Roman Kolacz
rk476@cam.ac.uk

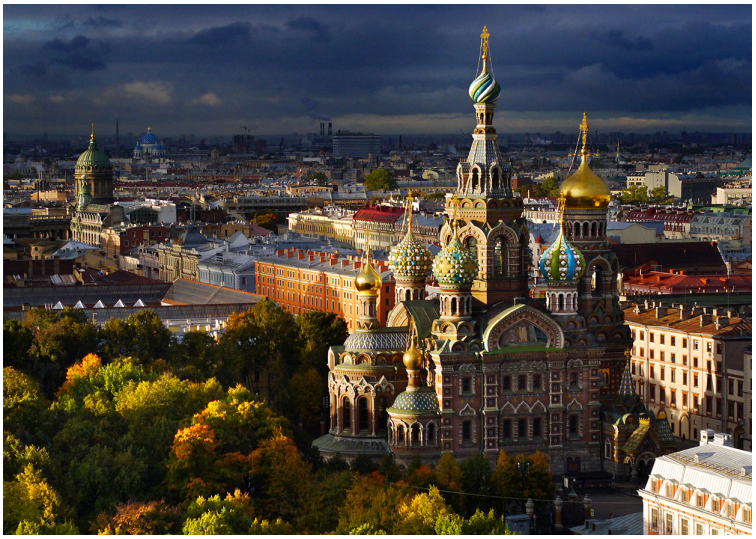
University of Cambridge

Computer Science Tripos

Part II Project

Time-Lapse Based Weather Classification

2014-2015



The landscape on the cover is of Saint Petersburg, Russia. It was taken with a drone, and shows an unexpected spell of sunshine illuminating The Church of the Saviour on Spilled Blood during a brewing storm.

Saint Petersburg is the birth place of Wladimir Köppen (1846-1940), who developed the *Köppen Climate Classification System*. It is the most widely used weather classification system to this day.

Proforma

Name:	Roman Kolacz
College:	Downing College
Project Title:	Time-Lapse Based Weather Classification
Examination:	Computer Science Tripos - Part II
Word Count:	≈ 8000
Project Originator:	Alan Blackwell
Project Supervisor:	Advait Sarkar

Initial Project Aims

The initial aim of the project was to associate time-lapse video with external data sources by using a time-lapse video of an outside location to tell the weather. This would then be evaluated against real weather information and demonstrated graphically.

Work Completed

Over a months worth of data (10,527 data points at 5 minute intervals), has been gathered. A classifier has been built which uses extracted image metrics and actual weather data to train a machine learning classifier which can then infer the weather. The accuracy of several machine learning algorithms has been tested and the best for each data set has been used to maximise overall accuracy. The plan was changed from using manually hard-coded rules for weather classification to using machine learning to do so instead.

Special Difficulties

A total of three Raspberry Pi's were used over the course of the data gathering stage, due to malfunctions and unfortunate mishaps. These significantly delayed the data gathering stages of the project, resulting in less data being gathered than planned.

Declaration of Originality

I, Roman Kolacz of Downing College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed:

Date:

Acknowledgements

Alan Blackwell, for help with the initial project idea and with directing the idea towards one which was suitable to use as a Part II project.

Advait Sarkar, for all his help and guidance as a supervisor and without whom this dissertation would not have been possible.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Challenges 2

1.3 Related Work 2

1.4 Related Courses 3

2 Preparation **4**

2.1 Requirements Analysis 4

2.2 Data Gathering 5

2.2.1 Hardware 5

2.3 Time Lapse 6

2.4 Machine Learning 7

2.4.1 Random Forest 7

2.4.2 Multilayer Perceptron 9

2.5 Choice of Tools 11

2.5.1 Programming Language 11

2.5.2 Libraries 11

2.5.3 Backups 13

3 Implementation **14**

3.1 Gathering Image Data 15

3.1.1 Feature Extraction 15

3.2 Data Storing 21

3.3 Sanity Checking 22

4 Evaluation **23**

4.1 Cross Validation 23

4.2 Description of Data 24

4.2.1 Weather Type 24

4.2.2 Cloud Cover 25

4.2.3 Precipitation 26

4.2.4 Temperature 27

4.2.5 Sunrise and Sunset 27

4.3 Analysis of Classifiers 27

4.3.1 Random Forest 28

4.3.2 Multilayer Perceptron 32

4.3.3 J48 32

4.4 Performance 33

4.5 Overall Results 33

5 Conclusions **35**

5.1 Summary 35

5.2 Future Work 35

5.2.1 Extensions 35

5.2.2 Changes 36

5.2.3 Closing Words 38

Bibliography **38**

6 Appendices **41**

6.1 Code Extracts 41

6.1.1 Data Harvester 41

6.1.2 Classifier 43

List of Figures

2.1 The Raspberry Pi and Camera Module Setup 6
2.2 A basic example decision tree for classifying the weather from an image . . 7
2.3 Random Forest with three trees 9
2.4 Three layer multilayer perceptron 10
2.5 Five layer multilayer perceptron 11

3.1 Processing pipeline 14
3.2 Comparison of greyscale obtained via average and luminance methods . . . 15
3.3 An image taken during rain 17
3.4 An image taken during sunlight 17
3.5 An image taken during a cloudy night 18
3.6 An image taken during snow 18
3.7 Brightness histograms of all images 19
3.8 RGB histograms of all images 20

4.1 Cross Validation 23
4.2 Distribution of weather types in the data set 24
4.3 Distribution of cloud cover percentages in the data set 25
4.4 Precipitation data bar chart 26
4.5 Temperature data bar chart 27
4.6 Weather type classification peformance per number of trees 28
4.7 Sunrise and Sunset regression peformance per number of trees 29
4.8 Cloud cover regression peformance per number of trees 30
4.9 Precipitation regression peformance per number of trees 30

4.10 Temperature regression peformance per number of trees 31

5.1 One of the images split into a 4x4 grid 36

5.2 Creation process of histogram of oriented gradients with the sun and rain images, respectively 37

5.3 Visual representation of a histogram of oriented gradients 38

List of Tables

2.1 Requirements breakdown 4

3.1 The average hues of the images, shown as RGB values on a scale of 0 to 255 21
3.2 Example schema for csv file 22

4.1 Complete table of results 33
4.2 Confusion matrix of inferred vs actual results. 34

Chapter 1

Introduction

This dissertation aims to describe the process of building, testing and evaluating a weather classifier from time-lapse video, including the process of gathering the data necessary to train such a classifier.

The dissertation will be split into three main chapters: Preparation, Implementation and Evaluation.

The preparation chapter will largely focus on data gathering and research undertaken to allow the project to proceed smoothly. This involved using hardware to take images and gather local weather data at regular intervals, while the research included reading around approaches to similar problems in the past, as well as learning about the algorithms and libraries which would be employed in the project.

The implementation chapter will go into detail on how a classifier was built to use the data gathered to classify the weather. It will describe the process of extracting features from the images gathered, and how the data was stored to then be used by the machine learning library. It will then detail how the data was used to train, run and test the classifier.

The evaluation chapter will then focus on the results obtained from the classifier built in the previous section. This will include how it was tested, the overall accuracy of the classifier, and comparisons between different machine learning algorithms for different measures of weather conditions.

1.1 Motivation

Time lapse videos are a relatively unexplored aspect of data visualisation, and have a rather unique property amongst video of being able to hold information about incredibly large periods of time in a relatively small amount of time and space, provided they are of an event which occurs slowly and gradually, or is otherwise relatively uneventful.

An example of such an event is weather changes, including day-night transitions. These can very reliably be captured using time-lapse video as there are only minor changes between frames and the events are relatively long, so a constant video stream would be unnecessary and would result in either a long uninteresting video, or a very large proportion of wasted frames should the video be sped up to a watchable speed.

Weather classification was one particular example of associating time-lapse videos with external data sources, though a very similar approach could be used for other data, such as time-lapse videos mounted to tourist locations to observe places of interest and 'hotspots' over different periods of time.

Weather has a vast and often understated impact on modern life. Weather variability cost the U.S economy an estimated $485billion in 2008[1], though it also undeniably affects even the smallest of decisions in peoples day to day lives (such as whether sunglasses or an umbrella should be taken for the journey to work). As such, weather classification and prediction has become an all but essential area of research, with the capacity to greatly impact the world.

1.2 Challenges

Numerous challenges had to be considered before work on the project began.

One such challenge was deciding which features from within an image should be extracted to maximise classifier accuracy. This involved finding the features with large differences between classes to ease classification and with minimum inter-class variance to decrease false classification rates.

Another challenge to be circumvented was how similar looking weather conditions would be differentiated, given the features extracted. An example of such conditions would be heavy snow and direct sunlight (in that the image would be largely filled with white), or fog and rain.

Thirdly, as an experiment, temperature was added despite there being few visual cues in an image which would accurately depict the temperature. This was especially the case in England where sub 10°C temperatures in direct sunlight (a visual cue generally associated with warmth) and rain or dark clouds (generally associated with coldness) on a warm day were not uncommon occurrences.

1.3 Related Work

A number of previously published papers were useful for learning about various aspects of the project.

An approach which forms the basis of the weather-labelling per frame is described in the paper "Classification of Weather Situations on Single Colour Images"[2]. It describes classifying weather using image metrics to be used to assist driving-assistance systems which are heavily dependent on clear conditions. It achieves very accurate results using only two labels (clear and rain), and reasonable accuracy when adding a third, "light rain" label. This project will build on this by adding further labels to the weather classifying, as well as incorporating a cloud-cover measure.

The paper also mentions inaccuracies relating to weather situations containing heavy fog or rain, as visibility is greatly reduced. These would have to be taken into consideration.

Another paper referenced is "Support Vector Machines for Histogram-Based Image Classification"[3], which describes that classification using support vector machines (and thus classification generally) is possible using only high dimensional histograms, whilst the paper "Image Classification using Random Forests and Ferns"[4] describes classifying images using Random Forests. This project will aim to incorporate aspects from both papers, in that it will be using Random Forests (alongside other classifiers) with mainly high dimensional histograms as feature set.

Although unrelated to weather classifying, "Time-Lapse Photography Applied to Educational Videos"[5] explores applying time-lapse videos to educational videos to aid with analysing long term trends and discarding short term 'noise'. This is comparable to what this project is aiming to achieve through the use of time-lapse video.

1.4 Related Courses

The project builds upon the following courses from the Computer Science Tripos,

1. Artificial Intelligence
2. Information Theory and Coding
3. Algorithms
4. Software Engineering
5. Computer Vision
6. Programming in Java and Further Java

Additional reading and research on machine learning algorithms, feature extraction and data management were also carried out to make the project possible.

Chapter 2

Preparation

This chapter will go into detail on the hardware used to gather data, the theoretical background behind the Random Forest and Multilayer Perceptron machine learning algorithms, and an explanation behind the choices made with regards to which tools to use throughout the project.

2.1 Requirements Analysis

Component	**Risk**	**Priority**	**Difficulty**
Data Gathering			
Set up image gathering hardware	H	H	M
Understand Forecast.io API	L	M	L
Set up file-backup	L	L	L
Implement data gathering	M	H	M
Classification			
Implement feature extraction	M	H	H
Understand WEKA API	L	M	M
Implement classification	M	H	H
Try different algorithms	L	L	M
Evaluation			
Describe gathered data	L	H	L
Implement cross validation	M	M	M

Table 2.1: Requirements breakdown
Values are High (H), Medium (M) and Low (L).

The project was split into key components, which were then assigned values for how much risk, priority and difficulty they carried with them (as shown in figure 2.1. High risk components had a large chance of failing early, or had failures which were difficult to recover from should they occur. High priority components were integral to the success of the project, and often had other components depend on them. Difficult components were likely to require a large amount of time and resource investments to get working to satisfactory standards. Development of components with high values in any of the three attributes was prioritised.

The Spiral software development model was used, as it allowed for frequent testing and sudden changes to be made throughout the course of the project. This was particularly useful as the exact requirements and specifications of the project were often only realised once development had begun.

2.2 Data Gathering

Due to the requirement of a large set of time constrained data, data gathering was a high priority to get under way and working as soon as possible. Every 5 minutes which passed was an image not taken and thus a data point lost (as although past weather information was available, an image from the past was not practical to obtain or emulate).

The data gathering involved using a small hardware setup to gather images and real weather information and then storing it in an easy to use, easy to read format.

2.2.1 Hardware

The Raspberry Pi[6] is a small, credit-card sized computer created with the intention of aiding the teaching of Computer Science to young children. The device is small, cheap and can run Linux, which makes it excellent for small hardware applications such as for what was required by this project.

