University of Reading

PhD in Computer Science

Meteorological Data Filtering for Tropical Cyclones using Deep Learning Techniques

Author:
Daniel Galea

Supervisors:
Prof. Bryan Lawrence
Dr. Julian Kunkel

Department of Computer Science

March 2022

Abstract

Tropical cyclones are severe weather events which have massive human and economic effect, so it is important to be able to understand how their location, frequency and structure might change in future climate. Climate models are utilised to create such projections, but they produce large amounts of data, most of which might not be relevant to understand tropical cyclones. To that end, a lightweight deep learning model is presented which is intended for detecting the presence of tropical cyclones in running numerical simulations. This model was trained using data from the ERA-Interim reanalysis dataset and obtained a recall rate of 92% on testing a previously-unseen dataset.

This deep learning model was then used to compare this to an observational dataset in the International Best Track Archive for Climate Stewardship (IBTrACS) and a state of the art tracking system (T-TRACK). This comparison shows how these methods differ with respect to a tropical cyclone's position, strength and structure. It was shown that stronger well-defined cyclones, i.e. those with a clear centre of circulation and with the least amount of noise in their fields, produce similar performance for both detection algorithms. On the other hand, weaker cyclones are only detected by one of the detection methods or are present only in the observational dataset. It was also shown that the deep learning model, while detecting the presence of a tropical cyclone, produced some ill-positioned tropical cyclone centres, especially for those in the Indian ocean. This was not surprising as the deep learning model was not trained for this function.

This deep learning model was then integrated into the Met Office's Unified Model to create an online filtering method for regional analysis data. It was shown that the method can mostly be applied to data originating from different simulations having different horizontal resolutions and for data coming from different simulated climates. It was shown that the method using a deep learning model trained on data obtained by simulations of the Met Office's Unified Model (UM) could be applied on ERA-Interim data, but not vice-versa and that a model trained on data reflecting the data it will be applied on will work better than a repurposed deep learning model. Also, the deep learning model can be applied to data of a higher resolution, albeit with a slight drop in performance, but not vice-versa. It was also shown that a model trained on data from the current climate can be applied to data from a simulated future climate, at least when TCs are considered. The effect of a change of labelling source was also discussed. The reduction in the volume of analysis data saved to disk was also quantified, as well as the method's computational cost.

Declaration: I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Tropical cyclones (TCs) are meteorological systems that leave a large impact in their wake. Hence, there is a need to predict the onset these systems to warn any people in their path. Also, with a changing climate, further studying how these systems will change is important. A tool that helps us do this is the modelling of the atmospheric system.

Over the years, a large amount of knowledge on how the atmospheric system functions has been built up. This knowledge is usually consolidated in the form of mathematical equations. Richardson (1922) imagined a group of 64,000 people, his computers, using these equations to predict the weather. This attempt failed as it was taking at least as long to produce the forecasts as for the weather to occur.

This changed when computers came into the picture. In 1950, John von Neumann and his group at Princeton University managed to use computers to produce forecasts up to 24 hours ahead, but took 24 hours to produce and were not very accurate. Despite this, they showed that it could be possible to model the Earth's atmosphere.

This progress in modelling continued when Phillips (1956) published a paper describing the first General Circulation Model (GCM). This was the first model that was capable of simulating the state of the atmosphere for the whole globe.

In the present day, these GCMs have become very complex and include components other than the state of the atmosphere, for example the states of the sea and ice, the chemistry of different components of the atmosphere and many other interactions. These are the main tool for studying how the world's climate might change in the future.

While these models are useful to understand how the climate and Earth system will change with a warmer climate, they are usually run at coarse horizontal resolutions. For a better representation of how climate change will effect a certain area, much higher horizontal resolutions are needed. For this, regional climate models (RCMs) are employed. These are higher resolution models which simulate the Earth system for a region of the Earth. However, they require boundary conditions which need to be obtained from the GCM.

Data outputted to disk from a GCM is processed to create the boundary conditions required for RCMs. Currently, however, all the data from a GCM is being written out to disk. Boundary conditions for an RCM are only needed when a meteorological system of note, a TC in this study, is present in the region being studied. This is what we are aiming to tackle in this study.

The method developed in this study has been focused on inferring the presence of TCs in meteorological data with the aim of determining when these boundary conditions are to be written out to disk from the GCM to be used later in the RCM, thus filtering out any unneeded data.

A few research questions that are to be answered in this study are laid out as follows:

- Can a deep learning model detect the presence of TCs with a good enough performance?

- How does such a deep learning model compare to a state-of-the-art non-machine learning TC-tracking algorithm? Is using the deep learning model in the climate model run better in terms of the accuracy of the technique when compared to a state-of-the-art non machine learning TC-tracking algorithm?

- Can the same deep learning model be used for data of varying resolutions, i.e. can the filtering technique be resolution-independent?

- Can the same deep learning model be used for different data sources, e.g. reanalysis vs model-produced data, i.e. can the filtering technique be data-independent?

- Can a deep learning model be easily included in FORTRAN-based climate model codes?

- How many gains are made from using the filtering technique? Is it worth implementing?

To answer these questions and show the steps taken during this study, the following text is structured as follows:

- Chapter 2 provides background information on the deep learning techniques and methods used further on. It then goes into some existing applications of these methods in the field of weather and climate. Some characteristics of TCs are described so that it is known what patterns are needed to be learnt and picked up by the deep learning model and what to look out for in subsequent analysis. Finally, some of the existing techniques, with and without machine learning, that have been used to detect and track TCs are described.

- Chapter 3 presents the deep learning model built to detect the presence of TCs in meteorological data, as well as the steps undergone to build it, as well as some checks to confirm that its predictions are produced as expected.

- Chapter 4 compares the deep learning model to a state-of-the-art non-machine learning TC tracking algorithm and an observational dataset.

- Chapter 5 describes how data can be filtered using an embedded inference engine using a deep learning model and how it was incorporated into a climate model. It also attempts to answer the research questions around the method's adaptability to data from different sources, as well as giving an idea of the computational cost incurred by running the method in a GCM run.
- Chapter 6 summarises the advancements achieved during this study.

Chapter 2

Background Information

This chapter provides background information on deep learning and tropical cyclones, and how deep learning and more conventional algorithms have been previously used for TC detection.

In Section 2.1, an overview of deep learning is provided. The chapter continues with an in-depth explanation of any specific methods used in Section 2.2. Section 2.3 details Convolutional Neural Networks (CNNs), which underpins the filtering technique being presented. The characteristics of Tropical Cyclones (TCs) are discussed in Section 2.4 to build enough knowledge such that any deep learning techniques employed can be sufficiently informed. Finally, some of the existing systems and methods, including those using machine learning and deep learning, to detect and track TCs are shown in Section 2.5, so that any techniques or systems developed in this study can be easily compared to any previously developed ones.

2.1 Deep Learning

Artificial Intelligence (AI) has been defined by the Cambridge dictionary as "the study of how to produce machines that have some of the qualities that the human mind has, such as the ability to understand language, recognize pictures, solve problems, and learn".

The field of AI, which is itself a subfield of computer science, contains a variety of methods and techniques, which include rule-based algorithms and genetic algorithms. Such methods have been in use for a long time, while others are relatively new. These newer methods are either previously known methods, for example feed-forward neural networks, which have become usable mostly because of the recent improvements in computer performance or are an evolution of these previously known methods, for example convolutional neural networks.

Some of the older methods include simple linear regression (Legendre (1805), Gauss (1809)), first introduced in the early 1800s, Support Vector Machines (SVMs), which were first implemented in 1996 (Boser, Guyon, and Vapnik (1996)), and shallow feed-forward networks, which were first implemented in 1943 (Mcculloch and Pitts (1943)). These methods are usually included in the subfield of AI known as Machine Learning (ML).

A further subfield of ML is Deep Learning (DL). This is a relatively new

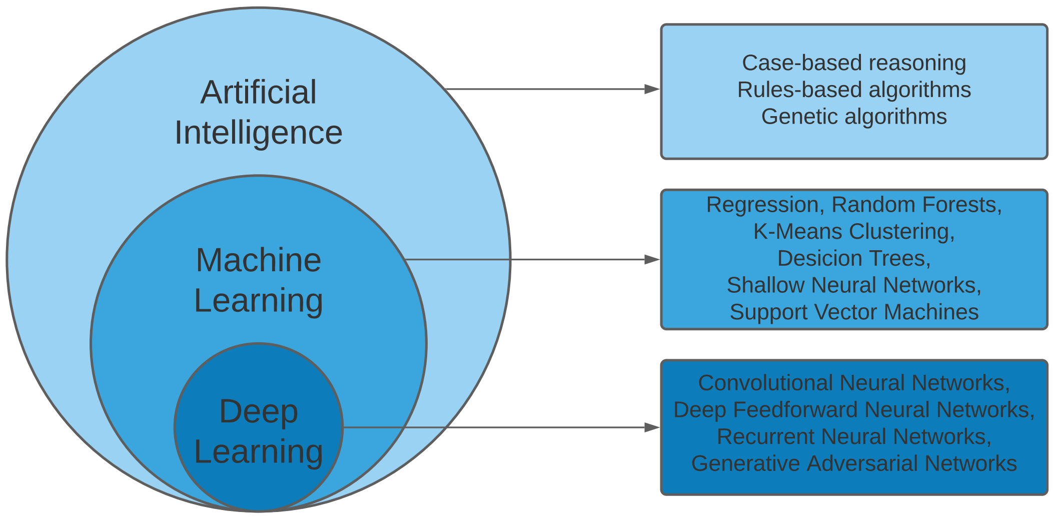


Figure 2.1: Fields in Artificial Intelligence

subfield and has come about largely due to improvements in computer performance, which allowed for the execution time of large computations to be reduced considerably, and improvements in the mathematics behind some of the methods. Deep artificial neural networks, convolutional neural networks and recurrent neural networks fall into this category.

2.2 Artificial Neural Networks

The initial neural networks developed were Artificial Neural Networks (ANNs). These are built on building blocks named perceptrons.

A perceptron, a diagram of which is shown in Figure 2.2, is made up of a few components: inputs, weights, a bias, an activation function, a loss function and its outputs.

Its inputs are usually floating point numbers which are normalised in some way to be in the range $[0, 1]$ or $[-1, 1]$. While normalisation is not necessary, it is strongly recommended. This process helps to keep the input distributions as in the original data but effectively resize their axes in relation to one another to stop the perceptron from over or under-utilising one part of the data more than others.

These inputs are each assigned a weight, θ, and a weighted sum is calculated. The initial weights can be assigned in various ways, from random initialisation to more sophisticated methods which will be delved into further at a later stage. A bias node is also usually included in the weighted sum. This is an input with a value of 1 and an associated weight. The bias node has the same effect as the intercept in a linear regression model.

