# Investigating the effectiveness of deep learning in the prediction of precipitation

Cameron F. T. Pope

University of London

BSc Computer Science Final Report

A public repository containing the full project code and data can be found at:
https://github.com/i360cammy/FinalProject

This report contains in-text images which take certain sections over the specified page limit, however these images do not count toward the page limit, and all text is within the specified page limit for each section. Evidence of this can be found at the above repository, where this report has been uploaded without any in-text images so the length of each section can be verified.

## Introduction

Weather forecasting plays a vital role in the way people live their lives, ranging anywhere from planning activities on the weekend, to knowing when to evacuate due to an approaching life-threatening storm. This importance is evidenced by US adults reportedly checking the weather 300 billion times per year (Lazo et. al. 2009).

My project expands and builds upon a given template - '*machine learning and neural networks template 1 - deep learning on a public dataset'*. The high-level concept for the project is the analysis of the effectiveness of deep learning in the ability to predict rainfall in comparison to other machine learning methodologies, such as support vector machines (SVMs) and random forests (RFs), as well as in comparison to current professional forecasting standards.

My motivation for conducting such a study comes from the conjunction of two of my personal passions - machine learning and meteorology. Meteorology is also an area I feel has yet to feel or embrace the advantages and power that machine learning has to offer as much of the industry still relies on atmospheric simulation models requiring vast computational power - and the costs associated with running and maintaining such systems.

I have chosen precipitation as the target metric as I feel machine learning models will be able to identify patterns in observations that lead to precipitation events, as well as rainfall prediction having a large impact on the perceived accuracy of weather forecasting by the public. This phenomenon is outlined in great detail by chief meteorologist JD Rudd at Spectrum News (2020). Rainfall is also a keystone to weather forecasting along with air temperature and as such carries a high degree of importance to optimize predictions accuracy.

The goal of the project is not only to discuss whether machine learning can be applied to rainfall prediction, but also whether deep neural networks outperform conventional machine learning algorithms. While ultimately I believe that machine learning will eventually overtake current forecasting methods as a complete forecasting tool, at this moment I believe the best use of machine learning is in conjunction with current atmospheric models, adding another layer of robustness and verification in the forecasting lifecycle.

### Current Standards

As outlined by the Met Office, the United Kingdom's national weather forecasting service, current forecasts are made using a 3-step cycle (Meteorological Office, 2022):

1. Observe – Recordings of weather variables around the world are collected in conjunction with satellite imagery
2. Predict – These observations are then used predict the state of the weather into the future, using the observations as a starting conditions for the prediction model
3. Refine – These predictions are constantly reviewed as conditions change and any alterations made before being published

This does not sound too dissimilar from a high-level machine learning workflow whereby we collect or source our data (observe), use this data to create predictions by passing it through a model (predict), before making small adjustments to the model to iron out any errors or inconsistencies (refine). However, current weather forecasting systems utilize vast supercomputers using hugely complex equations designed to model how weather will move and evolve. Essentially, it aims to create an atmospheric simulation model whereby we can then feed in start conditions and watch how the atmospheric model changes over time giving us our predictions. This is a very computationally expensive process and inaccessible for novices with an interest in weather forecasting, and as such is a very centralized process. Machine learning has potential to turn forecasting into a distributed process.

## Future

This method also gives rise to another problem, which is our changing climate. Reliable performance of forecasts falls by about a day when the atmosphere warms by a few degrees Celsius (Garthwaite, 2021). According to NASA's vital signs of the planet, global surface temperatures have increased by around 1.0°C since 1880 (NASA, 2022), and is more likely than not to surpass 2°C by the end of the century (Collins et al. 2013). This leads to uncertainty over the effectiveness of current forecasting systems as the climate changes. As these are based on atmospheric models, it remains to be seen how well these perform in a rapidly changing climate whereby the evolution of weather systems is affected, and how often these models have to be updated and calibrated.

Machine learning models provide potential to maintain a higher degree of accuracy while having the ability to be easily modified and distributed, while also being far more efficient once the model has been trained. This allows forecasts to be run in real-time on home machines such as laptops or phones, and the open-source nature allows anyone to view or modify the model in an attempt to improve performance. It would also be possible to 'close the loop', using the models predictions to continually update the model, allowing it to automatically keep up with a changing climate without the need for human intervention.

## Implementation

The data for the training and testing of the models comes from historic weather station data from Ohio State University Airport in Columbus, OH. This includes features such as minimum and maximum air temperature, wind speed and direction, weather conditions etc. The data underwent heavy cleaning and preprocessing ready to be passed to the models. While neural networks are more resilient to impure data compared with more traditional machine learning models such as SVMs, the data will be preprocessed to a standard such that all models will be able to use it. This avoids different models having different input data, which would result in making meaningful conclusions more difficult, although this is something we have to keep in mind when evaluating the usefulness of each model.

The architecture of the deep neural network will be a series of densely connected layers. This will start off as a small model at first, before being scaled up to overfit, scaled back and then optimized, as outlined in Francois Chollet's universal deep learning workflow (2017). The output layer of the model will be a single neuron layer with a sigmoid activation due to our binary classification problem. The neuron firing gives us a prediction of 1 (rain), while the neuron not firing gives us a prediction of 0 (no rain).

The remaining models will be imported from the sklearn library and trained using default parameters at first, before attempts at optimizing each model's performance. This allows a comparison of each model in its most basic form, and analysis of how well each model is able to be improved and scaled through optimization.

More detailed analysis of the implementation of each model will be provided in Section 3 - Design.

## Literature Review

### Paper 1

Weyn, J. A., Durran, D. R., Caruana, R., & Cresswell-Clay, N. (2021) created a deep learning weather prediction (DLWP) model capable of predicting six atmospheric variables with a 6-hour resolution. One of the huge benefits of this model was that once the model was trained, it was able to produce forecasts very quickly – just three minutes for 320 six-week forecasts.

The deep model uses convolutional neural networks with a randomized training process, leading to 32 models with slightly different weights. This is to produce slight variations of the forecasts with the aim to cover the range of possible future atmospheric states. This is derived from the fact that very small changes in initial weather conditions can produce vastly different weather days later, and as we are not able to measure or model the initial conditions with 100% accuracy, the model simulates small variations in the starting conditions around the perceived actual conditions to account for this.

The inputs for the model, which includes 2-m temperature and geopotential height at different pressure thresholds, were mapped to a cubed sphere representing the Earth, before being subsequently passed through the network. The network itself consisted of 2D CubeSphere convolution layers, up sampling, and average pooling.

The results of the model show that the model approaches state-of-the-art performance from the European Center for Medium Range Weather Forecasts (ECMWF) at lead times between 4-6 weeks, while being able to conduct many more forecasts (320 vs 50) in a shorter period of time, and on far inferior machinery. This opens the possibility of the decentralization of weather forecasting, allowing forecasts to be run directly on users' devices.

Where the model falls down is in its short-term localized forecasts, which were far outmatched by the current standard models used by organizations such as the ECMWF and the Meteorological Office. However, the model was able to demonstrate effectiveness at modeling larger weather systems on a short timescale, producing an accurate 4.5-day forecast for Hurricane Irma in 2017.