A 'Model B' Raspberry Pi was used for the project. It has a 700MHz ARM CPU, a dedicated Broadcom VideoCore GPU and 512 MB of shared RAM[7].

A Camera Module[8] is also available for the Raspberry Pi, which plugs into a dedicated port on the Pi using a flat cable, and can shoot 1080p video or take up to 5 megapixel photographs. Although the Camera Module software bundled includes a time-lapse command to automatically generate time-lapse video, a Java program was instead made to take the images as further actions such as weather querying and uploading also needed to be carried out on each interval.

A Raspberry Pi with the Camera Module was used to take pictures at a 5 minute interval out of a window overlooking the Downing College court.

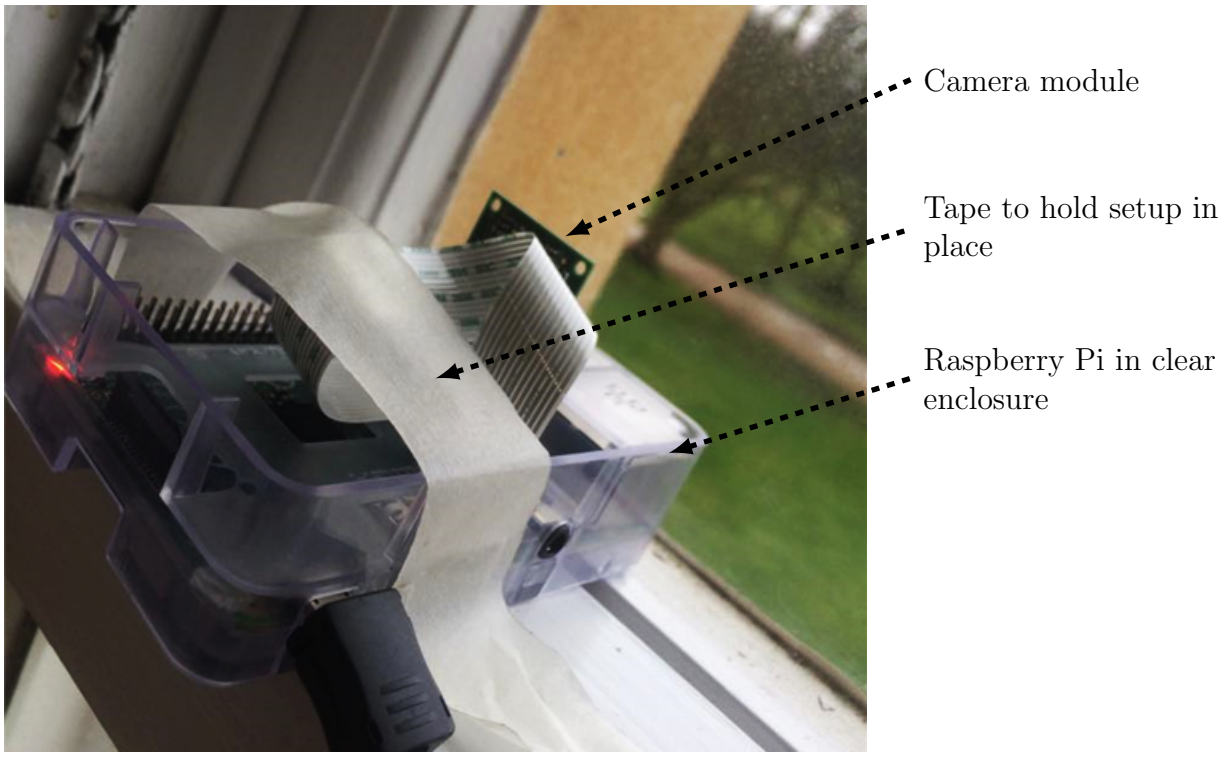


Figure 2.1: The Raspberry Pi and Camera Module Setup

At each interval, data was added to a file which contained the necessary image metrics to train the classifier, as well as actual weather data and basic file information such as the name, date and time. This file, along with each image, was then uploaded to an online file sharing platform so it could be accessed from other devices which required the gathered data. This is explored in more detail in section 3.2.

2.3 Time Lapse

Time-lapse video worked especially well for this project, as using a camera placed in a single location for the duration of the data gathering ensured that variables which would adversely impact the features extracted were kept constant. Using images taken from different places would have a different visibility of the sky, and would've contained different lighting features and shadows as well as different objects which would likely be differently coloured. These would be unideal for the purpose of training a classifier. Given that the images were taken at regular intervals from a single location, time-lapse video was a logical next step.

Time-lapse involves taking photographs at regular (variable length, generally no more than

a few minutes though can be as often as a few seconds) intervals which when played at a standard frame rate (30-60 FPS) look like time is sped up significantly. In this project, the interval was set at 5 minutes, which gives about two and a half hours of real-world time for every second of video when played at a standard 30 FPS.

2.4 Machine Learning

Research was devoted towards learning how different machine learning algorithms can be used and which would excel in different parts of the weather classification system.

2.4.1 Random Forest

A decision tree in machine learning is a tree-like structure which can be utilised to reach a conclusion based on a given set of parameters. At each branch, a path is chosen based on a condition, and the tree is traversed in this way until a leaf is reached with an outcome.

Decision trees are simple to generate and use. Unfortunately, they have a high variance as even small changes in inputs can cause the traversal of different branch paths to drastically different results. They also have a tendency to over fit data (so outliers are significantly accounted for and general patterns are not formed) when trained too much, and are thus not usable as a reliable machine learning algorithm without modifications. They are however, used as building blocks for many other machine learning algorithms, one of which being the Random Forest.

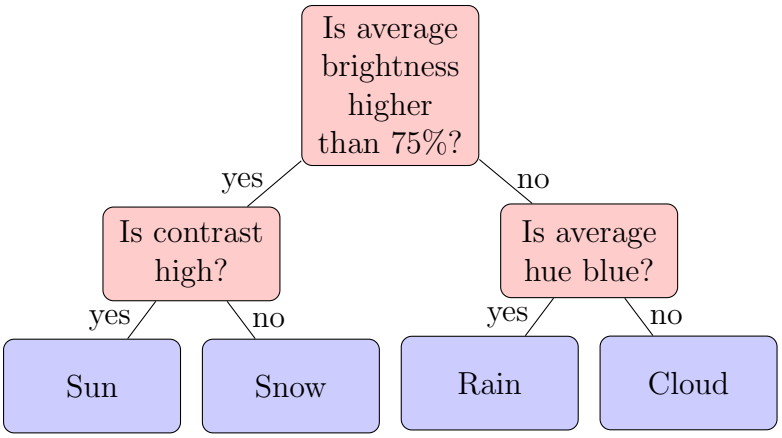


Figure 2.2: A basic example decision tree for classifying the weather from an image

Decision tree training involves creating branches based on the best splits of data at each stage, such that the uncertainty at each node is minimised. This is obtained by maximising

Information Gain, a variable which calculates the difference in entropy (a value used to measure the impurity of a set of random variables) of a set of data given a selected attribute to split the data against.

In a set of training data defined by

$$T = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$$

where $\mathbf{x} = \{x_1, \cdots, x_n\}$ is the set of attributes and y is the desired result of a value with the given attributes.

The entropy of a data set is defined by

$$H(T) = -\sum_{i=1}^{|T|} P(y_i) \log_2 P(y_i)$$

Where $P(t)$ is the probability mass function of T.

Information Gain over the a^{th} attribute is then defined by

$$IG(T, a) = H(T) - \sum_{v \in vals(a)} \frac{|T_{a,v}|}{|T|} H(T_{a,v})$$

where $T_{a,v}$ is the subset of T whose members contain value v in the a^{th} attribute, defined by

$$T_{a,v} = \{\mathbf{x} \in T | x_a = v\}$$

The trees are then created top-down by selecting the attribute with maximum information gain at each branch.

A random forest is a machine learning algorithm which builds upon decision trees and corrects for their variance and tendency to over fit. It works by using multiple decision trees each trained on randomly selected data points from the training set which individually classify the data. Although individual trees may overfit or be sensitive to outliers, the trees overall tend to have little to no correlation and thus tend not to overfit the data.

Either the mode (for classification) or the mean (for regression) of the results of the individual trees is used as the final outcome.

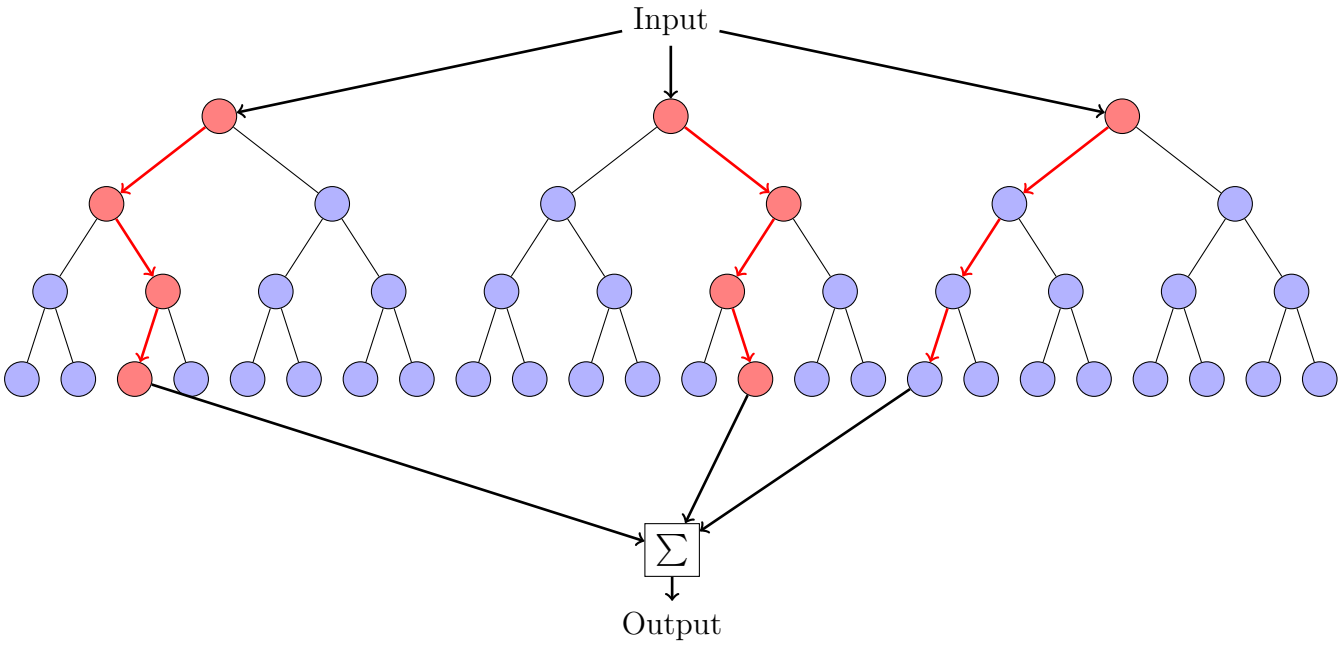


Figure 2.3: Random Forest with three trees

Random forests run relatively quickly and thus were used from an early stage in the project to see if classification was working and to give a rough estimate of the overall accuracy of classification (primarily to see if it was better than randomly guessing the weather).

2.4.2 Multilayer Perceptron

A single perceptron simply returns a value between 0 and 1 based on a given input and a given activation function. Two of the more commonly used activation functions are the threshold function, defined by

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

And the logistic function, defined by

$$f(x) = \frac{1}{1 + e^{-x}}$$

Where x is the output of the perceptron function defined by

$$Y = \mathbf{x} \cdot \mathbf{w} + b$$

In which Y is the output of the perceptron, $\mathbf{x}$ is an input vector, $\mathbf{w}$ is the weight vector and b is a constant term independent of the inputs. This can be simplified to simply

$$Y = \mathbf{x} \cdot \mathbf{w}$$

By treating the constant term, b as another element in the input and weight vectors.

It aims to divide the data into two sets- the number returned determines which side of the division the given value should lie, and does so by recursively comparing its output with the desired output and adjusting the weights and bias' accordingly.

A multilayer perceptron uses at least three layers of perceptrons: input, output and at least one layer in between, known as the hidden layer. Each individual perceptron in each layer connects with a certain weight to every perceptron in the following layer. The final layer (the output layer) contains only one perceptron so that only one output value is obtained.

