as shown by Figure 13b, but this would significantly increase the computational cost. Moreover, such a gap usually happens due to the incredibly fast speed of the meteor (Figure 13a) which means they are likely to be present on multiple tiles and be identified with another tile (Figure 13b and c). Therefore the trade-off, between downsizing and slicing, was essentially made upon the ability of a human to identify a rather small meteor streak as both parameters were varied.

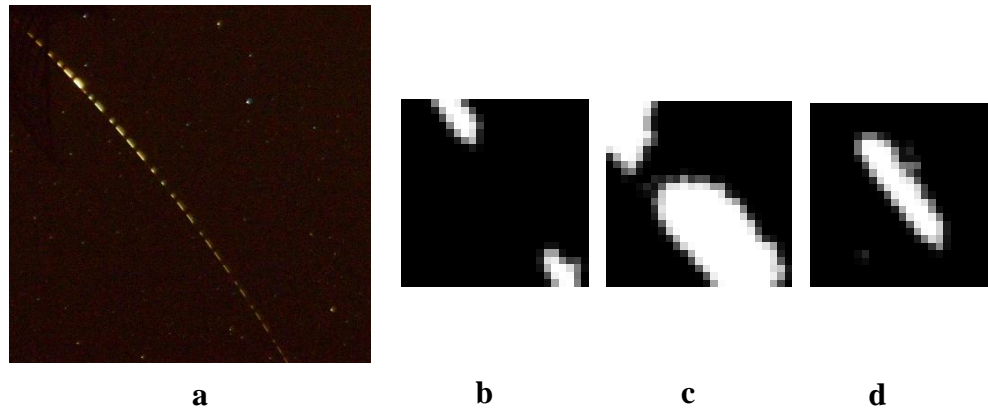


Figure 13: Slicing examples

3.5. Filter steps representation

A visual representation of each step of the image filtering process is illustrated on the next page.

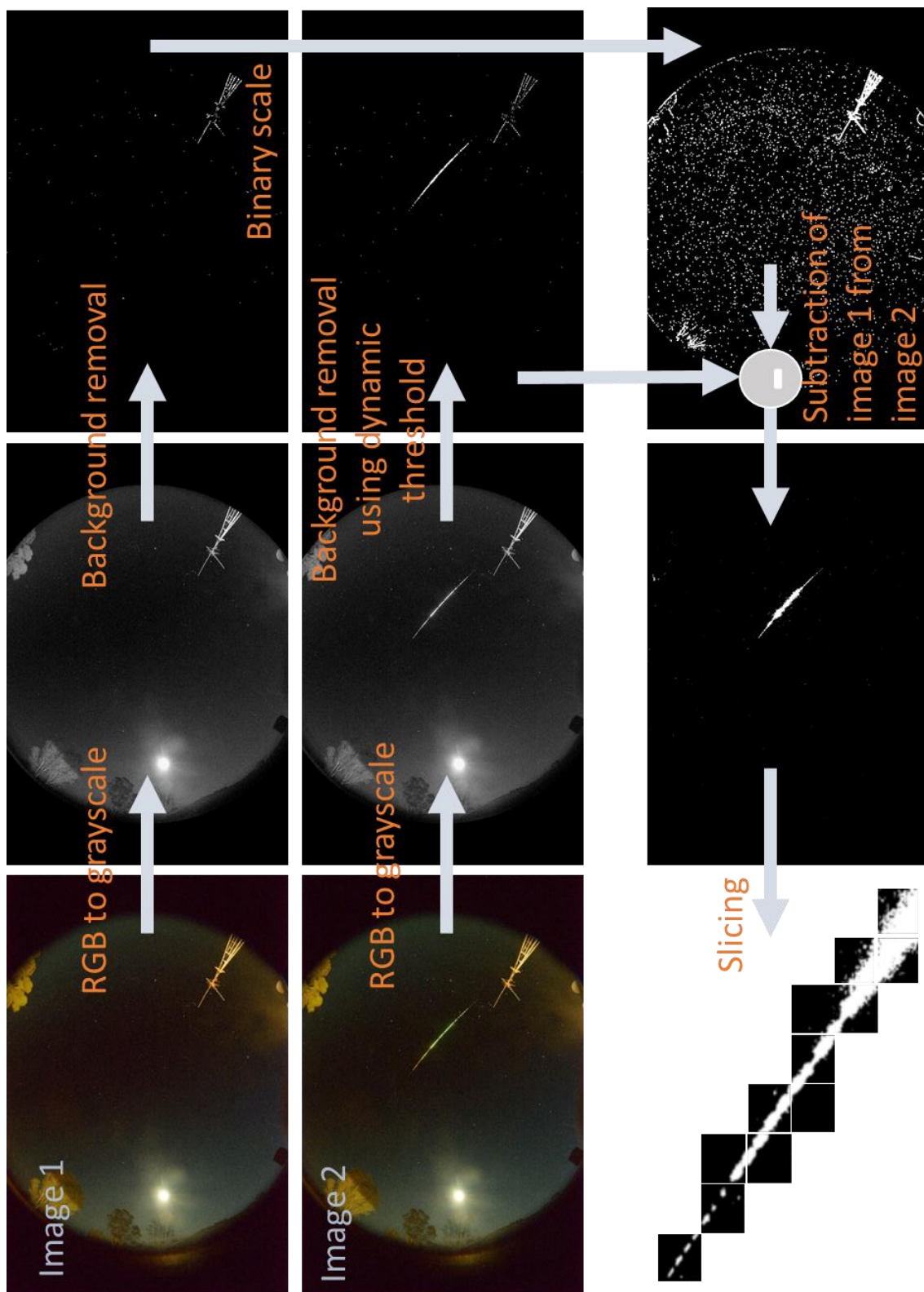


Figure 14: Filtering steps

Chapter 4: Design

The artificial neural network algorithm was first implemented on MATLAB during the accomplishment of the online course on “Neural Networks” (Ng, 2013) from the Stanford University website. Once the script was successfully implemented and tested on hand written digit examples, it was converted to Python. Despite the number of machine learning modules available on Python that offer premade functions to build a neural network, it was decided by the author that designing the algorithm from scratch would develop insight and allow for the modification of the internal functions as desired. Besides, writing the entire neural network algorithm facilitated debugging and permits for future work and development to be carried on without concern in regards of compatibility issues.

