Automated Front Detection

Using computer vision and machine learning to explore a new direction in automated weather forecasting

Simen Skaret Karlsen

Master thesis
Department of Information Science and Media Studies

UNIVERSITETET I BERGEN

16.03.2017

2017

Automated Front Detection

Simen Skaret Karlsen

https://bora.uib.no/

Abstract

In weather forecasting, automation and computing are the driving forces of innovation. More computing power and better techniques allow for faster and more accurate weather data systems. The task of detecting fronts (interfaces between different air masses) in weather systems has yet to be solved computationally with such accuracy. In computer science and information science research, the techniques in artificial intelligence used for pattern recognition are constantly evolving and solving new problems, both in the weather domain and elsewhere. I therefore explore whether artificial intelligence can be used to help detecting fronts in weather systems, as well as what weather features are useful to study in this endeavor.

In my Master's project I have developed an automatic front detection system in cooperation with weather service provider StormGeo, under the Design Science Research paradigm. The study aims to further our understanding of AI techniques and their use in weather analysis, through the design, development and use of an information system. The research follows in the footsteps of recent developments in several research fields, both within meteorology, weather prediction, data modelling, computer vision and machine learning. The system development was based on core principles of agile and lean software development methodologies, and used commonly available tools and techniques.

The resulting system identifies fronts using computer vision techniques, and classifies them using machine learning techniques and expert knowledge in meteorology. The system is fairly accurate in finding the major front lines in a weather system, and is even able to find some fronts that meteorologists have missed, but it fails to pick up many subtle details that expert use in front detection. The system excels at classifying some types of fronts, but performs poorly on others. Geopotential height, air temperature, specific humidity and relative vorticity are the weather features used by the system, that most accurately predicts the location of fronts, although other features could be used successfully as well.

This project could outline a new, computer driven way of discovering fronts in weather data, based on known concepts from computer vision. However, the techniques are in need of more development and refinement to be able to compete with expert human analysis, and to be employed in full scale by the industry. These developments and refinements should, however, be achievable with today's technology, given adequate time and resources. Finally, the project raises the discussion of the need of an objective, absolute definition of fronts, based on common front indicators, to objectively and quantitatively evaluate and further improve front detection systems of all types.

Preface

This thesis was written in the spring of 2017, for the Department of Information Science and Media Studies at the University of Bergen, in collaboration with weather service provider StormGeo in Bergen, Norway.

I would like to thank my supervisor Bjørnar Tessem at the department for invaluable guidance and feedback along the way. I would also like to thank the entire team at StormGeo for enabling this great opportunity. This includes, but is not limited to: Torleif Markussen Lunde, for continued support and collaboration; Martin Grønnevet, for framing and approving the project; Frode Korneliussen, for expert advice and input and Kent Zehetner, for engaging and productive discussions along the way.

There are several others worthy of mention, whose feedback has helped shape this project and thesis into its final form. I would therefore like to express my gratitude to Kristoffer Ålhus and Mange Karlsen for their contributions.

In the words of C. J. Cherryh: "*It is perfectly okay to write garbage—as long as you edit brilliantly.*" The editing and refinement process of this project has been a tremendous group effort, and I am extremely grateful to everyone who helped in this endeavor, both technically, academically, structurally, and artistically. The following is the final result of this effort, and credit is due, not only to the author on the front cover, but to everyone who made it happen.

Table of Contents

List of Figures and Tables

1 Introduction

Numerical weather prediction is the computing of gridded data of weather parameters and how these parameters change over time. These gridded data are the basis for all weather information services found in different weather websites and applications. Weather data are usually presented as weather forecasts for specific locations or smaller areas, often as numerical weather data in graphs or tables, or converted to automated texts. Computing, and automated procedures in general, are doing the brunt of the work in all kinds of weather forecasting, and to an ever increasing extent as well (Pagano et al. 2016, Karstens et al. 2014, Lazos, Sproul and Kay 2015, Fan, Bell and Infield 2016).

In most aspects of weather forecasting, information systems can outperform the human meteorologists. Having computers taking over increasingly complex tasks is a continuous and unavoidable trend (Elkins 2015). Machine automation is saving businesses like StormGeo valuable work hours and large sums of money every year, but there are a few tasks where the computers are still matched by experts in meteorology. One of these tasks is front detection, i.e. the location and classification of the interaction zones between different air masses in a weather system. I want to show that this task can be, at least partially, automated by information systems. This will be done in order to save precious time and money for businesses concerned with weather and meteorology, to improve the quality of weather prediction, and to increase our knowledge about fronts, frontal behaviour and how we can use advanced information technology to detect and classify them.

In this thesis, I will give an overview of the fields of automatic weather prediction and artificial intelligence, and show how they overlap and contribute to this project. I will present the overall goals of the the project and discuss important relevant literature in the fields of weather data analysis, machine learning and computer vision. I will describe the methodological framework for the project from a technical, academic and business point of view. I will detail and explain my technical solution and, finally, I will present the results and findings of the project and discuss their implications and possible future work.

1.1 Fronts

Fronts are meteorological phenomena where two distinctly different air masses meet and interact, i.e. "*the transition zone between two air masses of different densities*" (Ahrens 1994, p. 322). In weather maps (figure 1), fronts are usually indicated by red or blue dotted lines. Fronts are considered vital to study in weather forecasting, as their attributes, velocity and direction greatly influence the weather on both a local and regional scale (Ahrens 1994, p. 322).

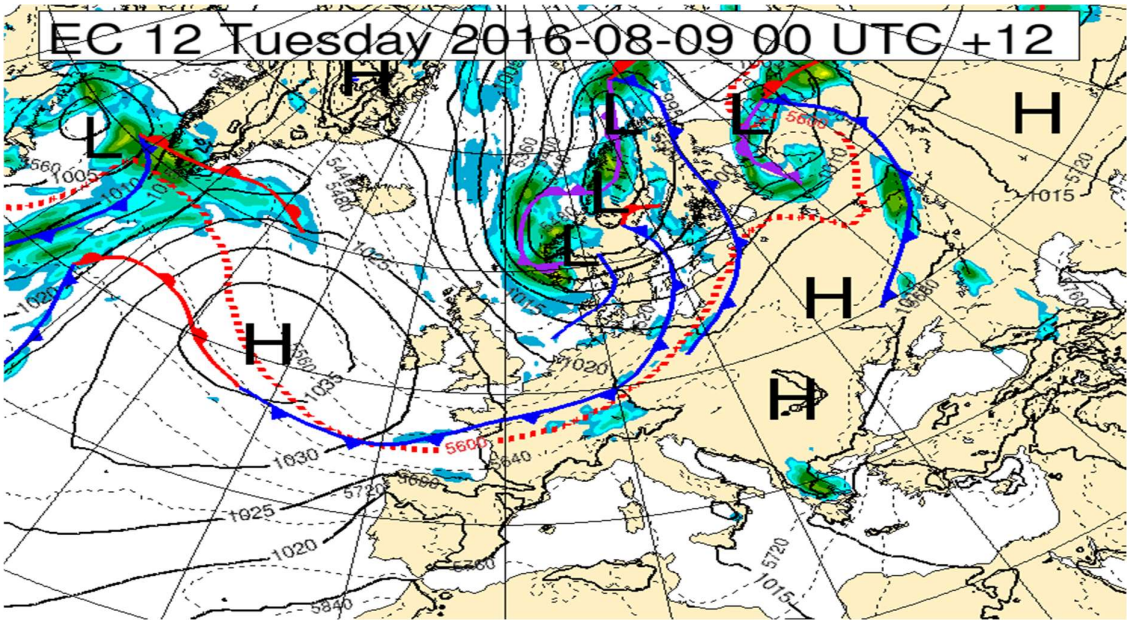

Figure 1: Weather map of Europe with fronts. StormGeo

There are four major types of fronts: Stationary, cold, warm and occluded fronts. The differences between these types rely largely on which air masses are moving. A cold air mass pushing warmer air up and away forms a cold front, while a warmer air mass overtaking a colder one forms a warm front. If there is little to no movement, we have a stationary front, and if one front catches up to another we get an occluded front or an occlusion (Ahrens 1994, p. 323). The different types of fronts have different manifestations and influence the weather in different ways, but they share the common definition cited above, and they can largely be identified in the same ways.

1.2 Front detection

For the most part, locating, identifying and classifying fronts is a task that is performed manually by meteorologists all over the world. At weather service provider StormGeo, new weather maps with fronts are drawn every 12 hours, and it is a complex and tedious task. In a time where data analysis is almost left entirely up to information systems, it seems strange that such a data driven task has not yet been, at least partially, automated.

One of the reasons front detection is still done manually is that it has proven difficult to formalize the task into definitive rules. Front detection is often context sensitive, and relies on the intuition of skilled meteorologists along with general heuristics. In general, a front will be found where there is a great change in air temperature, pressure and/or humidity over a short distance/time. The problem is, however, interpreting the minute details in a weather system. How great and how sudden must the change in weather state be to classify it as a front? Are two adjacent fronts separate entities, or are they part of one continuous front? Questions like these make it difficult, if not impossible, to design a set of definitive, exhaustive rules that will locate and classify fronts with satisfying accuracy. This task therefore, like many other entity or feature recognition tasks, requires more fine-grained and sophisticated techniques to yield satisfying results.

Another problem in this domain, that must be handled in some way by a front detection system, concerns input data. Which, and how many, weather variables meteorologists use when identifying fronts vary greatly, leading to further ambiguity regarding the nature of fronts and how to detect them. To investigate and develop front detection techniques further, it may be necessary to develop some theory on what weather features are essential for front detection, and which are not, based on current and potential future front detection strategies.

Further, there is a deeper, more fundamental problem about the task that makes it difficult to fully automate: A front is not a concrete, physical entity existing in the world. Rather, it is an abstraction and a simplification of a weather situation, which is used as a tool for visualizing and understanding the major patterns of air movement. A human meteorologist is aware of this context of front detection, and can make judgements about which abstractions are useful to make, and which are not, often regardless of the actual weather data present. This perspective is inherently difficult to “teach” a computer.

In the 21st century, attempts at formalizing and automating front detection tasks have started to gain some traction. Better tools and more computing power allow us to solve increasingly complex problems in reasonable time. Most of the work in the literature on this problem has been conducted using variations of edge detection techniques to detect fronts and frontal weather (Ullman and Cornillon 2000, Shaw and Vennel 2000, Hopkins et al. 2010). This direction has shown promise, and it is one that I will pursue as well.

1.3 Front detection as a computer vision problem

Computer vision is a field of artificial intelligence concerned with perception in computer systems. "*Perception provides agents with information about the world they inhabit by interpreting the response of sensors*" (Russel and Norvig 2014, p. 945). For a computer, "sensory input" is typically an image or video file, but it can, in principle, be any data representation of the world. A key problem in computer vision is edge detection. "*The goal of edge detection is to abstract away from the messy, multi-megabyte image and towards a more compact, abstract representation*" (Russel and Norvig 2014, p. 953). What this boils down to is detecting sharp and drastic changes in the visual data, either in terms of light, color or pattern. Edge detection is one of the oldest techniques in computer vision, but it is still vital for object and feature recognition in images and video.

The detection of fronts in weather systems is not immediately reducible to a traditional, image-based edge detection problem, for a number of reasons. Firstly, fronts are not concrete phenomena. They are an imposed abstraction on the natural world. Secondly, fronts are not visible per se. Where edge detection is about the visible differences in an environment, front detection must rely on hidden, less immediate data. Thirdly, fronts are detected through a number of different variables: Air temperature, pressure, wind speed, wind direction, precipitation, vorticity, geopotential height and others (Ahrens, 1994). Feeding a raw image of a weather map to a regular edge detection algorithm would therefore not do much good.

However, on a more fundamental level, detecting fronts is the same problem as detecting edges in an image. In both scenarios we rely on finding large and sudden changes in the properties of the world to find boundaries between distinct entities, either in an image or in a weather system. In an image, these changes are in the properties of the pixels, and the entities are the visual representations of actual objects. In front detection, the properties are the

weather features in different locations, and the entities are distinctly different air masses. Given the right variables to work with, a form of edge detection algorithm should therefore be able to at least detect and locate the most dominant fronts in a system.

If this claim turns out to hold true, it is important for a number of reasons: It allows us to work with weather analysis in a more universal, accessible manner, and it shows that computer vision techniques can be useful for more tasks involving spatial phenomena than just pure image analysis. Most importantly, it provides us with a way of finding patterns in weather data that is not reliant on previous observations. Where a conventional classifier in artificial intelligence needs examples to work on, edge detection only relies on the raw data, finding patterns as they unfold.

1.4 State of the art

Weather forecasting has been an important practical application for the use of artificial intelligence and automated data processing for decades (Bratko 1993, Lee and Liu 2004, Ghosh et al. 2011), but in some some subfields of AI, like feature extraction and entity recognition, it is rather underrepresented in the literature. These subfields could prove very useful for the important task of front detection, which is today performed largely manually. This lack of research is a shortcoming of the current state of the art, and one that should be amended to improve the quality and efficiency of automated weather forecasts, both from a technical and a human viewpoint.

2 Research Questions

Given the current state of the art of weather data analysis, machine learning and computer vision, I have tried to answer the following research questions:

1. *Is it possible to automatically locate and classify the fronts in a weather system?*

2. *What features of a weather system are critical in locating and classifying fronts?*

These two research questions, presented in order of importance, have some diverging foci. Question 1 is technically and practically oriented. By answering this, I attempt to improve the current state of the art and create tools that make weather forecasting easier and more accurate. Question 2 has a more theoretical, academic focus. In finding the most important weather features for front detection, I hope to improve and strengthen the knowledge in this field, and add to the groundwork for further study of the relationship between feature detection and weather data.

3 Project Description

This Master's thesis details an exploratory study in the fields of automated weather analysis and artificial intelligence. The goal is to show how weather forecasting can become more efficient and more accurate through the use of a new information system. This system employs advanced techniques in artificial intelligence and computer vision, as well as expert knowledge in meteorology, to automate and improve the crucial task of front detection. In this project, I have been more concerned with showcasing what is possible with our current knowledge and technology, rather than explaining what has been done before and what is currently being done.

Through the development of such an information system, I explore whether or not front detection can be done with more sophistication than simple rule based data analysis. Is it possible to view front detection as an entity recognition problem, or an edge detection problem, as these are defined in the field of computer vision? Is it possible for a computer system to detect fronts in the same way humans do: By looking at a spatial representation of the weather data, and identifying the important entities in the weather system and how they interact with each other?

To answer the research questions of this thesis, I have therefore developed such a weather analysis system, in cooperation with weather service provider StormGeo. This system uses detailed weather maps and historic data about front detection to identify and classify the fronts in a weather map. In brief, the target of the project is to be able to describe "*Where are the fronts in this weather map and what types of fronts are they? What features in the weather data were crucial to identifying and classifying these fronts?*" To accomplish the former, I have utilized computer vision techniques to visually distinguish front-like features, and machine learning techniques to learn the characteristics of different front types. To accomplish the latter, I have analyzed different weather variables and how these influence the front detection process. This analysis has yielded a set of variables that the system uses for front detection. The finished system has been tested against the analyses of professional meteorologists, and the results of this development and testing could hopefully inform and guide a future improvement of automated weather forecast systems.

4 Literature Review

Here I will investigate the current state of the art in more detail. I will analyze both classic and recent scientific publications involving all four main aspects of my research domain: Fronts and frontal weather, gathering and analysis of weather data, machine learning techniques, and edge detection in spatial domains. I will present the findings of the most relevant publications, discuss their merits and shortcomings, and explain where my research project fits into the current state of the art.

4.1 Weather data analysis

The history of weather data analysis outdates modern computing. Bjerknes (1904) is often credited with starting the modern school of meteorology, also known as the Bergen school, where numerical data analysis plays a large role. Today, computer systems underlies almost all weather prediction, and research in the field usually involves data analysis and computing. Because of this, developments in weather technology is tightly linked with research in information science and computer science.

Consequently, extensive research has been conducted in the computer and information sciences about analysis and application of weather data. Most of the work has been done in the cross-sections of information/computer science and other fields, especially geosciences, industrial processing and agriculture, and a large portion of the research has been dedicated to weather forecasting. This is natural, as forecasting is perhaps the key challenge in meteorology.

A typical example of this is Ghosh et al. (2011), who present a back-propagation neural network for weather prediction. They find that a "*Back Propagation Network and Hopfield Network based approach for weather forecasting is capable of yielding good results and can be considered as an alternative to traditional meteorological approaches.*" The research is based on classical weather prediction techniques, that are mostly concerned with the *temporal* dimension of weather data, whereas my thesis is mostly concerned with the *spatial* nature of weather systems, in a single point in time. However, one of the issues the article handles is the extraction of useful or interesting information from vast amounts of data, a technique known

as data mining. This aspect is highly relevant to my project, and provides important background knowledge for performing data mining on weather data.

Another example is Váscák et al (2015), who discuss a local weather prediction system for an industrial heating plant. This study is of particular interest as the researchers have developed a complex neural network that takes weather data input from many different sources, and it also employs several AI techniques. The study finds that neural networks can be useful in classifying and predicting weather, and most interestingly that multiple spatially-separated data gathering points can be helpful in weather prediction. The results, however, are quite industry specific and not particularly generalizable. The study is for instance not concerned with the representation of weather systems, but merely an input/output description of weather.

Lee et al. (2015) explores weather data in a different domain, and have developed a system to predict crop yields based on soil and weather data. Their system shows that systematic analysis of weather data can be useful for making predictions at geographically large scales. This is interesting, as it explores weather systems in a spatial domain, in addition to the temporal domain. However, the study is most relevant for its agricultural implications and not the gathering and analysis of weather data. Where its methodology is highly relevant, its application area is at best tangent to the one I am investigating.

The research of de Lima and Stephany (2013) is perhaps the most relevant to my project in the recently published literature. They propose a new approach for early detection of storm centers and extreme weather, using data spanning both the spatial and temporal dimensions. Their novel clustering algorithm has been successful in detecting emerging storm centers in Brazil. This study shows that artificial intelligence can be successfully employed to detect and classify entities in a weather system. This is good news for my project, as it shows that others have been successful in analysing spatial weather data, and using the acquired information for a practical, predictive purpose. Since this is, to a large extent, the same task that I am trying to perform, it is a sign that my research is in a promising direction; In a field that is currently being explored, and with techniques that are proving relevant and useful.

The work of Hoskins and Hodges (2002) is another highly relevant study in weather analysis, albeit a bit older. Kevin Hodges is a leading figure within feature detection in meteorology, and this study uses novel techniques to identify storm centres in the northern hemisphere.

Like front detection, this a typical feature detection task, and many of the perspectives presented in this paper are relevant for all kinds of feature detection, including front detection.

4.2 Fronts and frontal weather

Fronts and their effects on the weather have been thoroughly studied since the beginning of modern meteorology in the early 20th century. Bjerknes and Solberg (1922) first describe what has later been dubbed the "Norwegian cyclone model", which outlines the major movements of large air masses and how these movements manifest as fronts and frontal weather. Most of the general principles of the formation and evolution of weather systems presented in this paper are still accepted and used today. The fronts described by Bjerknes and Solberg are, principally, the same as the fronts I am working with in this project.

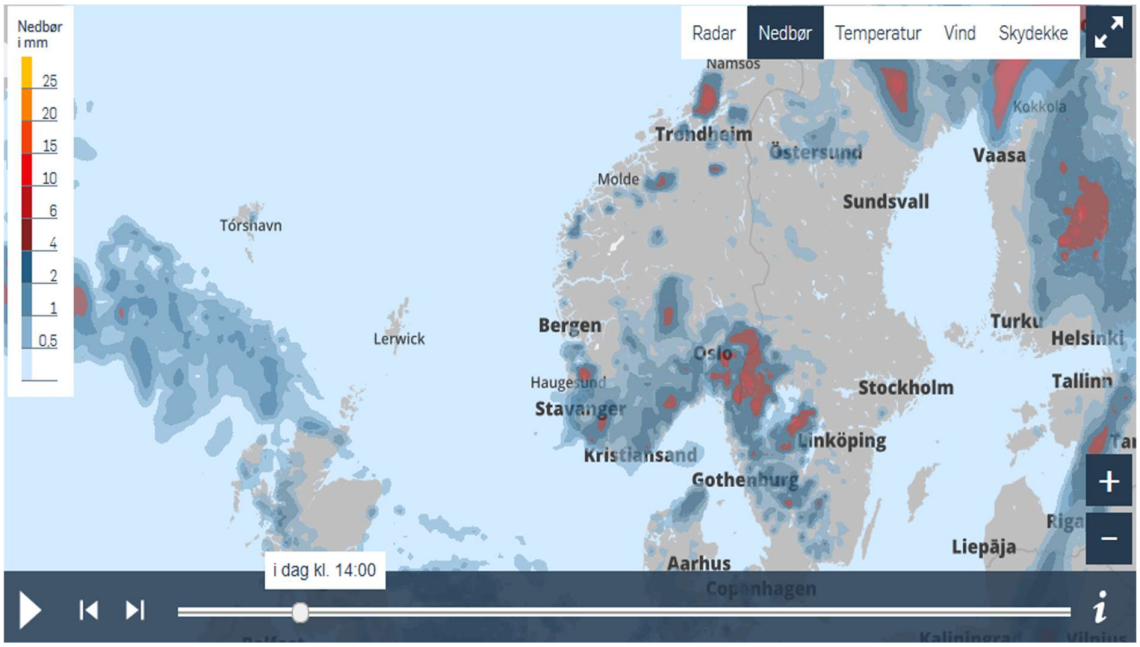

Figure 2: Short term precipitation prediction for Scandinavia and the North Sea (http://www.storm.no/)

Most of the research into fronts have been concerned with the consequences of fronts, rather than the fronts themselves. Browning et al. (1982) use distributed weather radar measurements to make quantitative short term predictions of frontal rain for small geographical areas. They find at the time that the predictions based on radar measurements were inaccurate and unsatisfactory, largely due to the technical limitations of the measurements. The study also emphasizes the importance of context-sensitive systems in weather prediction, as naïve judgements were accountable for about one quarter of the false

predictions from the system. Today, these "precipitation radars" are both more accurate and more detailed, and they are common tools for both weather prediction and presentation (figure 2).

Wilby (1995) finds that analysing the frontal situation and the likely "next weather type" using Lamb Weather Types (Lamb 1972), improved the accuracy of precipitation forecasts noticeably. This shows that reliable front detection, combined with knowledge of the movement and evolution of fronts and air masses, both in general and for local areas, can be highly advantageous to achieve higher accuracy weather forecasting.

Ullman and Cornillon (2000) present a study which bears a lot of similarities with mine. They use edge detection techniques to locate fronts using sea surface temperature readings from Advanced Very High Resolution Radiometer (AVHRR), and compare the results to human classification based on on-site measurements. They find that although the combination of remote observation and automatic classification have a slight negative impact on the accuracy of the classification, "*frontal climatologies developed from the application of automated edge-detection methods to long time series of AVHRR images provide acceptably accurate statistics on front occurrence.*" This is promising for my thesis, as I hope to show that previous human observations can not only be used as a yardstick to assess the quality of the automatic system, but also as a means to improve its performance.

Shaw and Vennel (2000) present an algorithm for detecting and "following" fronts over time, albeit fronts of a different nature than my study. Their algorithm detects oceanic fronts, i.e. sharp and sudden changes in the characteristics of seawater (in terms of temperature, salinity and other variables), and is showing remarkably strong results. Although designed for a different medium, the algorithm presented here, and further improved by Hopkins et al. (2010), shares striking similarities with the solution implemented in my project, most importantly being based largely on edge detection techniques.

4.3 Machine learning

"*An agent is learning if it improves its performance on future tasks after making observations about the world*" (Russel and Norvig 2014, p. 704). The field of machine learning is concerned with building systems that improve their own performance by analyzing their inputs and outputs. Machine learning is a general term that can be applied to many different

domains and techniques, including decision trees, linear classifiers, artificial neural networks and support vector machines. Machine learning is a cornerstone of both my research and my artefact, since "*making observations about the world*" and continually "*improving its performance*" are key elements in my implementation of a front detection system.

Bratko (*1993*) explores the usage of machine learning in artificial intelligence. Here, machine learning is classified into two distinct modes: *Learning by being told* and *learning by discovery*. This is commonly defined as supervised and unsupervised learning. In the field of artificial intelligence, supervised learning is the most explored, and it is also the most widely applied technique, used commonly in for instance medical diagnostics and, coincidentally, weather prediction. Supervised learning is also the most relevant for my research, as, realistically, some innate knowledge about weather is necessary to start drawing conclusions from weather data. Knowledge about the relationships between weather types, topography and geography would be extremely difficult to obtain in unsupervised learning.

In his classic paper, Bratko also accounts some of the problems with learning from examples (supervised learning), such as the impossibility of complete knowledge. This is still an important challenge today, and also a problem my system needs to handle. This is why I have developed a system where the front *identification* is performed without explicit learning, while front *classification* uses supervised learning based on expert judgement.

Lee and Liu (2004) introduce iJADE, an intelligent multi-agent platform, useful for all kinds of classification and decision making problems. They also display its usefulness with iJADE WeatherMAN, a weather forecasting system based on a multi-agent neural network. They also show that WeatherMAN is better at weather prediction than forecasts based on single station observations. These results are interesting, since they show machine learning techniques successfully applied to a meteorological problem. Further, they point to the usefulness of analyzing weather data from a larger geographical domain. It is interesting to note that the case for spatially distributed data gathering in weather prediction has been relevant in AI for a long time already. However, given the rapid rate of innovation in computer science, the techniques used in this study 12 years ago are to a large extent considered inadequate by today's standards. This study is therefore useful for obtaining an overview of the field, but its technical implementation is not likely to be helpful over a decade after its publication.

Xu et al. (2016) have conducted an interesting study in the field of medical informatics. They explore the usage of convolutional neural networks in diagnosing cancer patients based on histological images. Their study finds that these types of neural networks are very useful for feature recognition and feature extraction from images. This is interesting, as a large part of my research project revolves around this type of task. Of course, the paper is written in a completely different application of information science, but its technical relevance should be considered high. If convolutional neural networks can detect well defined patterns in the body's cellular structure using images, there is reason to believe the same task could be performed on structural weather data.