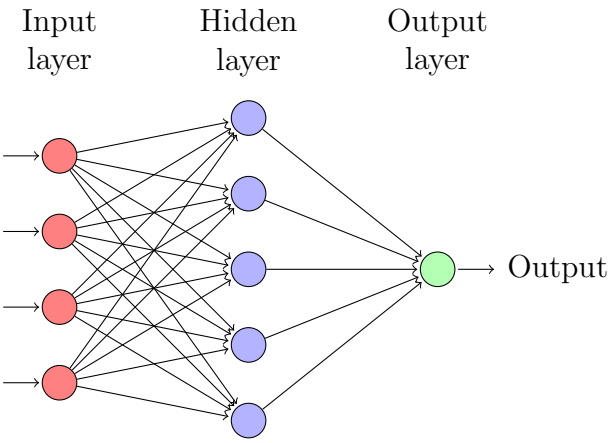


Figure 2.4: Three layer multilayer perceptron

The output of each perceptron in a layer other than the input layer in this case is

$$Y_{n,j} = f(\sum_{i=0}(Y_{n-1,i} \cdot w_{n,i,j}))$$

Where $Y_{n,j}$ is the output of a given perceptron in layer n and position j, $w_{n,i,j}$ is the weight of the output from layer $n-1$ at i to layer n at j and function f is the activation function as detailed above.

A multilayer perceptron is trained in a similar way to a single perceptron. The weights of the connections are changed to adjust the acquired value to the expected value in the training set, and this is done backwards through the layers and then recursively over the entire network.

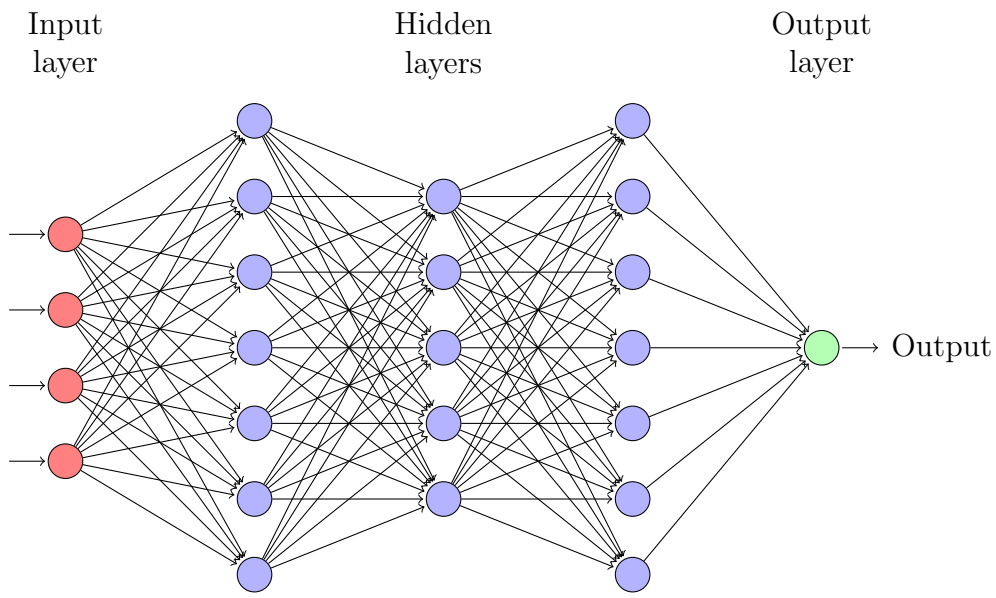


Figure 2.5: Five layer multilayer perceptron

Multilayer perceptrons tend to have multiple layers, and each layer can have any amount of perceptrons in it. Figure 2.5 shows a 5 layer MLP with 7, 5 and 7 nodes in its respective hidden layers.

2.5 Choice of Tools

2.5.1 Programming Language

Java was used for the entirety of the project. It was chosen to ease development across different operating systems and to be able to use the WEKA library.

Performance wasn't an issue as the classifying was done in a single run whenever results were required, rather than dynamically with the data gathering (few machine learning algorithms support dynamic training in this way even if it were to be undertaken). Performance of individual classifiers is discussed in greater detail in section 4.4.

2.5.2 Libraries

WEKA

The WEKA[9] library was used for machine learning. It provides the functionality to train its classifiers on CSV files with whatever options need to be set.

The API[10] contains features which can be split into three categories:

1. Preprocessing
2. Training
3. Classifying

The preprocessing classes include classes for managing data contained within CSV and ARFF (Attribute-Relation File Format) files, or databases; and preparing them to be used with the classifiers. Amongst them are included options for setting input attributes and the desired result column, as well as options for managing attributes by converting between continuous and discrete types or simply removing them altogether.

The training classes include the multitude of classifiers and options WEKA comes with, including Random Forests, Multilayer Perceptrons, J48, along with many others which weren't used for this project.

Once a classifier has been trained, WEKA provides functionality for providing an output based on a set of attributes. The CSV files were then modified with separate code to create columns with the results so that each row could be individually used for comparisons.

Forecast.io

The Forecast.io[11] API was used for gathering the ground truth weather data. It provides an accurate forecast based on longitude and latitude for all the data which was required, and more. Forecast.io allows 1,000 free queries daily, which is above the amount required for the 5-minutely querying of the project.

Forecast.io uses a wide range of data sources including the USA NOAA System[12], the UK Met Office[13] and various worldwide meteorological reports. The data gathered is statistically aggregated to provide the most accurate data available.

The Forecast.io API takes a longitude and latitude and returns a JSON file with detailed weather information subdivided into minutely, hourly and daily; each of which contain data for several units of time (60 minutes, 48 hours and 7 days, respectively) in either past or future directions. A wrapper[14] library was used to deserialize the JSON format into classes with the relevant data. For this project, the current weather data rather than forecasted or historical weather data was used, so the information was only gathered when the image was taken.

Unfortunately, visibility as a column had to be omitted because Forecast.io didn't provide real visibility data for the area.

2.5.3 Backups

Dropbox[15] and GitHub[16] were both used to ensure the project was safe should my hard drive become inaccessible. This was done by creating the git repository within a dropbox folder. Versioning and branching wasn't used due to the linear nature of the project and the fact that the work was done by a single person on a single computer at a time.

Chapter 3

Implementation

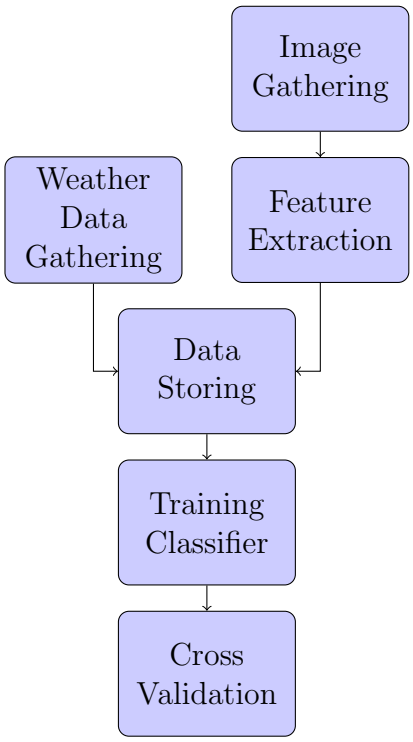


Figure 3.1: Processing pipeline

3.1 Gathering Image Data

3.1.1 Feature Extraction

Once the images had been obtained with the hardware detailed above, the features had to be extracted in order to train the classifier. Initially OpenCV[17] was going to be used, though soon into development there was a realisation that there was little need for it and the feature extraction phase was implemented manually in Java.

The features to be extracted had to be chosen such that there was as much variation as possible between classes, and as little variation as possible within classes, to maximise classification accuracy. In addition to this, the features had to be easy to extract and store within a useable and easily readable format.

The features extracted included:

- Average Hue
- Brightness Histogram
- Each of the red, green and blue (RGB) histograms

These were obtained by sampling each pixel in the images using the Java BufferedImage[18] class to return the RGB values of the pixels of an image in sequence as an array of bytes. The brightness value of a set of RGB values was determined by the luminance function.

$$Y = 0.2126R + 0.7152G + 0.0722B$$

where Y is the luminance and R, G and B are Red, Green and Blue respectively.

This formula reflects how much each light contributes to the perceived brightness by a human eye[19]. Green contributes the most, whereas blue contributes the least. Figure 3.2 shows how this method produces a better result than using the average of the RGB values[20].

Original Luminance Average

Figure 3.2: Comparison of greyscale obtained via average and luminance methods

The histograms are a size 256 array containing the number of pixels with the given RGBY value.

The average hue was obtained by simply averaging the values of each of the pixels in each image.

Examples of Images and Extracted Features

Figures 3.3 to 3.6 show four examples of images taken during the data gathering stage, each with a different weather class. Even from an initial glance, several observations can be made.

1. The picture of rain has a bluer tint due to the clouds and the refraction of the sky in the rain drops
2. The picture with sunlight is highly contrasted due to the bright sky and deep shadows
3. The picture of the night is darker than the other images, and should thus have a significantly lower brightness. The specific weather conditions are, however, near impossible to tell.
4. The picture with the snow is bright but significantly less saturated than the other images due to the greying effect of the snow on the image.

Figure 3.3: An image taken during rain

Figure 3.4: An image taken during sunlight

Figure 3.5: An image taken during a cloudy night

Figure 3.6: An image taken during snow

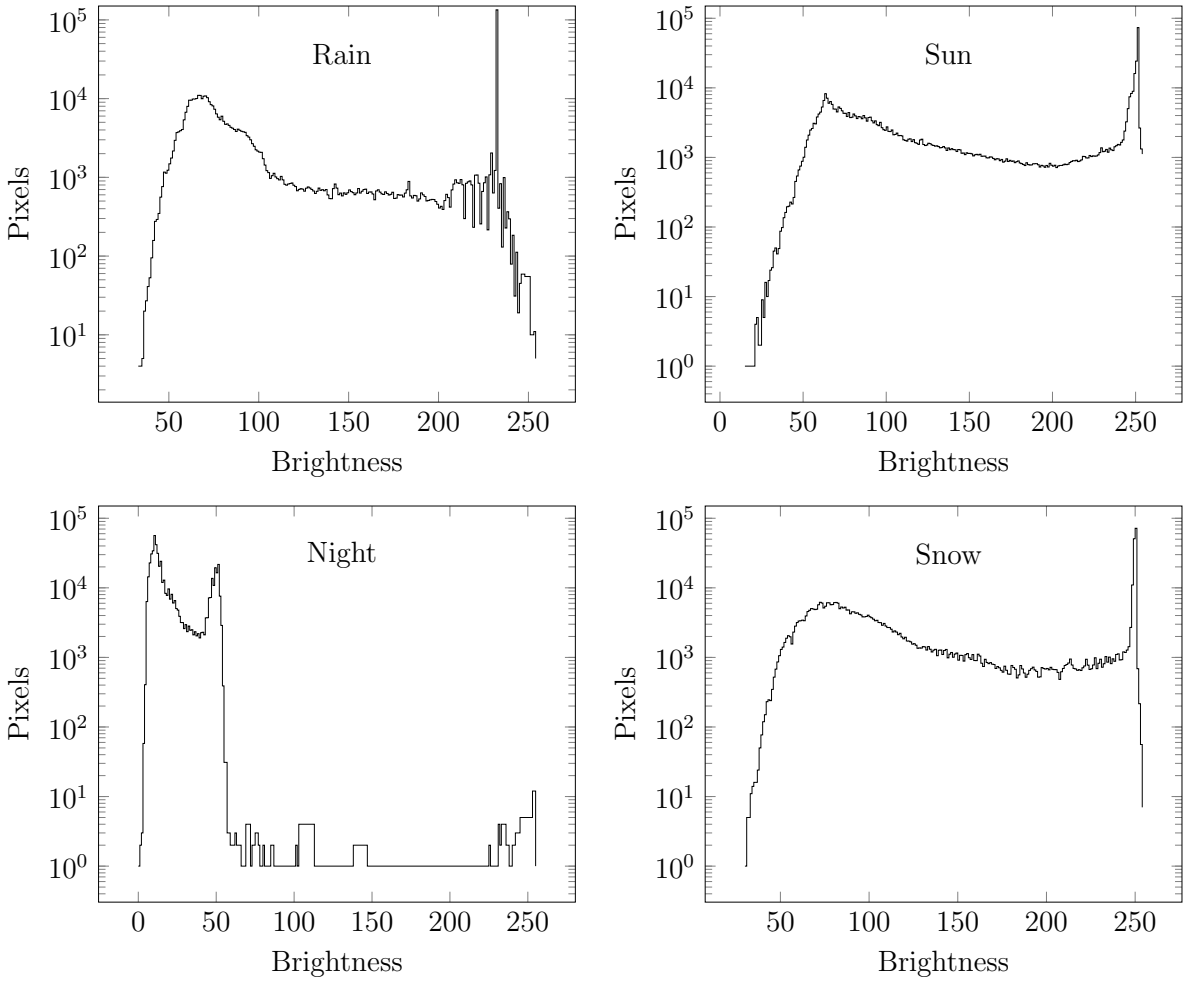


Figure 3.7: Brightness histograms of all images

Figure 3.7 shows that the image of the cloudy night has far more pixels with lower brightness than the daylight images, making brightness a very effective feature to use with regard to determining night from day. Although more subtle, the image with the rain had a peak brightness significantly lower than the images of snow and sun, indicating that the brightness was overall lower than the other two images, though still significantly higher than with the night. Although the brightness could be used to split rain against sun and snow, there are likely to be more reliable features for doing so.