4.1. Neural Network Architecture

Designing a neural network implied selecting the number of input neurons, hidden neurons, hidden layers and output neurons. As specified in section 3.3 and 3.4, the number of inputs or features corresponds to the number of pixels in a tile. Therefore a 25x25 pixels tile results in an input vector containing 625 values ranging from 0 to 255 for the brightness of each pixel. Due to the binary classification problem, the output layer possesses only one output unit which takes the value 1 or 0 according to the prediction of the vector fed into the network.

The number of hidden layers and hidden units the neural network should possess to develop its full potential has, however, not clearly been identified. A theorem found by George Cybenko and extended later by Kurt Hornik states that any function F can be approximated with a single hidden layer (Hal, 2014). For this reason, the network implemented was limited to one hidden layer. Nonetheless, the number of hidden units is recommended to be kept between the number of input features and the number of outputs but there is no specific formula to find the ideal number. Moreover, the number of hidden units highly depends upon the number of training examples. Assuming a training set of dimension D and a network with K hidden units and 2 bias nodes for the calculation of the hypothesis of each layer, the total number of parameters is then $(D+2)K$. Therefore

for a large number of training examples the number of hidden units could be large. In addition, because all the hyperparameters of the network are all dependent of each other, different network architectures were tested for each dataset created. The number of hidden units were varied from 1 to 25 with an increment of 5 hidden units. Under 5 hidden units the network was suffering from high bias as the model could never reach a higher accuracy than 70% on the training set. Above 5 hidden units, the difference in the outcome was so small that it was difficult to choose the best parameter. This experiment was restricted to 25 hidden units due to the excessive time required to train the neural network with more hidden nodes. In addition, although the data were not centred and as well oriented as the one used for the hand written digit recognition practice, 20 hidden units were sufficient to obtain a reasonably accurate model which predicted the cross validation and test sets with above 97% accuracy. Consequently it was expected that a similar performance could be attained for a binary classification problem with less hidden units. Hence, for the data obtained from the final pre-filter implemented, 10 hidden nodes seemed reasonable to achieve high performance without considerably suffering from ‘high variance’.

4.2. Datasets

Successful learning of a neural network is achieved by the quality and quantity of the materials provided to the network along with the right parameter selection. When dealing with numerous features, the datasets are usually partitioned in 3 parts: the training, cross validation and test dataset. These sets are strictly independent from each other but made of similar examples which could have been used for any set. A dataset is constituted by an input matrix X comprising of 25x25 tile examples in vector forms and their correct output label in a Y matrix. The training set is destined to show what goal the neural network should aim to reach. However when overtraining the network, overfitting might arise as explained in section 2.3.10, which means that the model performs poorly on the test set and hence does not generalise well. Multiple hyperparameters such as the regularisation term, the number of hidden units, the number of features and the number of examples can be modified to overcome overfitting. The only way to determine the best hyper-parameter value is by assessing the performance of the network on another dataset,

as this parameter is varied. However, the risk of overfitting the new dataset may occur if the selection of the hyperparameter is made on this dataset only. This is the reason why the labelled data are classified into 3 distinct groups with well-defined roles. The training set is the learning step of the network while the cross validation role is to the selection of the hyperparameters and the test set is the evaluation of how well the network generalises.

The 3 datasets were duly prepared by collecting some examples from manually searching the images. About 50 images of fireballs events were pre-filtered giving 200 tiles containing fireball streaks. The same quantity of negative examples were obtained from noisy images containing ‘salt’ (form of noise) or unidentifiable parts of meteor streaks as shown by the images below (Figure 15). The datasets were then made of an equal proportion of positive and negative samples randomly distributed to each dataset with 60% to the training set, 20% to the cross validation set and 20% to the test set.

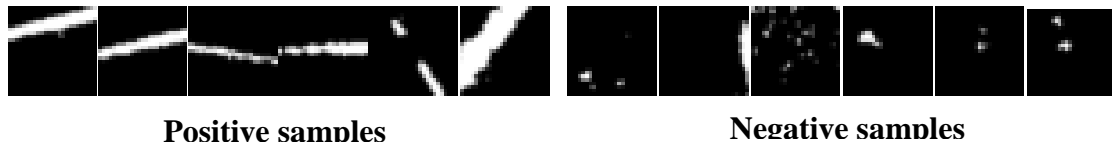


Figure 15: Subsets of positive and negative samples

4.3. Development of back propagation

The back propagation algorithm was also mostly written in vectorized form in order to take full advantage of the capability of Python to compute matrix operation. Therefore the code was optimised for fast processing at the expense of requiring more Random-Access Memory to store each result as it progresses. The computation of the gradient of every activation functions was performed by matrices operation as shown on the screenshot of the code (Figure 16). However, the gradient descent optimisation function was implemented using a function from the ‘SciPy’ module called ‘minimize’. This function offers the ability to minimise scalar functions of one or more variables, using a similar approach to gradient descent, while automatically sectioning a suitable learning

rate. This function is also designed for efficiency and hence stops iterating once the gradient of the cost function stops decreasing.

```
def sigmoidPrime(self,z):
    #Gradient of sigmoid
    return np.exp(-z)/((1+np.exp(-z))**2)

def costFunction(self, X, y):
    #Compute cost for given X,y, use weights already stored in class.
    self.yHat = self.forward(X)
    J = sum(0.5*sum((y-self.yHat)**2)/X.shape[0] + (self.Lambda/2)*(sum(self.W1**2)+sum(self.W2**2)))
    return J

def costFunctionPrime(self, X, y):
    #Compute derivative with respect to W and W2 for a given X and y:
    self.yHat = self.forward(X)

    delta3 = np.multiply(-(y-self.yHat), self.sigmoidPrime(self.z3))
    #Add gradient of regularization term:
    dJdW2 = np.dot(self.a2.T, delta3)/X.shape[0] + self.Lambda*self.W2

    delta2 = np.dot(delta3, self.W2.T)*self.sigmoidPrime(self.z2)
    #Add gradient of regularization term:
    dJdW1 = np.dot(X.T, delta2)/X.shape[0] + self.Lambda*self.W1

    return dJdW1, dJdW2
```

Figure 16: Gradient calculation on Python

Prior to the first feed forward propagation, the parameters (weights) of the network are randomly initialised to permit the backpropagation to adjust the weights depending on the gradient of each activation function. If the parameters were initialised to zero, the gradient of the output would also be zero and after each iteration of the backpropagation all the parameters would change equally, resulting in a small improvement and a slow convergence to the same local minimum. However, as explained in section 2.3.10, the main goal is to reach the lowest point of the valley denoted by the cost function. Hence, the advantages of randomly initialising the parameters with small weights ranging from -0.1 to 0.1. Indeed, first the small weights permit the reduction of the chance of starting at a high point and second, due to the random initialisation, the starting point will vary upon each initialisation resulting in different local optima.