The model also fails to be able to predict rainfall at all, which is a huge detriment to the ability of the model to be used as a complete weather forecasting tool. Rainfall is arguably one of the most important aspects of weather forecasting, and many people perceive the reliability of forecasts in relation to its ability to accurately predict rainfall.

My own models will attempt to fill the gap in the DLWP's capability with the means to predict short-term rainfall.

### Paper 2

Mark Holmstrom, Dylan Liu and Christopher Vo (2016) from Stanford University applied a linear regression model to short-term weather forecasting. This method did not use a neural network; however, they were still attempting to use machine learning as a weather forecasting tool.

The forecasting was limited in scope to predicting minimum and maximum temperatures over a seven-day period, using data from the previous two days as the basis for the prediction. The data used to inform the prediction includes the minimum and maximum temperatures, the mean humidity, mean atmospheric pressure and the classification (Clear, overcast, scattered clouds etc.) over the two-day period.

The total dataset spanned a 5-year period of daily weather data from Stanford, CA, with the first 4 years data being used to train the model, and the final year being used to test the resulting model. They used an exhaustive grid search to find the optimal weights for model performance. It was during this process that they found that mean humidity and mean atmospheric pressure did not correlate with maximum and minimum temperatures. Leaving this in would likely hamper model performance, as the model will not be able to represent a relationship between these features. Due to the little to no correlation, it means these features cannot be used to predict maximum and minimum temperatures, and as such, they were removed.

The model itself was greatly outperformed by professional forecasts. The root mean squared error (rmse) of the linear model on a one-day forecast was 5.039 (°F), while the rms from the professional forecast was 2.612 (°F). However, the model's performance approached the levels of professional performance on a seven-day forecast, with the rmse being 5.642 and 5.062 respectively. This could mean that the model will outperform professional forecasts at periods of longer than a week, however this was not tested by the researchers, despite them admitting that this could be the case.

The biggest problem with the model is that it uses such a small sample of two days' data to inform predictions up to a week away. This leads to the model simply extrapolating from these data points. The model also has no concept of weather systems – it simply says that if the last two days were hot, the next seven days will likely be hot too. In reality, a cold front may be moving in which will be identified by atmospheric models but not the linear model and it will throw the accuracy off. This model could potentially work in stable climates, however in areas where there are often large fluctuations in day-to-day temperatures, the model would likely be ineffective.

However, it is encouraging that such a simple model was able to approach the performance of professional forecasts at a seven-day lead time, albeit in a very narrow in scope model only capable of predicting maximum and minimum temperatures. It is possible to conclude that a more comprehensive model trained and tested over longer time periods could start to beat out conventional forecast models.

In my project I will also aim to critically evaluate my own results against professional industry standards, even if the performance is greatly outmatched by the professional forecasts. It is important to be able to admit that despite our best efforts, our model fell short.

**Paper 3**

Premachandra and Kumara (2021) attempted to use various different machine learning models and methods to predict rainfall patterns in Sri Lanka in an attempt to find the best model. This was in relation to the changing climate affecting the reliability of such forecasts, and the threat this changing climate has on Sri Lanka's agricultural industry.

The data for training the various models included hourly measurements of key atmospheric features such as humidity, wind speed, and temperature, among others, with the target being whether it rained or not. From this, daily data was generated through averaging of hourly data. The models used in the comparison were: multiple linear regression (MLR), support vector machines (SVM), K-nearest neighbor (KNN), and random forest (RF). Notably, this does not include any form of neural network, which leaves a gap in the analysis conducted by the authors.

The volume of data used to train the data was also very limited – containing just over 2 years of hourly data, which was subsequently reduced further by condensing this down into daily data. This can also lead to questions over the effectiveness of the resulting models due to the limited amount of data they were trained on, and no mention was made of K-fold cross-validation to help mitigate the effects of a small dataset.

The authors also state default parameters were used for each model. While this does help in the comparison of models, the results may change depending on how the model is optimized. For example, in their report MLR performed the worst by far. But based on their methodology it is not possible to know how this would change if they had attempted to optimize the parameters to improve the model. Perhaps MLR would become one of the better models.

Their results indicate that the random forest model performed the best, however as discussed this does not really provide any meaningful insights, only that this model performed best when in its default state. If such models were to be deployed, they would have to undergo a period of optimization of the model, so picking a model based on its default state is not advised.

That aside, the random forest model did actually perform well, achieving 84% precision and 89% recall, which is impressive given the limited dataset and default model.

In my project I will aim to compare my deep learning model with other machine learning models trained on the same data to help me reason about the effectiveness of my model, and whether another more appropriate model would have been more suitable for this purpose.

## Paper 4

Nitin Singh, Saurabh Chaturvedi, and Shamim Akhter (2019) created a rainfall forecaster using machine learning and a Raspberry Pi. Rather than relying on collecting data from a third-party source to train the model and keep it in operation once trained to create predictions, the Raspberry Pi could collect its own observations, removing the need for the third-party data collection. This also has the benefit of providing data accurate at the exact location the forecast is needed, rather than relying on data which comes from a weather station that could be situated many miles away. However, in this instance a dataset was sourced for the purposes of training the model.

The system even includes a graphical user interface (GUI), meaning such a system is very user friendly, hides away the complexity of the sensors and machine learning model, and makes it possible to distribute to households as a genuine product.

The data collected includes temperature, pressure and humidity which it uses to inform the likelihood of rain that day, using a random forest model to generate the prediction. The user can then ask for the prediction of rain for that day and the observations are fed through the model, outputting either a 0 for no rain, or 1 for rain.

When creating these models to predict rainfall you must be careful to avoid a biased model. Given that it is usually more likely to not rain than rain (depending on location and time of year), a model that simply predicts no rain every day will still achieve a high accuracy, but such a model is useless in terms of its purpose. In this paper the authors were aware of this and showed the confusion matrix. On the surface their results look good, however nearly a third of the time (56/178) the model predicted rain it did not rain, and over 10% of the time (166/1657) the model predicted no rain, it rained. This shows that the model has a tendency to wrongly predict days where rain is expected.

While the authors state the model has 87.9% accuracy, which is technically correct, precision and recall would be a more suitable metric to judge the performance of the model due to the unbalanced data between rain and no rain. Using this method would show that the precision of the model is far lower than the accuracy (68.5% vs 87.95%), and the F1 score is lower again at 52%.

One possible solution the authors could have proposed to solve this would be to up-sample the no rain data to match the number of rain data points during training. This would help the model avoid such bias and ensure the accuracy metric is not misleading.

In my project I will reason about which evaluation metrics would be most suitable for the data, and consider techniques such as up-sampling to help balance the dataset and avoid a biased model.

## Project Design

### Domain and Users

The domain of the project is weather forecasting - more specifically the short-term forecasting of rainfall based on a set of starting conditions - or observations.

The targeted user audience for the project is twofold. Firstly, the weather forecasting industry as a whole. Deep learning has the potential to advance forecasting accuracy and longevity in a changing climate which threatens to weaken the effectiveness of current forecasting techniques. Even if machine learning does not currently possess the power to surpass conventional forecasting, its value as an accompaniment to these current methods should still be explored.