The histogram of brightness is likely to have high variance as images during midday are generally going to be brighter than images in the morning or evening (which would still be classified as day time). As such, general shapes of the graphs rather than exact values would need to be considered.

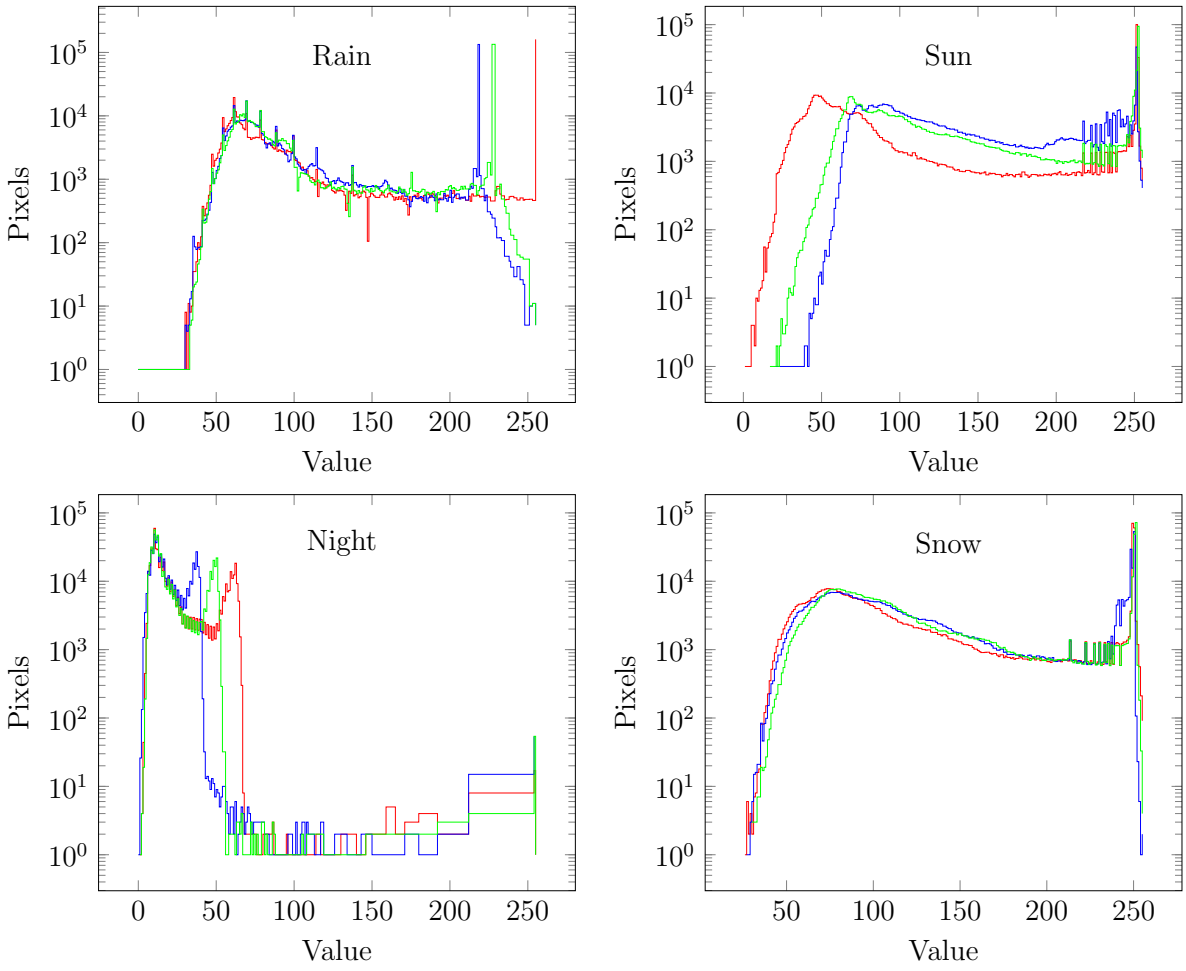


Figure 3.8: RGB histograms of all images

Figure 3.8 shows the histograms of the four example images (figures 3.3 - 3.6). A logarithmic scale is used as the peaks are extreme and would flatten the rest of the graph with a linear scale.

As with the brightness histograms (figure 3.7), the night images have large peaks for low values in all three colours, indicating an overall lack of brightness.

The histograms are significantly different for each of the four images. Although the individual colour histogram shapes are generally very similar for the snow and sun images, the colours for the snow image are more strongly correlated, with peaks and troughs in almost identical places (as shown by the overlap in the lines). This indicates an overall balance of colours, and contrasts the sun image, which contains a high number of pixels with less red but has otherwise similar histograms for green and blue.

The blue histogram of the rain image distinguishes it the most from the other images due to the very large peak at around 210 on the value axis. The red peak on the same image

also occurs significantly later, and doesn't feature a drop off like the other images do. This trend is largely consistent between different images of the same weathers at different times of the day.

Rain	Sun
(124, 128, 141)	(142, 166, 119)
Night	**Snow**
(20, 23, 27)	(145, 149, 144)

Table 3.1: The average hues of the images, shown as RGB values on a scale of 0 to 255

The average hue of the image at night is incredibly dark due to the lack of sunlight. The sunlit image has a relatively bright green average hue, largely due to the grass which is in a large portion of the image. This is a feature which stays consistent throughout sunlit images which isn't present in other weather conditions.

The snow image also has a green hue to it, though is significantly less saturated than the sun image. This is caused by the whiteness of the snowfall and the snow or ice that settles or forms on the grass in the image.

The image of the rain has a blue hue to it, which contrasts with the green of the sun and snow images. The grass appears less vivid due to the obstruction of the sun by cloudy skies and overall blue high can also be attributed to the refraction of the sky in the raindrops, as well as the overall darker blue hue of the sky itself.

3.2 Data Storing

Comma-Separated Value (csv) files were used to store the data in a convenient way for both anyone reading the data and for the WEKA library. The data stored included the name of the image, the time and date it was taken, as well as the actual weather information occurring in the image and the features extracted to train a classifier. Blank columns were left for the predicted weather information which would be filled in by the classifier once it was trained.

The csv file was stored on the Dropbox file sharing platform. It was updated on every interval so that other devices could access a complete set of data immediately when required.

name	date	time	actual icon	⋯	actual temperature	predicted icon	⋯
2015-02-09T23:55.jpg	2015-02-09	23:55:42	cloudy	⋯	3	–	⋯
2015-02-10T00:00.jpg	2015-02-10	00:00:42	cloudy	⋯	3	–	⋯
2015-02-10T00:05.jpg	2015-02-10	00:05:42	cloudy	⋯	2	–	⋯
2015-02-10T00:10.jpg	2015-02-10	00:10:42	cloudy	⋯	2	–	⋯
2015-02-10T00:15.jpg	2015-02-10	00:15:42	cloudy	⋯	2	–	⋯
2015-02-10T00:20.jpg	2015-02-10	00:20:42	cloudy	⋯	2	–	⋯
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋱

predicted temperature	b[0]	⋯	b[255]	r[0]	⋯	r[255]	g[0]	⋯	g[255]	b[0]	⋯	b[255]	average hue
–	124552	⋯	0	329362	⋯	10	21685	⋯	367	210463	⋯	398	#0F090F
–	187945	⋯	2	302139	⋯	15	22310	⋯	347	248283	⋯	492	#0A0A0C
–	183065	⋯	7	312448	⋯	22	23409	⋯	629	223409	⋯	491	#09100D
–	169364	⋯	1	294123	⋯	14	24012	⋯	711	257234	⋯	572	#040E12
–	10165	⋯	0	292341	⋯	12	21239	⋯	1	234829	⋯	481	#0F0511
–	109382	⋯	1	290312	⋯	19	25010	⋯	25	242348	⋯	748	#0F590B
⋮	⋮	⋱	⋮	⋮	⋱	⋮	⋮	⋱	⋮	⋮	⋱	⋮	⋮

Table 3.2: Example schema for csv file
(Split in half for ease of viewing)

3.3 Sanity Checking

Considering the high dependency of the project on the success and accuracy of the image gathering, it was vital to ensure that it was going smoothly. Although time-lapse footage exists which could otherwise have been used- the real weather information would have to be manually labelled on a frame by frame basis, which was impractical with so many data points.

Sanity checking ensures that everything is working, and if it isn't, that a user responsible is made aware as soon as possible to fix whatever problems may have arisen.

A sanity check which was employed in this project involved making sure that the Raspberry Pi was taking images and uploading them to dropbox continuously, as a large set of data was an integral part of the success of the project. The check employed confirmed that the real data was being gathered by querying the forecast.io API to ensure that weather information requests were occurring. This check did in fact come of use during the project, where a sudden disconnect in power supply from the Raspberry Pi due to cleaning meant that the existing error solutions built into the code did not have the chance to send an alert.

In addition to this, the main data gathering program was made to send an email alert should any exception be thrown throughout the image taking, data gathering, feature extraction and file saving/uploading stages. This provided a faster and more immediate notification when errors occurred than the sanity check detailed above.

Chapter 4

Evaluation

4.1 Cross Validation

Cross validation is a technique used to evaluate machine learning algorithms by testing them on all of the dataset rather than using most of the data to train the algorithm and using only a small portion for testing. In this project, the data was divided into testing and training sets with a 1 to 4 ratio, respectively. This was done five times such that each of the testing sets were unique and formed the complete dataset, so each point of data was used to train four classifiers and was tested by one (which it didn't train).

It was important not to train and test the machine learning classifiers using the same data, as classifiers which remember instead of learn (such as any database store and lookup based algorithms) would perform disproportionately well.

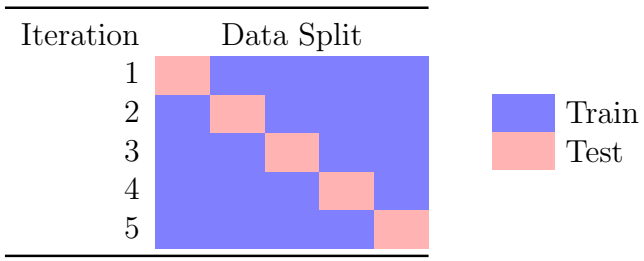


Figure 4.1: Cross Validation

The cross validation partitioning used is shown graphically in figure 4.1. At each iteration, a different set is used for testing and the rest of the data is used to train the classifier. After all 5 iterations, each data point will have been used to train the classifier four times and test it once. More generally, the data can be divided into k partitions such that the data set can be evaluated k times by averaging results without the need for a larger dataset.

The partitions were allocated randomly to avoid trends which would've been present due

to the ordering of the data.

4.2 Description of Data

The input data had to be analysed and described to give meaning to the evaluation. For the classification to be effective, the accuracy had to be higher than the mode of the data set; and for the regression based data, the root-mean-square error should have been within a standard deviation of the data.