An important part of any classification algorithm in machine learning is outlier detection. Outliers are data points that vary drastically from the mean, and in small sample sizes they could distort the results dramatically if they are not detected and handled. Rahmani et al (2014) present some interesting ideas on this topic. They show how outlier detection can be improved using a graph-based, "sliding window" approach, similar to how convolutional neural networks work. The paper also emphasises the importance of good outlier detection in, among other fields, weather data analysis. These findings are interesting, since they give good insight to an important and common source of error in weather analysis, and also provide a means of minimizing that source of error. Outlier detection is an important factor in the success of my system, and this study is helpful for understanding and handling the issue.

4.4 Edge detection in spatial data

Davis (1975) provides a good theoretical background for the problem of edge detection, and its different classifications and techniques. "*In a grey-level picture containing homogeneous (i.e., untextured) objects, an edge is the boundary between two regions of different constant grey level.*" The concept of edge detection as distinguishing the boundaries between two different regions is the definition that will be used throughout this thesis. An example of edge detection can be seen in figure 3 below.

Figure 3: Canny edge detection applied to a photograph (MacLoone 2010).

Davis describes three main types of edges. These are steps, roofs and spikes, referring to the general shape of the gradients around the edge. The front detection problem should be considered a form of step detection. A step is a single, sudden change between two regions, usually with a beginning and an end. Depending on your resolution, a front can be considered have only one edge in total, or one in either end of the region of change. In this project, I am working at such large scales that considering fronts as a single edge should prove sufficiently accurate.

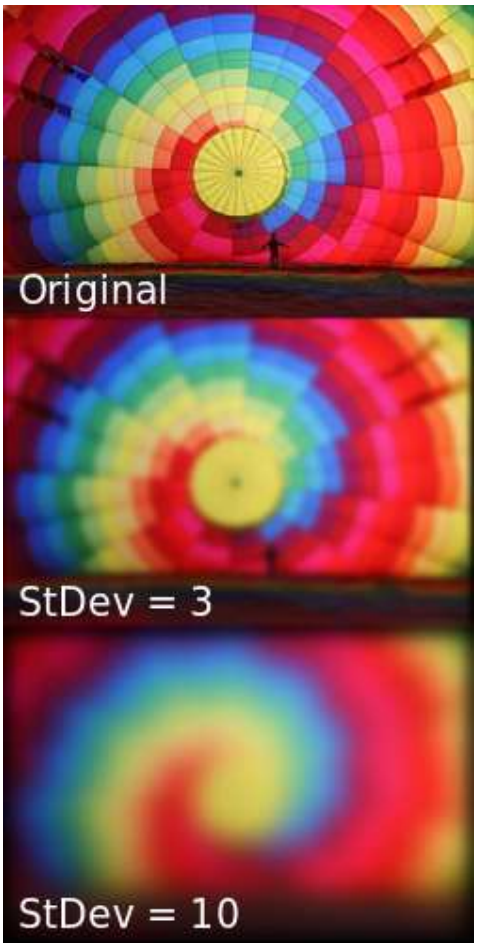

Figure 4: Example of gaussian blur applied to an image with different kernel sizes (IkamusumeFan 2015).

Further, Davis also outlines the main challenges in real-world edge detection. Some of these are image specific, like blurring and de-focus, while others are more general, like image resolution and quality, as well as irregularity and heterogeneity of the objects represented in the image. Both of these general problems are important for front detection as well. Firstly, weather systems are highly erratic and rarely follow completely predictable patterns, and secondly, the resolution at which we analyze them can be very influential on how well the edge detection will work. Too small resolution will make it difficult to find complete, smooth edges, while too large resolution could make it difficult to distinguish the fronts from the noise in the data.

An important step in edge detection is smoothing (figure 4). This is a process in which the data is blurred to remove insignificant noise. This is

traditionally done with a Gaussian blur, but Perona and Malik (1990) present a more sophisticated method, called *anisotropic diffusion*, that does not smooth uniformly, but rather "*encourage*(s) *intraregion smoothing rather than interregion smoothing*", by adapting the kernel size based on the image context. This can be especially useful when the edges in a domain are diffuse and difficult to detect, as they can be in the front detection scenario. Catté et al. (1992) expand and improve this method by introducing *nonlinear diffusion*, which avoids many of the problems with noise in the original publication.

4.5 Conclusions

In the field of weather data analysis, a lot of work has been done in recent years, but the vast majority of this is dedicated to weather prediction, i.e. weather analysis in the temporal dimension. Only a few publications touch on the most important aspect of my research, namely analysis in the spatial dimensions. However, those that do explore weather analysis in the spatial domain are highly relevant for my project, and there are other studies in the literature that provide useful background knowledge as well. Overall, the concept of spatial weather prediction is somewhat explored, but rarely in the manner and the scale at which I will be working.

Fronts and frontal weather have been an important part of weather prediction and forecasting for almost a century. Most of the research in this domain has been either conceptual; trying to accurately model and understand fronts and frontal models, or predictive; trying to use frontal information to predict weather, particularly in terms of precipitation. The exceptions are a few fairly recent studies, which, like this one, are concerned with front detection and classification, using modern AI techniques.

In the field of machine learning, the concepts of feature extraction, supervised learning, classification and outlier detection are well documented and well known. Neural networks, and especially convolutional neural networks, are interesting techniques, as they has proved useful in feature extraction from images. I have through this review discovered the most important techniques that I have used in my research, and further established the link between machine learning and weather prediction systems.

In edge detection the important factors, techniques and challenges are well defined in the literature. Technical improvements in smoothing, along with generally improved computing

power have made edge detection algorithms increasingly more powerful. We have also seen how edge detection algorithms were able to successfully detect warm and cold fronts based on satellite images, as early as 15 years ago (Ullmann and Cornillon, 2000). This is encouraging news, since it means I am working with well known, well documented techniques that have already been proven to excel in the project domain.

This literature review has established that all of the techniques I use in my project are well known and well documented, but the application domain is to some extent unexplored. The most important factor for this project is therefore how these AI techniques can be used to solve a novel problem, rather than the usage of the techniques themselves. Further, the review has shown how my study fits into the greater ecology of research in the cross-section of automatic data analysis and weather prediction. It shows that although my study is novel and explores some under-researched topics, it should still have a natural place in the current state of the art.

5 Methods

This section will outline the methodological foundation of the project, from both a scientific and a practical, applied viewpoint. I will discuss the merits of system design as a research discipline, I will describe the software development methods used in the project, and I will define the test and evaluation criteria for the finished artefact.

5.1 Design Science as a research method

Hevner and Chatterjee (2010) provide a good overview of the status of Design Science Research (DSR) in the field of Information Systems (IS). It is described as follows: "*It seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished.*" This project follows the DSR paradigm, in that the main goal of the project is to extend the knowledge of what is possible to do with weather data, by designing and creating an artefact that showcases these possibilities.

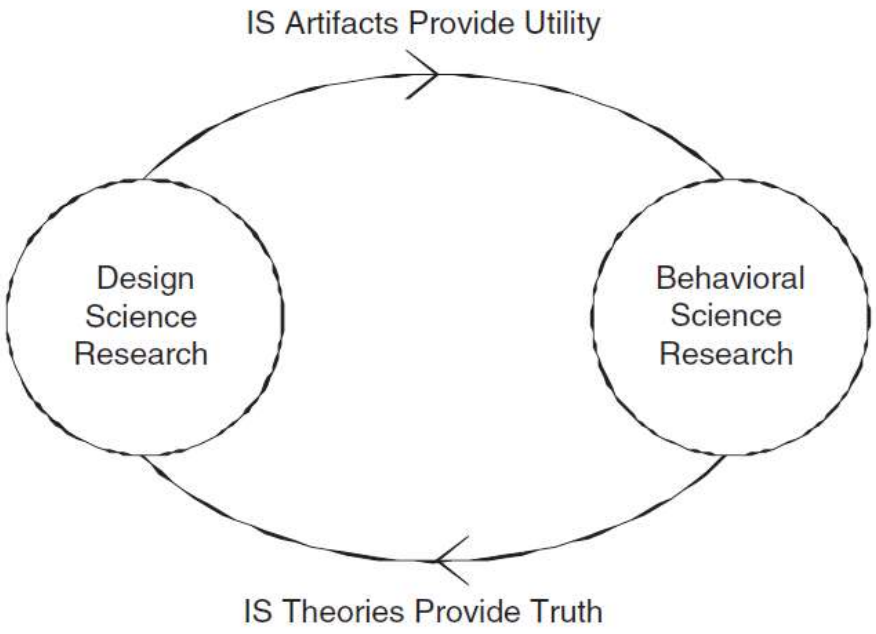

Figure 5: Representation of relationship between BSR and DSR (Hevner and Chatterjee 2010).

DSR is typically described in contrast to Behavioral Science Research (BSR) (figure 5 above), where DSR provides new insights and tests new concepts, and BSR serves to ground

new findings rigorously in the existing knowledge base. DSR can therefore be seen as the exploratory side of IS research, and BSR as the confirming. Hevner and Chatterjee argues that "*...the practical relevance of the research result should be valued equally with the rigor of the research performed to achieve the result.*" As stated in the introduction, my study aims to extend the understanding of what is possible to achieve in automatic front detection, and as such, it fits nicely in the definition for DSR provided by Hevner and Chatterjee.

The book by Hevner and Chatterjee draws heavily on Hevner et al. (2004), who aim to "*provide an understanding of how to conduct, evaluate, and present design science research*". Their often cited paper presents seven concrete guidelines on how to perform DSR. Following these helps to ground the research project in a widely accepted scientific framework. Throughout this project I have therefore aimed to work within these guidelines as follows:

- **Design as an Artifact**: The research project should produce a working information system that detects fronts in weather systems. This system should be runnable and usable, and function as a showcase for the developed technology.

- **Problem relevance:** The project should handle the problem of identifying and classifying fronts in weather data; previously largely unexplored territory. It should explore a frontier of automatic weather analysis, where human meteorologists still outperform computers.

- **Design evaluation:** The quality of the system will be measured by the quality of its output data, which will be tested against the judgements of expert meteorologists. The most important feature of the system will be its core functionality and output, and it is the quality of this functionality and output that will be at the core of the evaluation as well.

- **Research contributions:** The project will contribute with knowledge about the research domain; classification of weather data. It will hopefully provide new insights into the relationship between computer vision and weather analysis, as well as showcase new and unexplored possibilities for automatic weather analysis.

- **Research rigor:** The artefact will not be reliant on user interaction, and the output data is the most important result of the project. The quality of the results can therefore

be tested with well documented mathematical and statistical methods, as well as by experts in the field of meteorology.

- **Design as a search process:** The means of designing a good system will mostly arrive from AI techniques and data management, as well as expert input. The laws of the domain are exclusively defined by the nature of weather data and the limits of modern computing. The outer bounds of the research area should therefore be considered well-defined, and the design process will take place within these boundaries.

- **Communication of research:** The end product will be presented effectively to both the research community and the field of meteorology as a Master's thesis and subsequent thesis presentation.

5.2 System development methodology

5.2.1 Theoretical framework

In order to produce a well-functioning artefact, the project has been guided by software development methods commonly employed in the fields of IS research and development. My motivation for using the particular set of methods to be described, was the desire for a lightweight methodological framework that would not create unnecessary complications or bureaucracy in the development process. At the same time, the methodology needed to provide at least a bare minimum of control of workflow and collaboration between stakeholders.

With these requirements in mind, I decided on a development methodology in the family of Agile methodologies. Agile (Beck et al. 2001) methodologies emphasize the importance of customer collaboration and adaptability to rapidly changing requirements. Both of these factors were considered important to this project. Firstly, I have been working closely with the meteorological company StormGeo, on-site at their headquarters in Bergen. I was dependent on StormGeo for both acquisition of data and evaluation of the artefact. Maintaining frequent and productive contact was therefore considered important. Secondly, since the application domain is largely unexplored, the likelihood of significant changes in the requirements and specifications of the product along the way, was considered quite high at the onset. It was

therefore important to use a methodology that facilitates easy management of such rapid changes.

In addition to the focus on agile development, I also employed concepts from lean software development. Lean methodologies focus on low-waste, highly efficient and effective processes that utilizes the available resources in the best way possible. Waste is, in this setting, any activity that does not contribute to the production of good software. Another important construct in lean is the idea of a Minimum Viable Product (Samarchyan 2014), i.e. the simplest possible solution to a problem. This is of course linked to reducing waste, only developing what is necessary at each step of the way. I believed developing in a lean framework to be helpful in this project for several reasons. Firstly, because the timeframe was fairly limited, it was important to be able to implement the functionality fast. Secondly, since the application domain and the potential problems were to some extent unknown, the ability to stay on track and develop only the most essential functionality was also considered important.

Working with agile and lean is of course not the only possible solution for a project like this, and I could have utilized both more and less controlling methodologies with success. However, the combination of the lightness of lean principles and the active stakeholder engagement of agile seemed to match the predefined requirements rather well. Hoping to get the most out of this combination of lean and agile, I decided to use concepts from two different development methodologies: The agile methodology Scrum (Schwaber and Sutherland 2013) and the lean methodology Kanban (Peterson 2015).

From Scrum I used the concept of sprints: Short timeboxed events in which a predefined amount of system requirements are implemented. I worked with development in two-week iterations, using reviews and retrospectives with collaborators to continually optimize the development throughout the project. Further, I represented system requirements as user stories (Cohn 2004), as this is a well-tested, industry standard method for defining requirements for an information system. User stories define functionality in terms of what the users or owner of the system want to achieve with the system. "*As a user I want to see a list of the most important features in the weather map, so that I can get a better sense of what's going on"* is an example of a typical user story I could have used to keep track of progress. Having user stories were intended to divide the scope of the project into manageable pieces.

From Kanban I employed the concept of visualizing workflow, by having a visual representation of the project status available at all times (figure 6), and updating this continuously. I also focused on the idea of minimizing waste, i.e. removing or limiting all processes that were ineffective and slowing progress, among other things through the use of sprint retrospectives from Scrum.

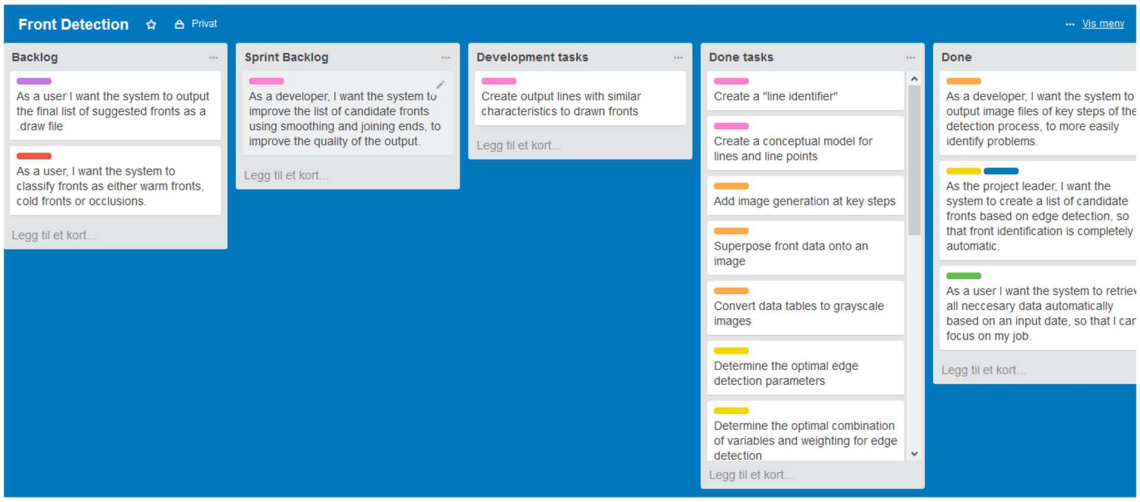

Figure 6: Visualization in Trello (2016) of the development methodology implemented.

All of these methodological concepts were chosen to optimize workflow and ensure a finished product that was within both the scope and the timeframe that was initially defined. The complete set of practices used can be compared to a form of ScrumBan (Nikitina, Kajko-Mattsson and Stråle 2012) that aims to utilize the best practices from both a lean and an agile development perspective. Figure 6 shows a snapshot of the visualization of the workflow from the fifth sprint (October 17th 2016), with user stories and connected tasks in different stages of development.

5.2.2 Practical application

The first development period was defined from August 15th to November 20th, and divided into seven sprints. During this development time, the following user stories were implemented:

- As a user I want the system to retrieve all necessary data automatically, based on an input date, so that I can focus on my job.

- As the project leader, I want the system to create a list of candidate fronts based on edge detection, so that front identification is completely automatic.
- As a developer, I want the system to output image files of key steps of the detection process, to more easily identify problems.
- As a developer, I want the system to improve the list of candidate fronts using smoothing and joining ends, to improve the quality of the output.
- As a user, I want the system to classify fronts as either warm fronts, cold fronts or occlusions.
- As a user I want the system to output the final list of suggested fronts as a text file.

For each sprint, one or more user stories were taken into the Sprint Backlog and divided into meaningful engineering tasks. The tasks were moved into the "done" column as they were being completed, while the user stories remained in the backlog until all their development tasks were finished. When all user stories were moved into "done", the first development period was considered completed.

The second development period was defined from December 5th to February 3rd. During this period, no new functionality was implemented. The focus of this period was on improving the performance of the existing system. Because of this, no new user stories were defined, and the period was not divided into sprints. Instead, the backlog was populated directly with development tasks, which were implemented or discarded sequentially. This can be considered a more pure form of Kanban, with no iterative separation of work. I used a maximum workload of one task in development, meaning that a task had to be completed or discarded before a new one could be started. When all development tasks in the backlog were completed or discarded, the second development period was considered completed.

5.3 Data acquisition and evaluation

As previously noted, the data required for the project was provided by expert meteorologists and data scientists at StormGeo. This data and expert knowledge was also used for the evaluation of the finished artefact.

5.3.1 Data acquisition

The system uses several different types of data. Firstly, it uses large scale weather data files for the Northern Atlantic Ocean and Europe. These files contain information about air pressure, humidity, wind speeds and direction, as well as temperature. This serves as the basis on which the system detects and classifies fronts. Furthermore, the system uses a data set of manually drawn fronts from StormGeo's archives. This serves as the training set for the system, allowing it to build a knowledge base of important contextual cues on which to improve its front classification. Both of these sets of weather data files were provided directly by StormGeo, from their archives.

5.3.2 Evaluation

The primary evaluation process was purely based on the output of the system. The system detects fronts for a time and a place where meteorologists at StormGeo have previously manually drawn front lines. This yields data that is qualitatively comparable, with the help of experts in the field. Using statistical methods to meaningfully quantify the differences between manually and automatically drawn fronts could have been useful, but this has not been done. Quantitatively analyzing fronts is inherently hard to do because of the fuzzy and somewhat undefined nature of the domain. This problem is discussed further under 8.3.

The evaluation of the output data has therefore exclusively been performed by four meteorologists at StormGeo, both individually and in group conversations. The evaluation is based on the output files of the system in the time period from September 5th 2016 to January 12th 2017. The meteorologists have, both in their own time and in semi-structured interviews, compared the output of the system with the manually drawn fronts for the same time. From from January 9th to March 3rd 2017, 56 observations were gathered in a result bank that forms the foundation for answering the first research question. Some examples of observations (in Norwegian) can be seen in table 1, while the full result set can be found in Appendix A. The results of the evaluation are presented and discussed in 7.1 and 8.1.1.

Dato/tid	Tema	Kommentar
6/9-2016 00:00	Identifisering	Systemet finner en front nord-sør i midten av bildet. Ikke tegnet opp av meteorolog.
7/9-2016 12:00	Identifisering	Okklusjon i nord, ikke mulig å se i datasett. Veldig like temperaturer. Nedbør nyttig for å finne denne.
6/10-2016 12:00	Identifisering	Små forskjeller, veldig få fronter tegnet.
24/10-2016 00:00	Identifisering	"Hull" i okklusjon.
24/10-2016 12:00	Identifisering	Nesten funnet et perfekt klassisk system i vest.
7/9-2016 00:00	Identifisering	Flere parallelle fronter. Kun en tegnet opp av meteorolog.
10/1-2017 12:00	Identifisering	Okkludert front over Østlandet omtrent samme plassering av meteorolog og automatisk analyse.
10/1-2017 00:00	Klassifisering	Varmfront mangler generelt.
30/11-2016 12:00	Klassifisering	Kaldfronter blir klassifisert riktig. Ellers mye rart.

Table 1: Example of records from the result bank.

Another important evaluation criteria was the overall functionality of the finished system. The primary goal of the project was to develop an artefact that classifies fronts consistently, and its relative success is determined by the fulfillment of the requirements for the system. These requirements were represented as a backlog of user stories (Cohn 2014, Schwaber and Sutherland 2013), previously detailed under 5.2. The evaluation of the system in terms of the fulfillment of user stories is presented in 7.3.

5.4 Tools and techniques

I have used several different tools during this project, for both technical and organizational purposes. These will now be listed and discussed. I will briefly describe why I have chosen these particular tools and techniques and, where applicable, discuss advantages and tradeoffs compared to alternative available solutions.

5.4.1 Canny edge detector

Canny (1986) introduced what is today one of the most widely used techniques for linear edge detection, Canny edge detection. This is a multi-stage algorithm that detects edges or boundaries in an image, and returns a simplified, binarized image where only the edges are marked. The algorithm can be summarized as follows:

1. **Blurring/smoothing**: Apply a Gaussian filter to the image in order to reduce noise. The Gaussian filter will blur all data points with its closest points, and results in a more homogenous image.

2. **Find the gradients and their direction**: These are the values that will determine if the algorithm will find an edge at a given point. The gradient describes how quickly the pixel values in the image changes, and in which direction, for any given point.

3. **Non-maximum suppression**: This step serves to "thin" the edges, and keep only the strongest gradient for any given point on an edge. This insures that all discovered edges will have a thickness of one pixel.

4. **Double thresholding**: All discovered edge points are now compared with two thresholds, a high and a low threshold. These threshold values determine how many potential edge points will be included in the final edges. Pixels with a gradient above the high threshold are considered strong edge points, while pixels with a gradient between the two thresholds are considered weak edge points. All other edge points are now discarded.

5. **Edge tracking by hysteresis**: Here all strong edge points, as well as weak edge points directly connected to at least one strong edge point, are collected for the final selection of edge points. These are finally imposed on the original image.

In this project, I have used a Java implementation of a Canny edge detector, made by Tom Gibara (2011). I have modified it to work on raw tables of integers rather than image representations. This edge detector allows the system to discover large and sudden changes in the values of different atmospheric variables, typically discovering the location of a front. The source code for this edge detector is found in Appendix B.

Although the Canny edge detector is a commonly employed edge detection technique, it is not without fault. Ding and Goshtasby (2001) highlights perhaps its biggest problem: The inability to consistently detect edges at cross-sections between more than two regions. Canny edge detection does not handle branching edges all too well. This is mostly due to its relatively basic calculation of gradients.

Ding and Goshtasby present a more sophisticated gradient detection which is better at finding branching edges and leaves fewer erroneous gaps in the output. There are also other, even more sophisticated methods of discovering edge points. A Laplacian edge detector (Davies 2005, p. 149) uses the second derivative of the gradient (rather than the first derivative) to find the sharpest and most distinct rates of change in the image.

Given the scope of this project, however, a more sophisticated gradient analysis was deemed an unnecessary complication, given the focus on creating a minimum viable product, and the Canny edge detector proved to be sufficiently precise to meet the goals of the project. The problems with gaps and branching edges was mediated by other means, such as joining line ends that are close together and removing parallel lines. More on this can be found under 6.4.

5.4.2 Neural network

Neural networks in artificial intelligence are designed to learn causal relationships in a system, in a manner similar to how neurological pathways in the human brain work (Russel and Norvig 2014, p. 739). A network always consists of a set of input nodes and a set of output nodes, as well as any number of hidden, intermediate nodes in between. The nodes are connected through weighted links, and these links “learn” the relationship between different inputs and outputs.

I have used a neural network to classify candidate front into different front types. This classification could be done with a multitude of different AI techniques, as it is a classic function learning problem, where the task is to learn the correct output (a front type) based on an input (the weather situation in a point). A neural network was chosen, as it is a well-known, well-tested technique that could succeed in delivering a sufficiently accurate classification.

There are three different freely available neural network implementations for Java. These are Encog (Heaton 2016), Java Object Oriented Neural Engine (Marrone 2004) and Neuroph (Sevarac 2016). CodeProject user taheretaheri (2010) has made a very thorough and well documented comparison of the three tools, and concludes that:

> "*the clear winner is Encog. It provides a clean and easy to use API and stunning performance. The performance of Encog currently cannot be matched."*

After making an implementation of both Encog and Neuroph, I also found Encog both faster and easier to use and modify. Consequently, I have used an Encog neural network to classify fronts into different types, based on a training set of manually drawn fronts by meteorologists. The source code for the neural network implementation can be found in Appendix C.

5.4.3 Software development tools

The Eclipse Foundation has one of the most widely used development environments for Java, the Eclipse Java IDE (The Eclipse Foundation 2016). I used the Eclipse IDE to develop the front detection system. Eclipse was chosen because of its general utility, available support and previous experience with the tool.

Trello (2016) is a free project organization tool. I used Trello in my development process to visualize workflow and organize the scrum sprints in a simple and accessible manner.

5.4.4 Auxiliary tools

In addition to the machine learning and computer vision techniques, I have used a few external libraries for handling and retrieving files. The weather data files provided by StormGeo is on the NetCDF format, a format commonly "*used in atmospheric research, GIS, and related fields.*" (Wolfram 2008). I have used the Java library NetCDF Java by Unidata (2016) to interpret the contents of these files.

Further, I have used Apache commons-net 3.5 (Apache 2016) to retrieve both weather data files and the training set of drawn fronts from StormGeo's local databases into the system.

6 The System

The front detection system is a standalone Java application that handles weather data for the North Atlantic for a given date and time, and provides a suggested set of fronts for that particular point in time. Theoretically, we can consider the system a function F, that takes an input of a particular point in time, and outputs a text file of a list of fronts for the North Atlantic at this time step. Throughout this section, I will refer to this entire transformation process as F.

F can be divided into six meaningful sub functions: G, H, I, J, K and L that can be described as follows:

- G: Data retrieval.
- H: Data normalization and transformation.
- I: Edge detection.
- J: Line identification.
- K: Front classification.
- L: Data generation.