Another target user group are novice forecasters or those with an interest in weather forecasting. As discussed in the introduction to the project, current forecasting models require supercomputers which consume vast amounts of resources and remain out of reach of anything but large corporations with the capability of funding and operating such machinery. Deep learning and the wider field of machine learning has the potential to make weather forecasting accessible to everyone, with models able to be run on phones and laptops.

Hence, we can break down the two target user groups and justify how the design satisfies the needs of each group. For the forecasting industry:

1. Potential for more effective forecasting models
2. More forecast simulations can be run compared to current models due to the decreased computational cost
3. Models than can potentially generalize better in a changing climate, and can even update itself as more data becomes available

For those with an interest in forecasting:

1. A vector to experience and experiment with forecasting on their home machinery
2. Model can be distributed and tweaked to increase its effectiveness
3. Can potentially give rise to a community of such forecasters each bringing their own expertise and perspective to help advance the field

It is unlikely in the short term that these deep learning models are able to compete with current forecasting standards in all aspects and as a complete forecasting tool, however the deep learning approach allows scalability as computational power improves, more data becomes available, and the field of deep learning advances further, taking advantage of new ideas and concepts. Indeed, as the Deep Learning Weather Prediction [Weyn et al. 2021] discovered, these models are already able to approach the industry standard in certain aspects, albeit in limited and cherry-picked scenarios.

### Project Structure

The project comes in the form of a complete Jupyter notebook containing a short introduction, data sourcing, data cleaning and preprocessing, exploration, training and testing of baseline models, improvements and optimizations of models, evaluation of models, and a representation of results.

The different machine learning models that will be used include Support Vector Machines, Random Forests, and Logistic Regression. These models will first be used in their most basic form with default hyperparameters as this will allow us to compare each model's baseline performance, before undergoing optimization to understand how well each model scales.

The main focus of the notebook will be on the deep model itself as our goal is the investigation of how deep models perform in this domain. This building of this model will follow closely with the universal workflow of machine learning as outlined by Francois Chollet's Deep Learning with Python (2017). This is expanded further in the methods sub-section below.

## Models Architecture

The deep model includes a series of densely connected layers. Each layer is made up of a specified number of neurons. Densely connected layers means that each neuron in a layer is connected to each and every neuron on the layers before and after the layer it belongs to. Each neuron has a weight and a bias which are initialized at random, and then optimized to improve the network's performance using back-propagation methods such as Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (ADAM), using loss functions such as Mean Squared Error (RSE) in regression tasks or Categorical Cross-Entropy in classifications tasks to inform the rate and direction of the descent. An example architecture of a 2-hidden layer, densely connected neural network is demonstrated using the diagram below, created using an open source online tool found here: http://alexlenail.me/NN-SVG/

SVM's work by mapping each sample to a high-dimensional space and then finding an optimal linear separator such that the categories are separated by the plane with the largest margin between each sample and the separator. This can then be transformed such that the linear separator can be represented in 2-dimensional space, giving us a class boundary.

Logistic Regression uses a sigmoid function to convert an input sample into a value between 0 and 1. This output represents the probability that the input sample will belong to a given class, with a 0 meaning certainty it belongs to class 1, whereas 1 means certainty it belongs to class 2. A value of 0.75 represents 75% certainty it belongs to class 2.

Random Forest models create an ensemble of decision trees. Each decision tree creates its own prediction for the class in which the sample belongs to, and the class with the highest number of predictions across the ensemble becomes the class prediction. The large number of trees creates a layer of robustness in the prediction as a majority of uncorrelated decision trees came to the same conclusion.

## Technologies and Methods

The project itself is being completed inside Jupyter Notebooks using Python. The deep learning model will be created using the Keras library which is built on TensorFlow. I will also be utilizing libraries such as numpy for numerical operations, scikit learn for splitting data and results validations, and pandas for its data structures. Scikit learn will also be used for its implementation of the other machine learning models - namely support vector machines, logistic regression, and random forests.

The project covers a wide range of concepts and fields, including neural networks, data sourcing, data manipulation and cleaning, classification, synthetic data creation, and atmospheric modeling. Many of the concepts and techniques I have learned over the course of the degree, some of which I have learned from extracurricular courses, and some of which I was required to learn over the course of the project.

The method followed over the course of the project was the universal machine learning workflow from Francois Chollet's Deep Learning With Python (2017). This workflow helps to ensure that we are following best practice and considering key decisions such as which optimizer to use when training the model, or whether to use K-fold cross validation. The workflow outlines 7 key steps in the deep learning workflow, namely: defining the problem and assembling the dataset, choosing a measure of success, deciding on an evaluation protocol, preparing the data, developing a model that does better than a baseline, developing a model that overfits, and regularizing the model and tuning hyperparameters.

## Work Plan

For my work plan I created a Gantt chart outlining key stages and steps for the project to be completed on time. This chart is not exhaustive, and due to the nature of development, certain tasks or sections were delayed or took longer than anticipated, while other sections were completed faster. This also does not factor in extra tasks I discovered during development, such as filling in gaps in my knowledge required for further development. The chart allowed me to understand the pace of development and whether I was on target throughout the course of the semester. If I fell behind, I could easily identify this and rectify it before falling further behind. An image of the Gantt chart can be found in the appendix labeled figure 1.

## Testing and evaluation

Testing and evaluation will take place during each iteration of the deep learning model. This is done through specifying a loss function for us to monitor the model's performance during training. For the deep model we are conducting a classification task, and as such we will be using binary cross entropy for our loss function and using Adam as our optimizer, which is similar to Stochastic Gradient Descent (SGD) with momentum added to reduce the chance of getting stuck in local minima and speed up the descent on steep gradients.

We will also split the training data further into training and a validation set. The validation set allows the model to be tested after each epoch on unseen data, giving us a better indication how the model is performing. This also allows us to save our test data for once the model is done training, allowing this data to remain unseen by the model until we are ready to evaluate the performance of the final model.

The model returns a history object which contains the loss for each training epoch, as well as the loss of the model on the validation set after each epoch is complete. This will allow us to plot a graph of loss over time, so we are able to identify when the model begins to overfit and allow us to tune the number of epochs, so we stop training right before overfitting occurs. The history object also tracks specified metrics over the course of training, which in this case will be accuracy, precision, and recall. Again, this allows us to then plot how these manifest over the course of training, identifying where the model is performing optimally.

The comparison machine learning models will be subject to testing also. This is done through passing the testing data into the model once training has been completed, and saving the predictions in a variable. We are then able to compare the actual true values to the values of the models' predictions. We can then use metrics such as accuracy, precision, recall, and f1 score to reason about the effectiveness of each model, and use techniques such as classification reports and confusion matrices to further delve into the models performance.

We will also conduct evaluation of the models at a project scale, which will be done based on our identified user and domain needs. This will be done through reasoning of results and whether these fulfill these requirements we had identified, and the reasons for why they do or do not.

We will also evaluate the performance of the model relative to current industry performance, as this ultimately dictates the effect such models have on the industry. I do not expect this project to achieve the levels of performance weather forecasting currently exhibits, as such performance from deep models is still likely a few years off, however this is still a comparison we must consider.