4.3.1. Hyperparameters

a. Number of examples

The number of examples in the training set is a judicious choice as having too few examples will likely end up in an overfitting model while having too many examples will be excessively computationally costly for no benefit. Therefore the cost function of the training set and the cost function of the cross validation set have been minimised and plotted against the number of training examples in the training set. As anticipated, the cost function of the training set overfits the data for a long period until it becomes to converge towards a constant value. However, due to the limited number of available data (190 positive samples in the training set), the training cost function did not seem to converge as shown by the graph Figure 17.

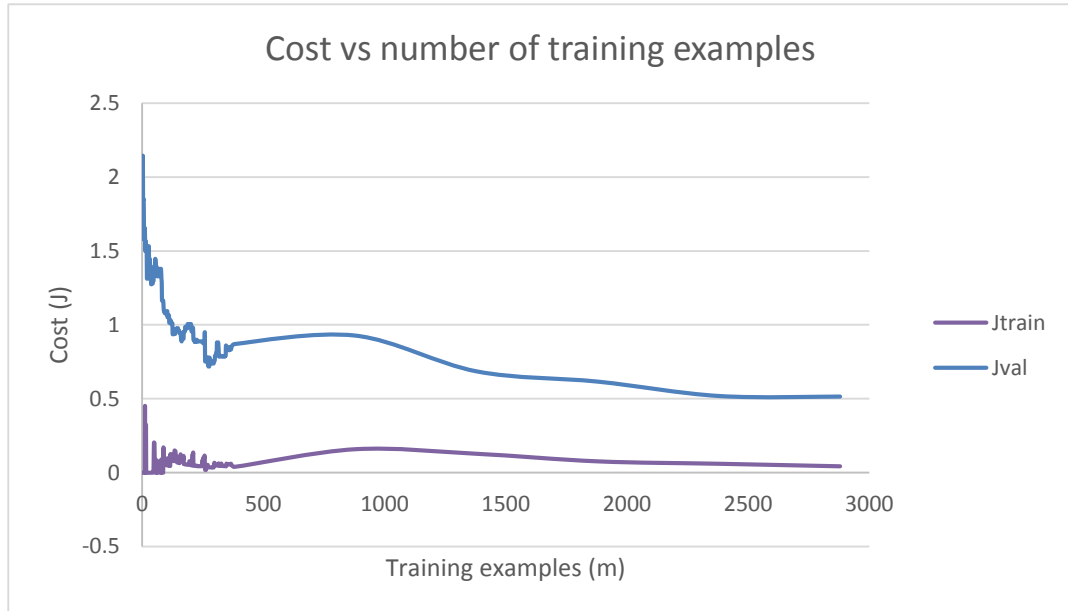


Figure 17: Plot of cost against number of training examples

The number of training examples has been multiplied by 8 by varying the orientation of each data (Figure 18), to extend the list of examples in the hope to regulate high variance. Nonetheless, the graph above shows that 3000 examples is still insufficient to avoid overfitting hence the need for regularisation.



Figure 18: 8 different orientation for one data sample

b. Regularisation parameter

The regularisation term in the modified cost function shown in section 2.3.10 permits to penalise the cost function in order to control overfitting. The regularisation parameter was obtained by plotting the cost of the training and cross validation set as the regularisation parameter varied. Due to the optimisation function that stops iterating when the cost of the training set slows down its descent, and the fact that each initialisation result in a new random position, the graph below does not show a nice curve of both error curves converging as explained in section 2.3.10. However, it confirms that from a certain value of lambda, both error curves are close together. From this graph, the best hyperparameter was 2^8 , or 0.00256, which resulted in the lowest cost for both datasets. Thus, the selection of this regularisation parameter to train the network.

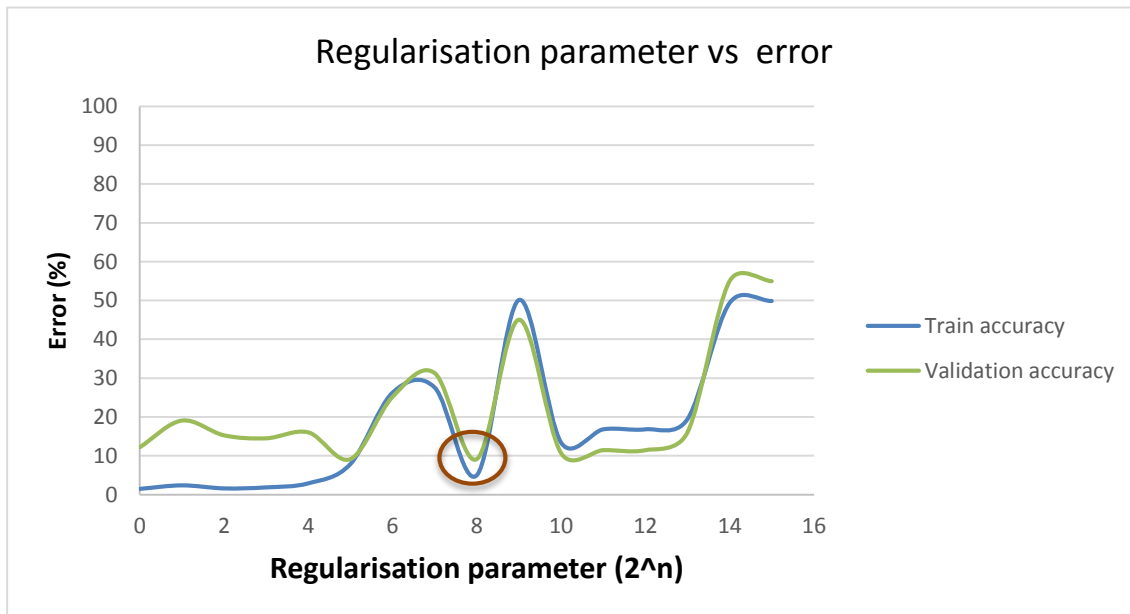


Figure 19: Plot of error against regularisation parameter

4.4. Program design

4.4.1. Input data

As mentioned in the ‘requirements’ section (Section 1.5), the final program was intended to run on an average computer or super computer to check if any meteor has been missed by the current program in all the pictures captured thus far. Therefore, the final design was implemented with a single command line to start the detection process. Although, this script was addressed to the DFN members who have excellent computer skills, the program was made easy and was well documented to assist any user without considerable computer knowledge or a deep understanding of neural networks. In case no path to the directory containing the images to be processed is provided in the command line upon execution, a window pops up for the user to browse the desired folder.