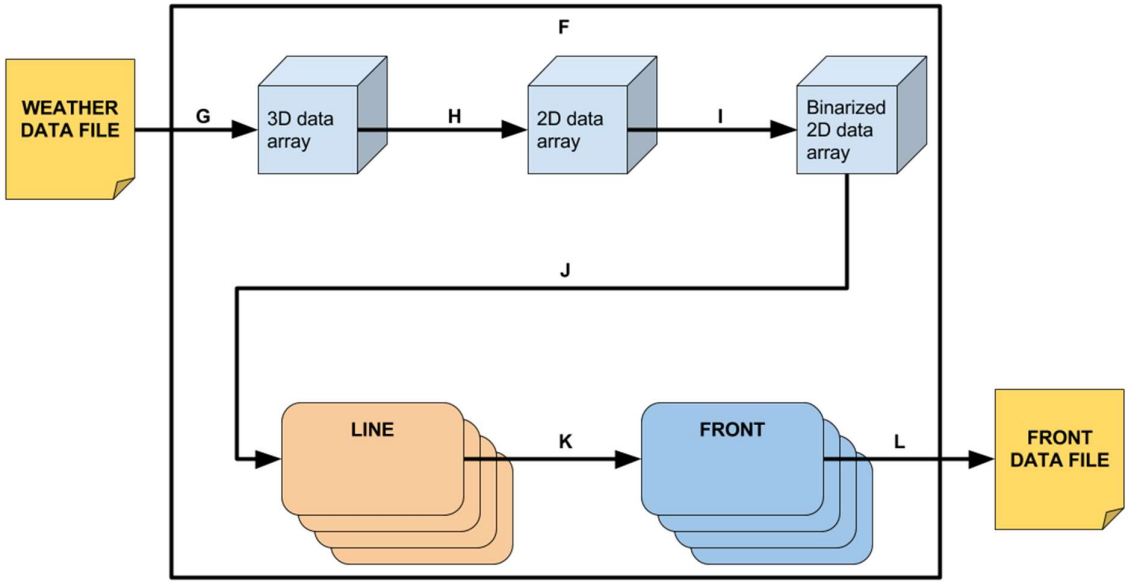

Figure 7: Visualization of the program flow, from input file to output file.

I will now describe each of these sub functions in detail. I will explain the algorithms employed and the motivation for their use. Where applicable, I will discuss alternative solutions and their merits and shortcomings. This section will in total be a complete

description of what the system *does*, from data input to data output. Figure 7 gives a graphical overview of the different sub functions G → L.

The input data for the system comes from a weather model. This model uses the weather situation for a point in time to generate predictions for the future weather states. The weather model is run every 12 hours, and generate predictions with 12 hour increments. F has been designed and tested for time increments between 12 and 120 hours. This is because these are the same time steps that the meteorologists currently draw fronts for.

The first step of the data retrieval is technically not part of the process F. This is to determine what data to retrieve. For any given date, there are two sets of files: The model data generated at 00:00, and the data generated at 12:00. Each of these sets of files contain a separate file for each time increment, from 12 hour prediction to 120 hour prediction. Given a user input date, the system loads the corresponding files from the StormGeo central repository. Each of these files is a "Weather data file" in figure 7 above.

From this point, and throughout section 6, I will only consider the process for one single time step, i.e. one file, but the entire process F is of course repeated for all files in the acquired data set.

6.1 Data retrieval

The second step is to read and interpret the file as a three-dimensional array. The data files are NetCDF files that contain multiple weather variables and their values for a given set of longitudes and latitudes. The latitudes for the files in this project vary between 73N and 25N, while the longitudes vary between 66W and 55E. The variables in each file are:

- The u component of wind[1].
- The v component of wind.
- The geopotential height at 500 hPa[2].

[1] Wind speed and wind direction are denoted as two vectors, one in the east/west direction (knows as the u component or the zonal velocity), and one in the north/south direction (known as the v component or the meridional velocity). The sum of these two vectors define the wind direction and speed for a given point (Hooper 2002).

[2] Geopotential height is a way to measure the thickness of the atmosphere. "*Geopotential height approximates the actual height of a pressure surface above mean sea-level. Therefore, a geopotential height observation represents the height of the pressure surface on which the observation was taken… heights are lower in cold air masses, and higher in warm air masses*" (SCOoNC 2010).

- The specific humidity[3].
- The temperature at 500 hPa.
- The temperature at 700 hPa.
- The temperature at 850 hPa.
- The relative vorticity[4].

All of these variables are saved as two-dimensional arrays, together forming a three-dimensional array.

6.2 Data normalization and transformation

In order to perform edge detection on the data set, it needs to be represented as a single 2D-array. This means that the third dimension of the data set must be collapsed in some way. In H, this is solved with a simple, linear function; A weighted average. This linear function is of course a substantial abstraction from what is most likely a non-linear relationship between weather variables and fronts, but it yields sufficiently accurate results.

Firstly, the data in each array is normalized to values between 0 and 200, based on the lowest and highest values in the array. This is to make sure that the data variance is comparable across variables. The vorticity variable, which varies inversely to the rest, is also inverted during normalization. Secondly, a weighting of all the variables is created, based on their contribution to front detection. The motivation for this selection and weighting can be found under 7.2. The weighting is as follows:

U component of wind	0	Temperature at 500 hPa	10
V component of wind	0	Temperature at 700 hPa	5
Geopotential height	20	Temperature at 850 hPa	0
Specific humidity	5	Relative vorticity	1

Table 2: The weighting of atmospheric variables.

[3] Specific humidity, as opposed to absolute humidity, is a measure of relative water mass in a given air mass. Given that no moisture is removed or added, the specific humidity of a system will not be altered by changes in pressure or temperature (Britannica 1998).
[4] Vorticity is the rotation of a system relative to its context. Relative vorticity is the tendency of a system to rotate relative to the Earth (Ahrens p. 351).

Finally, a new 2D array is created, and populated with the weighted average of all the variables. This means that for any point (x,y), the value will be:

$$\frac{\left((\mathrm{GeoH}(x,y) * 20) + (\mathrm{SpeH}(x,y) * 5) + (\mathrm{Temp500}(x,y) * 10) + (\mathrm{Temp700}(x,y) * 5) + (\mathrm{RelV}(x,y))\right)}{20 + 5 + 10 + 5 + 1}$$

After H, we are left with a 2D-representation of our weather state, which is used to detect fronts. An image rendering of this 2D-representation can be seen below. Each pixel in the image represents a geographical point in the North-Eastern Atlantic Ocean. A lighter area indicates lower temperature, geopotential height and humidity, and a higher vorticity. A darker area indicates the opposite.

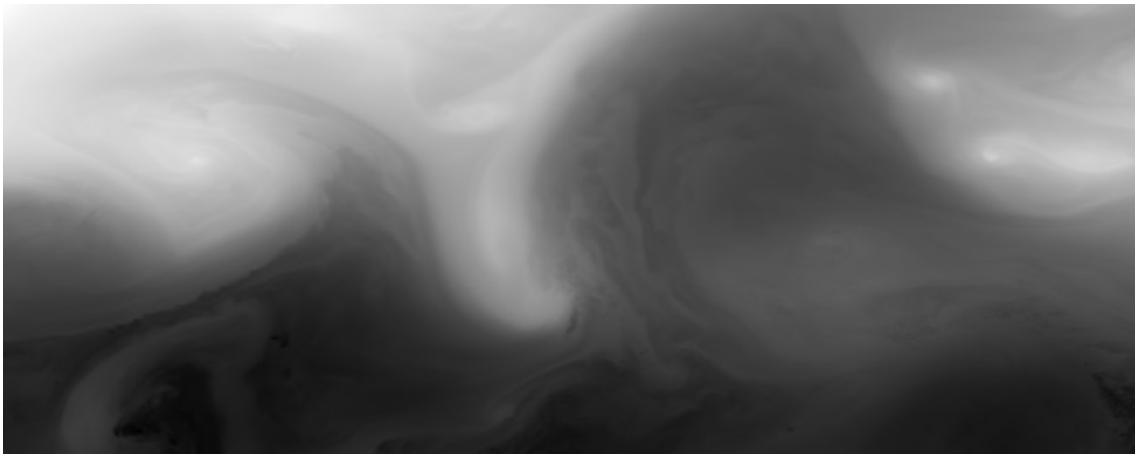

Figure 8: Visualization of the combined, normalized data set for September 13th 2016, 12:00PM, 12 hour prediction.

6.3 Edge detection

The algorithm used for edge detection is described in detail under 5.4.1. This process takes the normalized 2D array, and returns a binarized 2D array where only the discovered edges keep their values, while all other points are set to 0. An image rendering of the binarized 2D array from edge detection can be seen below. A black pixel represents an edge point discovered by the algorithm.

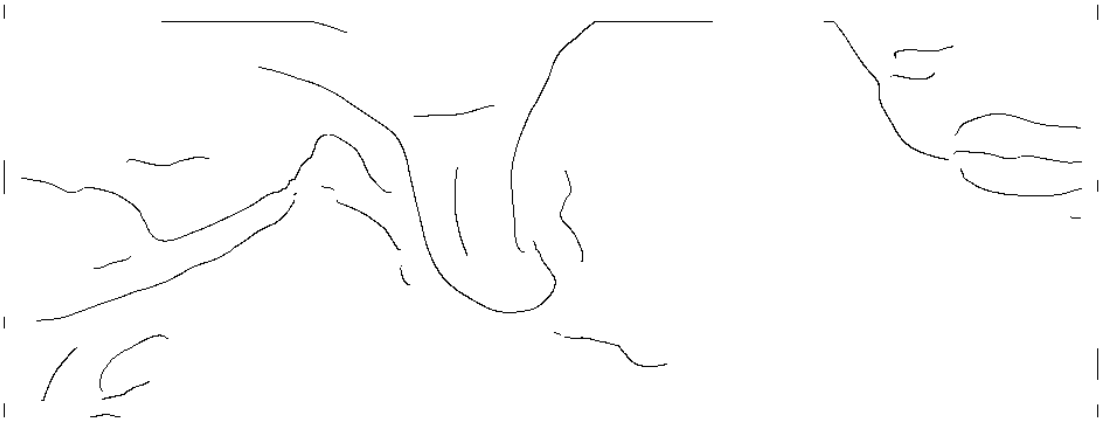

Figure 9: Visualization of the edge detected data set for September 13th 2016, 12:00PM, 12 hour prediction.

This results in a rough sketch of the major contours in the weather system. These detected edges are an abstraction from the full data set, and takes us a step closer to a representation of fronts. However, this representation remains too noisy to accurately represent the major front lines in a system. The lines are too scattered, there are too many parallel lines and too many tiny, irrelevant lines for this result alone to inform a detection of fronts. Some further processing is therefore required.

The edge detector uses a high threshold of 3, a low threshold of 2, where 2 and 3 are the gradient values around edge points, and a kernel size of 10 for gaussian blurring. The motivation for this particular threshold and kernel size assignment can be seen in figures 10-14 below. Having higher thresholds makes the algorithm too insensitive, and it picks up only the absolute clearest lines available. This gives too little input to create candidate fronts from, and in some weather situations no lines are discovered at all. Lowering the thresholds has the opposite unwanted effect, where far too many subtle value changes are picked up by the algorithm, and it becomes extremely difficult to extract the critical features of the system. Lowering the kernel size has a similar effect, where the noise becomes overpowering, and the system struggles to distinguish between major lines and small discrepancies.

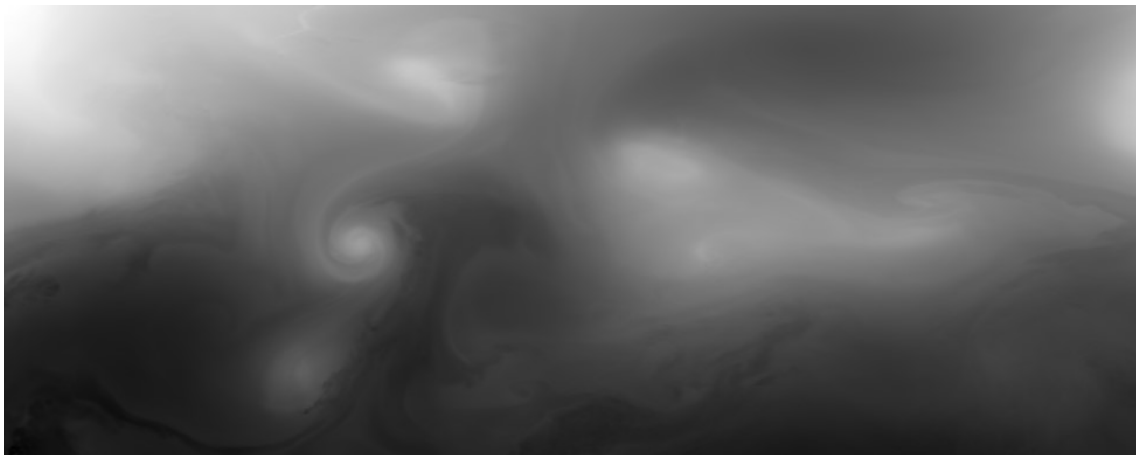

Figure 10: Original data table for showcasing effects of different edge detection thresholds and kernel sizes.

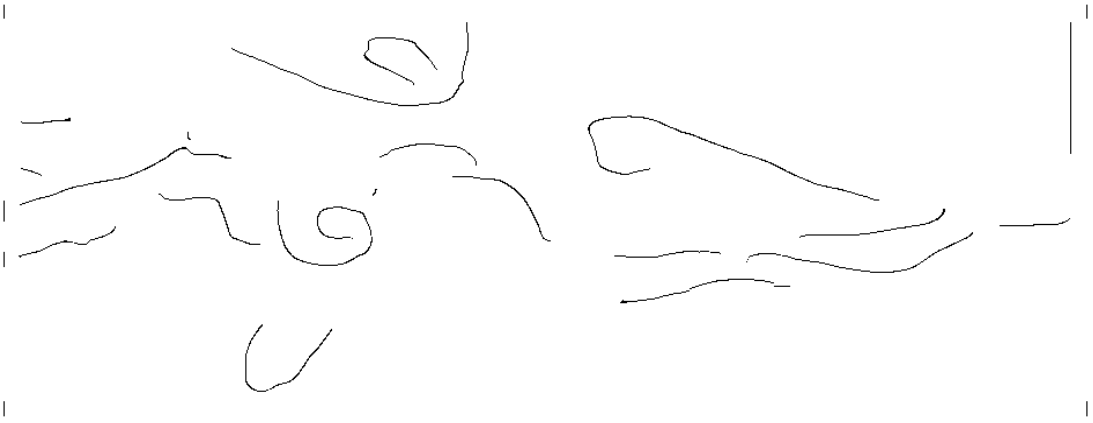

Figure 11: Result of edge detection with "correct" values. HT 3, LT 2, KS 10.

Figure 12: Result of edge detection with higher threshold. HT 5, other values equal.

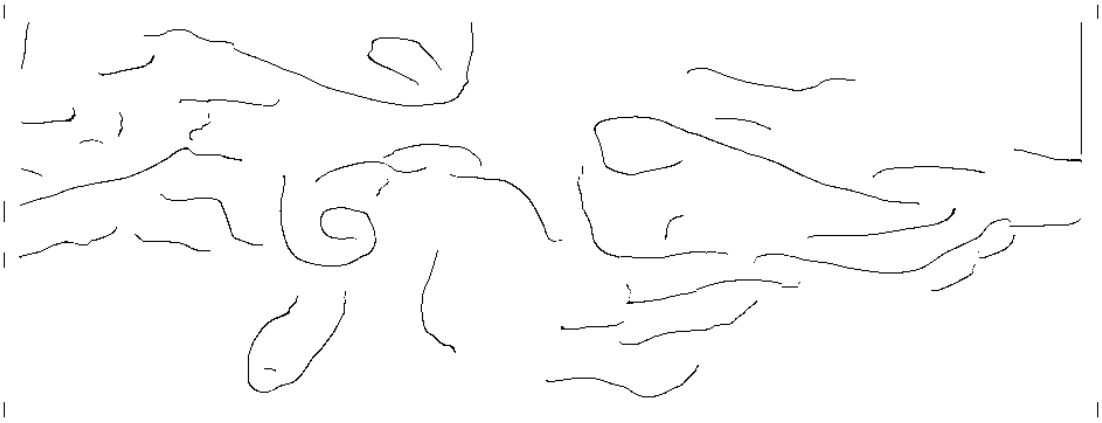

Figure 13: Result of edge detection with lower thresholds. HT 2, LT 1, kernel size equal.

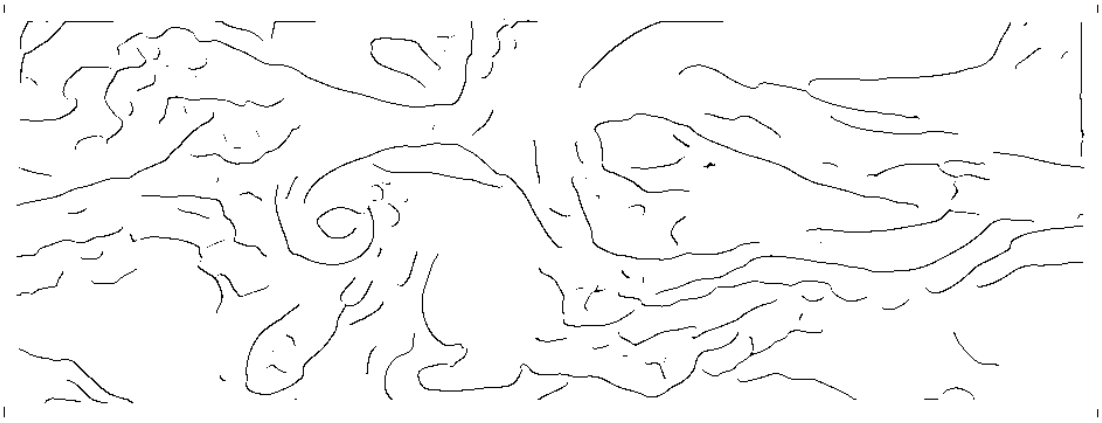

Figure 14: Result of edge detection with lower kernel size for blurring. HT 3, LT 2, KS 5.

6.4 Line identification

The next step of the process is the part of the algorithm that is the most novel, and the least grounded in existing theoretical frameworks. This is the part where the result of the edge detection process must be transformed into a set of identified candidate fronts. In this undertaking, I have used some innate knowledge about the structure and qualities of fronts, but mostly I have tried to make a system that identifies the major line structures in the data without considering heuristical approaches to front detection. This is of course motivated by my desire to find a computational, data driven approach to front detection, that is not biased or predisposed in the same way that human judgement is biased, unless it is strictly necessary for the system to have functional value.

The line identification can be divided into four sequential tasks:

- A: Find all the line segments in the edge detected data.
- B: Filter these line segments into meaningful lines
- C: Find the key points of each line.
- D: Smooth and filter the lines based on the key points and the curvature of the lines.

To easily understand the purpose of **task A**, it is helpful to consider the data structure of the final input and output of F. At some point in F, the gridded meteorological data table must be transformed into a list of lines, and the coordinates of the points of these lines. This is what happens in A.

The algorithm goes through the data table from top to bottom. Whenever an edge point is discovered, a recursive algorithm is employed to find the end of the line on which the discovered edge point is sitting.Then the same algorithm runs from the end of the line, this time storing all the points it passes from one end of the line to the other. When the other end of the line is discovered, the line is stored, and all its points are marked as discovered, making sure that the same line segment cannot be added more than once.

The recursive “Find Line From Point” - function works by searching for edge points in a 9x9 grid around the current point, starting closest to the point and working outward. When we do not find any new edge points in this grid, we have reached the end of the line. When a new edge point *is* discovered, we have found the next point to search from. If the next point is not the immediate neighbour of our current point, a straight line is also added between this point and the starting point, forming a complete, uninterrupted line segment. If the next point *is* the immediate neighbour, any *other* directly neighbouring edge points (that have not already been added to the line) are discarded. This final case often occurs in perfect diagonals, when the line forms a “stair” shape in the data points. If the additional neighbours are not excluded, these will often be rediscovered when we reach the end of the line, and the final line will run back on itself. This is of course unwanted behaviour.

At the end of task A, we are left with an abstract representation of all of the lines in our edge detected array.

In **task B**, we want to abstract away the smaller, insignificant and irrelevant line segments, as well as to join any line segments that belong to the same line, but are for whatever reason separated. This is where we alleviate some of the shortcomings of the Canny edge detection

algorithm. Since the edge detector has trouble finding branching edges, a lot of the major line structures will have larger gaps that should be bridged.

Firstly, all lines with less than 30 line points are discarded. This is done in order to remove unnecessary noise before the filtering stage.

Secondly, a function is run to remove parallel lines that are close together. This is because two lines with the same gradient in the same area are most likely denoting the same general structure. Whenever both ends of one line are in close proximity to some other line (within 15 data points), this line is removed. This strategy also ensures that the shortest parallel line segment is removed, keeping as much data as possible.

Thirdly, another function joins the ends of lines that are close together. This is run after the removal of parallels to insure that two parallel lines that point to the same structure are not joined to form a circle. In this step, if the ends of two lines are within 30 data points of each other, they are joined, forming one consecutive line. This approach is very simple, and leaves some lines joined at strange and unnatural angles, but this problem is partly mitigated in parts C and D.

After joining ends, the parallel removal is run again, to make sure that any potential newly joined lines will not form parallel structures with existing lines. Finally, all lines with less than 80 line points are now removed. Lines that are shorter than this are not considered part of the main line structures in the data, and these will thus only clutter the final result.

Part C is a very simple algorithm that defines the key points of the line. This step moves from a general line representation, where *all* the points of the line are recorded, into a representation where only a selected number of points are stored. These key points then define only the main shape of the line, in the same way that front lines are typically represented by a set of key points (see figure 1 for an example).

To begin with, the endpoints of the line, as well as every 30th point, are defined as key points. This set is then pruned based on the curvature of the line. In each key point (excluding the ends), the line has a particular angle coming into the point, and a particular angle going out of it, relative to the plane the points are sitting on. Whenever the difference in angle in and out of a point is less than 10 degrees, this key point is removed, as it does not aid in defining the shape of the line in any meaningful way. After this pruning, any lines that have less than three

key points are removed. This is done in order to, once again, remove insignificant lines that do not represent a major line structure in the data.

At this point, we have a relatively sound representation of our edge detected lines, but this representation has some important discrepancies from typical front structures. In **task D** we take the final step from a line representation into a front-like representation. This is therefore the only part of the function J where some assumptions about the meteorological structure of fronts has to be taken into consideration.

The most important assumption being made has to do with the curvature of fronts. Since weather fronts are directional, i.e. they are moving in a given direction, they tend to curve in this direction. This has the added consequence, as I have learned through discussion with StormGeo's meteorologists, that one individual front will almost never change its curvature along its length, without also changing front type. This again means that our candidate fronts should all have a uniform curvature, and that lines with changing curvature should either be split or cropped to conform to this requirement.

Although making assumptions about the internal structure of our candidate fronts could be considered problematic, given my goal of making a data driven approach to front detection, I believe this particular assumption is necessary to make, and that its implications for the final results should not be considered detrimental. Most importantly, the assumption about curvature has no implication for the identification of fronts. All key points remain in the same positions after the smoothing, and the general shapes of the candidate fronts are still based solely on the edge detection results.

Firstly in task D, the curvature at each point (i.e. the difference in angle going in and out of the point) is calculated for all lines. Then some basic smoothing is applied. All outlier points that have different curvature to both their neighbours are removed, as well as ends that cause the final points of the lines to have different curvature from the rest. After each deletion, the curvature of the line is re-evaluated, as it will have changed around the deleted point. When no more points can be deleted, we are left with lines that have either one homogenous curvature, or a few, large sections with the same curvature.

Secondly, all lines are split into candidate fronts based on their new curvature. Whenever the curvature of a line changes, this is considered the beginning of a new front, and it is split into

two. The splitting is performed such that one point remains common for both fronts. The new fronts therefore still form one continuous line, but they are considered separate entities that can, among other things, be classified into different front types in classification. An important factor to consider is that there are always two possible points at which splitting is possible to maintain correct curvature: Closer to the first front, or closer to the second front. The position to split the candidate fronts at is always chosen such that the shortest front remains as long as possible. This is to avoid classifying structures that are extremely tiny and unlikely to point to a significant frontal feature. An example of this candidate front splitting can be seen in figure 15. In A we see the curvature of the line, which changes between points 3 and 4. This means that the line can be split at either of these points, still ensuring homogeneous curvature for both candidate fronts. These alternative splits are highlighted in B and C. The algorithm chooses to split at point 3, as shown in C, to insure that both candidate fronts are of significant length.

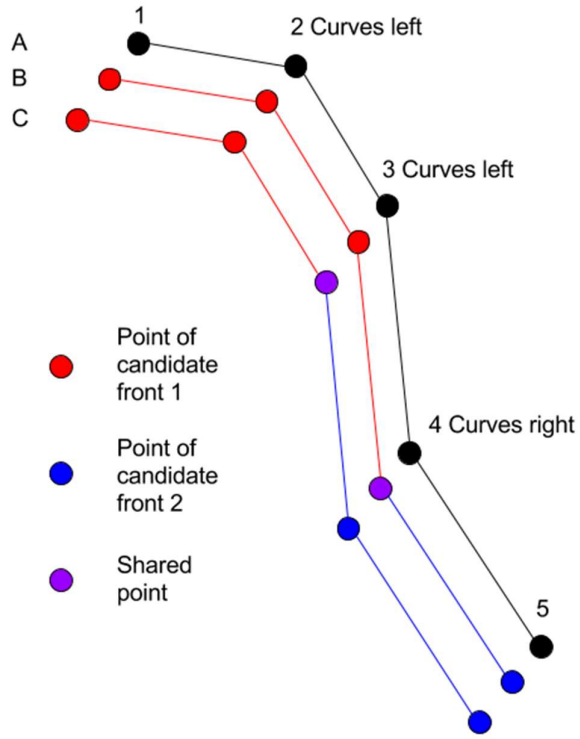

Figure 15: Example of candidate splitting.

At the end of J, we are finished with front identification, with the result being a set of lines with key points; The candidate fronts. What is left at this stage is the classification of these candidates into actual fronts. An image rendering of the candidate fronts superposed on top the original data table from 7.2, can be seen below. Each black square represents a key point in a candidate front. It is worth noting that the illustration does not highlight where the candidate fronts have been split, and only displays them as complete lines, even where splitting actually has occurred.