## Implementation

The project has been completed in a Jupyter notebook and has been divided into 8 sections, namely: Introduction, Data Cleaning, Target Feature Creation, Baseline Model Creation (iteration 1), Oversampling Data using SMOTE, Oversampled Models (iteration 2), Optimizing Models (iteration 3), and Results.

The introduction is purely a textual section containing no code, and is designed to give a brief overview of the goals of the notebook, where the data was sourced from, and the motivations involved for undertaking such a project such that someone can quickly understand the basis of the project without prior knowledge or having access to this report.

The data cleaning stage is the largest and was the most time consuming, beginning by importing all the libraries that are required throughout the notebook. The data itself, contained in a CSV file, is imported using the Pandas library and stored in an easy to use DataFrame. Other libraries used include scikit learn for the conventional models as well as utilities such as train test split and accuracy score, tensorflow for creating the neural networks, and imbalancedlearn for upsampling data using synthetic data creation.

Next, some basic exploration is done to get a starting understanding of the data we are dealing with. This includes methods such as looking at the head of the DataFrame, and using methods such as info() and describe() to learn more about the data, such as the number of NA values for each feature, the data type of each feature, and some basic statistical metrics such as mean and quartiles. It is during this we were able to see some features that are almost exclusively missing data, such as the ‘SNOW’ and ‘SNWD’ features.

Because most of these values are missing there is not much we can do to fill in the missing values, so we remove these features entirely. We also drop some features that we do not require, such as the station code and station name.

Next, the data was converted into a Pandas Timestamp and set as the index. This allows us to properly represent the time series and make it easy to plot any feature over time.

Once this was done, we needed to start dealing with the remaining missing values. The method used to fill the missing values depended on the feature. Some, such as average temperature, we were able to fill using data that was present in the data. We had data for the maximum and minimum temperature, so we were able to create an estimate of the average temperature simply by taking the average of these two.

For other pieces of data this was not possible. For missing maximum and minimum temperature values we used interpolation to fill any missing values, based on the reasoning that daily changes in temperature tend to be gradual and close to the surrounding days data. This obviously isn't always true, but allows us a decent estimate of the missing value.

Other missing data did not have any relationship between consecutive observations. For example, maximum wind speed varied greatly from day to day in a seemingly random fashion. To decide a fill method, some visualizations were created, an example of which can be seen from the image below. In this case, we could see the data was skewed and contained quite a few outliers, so the mean would not be an optimal fill method either here, so the median was chosen due to its resilience to these outliers.

These steps were repeated until all missing data was filled using an appropriate method, except for precipitation. Any samples with missing precipitation data were removed completely. This is because this feature is our target and as such we want to avoid creating guesses that could be off by a significant margin and harm the effectiveness of the model, both in training and testing.

Once the data was in sufficient state that it could be accepted by all of the models, we needed to create our target feature, which was completed by duplicating the 'precipitation' feature into a new column called 'rain tomorrow', shifting this new column up by one, and encoding the value such that any non-0 value became a 1, and all values of 0 were kept at 0. This means a value of 1 in the 'rain tomorrow' column means it rains on the subsequent day, while a 0 means it does not. At this stage a heatmap was also created with the correlation between all feature pairs. There were a few features that seemed to not be correlated at all with 'rain tomorrow', however we have to be very careful removing these as they could be correlated with 'rain tomorrow' in a non-linear fashion and we have to be aware that all weather features are influenced by one another.

Each model went through 3 iterations with each subsequent iteration intended to improve the performance. To begin with, baseline models were created using default hyperparameters. This was done to ensure we can complete a fair comparison of models in relation to their default state, and to each other. As neural networks have no default, instead a small model with only 3 densely connected hidden layers was created to act as a baseline. The models were then tested by creating predictions of whether it will rain tomorrow based on current observations, and then comparing these predictions against the actual true value.

The comparisons between predicted and true values could then be used to inform further evaluation using metrics such as accuracy, precision, recall, and f1 score, with these metrics used to create a classification report and confusion matrix for each model. The classification report allows us to see the aforementioned metrics on a per-class basis, while a confusion matrix allows us to see the amount of true/false positives and negatives. A seaborn heatmap was created from the confusion matrix, allowing more intuitive and visual reasoning of results.

The second iteration kept the default hyperparameters from the first series of models, but this time using an upsampled dataset. The dataset contains roughly twice as many no rain instances than days it did rain. This caused problems with the performance of the models which is described in more detail in the evaluation section below. In order to remedy this and improve performance, the positive rain class was upsampled so that each class contained an equal number of samples. This was done using 'Synthetic Minority Oversampling Technique', or SMOTE. This technique synthetically creates instances of a class using a K-nearest neighbors algorithm on existing class members. This works by picking a data point at random, finding the K-nearest neighbors to that data point (k=5 by default), selecting one of these neighbors, drawing a line in the feature space between the original datapoint and the selected neighbor, and creating a new datapoint somewhere along the line joining the two data points. This creates a new sample of the rain class which is derived from existing samples.

The default models were then trained again on this new dataset and were re-evaluated using the same techniques as the previous iteration of models, such as the use of classification reports and confused matrices.

The third and final iteration of each model aimed to optimize the hyperparameters of the Logistic Regression, Random Forest, and Support Vector Machine models, and to optimize the architecture of the neural network. This was done by conducting a grid search for the LR and RF models, and a random search for the SVM model. For grid search, a search space is specified which contains different values for model hyperparameters and a seperate model is created for each combination of hyperparameters specified in the search space. This is also then cross validated to ensure we get the best model hyperparameters from the search space. Once the search has been exhausted, the combination of hyperparameters are returned which scored the highest based on our evaluation metric - in this case, f1 score. We can then train a new model with these optimal hyperparameters to get an improved model.

However, grid search is computationally inefficient, creating a large number of models and evaluating them each separately. For models such as LR, which have a relatively small number of hyperparameters and as such, a small search space, this is not usually a cause for concern. RF models, however, have more hyperparameters which need tuning, and the search space quickly becomes very large. For example, a search space of 5x5x5x5 which 5-fold cross validation creates, fits, and evaluates 3125 different models. For RF this is also manageable due to their efficient nature, with training time scaling logarithmically with input size (X. Zheng *et al*. 2021).

SVM's also have a large search space, but do not benefit from the same efficiency that RF's enjoy, with a time complexity of O($n^3$) (Abdiansah and Wardoyo, 2015). This makes such a search computationally extremely expensive, with a search running for over 8 hours on an octa-core CPU without nearing completion. Instead, a random search was conducted to ensure control over search execution time. Rather than exhausting every combination of hyperparameters, they are instead selected at random and we specify the number of random combinations we want to test. This also has a benefit over grid search of having a continuous search space, and so random search can potentially find better combinations of hyperparameters in the 'gaps' between the grid search.

For the neural network, this was more of an art than a science, and trial and error dictated the optimization method. Experiments were conducted with different numbers of layers, different numbers of neurons per layer, adding varying degrees of dropout etc. The models were trained for a longer period of time to ensure overfitting occurred. This was important as we could then scale back the model to the exact epoch overfitting begins, leaving us with the optimal model for any given architecture. To avoid having tens of different models in the notebook, only the best performing model was kept.