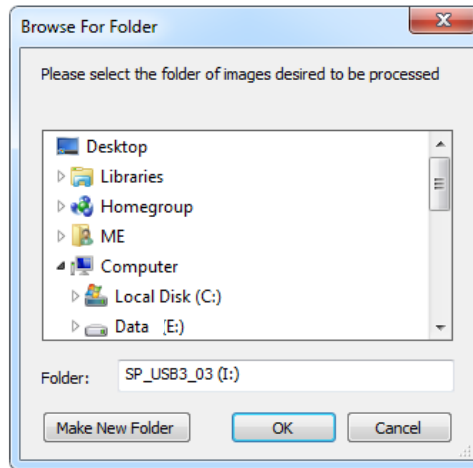


Figure 20: User interface to browse directory

To ease the continuation of the project, the final program conserved all the learning functions which can be enabled and executed from a single command line (displayed below), as explained in the instructions at the beginning of the code, or by changing the default values directly in the script.

```
I:\>python NN_event_detection.py I:\2013-11-01 10 0.0128 1
```

The parameters of the neural network are by default the weights that resulted to the best performances and its corresponding regularisation parameter. However, the weights could be randomly reinitialised by training the network without passing a path to the weights in the command line window. More detailed instructions are provided at the beginning of the code to explain how to use the different options.

4.4.2. Structure

The code was split into 3 classes: Filter, Neural Network, and Performance Evaluation which encompasses all the necessary functions for each of those steps. By doing so the program could be divided into few files comprising each class which enhanced the overall clarity and readability of the code while allowing the modification of each file individually without compromising the others. In addition, the benefit of Object Oriented Programming is the ability to create multiple objects of the same class with different parameters for testing purposes.

The ‘Filter’ class contains all the functions from the conversion of the RGB image to the slicing and preparation of the tiles from a matrix form to a vector form. The object returned is then passed to the ‘Neural Network’ class which loads or creates the network parameters in a matrix form and proceeds with the feed forward propagation. Depending of the values of certain variables passed to the initialisation function of this class, the prediction value is returned to the main function or otherwise the training stage begins. This training stage consists of the back propagation steps from the cost function calculation to the gradient descent and updates of the network parameters. And lastly, the ‘Performance Evaluation’ class assesses the performance of the network once it has been trained by plotting the learning curves, ROC curves and confusion matrices. Thus, the object of the class authorizing the use of all internal functions is only produced if learning has been enabled and the program was given a set of data to learn from.

4.4.3. Output

For a simple detection command, the program runs through all images inside the folder inputted by the user and saves the positive tiles, of the images that have been sliced during the process, into a folder labelled with the name of the individual images. After each picture, the logs, such as the outcome of the detection, are saved in a text file and also transmitted to the user through the Python ‘shell’ window.

In case the program was executed to train the neural network, the new parameters of the network along with the network architecture and the regularisation parameter used, are saved in a file and updated at every iteration. This file can then directly be loaded using a command line to rebuild the artificial neural network and perform the detection process.

Chapter 5: Performance evaluation

5.1. Testing set

Once the best hyperparameters of the network were established, the overall performance of the system was tested on the test set to evaluate how consistent the accuracy of the network was. This set is to be distinguished with the cross validation set as the cross validation set is simply used to determine the set of ‘winner’ hyperparameters that results in the best performance of the model on this dataset. Thus, the same dataset could not be used to also assess the overall performance of the network, hence the purpose of having a test set.

5.2. Comparison with current detection process

5.2.1. ROC

A complete and appreciated way to assess the accuracy of an artificial neural network is to plot the ROC curves. The accuracy is then evaluated from the area under the ROC curve.

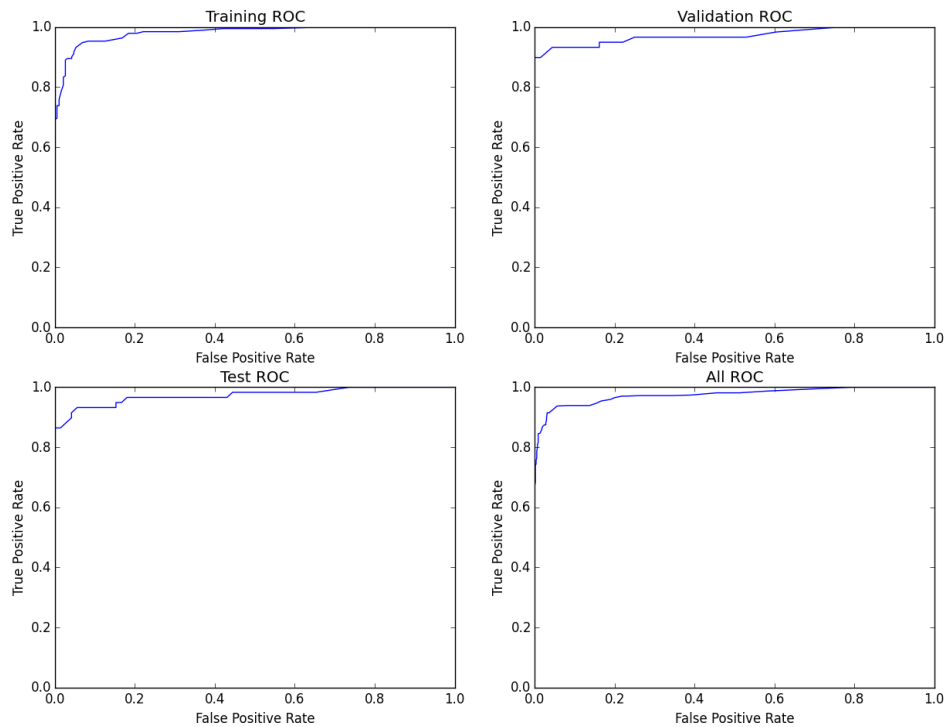


Figure 21: ROC curves

By visual inspection it can be seen that the neural network has achieved high prediction accuracy on the three sets. The threshold of the output unit could also be varied based on the ROC graphs to increase the TPR or avoid false negatives.