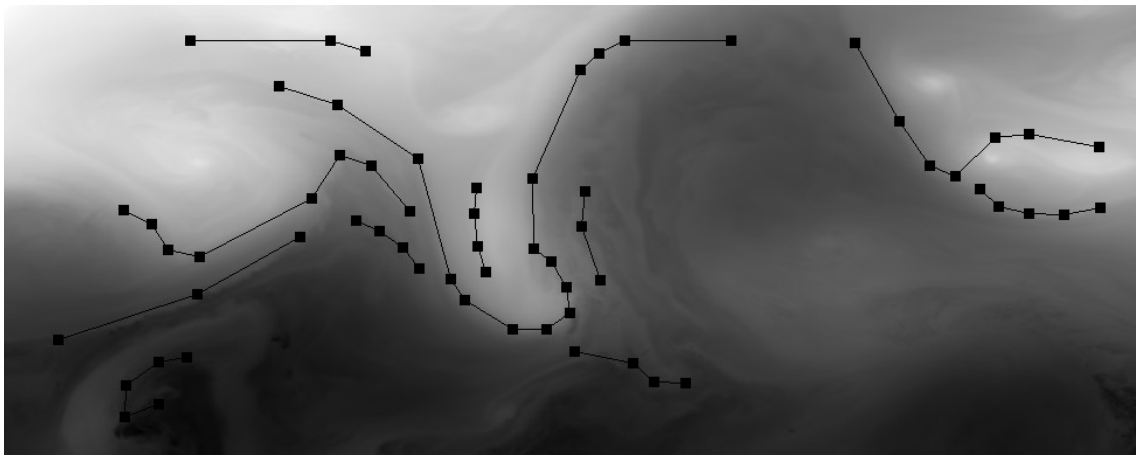

Figure 16: Visualization of the candidate fronts for September 13th 2016, 12:00PM, 12 hour prediction, discovered by edge detection, on top of the original data table.

6.5 Front classification

The front classification essentially takes our data model back into the weather domain, from lines with points, to fronts with positions. This is being done with the help of a classifier. The translation is a twofold process: First classifying the single points of the candidate fronts into front positions with front types and directions, and afterward using this classification to decide the types and directions of the whole fronts. This means that every key point on a line is classified as a position in a front, before the front is created, based on the result of this classification. Figure 17 contains an illustration of the relationship between the two concepts.

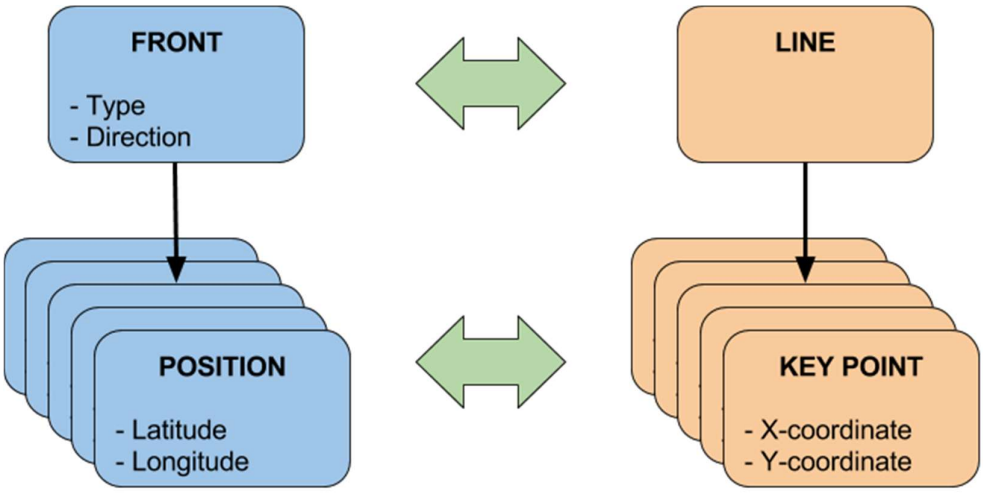

Figure 17: The relationship between lines and fronts in the system.

The end goal of front classification is to have transformed all of the identified lines (candidate fronts), into actual fronts, with a front type, a direction, and a list of coordinates that make up

its positions. Front type can have one of three values: Cold front, warm front or occlusion, while direction can be either left or right. Left and right in this case refers to the direction the front is moving, relative to the direction it is being drawn. A front that has its positions defined from west to east, and a right direction for instance, is moving southward.

The classifier is an artificial neural network, designed to classify single points, giving them a direction and a front type based on the values around the point. For every point (x,y), there are five inputs:

- The absolute value of the point.
- The change in value (x - 10, y) → (x + 10, y), i.e. north → south.
- The change in value (x, y - 10) → (x, y + 10), i.e. west → east.
- The change in value (x - 7, y - 7) → (x + 7, y + 7) i.e north/west → south/east.
- The change in value (x + 7, y - 7) → (x - 7, y + 7) i.e. south/west → north/east.

These are being classified into four outputs with decimal values between 0 and 1:

- The direction of the front in the point (<0.5 = left, >0.5 = right).
- The likelihood of the point belonging to a warm front.
- The likelihood of the point belonging to a cold front.
- The likelihood of the point belonging to an occluded front.

The network is trained on the positions of fronts drawn by meteorologists at StormGeo between September 5th and November 27th 2016, as well as the direction and type belonging to these fronts.

The input variables for the network are neither arbitrary nor definitive, but have been chosen through a trial and error testing. The assumption used is that the relevant input for classification consists of the weather state on the frontal surface (i.e. on the point), as well as the change in weather state around the frontal surface, since these are the features meteorologists study when deciding the type of a front. For the inputs representing the change in weather state, I have tried three different strategies:

1. Grid approach: Computing a grid around the front point of varying sizes and using the average difference in value from the front point to the grids as the input values.
2. Gradient approach: Computing the changes in four cardinal directions through the point. As shown earlier, this is the strategy currently being used.
3. A combination of 1 and 2.

A visualization of the two strategies can be seen in figure 18 below. Left: The grid approach with corresponding input values. Right: The gradient approach with corresponding input values.

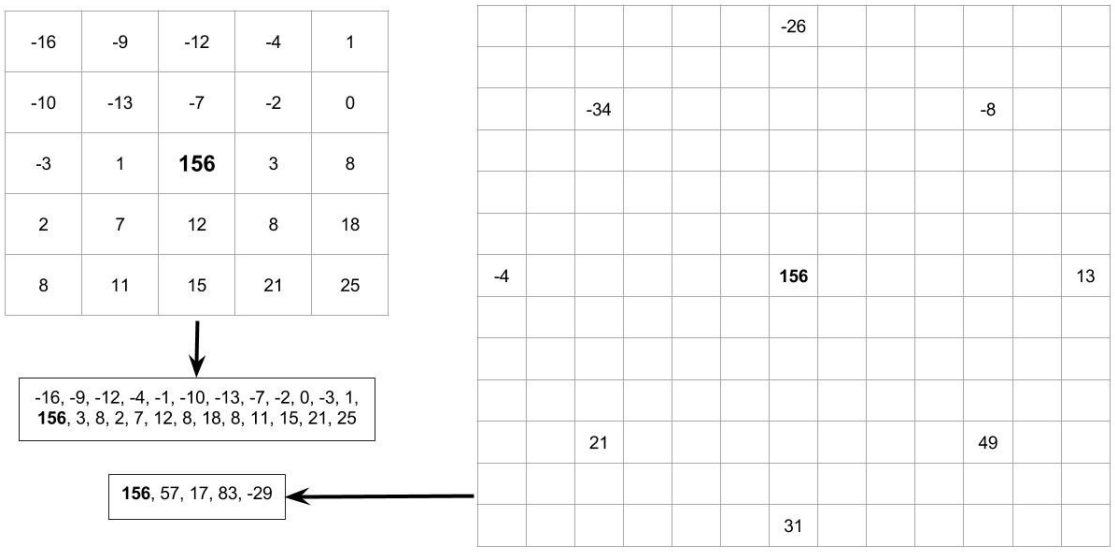

Figure 18: Comparison of the two strategies for calculating inputs to the neural network (not to scale).

With strategy 2, I have tested several different radius sizes for the gradient calculation, settling on 10 data points (= 2,5 degrees lat/lon). The ideal radius size will vary with different resolutions and different latitudes. This combination of strategy and radius size yields the lowest error rate (~18% erroneous classifications) with the current training set.

When all the front positions have been classified, they are rejoined to form fronts, based on their typing. This starting point of classified positions, assumes that a position belongs to a front of the type and direction that is most strongly classified for that position. For instance, a position with a cold front score of 0.35, a warm front score of 0.71 and an occlusion score of 0.24 (referring to output variables 2-4 above), will be classified as a warm front position.

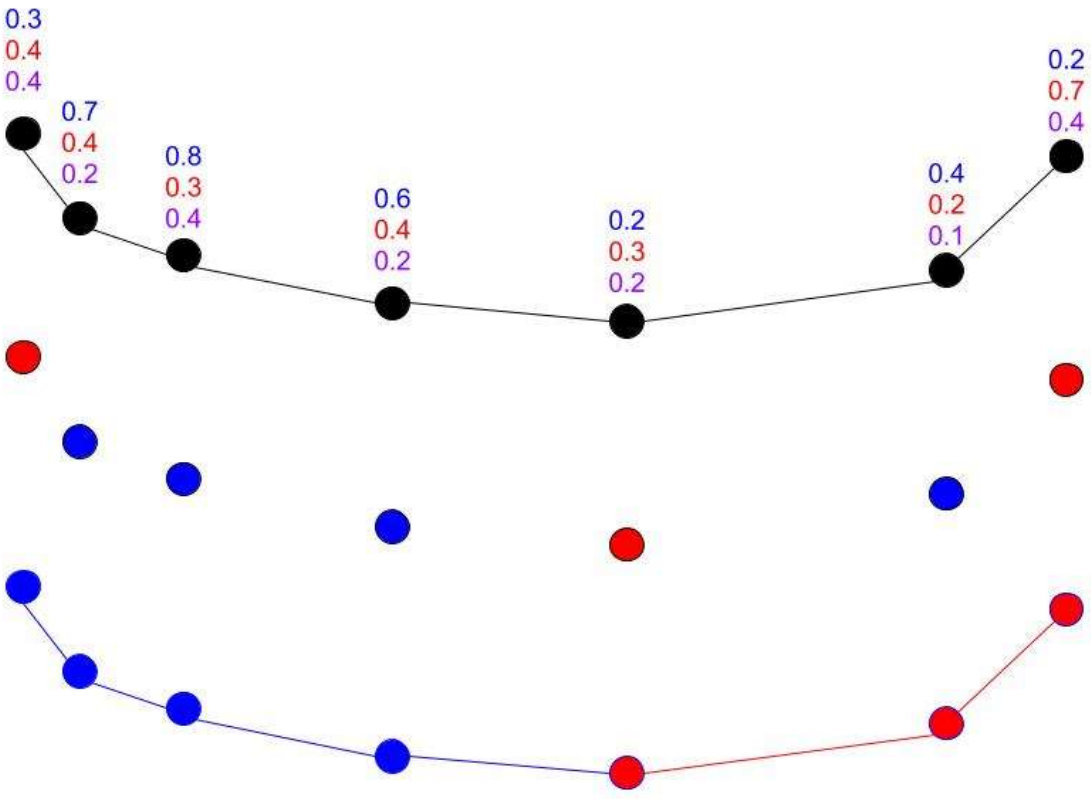

Figure 19: Example of a front classification.

Given the classification of its positions, a front may remain whole, or split into different fronts of different types. This splitting task is performed similarly to how the candidate fronts are spit in 6.4. Here, the splitting was based on the curvature of the lines, while in classification, the splitting is based on the classified types of each of the positions. As with the line splitting, single anomalies within or at the end of fronts are overridden, while changes in front type that lasts for two positions or more are counted as a new front. This insures that only definitive changes in front type are considered by the system. An example of how this splitting works can be seen in figure 19 above. Top: The classification scores for each position. Blue = cold front, red = warm front, purple = occlusion. Middle: The immediate classification of the positions based on scores. Bottom. The final classified fronts, split into one cold front and one warm front. The first and sixth positions have had their classification overridden due to the context of surrounding positions.

Finally, each of these new fronts are given their direction based on the direction of the majority of their positions. A front with two left positions and six right positions will be classified as a right moving front. After classification, we are left with our final set of

identified and classified fronts. A visualization of these fronts can be seen in figure 20 below. Red lines are warm fronts, blue lines are cold fronts, and purple lines are occlusions. Notice how all the front positions mirror a key point in figure 16.

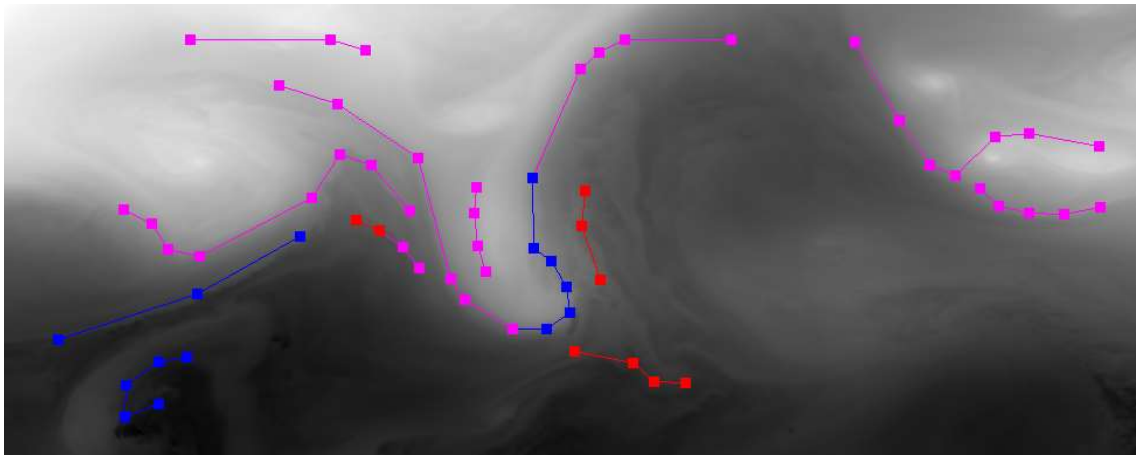

Figure 20: Visualization of the final front classification for September 13th 2016, 12:00PM, 12 hour prediction, superposed onto the data set it was discovered from.

6.6 Data generation

This is the final step of the process, and it does not contain any technically or academically noteworthy elements. The derived list of fronts is translated into a plain text file with info about front type, direction and coordinates of all the positions of the front. This is the data type used by the meteorologists at StormGeo to visualize the detected fronts. This data file is the sole output of the system, and it represents the main result of the project: A set of automatically detected fronts, based on a specific time slice of weather data. This result will be further discussed below.

7 Results

In this section I will present the results of the project, both technically, academically, and organizationally. This includes all results related to my two research questions, as well as the concrete use of the software currently by StormGeo.

7.1 Overall system success

The results presented in the following serve as the basis for answering the first research question: *Is it possible to automatically locate and classify the fronts in a weather system?* They are the outcomes of analyses of the observations made by meteorologists at StormGeo, gathered throughout January and February 2017, described in 5.3.2. In the evaluation they have compared the output of the system with manually drawn fronts for the same time period, comparing the relative strengths and weaknesses of both approaches. This relates most importantly to the identification of fronts, but some attention has also been given to the classification of front types. Throughout this subsection, results will be illustrated by figures like 21, below, showing the difference between the manual and automatic front detection procedures.

When discussing results related to front identification, we consider the relative placement of front positions by the system (the location of the coloured squares in the figure), while front classification refers to the typing of the front positions (the colouring of the squares in the figure).

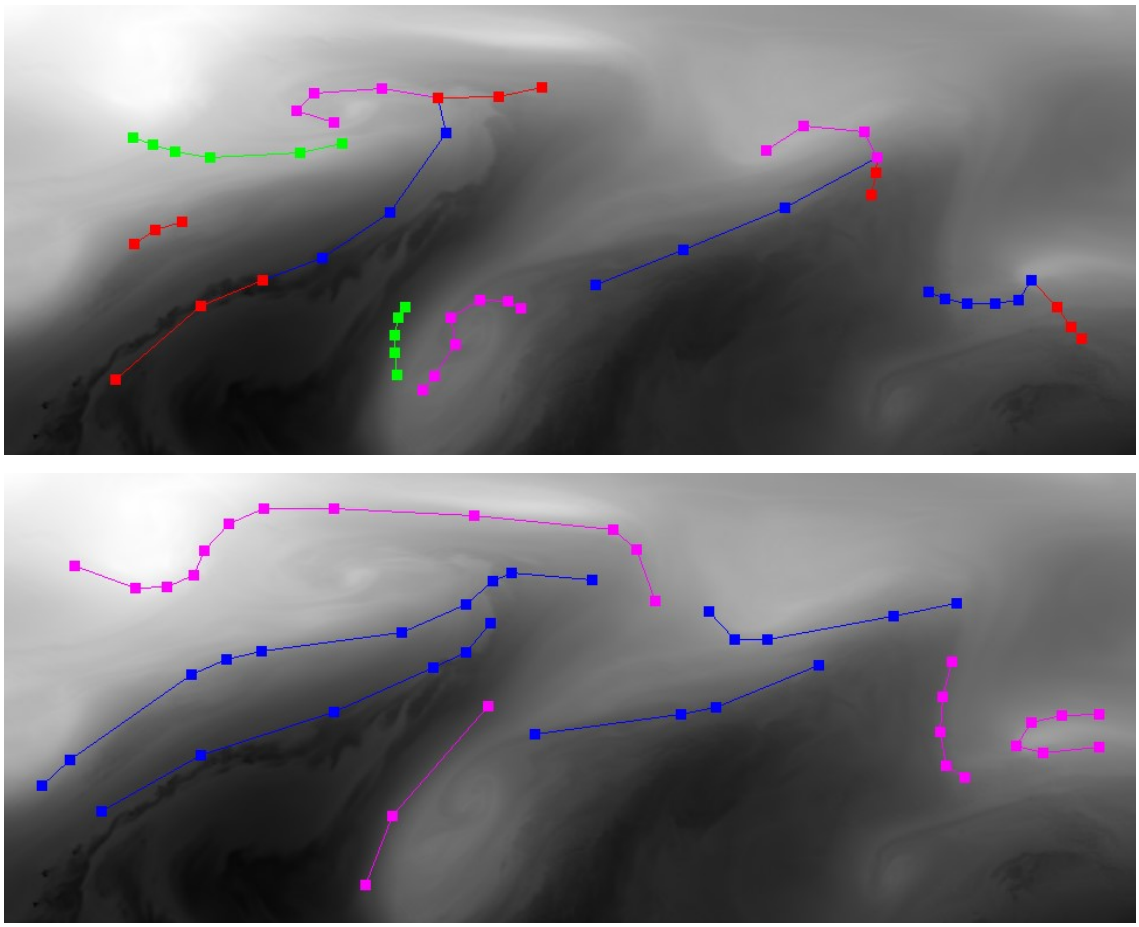

Figure 21: Comparison between automatic (top) and manual (bottom) front detection results for October 25th 2016.

7.1.1 Front identification

In general, we find that the front detection system excels at finding the approximate positions of the major front systems, but it is lacking when it comes to detailed placement, as well as some specific quirks. Figure 22 is a good example of this.

This figure serves as an example to display many of the major findings from the evaluation, related to identification:

The system shows good results in detecting where the major frontal systems are. We have a long, north/south oriented front on the west side, sweeping eastward, we have a swirling occlusion (purple coloured) in the south, and we have a small front in the south-east, that the system picks up rather well.

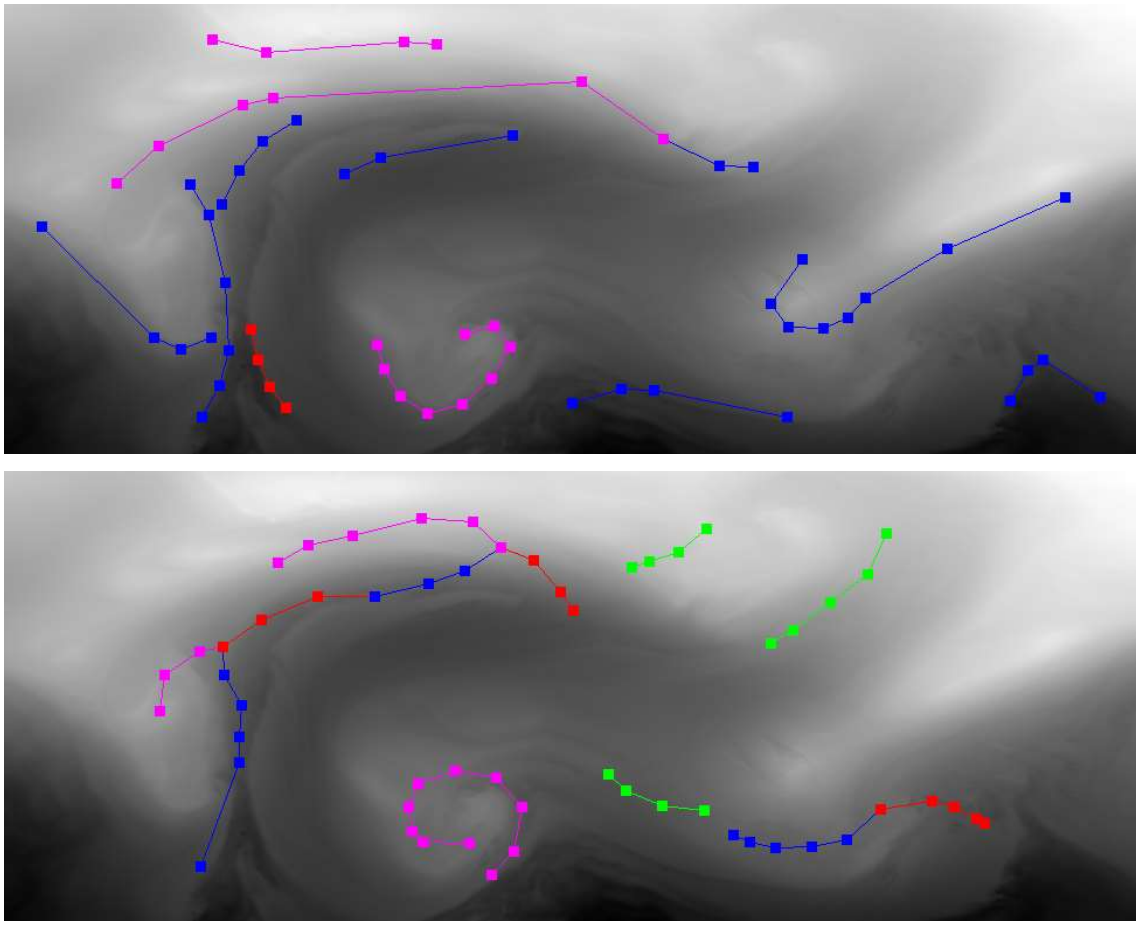

Figure 22: Comparison between automatic and manual front detection results for November 30th 2016.

The system easily finds large, long and well-defined fronts. Once again the system to the west is a good example. We can see just from the gradient in the background image that this front is clearly identifiable in the data, and the system has no problems finding its general location.

The system sometimes finds fronts that are not drawn by the meteorologist, but can still be considered correctly identified. The cold (blue) front in the center/east portion of the image, has some typical cold front characteristics (temperatures and pressure rising quickly around the front line), and should most likely be part of the frontal analysis, but it is missing from the manually drawn figure. This is just one example of how comparing directly with manually drawn fronts will not always provide a good basis to evaluate the system, since the manually drawn fronts are never 100% correct.

The system has substantial problems with forking fronts. Once again looking at the system in the west, the meteorologists have drawn two classic cyclone structures, with a cold front forking into an occlusion and a warm front. Looking at the points where the manually drawn

fronts fork in different directions, the algorithm is usually unable to determine which lines should be connected, and at what angles fronts should intersect. Often we see that the fronts are not intersected at all, and that they form structures that are very unlikely to occur in nature.

The system is often unable to draw a front on the "correct" side of an air mass. On the cold (white) air mass to the south, the system has drawn a swirling occlusion, even swirling in the correct, anti-clockwise direction, but it starts on the wrong side of the air mass. This gives a completely wrong image of how the occlusion is actually moving.

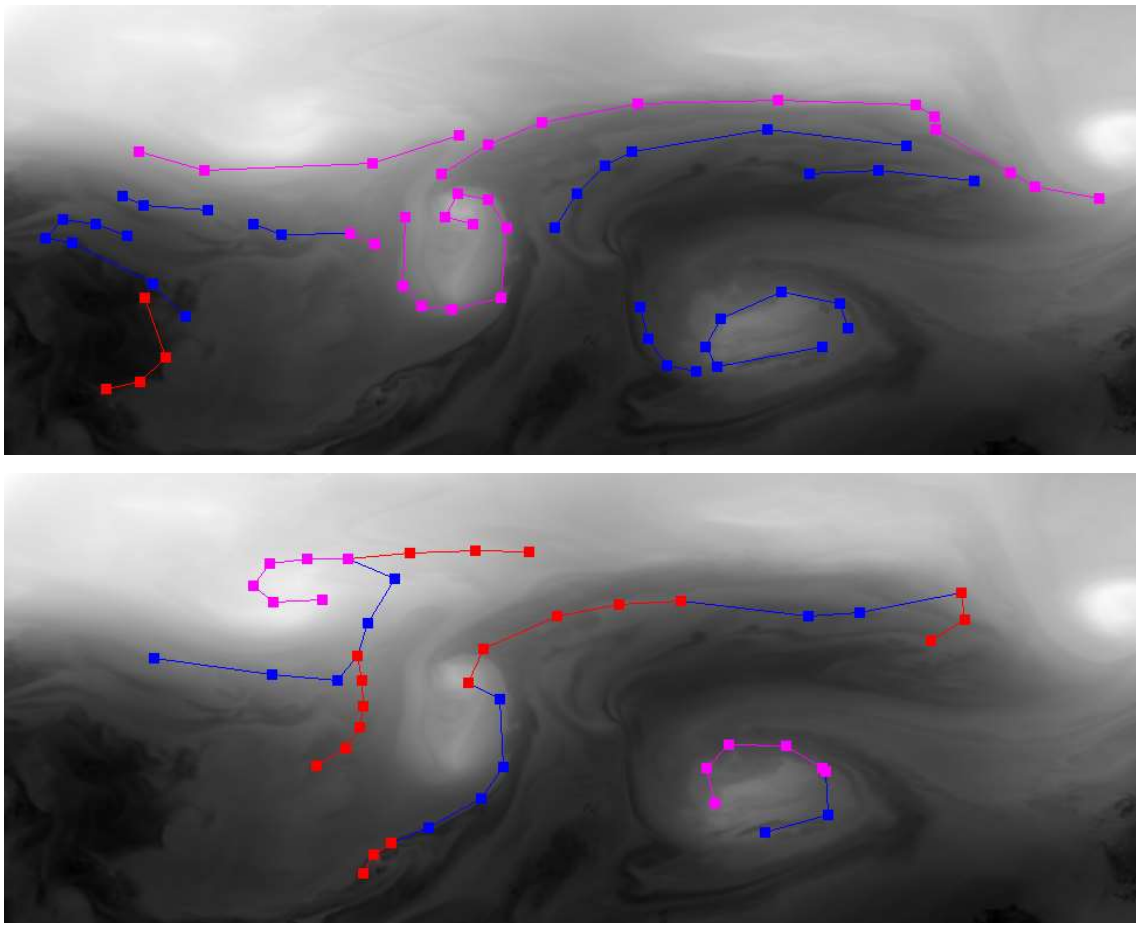

Figure 23: Comparison between automatic and manual front detection results for September 7th 2016 (12PM).