An important part of the project was adhering to the scientific method to ensure robustness and repeatability of results, and so clear conclusions could be drawn. Many of the techniques used throughout the project use randomness by design. For example, the weights and biases of each neuron in each layer is initialized randomly. The scikit learn function which splits data into training and testing sets also uses randomness to split the data. When conducting these tasks we want to ensure we are getting the same split every time, or the same model initialization every time. This is so we can be sure that not only can we repeat results, but also that when a model's performance improves that this is purely down to actions we have taken, rather than a different random data split or model initialization causing improved performance. This is done by passing a random seed to any function or model that uses randomness, which will mean the randomness will now be the same every time. In the case of the neural network, we cannot pass a random seed to the model. Instead, the random seed is set globally using a built-in Keras function.

## Evaluation

Our original outlined goal was to investigate whether deep learning is able to effectively predict precipitation both in relation to other machine learning algorithms, and compared to industry standard forecasting. This will ultimately be used to measure the success of the project, as well as our defined domain specific user goals outlined in the design section.

### Model Comparison

In order to do that, we need to better understand the results obtained from each model at each stage (basic model, basic model on oversampled data, optimized model on oversampled data). Each model was evaluated by first training the model, then creating predictions from this trained model. We can then calculate the performance of the model using metrics such as accuracy, precision, recall, f1 score, and using classification reports and confusion matrices to see these in more detail, such as seeing per-class metrics and number of true/false positives and negatives. The table below contains the f1 score of each model on the test set. Figures 2 and 3 in the appendix show how the neural network performance was evaluated.

We use the f1 score here as it is the harmonic mean of precision and recall, and as such, heavily penalizes low scores. For example, if a model obtains a precision of 0.1 and a recall of 0.9, we get an F1 score of 0.18 rather than the 0.5 we would get if we simply took the average. This ensures the models require both a high level of precision *and* recall to score well. It also ensures we are able to compare the models over the iterations more fairly. This is because before our upsampling, the ‘no rain’ class took up around two thirds of the dataset, so a model predicting no rain every time would score around 66% accuracy. After our upsampling, each class is represented equally, so a model predicting no rain every time would drop to 50% accuracy. Both models are equally poor, but one still scored better than the other. F1 score equalizes this, as both models would score 0.

From this summary we can see that logistic regression consistently performed poorly and scored the lowest of all models across all iterations. The optimized LR model also only marginally outperformed the non-optimized oversampled model, suggesting that this is close to the optimum for logistic regression.

Random forests base model performed well comparatively to other models, which is somewhat surprising given that RF models traditionally suffer from unbalanced datasets. This is evidenced by the large jump in performance when we upsample the data, beating all other models at this stage. After optimization, and despite exhausting a large search space that took several hours to run, the model performed almost identically to a model with default hyperparameters, also suggesting that this is close to peak performance for random forests on this dataset.

Support vector machines had probably the most interesting development throughout iterations. It started out by performing worse than random forest - albeit still far ahead of logistic regression. The gap to random forest widened when the data was oversampled. However - and despite not being able to carry out a full grid search on SVM due to its poor time complexity - it ended up being the best overall performing model of the project.

The neural network outperformed all other models in respect to its performance on the unbalanced dataset, reasoning that the networks are able to create better class boundaries with less data. However, the networks did not scale as well as the RF and SVM models, dropping behind both in performance by the third iteration. Although this can largely be down to less effective optimisation as grid search or random search could not be performed, and instead different architectures had to be tested manually in order to find ever decreasing performance gains. With more time to optimize and a better optimization technique such as through the use of libraries such as Keras-tuner, it is likely performance would approach and maybe surpass that of SVMs.

While the performance of deep neural networks was underwhelming, the networks would be better suited to real-world scenarios when compared to the other models considered in this study. This is due to the fact that neural networks are able to process data with missing values. This is important as observations could be missing due to sensor fault for example. While Random Forest also technically can handle samples with missing data, this is not included in the implementation used by this study.

## User Needs Evaluation

During the project design phase two user groups and their needs were identified. Those groups were the forecasting industry, and those with an interest in meteorology and weather forecasting. For the industry those needs were as follows:

1. Potential for more effective forecasting models
2. More forecast simulations can be run compared to current models due to the decreased computational cost
3. Models than can potentially generalize better in a changing climate, and can even update itself as more data becomes available

For the effectiveness of forecasting models, this project alone has not satisfied that. Industry standard models are not only more accurate, but more resilient and the atmospheric model approach brings its own advantages when compared to simple regression or classification. However, even relatively simple models like the ones in this study are able to create reasonable predictions, and some of those more advanced models discussed through the literature review are able to beat out industry standards in certain conditions.

The second point however, has been achieved. Not only were the models themselves fast to train - just a few seconds to fit the 10,000+ sample training data - but once they are trained, predictions can be created even faster, giving rise for the possibility to generate thousands of forecasts in a fraction of the time it would take to create a single forecast using atmospheric models despite being executed on far inferior machinery. While the current models in this study would generate the same prediction every time, random amounts of noise can be added to the observations to create different forecasts, simulating the small inaccuracies when sampling atmospheric conditions.

The third point remains to be seen and its effects would have to be observed over a long period of time. However, the point of models being able to constantly update themselves based on current climate conditions provides assurances that they would at least be able to keep up with a rapidly changing climate - something which is already occurring due to the climate emergency.

For those with an interest in forecasting, the following needs were identified:

1. A vector to experience and experiment with forecasting on their home machinery
2. Model can be distributed and tweaked to increase its effectiveness
3. Can give rise to a community of such forecasters each bringing their own expertise and perspective to help advance the field

The first need has been met entirely, evidenced by this study itself. I was able to experiment with weather forecasting from my own home, using a publicly available dataset, and created a model which could predict whether any given day will see rainfall or not with ~77% certainty based on the optimized SVM model. Before the rise of machine learning and artificial intelligence, this kind of performance could not be seen outside of the professional forecasting industry.

The second need is also one which has been met - again evidenced by this study - due the fact the notebook is available in a public repository, accessible to anyone, who tweak any of the models in the search for more effective performance, or import their own data to see how the models perform on completely new data.

The third point is one which is already underway. A quick search on Google Scholar demonstrates the sheer number of papers being published using machine learning and artificial intelligence to model different aspects of weather, ranging anywhere from university thesis', to professional level studies at the cutting edge of machine learning and meteorology, and ranging from applications such as predicting floods, wildfire risk zones, long term climate forecasts etc.

## Overall Project Evaluation

The baseline models were able to predict the majority of no-rain instances, but failed most of the time to predict when it would rain. If we looked at accuracy only, it would seem that the models were fairly competent at predicting rainfall, but it's only when we look deeper do we see the failures of these models, which is why we used metrics such as precision, recall, and f1 score. This proved that the unbalanced dataset was leading to the models placing too much emphasis on the 'no rain' class - where it was able to perform well - at the expense of the 'rain' class. Predicting the positive cases of rainfall is arguably more important too, with an emphasis on public opinion towards forecasting accuracy being its ability to predict rainfall.

Once the data was oversampled such that there were an even number of samples with rain and no rain, the accuracy dropped in some cases, but this forced the models to have equal emphasis on each class and not be 'lazy' to get good scores. This led to a balance with precision, recall, and ultimately, f1 score. With an equal class distribution, the baseline is no longer 66%, but 50% accuracy.