Once the weighted sum is calculated, it is used with an activation function, which maps the weighted sum to the returned output. This signifies the end of the forward-pass. There are various activation functions one can use, with specific ones being more applicable to certain applications. Again, these will be discussed at a later stage. This forward-pass that the inputs take through

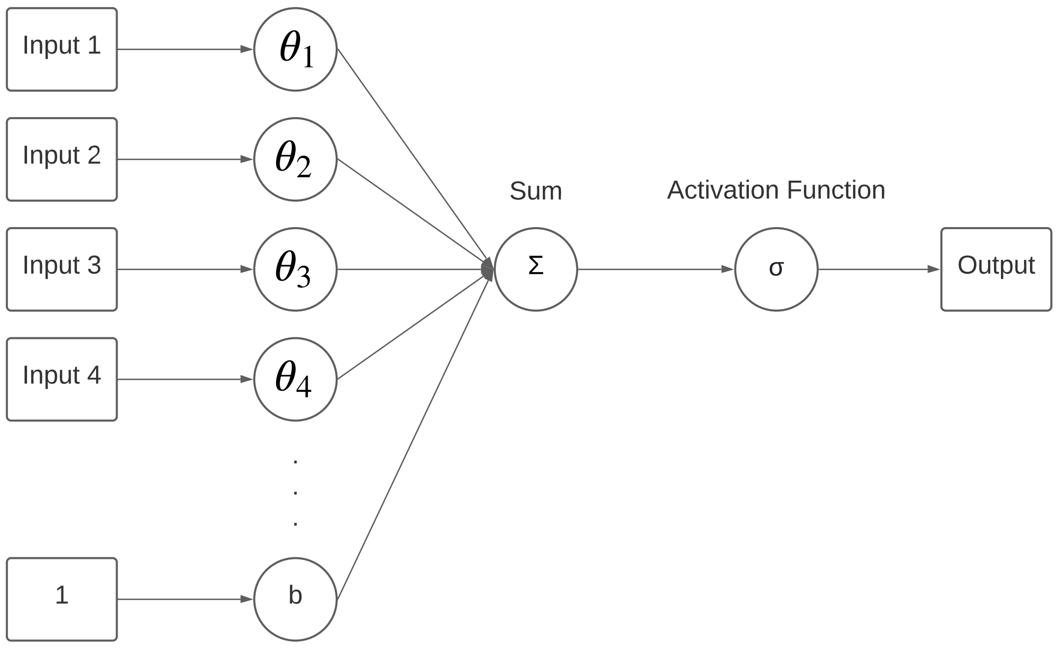


Figure 2.2: Perceptron

the deep learning model is the same process taken for the model to produce its inference when deployed.

In a supervised learning setting as used in this study, a deep learning model attempts to map the inputs given to a predetermined value, usually termed a label. To train the model, the backpropagation algorithm is employed where a forward-pass is performed and the output obtained is then compared to the given label. The difference between the two is calculated via a loss (or cost) function, of which there are many. Once the difference is known, the weights of the perceptron are updated, with the intention of getting an output that is closer to the given label.

The chain rule is used to calculate how much each individual weight is responsible for the error between the given output and label. In mathematical terms, this is given as the partial derivative of the difference between the output and the label and is obtained by:

$$\nabla_\theta J(\theta) = \frac{\partial\,\text{diff}}{\partial \theta_i} = \frac{\partial\,\text{diff}}{\partial\,\text{out}} \times \frac{\partial\,\text{out}}{\partial\,\text{sum}} \times \frac{\partial\,\text{sum}}{\partial \theta_i} \tag{2.1}$$

where "diff" is the difference between the output and the label as given by the loss function, "out" is the output generated by the forward-pass, "sum" is the weighted sum and θ_i is the weight for which an update is being calculated.

The previous result is then used to update the weight in question:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.2}$$

where η is the learning rate and takes values between 0 and 1 and $\nabla_\theta J(\theta)$ is the gradient of the difference between the generated output and the given label for the given weights θ. The learning rate controls how quickly weights are adapted to the problem. If this is too big, weights can fail to converge whereas if it is too small, it would take too long for weights to converge to an acceptable solution.

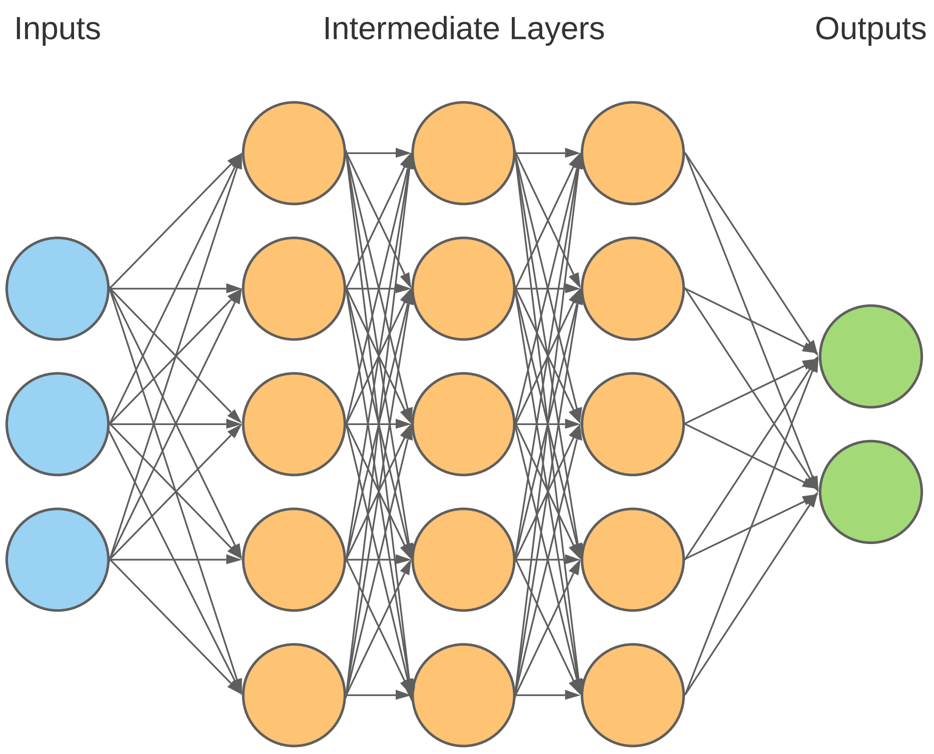


Figure 2.3: Fully-connected deep learning model

While various algorithms exist for calculating and updating model weights, most build on an early process named `Stochastic Gradient Descent (SGD)`. In this, the gradient of the difference between the output and the label with respect to each individual weight is being used to descend to the optimal weight. A variant of `SGD` is `Batch Gradient Descent`. This is where all the possible input cases are used to get the updated weights, but they are averaged before the weight update actually takes place. Most commonly `Mini-Batch Gradient Descent`, where only a section of n inputs is used at a time, is utilised (Ruder (2016)). `Batch Gradient Descent` and `Mini-Batch Gradient Descent` are usually used interchangeably. This helps to reduce the variation of updates produced by singular updates thus helping the model to converge more quickly, but still efficiently utilising the computational power available.

Deep learning models utilise these perceptrons by stacking them in layers to increase the complexity of the model, as shown in Figure 2.3.

These layers are intended to learn progressively more complex patterns which are composed from lower-level features that have been learnt in previous layers (Conneau et al. (2016)).

There are various architecture options, called hyperparameters, which were tested during this study. These include different loss functions, activation functions, dropout, and weight initialisation methods. A summary of these is tabulated in Table 2.1 and will be discussed below.

Type	Name	Expression
Loss Functions	Mean Absolute Error (MAE)	$MAE = \frac{\sum_{i=1}^{n} abs(y_i - \hat{y})}{n}$
	Mean Squared Error (MSE)	$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{n}$
	Binary Cross-Entropy (BCE)	$BCE = \frac{1}{n}\sum_{i=1}^{n} y_i \times log(p(y_i)) + (1 - y_i) \times log(1 - p(y_i))$
Optimisers	Stochastic Gradient Descent (SGD)	$\theta = \theta - \eta \cdot \nabla_\theta J(\theta, x^i, y^i)$
	SGD with Momentum	$v_t = \gamma v_{t-1} + \eta \cdot \nabla_\theta J(\theta); \theta = \theta - v_t$
	RMSProp	$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2; \theta_{t+1} = \theta_t - \frac{\eta}{E[g^2]_{t+\epsilon}}$
	Adam	$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t; v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2;$ $\hat{m}_t = \frac{m_t}{1-\beta_1^t}; \hat{v}_t = \frac{v_t}{1-\beta_2^t}; \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}+\epsilon}}\hat{m}$
	Adagrad	$\theta_{i+1} = \theta_t - \frac{\eta}{\sqrt{G_t+\epsilon}} \cdot \nabla_\theta J(\theta)$
	Adamax	$v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p)\lvert g_t\rvert^p; u_t = max(\beta_2 \cdot v_{t-1}, \lvert g_t\rvert);$ $\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t$
	Nadam	$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}(\beta_1 \hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}))$
Learning Rate	Rectified Linear Unit (ReLU)	$f(x) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0 \end{cases}$
	Leaky ReLU	$f(x) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \alpha x_i & \text{if } x_i < 0 \end{cases}$
	Exponential Linear Unit (ELU)	$f(x) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \alpha \exp x_i - 1 & \text{if } x_i < 0 \end{cases}$
	Hyperbolic Tangent (Tanh)	$f(x_i) = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}}$
	Softmax	$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$
Weight Initialisation	He	$\overline{x} = 0; \sigma = \pm\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}$
	Lecun	$\overline{x} = 0; \sigma = \sqrt{6/n_i}$
	Glorot	$\overline{x} = 0; \sigma = \sqrt{1/n_i}$
	Variance Scaling	$\overline{x} = 0; \sigma = \sqrt{a/n_i}$
Normalisation	Dropout	
	L2	$\frac{\alpha}{2}\sum_i \sum_j w_{ij}^2$
Dataset Balancing		

Table 2.1: Summary of hyperparameters used or tested while developing the deep learning model used in this study.

2.2.1 Loss Functions

A loss function is one that calculates the difference between the output produced by the deep learning model and the label given to the input data. The loss function is important as its output is to be minimized during the training of the model, so it is crucial to the final model's performance. There is an endless number of possible loss functions as these can be built around the intended use of the model, but their only requirement is to be differentiable. As will be seen further on, the output of loss function is used to calculate the weight updates during a model's training, which involves getting the gradient of the loss function with respect to each weight, hence the need for the loss function to be differentiable. Three of the standard loss functions have been used in this study and therefore will be explained below. These are the `mean absolute error (MAE)`, `mean standard error (MSE)`, and `binary cross-entropy (BCE)`.

`MAE` calculates the average of the magnitude of the error between the given prediction and label for all of the batch inputs. This is translated into the following mathematical equation:

$$MAE = \frac{\sum\limits_{i=1}^{n} |y_i - \hat{y}|}{n} \tag{2.3}$$

where y_i is the prediction given by the model, $\hat{y}$ is the label for the associated input and n is the number of cases in the batch considered. This loss function is usually used in regression problems as it works best for continuous values of y_i and $\hat{y}$.

MSE is very similar to the previous loss function. It calculates the average of the squared difference between the given prediction and associated label for all of the batch inputs. This is translated into the following mathematical equation:

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{n} \tag{2.4}$$

where y_i is the prediction given by the model, $\hat{y}$ is the label for the associated input and n is the number of cases in the batch considered. This loss function is also usually used in regression problems as it works best for continuous values of y_i and $\hat{y}$.

The final loss function considered is Binary Cross-Entropy (BCE). As the name suggests, it is used in models that attempt to classify an input case into one of two possible classes. In such situations, cases are labeled either 0 or 1, corresponding to each class and the model outputs a single value that ranges in value between 0 and 1. This is interpreted as the probability of the case being in the class given the label of 1. Inversely, the probability of the input being classified as being in the class labelled by 0 is 1 minus the prediction given by the model.

This behaviour is used in the BCE loss function, sometimes referred to as the log loss function, to produce a loss function based on the log of the probability given by the model. The mathematical equation is as follows:

$$BCE = \frac{1}{n} \sum_{i=1}^{n} y_i \times \log(p(y_i)) + (1 - y_i) \times \log(1 - p(y_i)) \tag{2.5}$$

where n is the number of cases in the batch used, y_i is the label given to an input case and $p(y_i)$ is the probability of the input being in the class denoted by the label 1 as given by the model.

This uses the negative log value of the difference between the probability of an input case of being in a certain class and the actual label class as a penalising factor. This is desired as this factor increases exponentially with larger differences, so small differences are penalised lightly, while larger differences are penalised much more harshly.