Figure 23 above highlights a different problem. While we also recognize that many of the previous observations also hold true here, there is another problem to consider. In the north-western corner the meteorologist has drawn an occlusion as part of a classic, forking system. This front is not even hinted at, let alone fully dawn by the system. Looking at the background data it is easy to see why. The data set is almost completely uniform in this area.

If the edge detection algorithm were tuned to find a line here, it would also find hundreds of erroneous fronts in other locations in the same time slice. This is a case where the system struggles where the meteorologist has no problems finding a front. Such examples are common with the current version of the system, particularly with occlusions in colder regions, such as in this example.

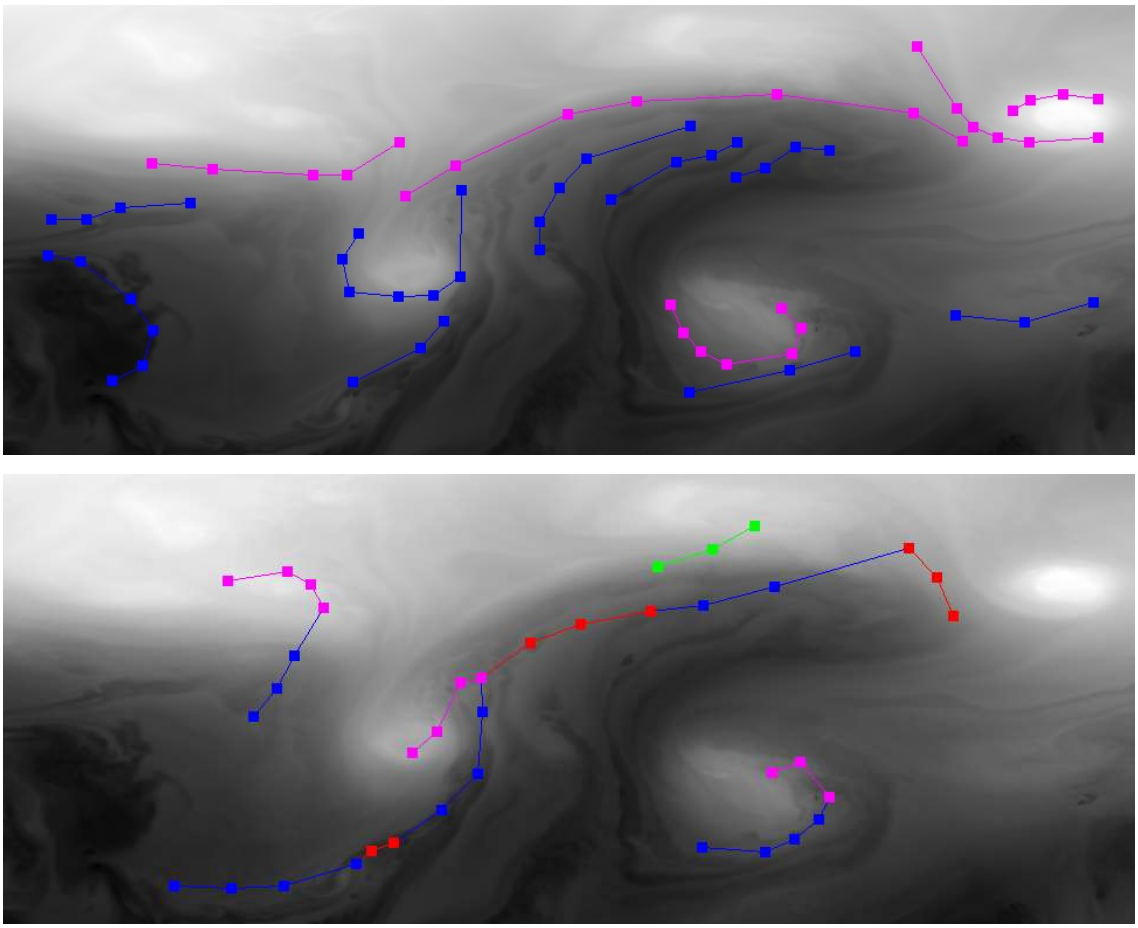

Figure 24: Comparison between automatic and manual front detection results for September 7th 2016 (12AM).

On the other hand, we have some opposite types of problems, where the system finds fronts where it perhaps should *not*. In figure 24 above, the system finds four parallel fronts in the centre of the image. Only one of these are drawn by the meteorologist. This discrepancy appears several times in the test data set, but determining the correctness or incorrectness of the system in this case is far from a straightforward matter (see 8.1.2 for more).

In figure 25 below, however, this judgement of correctness is much easier to make. Here we see that the system has discovered a vast amount of fronts compared to the meteorologist. Looking at the weather data behind this reveals the reason: Because the algorithm normalizes

the data before edge detection, to the same value intervals (0-200, see 6.2 for explanation), whenever there are no large gradients in the data, the smaller gradients will be artificially inflated to conform to the normalization process. This means that to the system, there are sharper lines in the representation than in the actual corresponding weather data, leading to a front identification that is too "eager", that finds too many fronts.

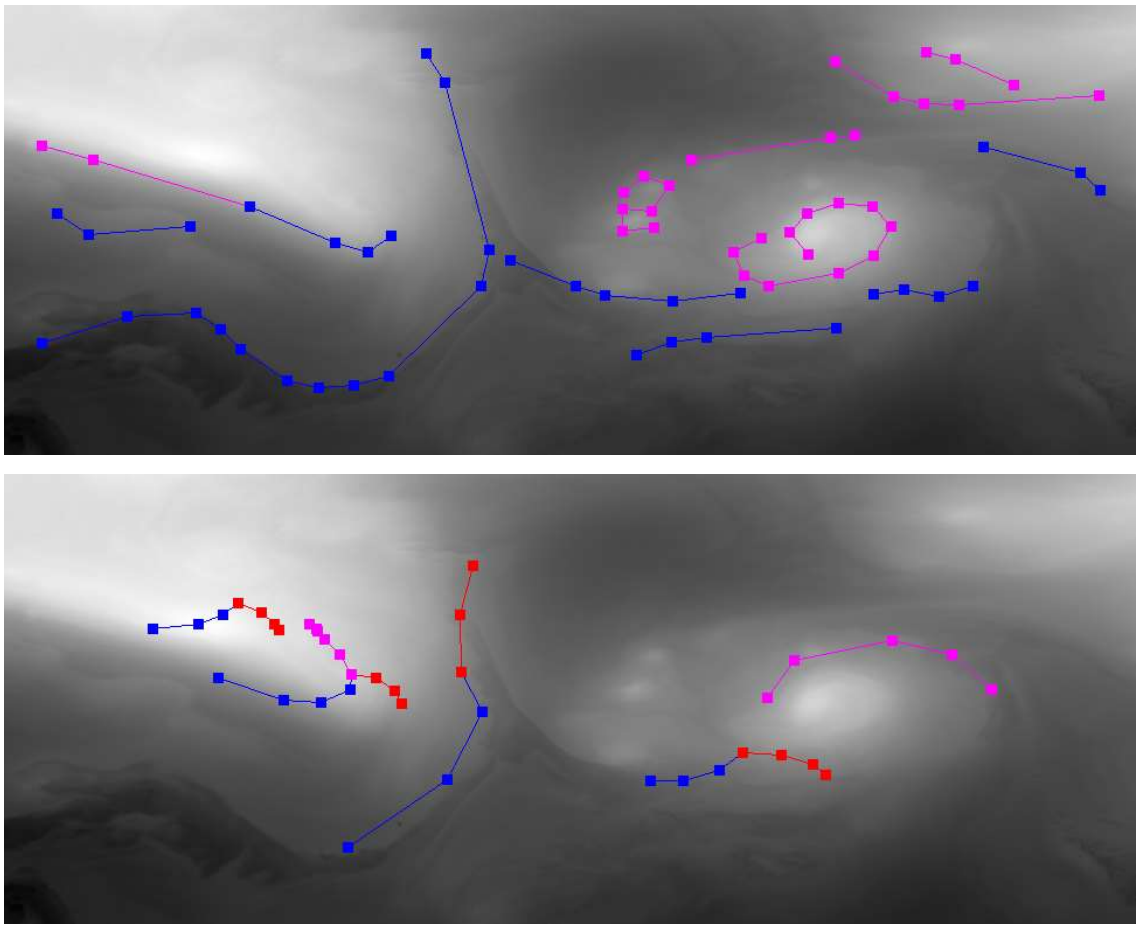

Figure 25: Comparison between automatic and manual front detection results for October 6th 2016.

Although the results point to many flaws in the front identification, it is important to note that on a more general level, the algorithm performs rather well. When we consider finding the major front lines in a system, figures 21-25 all show that the system can perform this task quite comfortably in most cases. Although the majority of the presentation and discussion of the results are devoted to the details and discrepancies, one should not lose track of that higher-level success. Given that the details can be solved better, the basic structure of the algorithm already solves the general problem of front detection remarkably well.

7.1.2 Front classification

When it comes to classification, the system does well on a few things, but still leaves much to be desired. The most glaring shortcoming is the sheer lack of warm fronts in the data. Figure 26 is a very typical example of a classification. We see too many occlusions, and very few warm fronts, often none at all. Cold fronts are the only types that at this point I would consider sufficiently well classified.

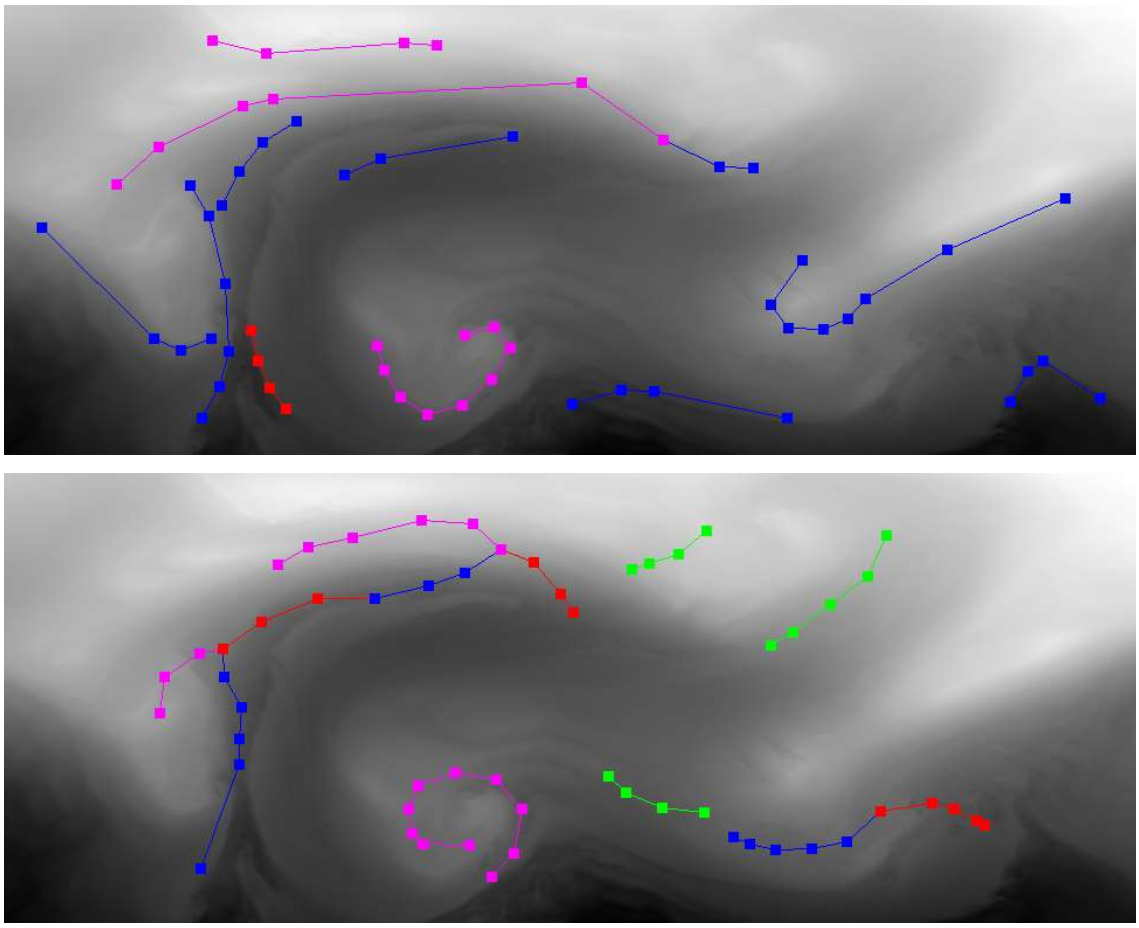

Figure 26: Comparison between automatic and manual front detection results for November 30th 2016.

A second problem with classification is the uniformity of the classification. When an area has mostly occlusions, it tends to have *only* occlusions, and when an area has mostly cold fronts, it has almost always *only* cold fronts. The cases where the front type changes along the same front line are extremely rare in the automated results, while with the meteorologists, this occurs frequently.

7.2 Variable selection and weighting

This section serves to provide data for answering the second research question: *What features of a weather system are critical in locating and classifying fronts?* In order to answer this question, we must look at how different weather variables influence the front detection process. A theoretical, perfect variable will have distinct and sudden changes in value if and only if there is a front at that location. In reality, no such variable exists. What we want to do then, is to find a combination of variables that together limits erroneous front identification.

In this endeavour, I have considered 17 different weather variables, 7 at sea/surface level and 10 higher up in the atmosphere. These are:

- Geopotential height
- Humidity at two levels (surface and atmosphere).
- Precipitation.
- Relative vorticity.
- Temperature at four levels (surface and three different heights in the atmosphere).
- Zonal wind velocity and meridional wind velocity at three levels (surface and two different heights in the atmosphere).
- Air pressure adjusted to sea level.
- Dew point temperature adjusted to sea level.

These 17 variables form a fairly comprehensive set of weather features that could be used to identify fronts and changing air masses. Figures 27-39 display the value distributions for most of these variables for a given time, with the manually drawn fronts from the same time step superposed on top. I will explain why every variable is considered critical or not for front detection, and what weighting it has in the normalization process. It is important to note that a variable that I do not consider critical can still be useful in front detection, just not for the particular system I have developed.

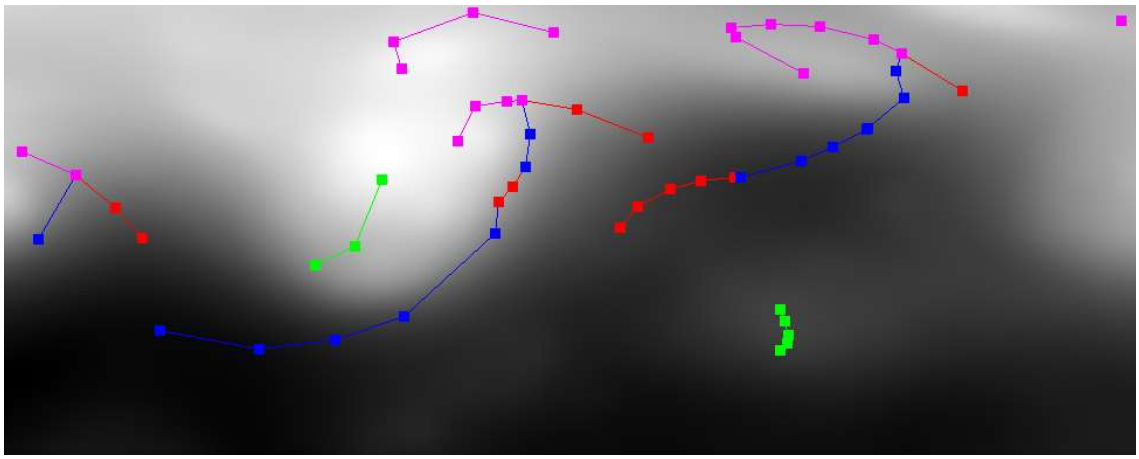

Figure 27: Geopotential height compared with manually drawn fronts.

Through conversations with meteorologists at StormGeo I learned that geopotential height is one of the variables commonly used by meteorologists to detect fronts, so it is no surprise that this variable correlates well with manually drawn fronts. An important feature of geopotential height is the relative lack of noise. As figure 27 shows, the value distribution is uniform and smooth, making it ideal for discovering major front lines. This is why geopotential height is considered a critical feature, and is in the variable set used by the system, with a high weighting as well. The only major fault with this variable is its inability to predict the location of occluded fronts (purple in the figure), which we can see occur in areas with fairly uniform geopotential height. This problem we saw highlighted in 7.1.1.

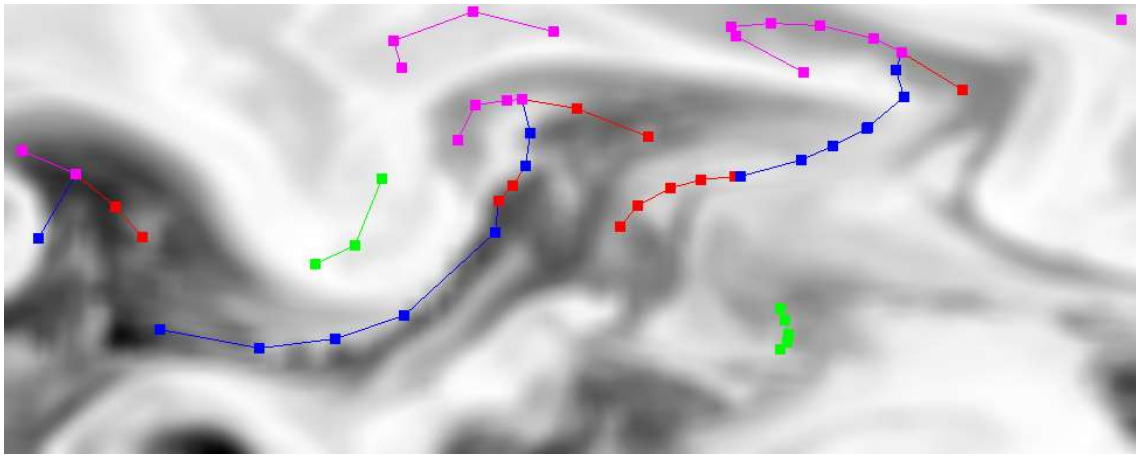

Figure 28: Absolute humidity compared with manually drawn fronts.

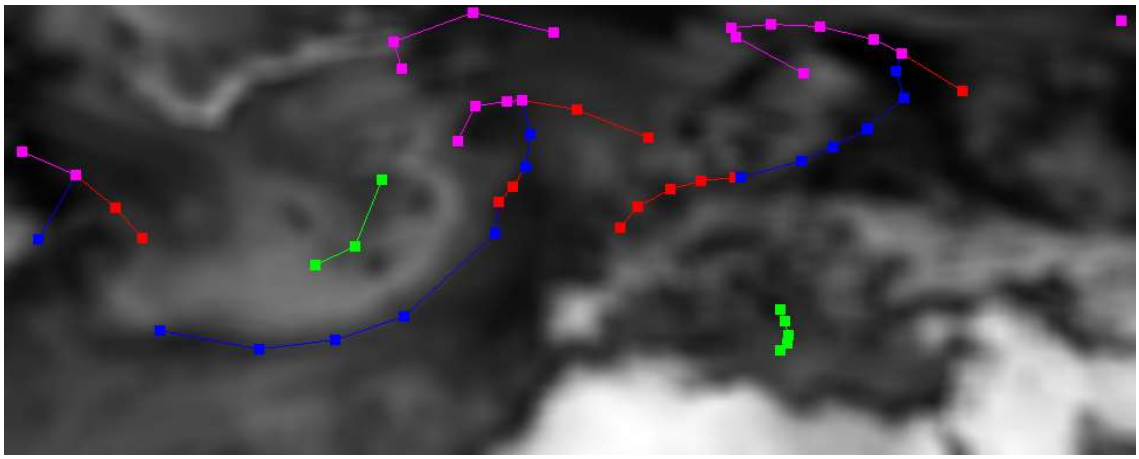

Figure 29: Relative humidity compared with manually drawn fronts.

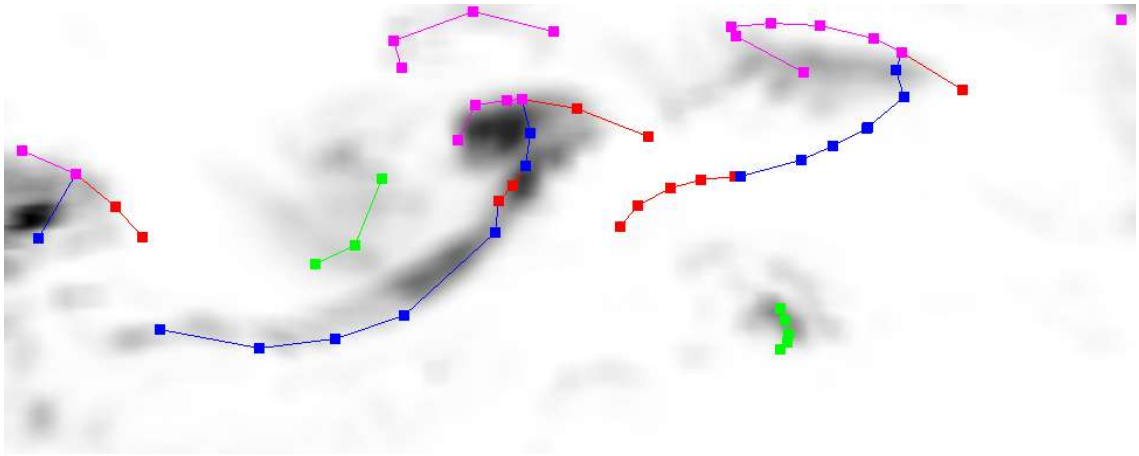

Figure 30: Precipitation compared with manually drawn fronts.

The humidity and precipitation variables should in principle be good indicators of frontal activity, as fronts tend to push moist air up in front of them, raising humidity and causing frontal precipitation (Ahrens 1994, p. 324). We see this pattern clearly in figures 28 and 30. Fronts tend to follow areas of higher humidity and precipitation. When it comes to relative humidity (figure 29), however, we see *some* of the same patterns as absolute humidity, but this effect is overshadowed by a problem shared by almost all surface level measurement: Noise. The major changes in relative humidity we can visually identify as being caused by geographical features, particularly the Sahara desert, and the Arabian and Iberian peninsulas. These features overpower any effect of the frontal situation. For this reason, *absolute* humidity is considered a critical identification factor, and it is in the variable set with a medium weighting, while *relative* humidity is not used at all.

In the case of precipitation, there is undoubtedly a correlation between this variable and the location of fronts, and it should be considered a critical feature for identifying fronts.

However, the current front detection system has some limitations that leads to precipitation currently being excluded from the variable set:

Firstly, precipitation values varies differently from the other variables. Where geopotential height for instance forms a step edge (Davis 1975) around a front, precipitation forms a roof edge, having gradients on either side of the front, rather than directly on it. This makes it counterproductive to add precipitation to the variable set, since it is uniform where the other variables vary, and vice versa. This means that precipitation would only serve to blur out the edges in the weighted average.

Secondly, the location of a front relative to the precipitation field is dependent on the front type (Ahrens 1994, p.323). A warm front will typically only have precipitation in front of it, while a cold front will be in the middle of a precipitation belt. Since the system performs front identification before classification, adding precipitation to the variable set could lead to less accurate front identification. Therefore, precipitation is the only surface measured variable that should be considered a critical identification feature, but it is excluded from the variable set used by the system due to technical limitations.

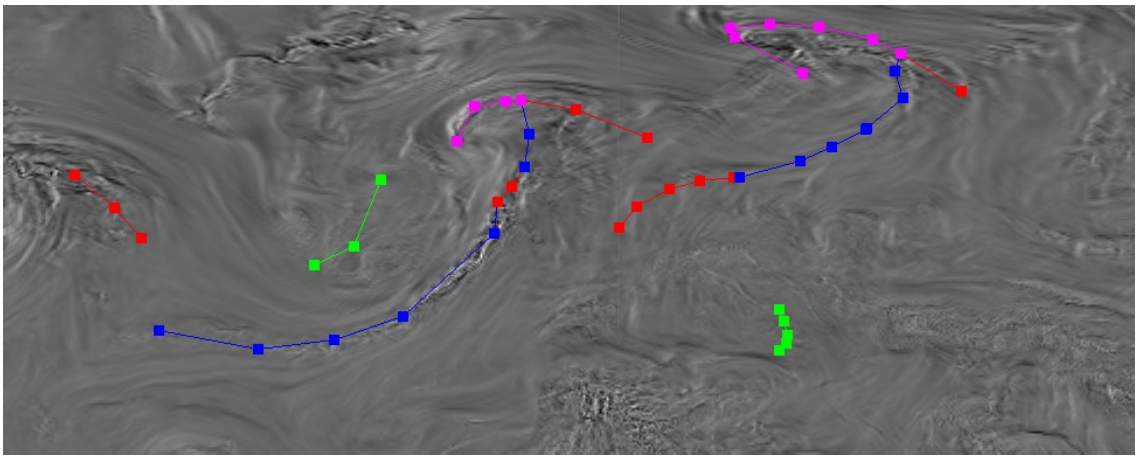

Figure 31: Vorticity compared with manually drawn fronts.