Overall, the project demonstrated machine learning is capable of modeling weather and predicting instances of rain. However, these models are not capable of predicting when rainfall will occur with higher resolution, nor predict the expected amount of rainfall. This is partly due to the resolution of the data, which contained only one set of observations per day. It's feasible to assume that a model receiving data every hour would then be able to predict rainfall at an hourly resolution.

The models are also currently geographically limited to the point in which data collection occurred. While this could be beneficial in home devices which are able to observe and create their own predictions, the models have no concept of geographical space, unlike industry standard models which can model entire weather systems and how they evolve not only over time, but over geographical distance.

This study also only considered data from one weather station, and as such it is not clear how well these models would perform when applied to different locations. The models could have inferred something from the data that is specific to a single location in its predictions which would hinder its performance at other locations. This is not an unreasonable assumption as many regions have different and at times localized climates. I invite further study to determine the degree of this effect, and whether new models would have to be trained for each location, and the size of the region in which the models retain accuracy.

The data itself was also missing some key features such as humidity, which is typically correlated with an increased chance of rainfall. It is worth suggesting that with the increased amount of data in both features and samples available to forecasting agencies, these models could undergo further improvement.

## Conclusion

To conclude this report, I want to discuss some of the challenges faced over the course of the project, and how I was able to overcome them and reach the current stage of the project, as well as improvements I would make and ideas for further development.

Firstly, a large challenge I faced was the overhaul of the project after the midterm submission. My original project idea was a deep learning model capable of predicting and identifying large meteorological events such as hurricanes. This is still an idea I am very much interested in, however I vastly underestimated the scale and complexity of such a project, and it was during my domain research and literature reviews that I realized that my original goal was not realistic in the time frame I would have to complete it. I attempted to scale back the project, however the core methodologies remained out of reach, as this required a vast amount of knowledge of atmospheric theory and science - something which I do not currently possess. I kept the core concept of the prediction of meteorological events, however switched over to the prediction of short-term localized rainfall. It was during this phase I was asking myself whether deep learning really is the best method for this task, and thus conceived the idea of comparing deep learning with other machine learning models and techniques. This was also helped by one of the papers in my literature review that had attempted a similar concept, however I felt their analysis was incomplete and that I could improve upon it. The original idea of predicting large scale meteorological events is something I wish to take with me to further study at masters level.

Another challenge was finding and then dealing with the dataset. I wanted to find a real-world dataset directly from the source. I found many of the datasets on Kaggle were simple and mostly cleaned such that they were ready to go. I wanted to experience the full lifecycle of a machine learning project, and after extensive searching, found open-source weather station data from across the US. However, I underestimated the scale of bringing the data to a state in which it is ready to be used by machine learning algorithms. The dataset contained many features with high numbers of missing values, and also a number of samples with most data missing. Reasoning about which samples to keep and which to try and fill was challenging, and then reasoning about which methods to use to fill the data was also challenging due to different methods being most effective for each feature. For example, missing values for average temperature could be estimated by averaging the maximum and minimum temperature. Missing values for maximum and minimum temperature could be best estimated using interpolation, and missing values for maximum wind gust speed could be best estimated by using the median.

If I were to start over and target areas I could improve, I would avoid the deviation from my original idea, as this massively cut down on my total time I could allocate to the project. I would also not underestimate the scale of the task of cleaning and preprocessing extremely raw data, as this then took up a substantial amount of my already limited time left to complete the project.

As a target for further development of the project, I would like to experiment more with the data itself in an attempt to improve performance. For example, experimenting with dropping different features, scaling the data differently, or finding more features to add to the data, such as humidity.

I also feel the neural networks require far more fine tuning, and there is a lot of untapped performance which I was not able to extract. It wasn't until late on in the project I discovered a library called keras-tuner, which is used similar to grid search to try and find the optimal architecture for the network, and by this point I did not have enough time left to implement the search and find better models.

However, the largest area for further study would be to experiment with how the models perform on data collected from different locations. If these models are only effective for the immediate area around the data collection point and new models have to be trained for each area of the world, then their practical application drops substantially. Leading on from this would be to create models capable of performing well in various climates - even if they need some minor adjustments to become optimal. I feel this is currently the weakest area of my project, as it is unclear whether my findings are applicable to any location other than the one from which the data was obtained.

I do believe it is studies like these that will end up driving change in the forecasting industry as machine learning models close in and eventually surpass the performance of current forecasting techniques. Until then, these models have potential to be used alongside professional forecasts as an extra layer of verification and robustness, and help inform which forecasts are most likely.

As a final note, the entire project has been an enormous learning experience from start to finish. Although I ran into many issues and problems, it was these that provided the greatest opportunities to learn. I have also enjoyed seeing a project be built from the ground up - starting with an idea, researching the domain and literature, planning my own implementation and executing it. While it is true that if I were to start over I would do things differently, this is ultimately down to the fact I have learnt much over the course of the project and the fact I would do things differently is a testament to that.

## References

[1] Jeffrey K. Lazo, Rebecca E. Morss, and Julie L. Demuth. 2009. 300 Billion Served. *Bulletin of the American Meteorological Society*, *90*(6), 785-798.

[2] JD Rudd. 2020. Spectrum News. Meteorologists are always wrong, right? Retrieved from https://spectrumnews1.com/wi/green-bay/weather/2020/10/08/wisconsin-weather-blog-meteorologist-wrong-rudd

[3] Met Office: How weather forecasts are created. Date Unknown. Retrieved from https://www.metoffice.gov.uk/weather/learn-about/how-forecasts-are-made

[4] Josie Garthwaite. 2021. Climate of chaos: Stanford researchers show why heat may make weather less predictable. Retrieved from https://news.stanford.edu/2021/12/14/warming-makes-weather-less-predictable/

[5] NASA Change. Date Unknown. Global Surface Temperature | NASA Global Climate Change. Climate Change: Vital Signs of the Planet. Retrieved from https://climate.nasa.gov/vital-signs/global-temperature

[6] Collins, M., R. Knutti, J. Arblaster, J.-L. Dufresne, T. Fichefet, P. Friedlingstein, X. Gao, W.J. Gutowski, T. Johns, G. Krinner, M. Shongwe, C. Tebaldi, A.J. Weaver and M. Wehner. 2013. Long-term Climate Change: Projections, Commitments and Irreversibility. In: Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change [Stocker, T.F., D. Qin, G.-K. Plattner, M. Tignor, S.K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex and P.M. Midgley (eds.)]. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA.

[7] Francois Chollet. (2017). Deep Learning with Python. Manning Publications, New York, United States.

[8] Weyn, J. A., Durran, D. R., Caruana, R., & Cresswell-Clay, N. 2021. Sub-seasonal forecasting with a large ensemble of deep-learning weather prediction models. *Journal of Advances in Modeling Earth Systems*, 13, e2021MS002502. https://doi.org/10.1029/2021MS002502

[9] Mark Holmstrom, Dylan Liu, Christopher Vo. 2016. Machine Learning Applied to Weather Forecasting. Stanford University, CA, United States.