2.2.2 Optimisers

An optimiser is an algorithm that is employed to update a model's weights to converge to a viable solution. The various variants of Stochastic Gradient Descent (SGD) already described are some examples. However, as Ruder (2016) discusses, SGD has various problems namely:

- choosing an appropriate learning rate can be difficult,
- not being able to update only certain weights in case of sparse data or inputs of different frequencies
- it can be slow to converge to a viable solution
- it can easily get stuck in local minima

To help alleviate some of these problems, various algorithms have been developed. Some of those tested in this study, and thus described below, are `SGD with momentum` (Qian (1999)), `RMSprop`[1], `Adam` (Kingma and Ba (2014)), `Adagrad` (Duchi, Hazan, and Singer (2011)), `Adamax` (Kingma and Ba (2014)) and `Nadam` (Dozat (2016)).

`SGD` is known to suffer from being slow to converge in places in the optimisation space where the slope in one dimension is much larger than the others (Sutton (1986)), which usually happens near local optima. `Momentum` (Qian (1999)) helps to accelerate `SGD` in the direction of the optimum. This is done by adding a fraction of the previous weight update to the current weight update, as shown in equation 2.6, where η controls the magnitude of the previous weight update on the next update. This effectively amplifies updates in the direction of the optimum and dampens those in other directions.

$$\begin{aligned} v_0 &= \nabla_\theta J(\theta)_0 \\ v_t &= \gamma v_{t-1} + \eta \cdot \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{2.6}$$

To help address the issue of regulating the size of the updates of different weights, `Adagrad` (Duchi, Hazan, and Singer (2011)) was developed. This algorithm changes the learning rate, i.e. the size of the weights' update, to each of the weights θ_i so performing larger updates for those weights rarely changed and smaller updates for those changed more frequently. This algorithm achieves this goal by keeping a separate learning rate for each weight. These learning rates are calculated by considering the squared gradient of the loss function with respect to the individual weight for all the previous updates. The actual weight update, shown in equation 2.10, is performed by using using a fraction of the global learning rate, the magnitude of which is controlled by the individual weight's learning rate, G_t, which is the sum of squares of gradients with respect to θ_i.

$$\theta_{i+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta) \tag{2.7}$$

One drawback of `Adagrad` is that the size of the weight updates is monotonically decreasing over time, thus still taking longer than optimal to converge to a solution. `RMSprop` aims to fix this by giving the latest updates more importance when calculating the individual weights' learning rate. This is done by only using the last updates for the calculation, as well as using a time-dependent exponentially-decaying average of these updates, hence the latest update has more importance and than that of two updates before, and so on. Equation 2.8 show the weight update, with $E[g^2]_t$ being the exponentially decaying average for the sum of squares of gradients with respect to θ_i up to the current weight update.

[1]presented in `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

$$\begin{aligned} g_t &= \nabla_\theta J(\theta) \\ E[g^2]_t &= 0.9E[g^2]_{t-1} + 0.1g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{E[g^2]_t + \epsilon} \end{aligned} \tag{2.8}$$

`Adam` (Kingma and Ba (2014)) is another adaptive learning rate algorithm. Similar to `RMSprop`, it keeps a time-dependent exponentially-decaying average of the past updates, m_t, but it also keeps a time-dependent exponentially-decaying average of the past updates' gradients v_t, similar to what is done in momentum. Both of these are included as factors when calculating the individual weight's next update, shown in equation 2.9. As a consequence of this, `Adam` can be seen as a combination of `RMSprop` and momentum.

$$\begin{aligned} g_t &= \nabla_\theta J(\theta) \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1)g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}} + \epsilon}\hat{m} \end{aligned} \tag{2.9}$$

Kingma and Ba (2014) realised that `Adam` scales the gradient used to be inversely proportional to the l_2 norm of the past gradients and the current gradient. However, they show that the $p = l_\infty$ norm converges to a more stable value so they proposed `AdaMax` (Kingma and Ba (2014)) which is the same as Adam with the only difference being that the l_∞ norm is used instead of the l_2 norm during its calculations. The AdaMax weight update is shown in equation 2.10, with $p = 2$ to obtain `Adam` and $p = \infty$ to obtain `AdaMax`.

$$\begin{aligned} v_t &= \beta_2^p v_{t-1} + (1-\beta_2^p)|g_t|^p \\ u_t &= max(\beta_2 \cdot v_{t-1}, |g_t|) \\ \theta_{t+1} &= \theta_t - \frac{\eta}{u_t}\hat{m}_t \end{aligned} \tag{2.10}$$

Similar to `Adam` being considered a combination of `RMSprop` and `momentum`, `Nadam` (Dozat (2016)) is considered as a combination of `Adam` and `Nesterov accelerated gradient` (`NAG`, Nesterov (1983)).

`NAG` is similar to momentum as it also aims to speed up `SGD` by directing the algorithm to the right point in the optimisation space. However, momentum can overshoot the optimal point if too much momentum is applied. Therefore, `NAG` tries to taper its momentum as the algorithm approaches the optimal point. It does this by calculating the gradient at the approximate next weight update before making said weight update. The approximate point of the next weight update is obtained by using the difference between the past and present weights and extrapolating to obtain an approximate next point, as shown in equation 2.11.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{2.11}$$

Therefore, `Nadam` uses the `Adam` algorithm, but the gradients used in the factor when calculating the individual weight's update are changed to those computed on an approximate future weight, as computed by `NAG`. The resulting weight update is given in equation 2.12.

$$\begin{aligned} g_t &= \nabla_\theta J(\theta - \gamma v_{t-1}) \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}(\beta_1 \hat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}) \end{aligned} \tag{2.12}$$

2.2.3 Learning Rate

The learning rate controls the size of the step made when an optimiser updates the model's weights during training. With `SGD`, this hyperparameter is very important to the final model's performance. Picking too small a learning rate would mean that training would struggle to obtain the optimal solution for the model's weights, and by extension the model's performance. This could be due to the optimiser getting stuck in a suboptimal local optimum or just taking too long to converge, thus making it infeasible to wait the required amount of time for the optimiser to converge. On the other hand, too large a learning rate would mean that the optimiser skips over the optimal solution and leads to unrealistic weight values. As can be inferred, the right value for the learning rate when using `SGD` as an optimiser heavily depends on the underlying optimisation space as given by the loss function utilised.

This hyperparameter has become important to tune since the advances in optimisers used, namely those adopting adaptable learning rates. However, the initial global learning rate might still make some difference in the performance of the end model.

2.2.4 Activation Functions

Activation functions are used to insert non-linearity into a deep learning network. As shown in Figure 2.2, an activation function maps the weighted sum of the inputs onto a single value. An activation function, therefore, converts the learned mappings to a single value via a non-linear mapping. If an activation function is not used in a perceptron, the building block of a deep learning network, then it would produce the same behaviour as simple linear regression. Stacking such perceptrons and including activation functions gives a deep learning network to produce a regression of higher orders.

The activation function can be a factor in a specific problem that deep learning models can suffer from – exploding or vanishing gradients – so careful consideration to which one is chosen can determine the model's final performance. This problem occurs when the gradients calculated during a the training of a model are far away from the value of 1 and there are many layers in the network. Many successive multiplications of small gradients would produce very small weight updates, leading to the vanishing gradient problem, while multiple multiplications of large gradients would produce very large weight updates, leading to the exploding gradients problem. An activation function can help alleviating these problems by mapping their inputs to a value close to 1.

There are various standard activation functions but those used during this study are described below: the `rectified linear unit` (`ReLU`, Nair and Hinton (2010)); `leaky ReLU`; the `exponential linear unit (ELU)`; the `hyperbolic tangent (Tanh)`; and the `softmax` function.

`ReLU` (Nair and Hinton (2010)), the rectified linear unit, is one of the most commonly used activation functions. It maps its inputs to themselves if their value is greater than 0 and to 0 if their value is less than 0. In mathematical terms, it performs the following mapping:

$$f(x) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0 \end{cases} \tag{2.13}$$

This activation function is useful as it can be computed very quickly. Since the gradient of `ReLU` can either be 0 or 1, it manages to evade the vanishing gradients problem as the activation function is implicitly used in the ∂out term in Equation 2.1. This action of omitting values less than 0 can lead to a desirable side-effect where not all weights are updated in each training step as some will have a gradient of 0, but this behaviour can lead to neurons that do not update ever if their gradients stay at 0. Also, due to using a linear function for values larger than 0, it can to prone for any network that utilises it to overfit to the data if the data is composed mostly of values above 0.

To rectify this problem, `leaky ReLU` (Maas, Hannun, and Ng (2013)) was proposed to solve the issues described in `ReLU`. Instead of setting any negative values to 0, it suppresses them to a fraction of their original magnitude. In mathematical terms, it performs the following calculation:

$$f(x) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \alpha x_i & \text{if } x_i < 0 \end{cases} \tag{2.14}$$

where α takes a value less than 1, usually around 0.01. This solves the problem of non-updating neurons.

The `exponential linear unit (ELU, Clevert, Unterthiner, and Hochreiter (2016))`, tries to solve the same problems in a slightly different way. In `leaky ReLU`, any negative values are treated linearly, albeit with an associated factor. This means that in theory, there is no lower limit for any negative value. In `ELU`, the exponential of any negative value is used and the value of 1 is then subtracted. In mathematical terms, it follows the following computation:

$$f(x) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \alpha \exp x_i - 1 & \text{if } x_i < 0 \end{cases} \tag{2.15}$$

The value of 1 is usually used for α so this effectively implies a lower limit of -1 for any output. This helps the deep learning network avoid any biases by pushing the mean activation value throughout the network to be closer to zero than any other other activation function examined so far, and thus helps the network converge to a suitable solution during training quicker than it would otherwise.

A zero-centered mean activation value is able to be produced by the `hyperbolic tangent (Tanh)` activation function. It does this via the calculation of

$$f(x_i) = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}} \tag{2.16}$$

This is a zero-centered function, which is why a zero-centered mean activation function can be obtained, which can help the training of a deep learning network converge to an optimal solution rather quickly. However, it cannot handle the vanishing gradient problem.

While the previous functions are usually used in the hidden layers of a deep learning network, the `softmax` (Goodfellow, Bengio, and Courville (2016)) activation function is usually used in the output layer when the problem to be solved is one of classification. The `softmax` function takes its inputs and outputs a classification probability according to:

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.17}$$

It produces on output with values between 0 and 1 with it corresponding to a probability of the input being in a certain class. In a binary classification problem, it corresponds to the probability of the inputs being in the class labelled by the value of 1 with the probability of being in the class labelled by 0 being 1 minus the value given. In multi-classification problems, the activation function is used on each of the output nodes and each output corresponds to the probability of the inputs being in a certain class, with all the probabilities summing to 1.

2.2.5 Weight Initialisation Methods

An important factor when training a deep learning network is how to set the starting weights when starting the network's training. The effectiveness of these methods are linked to the activation functions used in the network. In this study, the following weight initialisation methods were tested: `He` (He et al. (2015)), `Lecun` (LeCun et al. (2012), Klambauer et al. (2017)), `Glorot` (Glorot and Bengio (2010)) from a uniform distribution and `VarianceScaling`.

At the start of deep learning, biases were set to 0 and the weights were initialised to random values. While this worked for the initial models, this was far from optimal as training took a longer time to converge to an acceptable solution. Also, due to large weight possible, the exploding gradients problem was common.

The `Glorot` weight initialisation method (Glorot and Bengio (2010)) was devised to solve these problems. At the time, the most common activation functions were those that were centered about a midpoint and travelled a certain distance away from this midpoint. An example of such activation functions is the `Tanh` function seen previously. The authors then used a method that produced weights that were also centered around zero and bounded by a certain distance from zero. They did this by using the number of inputs and outputs that the weight's layer is connected to. Therefore, a layers weights were initialised to be in the uniform distribution bounded by:

$$\pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \tag{2.18}$$

where n_i is the number of input nodes to and n_{i+1} is the number of output nodes for the layer for which the weights are being calculated and $\sqrt(6)$ is the variance of a uniform random variable.