Upon immediate inspection of figure 31, vorticity seems to yield a very noisy, chaotic data set. However, we see that wherever there is a front, there are distinct value changes. Where the rest of the data set has relatively little variance, around the fronts we find both the highest and the lowest vorticity values. This makes it possible both for the human eye and for edge detection to discover the major front lines. Because of this, vorticity is a critical identification factor, and it is in the variable set, albeit with a low weighting.

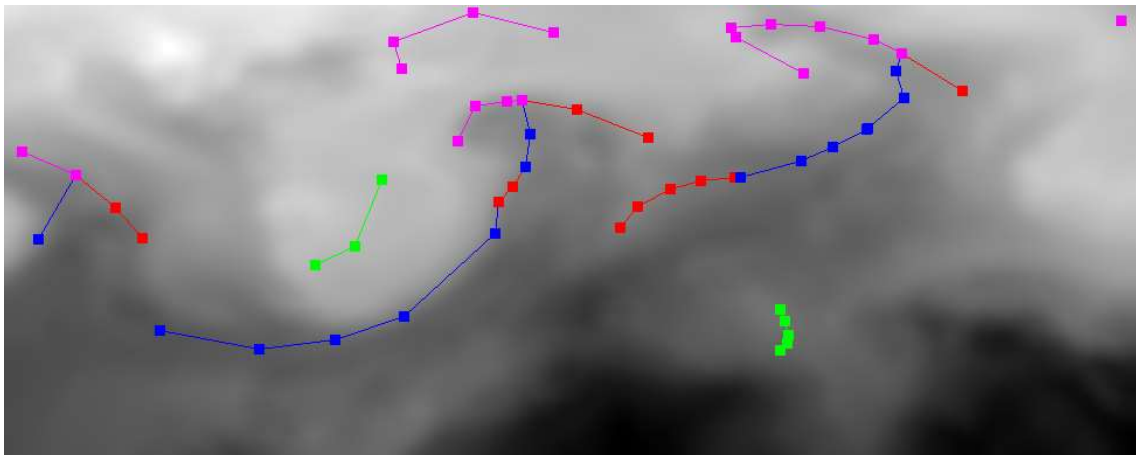

Figure 32: Temperature at 850hPa compared with manually drawn fronts.

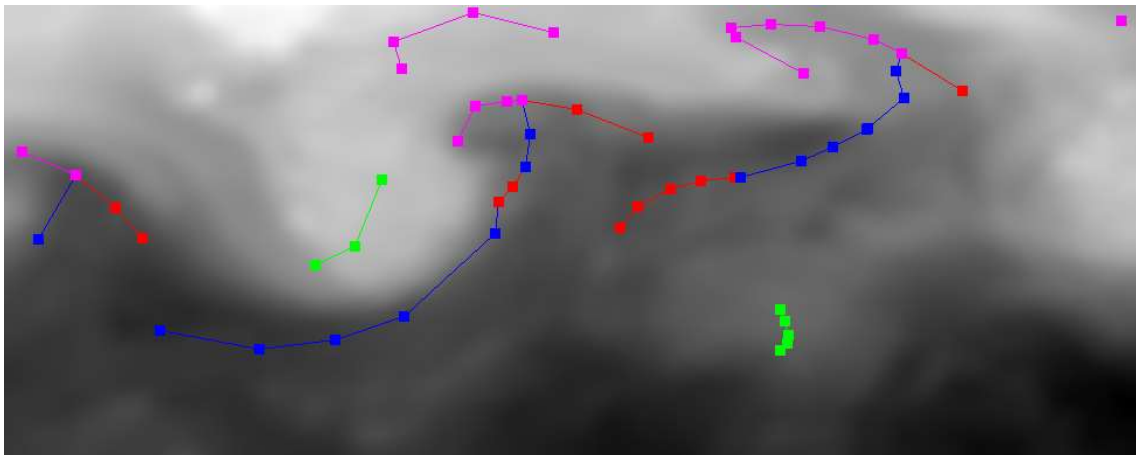

Figure 33: Temperature at 700hPa compared with manually drawn fronts.

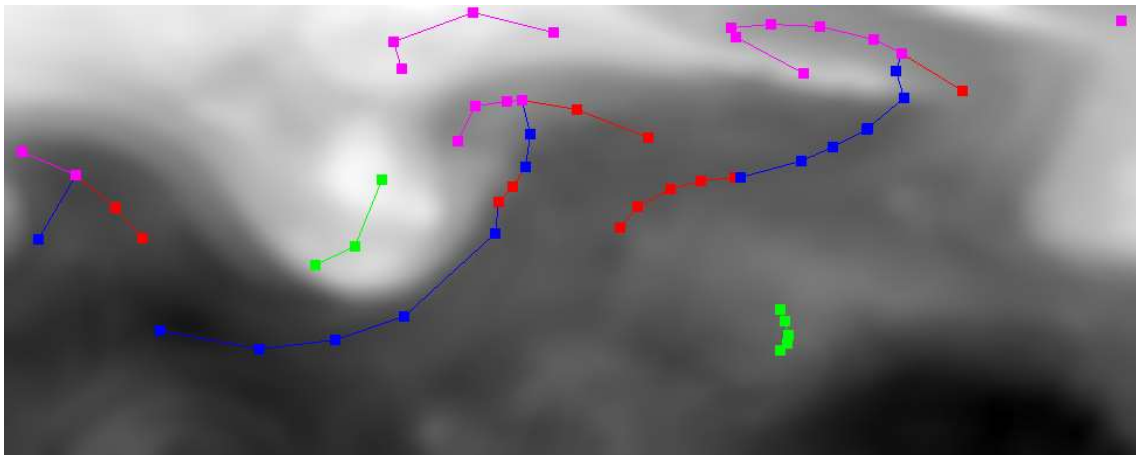

Figure 34: Temperature at 500hPa compared with manually drawn fronts.

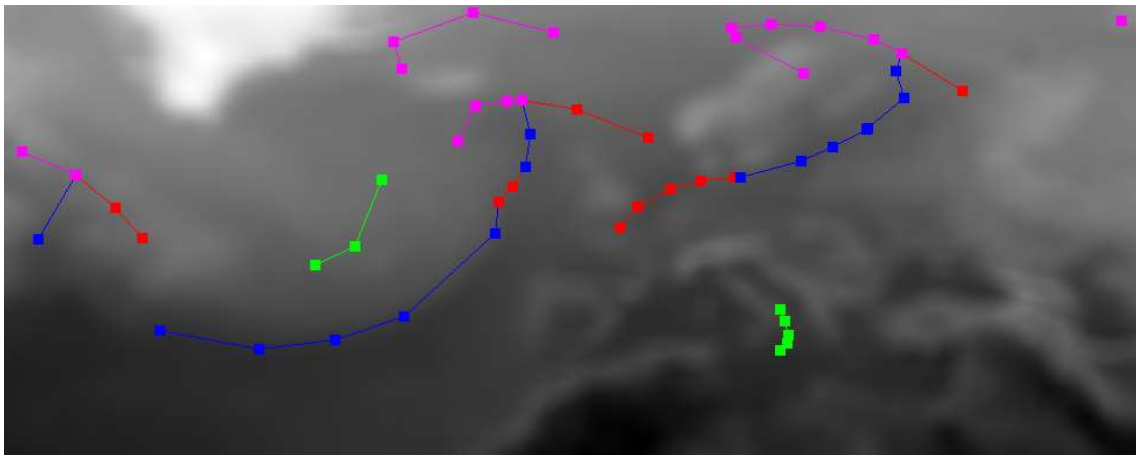

Figure 35: Sea level temperature compared with manually drawn fronts.

Temperature change is another telltale sign of a passing front. Since a front is the interface between two different air masses, and different air masses typically have different temperatures, a sudden change will often mean that a front can be found. In figures 33 and 34, we see that in the lower atmospheric levels, we can find the fronts using temperatures quite easily. We see a large, cold (white) air mass moving towards the south-east, forming a cold front, several thousand kilometers long. In the higher atmosphere (figure 32) and at the surface (figure 35) we can also recognize this cold front. Looking at the surface temperature, we see that this is probably a variable the meteorologist has used to draw this front, seeing how it follows the temperature gradient almost perfectly.

However, in the cases of figure 32 and 35, the data is a lot more noisy, and likely more difficult for an automatic edge detector to use for finding the correct features. On the surface, once again, geographical features become overpowering, meaning that an edge detector will work well at sea, but poorly over land. Higher up in the atmosphere, other factors that do not directly influence the frontal situation, add unnecessary noise to the data. For this reason, temperatures at 500 and 700hPa are considered critical features for front identification, and both are used in the variable set with a medium weight. The measurement at 700hPa is given a slightly higher weight than 500hPa, as it generally shows sharper changes along the front lines. This is also apparent in the figures above.

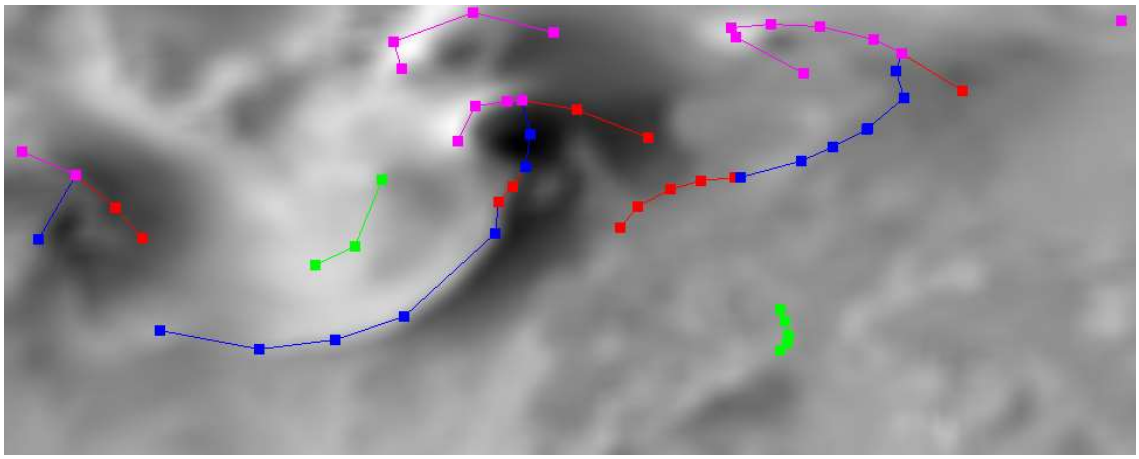

Figure 36: Sea level meridional wind velocity compared with manually drawn fronts.

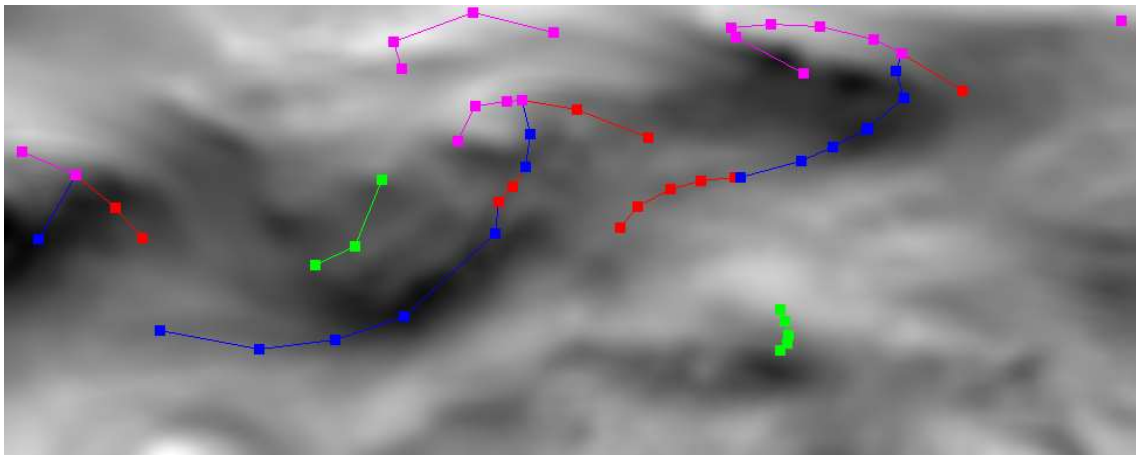

Figure 37: Zonal wind velocity at 700hPa compared with manually drawn fronts.

Wind speed and wind direction are also variables that change noticeably over a front line (Ahrens 1994, p.324), and it can theoretically be used to identify a front. However, as we can see from the examples of figures 36 and 37, wind is rarely uniform over large areas, and a lot of interregional noise can be seen. This is especially apparent at surface level, where geography plays an important role in shaping wind systems, but also in the lower atmosphere. This noisiness in wind data would only serve to blur out any noticeable front lines in the data set, and for this reason no wind variables are considered critical for front detection, and they are excluded from the variable set.

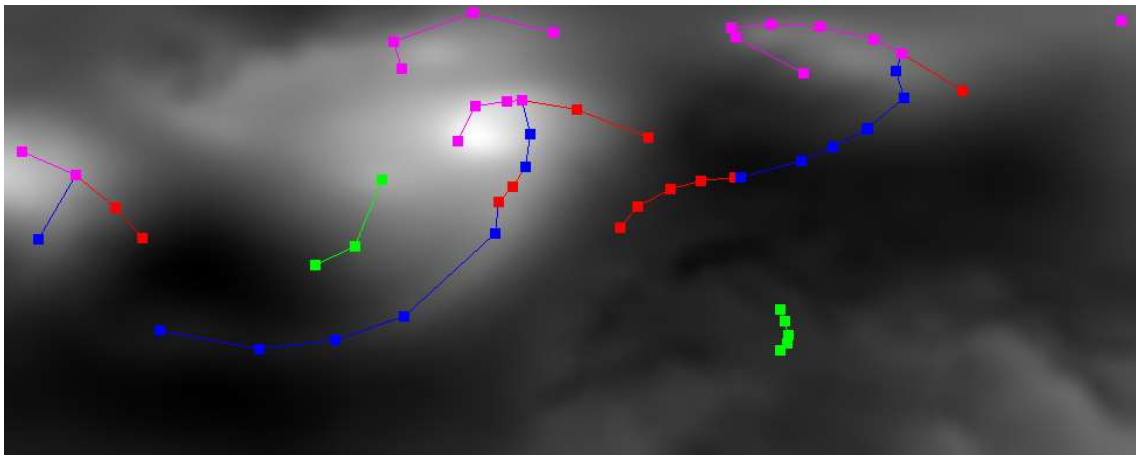

Figure 38: Sea level pressure compared with manually drawn fronts.

Air pressure is another common variable used in front detection, but it is more commonly used in the form of geopotential height, which is already in the variable set. The problem with sea level pressure is the same as with other surface level measurements; The noise from coastlines are significant and disruptive. Ignoring this noise, the pressure variable highlights the major front lines decently, but because of this shortcoming it is not considered a critical variable, and is not included in the variable set.

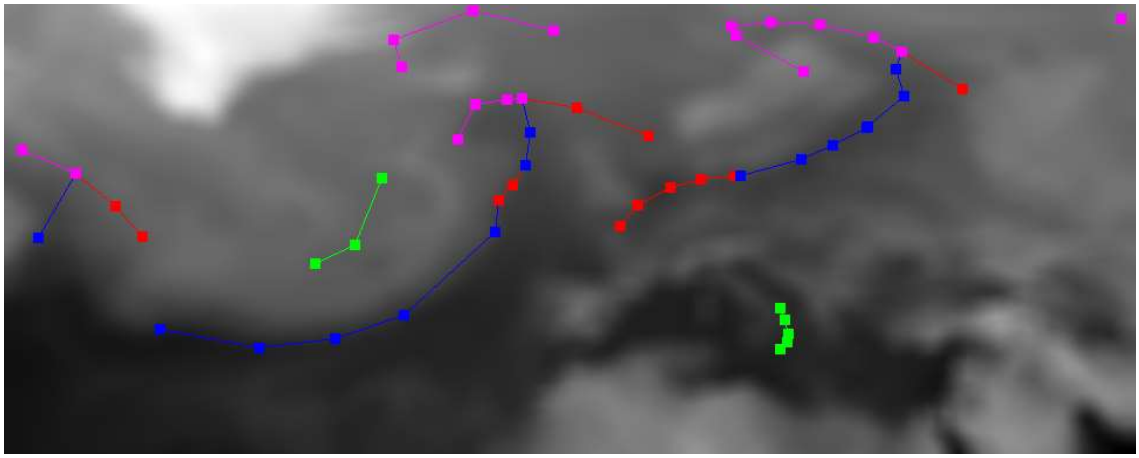

Figure 39: Sea level dew point compared with manually drawn fronts.

Dew point is the final variable that was considered. Ahrens (1994) mentions dew point as a typical indicator of fronts, and comparing figures 35 and 39, we see that it also correlates greatly with temperature measurements. For this reason, it has the same problems as the other surface level variables, and it is not considered critical for the system or included in the variable set.

7.3 The system in use

At the time of writing, StormGeo has not employed a version of the front detection system in active use. The system currently has the right input and output formats, and serves as an implementation of all the user stories defined in 5.2, but is not currently employed in a production environment. The two main reasons for this are:

1. The system is not yet sophisticated enough to give consistently good results.
2. Putting a new system into use requires time and effort, and this time and effort does not yet make sense in a simple cost-benefit analysis.

Any concrete adoption of the system is therefore outside the time frame of this Master's project.

However, there is reason to believe that some further improvements to the system (see 8.3) would be sufficient for StormGeo to justify incorporating the system into the front detection process, firstly as a decision support tool for the meteorologists, later hopefully as a full automation of the front detection procedure. Further, some of the meteorologists have commented that the current output of the system could be of some help in the front detection process already at this stage, without major improvements to the algorithm. This is promising, as it shows the value of the type of the system I have developed, even with incomplete technical implementations.

The user stories defined as the requirements for the system, were not sufficient to ensure a deployable, fully functioning product at the end of the development period. This is due to the relatively low sophistication and accuracy in implementation of some of the user stories, rather than their total omission. All six of the predefined user stories have been implemented and accepted. For this reason, the user stories still function as a basis for evaluating the system, and the development should be considered a success. The final result is a prototype that showcases the possibilities of the technologies used, but remains too unstable for full-scale, real world use. The predefined functional requirements have been met, but further development is needed to reach the accuracy required for a system in full production.

8 Discussion

In this section I will discuss the findings in the project, provide an answer for my research questions, and reflect on the utility and effectiveness of the methodological framework employed in the project. I will try to explain and/or hypothesise why the system performs as it does, and in what areas improvements would be most effective, concerning both the front detection system specifically and research into front detection in general.

8.1 Research questions

This subsection will explore and discuss the results of the project, in light of the defined research questions. This can be seen as a direct continuation of section 7.

8.1.1 Front detection success

The answer to the primary research question I believe to be fairly straightforward: Yes, it is definitely possible to automatically detect and classify fronts in a weather system. I do not, however, claim that the system developed in this project is a complete proof of this. I have previously outlined the main results of the algorithm, and they show that although the general front detection is quite strong, much is left to be desired of the accuracy and reliability of the detailed analysis. However, I will now argue that the problems with the current system are either obviously solvable, or an inherent problem for any front detection procedure, automatic or otherwise.

The three important problems of forking fronts, drawing fronts on the wrong side of the air mass, and missing fronts (such as the “invisible” occlusion in the example earlier), is likely attributed to the choice of variables used by the system. In many cases, precipitation is a very important factor in finding the right front in these tricky scenarios, and the system does not currently use precipitation data (see 7.2 and 8.1.2). Further, a more sophisticated and context-aware implementation of the front identification procedure in general would alleviate the majority of all these three problems.

The issue of too “eager” identification is directly due to a weakness in the current normalization process, and would be solved with a more robust implementation, with a dynamically calculated normalization range.

In the case of parallel fronts, it is very difficult to separate correct and incorrect front identification. Often times, several parallel fronts actually exists. In some meteorological schools it is also customary to draw them all, while others consider this parallel drawing pointless, as it gives very little additional information about the weather system, only cluttering the visual representation. Whether or not to draw many parallel fronts close together depends therefore on who you ask, rather than the quality of the front detection procedure.

The most important classification errors can be attributed to the current training set for the neural network that performs classification. This training set is rather small (around 5000 entities) and uniform. It consists of all the manually drawn front points from September 5th to December 4th 2016. 5000 entities may seem like a large amount of training data, but given how noisy the domain is, it proves insufficient. Having 10 times the amount of training data, from a data set that is more spread out in time would almost certainly increase the accuracy.

The majority of classification problems I believe we can trace back to two other factors: Firstly, the interpretation of the classified data is somewhat crude. This has the effect that single front lines are rarely split into different types even when they should have been (although this is also a result of an erroneously low classification rate for warm fronts). Secondly, since the front *identification* is not perfect, the training set for classification is based on points that are in slightly different locations than the ones the system finds. The training set consists of manually drawn fronts, which are being drawn slightly differently from the automatic ones, as we have just seen. Using a training set of one type to train on a set of a different type is not without problems. I believe, however, that with improved identification and a larger training set, the effects of this will be negligible.

The combination of these potential improvements to the current system (which are detailed further under 8.3), I believe would produce a system that consistently detects front at the same level as a human meteorologist, and thereby likely verifies my research question. Further, we have also seen that the current prototype already is able to detect fronts that the meteorologists have missed, showing that not only should it be possible to make such an automatic front detection system, but also that this system could well perform better than expert meteorologists in both time and accuracy.

8.1.2 Critical detection factors

The answer to the second research question, however, is not as clear cut as one could hope. The system I have made uses geopotential height, temperature, humidity and vorticity, while I have rejected many other variables for various reasons. My list of variables can by no means be considered an absolute definition of the critical factors for discovering a front. We know for instance that precipitation is a very good indicator of fronts, but it was not used in this project due to technical limitations of the edge detection technique. Many variables, if normalized for land/sea differences, would undoubtedly be useful as well. Most, if not all primary attributes of the atmosphere changes over a front, so almost any atmospheric measurement could be used to some extent to detect fronts.

What features are critical can therefore vary greatly. Defining a set of variables that fully and solely can be used to detect fronts anywhere in the world at any resolution, does not seem to be possible. Some features work better in colder areas than in warmer areas, some work better in the opposite. Some work better on small scale, high resolution data, while other are better at detecting major front lines. What we can say for certain is that the density (pressure), energy (temperature) and water content (humidity, precipitation) in air masses are very good indicators of fronts, and that these variables combined can be used in a concrete front detection implementation for Europe and the Northern Atlantic Ocean.

8.2 Methods

Here I will highlight some issues and noteworthy points concerning the methodological frameworks I have employed in this project, both in the research and development domain. I will discuss what features were successfully employed and which should have been changed, employed differently or not employed at all.

8.2.1 Design Science Research

Following the DSR paradigm in this project has generally been useful. Answering my research questions through concrete design and development has made it possible to detect and understand concrete challenges related to front detection, while at the same time showing how familiar techniques and technologies can be sufficient to solve the task. Since this project originates in a need directly identified by the industry, employing a research strategy that

values the practical relevance of results highly has also been favorable. In 5.1 I discussed how the project would fit into the seven guidelines for DSR defined by Hevner et al. (2004). This can now be summed up as follows:

1. **Design as an Artifact**: This guideline should be considered adhered to, trivially. The front detection system is a working, runnable computer program that produces testable result data. Although the system is not currently in active use, the end result of the project is an artefact.

2. **Problem relevance:** This is explored in 5.1. The problem is a legitimate business problem in the industry, and a satisfactory solution would be of great interest from both an academic and business perspective.

3. **Design evaluation:** The evaluation of the design is twofold. The evaluation of the output data and its quality is explored in 7.1 and 8.1.1, while the evaluation of the technical solution and implementation can be found under 8.3 and 8.4 below.

4. **Research contributions:** The main research contribution provided by the project is the link between computer vision techniques and weather data processing. Concretely, the findings discussed in 8.1, grounded in the predefined research questions, should be considered the primary contributions of this project.

5. **Research rigor:** This point is somewhat troublesome for the project. Due to the subjective nature and broad definition of fronts, comparing the results of manual and automatic front detection methods will always be subjective in nature. There is no definitive answer to the problem of front detection, and therefore, no fully quantifiable evaluation of front detection systems is possible. Expert evaluation, as I have used, should therefore be considered the most accurate form of evaluation. However, the evaluation should probably be based on the judgements of many meteorologists with different backgrounds, rather than just four from StormGeo. This would most likely yield more findings, as well as help to understand what views on fronts are shared among most meteorologists, and what views are entirely personal or culturally dependent.

6. **Design as a search process:** In this respect, the project has not been a textbook example of DSR. Ideally, the project should have started with a less strict problem

definition, and allow for more time and space to explore the domain. As the project unfolded, the only "search processes" were related to the types of technologies used to solve the defined problem, rather than to the solutions themselves. For the most part, I have simply employed the most straightforward and most basic solutions whenever possible, in order to create a working artefact within the scope of a Master's thesis. Given a bigger project and more resources, this search process of design could be emphasized much more. This is further discussed under 8.4.

7. **Communication of research:** Given the requirements surrounding a Master's degree, this communication is in total handled by the Master's thesis and subsequent presentation. The project would also lend itself well to a shorter, more focused article, to reach a broader audience of researchers and practitioners. This has not been undertaken at the time of writing.

Concerning the relationship between DSR and BSR, there does not seem to be any immediately obvious, testable hypotheses raised by this project that could easily be tested by a BSR project. The research domain is in my opinion too new and vague, and the technical implementation(s) are insufficiently sophisticated for this to be fruitful yet. More exploratory research is probably needed to better understand both the meteorological and technical domains before any useful theories can be derived. More on this in 8.4.

8.2.2 System development methodology

The motivations for my choices of system development methods are discussed under 5.2. Here I will go through the merits and shortcomings of the different techniques and methods I have used throughout development.

Of the Scrum-like elements of my methodology, some have been more successful than others. The segmentation into two week sprints was useful to remain productive throughout the development period, as continuous short-term goals pushed me to keep producing functionality at a high pace. The day-to-day planning also worked really well, seeing as the development team consisted only of myself, and no issues with coordination could occur. Finally, the retrospective meetings at the end of sprints worked well. I was able to stay disciplined and perform retrospections after each sprint, even though I was alone in this most of the time. It helped me identify sources of waste and remove these continuously.