4.2.1 Weather Type

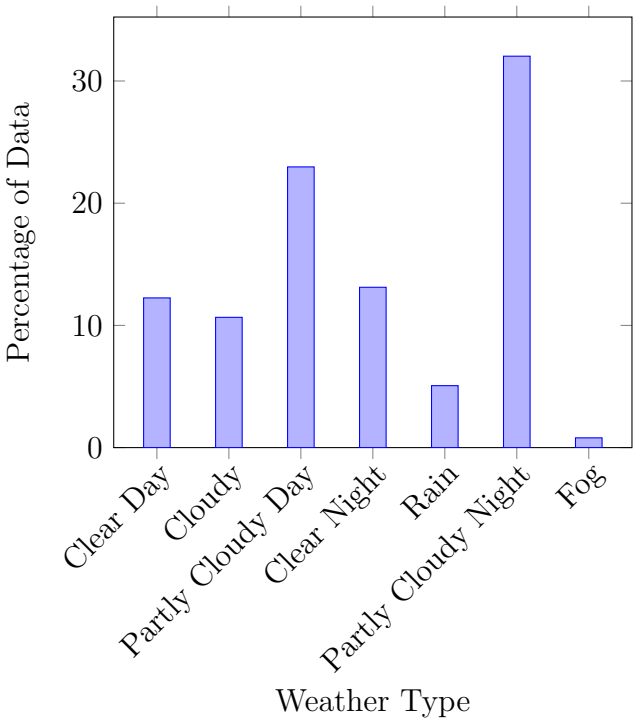


Figure 4.2: Distribution of weather types in the data set

There were initial concerns with how varied the weather type would be given the high likelihood of cloudy weather in the UK, especially during the period of data gathering. Fortunately, Forecast.io distinguishes between cloudy and partly cloudy, with the latter being further split into day and night classes. Due to this, the data is relatively spread out and a highly accurate classifier could not have been created by simply guessing cloudy all the time, as per the initial concerns.

Partly cloudy nights were still by far the most common type, with 32% of the data points.

The combined night values also account for 45% of the data, which as expected, accounts for about half the data.

4.2.2 Cloud Cover

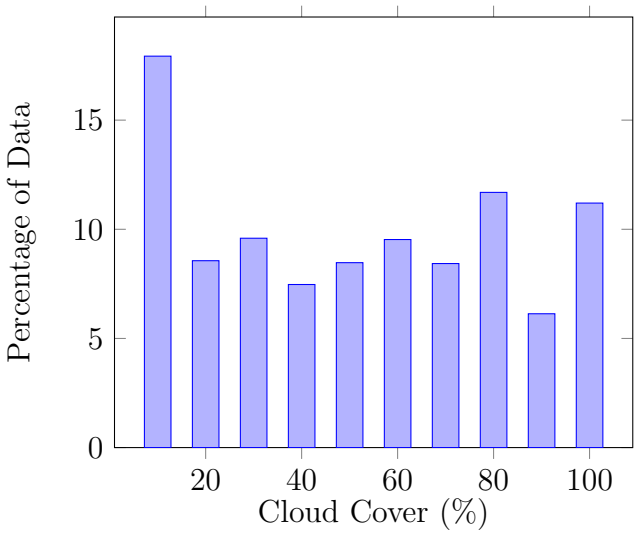


Figure 4.3: Distribution of cloud cover percentages in the data set

As figure 4.3 shows, the cloud cover is distributed quite uniformly between 0% (no cloud cover) and 100% (complete cloud cover), with the mean and median being almost equal. By comparing this graph to the icon graph (figure 4.2), it can be deduced that the 'Partially Cloudy' cweather type classification occurs approximately between 20% and 90% cloud cover, and that cloudy occurs above 90%. The standard deviation is 31% cloud cover, so the classifier should aim to classify within that percentage.

4.2.3 Precipitation

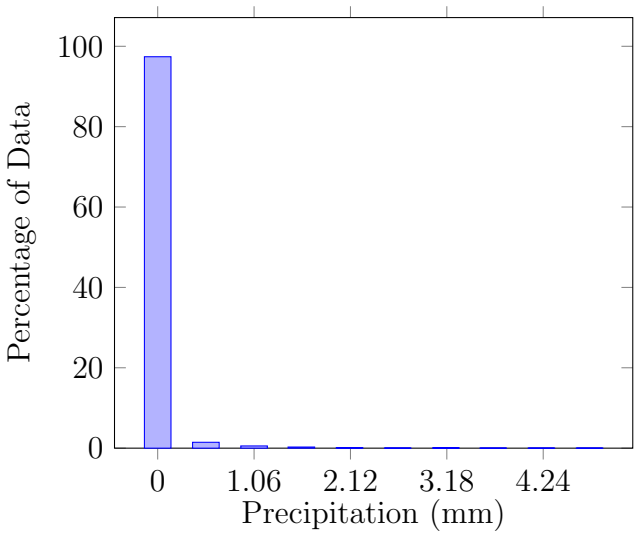


Figure 4.4: Precipitation data bar chart

Unlike the cloud cover, the precipitation is massively skewed toward the lower 10th percentile, which indicates that there was generally very little rain in the data gathering stage of the project. Over 97.4% of the data was within the lowest percentile.

Although the standard deviation was a relatively low 0.23mm, a highly accurate approximation of the precipitation level can be made by simply guessing 0.23mm in all cases, which would be within a standard deviation of the actual value in over 90% of the test cases. This is an unfortunate distribution, as the precipitation levels are not going to be a good indicator of classifier performance.

4.2.4 Temperature

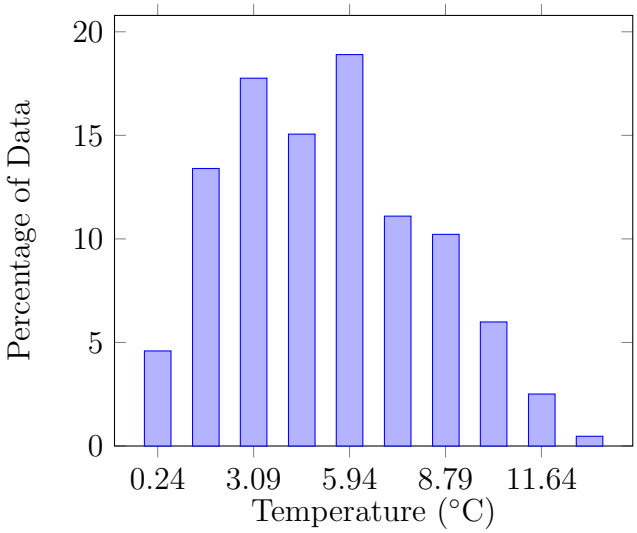


Figure 4.5: Temperature data bar chart

The temperature is more distributed than the precipitation, though not as uniformly as the cloud cover. The data was gathered over the winter and early spring months, and so the temperature is generally very low with later dates having the higher temperatures. This would likely be a more uniform distribution if the data was to be taken over a longer period of time encompassing at least all four seasons. The average temperature was 6.08°C and there was a standard deviation of 2.9°C.

4.2.5 Sunrise and Sunset

The sunrise and sunset times changed predictably throughout the data gathering process to reflect on the increasing daylight hours as winter ended. For the data as a whole, the standard deviation for the sunrise is 22 minutes whilst the standard deviation for the sunset is 19 minutes. These could likely be predicted with high accuracy if the time and dates were used as features to train the classifier.

4.3 Analysis of Classifiers

The discrete data was analysed by simply comparing the percentage of correct classifications for each class. The continuous data was analysed by calculating the root-mean-square

error, given by

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(x_t - y_t)^2}{n}}$$

where x_t is the actual value and y_t is the predicted value of the t^{th} point of data in a set of n data points.

4.3.1 Random Forest

Random forests were analysed with a varying number of trees ranging from 1 to 100, and also 200 and 1000. Results were obtained with each of the number of trees to analyse how performance changes for different classes with the increase in trees.

Weather Type

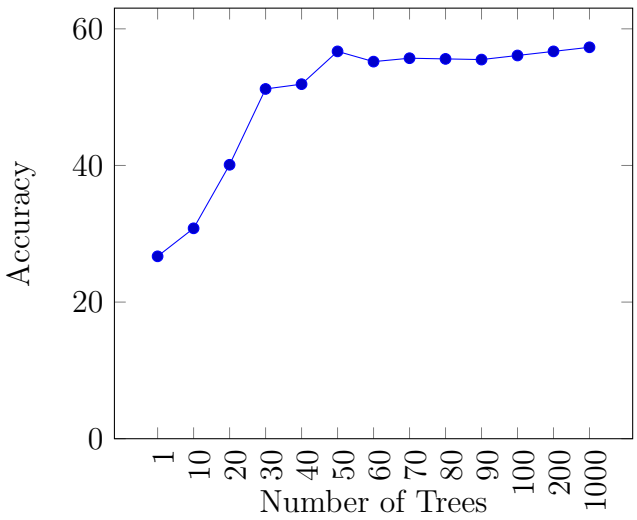


Figure 4.6: Weather type classification peformance per number of trees

The accuracy of the random forests increased significantly up to 50 trees, at which point it was 56.7% accurate. From there, the accuracy fluctuated between a maximum of 57.3% and a minimum of 55.2%. The maximum performance of the random forest for this feature set was therefore realised between 40 and 50 trees.

57% as an accuracy is good- well above randomly guessing and well above the mode value percentage, though still not accurate enough for any real world usage where potentially dangerous decisions can be influenced by weather.

Sunrise and Sunset

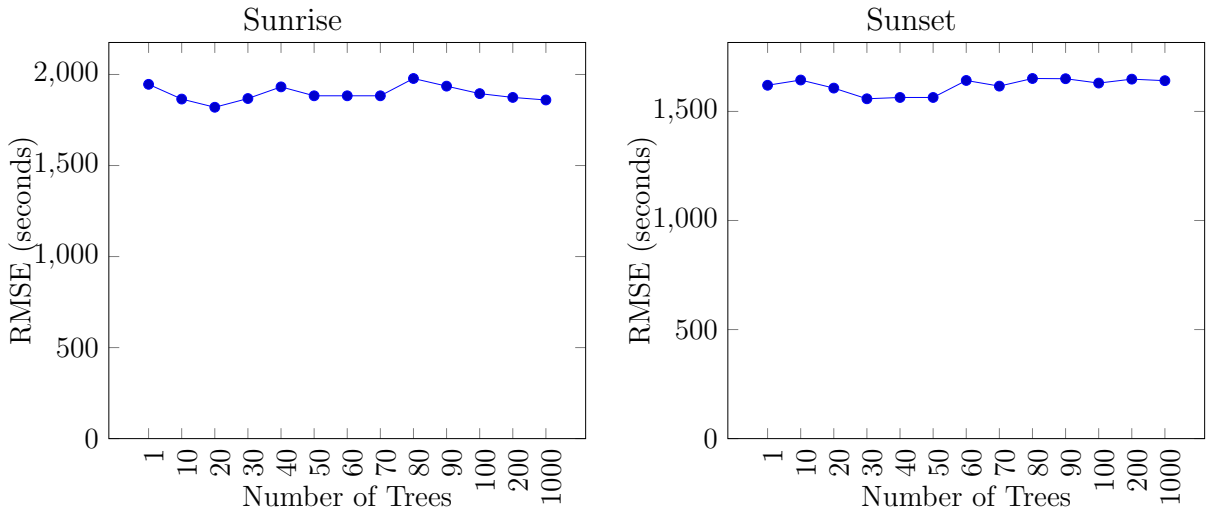


Figure 4.7: Sunrise and Sunset regression peformance per number of trees

There was no improvement in the accuracy of the prediction of sunrise as the number of trees increased, with the mean error continuously being about 30 minutes, which is over the standard deviation of 22 minutes, though still well within two standard deviations.

The sunset, as with the sunrise, also doesn't increase in accuracy as more trees are added to the forest. The average root-mean-square error of about 27 minutes is also outside of the standard deviation of 19 minutes, and for similar reasons to above, renders random forests a lacklustre way of predicting the sunset.

Given the algorithmic nature of sunset and sunrise times with respect to the day of the year, a significantly better set of results could be expected if the date was used as a feature.

To follow on this, sunrise and sunset times generally have little correlation with the features of individual images which were used to train the classifiers (it is almost impossible to tell when the sun will set from an image of a cloudy day). It is therefore unsurprising that the random forest classifier did not perform well for this class.