`He` (He et al. (2015)) weight initialisation is usually used with the `ReLU` activation function. This also sets the weights of a certain layer using values from a normal distribution of mean equal to zero but with a standard deviation of $\sqrt{6/n_i}$, using only the number of inputs as a variable.

`Lecun` (LeCun et al. (2012), Klambauer et al. (2017)) weight initialisation continues with this trend of using the number of inputs to define a normal distribution for setting a layer's weights. This uses a distribution of mean 0 and a standard deviation of $\sqrt{1/n_i}$. `VarianceScaling` is very similar to `Lecun` but uses a standard deviation defined by $\sqrt{a/n_i}$ where a is a value selected by the user. During testing, the default value of $a = 1$ was used, which made the `VarianceScaling` and `Lecun` weight initialisation methods the same.

2.2.6 Dropout Regularisation

One issue with deep learning networks, especially those with a large number of intermediate layers or those trained on a small dataset, is that of overfitting. This is when the model learns to replicate the exact examples given in the training dataset but is not able to generalise to other examples from the same distribution, so the performance of the model on never seen before examples is less than on the training set. The issue arises because the large amount of tunable parameters will be able to produce a model which approximates the exact dataset that the model is trained on. `Dropout` (Srivastava et al. (2014)) aims to remedy this issue.

In a world with unlimited computational power and time, the way to solve this issue is to train an ensemble of networks, made up of all the possible combinations of network architectures, and then using an average of the inferences given by all of the models in the ensemble. In reality, this is not possible so `dropout` aims to approximate this by making different random nodes inactive across different stages of training by setting their weights and gradients to zero. This means that any connections made to the node are also made inactive. This behaviour means that effectively different architectures are being trained on the dataset during the model's training. It should be noted that no nodes, and by extension any of their connections, are inactive during inference outside of the training steps, which is why a deep learning model utilising dropout is able to approximate an ensemble of models.

Added benefits from the use of `dropout` include that the network does not rely on certain nodes to produce the right inference, hence making the network more robust. At the same same, it helps the network produce a sparse representation of the inputs. One negative side-effect is that `dropout` effectively thins out the network during training, so an extra amount of nodes have to be included for the network to not become too small to learn an accurate representation of the inputs.

2.2.7 L2 Normalisation

Another way of avoiding overfitting is the use of `L2 normalisation`. This method involves adding a term to the loss function that penalises any weights in the network that are too large or too small. The penalising term is calculated using

$$\frac{\alpha}{2}\sum_i\sum_j w_{ij}^2 \tag{2.19}$$

where w_{ij} is weight j of layer i, and α is called the regularisation rate, which controls the magnitude and hence the effect of the regularisation.

The loss function used then becomes

$$\hat{L} = L + \frac{\alpha}{2}\sum_i\sum_j w_{ij}^2 \tag{2.20}$$

with L being the original loss function and $\hat{L}$ the amended loss function.

Then, the weight update with this updated loss function becomes

$$w_i^{new} = w_i - \eta \times (\alpha w_i + \frac{\partial \text{diff}}{\partial w_i}) \tag{2.21}$$

The effect that this type of normalisation has on the network is that weights are encouraged to be close to 0, and thus reducing the importance of the hidden layers and so reducing the opportunity to overfit during training. The only issue that remains is that of choosing the value for α. This is a hyperparameter that needs to be tuned as well but too large a value for α would mean that too simple a model is being trained, thus producing an underfitted network and too large a value for α would produce too complex a model, thus producing an overfitted network.

2.2.8 Dataset Balancing

A common issue in classification problems is that of a highly skewed dataset, i.e. one of the classes in the dataset dominates the dataset. This is a problem as the model would tend to produce an inference that favours that class. For example, a model trained on a dataset that is skewed 1000:1 towards class 1 of 2, will almost always predict that an input belongs to class 1 if the training dataset is not balanced. Such a model would produce an accuracy of 99.99% but it would be unusable in practice, as it would have no ability to classify inputs between the two classes present. This becomes very costly when picking the minor class with a good ability is important, for example in fraud detection applications.

To remedy this issue, a few techniques have been used in this study, and thus are described below. These are

- `Naive Oversampling`
- `Weighting the cases`
- `Adding Bias`
- `Weighting the Cases and Adding Bias`
- `Undersampling without Replacement`
- `Undersampling with Replacement`

The obvious method to use is to oversample the minority class(es). This means that cases of any minority class are copied multiple times to make the number of examples of the class in the training set approximately equal to the number of cases in the majority class. This has two drawbacks. The first is that the trainig set is expanded - by a considerable factor if the training dataset is highly skewed - putting more pressure on the computational and time resources. Also, the effect of copying cases means that the model is exposed to training on a small number of examples, thus making it easier to overfit onto these examples. This can be offset somewhat by the use of `data augmentation`, where the examples are modified slightly for the examples to not be exactly the same, but the risk of overfitting will still remain.

Another way to produce a better performing model would be to make the model learn more from the cases in the under-represented class/es. This is done by assigning a weight to each class and use these weights in the loss function during training. Therefore, any loss for the under-represented class(es) is amplified while any loss of the over-represented class(es) is dampened, resulting in a model that learns from each class approximately equally. One issue of this method is that it is still prone to overfitting.

It is also possible to add bias to the model to reflect the skewness in the dataset. This is calculated as $\log_e(\text{pos}/\text{neg})$, where pos is the number of positive cases and neg is the number of negative cases, and is added to all the nodes in the network. This pre-conditions the network to start training from a better initial point in the optimisation space, hence speeding up the training. A combination of weighting the cases and adding bias was also trialled during the development of our model.

The final two methods trialled to help with the skewness of the dataset are those of undersampling the majority class with and without replacement. Undersampling the majority class has the advantage of making the dataset somewhat smaller, thus making training times quicker, while making the training dataset more equal in distribution. Undersampling without replacement means that only a section of the cases of the majority class is used for training. The disadvantage of doing this is that perfectly usable data is not used for training. Also, if the initial dataset is highly skewed, the end training dataset would become very small, so the final model is likely to be overfit.

A better way would be to undersample the majority class with replacement, i.e. different parts of the majority cases are used at different steps during training, effectively having different training datasets over the training process. This method utilises all the available data while presenting a balanced dataset to the model during training. The model is still prone to overfitting on the minority class/es cases but this risk can be somewhat mitigated by the use of `data augmentation`.

2.3 Convolutional Neural Networks

All of the previous explanations are based on the basic feedforward network but apply to any type of deep learning network. Multiple types of deep learning networks exist and are used according to the problem that is being addressed and the type of data used. In this study, the deep learning model is aimed at trying to classify meteorological data into one of two classes. For this intended use, a type of network called a Convolutional Neural Network (CNN), an example of which is shown in Figure 2.4, is used. This type of network is adept at learning and extracting patterns that have a spatial dependency, hence it s useful on the meteorological data used in this study.

A CNN is preferred to a feedforward network due to the much smaller number of parameters used while managing to learn the spatial relationships that make up the patterns to be learnt.

There are two important operations present in a CNN that makes it different and have fewer parameters than a feedforward network. The first is the convolution operation. In this operation, the inputs, usually an image of multiple channels, is traversed using a set of weights called a convolutional window, which acts as a learnable filter. This set of weights is used to compute a dot product of all the input's values inside the window with all the window's weights. This produces a single value. When the window traverses the whole image, the values produced are collated back into an image, which is the output of a convolutional layer. The number of filters learned are a hyperparameter of a CNN. The space left between windows when traversing the inputs, called the stride length, is also a hyperparameter. This final parameter is used to control the amount of downsampling of the inputs between convolutional layers.

The effect of downsampling is to compress the inputs to a high fidelity representation inside the network by optimising the filters during training. This is desired as smaller intermediate images used require less weights to process and to be learnt, and hence make the trained network less prone to overfitting. It also helps to place less pressure on any computational resources in use.

The second important operation is that of pooling. This operation is also meant to downsample the inputs or any outputs produced by any convolutional layers. There are various ways to downsample any inputs but there are two methods which are commonly used. These are called MaxPooling and Average-Pooling. In both of these methods, a window is used to traverse any inputs. The maximum or average of any values inside this window is then calculated. These are then collected in a similar way to that done in a convolutional layer to produce a downsampled version of the inputs. Similar to the convolution operation, the window size and the stride length used betwen movements of the pooling window are hyperparameters to be tuned.

After stacking these convolutional and pooling layers, the final outputs are a dense representation of the initial inputs. These inputs are reshaped into a 1D array and then passed to a traditional feedforward network for the final inference. Therefore, the stack of convolutional and pooling layers are termed the convolutional base and the feedforward network is termed the classifier (or a regressor if a regression problem is being solved).

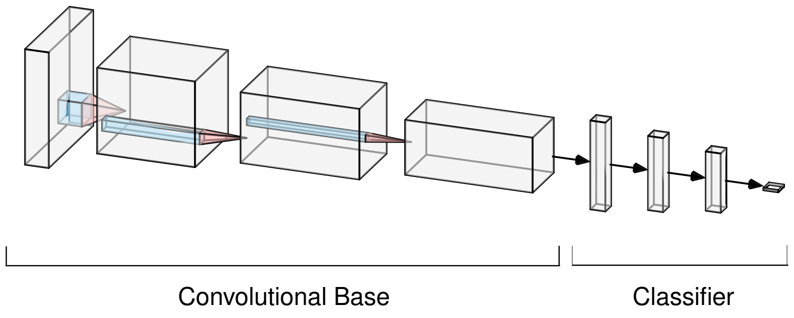


Figure 2.4: Example of a CNN, showing a convolutional base attached to a classifier. Blue areas are convolutional windows and red areas are pooling operations.

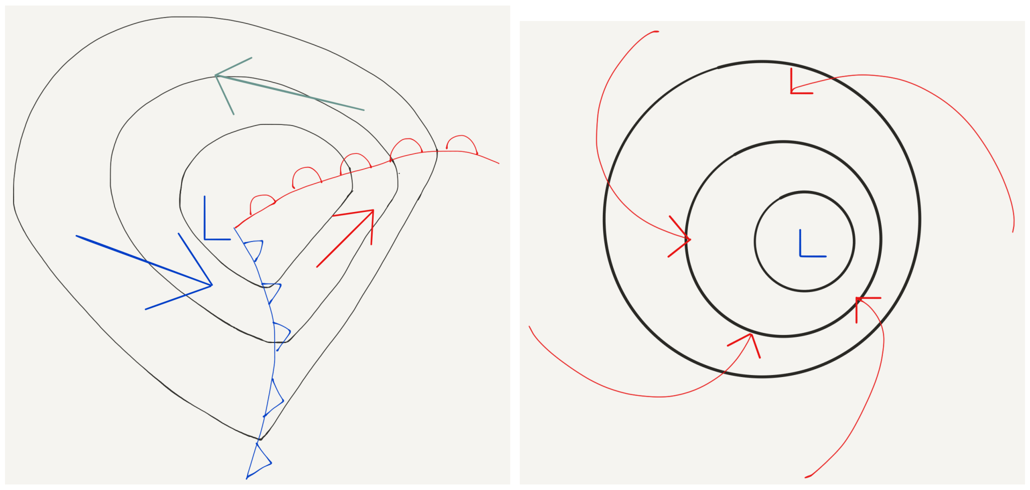
2.4 Tropical Cyclone Characteristics

Tropical Cyclones (TCs) are extreme weather events that can leave devastating effects on human populations; for example Hurricane Irma impacted the Caribbean Islands and the Southeast USA in September 2017 causing 47 direct deaths, 82 indirect deaths, hundreds of injuries and an estimated monetary damage to the US of around 50 billion USD (Cangialosi, Latto, and Berg (2018)).