[10] J. S. A. N. W. Premachandra and P. P. N. V. Kumara. 2021. "A novel approach for weather prediction for agriculture in Sri Lanka using Machine Learning techniques," *2021 International Research Conference on Smart Computing and Systems Engineering (SCSE)*, 2021, pp. 182-189, doi: 10.1109/SCSE53661.2021.9568319

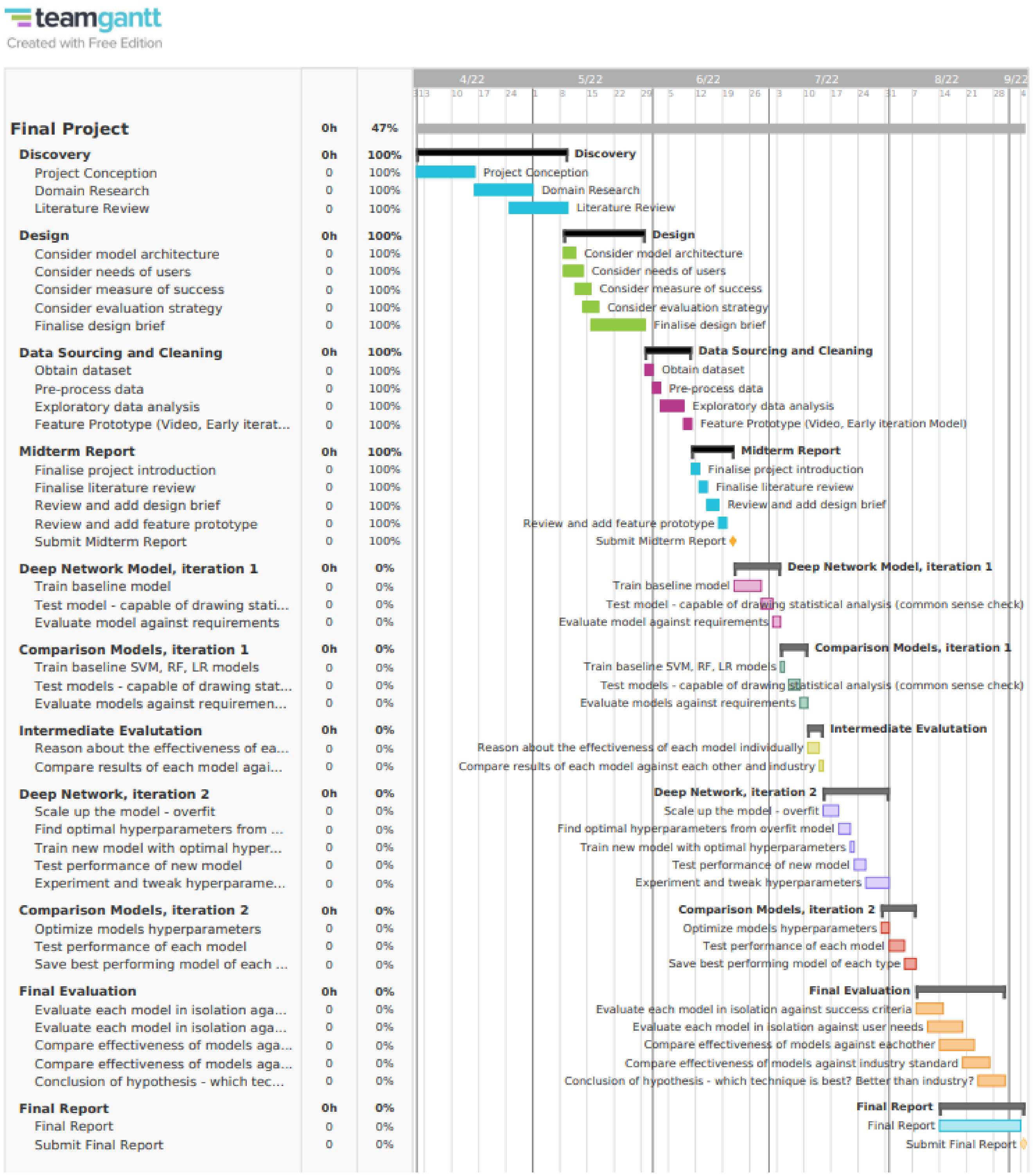
[11] N. Singh, S. Chaturvedi and S. Akhter. 2019. "Weather Forecasting Using Machine Learning Algorithms". *International Conference on Signal Processing and Communication (ICSC)*, 2019, pp. 171-174, doi: 10.1109/ICSC45622.2019.8938211.

[12] Zheng, X., Jia, J., Guo, S., Chen, J., Sun, L., Xiong, Y. and Xu, W., 2021. Full parameter time complexity (FPTC): A method to evaluate the running time of machine learning classifiers for land use/land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *14*, pp.2222-2235

[13] Abdiansah, A. and Wardoyo, R., 2015. Time complexity analysis of support vector machines (SVM) in LibSVM. *International journal computer and application*, *128*(3), pp.28-34.

## Appendix

Figure 1. Gantt chart for project development timeline.



| | | |
|---|---|---|
| **Final Project** | **0h** | **47%** |
| **Discovery** | **0h** | **100%** |
| Project Conception | 0 | 100% |
| Domain Research | 0 | 100% |
| Literature Review | 0 | 100% |
| **Design** | **0h** | **100%** |
| Consider model architecture | 0 | 100% |
| Consider needs of users | 0 | 100% |
| Consider measure of success | 0 | 100% |
| Consider evaluation strategy | 0 | 100% |
| Finalise design brief | 0 | 100% |
| **Data Sourcing and Cleaning** | **0h** | **100%** |
| Obtain dataset | 0 | 100% |
| Pre-process data | 0 | 100% |
| Exploratory data analysis | 0 | 100% |
| Feature Prototype (Video, Early iterat... | 0 | 100% |
| **Midterm Report** | **0h** | **100%** |
| Finalise project introduction | 0 | 100% |
| Finalise literature review | 0 | 100% |
| Review and add design brief | 0 | 100% |
| Review and add feature prototype | 0 | 100% |
| Submit Midterm Report | 0 | 100% |
| **Deep Network Model, iteration 1** | **0h** | **0%** |
| Train baseline model | 0 | 0% |
| Test model - capable of drawing stati... | 0 | 0% |
| Evaluate model against requirements | 0 | 0% |
| **Comparison Models, iteration 1** | **0h** | **0%** |
| Train baseline SVM, RF, LR models | 0 | 0% |
| Test models - capable of drawing stat... | 0 | 0% |
| Evaluate models against requiremen... | 0 | 0% |
| **Intermediate Evalutation** | **0h** | **0%** |
| Reason about the effectiveness of ea... | 0 | 0% |
| Compare results of each model agai... | 0 | 0% |
| **Deep Network, iteration 2** | **0h** | **0%** |
| Scale up the model - overfit | 0 | 0% |
| Find optimal hyperparameters from ... | 0 | 0% |
| Train new model with optimal hyper... | 0 | 0% |
| Test performance of new model | 0 | 0% |
| Experiment and tweak hyperparame... | 0 | 0% |
| **Comparison Models, iteration 2** | **0h** | **0%** |
| Optimize models hyperparameters | 0 | 0% |
| Test performance of each model | 0 | 0% |
| Save best performing model of each ... | 0 | 0% |
| **Final Evaluation** | **0h** | **0%** |
| Evaluate each model in isolation aga... | 0 | 0% |
| Evaluate each model in isolation aga... | 0 | 0% |
| Compare effectiveness of models aga... | 0 | 0% |
| Compare effectiveness of models aga... | 0 | 0% |
| Conclusion of hypothesis - which tec... | 0 | 0% |
| **Final Report** | **0h** | **0%** |
| Final Report | 0 | 0% |
| Submit Final Report | 0 | 0% |

Feature Prototype (Video, Early iteration Model)
Test model - capable of drawing statistical analysis (common sense check)
Test models - capable of drawing statistical analysis (common sense check)
Reason about the effectiveness of each model individually
Compare results of each model against each other and industry
Find optimal hyperparameters from overfit model
Train new model with optimal hyperparameters
Experiment and tweak hyperparameters
Save best performing model of each type
Evaluate each model in isolation against success criteria
Evaluate each model in isolation against user needs
Compare effectiveness of models against eachother
Compare effectiveness of models against industry standard
Conclusion of hypothesis - which technique is best? Better than industry?