Cloud Cover

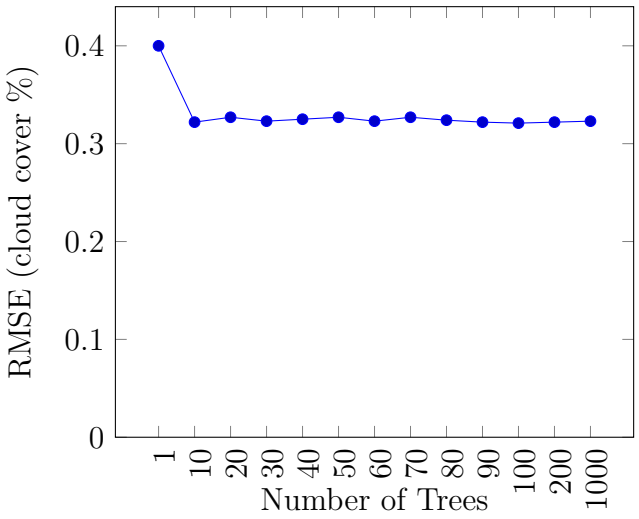


Figure 4.8: Cloud cover regression peformance per number of trees

From 10 trees onwards, the prediction of cloud cover did not increase in accuracy as the number of trees changed, staying at a root mean square error of about 32% throughout the tests (and 40% between 1 and 10 trees). This is almost exactly one standard deviation, and so Random Forests with at least 10 trees are accurate to an acceptable degree of error.

Precipitation

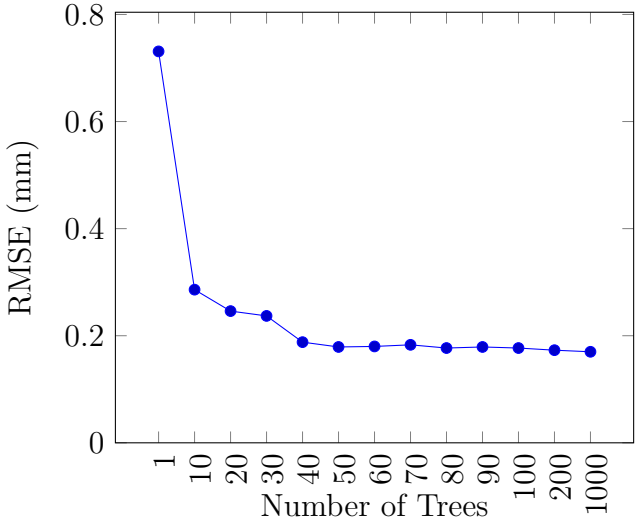


Figure 4.9: Precipitation regression peformance per number of trees

The precipitation prediction increased in accuracy as the number of trees increased, up to about a 0.17mm error, which is well within the 0.23mm standard deviation of the precipitation data. This performance was reached at about 40 trees, after which the performance changes were negligible as the number of trees increased.

Rather surprisingly, random forests are a rather good predictor of precipitation levels using the features that were extracted from the images.

Temperature

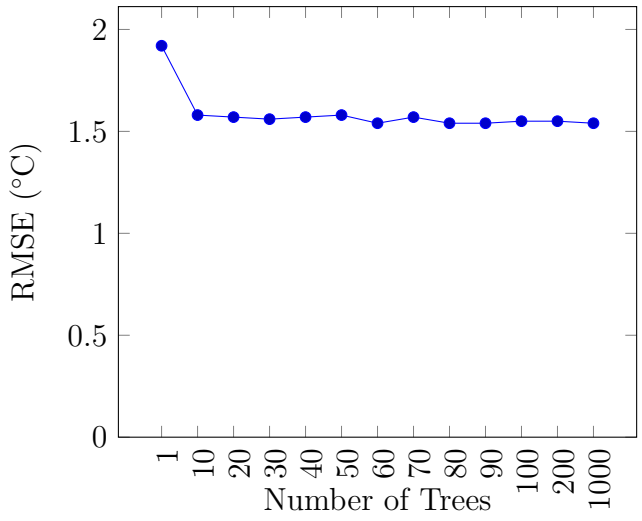


Figure 4.10: Temperature regression peformance per number of trees

The temperature prediction did not increase in accuracy as the number of trees increased (with at least 10), though the root-mean-square error came to about 1.54°C, which is significantly under the standard deviation of 2.9°C. As such, rather surprisingly, random forests are not only quite good at inferring temperature from images, but more generally it is possible to infer temperature from an image to a relatively high degree of accuracy. Of course, the temperature range of the data is rather limited (15°C) in terms of real-world temperature ranges, given the span of time of data gathering.

Summary

Overall, the Random Forest algorithm is generally the most successful, with very good results in predicting the weather type, temperature and precipitation, and with decent performance with the other classes. All the classifications and regressions reached their maximum accuracy by 50 trees, with most of them taking only 10 before significant performance gains were no longer occurring.

4.3.2 Multilayer Perceptron

Weather Type

The multilayer perceptron had a 36.1% accuracy rate with classifying the weather type. This is marginally better than assigning the mode result to every instance, and also better than the average case of randomly assigning types. However, this is by no means a reliable result for classifying the weather type as 36% is a generally low success rate for classification (it's wrong more often than it's right).

Sunrise and Sunset

The sunrise and sunset had a root-mean-square error of 8:12 and 5:11 minutes respectively. These are significantly better than the random forest, and overall means that the multilayer perceptron can (to a relatively good degree of accuracy) determine sunrise and sunset times.

Cloud Cover

The multilayer perception had a 35% root-mean-square error for cloud cover, which is outside of the 31% standard deviation, though only just. Being within two standard deviations, it's still a somewhat accurate way of deducing cloud cover.

Precipitation

The precipitation predictions had a root-mean-square error of 0.33mm, which is well outside of the standard deviation of 0.23mm.

Temperature

The multilayer perceptron had a 1.4°C root-mean-square error when telling the temperature, which is well within a standard deviation of 2.9°C. This is an excellent result and is unlikely to be beaten by the performance of any other classifiers.

4.3.3 J48

The J48 algorithm was also tried, though as it only works for classification and not regression, could only be tested on the weather type. It obtained a poor 16% accuracy for this; significantly lower than the mode of the data set.

	Random Forest	Mutilayer Perceptron	J48
Weather Type	**57.3%**	36.1%	16%
Sunrise	30:00	**8:12**	N/A
Sunset	27:00	**5:11**	N/A
Cloud Cover	**32%**	35%	N/A
Precipitation	**0.17mm**	0.33mm	N/A
Temperature	1.54°C	**1.4°C**	N/A

Table 4.1: Complete table of results
The best results are show in bold. The weather type is a percentage accuracy (higher is better), whereas the other results are errors (lower is better).

Summary

Overall the Multilayer Perceptron is a rather inaccurate classification algorithm in all cases with the exception of the rather surprisingly accurate temperature reading.

4.4 Performance

Performance was measured by comparing system nanosecond clocks before and after running the classifier. Both measurements were taken on the same machine with no interference or background processes running such that ideal conditions could be imitated as closely as possible.

The Random Forest was made with 50 trees (to reflect the best results) and the Multilayer Perceptron was tested with a 3, 5, 3 layer configuration, as was used to obtain results.

The Random Forest ran at an average time of 0.099 seconds per data point, whilst the Multilayer Perceptron averaged 0.904 seconds. This is a difference in one order of magnitude, and the conclusion can be drawn that with the configurations used for these results, Random Forests are significantly faster than Multilayer Perceptrons.

4.5 Overall Results

The overall results show that the random forest and multilayer perceptron algorithms each performed the best in three of the six features. Neither algorithm performed within the standard deviation for Cloud Cover regression (though random forests were closer).

The weather type classification accuracy of random forests can be analysed further to see which expected and actual results were the most and least likely.

	Partly Cloudy Night	Fog	Clear Night	Cloudy	Partly Cloudy Day	Rain	Clear Day
Partly Cloudy Night	**96.48**	25.00	94.40	26.98	0.96	76.44	0
Fog	0	**73.53**	0	0	0.32	0	0
Clear Night	1.35	0	**2.88**	0.70	0.21	9.42	0
Cloudy	2.01	1.47	2.64	**72.33**	19.57	2.09	6.02
Partly Cloudy Day	0	0	0	0	**69.43**	1.57	18.05
Rain	0.15	0	0.08	0	2.23	**10.47**	9.02
Clear Day	0	0	0	0	7.11	0	**66.92**

Table 4.2: Confusion matrix of inferred vs actual results.
Shown as a percentage of the total real values. Real values top, inferred down.

The confusion matrix is laid out with actual weather types across the columns, and predicted types on the rows. Each column shows with which percentages a given weather type was classified into which other weather types.

As table 4.2 shows, the classifiers struggled with clear nights and rain, classifying each with 2.88% and 10.47% accuracies, respectively. In both cases, they were classified as partly cloudy nights, largely because the features in each were likely very similar (dark and blue) and partly cloudy night was the mode, so outputting it in every unsure case was the best way to maximise accuracy.

Chapter 5

Conclusions

5.1 Summary

A classifier was built successfully with image and truth data gathered throughout the project. The accuracies of different machine learning algorithms were explored and compared to give the overall best results. The success criteria were fully met.

1. The sunrise and sunset times were inferred.
2. The weather type was inferred in each frame.
3. The cloud cover was inferred in each frame.
4. An accuracy of the classifier was obtained for each feature, which represents the likelihood of correct classification, or the error for regressions.
5. Graphs were used to visually demonstrate the accuracy of the classifier.

5.2 Future Work

Mistakes and oversights as well as interesting discoveries were made throughout the project which would likely change the direction of the work should the same project be repeated in the future. Should more time have been made available, more areas also would've been explored. This section details some of these points.

5.2.1 Extensions

A larger data set would give considerably more variability to the results, given that there wasn't much snow or direct sunlight throughout the data gathering stages of the project.

At least a years worth of data would also give the option of predicting future weather based on past trends in pictures, whereas with the current data set (until March), it's unreasonable to assume any level of accuracy for predicting that there will be sun in June (though the general trend is that temperature is increasing, it'll still be difficult without more data).

More classifiers could also have been explored or even custom built (using either hard coded rules or known algorithms adapted to maximise performance for the data set specifically) rather than using in-built WEKA libraries to compare performances.

5.2.2 Changes

A limiting factor in the accuracy of the classifiers is the limited feature set. Splitting the image into a set amount of segments (4x4 areas) for instance and computing the histograms for each of those would preserve the location of features within the image rather than treating it as just an array of pixels with no consideration for the order of them.

Figure 5.1: One of the images split into a 4x4 grid

Another feature which could've been added was using histograms of oriented gradients (HOGs) to store the gradients throughout the image, as this would also preserve the general 'flow' of the images. This would generally have not differentiated most images apart, as the images are taken of a single location, though it would've been worth exploring whether rain or other precipitation effects create additional gradients which could be used to increase accuracy.

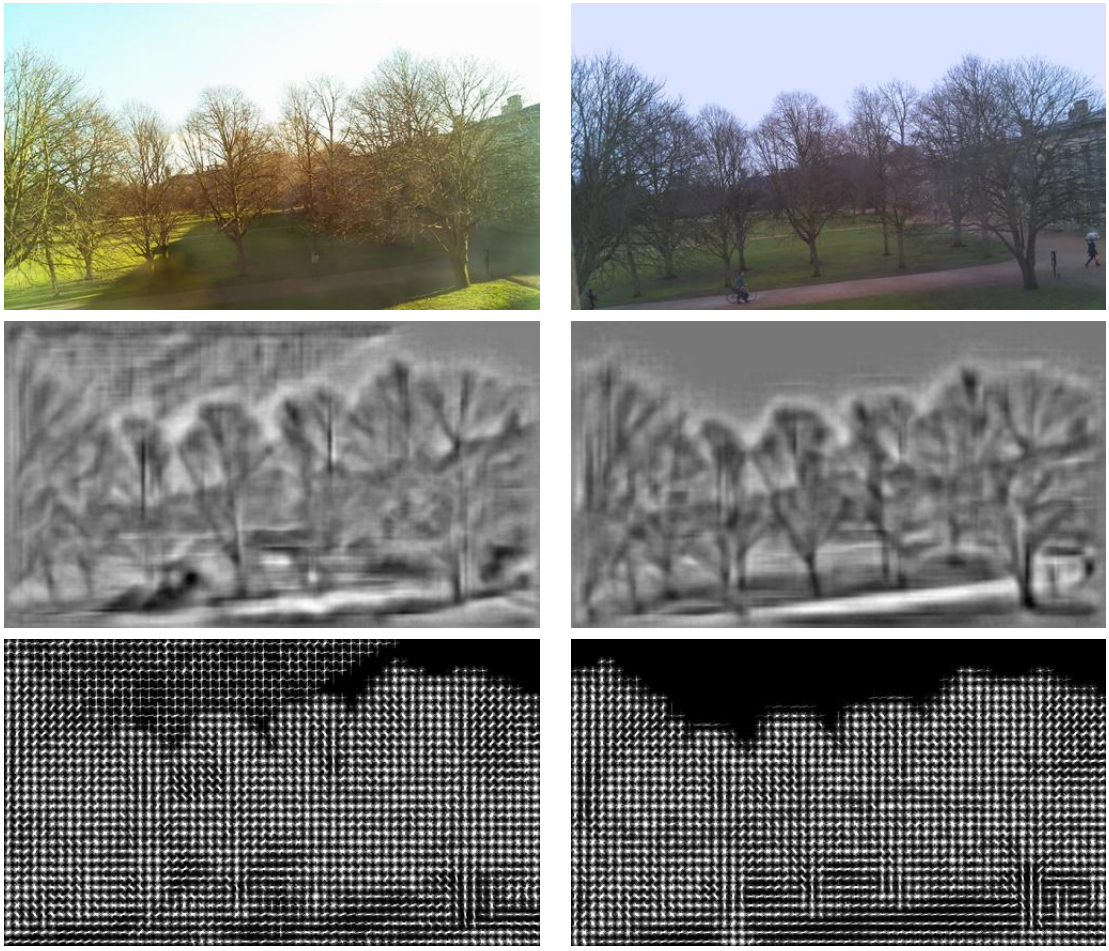


Figure 5.2: Creation process of histogram of oriented gradients with the sun and rain images, respectively

Figure 5.2 shows the creation process of the HOG. Beginning initially with the original image, the edges are detected and the sharpness of the edge is represented by the strength of the gradient through it. A HOG matrix is then generated which contains all the gradients represented as vectors.