The National Oceanic and Atmospheric Administration's (NOAA) National Hurricane Center (NHC) define a TC as "a warm-core non-frontal synoptic-scale cyclone, originating over tropical or subtropical waters, with organized deep convection and a closed surface wind circulation about a well-defined center".

A cold core system, for example an extratropical cyclone, has cold air at the centre of the system. Figure 2.5a, an idealised diagram of a cold core frontal system, shows that the types of air are present - cold air (cyan), warm air (red) and colder air (blue). Fronts are used to help distinguish between the three. In this type of system, the warm air is forced to rise due to the colder air undercutting it. Hence, while there is warm air aloft, colder air is present at and near the surface. Also, due to the temperature gradients present, the isobars (black curves, curves of constant pressure) are not concentric around the pressure low.

A warm core system, on the other hand, is termed as such due to a convergence of warm air at the centre of the system. This is shown in Figure 2.5b via the red arrows. This convergence of air usually occurs due to latent heat release from evaporation of warm ocean water causing the warm air to rise, creating a low pressure centre. This is shown with the black lines that show the isobars are expected to be concentric around the pressure low.



(a) Cold Core system (b) Warm Core system

Figure 2.5: Pressure fields (black lines) and wind flows (arrows) for a cold core system (a) and a warm core system (b).

TCs originate in an area called the Intertropical Convergence Zone (ITCZ). This area is characterised as a broad area of low pressure around the place of maximum solar radiation. The ITCZ flutuates about the equator as the seasons change, so is mostly in the Northern Hemisphere during its summer, and the same of the Southern Hemisphere.

Air in this area is usually heated by the warm oceans, which are being warmed by the strong sun, forcing it to rise. This air condenses as it rises, producing thunderstorms. This rising column of air produces an area of low pressure and can be seen as such in the mean sea level pressure (MSLP) field. These thunderstorms can combine to make up a cluster of thunderstorms, which helps sustain them due to stronger rising columns of air, called updraughts. This cluster then starts to rotate as it interacts with the spin of the Earth and thus being the start of a TC.

Some meteorological conditions are also needed for the cluster of thunderstorms being able to turn into a TC, namely:

- a source of warm air, usually provided by a warm ocean, of temperatures of at least 27 °C, heating a layer of air above it
- winds blowing towards the same point from different directions, thus forcing the air to rise
- low wind shear, i.e. similar wind speeds and directions throughout all levels of the atmosphere, to allow the column of rising air to extend into the atmosphere as much as possible
- enough distance from the equator for the Coriolis effect to impart spin on the cluster of thunderstorms

As the cluster of thunderstorms matures into a TC, a column of descending air also becomes present. This can become so strong that it prevents any condensation within it, which in mature TCs can be seen as the eye of a TC.

As the TC matures, the strength of the updraughts increase, thus deepening the low pressure centre of the TC. This increases the gradient of air pressure thus increasing the strength of the wind speeds at the centre of the circulation, with the strongest winds usually found at the centre of the circulation just outside the eye, in a region called the eyewall. These strong horizontal wind speeds occurring in a concentric fashion around the eye makes such mature TCs relatively clear to pick out in meteorological data by a trained individual.

The maximum sustained wind speed of a TC, usually measured as the average wind speed for 1 minute, is used to classify a TC's strength. The Saffir–Simpson hurricane wind scale (SSHWS) is the standard way to do this for TCs in the Northern Pacific ocean and the Atlantic ocean. It classifies a TC into one of five categories, as below:

- Category 1: Maximum wind speed between 33ms^{-1} (119kmh^{-1}) and 42ms^{-1} (153kmh^{-1})
- Category 2: Maximum wind speed between 43ms^{-1} (154kmh^{-1}) and 49ms^{-1} (177kmh^{-1})
- Category 3: Maximum wind speed between 50ms^{-1} (178kmh^{-1}) and 58ms^{-1} (208kmh^{-1})
- Category 4: Maximum wind speed between 58ms^{-1} (209kmh^{-1}) and 70ms^{-1} (251kmh^{-1})
- Category 5: Maximum wind speed greater than 33ms^{-1} (251kmh^{-1})

Two additional classifications exist whereby any TC with a maximum wind speed between 63kmh^{-1} and 118kmh^{-1} is termed a Tropical Storm and any TC with a maximum wind speed less than 63kmh^{-1} is termed a Tropical Depression. TCs in other parts of the world use slightly different systems depending on the meteorological agency issuing the classifications, but most do not differ by much from the ones outlined above. Also, the size of a TC can range from a few hundred to around a thousand km.

A TC comes to the end of its lifespan when one or more of the conditions listed above are not met. This usually happens due to interaction with land, as it can cut off the source of warm air flowing over the ocean and/or produce higher wind shear further aloft in the atmosphere, or the TC moving North where colder waters are encountered.

One important meteorological variable associated with TCs and used in our model and other techniques is relative vorticity. This is a measure of the spin given to an air parcel due to its flow. The flow of air is shaped by areas of high or low pressure or the land surface. Air moving in a straight line has zero vorticity. However, air in and around a high pressure system moves clockwise (in the Northern Hemisphere, anti-clockwise in the Southern Hemisphere) around the center of the system and gets negative vorticity. On the other hand, air moving in and around an area of low pressure moves in an anti-clockwise direction (in the Northern Hemisphere, clockwise in the Southern Hemisphere) around the center of the system and gets positive vorticity. The speed of the flow is also important, as the higher the speed, the higher the relative vorticity. Hence, areas of very high vorticity are found at the centre of TCs.

2.5 Previous TC detection systems

There have been many algorithms designed to detect and track TCs. Most of these are classical threshold-based algorithms, but deep learning has also been used in a few of the newer algorithms. An overview of both the former and the latter will be described.

2.5.1 Classical algorithms

There is extensive previous literature detailing different automatic TC detection algorithms that do not utilise any type of machine learning.

Conventional techniques usually work by identifying TC centres by applying various thresholds to the available data. TC tracks are then created by arranging the identified TC centres according to some mathematically-based method. The following show a few examples of such methods, with a tabular summary in Table 2.2.

Kleppek et al. (2008) use multiple thresholds to identify the TC centers. The first is that a local minimum of sea-level pressure (SLP) needs to observed within a neighbourhood of eight grid points. This is assigned as a storm centre. For it to be designated a TC centre, there needs to be a maximum relative vorticity at 850hPa above 5×10^{-5} s^{-1} at the storm centre. The presence of vertical wind shear between 850hPa and 200hPa of at least 10 ms^{-1} is also required, as well as an event lifetime of 36 or more hours. Finally, if the storm centre is over land, the relative vorticity condition has to be fulfilled or the wind speed maximum at 850hPa needs to be inside 250km from the TC centre.

Similarly, Vitart, Anderson, and Stern (1997) used the closest minimum of mean sea level pressure (MSLP) to a local maximum of relative vorticity at 850hPa over 3.5×10^{-5} s^{-1} as a storm centre. A warm-core check, similar to Roberts et al. (2015), is performed to classify the storm centre as a TC. This requires that the closest local maximum of the average temperature between 550hPa and 200hPa must be within 2^{o} of the storm centre and the temperature decreases by at least 0.5^{o}C for at least 8^{o} latitude in all directions. Also, the closest maximum thickness between 1000hPa and 200hPa must be within 2^{o} of the storm centre and the thickness must decrease at least 50 metres for at least 8^{o} latitude in all directions.

Camargo and Zebiak (2002) introduce a detection method that uses vorticity at 850hPa, surface wind speed and a vertically integrated temperature anomaly as variables on which to impose basin-dependant thresholds.

A final example is Roberts et al. (2015) who use the method explained by Hodges (1995), Hodges (1996), Hodges (1999) and Bengtsson, Hodges, and Esch (2007), where a TC is identified by a maximum of 850hPa relative vorticity, in data which has been spectrally filtered using a T42 filter (i.e. keeps features greater than 250km in scale) and a warm-core check on a T63 grid (to keep features larger than 180km) using the 850hPa, 500hPa, 300hPa and 200hPa levels.

There is also available literature comparing such algorithms: Horn et al. (2014) compare four different detection algorithms, namely a modified version of the Commonwealth Scientific and Industrial Research Organisation (CSIRO) tracking scheme (Walsh et al. (2007), Horn, Walsh, and Ballinger (2013)), the Zhao tracking scheme (Zhao et al. (2009)), and those developed by the modelling groups whose data was involved, i.e. the groups from the Meteorological Research Institute (MRI), the National Aeronautic and Space Administration (NASA) Goddard Institute for Space Studies (GISS) and the Centro Euro-Mediterraneo per i Cambiamenti Climatici-Istituto Nazionale di Geofisica e Vulcanologia (CMCC-INGV). The models used were the CMCC-INGV ECHAM5 model which has ∼90-km grid spacing at equator (Roeckner et al. (2003)), the NASA-GISS model which has ∼110-km grid spacing at the equator (Schmidt (2014)), the National Center for Environmental Prediction (NCEP) Global Forecast System (GFS) which has ∼110-km grid spacing at equator (Saha (2014)), and version 3.2 of the Meteorological Research Institute Atmospheric General Circulation Model (MRI AGCM3.2) which has ∼60-km grid spacing at equator (Mizuta et al. (2012)).

They showed that the group method was at worst equal-best when comparing TC counts to observations and usually outperformed the others. This is due to the group having tuned their method's thresholds to obtain close to the number of TCs observed. They also show that detection methods which weren't optimised on the data being tested do not work as well as if they had been optimised. Similarly, Onogi et al. (2007) also found that a detection algorithm developed for the Japanese Meteorological Agency (JMA) obtained 80% of TCs in their JRA-25 reanalysis but less than 60% of TC in the ERA-40 reanalysis (Uppala et al. (2005)).

Given that the requirement for these automatic tracking algorithms is to detect TCs in a particular set of data which correspond to those that occurred in real-life, it is only natural that the threshold values are tuned to obtain the same number of TCs.

Zarzycki and Ullrich (2017) conducted sensitivity analysis on the thresholds used for one tracking algorithm, TempestExtremes (Ullrich and Zarzycki (2017)) applied to four different reanalysis datasets. They found that the most sensitive thresholds were those defining the TC vortex strength, for example for the depth of the minimum of sea level pressure (SLP) or warm core strength. They reported a larger difference when comparing storm count rather than integrated or weighted metrics such as the number of days with a TC present or accumulated cyclone energy (ACE). Zhao et al. (2009) also found that the threshold for minimum duration of a TC was sensitive to the choice made while previous literature also seems to agree that even though some differences might be observed between detection methods, there are little disagreements on strong TCs, i.e. those that are at least of category 1 on the Saffir-Simpson scale.