Figure 2. Plots for evaluating performance of neural networks over training on the training and validation data sets

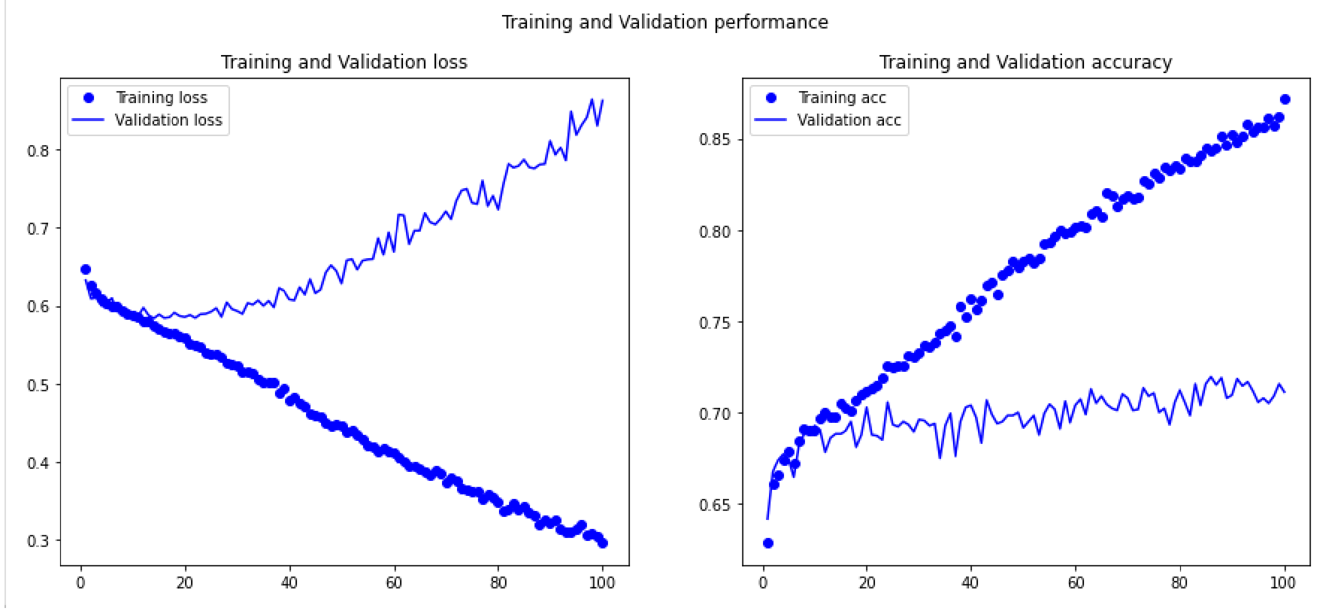
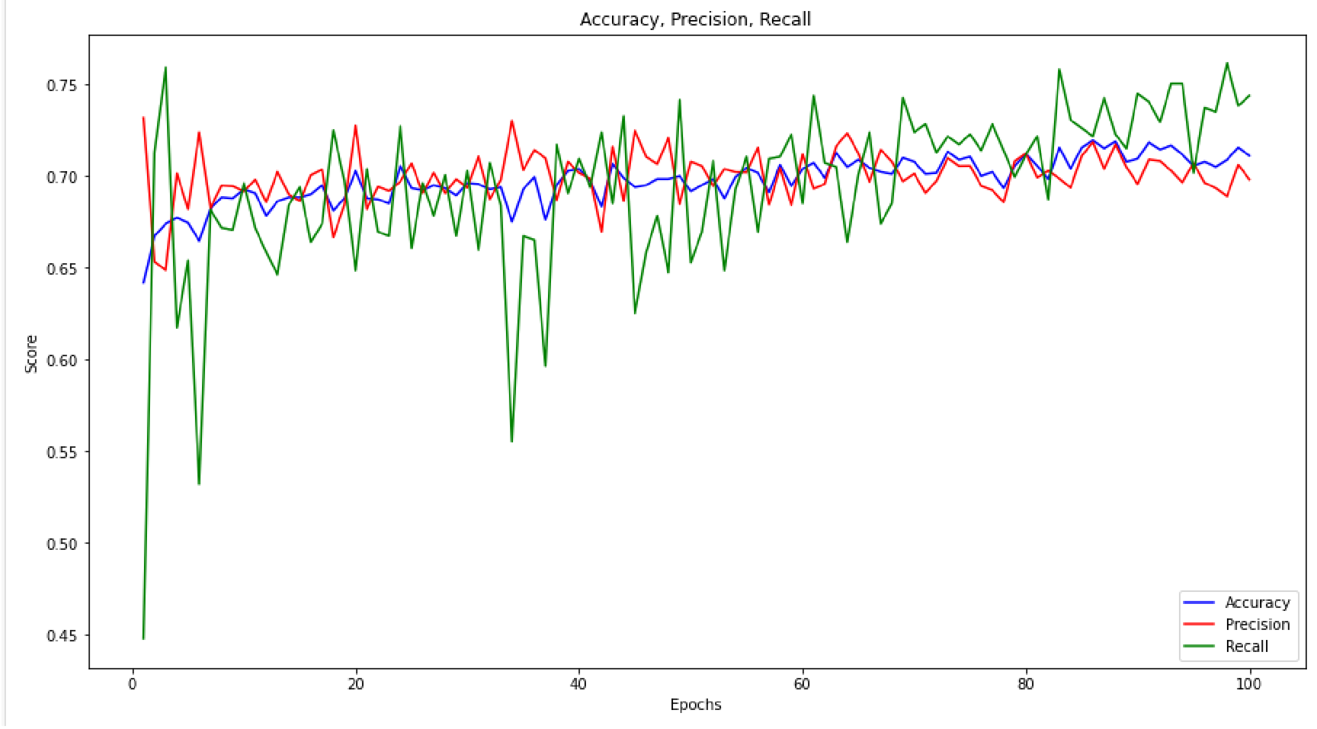


Figure 3. Plot for evaluating how validation accuracy, precision, and recall change over the course of neural network training


Accuracy, Precision, Recall
Score
Epochs
Accuracy
Precision
Recall