However, the meeting activities involving other stakeholders proved difficult to maintain. This was mostly due to the amount of stakeholders and their geographical distribution. One sprint review and subsequent planning was performed with the entire group of stakeholders, but most of the time they were held with only myself, or with either a representative from StormGeo or my supervisor at the university. This was problematic at times, but for the most part the problem was well defined, and the larger review and planning meetings set the agenda for longer periods of time. For this reason, all sprints were well planned and distributed, and the workload remained consistently high throughout the development period.

The two general agile principles I focused most on maintaining was customer collaboration and responding to change. The first principle was difficult to maintain at times, since my contact points at StormGeo were, naturally, occupied with many tasks and projects, and did not always have much time to respond during the development periods. However, StormGeo are overall pleased with the results of the project, and the informal, day-to-day communication worked well. The second principle was for the most part irrelevant. The requirements for the front detection system underwent very little change between the final pre-project planning meeting and the end of the final sprint. Most of the organizational tasks involved implementing the already defined requirements, rather than making room for new ones. What did change underway was the choice of evaluation method, and this change was handled well by all parties.

The lean elements of the methodology worked extremely well, almost without fault. The waste reduction process, through sprint retrospectives, continually improved efficiency and effectiveness of the development process. The visualized workflow in Trello and single Work-in-Progress item kept the work focused and under control. New functionality was completed and tested sequentially, rather than developed all together at the same time. This was especially important during the second development cycle, when there were no sprints to control the pace.

8.3 Problems and challenges

There are a number of problems with the front detection algorithm at its current stage. This should be evident from the number of discrepancies discussed in the results (7.1). I will now present these problems and their potential solutions, in the order they present themselves in the algorithm.

8.3.1 Data resolution and variance

The current data resolution is one data point per 0.25 degrees, latitude and longitude. At 60 degrees north, this equates to about 28 km per data point north/south and about 14 km per data point east/west. It is not currently known whether this resolution is ideal for discovering front lines. Furthermore, the system has only been used on weather data from the same geographical area over a relatively short time period (May to December 2016) so some experimentation with different resolutions, locations and time frames could produce significantly different results.

8.3.2 Edge detection

Concerning the edge detection, several problems are apparent: The algorithm is static, and often fails in extending and joining the correct edges. The problem with finding edges on "the wrong side" of air masses can also be attributed largely to the edge detection process. Several improvements could be made to remedy these problems:

- Different types of edge detection, such as Laplacian or second derivative methods.
- Dynamic thresholding, ensuring better results in more heterogenous data.
- Smarter smoothing, helping the edge detection by maintaining stronger edges through smoothing.

The choice of edge detection method is discussed in 5.4.1, and the work of Perona and Malik (previously discussed in 4.4) gives good ideas on how to improve the current algorithm.

Further, the edge detection process has a more fundamental problem, namely that it is performed on an excessive abstraction of the raw data. For the sake of simplicity, the system first averages and normalizes the different variables into a single table, that is then subjected to edge detection. This is of course a significant source of error, and much detail is certainly lost during normalization. The selection of variables is treated as a linearly solvable problem,

although it most likely is not. Furthermore, the problem of absolute vs relative difference, where the gradients in the data no longer matches those of the original data files, stems from the normalization process and its lack of context awareness. There are several ways this could be improved, either by changing the order of operations in the algorithm, or by removing the normalization process altogether.

The first option would still employ edge detection as the main technique for front identification, but performed before normalization. This would involve finding the edges present in each variable first, and then normalizing the discovered edges across the different variables (finding common edges), rather than the other way around (making a common data set). With this solution, the individual contributions of each variable would be easier to differentiate and consider.

The second option also holds a lot of promise. The system could use a Convolutional Neural Network (CNN), in a manner similar to Xu et al. (explored in 4.3), rather than a purely linear normalization + edge detection algorithm. This could improve front identification greatly, by considering all possible contributing variables (rather than a defined subset) and employing a more generalized, context-sensitive algorithm. A CNN could also help detect several other features of weather systems, such as the air masses themselves, as well as the fronts between them. Such an improvement, however, would require a scope and time frame beyond that of a Master's project.

8.3.3 Line identification

The line identification process is, as explained previously, not that well grounded in known techniques and accepted theories. Therefore it has its fair share of problems. Because of my decision to make the implementation as simple and straight-forward as possible, a decision rooted in the idea of a Minimum Viable Product from lean software development, many of the concrete solutions in the line identification process are largely ad hoc. This is to say that many have been chosen purely based on intuition, and that we only know that they *seem* to work well for their task. This is a good way to work when the goal is to get functioning software out quickly, but it is not the best way of creating the most accurate solution, and certainly not the most generalizable.

The primary step of identification, finding the exact edges drawn by the edge detection, is not so problematic, but all of the other steps, including removing parallels, removing insignificant lines, joining edges, defining key points and final smoothing, all have room for improvement, in that they should be more general, better defined and more clearly motivated. Since this step of the algorithm is where some knowledge of meteorology is being employed as well, the role of this expert knowledge could be better defined and perhaps expanded on.

8.3.4 Classification

The general structure of the classification process is well functioning. Classifying each key point based on its surroundings and using this classification to label each front makes sense. However, both of the steps involved in this process are in need of refinement.

The classification of points using an artificial neural network is currently the source of most of the classification errors. Since its training set is fairly small and mostly contains data from a small time frame, the error rate in classification is unnecessarily high, especially in the case of warm fronts, which are almost never classified. The natural solution to this problem is a bigger and more heterogenous data set. Another problem is that since the system and the meteorologists find fronts at slightly different locations, using one to classify the other will in itself be a source of error, since the surroundings will differ for the two. This problem does not have an immediately available solution, but the source of the problem will be reduced with better handling of the previous stages in the algorithm. Finally, the choice of input variables for the classification is not fully explored. The current set of inputs gives a relatively low error rate compared to other, previously used sets, but there could quite possibly be other, more effective input set that are yet to be tested.

The interpretation of the classification also has room for many improvements. Like with line identification, the implementation is in many parts ad hoc, and often too naïve to handle all cases. There are two major problems: The relative high weighting of the classification of the first point on a front (since all points are compared with its predecessors), and the failure to compensate for the strength of the classification score. When five points in a line have a very strong classification, while the other two have a relatively weak classification, the five should have a stronger influence on the final classification of the front. See figure 40 for an example. Top: Classification scores for a front with seven positions. Blue score is cold front, red score is warm front, purple score is occlusion. Middle: Current classification of the same front.

Notice how the score for warm front stays fairly consistent, and usually lower than the score for cold front, but it still influences the "weak" classification of the middle two positions. Bottom: "Ideal" classification of the same front.

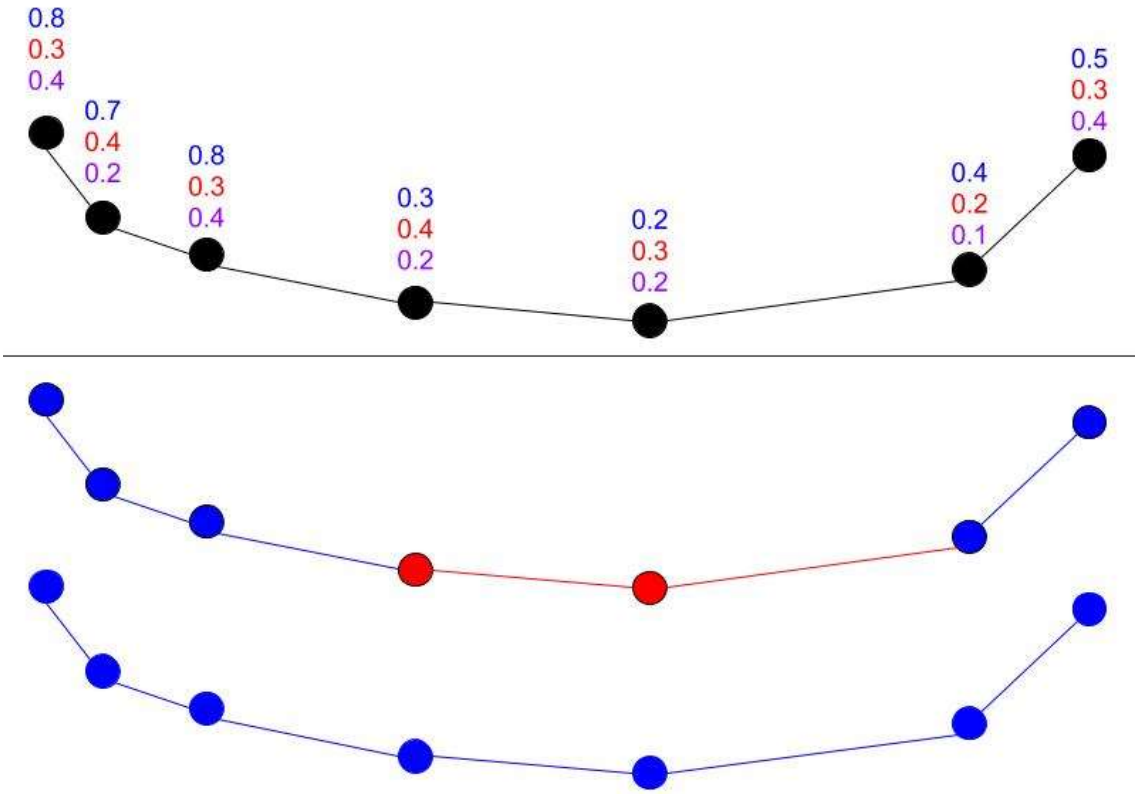

Figure 40: Example of classification.

8.3.5 Additional data sources

Currently, the system utilizes a rather small input data set to detect fronts. It uses five different weather variables from a single point in time, and no surrounding context. Considering that human meteorologists employ context extensively when drawing fronts, this is a significant source of error, and one that can be amended with improvements to the algorithm. In addition to performing edge detection "in a vacuum", the system could use more data inputs to guide the front detection process, with the help of surrounding context.

In particular, there are two types of context that are very important in front detection:

1. **Temporal context**: Where fronts have been drawn in previous time steps are usually solid indicators of where to find fronts in the current time step.

2. **Meteorological context**: Knowledge of the nature of fronts and how they form and evolve are often useful in the front detection process.

If the system could use more cues from these context domains effectively, it could serve to reduce inaccurate and erroneous detection significantly.

The temporal aspect is undoubtedly very important. Although the primary concern of this project has been to discover patterns in *spatially* distributed data, the influence of the *temporal* in weather analysis should not be underestimated. When the meteorologists at StormGeo draw fronts, they do so on the basis of where the fronts were drawn in the previous time step. Fronts as entities form and evolve over time, so following their patterns from time step to time step could be helpful in predicting their locations in future time steps, both faster and more precisely. At the same time, one should be careful not to be carried away in this direction since, in theory, a front should be detectable solely on the basis of the weather *as it is*. Too much focus on previous time steps could to lead to new and different erroneous classifications.

While the time aspect would be useful to improve speed and precision, using meteorological knowledge could help reducing the likelihood of incorrect detection. As we have seen in 7.1, the system often finds fronts that could not possibly exist in the real world. During line identification, we have used some knowledge about fronts, to make sure that no candidate fronts have heterogenous curvature, since this almost never occurs in a natural weather system. Other such general heuristics could help the system choose the the “correct” candidate fronts, and remove many of the erroneous ones. Naturally, the more heuristical knowledge the system incorporates, the less general it becomes, but as a tool to improve system performance, employing more rule-based selection when finding candidate fronts could be very useful.

8.4 Future work

There are a number of things related to my project that requires further study. Since the project is an exploratory study in a domain that is largely unexplored, there are both exploratory and behavioural (or theory-building) studies that could be performed as a continuation of, or tangential to, this project.

On the exploratory side, a natural progression would be to implement the improvements suggested in 8.3, and see how this affects the overall results. Further, making the algorithm more general, and testing it on different locations, different resolutions and more and different weather variables could be interesting. Such improvements could hopefully also help to cement my conclusions about the feasibility and merits of automatic front detection, by actually displaying its full potential.

Another interesting direction would be to explore other novel uses of edge detection and other computer vision techniques, on non-images. In my project, I have shown that edge detection can be used successfully on a purely numerical representation of weather data, rather than an image representation. In some entity/feature recognition tasks, working with images can be an unnecessary complication. A good example of this is the earlier attempt at automated front detection (Ullman and Cornillon, 2000). Opening up computer vision to more diverse spatial problems could help in discovering better solutions to existing challenges, with tools that are already well-known and available.

On the behavioural side, it could be useful to study a real world, full-scale implementation of an automated front detection system. This could reveal what the most useful features of such a system are, and help us learn more about how automation changes business processes, both generally and in weather analysis especially. Such a study is for course reliant on a better, more refined algorithm for front detection than the one described in this thesis.

Finally, this project highlights a general problem with front detection, namely how difficult it is to evaluate. Even though fronts have a clear definition (the interface between different air masses), this is not enough to systematically evaluate front detection. All meteorologists will draw fronts slightly differently. This is because of two important problems:

1. We do not have perfectly accurate data to pinpoint the precise location of fronts.
2. What constitutes two different air masses, as well as the interface between them, is not always universally agreed upon.

Because of this, front detection is never completely verifiable, since we cannot determine definitively whether or not there was a front at the point in time and space that we predicted. As apparent from the presentation of the results of my system, it is still possible to make judgements about what is good and bad front detection, but this will always require expert knowledge and human judgement.

A research project that tries to pin down an objective, general and data driven definition of different types of fronts could therefore be extremely interesting and useful. Given that the field could agree on such absolute definitions, automated detection systems would be much easier to design, and much easier to trust. Of course, such a definition may not be possible to make, but the possibility should certainly be explored.

9 Conclusions

For my Master's project, in the cross-section of weather data analysis, computer vision, system development and Design Science Research, I have developed a system for automatic front detection, in collaboration with weather service provider StormGeo. Throughout this thesis, I have presented and described this research project. I have given an overview of the problem domain and the research questions to be explored. I have detailed important findings in the literature, related to the project. I have explained and motivated the methodological framework of the project, from both a research and software development perspective. I have described in detail the front detection system and the algorithms it employs, and I have presented all results of the project and discussed these.

In the end, I am left with an information system that shows that automated front detection is, most likely, possible. This system is not yet sophisticated enough to outperform human meteorologists, but I have found that with targeted improvements, this level of sophistication is likely achievable. I have also found that a large number of weather variables can be successfully used in front detection, but that no definitive set of variables can be defined at this point.

This project has shown that already familiar techniques in computer vision and machine learning can be used successfully in weather data analysis to improve and extend the state of the art. At the same time, it has shown that the fuzzy nature of weather phenomena, like fronts, make the domain difficult to automate and evaluate. Further, we see that much work remains, both in terms of technical development and academic endeavor.

Finally, I am pleased with the overall results of this Master's project and thesis, and I hope to see this research area further explored in the coming years. I hope to have helped shed some light on the current state of automation in weather analysis in general, and front detection especially. I hope that new insights into technical implementation and domain-specific problems and challenges, highlighted in this thesis, will help pave the way for even better automated systems in the coming years. And lastly, I hope that researchers and practitioners continue to improve the technical and business processes involved in weather analysis, with the help of expert knowledge in both meteorology and artificial intelligence, as I hope to have done in the preceding.

References

Ahrens, C. D. (1994) Meteorology Today - An Introduction to Weather, Climate and the Environment (5th ed.). St. Paul, MN. West Publishing Company.

The Apache Software Foundation (2016). Apache Commons Net. Retrieved from https://commons.apache.org/proper/commons-net/

Beck, K. e. a. (2001). The Agile Manifesto.

Bjerknes, J., and Solberg, H. (1922). Life Cycle of Cyclones and the Polar Front Theory of Atmospheric Circulation. Monthly Weather Review, 50(9).

Bjerknes, V. (1904). Das Problem der Wettervorhersage, betrachtet vom Standpunkte der Mechanik und der Physik. Meteorologische Zeitschrift, 21, 1-7.

Bratko, I. (1993). Machine learning in artificial intelligence. Artificial Intelligence in Engineering, 8(3), 159-164.

Encyclopedia Britannica (1998). Specific Humidity in Encyclopedia Britannica. Retrieved from https://global.britannica.com/science/specific-humidity

Browning, K. A., Collier, C. G., Larke, P. R., Menumir, P., Monk, G. A., and Owens, R. G. (1982). On the Forecasting of Frontal Rain using a Weather Radar Network. Monthly Weather Review, 110(6).

Canny, J. F. (1986). A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6), 679-698.

Catté, F., Lions, P.-L., Morel, J.-M., and Coll, T. (1992). Image Selective Smoothing and Edge Detection by Nonlinear Diffusion. SIAM Journal on Numerical Analysis, 29(1).

Cohn, M. (2004). User Stories Applied: For Agile Software Development. Boston, MA: Pearson Education, Inc.

Davies, E. R. (2005). Machine Vision (3rd ed.). San Francisco, CA: Elsevier.

Davis, L. S. (1975). A survey of edge detection techniques. Computer Graphics and Image Processing, 4(3).

Ding, J., and Goshtasby, A. (2001). On the Canny edge detector. Pattern Recognition, 34(3), 721-725.

The Eclipse Foundation (2016). Eclipse IDE for Java Developers. Retrieved from https://eclipse.org/downloads/packages/eclipse-ide-java-developers/neon1a

Elkins, K. (2015). Experts predict robots will take over 30% of our jobs by 2025 — and white-collar jobs aren't immune. Business Insider. Retrieved from http://www.businessinsider.com/experts-predict-that-one-third-of-jobs-will-be-replaced-by-robots-2015-5?r=US&IR=T&IR=T

Fan, F., Bell, K., and Infield, D. (2016). Probabilistic weather forecasting for dynamic line rating studies. Paper presented at the Power Systems Computation Conference.

Ghosh, S., Nag, A., Biswas, D., Singh, J. P., Biswas, S., Sarkar, D., and Sarkar, P. P. (2011). Weather Data Mining using Artificial Neural Network. Paper presented at the Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE, Trivandrum.

Gibara, T. (2011). Canny Edge Detector Implementation. Retrieved from http://www.tomgibara.com/computer-vision/canny-edge-detector

Heaton, J. (2016). Encog Machine Learning. Retrieved from http://www.heatonresearch.com/encog/

Hevner, A., and Chatterjee, S. (2010). Design Science Research in Information Systems, Design Research in Information Systems - Theory and Practice: Springer US.

Hevner, A., March, S., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. MIS Quarterly, 28(1).

Hooper, D. (2002). Wind vector notation conventions. Retrieved from http://mst.nerc.ac.uk/wind_vect_convs.html

Hopkins, J., Challenor, P., and Shaw, A. G. P. (2010). A New Statistical Modeling Approach to Ocean Front Detection from SST Satellite Images. Journal of Atmospheric and Oceanic Technology, 27(1).

Hoskins, B. J., & Hodges, K. I. (2002). New Perspectives on the Northern Hemisphere Winter Storm Tracks. Journal of the Atmospheric Sciences, 59(6), 1041-1061.

IkamusumeFan. (2015). Cappadocia Gaussian Blur.svg. Retrieved from https://en.wikipedia.org/wiki/File:Cappadocia_Gaussian_Blur.svg

Karstens, C. D., Stumpf, G., Ling, C., Hua, L., Kingfield, D., Smith, T. M., . . . Rothfusz, L. P. (2014). Evaluation of a Probabilistic Forecasting Methodology for Severe Convective Weather in the 2014 Hazardous Weather Testbed. Weather and Forecasting, 30(6).

Lamb, H. (1972). British Isles Weather Types and a register of the daily sequence of circulation patterns, 1861-1971. Geophysical Memoir, 116.

Lazos, D., Sproul, A. B., and Kay, M. (2015). Development of hybrid numerical and statistical short term horizon weather prediction models for building energy management optimisation. Building and Environment, 90.

Lee, C.-K., Kim, J., Kim, S. K. (2015). Development and application of a weather data service client for preparation of weather input files to a crop model. Computers and Electronics in Agriculture, 114, 237-246.

Lee, R., Liu, J. (2004). iJADE WeatherMAN: a weather forecasting system using intelligent multiagent-based fuzzy neuro network. Paper presented at the IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews.

de Lima, G. R. T., Stephany, S. (2013). A new classification approach for detecting severe weather patterns. Computers and Geoscience, 57, 158-165.

MacLoone, J. (2010). Ääretuvastuse näide.png. Retrieved from https://en.wikipedia.org/wiki/File:%C3%84%C3%A4retuvastuse_n%C3%A4ide.png

Marrone, P. (2004). Joone - Java Object Oriented Neural Engine. Retrieved from http://www.jooneworld.com/

Nikitina, N., Kajko-Mattsson, M., and Stråle, M. (2012). From scrum to scrumban: a case study of a process transition. Paper presented at the International Conference on Software and System Process.

North Carolina, State Climate Office of (2010). Geopotential Height. Retrieved from https://climate.ncsu.edu/images/climate/enso/geo_heights.php

Pagano, T. C., Pappenberger, F., Wood, A. W., Ramos, M.-H., Persson, A., and Anderson, B. (2016). Automation and human expertise in operational river forecasting. Wiley Interdisciplinary Reviews: Water, 3(5).

Perona, P., and Malik, J. (1990). Scale-space and edge detection using anisotropic diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(7).

Peterson, D. (2015). What is Kanban? available at kanbanblog.com

Rahmani, A., Afra, S., Zarour, O., Addam, O., Koochakzadeh, N., Kianmehr, K., Alhajj, R., Rokne, J. (2014). Graph-based approach for outlier detection in sequential data and its application on stock market and weather data. Knowledge-Based Systems, 61, 89-97.

Russell, S., Norvig, P. (2014). Artificial Intelligence - A Modern Approach (3rd ed.). Harlow, UK: Pearson.

Samarchyan, S. (2014). What is a minimum viable product? Retrieved from https://www.quora.com/What-is-a-minimum-viable-product

Schwaber, K., and Sutherland, J. (2013). The Scrum Guide.

Sevarac, Z. (2016). Java Neural Network Framework. Retrieved from http://neuroph.sourceforge.net/

Shaw, A. G. P., and Vennel, R. (2000). A Front-Following Algorithm for AVHRR SST Imagery. Remote Sensing of Environment, 72(3).

StormGeo (2016). About StormGeo. from http://www.stormgeo.com/about/weather-forecast-company-about-stormgeo/.

taheretaheri. (2010). Benchmarking and Comparing Encog, Neuroph and JOONE Neural Networks. Retrieved from http://www.codeproject.com/Articles/85487/Benchmarking-and-Comparing-Encog-Neuroph-and-JOONE

Trello (2016). About Trello. Retrieved from https://trello.com/about

Ullman, D. S., and Cornillon, P. C. (2000). Evaluation of Front Detection Methods for Satellite-Derived SSt Data Using In Situ Observations. Journal of Atmospheric and Oceanic Technology, 33(8).

Unidata. (2016). NetCDF Java. Retrieved from http://www.unidata.ucar.edu/software/thredds/current/netcdf-java/

Vascák, J., Jaksa, R., Koscak, J., Adamcák, J. (2015). Local weather prediction system for a heating plant using cognitive approaches. Computers in Industry, 74, 110-118.

Wilby, R. (1995). Simulation of precipitation by weather pattern and frontal analysis. Journal of Hydrology, 173(1-4).

Wolfram. (2008). Wolfram Language & System Documentation Center NetCDF: Wolfram.

Xu, J., Luo, X., Wang, G., Gilmore, H., Madabhushi, A. (2016). A Deep Convolutional Neural Network for Segmenting and Classifying Epithelial and Stromal Regions in Histopathological Images. Neurocomputing.

Appendix A: Full Result Set

Kildefil	Person	Tema	Kommentar
2017-01-10-1200	Beathe	Front ikke tegnet	Varmfront i Atlanteren i forbindelse med nytt lavtrykk ikke analysert i den automatiske deteksjonen.
2017-01-09-0000	Beathe	Front tegnet på feil sted	Den automatiske analysen har kaldfront heile veien, og fronten ligger lenger vest.
2017-01-11-0000	Beathe	Front tegnet på feil sted	Gammel okkludert front over Finnmark og Troms analysert delvis likt av meteorolog og automatisk. Den automatiske analysen mangler varmfront, og har derimot kaldfront over Nordsjøen øst for meteorologens varmfront. En annen kaldfront ligger vest for meteorologens kaldfront over Irland. De automatiske frontene ser altså ut til å ligge forskyvet østover, og er delvis av feil karakter.
2017-01-08-1200	Beathe	Front tegnet riktig	Meteorologens analyse ser litt rar ut med kaldfront sørover til Vestlandet, og den automatiske er her kanskje hakket bedre med kaldfront fra Island mot Nordland.
2017-01-10-0000	Beathe	Front tegnet riktig	Mye bra med frontsystem som strekker seg rundt et lavtrykk i Norskehavet og sørvestover over Vestlandet mot NV-Frankrike.
2017-01-10-1200	Beathe	Front tegnet riktig	Okkludert front over Østlandet omtrent samme plassering av meteorolog og automatisk analyse.
2017-01-10-1200	Beathe	Front tegnet riktig	Okkludert front over Skotland/England omtrent lik.
2017-01-08-1200	Beathe	Klassifisering	Varmfront mangler generelt.
2017-01-09-0000	Beathe	Klassifisering	Varmfront mangler generelt.
2017-01-09-1200	Beathe	Klassifisering	Den automatiske analysen har kaldfront heile veien, og fronten ligger lenger nord over N-England og Irland.
2017-01-09-1200	Beathe	Klassifisering	Varmfront mangler generelt.
2017-01-10-0000	Beathe	Klassifisering	I den automatiske analysen er fronten derimot okkludert hele veien, mens meteorologen har kaldfront over Vestlandet mot kontinentet.
2017-01-10-0000	Beathe	Klassifisering	Varmfront mangler generelt.
2016-09-12-1200	Cecilie	Ekstra front fra system	Ekstrafront over UK.