It was also noted in some of this work that the intensity of TCs, whether surface winds or the depth of the minimum mean sea level pressure (MSLP), is underestimated in all of the reanalyses datasets. Strachan et al. (2013) noted that resolution alone does not explain this observation due to more feedback

Author/s	**Variable**	**Threshold**
Kleppek et al. (2008)	Sea Level Pressure (SLP)	Local minimum in a neighbourhood of eight grid points
	Relative Vorticity at 850hPa	$> 5 \times 10^{-5}$ s^{-1}s and positioned at SLP minimum
	Vertical Wind Shear between 850hPa and 200hPa	> 10 ms^{-1}
	Event time	> 36 hours
	SLP Minimum Position	If over land, relative vorticity at 850hPa $> 5 \times 10^{-5}$ s^{-1}s and positioned at SLP minimum; Otherwise, wind speed maximum at 850hPa needs to be inside 250km from the TC centre
Vitart, Anderson, and Stern (1997)	Relative Vorticity at 850hPa	Local maximum $> 3.5 \times 10^{-5}$ s^{-1}
	Mean Sea Level Pressure (MSLP)	Minimum closest to relative vorticity local maximum – taken as storm centre
	Average Temperature between 550hPa and 200hPa	Closest maximum within 2^{o} of the storm centre and the temperature decreases by at least 0.5^{o}C for at least 8^{o} latitude in all directions
	Maximum Thickness between 1000hPa and 200hPa	Closest maximum within 2^{o} of the storm centre and the thickness must decrease at least 50 metres for at least 8^{o} latitude in all directions
Camargo and Zebiak (2002)	Relative Vorticity at 850hPa	> twice the vorticity standard deviation in each basin
	Surface Wind Speed	> the sum of wind speed standard deviation in each basin and the global average oceanic wind speed in a 7x7 box centered around the relative vorticity maximum
	Sea Level Pressure (SLP)	A local minimum is present in a 7x7 box centered around the relative vorticity maximum
	Temperature Anomaly	Anomaly averaged over a 7x7 box centered around the relative vorticity minimum and over the 300, 500, 700 hPa pressure levels > the basin standard deviation
		Anomaly averaged over a 7x7 box centered around the relative vorticity minimum is positive in all three of 300, 500, and 700 hPa pressure levels
		Anomaly averaged over a 7x7 box centered around the relative vorticity minimum is greater at 300hPa than at 850hPa
	Wind Speed	Wind speed averaged over a 7x7 box centered around the relative vorticity minimum is greater at 850hPa than at 300hPa
	Distance Travelled	Storm centre – defined as the relative vorticity minimum – must not have travelled a distance greater than 5.6^{o} if 6-hourly output or 8.5^{o} if daily output
	Event Time	At least 1.5 days if 6-hourly output or 2 days if daily output
Roberts et al. (2015)	Relative Vorticity	$> 6 \times 10^{-5}$ s^{-1} at 850 hPa
		Reduction of at least 6×10^{-5} s^{-1} in vorticity between 850 and 250 hPa at a T63 resolution
		Positive vorticity centre at all available levels between 850 and 250 hPa
	Wind Speed	Wind speed averaged over a 7x7 box centered around the relative vorticity minimum is greater at 850hPa than at 300hPa
	Event Time	At least 1.5 days

Table 2.2: Overview of thresholds applied to meteorological variables for detecting and tracking Tropical Cyclones with the conventional techniques given.

processes present in the model. Despite this, they still noted that any wind speed threshold should vary linearly with resolution and any deviations from this relationship are due to model biases and errors.

Schenkel and Hart (2012) also noted that the choice of data assimilation method is important to get realistic surface wind speeds. For this reason, the JRA25 and JRA55 reanalyses are most realistic, due to the vortex relocation step performed during their creation.

Despite all these considerations when it comes to the resolution of different reanalyses, Strachan et al. (2013) show that those datasets with a resolution higher than 60km are capable of showing the correct inter-annual variability but even a resolution of 20km is not capable of producing the right intensities.

Hodges, Cobb, and Vidale (2017) investigated how TRACK (Hodges (1995), Hodges (1996), Hodges (1999)) performed using six different reanalysis datasets. It was found to work well (97% in NH; 92% in SH) at tracking TCs across all basins, but that it had a high false detection rate, especially in the Southern Hemisphere. Most of these false positives had their genesis at a latitude greater than 20°S, leading to the conclusion that these may have been hybrid TCs of some sort. An additional conclusion was that the observations may have missed recording some storms as there were around 20% more advisories issued than storms present in the data. Hodges, Cobb, and Vidale (2017) opined that such storms may have been omitted due to the lack of human impact and/or accurate measurements.

2.5.2 Algorithms utilising Deep Learning

A newer crop of algorithms have been developed to detect and track TCs using deep learning methods. These are summarized in Table 2.3.

Racah et al. (2017) created a method where a deep learning model takes in a snapshot of the world as given by a CAM5 climate model with 16 different meteorological variables and creates bounding boxes around the detected TCs. The variables used were: total precipitation rate, surface pressure, sea level pressure, reference height humidity, temperature at 200 millibar and 500 millibar, total vertically integrated precipitable water, reference height temperature, radiative surface temperature, meridional and zonal wind speed at 850 millibar and at the lowest available model level, geopotential at 100 millibar and 200 millibar and the lowest model level height. The architecture used was that of an auto-encoder with three smaller networks using the bottleneck layer to draw a box around a suspected TC. Given the size of the inputs and number of kernels used in the convolution layers, the model presented was expected to be time consuming to train. It certainly required supercomputing: an adaptation of this deep learning model was trained using 9622 nodes of 68 cores each with a peak throughput of 15.04PF/s and reached a sustained throughput of 13.27 PF/s, although the total time to train was not reported (Kurth et al. (2017)). The accuracy for this model was specified as the percentage of overlap between the predicted box and the box given as the ground truth — an Intersection of Union (IOU) — which was created using the Toolkit for Extreme Climate Analysis (TECA) (Prabhat et al. (2012), Prabhat et al. (2015)). The model had 24.74% of the predicted boxes having at least an overlap of 10% with the ground truth, while 15.53% of the predicted boxes had at least an overlap of 50% with the ground truth.

Mudigonda et al. (2017) created a deep learning model which used integrated water vapour (IWV) snapshots and image segmentation techniques to classify whether each pixel in an image was a part of a TC or not. It used an adaptation of the Tiramisu model, which applies the DenseNet architecture to semantic segmentation. The labels were created using TECA (Prabhat et al. (2012), Prabhat et al. (2015)) and Otsu's method (Otsu (1979)). It was trained and tested on images which had at least 10% of the pixels which were not background pixels. An accuracy of 92% was obtained but it was noted that had the model predicted all the pixels as being all background pixels, the accuracy would have been of 98%.

Similarly, Kumler-Bonfanti et al. (2020) use a U-net network to perform image segmentation for TCs using the total precipitable water field from the GFS output as an input, where the Global Forecast System (GFS) is a National Centers for Environmental Prediction (NCEP) weather forecast model. A U-net network is very similar to an auto-encoder network, with the difference that connections between the two branches of the network are used to convey any information missed while creating the dense representation of the inputs at the end of the encoder branch of the network. The inputs used also take in a measure of time as two snapshots are given to the deep learning model, one approximating the state of the atmosphere at the time the inference was initiated and another having the forecasted state of the atmosphere at the next forecast step available, usually three hours after. Labels of areas belonging to a TC were generated by creating a 25x25 pixel box, approximating 300km^2, around a latitude and longitude pair obtained from International Best Track Archive for Climate Stewardship (IBTrACS, Knapp et al. (2010), Knapp et al. (2018)) dataset. This model managed to obtain an IOU of 75% but it should be noted that the labelling boxes were of the same size, so the value for the IOU is possibly inflated.

Finally, Liu et al. (2016) used an image cropped in such a way that if a TC was present, it was centred in the image. They used 8 different meteorological variables: surface level pressure, meridional and zonal wind speed at 850 millibar and at the lowest available model level, temperature at 200 millibar and 500 millibar and total vertically integrated precipitable water. They then predicted whether the image was one of a TC or not. The model obtained a 99% accuracy with a relatively simple model, but the pre-processing cropping step involved significant noise reduction, which would have helped obtain good performance.

Authors	Data Used	Ground Truth	Architectures	Results

Racah et al. (2017)	CAM5 Climate Model
	1979-2005 3-hourly
	25km resolution
	Image size of 768 x 1158 pixels
	16 channels
	
	Training Set:
	Timesteps during 1979
	
	Testing Set:
	Timesteps during 1984
		TECA	Encoder:
			Conv: 8(layers) x 384(height) x 576(width) @ 64(kernels)
			Conv: 8 x 192 x 288 @ 128
			Conv: 8 x 96 x 144 @ 256
			Conv: 8 x 48 x 72 @ 384
			Conv: 8 x 24 x 36 @ 512
			Conv: 8 x 12 x 18 @ 640
			
			Decoder:
			Conv: 8 x 12 x 18 @ 640
			Conv: 8 x 24 x 36 @ 512
			Conv: 8 x 48 x 72 @ 384
			Conv: 8 x 96 x 144 @ 256
			Conv: 8 x 192 x 288 @ 128
			Conv: 8 x 384 x 576 @ 64
			
			Box Locator:
			Conv: 4 x 12 x 18 @ 4
			
			Class Probabilities:
			Conv: 4 x 12 x 18 @ 4
			
			Objectiveness Probabilities:
			Conv: 4 x 12 x 18 @ 2
				IOU = 0.1: 24.74%
				IOU = 0.5: 15.53%

Liu et al. (2016)	CAM5.1 Climate Model 1979-2005 3-hourly 0.23^o x 0.31^o res. ERA-Interim Climate Model 1979-2011 3-hourly 0.25^o x 0.25^o res. 20th Century Reanalyses 1908-1948 daily 1^o x 1^o res. NCEP-NCAR Reanalyses 1949-2009 daily 1^o x 1^o res. Images cropped to 32 x 32 pixels	TECA Manual expert labelling	Conv: 5 x 5 @ 8 Pool: 2 x 2 Conv: 5 x 5 @ 16 Pool: 2 x 2 Dense: 50 Dense: 2	99%
Kumler-Bonfanti et al. (2020)	GFS NWP model 720 x 361 pixels 0.5^o res.	IBTrACS	U-net of 6 layers	IOU = 100%: 75%
Mudigonda et al. (2017)	CAM5 Climate Model 1996-2015 Images cropped to 96 x 144 pixels	TECA Otsu's method	Adaptation of Tiramisu Model	92%

Table 2.3: Previous Deep Learning models that detect and track Tropical Cyclones

Chapter 3

Detecting the Presence of Tropical Cyclones

Tropical cyclones are severe weather events which have large human and economic effect, so it is important to be able to understand how their location, frequency and structure might change in future climate. Here, a lightweight deep learning model is presented which is intended for detecting the presence of tropical cyclones in running numerical simulations. This has been developed to investigate the avoidance of saving vast amounts of data for analysis by filtering data during simulations to save only relevant data. Subsequent analysis workflow can target that data, avoiding the need to save all simulation outputs for cyclone analysis. The model was trained on ERA-Interim reanalysis data from 1979 to 2017 and the training concentrated on delivering the highest possible recall rate (successful detection of cyclones) while rejecting enough data to make a difference in outputs. When tested using data from the two subsequent years, the recall rate was 92% and the precision rate was 36%. However, if relevant meteorological events are considered to be positive cases, the effective precision was 85%. The recall rate compares favourably with other more standard deep learning models having the best Area Under Curve for the Precision/Recall (AUC-PR) and using the smallest number of parameters for both training and inference.

There have been a few deep learning-based methods to detect and track TCs as described above, but they either require significant preprocessing of the data to work, use a large amount of inputs or use a complicated network to produce their inferences. If one of these options are used in a method to detect or track TCs during a running simulation of the climate, significant computational resources would need to be dedicated to this task. Here, a new deep learning model that aims to detect the presence of TCs using quick data processing techniques and a relatively simple network is presented.

This chapter has been prepared for publication and a preprint version is available at https://www.essoar.org/doi/10.1002/essoar.10508127.1; contributions - Daniel Galea: coding, data collection, writing, editing; Bryan Lawrence: writing, editing, direction; Julian Kunkel: setting up initial data and direction

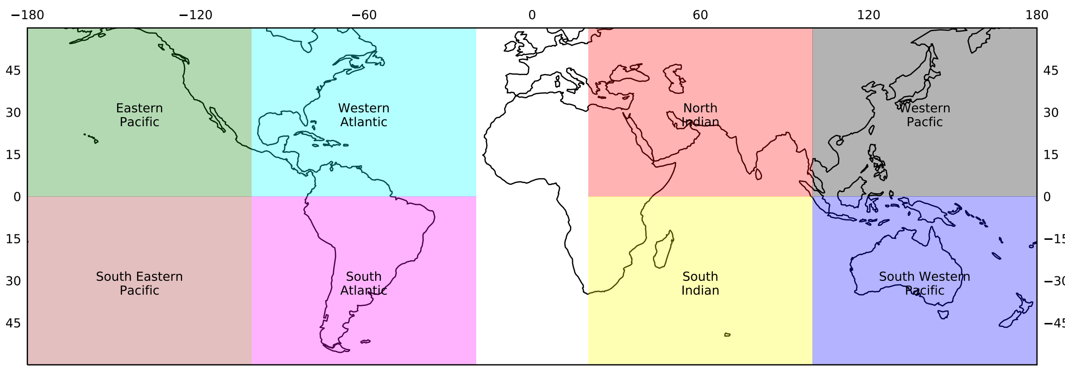


Figure 3.1: Figure showing how each timestep in the ERA-Interim dataset was split into 8 equal parts to create the training dataset for the Deep learning model.