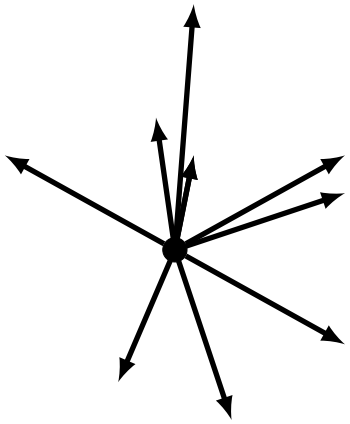


Figure 5.3: Visual representation of a histogram of oriented gradients

The vectors can then be combined into a single graph as shown in Figure 5.3, which would be stored on a degree and magnitude basis to be fed to the classifier.

Multiple pictures with varying exposures and gain settings could also be used rather than relying on the automatic settings on the Raspberry Pi. This would have a large impact on the classification of weather types at night, as well as on the classification of bright weather types such as direct sunlight and snow.

5.2.3 Closing Words

In conclusion, time-lapse video footage was indeed able to be associated with external weather data, as was demonstrated with the creation of a reasonably accurate weather classification system. Time lapse footage created an environment in which the image was kept completely unchanged with the exception of variables which could be extracted to become features to train a classifier- something which was not obtainable with a generic image set.

With some further work and research into feature extraction and a larger dataset, this could certainly reach a high enough accuracy to be used reliable in day to day weather classification, and the work done here could be extended into other steady-position time-lapse video applications.

Bibliography

[1] J. K. Lazo, M. Lawson, P. H. Larsen, and D. M. Waldman, "U.s. economic sensitivity to weather variability," 2010.

[2] M. Roser and F. Moosmann, "Classification of weather situations on single color images," in *Intelligent Vehicles Symposium, 2008 IEEE*, pp. 798–803, June 2008.

[3] O. Chapelle, P. Haffner, and V. Vapnik, "Support vector machines for histogram-based image classification," *Neural Networks, IEEE Transactions on*, vol. 10, pp. 1055–1064, Sep 1999.

[4] A. Bosch, A. Zisserman, and X. Muoz, "Image classification using random forests and ferns," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, Oct 2007.

[5] W. Liu and H. Li, "Time-lapse photography applied to educational videos," in *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pp. 3669–3672, April 2012.

[6] R. P. Foundation, "Raspberry pi," 2015. `https://www.raspberrypi.org/`.

[7] R. P. Projects, "Raspberry pi model b hardware general specification," 2015. `http://www.raspberry-projects.com/pi/pi-hardware/raspberry-pi-model-b/hardware-general-specificationshttp://www.raspberry-projects.com/pi/pi-hardware/raspberry-pi-model-b/hardware-general-specification`.

[8] R. P. Foundation, "Raspberry pi camera module," 2015. `https://www.raspberrypi.org/products/camera-module/`.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, Nov. 2009.

[10] WEKA, "Weka api," 2015. `http://weka.sourceforge.net/doc.stable/`.

[11] "Forecast.io," 2015. `https://www.forecast.io/`.

[12] "National oceanic and atmospheric administration," 2015. http://www.noaa.gov/.

[13] "Met office," 2015. http://www.metoffice.gov.uk/.

[14] D. E. (dvdme), "Forecastio-lib-java," 2015. https://github.com/dvdme/forecastio-lib-java.

[15] I. Dropbox, "Dropbox," 2015. https://www.dropbox.com/.

[16] P. H. Tom Preston-Werner, Chris Wanstrath, "Github," 2015. http://www.github.com.

[17] G. Bradski, "Opencv," *Dr. Dobb's Journal of Software Tools*, 2000.

[18] "Class buffered image," 2015. http://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html.

[19] E. E. Sutter and D. Tran, "The field topography of {ERG} components in man—i. the photopic luminance response," *Vision Research*, vol. 32, no. 3, pp. 433 – 446, 1992.

[20] J. D. Cook, "Three algorithms for converting color to grayscale," 2015. http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/.

Chapter 6

Appendices

6.1 Code Extracts

6.1.1 Data Harvester

The following code was ran on the Raspberry Pi and was responsible for the data gathering and uploading.

```
1  public class DataHarvester extends TimerTask {
2
3    // 5 minute intervals
4    private static final int TIMEOUT = 5 * 60 * 1000;
5
6    private static DropboxUploader _uploader;
7
8    private static Date getNext5MinuteMark() {
9      Calendar now = Calendar.getInstance();
10     int mod = now.get(Calendar.MINUTE) % 5;
11     now.add(Calendar.MINUTE, mod != 0 ? 5 - mod : 0);
12     return now.getTime();
13   }
14
15   private static String getFormattedDateTime(Date date) {
16     SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-dd'T'HH:mm:ss");
17     return sdf.format(date);
18   }
19
20   private static String getFormattedDate(Date date) {
21     SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-dd");
22     return sdf.format(date);
23   }
24
25   private static String getFormattedTime(Date date) {
26     SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
```

```
27      return sdf.format(date);
28    }
29
30    @Override
31    public void run() {
32      try {
33        Date datetime = new Date();
34        CsvData data = new CsvData();
35
36        // Take photo
37        new PiCamera().capture(getFormattedDateTime(datetime) + ".jpg");
38        data.setName(getFormattedDateTime(datetime) + ".jpg");
39        data.setDate(getFormattedDate(datetime));
40        data.setTime(getFormattedTime(datetime));
41
42        // Get forecast
43        Forecast forecast = new Forecast(getFormattedDateTime(datetime));
44        data.setActualIcon(forecast.getIcon());
45        data.setActualSunrise(forecast.getSunriseTime());
46        data.setActualSunset(forecast.getSunsetTime());
47        data.setActualCloudCover(String.valueOf(forecast.getCloudCover()));
48        data.setActualPrecipitation(String.valueOf(forecast.getPrecipitation()));
49        data.setActualTemperature(String.valueOf(forecast.getTemperature()));
50        // data.setActualVisibility(String.valueOf(forecast.getVisibility()));
51
52        // Predicted data will be filled later
53        data.setPredictedCloudCover("--");
54        data.setPredictedIcon("--");
55        data.setPredictedSunrise("--");
56        data.setPredictedSunset("--");
57        data.setPredictedPrecipitation("--");
58        data.setPredictedTemperature("--");
59        data.setPredictedVisibility("--");
60
61        // Retrieve captured image
62        String imageFile = getFormattedDateTime(datetime) + ".jpg";
63        Image image = new Image(imageFile);
64
65        // Upload image
66        _uploader.uploadFile(imageFile, "/project/data/images/" + imageFile);
67        System.out.println("image uploaded");
68
69        if (_uploader.doesFileExist("/project/data/images/" + imageFile)) {
70          Files.deleteIfExists(Paths.get(imageFile)); // Save space on the pi.
71        }
72
73        // Image feature extraction
74        data.setAverageHue(Hue.getAverageHueString(image));
75        data.setBrightnessHistogram(Brightness.getBrightnessHistogram(image));
76        data.setRedHistogram(Hue.getRedHistogram(image));
77        data.setBlueHistogram(Hue.getBlueHistogram(image));
```

```
78          data.setGreenHistogram(Hue.getGreenHistogram(image));
79
80          // Add line to csv file
81          new CsvWriter("data.csv").addEntry(data);
82
83          // Upload csv file
84          _uploader.uploadFile("data.csv", "/project/data/data.csv");
85
86      } catch (Exception e) {
87          e.printStackTrace();
88          EmailSender.sendEmail(e.message);
89      }
90    }
91
92    public static void main(String[] args) throws IOException {
93      _uploader = new DropboxUploader();
94
95      Timer timer = new Timer();
96      timer.scheduleAtFixedRate(new DataHarvester(), getNext5MinuteMark(), TIMEOUT);
97    }
98  }
```

6.1.2 Classifier

The following code was ran once the data was gathered to train the classifier and fill the predicted value columns on the csv file.

```
1  public class DataClassifier {
2
3      private Classifier _classifier;
4      private int _classIndex;
5      Instances _trainingData;
6
7      public DataClassifier(String trainingfileName, int classIndex, Classifier classifier
8      DataSource source = new DataSource(trainingfileName);
9
10     try {
11         _trainingData = source.getDataSet();
12     } catch (Exception e) {
13         e.printStackTrace();
14         System.out.println(trainingfileName);
15         return;
16     }
17
18     // String attribute change in weka, fix.
19     StringToNominal stn = new StringToNominal();
20     stn.setInputFormat(_trainingData);
21     String[] options = new String[2];
22     options[0] = "-R";
23     options[1] = "";
```

```
24
25      if (_trainingData.checkForStringAttributes()) {
26        for (int i = 0; i < _trainingData.numAttributes(); i++) {
27          if (_trainingData.attribute(i).type() == Attribute.STRING) {
28            options[1] += (i + 1) + ",";  // i + 1 because columns are 1-indexed only when
29          }
30        }
31
32        // remove last comma
33        options[1] = options[1].substring(0, options[1].length() - 1);
34      }
35
36
37      stn.setOptions(options);
38      _trainingData = Filter.useFilter(_trainingData, stn);
39
40      _trainingData.setClassIndex(classIndex);
41
42      _classifier = classifier;
43
44      // Remove name/data/time from training features
45      Remove remove = new Remove();
46      String splitOptions = "-R 1-" + classIndex + "," + (classIndex + 2) + "-17";
47      remove.setOptions(Utils.splitOptions(splitOptions)); // Name, date, time, actuals, p
48      remove.setInputFormat(_trainingData);
49      _trainingData = Filter.useFilter(_trainingData, remove);
50
51      try {
52        _classifier.buildClassifier(_trainingData);
53      } catch (Exception e) {
54        e.printStackTrace();
55        System.out.println(trainingfileName);
56        return;
57      }
58
59      _classIndex = classIndex;
60    }
61
62    public void fillFile(String fileName) throws Exception {
63      DataSource source = new DataSource(fileName);
64      Instances data = source.getDataSet();
65
66      data.setClassIndex(_classIndex);
67
68      _trainingData.delete();
69
70      for (int i = 0; i < data.numInstances(); i++) {
71        _trainingData.add(data.instance(i));
72      }
73
74      CsvManipulator manipulator = new CsvManipulator(fileName);
```

```
75
76      Instances labelled = new Instances(data);
77
78      // Keep labels, remove training data.
79      for (int i = 0; i < _trainingData.numInstances(); i++) {
80        try {
81          double label = _classifier.classifyInstance(_trainingData.instance(i));
82
83          labelled.instance(i).setClassValue(label);
84
85          String s = _trainingData.classAttribute().value((int) label);
86          if (s.isEmpty())
87            s = String.valueOf(_classifier.distributionForInstance(data.instance(i))[0]);
88
89          manipulator.replaceEntryOnLine(i, _classIndex + 7, s);
90
91        } catch (Exception e) {
92          e.printStackTrace();
93        }
94      }
95    }
96
97    public void finalize() {
98      _trainingData.clear();
99      _trainingData.delete();
100   }
101
102   private static final String _prefix = "/home/romek/projects/timelapseweather/data/";
103   private static final String _sep = "/";
104
105   private static void doClassify(Classifier classifier, String name) throws IOException,
106     FileUtils.copyDirectory(new File(_prefix + "split" + _sep), new File(_prefix + name
107
108     for (int j = 0; j < 5; j++) { // Files
109       for (int i = 5; i <= 9; i++) { // Columns
110         try {
111           // Train classifier
112           DataClassifier dataclassifier = new DataClassifier(_prefix + name + _sep + "sp
113
114           // Use classifier
115           dataclassifier.fillFile(_prefix + name + _sep + "split" + _sep + "test" + j +
116           dataclassifier.finalize();
117         } catch (Exception e) {
118           System.out.println("Error on file " + j + " and column " + i);
119           e.printStackTrace();
120         }
121       }
122     }
123
124     // Merge separate files
125     String[] files = new String[5];
```

```
126       for (int i = 0; i < 5; i++) {
127         files[i] = _prefix + name + _sep + "split" + _sep + "test" + i + ".csv";
128       }
129
130       CsvManipulator.mergeCsvFiles(files, _prefix + name + _sep + "datamerged.csv");
131     }
132
133     public static void main(String[] args) throws IOException {
134       // Split the csv file into partitions to be used for cross validation. Does so rando
135       new CsvManipulator(_prefix + "data.csv").splitCsvFile(5, _prefix + "split\\");
136
137       MultilayerPerceptron mlp = new MultilayerPerceptron();
138       mlp.setHiddenLayers("3, 5, 3");
139       try {
140         doClassify(mlp, "mlp");
141       } catch (IOException | InterruptedException e) {
142         e.printStackTrace();
143       }
144     }
145
146 }
```