5.2.2. Confusion matrix

a. Tile basis

The confusion matrices provides all measures of performance of the network for each dataset. Therefore, the confusion matrices below outline the performance of the network on each dataset on a tile basis, or 640 examples. Because the goal of the program was to detect the more meteor streaks, recall was more important than precision although the optimal objective was to have a high percentage for both measurements. However these tables indicate a high accuracy and precision rate achieved by using the hyperparameters selected in section 4.3.1. The overall accuracy was 92.8% which corresponds to 594 tiles correctly classified out of 640. Whereas the precision was near 97% for the whole datasets compared to 87% for the recall (A complete description of the confusion matrix below is provided in the appendix C). By looking through the datasets it was noticed that the misclassified positive examples were also confusing for a human.

| <i>Training Confusion Matrix</i> | | | | | <i>Validation Confusion Matrix</i> | | | | |
|----------------------------------|---|-----------------|-------|-------|------------------------------------|---|-----------------|-------|--------|
| | | Predicted class | | | | | Predicted class | | |
| | | 1 | 0 | | | | 1 | 0 | |
| Actual class | 1 | 44.9% | 5.2% | 89.5% | Actual class | 1 | 38.6% | 7.9% | 83.1% |
| | 0 | 1.6% | 48.3% | 96.8% | | 0 | 0.0% | 53.5% | 100.0% |
| | | 96.6% | 90.2% | 93.2% | | | 100.0% | 87.2% | 92.1% |
| | | 3.4% | 9.8% | 6.8% | | | 0.0% | 12.8% | 7.9% |

| <i>Test Confusion Matrix</i> | | | | | <i>All Confusion Matrix</i> | | | | |
|------------------------------|---|-----------------|-------|-------|-----------------------------|---|-----------------|-------|-------|
| | | Predicted class | | | | | Predicted class | | |
| | | 1 | 0 | | | | 1 | 0 | |
| Actual class | 1 | 39.7% | 5.3% | 88.1% | Actual class | 1 | 41.1% | 6.2% | 87.0% |
| | 0 | 1.5% | 53.4% | 97.2% | | 0 | 1.0% | 51.8% | 98.0% |
| | | 96.3% | 90.9% | 93.1% | | | 97.5% | 89.4% | 92.8% |
| | | 3.7% | 9.1% | 6.9% | | | 2.5% | 10.6% | 7.2% |

Figure 22: Confusion matrices

Due to the difference in size between the tiles used by the current code implemented by Dr Martin Towner and the neural network, a direct comparison on the number of tiles correctly identified by both approaches would not be meaningful. Therefore the comparison was done on an image basis.

b. Image basis

The comparison between the two approaches was conducted on 50 fireball images with diverse shapes, length, speed, orientation and background brightness. This set also included 3 images with a cloudy sky and 20 photos with faint streaks that were likely to be meteor trails due to their speed and shapes. From this test, shown in detail in appendix A, both approaches performed reasonably well overall as most of the long and crucial meteor streaks with high probability to form a meteorite, were successfully identified. However, for the rest of the examples each program had upsides and downsides.

The explicitly programmed detection code correctly classified 22 images out of 50 compared to 43 from the neural network approach. It was noticed that every faint meteor streaks were ignored by Martin's code. In addition, most of the visually recognizable meteor streaks that have been missed by Martin's code were either small, forming a solid line, partially hidden by trees, or in a cloudy image. However, this program successfully dismissed clouds, airplanes, satellites or light shines produced by the bright moon.

On the other hand, the neural network failed to classify several unclear streaks and misclassified the cloud edges appearing after the first filter was applied as well as on a few noisy tiles.

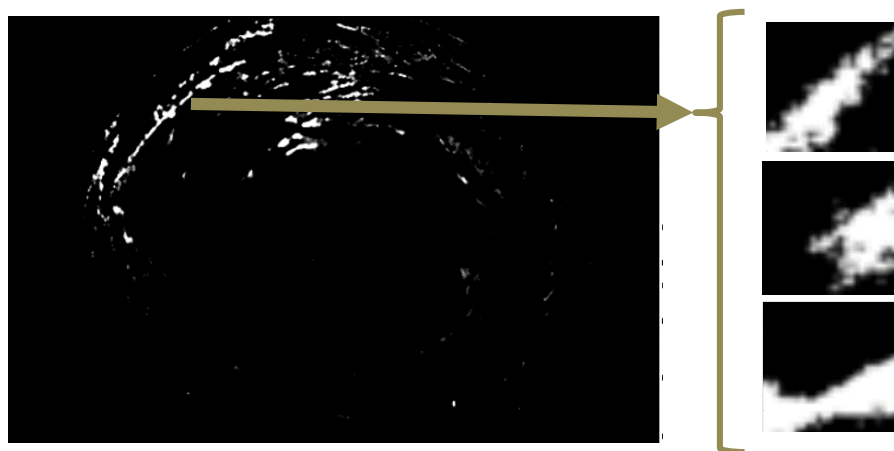


Figure 23: Filtered cloudy image

5.2.3. Speed

When assessing performance, speed should not be neglected. In fact, if the code was to be used on the observatory processor, the neural network should perform as fast or faster than the current code, as the detection process can only be executed during daytime. Both programs were tested on the same set of pictures from one entire night recording. This set was composed of 1076 exposures of a clear night, with 7 exposures that have been manually classified as having meteor streaks. From this test performed on an average computer described in appendix B, Martin's code took 6 hours 20 minutes and 10 seconds to process the entire folder while the neural network program took 6 hours 6 minutes and 10 seconds. This corresponds to 21.2 sec per picture and 20.4 sec per picture respectively. Therefore, the program developed throughout this project is expected to reduce the original process time. Furthermore, as detailed in the table below, the pre-filter is much more time consuming than the forward propagation which means with less filtering process the speed could considerably be improved.