3.1 Deep Learning Model

3.1.1 Data

Our deep learning model, named TCDetect, is trained and tested on a dataset extracted from the ERA-Interim reanalysis (Dee et al. (2011)).

Five six-hourly fields were used: mean-sea level pressure (MSLP), 10-metre wind speed, vorticity at 850hPa, vorticity at 700hPa and vorticity at 600hPa, each at a spatial resolution of $\approx 0.7^{o}$x0.7^{o} from the 1st of January 1979 until the 31st of July 2017. These fields were chosen because they had been used in previous TC detection algorithms and produced the best-performing deep learning model during hyperparameter tuning, as shown in the Section 3.1.3. Spherical filtering was performed on each field to reduce some of the smaller scale features. For the MSLP and 10-metre wind speed fields, spherical harmonic filtering is performed to keep wave numbers between 5 and 106. The vorticity fields were spherical harmonic filtered between wave numbers 1 and 63.

Each field was further split into eight regions as shown in Figure 3.1. These regions are loosely based on those used by the International Best Track Archive for Climate Stewardship (IBTrACS, Knapp et al. (2010), Knapp et al. (2018)) dataset. Thus, the whole dataset had 450,944 cases, each with dimensions of 86 rows, 114 columns and 5 channels.

Labels for these cases were derived from the IBTrACS dataset, which contains temporal information, including category, latitude and longitude of the storm centre, on all major storms across the globe. The labels were set up in such a way that each region was labelled according to the presence or absence of a TC in that case in IBTrACS. At the end of the labelling process, 22,826 (5.06%) positive labels were generated as well as 428,118 (94.94%) negative labels.

This dataset was used for training and validation; it was split by taking data from 1979, 1986, 1991, 1996, 2001, 2006, 2011 and 2016 to make up the validation set and the rest of the data to make up the training set. This method of splitting the available data was chosen so that the possible effects of a changing climate were taken into consideration, so that any hyperparameter tuning performed would not be skewed.

Partition	Years Included	Positive Cases	Negative Cases
Training	1980 - 1981, 1982 - 1985, 1987 - 1990 1992 - 1995, 1997 - 2000, 2002 - 2005 2007 - 2010, 2012 - 2015, 2017	17862 (5.00%)	339546 (95.00%)
Validation	1979, 1986, 1991, 1996 2001, 2006, 2011, 2016	4853 (5.19%)	88651 (94.81%)
Testing	2017 - 2019	1342 (5.51%)	23010 (94.49%)

Table 3.1: Dataset Splits

After splitting the available data, the training set had a total of 357408 cases, with 339,546 (95%) not having a TC and 17,862 (5%) having a TC. The validation set had a total of 93,504 cases, with 88,651 (94.81%) not having a TC and 4,853 (5.19%) having a TC.

Data from the 1st of August 2017 until the 31st of August 2019 was used as a testing set. This had a total of 24,352 cases, with 23,010 (94.49%) not having a TC and 1,342 (5.51%) having a TC present. Table 3.1 shows how the splits are made and that the split between positive and negative cases is mostly kept.

All data was preprocessed to reduce resolution ta a sixteenth of the original ERA-Interim resolution by taking the mean value of all data points inside a 4x4 box, thus reducing the dimensionality of each case to 22 rows, 29 columns and 5 channels. This reduction of resolution was also arrived to during hyperparameter tuning. Then, standardisation for each of the fields used as the input variable for each channel was performed according to

$$field = \frac{field - \mu_{field}}{\sigma_{field}} \tag{3.1}$$

where $\mu field$ is the mean of the values in the field and σ_{field} is the standard deviation of the values in the field, to standardize each value around 0 with a standard deviation of 1.

Figure 3.2 shows an example of the data used, before and after preprocessing, from the time when Hurricane Katrina obtained its maximum strength, the 28th of August 2005 at 18Z.

3.1.2 Architecture

TCDetect uses a convolutional base attached to a fully-connected classifier which outputs a value between 0 and 1, with any values larger than 0.5 signifying that the model detects a TC and any values smaller or equal to 0.5 meaning that the model does not detect a TC.

An input of dimensions 22 rows by 29 columns by 5 fields is fed through five convolutional blocks, each made up of a convolutional layer of 8, 16, 32, 64 and 128 kernels respectively, with weights initialized using the Glorot Uniform method (Glorot and Bengio (2010)) with ReLU (Nair and Hinton (2010)) activation functions, each with strides of 1 and a kernel size of 2x2; a dropout layer (Srivastava et al. (2014)) with a dropout rate of 10%; and a maximum pooling layer with strides equal to 1. The resulting kernels are flattened and passed through three fully-connected blocks, each made up of a dense layer of

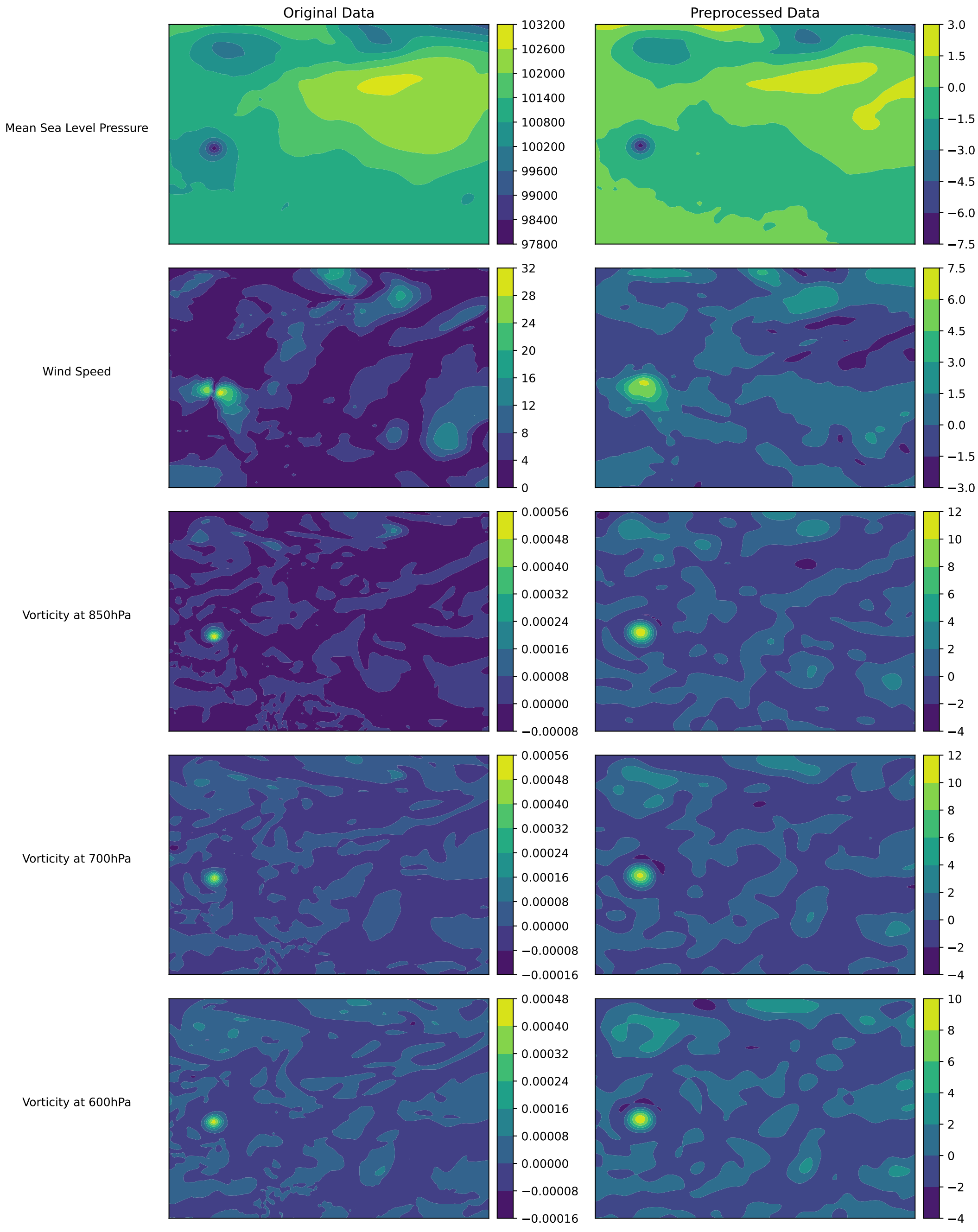


Figure 3.2: An example of the data that was used to train the Deep Learning model. Left column shows data from ERA-Interim and right column show how this data is transformed after preprocessing. The fields of Mean Sea Level Pressure (MSLP) (first row), 10-metre wind speed (second row), vorticity at 850hPa (third row), vorticity at 700hPa (fourth row) and vorticity at 600hPa (fifth row) from the timestep of the 28th of August 2005 at 18Z, i.e. the timestep when Hurricane Katrina obtained its maximum strength.

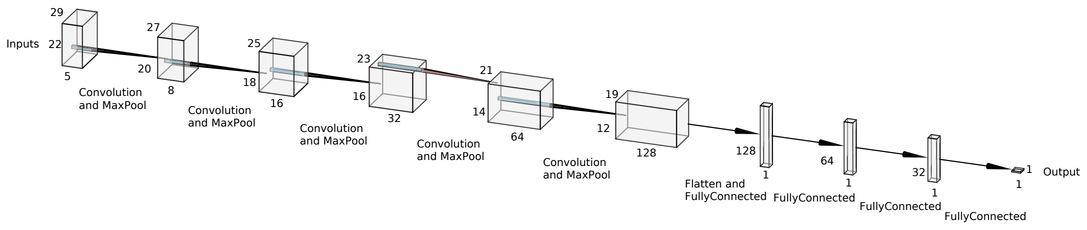


Figure 3.3: Visual representation of the architecture of TCDetect. The inputs, having 22 rows, 29 columns and 5 fields, are passed through a 2x2 convolution window whose weights are learnt producing 8 feature maps, but losing one row and column. The resulting feature maps are passed through a MaxPool window, of size 2x2, which takes the maximum value in its window. This further reduces the feature maps' size by a row and a column, to 20 rows and 27 columns. This is repeated for 3 more times, with each step producing more feature maps. The final ones of these are combined and reshaped into one long array. This array is used as the input to a fully-connected layer of 128 nodes, where all the values in the array are passed onto a mathematical calculation which gives out a single value. Hence, 128 values are now obtained and are used as inputs to the next layer. This is repeated for three more times, ending with a final inference.

128, 64 and 32 hidden nodes respectively, with L2 normalisation with a normalisation factor of 0.005, weights initialized by the Glorot Uniform method and a dropout layer with a dropout rate of 10%. TCDetect finishes off with another fully-connected layer of one node, this time using the sigmoid activation function with weights initialized by the Glorot Uniform method, as well as L2 normalisation with a normalisation factor of 0.005 which outputs a prediction. The optimizer used was the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.01 and momentum of 0.8 with the loss function being that of binary cross-entropy. A graphical view of this architecture is shown in Figure 3.3 and is detailed in Table 3.2.