Roman Kolacz
Downing College
rk476

Part II Project Proposal

Time-Lapse Based Weather Classification

19th October 2014

Project Originator: *Alan Blackwell*

Resources Required: See attached Project Resource Form

Project Supervisor: *Advait Sarkar*

Signature:

Director of Studies: *Robert Harle*

Signature:

Overseers: *Markus Kuhn* and *Neal Lathia*

Signatures:

Introduction and Description of the Work

Time-lapse videos are a relatively unexplored aspect of data visualisation, and have a rather unique property amongst video of being able to hold information about incredibly large periods of time in a relatively small amount of time and space. This is especially true in the case of events which happen slowly and gradually, such as the weather conditions in a particular place over a long period of time.

This project will aim to develop an alternative way of associating time-lapse videos with external data sources by attempting to determine the weather from time-lapse footage and evaluating it against true weather information. This will be demonstrated graphically.

The time-lapse video will be made as part of the project, though any existing video will also suffice for the project.

Literature Review

Classification of Weather Situations on Single Colour Images

An approach which forms the basis of the weather-labelling per frame is described in the paper "Classification of Weather Situations on Single Colour Images"[4]. This paper describes classifying weather using image metrics to be used to assist driving-assistance systems which are heavily dependent on clear conditions. It achieves very accurate results using only two labels (clear and rain), and reasonable accuracy when adding a third, "light rain" label. This project will build on this by adding further labels to the weather classifying, as well as incorporating a cloud-cover measure.

The paper also mentions inaccuracies relating to weather situations containing heavy fog or rain, as visibility is greatly reduced. These will have to be taken into consideration and/or mentioned in the evaluation.

Weather Identifying System Based on Vehicle Video Image

"Weather Identifying System Based on Vehicle Video Image"[5], describes using image comparison to get the weather conditions on a road. A 'background' image is obtained (and updated should a more suitable image be detected), and then weather conditions are inferred by analysing the images obtained by subtracting each of 1000 frames in a video from the background image. Although this technique isn't going to be used to label weather in the approach which will be presented in this project, a similar technique may very well be employed to determine cloud cover percentage.

Classification of Road Conditions

The paper, "Classification of Road Conditions"[6], also applies weather classification to driving conditions. It uses an inverse-least-squares method of inferring the weather from not only images, but a multitude of weather data including temperature, humidity and wind speed. This project will not be using these additional variables, though the results will likely be evaluated with their aid.

Time-Lapse Photography Applied to Educational Videos

Although unrelated to weather classifying, "Time-Lapse Photography Applied to Educational Videos"[7] explores applying time-lapse videos to educational videos to aid with analysing long term trends and discarding short term 'noise'. This is comparable to what this project is aiming to achieve through the use of time-lapse video.

Two-Class Weather Classification

"Two-Class Weather Classification"[8] also uses image processing techniques to classify weather into two distinct labels: cloud and sun. It initially explores analysing pixel brightness distribution, which appears to fail in a general case. The paper then explores various techniques for looking for signs of a sunny day. One such technique involves detecting the sky in the image (something which this project may well have to incorporate) and checking for a blue hue. The paper also explores using edge detection to find shadows, which is found to be very vulnerable to false positives in a cloudy case; as well detecting bright reflections in reflective objects- two techniques which will not be employed in this project, though may be useful for discussion purposes.

Resources Required

- Obtaining the data will require a Raspberry Pi[1] with an attached camera module.
- Depending on the frequency of frame capture and resolution, some additional disk space may be required to store the data.
- OpenCV[2] will need to be installed.
- If the machine-learning part of the project is to be undertaken, Weka[3] will need to be installed.

Substance and Structure of the Project

The OpenCV library will be used to obtain the following image metrics from individual frames:

- Brightness
- Average Hue/Saturation
- Brightness curve

This data will then be analysed and compared to the images so that a set of hard-coded rules can be derived which will determine which weather label can be attached to each image.

Frames in which the sun is shining are likely to have have brightness curves with very high peaks and low troughs, in contrast to relatively flat curves of cloudy images. This would come as a result of shadow and reflections skewing the curve in sunlit images, whereas cloudy images will be generally more evenly (and dimly) lit as a result of the clouds dispersing and absorbing sunlight.

Snow will likely cause the entirety of the image to generally appear brighter than expected and with less saturation, as a result of the reflective properties of snow.

Rain will likely be the most challenging weather label to place, as the conditions will decrease the brightness of the frame in a similar way to dawn and dusk would, and otherwise be relatively similar to cloudy conditions. One potential situation to look out for would be reflections in puddles, though these aren't guaranteed to happen, and may have insignificant effects on the image even if they do. A probable situation is that the saturation of the frame decreases and the hue shifts towards a blue/grey shade, though this will need to be explored during the progress of the project, and the accuracy of which will need to be heavily analysed.

The cloud cover will be inferred by isolating the sky from the rest of the image, and then analysing the brightness and hue of the relevant part of each frame.

Evaluation

The weather classifier will need to be evaluated alongside real weather data. This will be obtained from forecast.io[9], which contains an API[10] for obtaining precise (latitude and longitude) weather information. This is free to use for up to 1000 polls a day, which will easily suffice for this project.

The data will be split into training and testing sections with a 4:1 split respectively, as (especially in the extension case involving machine learning), data which is used to train

the classifier will have a positively skewed accuracy. The testing section will be used to evaluate the classifier.

Each aspect of the classifier will then be evaluated separately.

Weather Labels

The weather labels, being discrete sets of data, will be tested for accuracy by simply calculating the percentage of correctly assigned labels. A truth matrix will be created to highlight the concentration of errors on a per-case basis.

Cloud Cover

The cloud cover will be calculated as a percentage and compared to the real value which is available from the Forecast API. The accuracy of this will be manually checked, given that a camera pointing outside will cover a relatively large section of the sky. The accuracy of this aspect of the classifier will be calculated using root mean square error.

Sunset and Sunrise

The sunset and sunrise times will be tested for accuracy by also using root mean square error, using the difference between actual and inferred time, in minutes.

Extensions

Depending on the progress of the project at specific points, the following additional features may be implemented, in order of priority:

- Multiple images per frame may be taken at varying exposure and gain settings as a large difference in luminance within a single frame (which will likely happen on a sunny day), may make other parts of the image unnaturally dark should the camera use automatic settings.
- Weather prediction and likely trends can be derived from existing data. Information predicted in such a way can be compared to inferred data and accurate data when the related time period passes.
- Machine learning can be used instead of hard-coded rules on determining weather, as both the image metrics and accurate weather information will be available as a learning data set. Weka will be used for this.

Success Criterion

For the project to be deemed a success the following items must be successfully completed.

1. The sunrise and sunset times can be inferred from a time lapse video.
2. The weather in each frame can be inferred and labelled one of the following:
    - Clear
    - Cloud
    - Rain
    - Snow
3. The cloud cover can be inferred from each frame within the video.
4. A measure of confidence can be assigned to each of the above criteria to reflect the probability of incorrectly inferred data.
5. The above data can be graphically demonstrated and shown to work to some degree of accuracy.
6. The accuracy of the data must be evaluated.
7. The dissertation must be planned and written.

Timetable and Milestones

24th October - 6th November

Research how to use the Raspberry Pi and camera module to take time lapse videos, and set up a fixed position for it to gather data from for the rest of the project.

Research the OpenCV library and how it can be used to gather the data required, as well as how the data gathered can be used to infer the conditions described in the success criteria.

Milestones

- Produce a short, not necessarily usable time lapse video using the Raspberry Pi.
- Begin gathering data for remainder of project.
- Found/made a relatively hassle free mount for the Raspberry Pi and camera.
- Produce some small demo's and examples using OpenCV.

7th November - 18th December

Use the research gathered to design and implement algorithms which can establish weather conditions from frames based on hard coded rules.

Milestones

- A working library for inferring weather conditions from images.
- Tests written for said library.

Key Dates

- NPR End: 14th December

19th December - 1st January

Review algorithms and adjust timetable if further tests and research is needed for a working, usable library.

Implement the library to the thus-gathered time lapse video, and design a graphical demonstration program to view the data through. Assign a confidence value to each frame to reflect the chance of a correct result.

Milestones

- A graphical visualisation of the data extracted from the time lapse video.
- A confidence measure for the weather at each given frame.

2nd January - 29th January

Work on evaluating the data by gathering accurate weather information for each obtained frame and automatically labelling them with said data. Compare some specific instances to vaguely gauge the accuracy of the program.

Milestones

- Accurate data gathered for evaluation.
- A general idea of how accurate the data is.

Key Dates

- NPR Start: 4th January

30th January - 19th February

Work on more formally evaluating the accuracy of the data, along with the reliability of the confidence measure assigned to each piece of data. This can also be graphically shown.

Milestones

- Complete data on accuracy of algorithm.
- Graphical visualisation of said accuracy.

Key Dates

- Progress Report Deadline: 30th January
- Progress Report Presentation: 5th, 6th, 9th or 10th February
- Dissertation Lecture: 11th February

20th February - 5th March

Review work and adjust timetable if needed. Enhance test program for a more friendly UI which is more suitable for demonstrative purposes.

Milestones

- Program should be fully complete, usable and presentable.

6th March - 2nd April

Write dissertation. Notes should have been gathered throughout the previous stages, so these should be read first for familiarisation of the previous weeks of work.

Milestones

- First draft of dissertation should be complete.

Key Dates

- NPR End: 15th March

3rd April Onwards

Take a look at what needs attention, if anything, and work on that. Look through dissertation and ensure it's up to submission standards.

Milestones

- Final dissertation should be complete.

Key Dates

- NPR Start: 12th April
- Project Deadline: 15th May
- Supervisor Report Deadline: 20th May
- Exams: 2nd-4th June
- Viva Announcement: 12th June
- Vivas: 15th June
- NPR End: 21st June

References

[1] Raspberry Pi Computer
`http://www.raspberrypi.org/`

[2] Open CV Library
`http://opencv.org/`

[3] Weka 3 Library
`http://www.cs.waikato.ac.nz/ml/weka/`

[4] Classification of Weather Situations on Single Colour Images
Martin Roser and Frank Moosmann, 2008
`http://www.mrt.kit.edu/z/publ/download/Roser_al2008iv.pdf`

[5] Weather Identifying System Based On Vehicle Video Image
Hong Jun Song, 2011
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5970722

[6] Classification of Road Conditions
Patrik Jonsson, 2011
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6059917

[7] Time-Lapse Photography Applied To Educational Videos
Wei Liu and Hongyun Li, 2012
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6202303&tag=1

[8] Two-Class Weather Classification
Cweu Lu, Di Lin, Jiayi Jia and Chi-Keung Tang, 2014
http://www.cse.cuhk.edu.hk/leojia/papers/weatherclassify_cvpr14.pdf

[9] Forecast.io
https://forecast.io/

[10] Forecast.io Developers API
https://developer.forecast.io/docs/v2