| Actions | Average time (Seconds) |
|---------------------------|------------------------|
| Blurring and thresholding | 0.915468 |
| Dilation | 10.29967 |
| Down sampling | 0.530771 |
| Slicing | 0.005329 |
| Forward propagation | 0.004111 |
| TOTAL | 12.4317 |

Chapter 6: Conclusion and Future Work

The artificial neural network algorithm is a very complex machine learning algorithm that is capable of learning almost anything, to the extent of the available resources, but requires scrupulous insight and tuning to exploit its potential. Through extensive tests and adjustments of each parameter in the artificial neural network, a high accuracy was reached on the 3 datasets. This proves that the algorithm has acquired the faculty to distinguish fireball images from non-fireball images. Overall, the final design performed better than the current code. From the multiple tests explained earlier, the neural network identified several fireball images that have previously been missed by the code currently operating. On a downside, the neural network failed to classify a few meteors and had a high false positive rate, especially on cloudy images. However, this was tolerated due to the fact that the next step, being the triangulation process, will deny those false positives.

Many challenges and difficulties have been encountered throughout the design and implementation stage of the network. Firstly, the trade-off between the quality of the data, the size of the data and the computational cost was a continuous challenge as any change in the pre-filtering process had an impact on the final outcome. Secondly, the time constraint and hardware limit had to be dealt with a good planning and management of the computer resources, in order to maximize the efficiency. The computationally demanding operations were executed overnight to optimise the time. And lastly, the choice of the neural network parameters remained the most time consuming and confusing part. However, these difficulties were tackled by supplementary learning, training, testing and discussions with the project supervisor.

To recap in a few words, the prototype implemented using a machine learning algorithm, to detect fireballs in night sky images, fulfilled the requirements and the goal stated in section 1.5.

6.1. Pre-processing improvements

Although the overall performance of the neural network was satisfactory, the meteor detection process could be improved in different ways to produce an extremely accurate

and robust design. First, the background removal step could be enhanced to reduce the ‘salt’ noise in the resulting image and eliminate the cloud edges to minimise the false positive rate. Moreover, some techniques used in Martin’s code could be combined to the image pre-processing steps, discussed in Chapter 3:, in order to improve the clarity the data and pre-filter the clouds prior to be fed into the neural network.

6.2. Neural network improvements

One of the main difficulties of this project was the time consuming learning stage. To minimize the learning stage and take full advantage of the fast feed forward propagation, different learning algorithms and neural network architecture could be implemented to find the optimal design. Alternatively, the graphic card RAM could be utilized to reduce the time taken by the learning stage.

6.2.1. Deep neural network

A network with multiple hidden layers could be considered to extend the function of the design reached in order to directly process the raw images and hence reduce the processing time.

6.2.2. Learning methods

While backpropagation and gradient descent are the simplest algorithms that offer reasonable performances, many other learning algorithms could be used to reduce the cost function of a bigger network with higher efficiency.

6.3. Hardware improvement

Another important factor that contributed to the choice of the parameters was the hardware. The current computer was found to be out of date for such demanding algorithm. Therefore, with a better configuration, the network could be enlarged by increasing the number of features hence the number of pixels fed into the network. This would likely reduce the false positive detection currently obtained from noisy images or cloudy images.

6.4. Next step

The next step would be to improve the prototype and run it on all the data accumulated since the beginning of the DFN project. The Python script may also be modified in order to be executed on a super computer.

References

- Andres Munoz. (1959). *Machine Learning and Optimization*. Courant Institues of Mathematical Sciences. New York.
- Bailer-Jones, Coryn. (2009). *Applications of Machine Learning in Astronomy*. [online] Available at: http://www.mpia.de/~calj/amla_ss2009/introduction.pdf
- Bland, Phil and Steven Tingay. (2013). *Large scale searches for astronomical transients using a geographically distributed automated camera system*. Desert Fireball Network. Curtin University, Perth.
- C. M. Bishop. (2006). *Pattern Recognition and Machine Learning*. Springer.
- Dalessandro, Brian, Rod Hook, Ziaohan Zhang, Alan Murray, and Claudia Perlich. (2010). *Machine Learning for Display Advertising*. Foster Provost. New York University and Coriolis Ventures. [online] Available at: <http://pages.stern.nyu.edu/~fprovost/Papers/MLOAD.pdf>
- Deshayes, J. (2015). *Extended abstract*. Undergradutae. Curtin University.
- Deshayes, J. (2015). *Progress Report*. Undergradutae. Curtin University.
- DesJardins, Marie E. (1997). *Machine Learning for Military Operations*. Air force material command. Rome, New York.
- Dreiseitl, S. and Ohno-Machado, L. (2002). Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, 35(5-6), pp.352-359.
- Fireballs in the sky*. (2015). Curtin University <http://fireballsinthesky.com.au/about/>
- Fries, M., R. Matson, J. Schaefer, J. Fries, M. Hankey. (2013). *Faster Recovery, Better Sciebcce: Meteorite Fall Events Detected with Weather radars and Seismometers in 2012*. 44th Lunar and planetary science conference CA90740. American Meteor Society, Geneseo, New York.
- Hal Daume. (2014). *A Course in Machine Learning*. TODO. <http://ciml.info/>
- Jordan, M.I., Christopher M. Bishop. (2004). *"Neural Networks"*. In Allen B. Tucker. *Computer Science Handbook, Second Edition (Section VII: Intelligent Systems)*. Boca Raton, FL: Chapman & Hall/CRC Press LLC. ISBN 1-58488-360-X.