2016-09-12-1200	Cecilie	Ekstra front fra system	System over Russland funnet, men front dratt lenger nord enn meteorologens forslag.
2016-09-12-1200	Cecilie	Ekstra front fra system	Ellers litt "ekstrafronter" her og der hvor det egentlig ikke skjer noe.
2016-09-12-1200	Cecilie	Front tegnet på feil sted	Halen på fronten ligger for langt vest.
2016-09-12-1200	Cecilie	Front tegnet på feil sted	System ved Grønland funnet, halen på fronten dratt for langt vest.
2016-10-12-0000	Cecilie	Front tegnet på feil sted	Lavtrykk ved Portugal og SV for Island. Front tegnet fra det ene lavtrykket til det andre rundt høytrykket som ligger mellom. De to lavtrykkene er ikke identifisert hver for seg som meteorologen har tegnet.
2016-10-12-0000	Cecilie	Front tegnet på feil sted	Kaldfront funnet på Balkan, men dratt helt vest til Frankrike.
2016-09-12-1200	Cecilie	Front tegnet riktig	System ved Island funnet
2016-09-12-1200	Cecilie	Klassifisering	Meteorolog har markert tråg i Atlanteren, systemet har markert dette med en okkludert. Den viser tegn på at "det er noe der".
2016-10-12-0000	Cecilie	Klassifisering	Tråg i Barentshavet nær Kola, markert som okkludert.
2016-10-12-0000	Cecilie	Klassifisering	Ingen varmfront eller okklusjon funnet.
???	Frode	Ekstra front fra system	Lang buet front tegnet for langt rundt
2016-09-06 0000	Frode	Ekstra front fra system	Systemet finner en front nord-sør i midten av bildet. Ikke tegnet opp av meteorolog.
2016-09-07 0000	Frode	Ekstra front fra system	Flere paralelle fronter. Kun én tegnet opp av meteorolog.
2016-10-06 1200	Frode	Ekstra front fra system	Små forskjeller, veldig få fronter tegnet.
2016-09-07 1200	Frode	Front ikke tegnet	Okklusjon i nord, ikke mulig å se i datasett. Veldig like temperaturer. Nedbør nyttig.
2016-10-24 0000	Frode	Front ikke tegnet	Hull i okklusjon.
???	Frode	Front tegnet på feil sted	Systemet finner okklusjon på "feil side", skal være på utsiden

2016-09-06 1200	Frode	Front tegnet på feil sted	Systemet bommer på en okklusjon, tegner videre ut i intet.
2016-10-24 1200	Frode	Front tegnet riktig	Nesten funnet et perfekt klassisk system i vest.
2016-11-30 0000	Frode	Front tegnet riktig	Lange, store fronter fungerer bra
???	Frode	Klassifisering	Varmfronter blir klassifisert som fortsettelse av kaldfront
2016-11-30 1200	Frode	Klassifisering	Kaldfronter blir klassifisert riktig. Ellers mye rart
2016-12-03-1200	Ina	Ekstra front fra system	Front over Nord-Afrika via Spania og til området vest av Irland er feil, den finnes ikke.
2016-12-04-1200	Ina	Ekstra front fra system	Frontsystem tegnet like ved Irland, det finnes ikke hos met.
2016-12-31-1200	Ina	Ekstra front fra system	EN del okkluderte fronter over/ved Grønland, dette er inni ett høytrykk og feil.
2016-12-31-1200	Ina	Ekstra front fra system	Frontsystem øst i Middelhavet, dedektert med altfor mange fronter.
2016-12-02-1200	Ina	Front ikke tegnet	Gamle fronter/okkluderte fronter mangler over øst/nord Skandianvia
2016-12-04-1200	Ina	Front ikke tegnet	Varmfront mangler over Sverige
2016-12-27-1200	Ina	Front ikke tegnet	Lavtrykk med fronter over Baltikum, fronter ikke oppdaget av auto.
2016-12-02-1200	Ina	Front tegnet på feil sted	En kald front over Spania/Portugal er tegnet uten at det skal være noe der. Det er ett frontsystem lengre vest.
2016-12-03-1200	Ina	Front tegnet på feil sted	Auto har tegnet en lang kald front fra Svartehavet til Sverige. EN kald front skal det være over Svartehavet, litt lengre nord, men den skal ikke strekke seg til Sverige.
2016-12-04-1200	Ina	Front tegnet på feil sted	Kaldfront over Svartehavet er for lang igjen, plassering over SVartehavet ganske riktig, tror meteorologen har tegnet den litt for langt nord
2016-12-04-1200	Ina	Front tegnet på feil sted	Met oppdaget system vest-sørvest for Portugal. Auto har markert noen fronter, men feil plassering og klassifisering
2016-12-27-1200	Ina	Front tegnet riktig	Kaldfront funnet like nord for Svartehavet, men igjen tegnet for lang.

2016-12-31-1200	Ina	Front tegnet riktig	Kaldfront strekker seg fra nordvest Russland via Skandinavia mot SKottland. Riktig klassifisering, men feil vei. Litt for langt nord.
2016-12-02-1200	Ina	Klassifisering	Lang okkludertfront. Fronten er stort sett på rett sted, men feil klassifisering. Litt feil plassering over Norge, her deles fronten i to system, noe autofronten ikke har gjort.
2016-12-03-1200	Ina	Klassifisering	Varmfront mangler generelt.
2016-12-03-1200	Ina	Klassifisering	I velorganiserte lavtrykk der fronter skifter fra kaldt, til varmt til kaldt osv, tegner auto ofte en lang okkludert eller flere mindre oklkuderte. Det er blitt gjort med front som streekker seg fra Norge til Island og så sørvest i Nord-Atlanteren.
2016-12-03-1200	Ina	Klassifisering	System vest-sørvest for Portugal er dedektert, men feil klassifisering.
2016-12-04-1200	Ina	Klassifisering	Front som strekker seg fra Nord-Nordland mot Island, feil klasssifisering og litt for langt nord.
2016-12-27-1200	Ina	Klassifisering	Klassisk frontsystem ved Island, dårlig fanget av auto. Varmfront mangler, delvis merket av en okkludert, men feil plassert. Kaldfronten er funnet, men feil vei og litt for langt vest.
2016-12-31-1200	Ina	Klassifisering	Ubestemmelig frontsystem vest av Portugal. vanskelig å klassifisere, auto har market systemet emd en okkludert

Appendix B: Modified Canny Edge Detector

```
/**
* <p><em>This software has been released into the public domain.
* <strong>Please read the notes in this source file for additional information.
* </strong></em></p>
*
* <p>This class provides a configurable implementation of the Canny edge
* detection algorithm. This classic algorithm has a number of shortcomings,
* but remains an effective tool in many scenarios. <em>This class is designed
* for single threaded use only.</em></p>
*
* Modified by Simen S. Karlsen to work on short[][]
*
* @author Tom Gibara
*
*/

public class CannyEdgeDetector {

   //Statics
   private final static float GAUSSIAN_CUT_OFF = 0.005f;
   private final static float MAGNITUDE_SCALE = 100F;
   private final static float MAGNITUDE_LIMIT = 1000F;
   private final static int MAGNITUDE_MAX = (int) (MAGNITUDE_SCALE *
MAGNITUDE_LIMIT);

   //Fields
   private int height;
   private int width;
   private int picsize;
   private short[] data;
   private int[] magnitude;
   private short[][] sourceTable;
   private short[][] edgesTable;

   private float gaussianKernelRadius;
   private float lowThreshold;
   private float highThreshold;
   private int gaussianKernelWidth;

   private float[] xConv;
   private float[] yConv;
   private float[] xGradient;
   private float[] yGradient;

   /**
    * Constructs a new detector with default parameters.
    */
   public CannyEdgeDetector() {
      lowThreshold = 2.5f;
      highThreshold = 7.5f;
      gaussianKernelRadius = 2f;
      gaussianKernelWidth = 16;
```

```
}

/**
 * The data table used by this detector to
 * generate edges.
 *
 * @return the source table, or null
 */
public short[][] getSourceTable() {
   return sourceTable;
}

/**
 * Specifies the data table in which edges
 * will be detected. A source table must be set before the process method
 * is called.
 *
 * @param a data table
 */
public void setSourceTable(short[][] table) {
   sourceTable = table;
}

/**
 * Obtains a data table containing the edges detected during the last call to
 * the process method.
 *
 * @return a data table containing the detected edges, or null if the process
 * method has not yet been called.
 */
public short[][] getEdgesTable() {
   return edgesTable;
}

/**
 * Sets the edges table. Calling this method will not change the operation
 * of the edge detector in any way. It is intended to provide a means by
 * which the memory referenced by the detector object may be reduced.
 *
 * @param edgesTable expected (though not required) to be null
 */
public void setEdgesTable(short[][] edgesTable) {
   this.edgesTable = edgesTable;
}

/**
 * The low threshold for hysteresis. The default value is 2.5.
 *
 * @return the low hysteresis threshold
 */
public float getLowThreshold() {
   return lowThreshold;
}

/**
 * Sets the low threshold for hysteresis. Suitable values for this parameter
 * must be determined experimentally for each application. It is nonsensical
 * (though not prohibited) for this value to exceed the high threshold value.
```

```
 *
 * @param threshold a low hysteresis threshold
 */
public void setLowThreshold(float threshold) {
   if (threshold < 0) throw new IllegalArgumentException();
   lowThreshold = threshold;
}

/**
 * The high threshold for hysteresis. The default value is 7.5.
 *
 * @return the high hysteresis threshold
 */
public float getHighThreshold() {
   return highThreshold;
}

/**
 * Sets the high threshold for hysteresis. Suitable values for this
 * parameter must be determined experimentally for each application. It is
 * nonsensical (though not prohibited) for this value to be less than the
 * low threshold value.
 *
 * @param threshold a high hysteresis threshold
 */
public void setHighThreshold(float threshold) {
   if (threshold < 0) throw new IllegalArgumentException();
   highThreshold = threshold;
}

/**
 * The number of pixels across which the Gaussian kernel is applied.
 * The default value is 16.
 *
 * @return the radius of the convolution operation in pixels
 */
public int getGaussianKernelWidth() {
   return gaussianKernelWidth;
}

/**
 * The number of pixels across which the Gaussian kernel is applied.
 * This implementation will reduce the radius if the contribution of pixel
 * values is deemed negligable, so this is actually a maximum radius.
 *
 * @param gaussianKernelWidth a radius for the convolution operation in
 * pixels, at least 2.
 */
public void setGaussianKernelWidth(int gaussianKernelWidth) {
   if (gaussianKernelWidth < 2) throw new IllegalArgumentException();
   this.gaussianKernelWidth = gaussianKernelWidth;
}

/**
 * The radius of the Gaussian convolution kernel used to smooth the source
 * image prior to gradient calculation. The default value is 16.
 *
 * @return the Gaussian kernel radius in pixels
```

```
 */
public float getGaussianKernelRadius() {
   return gaussianKernelRadius;
}

/**
 * Sets the radius of the Gaussian convolution kernel used to smooth the
 * source image prior to gradient calculation.
 *
 * @return a Gaussian kernel radius in pixels, must exceed 0.1f.
 */
public void setGaussianKernelRadius(float gaussianKernelRadius) {
   if (gaussianKernelRadius < 0.1f) throw new IllegalArgumentException();
   this.gaussianKernelRadius = gaussianKernelRadius;
}

//Runs edge detection
public void process() {
   //Find size
   width = sourceTable[0].length;
   height = sourceTable.length;
   picsize = width * height;
   //Get data
   initArrays();
   readData();
   //Perform smoothing
   computeGradients(gaussianKernelRadius, gaussianKernelWidth);
   int low = Math.round(lowThreshold * MAGNITUDE_SCALE);
   int high = Math.round( highThreshold * MAGNITUDE_SCALE);
   //Edge detection
   performHysteresis(low, high);
   thresholdEdges();
   //Write to new data table
   writeEdges(data);
}

//Private utility methods
private void initArrays() {
   if (data == null || picsize != data.length) {
       data = new short[picsize];
       magnitude = new int[picsize];

       xConv = new float[picsize];
       yConv = new float[picsize];
       xGradient = new float[picsize];
       yGradient = new float[picsize];
   }
}

private void computeGradients(float kernelRadius, int kernelWidth) {
   //Generate the gaussian convolution masks
   float kernel[] = new float[kernelWidth];
   float diffKernel[] = new float[kernelWidth];
   int kwidth;
   for (kwidth = 0; kwidth < kernelWidth; kwidth++) {
       float g1 = gaussian(kwidth, kernelRadius);
       if (g1 <= GAUSSIAN_CUT_OFF && kwidth >= 2) break;
       float g2 = gaussian(kwidth - 0.5f, kernelRadius);
```

```
            float g3 = gaussian(kwidth + 0.5f, kernelRadius);
            kernel[kwidth] = (g1 + g2 + g3) / 3f / (2f * (float) Math.PI *
kernelRadius * kernelRadius);
            diffKernel[kwidth] = g3 - g2;
        }

        int initX = kwidth - 1;
        int maxX = width - (kwidth - 1);
        int initY = width * (kwidth - 1);
        int maxY = width * (height - (kwidth - 1));

        //Perform convolution in x and y directions
        for (int x = initX; x < maxX; x++) {
            for (int y = initY; y < maxY; y += width) {
                int index = x + y;
                float sumX = data[index] * kernel[0];
                float sumY = sumX;
                int xOffset = 1;
                int yOffset = width;
                for(; xOffset < kwidth ;) {
                    sumY += kernel[xOffset] * (data[index - yOffset] + data[index +
yOffset]);
                    sumX += kernel[xOffset] * (data[index - xOffset] + data[index +
xOffset]);
                    yOffset += width;
                    xOffset++;
                }

                yConv[index] = sumY;
                xConv[index] = sumX;
            }
        }
        for (int x = initX; x < maxX; x++) {
            for (int y = initY; y < maxY; y += width) {
                float sum = 0f;
                int index = x + y;
                for (int i = 1; i < kwidth; i++)
                    sum += diffKernel[i] * (yConv[index - i] - yConv[index + i]);
                xGradient[index] = sum;
            }
        }

        for (int x = kwidth; x < width - kwidth; x++) {
            for (int y = initY; y < maxY; y += width) {
                float sum = 0.0f;
                int index = x + y;
                int yOffset = width;
                for (int i = 1; i < kwidth; i++) {
                    sum += diffKernel[i] * (xConv[index - yOffset] - xConv[index +
yOffset]);
                    yOffset += width;
                }

                yGradient[index] = sum;
            }
        }
        initX = kwidth;
        maxX = width - kwidth;
```

```
        initY = width * kwidth;
        maxY = width * (height - kwidth);
        for (int x = initX; x < maxX; x++) {
            for (int y = initY; y < maxY; y += width) {
                int index = x + y;
                int indexN = index - width;
                int indexS = index + width;
                int indexW = index - 1;
                int indexE = index + 1;
                int indexNW = indexN - 1;
                int indexNE = indexN + 1;
                int indexSW = indexS - 1;
                int indexSE = indexS + 1;

                float xGrad = xGradient[index];
                float yGrad = yGradient[index];
                float gradMag = hypot(xGrad, yGrad);

                //Perform non-maximal supression
                float nMag = hypot(xGradient[indexN], yGradient[indexN]);
                float sMag = hypot(xGradient[indexS], yGradient[indexS]);
                float wMag = hypot(xGradient[indexW], yGradient[indexW]);
                float eMag = hypot(xGradient[indexE], yGradient[indexE]);
                float neMag = hypot(xGradient[indexNE], yGradient[indexNE]);
                float seMag = hypot(xGradient[indexSE], yGradient[indexSE]);
                float swMag = hypot(xGradient[indexSW], yGradient[indexSW]);
                float nwMag = hypot(xGradient[indexNW], yGradient[indexNW]);
                float tmp;

                if (xGrad * yGrad <= (float) 0 /*(1)*/
                    ? Math.abs(xGrad) >= Math.abs(yGrad) /*(2)*/
                        ? (tmp = Math.abs(xGrad * gradMag)) >= Math.abs(yGrad *
neMag - (xGrad + yGrad) * eMag) /*(3)*/
                          && tmp > Math.abs(yGrad * swMag - (xGrad + yGrad) *
wMag) /*(4)*/
                        : (tmp = Math.abs(yGrad * gradMag)) >= Math.abs(xGrad *
neMag - (yGrad + xGrad) * nMag) /*(3)*/
                          && tmp > Math.abs(xGrad * swMag - (yGrad + xGrad) *
sMag) /*(4)*/
                    : Math.abs(xGrad) >= Math.abs(yGrad) /*(2)*/
                        ? (tmp = Math.abs(xGrad * gradMag)) >= Math.abs(yGrad *
seMag + (xGrad - yGrad) * eMag) /*(3)*/
                          && tmp > Math.abs(yGrad * nwMag + (xGrad - yGrad) *
wMag) /*(4)*/
                        : (tmp = Math.abs(yGrad * gradMag)) >= Math.abs(xGrad *
seMag + (yGrad - xGrad) * sMag) /*(3)*/
                          && tmp > Math.abs(xGrad * nwMag + (yGrad - xGrad) *
nMag) /*(4)*/
                    ) {
                    magnitude[index] = gradMag >= MAGNITUDE_LIMIT ? MAGNITUDE_MAX :
(short) (MAGNITUDE_SCALE * gradMag);
                    //NOTE: The orientation of the edge is not employed by this
                    //implementation. It is a simple matter to compute it at
                    //this point as: Math.atan2(yGrad, xGrad);
                } else {
                    magnitude[index] = 0;
                }
            }
```

```
    }
}
private float hypot(float x, float y) {
    return (float) Math.hypot(x, y);
}
private float gaussian(float x, float sigma) {
    return (float) Math.exp(-(x * x) / (2f * sigma * sigma));
}
private void performHysteresis(int low, int high) {
    Arrays.fill(data, (short) 0);
    int offset = 0;
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            if (data[offset] == 0 && magnitude[offset] >= high) {
                follow(x, y, offset, low);
            }
            offset++;
        }
    }
 }

private void follow(int x1, int y1, int i1, int threshold) {
    int x0 = x1 == 0 ? x1 : x1 - 1;
    int x2 = x1 == width - 1 ? x1 : x1 + 1;
    int y0 = y1 == 0 ? y1 : y1 - 1;
    int y2 = y1 == height -1 ? y1 : y1 + 1;

    data[i1] = (short) magnitude[i1];
    for (int x = x0; x <= x2; x++) {
        for (int y = y0; y <= y2; y++) {
            int i2 = x + y * width;
            if ((y != y1 || x != x1)
                && data[i2] == 0
                && magnitude[i2] >= threshold) {
                follow(x, y, i2, threshold);
                return;
            }
        }
    }
}

private void thresholdEdges() {
    for (int i = 0; i < picsize; i++) {
        data[i] = (short) (data[i] > 0 ? 0 : 1);
    }
}

private void readData() {
    for (int i = 0; i < sourceTable.length; i++) {
        for (int j = 0; j < sourceTable[i].length; j++) {
            data[(i * sourceTable[i].length) + j] = sourceTable[i][j];
        }
    }
}

private void writeEdges(short values[]) {
    if (edgesTable == null) {
        edgesTable = new short[height][width];
```

```
            for (int i = 0; i < edgesTable.length; i++) {
                for (int j = 0; j < edgesTable[i].length; j++) {
                    edgesTable[i][j] = values[(i * edgesTable[i].length) + j];
                }
            }
        }
    }
}
```

Appendix C: Artificial Neural Network

```
/**
* Contains a neural network that classifies points in a frontal weather system
* Calculate(Point point) is called to run classification
*
* @author simen
*/
public class EncogClassifier {
   //Filenames for the network
   public static final String filename = "network/testNetwork";
   public static final String trainingSetName = "network/trainingSet";
   //Network variables
   BasicNetwork network;
   MLDataSet trainingSet;
    //Working data sets
   public List<int[]> inputs = new ArrayList<int[]>();
   public List<int[]> outputs = new ArrayList<int[]>();
   //Input data
   short[][] data;
   int inputSize = 5;
   int outputSize = 4;
   //The instance
   static EncogClassifier instance;

   /**
    * Constructs a classifier
    * @param data the input data for the instance, may be null
    */
   private EncogClassifier(short[][] data) {
      this.data = data;

      network = new BasicNetwork();
      network.addLayer(new BasicLayer(null,true,inputSize));
      //network.addLayer(new BasicLayer(new ActivationSigmoid(),true,5));
      network.addLayer(new BasicLayer(new ActivationSigmoid(),false,outputSize));
      network.getStructure().finalizeStructure();
      network.reset();
   }

   /**
    * Gets a classifier, new if none exists
    * @param data the data for the network, may be null
    * @param fromFile whether to load network from file or create new
    * @return the Classifier
    */
   public static EncogClassifier getInstance(short[][] data, boolean fromFile) {
      if (instance == null) {
          instance = new EncogClassifier(data);
          if (fromFile) {
              try {
                  instance.network = (BasicNetwork) SerializeObject.load(new
File(filename));
              } catch (ClassNotFoundException | IOException e) {
                  System.out.println("No saved network found, creating new");
              }
```

```
            }
        } else {
            instance.data = data;
        }

        return instance;
    }

    /**
     * Trains the network on a set of fronts
     * @param record the fronts
     * @param inputDirectly whether or not to add the points to the existing
training set
     * @param numberOfInvalidFronts how many non-fronts to be added to the set
     */
    public void addToTrainingSet(FrontRecord record, boolean inputDirectly, int
numberOfInvalidFronts) {

        if (inputDirectly) {
            loadTrainingSet();
        }

        //For all fronts
        for (int i = 0; i < record.getFronts().size(); i++) {
            Front front = record.getFronts().get(i);
            //If it is of relevant type
            if (front.getType() != Front.SQUALL_LINE && front.getType() !=
Front.TROUGH ) {

            //Set output values
            int[] output = new int[outputSize];
            output[0] = front.getDirection();
            for (int j = 1; j < outputSize; j++) {
                output[j] = 0;
            }
            switch (front.getType()) {
            case Front.WARM_FRONT:
                output[1] = 1;
                break;
            case Front.COLD_FRONT:
                output[2] = 1;
                break;
            case Front.OCCLUSION:
                output[3] = 1;
                break;
            default:
                break;
            }
            //For each position
            for (int j = 0; j < front.getPositions().size(); j++) {
                //Generate input values and add to training set
                Point point = front.getPositions().get(j).getPoint();
                if (Util.validPosition(data, point.getX(), point.getY(), 13)) {
                    int[] input = inputFromData(point);
                    if (inputDirectly) {
                        MLDataPair pair = new BasicMLDataPair(new
BasicMLData(copyFromIntArray(input)),
                                new BasicMLData(copyFromIntArray(output)));
                        trainingSet.add(pair);
```

```
                } else {
                    inputs.add(input);
                    outputs.add(output);
                }
            }

         }
         }

      }

      if (!inputDirectly) {
           createTrainingSet();
      } else {
          saveTrainingSet();
      }

   }

   /**
    * Creates a new training set from the working data set
    */
   public void createTrainingSet() {
      int[][] ins = (int[][]) inputs.toArray(new int[inputs.size()][inputSize]);
      int[][] outs = (int[][]) outputs.toArray(new
int[outputs.size()][outputSize]);

      // create training data
      trainingSet = new BasicMLDataSet(copyFromIntArray(ins),
copyFromIntArray(outs));
   }

   /**
    * Copys a data set of ints to a data set of doubles
    * @param source the source table
    * @return the double table
    */
   private double[][] copyFromIntArray(int[][] source) {
       double[][] dest = new double[source.length][source[0].length];
       for(int i=0; i<source.length; i++) {
          for (int j = 0; j < source[i].length; j++) {
              dest[i][j] = source[i][j];
          }
       }
       return dest;
   }
   private double[] copyFromIntArray(int[] source) {
       double[] dest = new double[source.length];
       for(int i=0; i<source.length; i++) {
              dest[i] = source[i];
       }
       return dest;
   }

   /**
    * Saves a training set to File
    */
   public void saveTrainingSet() {
      EncogUtility.saveEGB(new File(trainingSetName), trainingSet);
```

```
   }

   /**
    * Gets a training set from file
    */
   public void loadTrainingSet() {
      trainingSet = EncogUtility.loadEGB2Memory(new File(trainingSetName));
   }

   /**
    * Trains the network down to a given error threshold
    * @param threshold
    */
   public void doTraining(double threshold) {
      //Save the current training set
      saveTrainingSet();
      //Train the neural network
      final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);
      int epoch = 1;
      do {
          train.iteration();
          System.out.println("Epoch #" + epoch + " Error:" + train.getError());
          epoch++;
          } while(train.getError() > threshold);
      train.finishTraining();
      //Save the network
      try {
          SerializeObject.save(new File(filename), network);
      } catch (IOException e) {
          e.printStackTrace();
      }
      saveTrainingSet();
   }

   /**
    * Classifies a single point
    * @param point the point to classify
    * @return the classification
    */
   public double[] calculate(Point point) {
      //Format input
      int[] input = inputFromData(point);
      MLData in = new BasicMLData(copyFromIntArray(new int[][] {input})[0]);
      //Get output
      final MLData output = network.compute(in);
      double[] out = new double[outputSize];
      for (int i = 0; i < out.length; i++) {
          out[i] = output.getData(i);
      }

      return out;
   }

   /**
    * Tests the performance of the network on a sample of the test data
    */
   public void test() {
      //Test the neural network
```

```
      System.out.println("Neural Network Results:");
      int count = 0;
      for(MLDataPair pair: trainingSet ) {
          if (count % 10 ==0) {
              final MLData output = network.compute(pair.getInput());
              DecimalFormat outputFormat = new DecimalFormat("0.0000");
              //Print results
              System.out.println(pair.getInput().getData(0));
              for (int i = 1; i < 5; i++) {
                  System.out.print(pair.getInput().getData(i) + ", ");
              }
              System.out.println();

              for (int i = 0; i < outputSize; i++) {
                  System.out.print("actual=" +
outputFormat.format(output.getData(i)) +
                          ",ideal=" + pair.getIdeal().getData(i) + ", ");
              }
              System.out.println("");
          }
          count++;
      }
      Encog.getInstance().shutdown();
   }

   /**
    * Transforms a given point to a set of input variables for classification
    * @param point the point to classify
    * @return the input data for the network
    */
   int[] inputFromData(Point point) {
      //Coordinates
      int x = point.getX();
      int y = point.getY();
      //Date set to fill
      int[] result = new int[inputSize];
      short base = (short) (data[x][y]);
      //First value is the value of the point
      result[0] = base;
      //Values 2-5 are the rates of change horizontally, vertically and in both
diagonals through the point
      result[1] = (int) (data[x][y + 10] - data[x][y - 10]);
      result[2] = (int) (data[x + 10][y] - data[x - 10][y]);
      result[3] = (int) (data[x + 7][y + 7] - data[x - 7][y - 7]);
      result[4] = (int) (data[x - 7][y + 7] - data[x + 7][y - 7]);

      return result;
   }
}
```