- Margaret Rouse. (2011). Machine learning. Programming glossary. <http://whatis.techtarget.com/definition/machine-learning>
- Meteorscan. (2015). *N.L.O Meteor Detection live view*. Meteor Detection and Radio Astronomy. [online] Available at: <http://www.meteorscan.com/meteor-live.html> [Accessed 25 Oct. 2015].
- Michael A. Nielsen. (2015). *Neural Networks and Deep Learning*. Determination Press. [online] Available at: <http://neuralnetworksanddeeplearning.com/chap1.html> [Accessed 25 Oct. 2015].
- Motor Neuron*. (2015). Pixshark. [online] Available at: <http://home.agh.edu.pl/~vlsi/AI/intro/neuron.png>
- Ng, Andrew. (2010). *10: Advice for applying Machine Learning*. Stanford Machine Learning. [online] Available at: http://www.holehouse.org/mlclass/10_Advice_for_applying_machine_learning.html [Accessed 8 April. 2015].
- Ng, Andrew. (2015). *Neural Networks*. Coursera. [online] Available at: <https://www.coursera.org/learn/machine-learning/programming/AiHgN/neural-network-learning> [Accessed 9 April. 2015].
- Oxford English Dictionary, 'Meteorite'. Oxford English Dictionary. Oxford University Press, (2015). <http://www.oxforddictionaries.com/definition/english/meteorite>
- Perceptron*. (2012). Bethopedia. [online] Available at: <http://slidewiki.org/upload/media/images/25/4048.png?filter=Resize-width-418.125> [Accessed 18 Oct. 2015].
- Powers, David M W. (2007). Evaluation: From Precision, Recall, and F-Factor to ROC, Informedness, Markedness & Correlation. School of Informatics and Engineering. Technical Report SIE-07-001. Flinders University of South Australia. Adelaide, Australia. [online] Available at: http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf [Accessed 20 Oct. 2015].
- Richardson, James. (2013). Fireball FAQs. American Meteor Society. [online] Available at: <http://www.amsmeteors.org/fireballs/faqf/#5> [Accessed 25 Oct. 2015].

- Rossum, Guido Van. (1997). *Comparing Python to Other Languages*. [online] Available at: <https://www.python.org/doc/essays/comparisons/> [Accessed 25 Oct. 2015].
- Stephan, Clemencon. (2015). *A Machine –Learning View of Quantitative Finance*. Institut Mines Telecom. LTCI UMR Telecom ParisTech & CNRS No. 5141. [online] Available at: http://www.qminitiative.org/UserFiles/files/S_Cl%C3%A9men%C3%A7on_ML.pdf [Accessed 25 Oct. 2015].
- Szepesvari, Csaba. (2009). *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning series. Morgan & Clayrpool Publishers
- Tadiou. K, Mamadou, (2015). *The future of Human Evolution*. Basic structure of an Artificial Neural Network. [online] Available at: <http://futurehumanevolution.com/wp-content/uploads/Artificial-Intelligence-Neural-Network-Nodes.jpg> [Accessed 25 Oct. 2015].
- Wichman, Eric. (2010). *How to Find Meteorites*. The search for cosmic treasures. [online] Available at: <http://www.meteoritesusa.com/how-to-find-meteorites.htm> [Accessed 10 Oct. 2015].
- Woods, K and Bowyer, K. W. (1997). *Generating ROC Curves for Artificial Neural Networks*. IEEE transactions on medical imaging. Vol.16, no. 3, pp.329-337. [online] Available at: <http://www.lcc.uma.es/~jja/recidiva/048.pdf> [Accessed 22 Sept. 2015].
- Zhao, Siming. (2010). Radar Meteor Trail Detection and Classification. Thesis. The Pennsylvania State University
- Wikipedia, (2015). *Confusion matrix*. [online] Available at: https://en.wikipedia.org/wiki/Confusion_matrix [Accessed 2 Nov. 2015].


















Appendix A: Performance results of NN and MT codes

| | | Neural Network | | Martin's code | | Image correctly identified | |
|-------------------------------|-----------|-----------------|----------------------------|-----------------|----------------------------|----------------------------|----|
| Image name | Folder Nb | Tiles predicted | Tiles correctly identified | Tiles predicted | Tiles correctly identified | NN | MT |
| 10_2014-11-29_143129_DSC_0294 | 1 | 10 | 9 | 3 | 2 | 1 | 1 |
| 22_2014-11-29_143128_DSC_0334 | 2 | 5 | 5 | 2 | 2 | 1 | 1 |
| 16_2014-11-29_143129_DSC_0325 | 3 | 19 | 18 | 7 | 7 | 1 | 1 |
| 24_2014-11-29_143128_DSC_0311 | 4 | 3 | 3 | 0 | 0 | 1 | 0 |
| 22_2014-10-27_175358_DSC_0829 | 5 | 8 | 8 | 3 | 3 | 1 | 1 |
| 31_2014-09-26_164258_DSC_0862 | 6 | 12 | 11 | 0 | 0 | 1 | 0 |
| 15_2014-08-04_102428_DSC_0866 | 7 | 1 | 1 | 0 | 0 | 1 | 0 |
| 15_2014-08-04_154858_DSC_1515 | 8 | 1 | 0 | 0 | 0 | 0 | 0 |
| 15_2014-08-04_173658_DSC_1731 | 9 | 4 | 1 | 0 | 0 | 1 | 0 |
| 15_2014-08-05_150128_DSC_1419 | 10 | 4 | 2 | 0 | 0 | 1 | 0 |
| 15_2014-08-05_161058_DSC_1558 | 11 | 2 | 2 | 0 | 0 | 1 | 0 |
| 15_2014-08-05_202158_DSC_2060 | 12 | 3 | 3 | 1 | 1 | 1 | 1 |
| 15_2014-08-05_203358_DSC_2084 | 13 | 2 | 1 | 2 | 0 | 1 | 0 |
| 15_2014-08-05_203528_DSC_2087 | 14 | 2 | 0 | 2 | 0 | 0 | 0 |
| 15_2014-08-05_204928_DSC_2115 | 15 | 1 | 1 | 4 | 2 | 1 | 1 |
| 15_2014-08-05_210258_DSC_2142 | 16 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15_2014-08-05_211758_DSC_2172 | 17a | 0 | 0 | 0 | 0 | 0 | 0 |
| 15_2014-08-05_211828_DSC_2173 | 17b | 0 | 0 | 0 | 0 | 0 | 0 |
| 15_2014-08-05_212358_DSC_2184 | 18 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15_2014-08-04_155828_DSC_1534 | 19 | 2 | 1 | 1 | 1 | 1 | 1 |
| 15_2014-08-05_133328_DSC_1243 | 20 | 1 | 1 | 2 | 2 | 1 | 1 |
| 15_2014-08-05_170858_DSC_1674 | 21 | 3 | 3 | 1 | 1 | 1 | 1 |
| 24_2014-09-23_123029_DSC_0210 | 22 | 11 | 9 | 4 | 4 | 1 | 1 |
| 42_2014-09-26_164259_DSC_0900 | 23 | 7 | 6 | 1 | 1 | 1 | 1 |
| DSC_5937 | 24 a | 46 | 42 | 8 | 8 | 1 | 1 |
| DSC_5938 | 24 b | 2 | 2 | 1 | 1 | 1 | 1 |
| DSC_5939 | 24 c | 2 | 2 | 1 | 1 | 1 | 1 |
| DSC_5940 | 24 d | 1 | 1 | 0 | 0 | 1 | 0 |
| DSC_5941 | 24 e | 1 | 0 | 1 | 1 | 0 | 1 |
| 08_2013-11-01_112128_DSC_0074 | 25a | 0 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-01_112158_DSC_0075 | 25b | 2 | 1 | 0 | 0 | 1 | 0 |
| 08_2013-11-01_130458_DSC_0281 | 26 | 1 | 1 | 0 | 0 | 1 | 0 |
| 08_2013-11-01_154958_DSC_0611 | 27 | 1 | 1 | 0 | 0 | 1 | 0 |
| 08_2013-11-01_174228_DSC_0836 | 28 | 1 | 1 | 0 | 0 | 1 | 0 |
| 08_2013-11-01_185028_DSC_0972 | 29 a | 4 | 4 | 0 | 0 | 1 | 0 |
| 08_2013-11-01_185058_DSC_0973 | 29 b | 1 | 1 | 0 | 0 | 1 | 0 |
| 08_2013-11-04_113259_DSC_0091 | 30 | 2 | 2 | 0 | 0 | 1 | 0 |

| | | | | | | | |
|---|------|-----|----|---|---|-----------|-----------|
| 08_2013-11-01_185258_DSC_0977 | 31 | 3 | 2 | 3 | 1 | 1 | 1 |
| 08_2013-11-03_113928_DSC_0095 | 32 a | 4 | 3 | 2 | 1 | 1 | 1 |
| 08_2013-11-03_113958_DSC_0096 | 32 b | 0 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-04_193559_DSC_0520 | 33 a | 188 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-04_193629_DSC_0521 | 33 b | 188 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-03_114728_DSC_0111 | 34 a | 4 | 4 | 2 | 2 | 1 | 1 |
| 08_2013-11-03_114758_DSC_0112 | 34 b | 3 | 3 | 2 | 1 | 1 | 1 |
| 08_2013-11-03_120158_DSC_0140 | 35 a | 0 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-03_120228_DSC_0141 | 35 b | 0 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-03_120258_DSC_0142 | 35 c | 0 | 0 | 1 | 0 | 0 | 0 |
| 08_2013-11-03_145458_DSC_0486 | 36 a | 0 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-03_145528_DSC_0487 | 36 b | 0 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-03_145558_DSC_0488 | 36 c | 0 | 0 | 0 | 0 | 0 | 0 |
| 08_2013-11-03_145628_DSC_0489 | 36 d | 0 | 0 | 1 | 0 | 0 | 0 |
| 08_2013-11-03_173058_DSC_0798 | 37 | 0 | 0 | 0 | 0 | 0 | 0 |
| DSC_3283 | 38 | 3 | 3 | 0 | 0 | 1 | 0 |
| DSC_3048 | 39 | 2 | 2 | 0 | 0 | 1 | 0 |
| DSC_3770 | 40 | 8 | 6 | 0 | 0 | 1 | 0 |
| DSC_5207 | 41 | 3 | 3 | 0 | 0 | 1 | 0 |
| DSC_5651 | 42 | 19 | 19 | 0 | 0 | 1 | 0 |
| DSC_0190 | 43 | 164 | 0 | 0 | 0 | 0 | 0 |
| 06_2014-05-30_103047_DSC_0107 | 44 | 25 | 24 | 9 | 5 | 1 | 1 |
| 15_2015-01-21_190558_DSC_1646 | 46 | 10 | 10 | 4 | 0 | 1 | 0 |
| 18_2015-01-21_190558_DSC_1151 | 47 | 3 | 3 | 2 | 1 | 1 | 1 |
| 20_2015-01-21_190558_DSC_0573 | 48 | 5 | 5 | 0 | 0 | 1 | 0 |
| 34_2015-01-21_190559_DSC_0886 | 49 | 7 | 7 | 7 | 2 | 1 | 1 |
| Total number of images correctly identified: | | | | | | 43 | 22 |

The orange cells denote the images containing faint streaks while the blue cells represent the images with clouds.

Appendix B: System Summary

| System Summary | |
|---|--|
| Item | Description |
|  Mainboard | ASUSTeK COMPUTER INC. P8Z77-V LX |
|  Chipset | Intel Z77 |
|  Processor | Intel Core i5 2400 @ 3100 MHz (max : 6489) |
|  Physical Memory | 16384 MB DDR3-SDRAM |
|  Video Card | NVIDIA GeForce GTX 760 |
|  Hard Disk | Hitachi HDS721010CLA332 (1000GB) |
|  Hard Disk | KINGSTON SH103S3120G (120GB) |
|  Hard Disk | MAXTOR STM3500320AS (500GB) |
|  Hard Disk | PHD 3.0 Silicon-Power (1000GB) |
|  CD-Rom Drive | DTSOFT Virtual CdRom Device |
|  Monitor Type | LG Electronics E2340 - 23 inches |
|  Monitor Type | Samsung SMS27A350H - 27 inches |
|  Network Card | Realtek Semiconductor RTL8168/8111 PCIe Gigabit Ethernet Adapter |
|  Network Card | Atheros Communications AR9287 Wireless Network Adapter |
|  Operating System | Windows 7 Ultimate Professional Media Center 6.01.7600 (64-bit) |
|  DirectX | Version 11.00 |
|  Windows Performance Index | 7.4 on 7.9 |

Appendix C: Description of the confusion matrix

| True condition | | | |
|------------------------------|--------------------|--------------------|--|
| Predicted condition | Condition positive | Condition negative | |
| | True positive | False positive | Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$ |
| | False negative | True negative | False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$ |
| | | | False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$ |
| Predicted condition positive | | | Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$ |
| | | | Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$ |
| | | | |
| | | | |
| Predicted condition negative | | | Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$ |
| | | | |

Modified table extracted from Wikipedia (Wikipedia, 2015)