To arrive to the model architecture, manual hyperparameter tuning was used to determine which changes to the architecture performed well, as explained in Section 3.1.3.

The training of the deep learning model was done using a NVIDIA Volta 100 GPU on a node having 32GB of RAM and 32 CPU cores. Some initial tests were performed on a cloud instance provided by Oracle, while the main model development described was performed on the JASMIN platform (Lawrence et al. (2012)). The software packages used were Python 3.6.8 and Tensorflow v2.20 (Abadi et al. (2015)). Training took 21 epochs to finish with a total time to train of 12 minutes. Despite the relatively short time taken to train the final model, it took a much longer time to progress through the various optimisations detailed in the appendix, mainly due to the use of 10-fold cross-validation.

3.1.3 Hyperparameter Tuning

The full deep learning model was developed on data from the Western Atlantic and Western Pacific regions. The training set, as described in Section 3.1.1, was used to perform 10-fold cross-validation. Each fold was then evaluated using the validation set. The evaluation produced a set of metrics which offered an

Layer (type)	**Layer (specification)**	**Output Shape**	**Number of parameters**
Input		22, 29, 5	
Conv2D	Glorot Uniform Weight Initialisation; ReLU Activation Function 8 Kernels; Size = 2x2; Strides - (1, 1)	21, 28, 8	168
Dropout	Dropout Rate - 0.1	21, 28, 8	
MaxPooling2D	Strides - (1, 1)	20, 27, 8	
Conv2D	Glorot Uniform Weight Initialisation; ReLU Activation Function 16 Kernels; Size = 2x2; Strides - (1, 1)	19, 26, 16	528
Dropout	Dropout Rate - 0.1	19, 26, 16	
MaxPooling2D	Strides - (1, 1)	18, 25, 16	
Conv2D	Glorot Uniform Weight Initialisation; ReLU Activation Function 32 Kernels; Size = 2x2; Strides - (1, 1)	17, 24, 32	2080
Dropout	Dropout Rate - 0.1	17, 24, 32	
MaxPooling2D	Strides - (1, 1)	16, 23, 32	
Conv2D	Glorot Uniform Weight Initialisation; ReLU Activation Function 64 Kernels; Size = 2x2; Strides - (1, 1)	15, 22, 64	8256
Dropout	Dropout Rate - 0.1	15, 22, 64	
MaxPooling2D	Strides - (1, 1)	14, 21, 64	
Conv2D	Glorot Uniform Weight Initialisation; ReLU Activation Function 128 Kernels; Size = 2x2; Strides - (1, 1)	13, 20, 128	32896
Dropout	Dropout Rate - 0.1	13, 20, 128	
MaxPooling2D	Strides - (1, 1)	12, 19, 28	
Flatten		29184	
Dense	Glorot Uniform Weight Initialisation; ReLU Activation Function 128 nodes; L2 Norm; Factor = 0.005	128	3735680
Dropout	Dropout Rate - 0.1	128	
Dense	Glorot Uniform Weight Initialisation; ReLU Activation Function 64 nodes; L2 Norm; Factor = 0.005	64	8256
Dropout	Dropout Rate - 0.1	64	
Dense	Glorot Uniform Weight Initialisation; ReLU Activation Function 32 nodes; L2 Norm; Factor = 0.005	32	2080
Dropout	Dropout Rate - 0.1	32	
Dense	Glorot Uniform Weight Initialisation; Sigmoid Activation Function 1 node; L2 Norm; Factor = 0.005	1	33

Table 3.2: The architecture of the Deep Learning model

Step	**Choice**	**K-fold CV Score Mean AUC-PR**
Choice of Data	Filtered Data 5 Fields	0.5309
Early Stopping	Patience = 10	0.6788
Normalisation	Standarisation	0.7404
Resolution	Sixteenth	0.7842
Dataset Balancing	Undersampling with Replacement	0.7839
Loss and Optimiser	Binary Cross-Entropy Momentum	0.7890

Learning Rate
Momentum
	LR = 0.01
	Momentum = 0.8
		0.7891

Data Augmentation	Roll in x direction Rotation by random angle Flip Left-Right	0.7988
Data Augmentation Rate	0.6	0.8018
Dropout Position Dropout Rate	Conv Base & Classifier Rate = 0.1	0.8104
L2 Norm Position L2 Norm Rate	Classifier Rate = 0.005	0.8128
Batch Size	8	0.8135

Table 3.3: K-Fold Cross Validation Results

insight into the effectiveness of any given configuration of deep learning model choices.

The metric used to assess whether a change improved the model performance was the Area-under-Curve for the Precision-Recall curve (AUC-PR). Given the major class imbalance in the dataset used and that as the model is intended to be used as a filtering technique, this metric is used due to it weighting both precision and recall equally. Given this model was being developed to identify data for further post-processing, false negatives are a bigger problem than false positives, and so improvements in recall were favoured over those in precision if AUC-PR varied only marginally as a change was implemented.

Development and optimisation using this value proceeded as described below, with the final models being described and evaluated using the testing dataset in Sections 3.1.2 and 3.2.1 respectively. Table 3.3 shows a summary of the steps taken during hyperparameter tuning.

The initial architecture that was used as the starting point for developing TCDetect consisted of an input of dimensions 84 rows by 110 columns by 2 channels which passed through five convolutional blocks, each made up of a convolutional layer of 8, 16, 32, 64 and 128 kernels respectively, with weights initialized using the Glorot Uniform method with ReLU activation functions, each with strides of 1 and a kernel size of 2x2; and a MaxPooling2D layer with strides equal to 1. The resulting kernels are flattened and passed through three fully-connected blocks, each made up of a dense layer of 128, 64 and 32 hidden nodes respectively, with weights initialized by the Glorot Uniform method. The model finishes off with another fully-connected layer of one node, this time using the sigmoid activation function with weights initialized by the Glorot Uniform method which was used to output a prediction. The optimizer used was the Stochastic Gradient Descent (SGD) optimizer with the default learning rate of 0.01 with the loss function being binary cross-entropy. Finally, a batch size of 32 cases was initially used.

Choice of Data

The first optimisation made was to choose the number and type of meteorological fields to supply to the model for it to make its predictions. Four possible configurations were tested:

- MSLP and 10-metre wind speed
- MSLP, 10-metre wind speed and vorticity at 850hPa, 700hPa and 600hPa
- MSLP and 10-metre wind speed with spherical harmonic filtering between wave numbers 5 and 106
- MSLP, 10-metre wind speed with spherical harmonic filtering between wave numbers 5 and 106 and vorticity at 850hPa, 700hPa and 600hPa with spherical harmonic filtering between wave numbers 1 and 63

The last option provided the best mean AUC-PR, that of 0.5309.

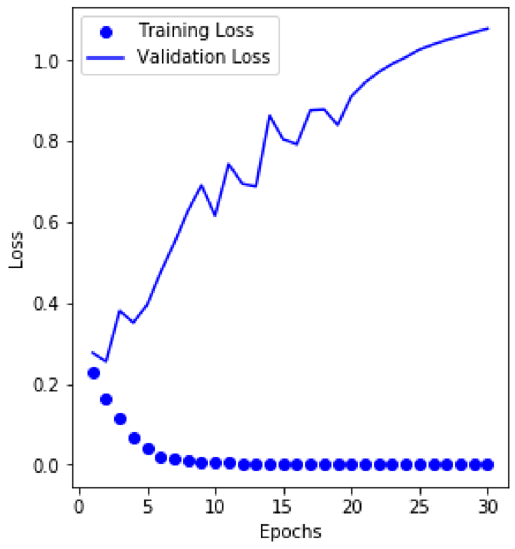
Early Stopping

Next, it was noted that the model was overfitting as Figure 3.4a shows that except for the first two epochs, the training loss gets smaller while the validation loss gets larger with am increasing number of epochs. Figure 3.4b shows similar behaviour with AUC-PR.

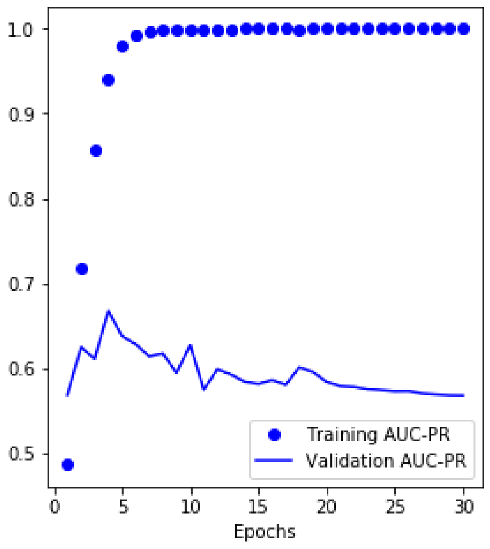
To overcome this issue, model training was stopped earlier by stopping training when the training and validation AUC-PR start to diverge. A number of epochs of patience, i.e. the number of epochs to wait until stopping to make sure that training was not stopped too early, were trialled to get the best possible performance. Patience values trailled were of 2, 5, 10 and 20 epochs. That of 10 epochs obtained the best mean AUC-PR of 0.6788.

Normalisation

A few methods for normalisation were trialled, namely of normalising values to lie in the range of 0 to 1 or -1 to 1, standardising value to have a mean of 0 and a standard deviation of 1 and a combination of normalisation and standardisation. The method of standardisation produced the best model performance with a mean AUC-PR of 0.7404.



(a) Loss for model trained and tested on data from the Western Atlantic and Western Pacific regions before applying Early Stopping.



(b) AUC-PR model trained and tested on data from the Western Atlantic and Western Pacific regions before applying Early Stopping.

Resolution

Resolution of the data used was next checked. The resolution used up to the current stage was that of the original ERA-Interim dataset, but resolutions of 1.4°x1.4°, 2.1°x2.1°, 2.8°x2.8° and 3.5°x3.5° were tested. The resolution of 2.8°x2.8°, which was obtained by taking every fourth pixel of the image in both the x and y directions, produced the best mean AUC-PR of 0.7842.

Dataset Balancing

One problem that was known when starting hyper-parameter optimisation was that the dataset was heavily dominated by negatively labelled cases. In fact, the training dataset having data from the WAWP regions had 89.46% of the cases negatively labelled, while that having data from all regions had 95% of cases negatively labelled. This split of data would inhibit the model learning the right pattern to maximise its performance. Therefore, six ways of balancing the dataset were investigated.

- Naive Oversampling - Making copies of the positively labelled cases until the dataset is balanced.

- Undersampling without Replacement - Undersample the negatively labelled cases prior to training. Therefore, some data is not used.

- Undersampling with Replacement - Undersample the negatively labelled cases during training, so they change from epoch to epoch. Possible overfitting on positively labelled cases.

- Weighting the Cases - Weighting the cases so that the negatively labelled cases have less influence on the learning process.

- Adding Bias - Add a bias to the output layer to prevent the model from learning the bias.

- Weighting the Cases and Adding Bias - A combination of the previous two options.

The option that produced the best performance of a mean AUC-PR of 0.7839 was that of undersampling with replacement. It can be noted that the model's performance decreased marginally from the previous step, but this was still selected as recall became much favoured by the model, which is important for the use case in mind.

Loss and Optimiser

The model so far used the binary cross-entropy loss function with the Stochastic Gradient Descent (SGD) optimizer. All possible combinations of the mean absolute error, mean standard error and binary cross-entropy loss functions and SGD, RMSprop, SGD with Momentum using a momentum parameter of 0.9, Adam, Adagrad, Adamax and Nadam optimizers were examined.

The combination which obtained the best mean AUC-PR of 0.7890 was binary cross-entropy loss with the SGD optimiser with Momentum using a momentum parameter of 0.9.

Learning Rate and Momentum

A grid search for the best learning rate and momentum parameters was performed. The values for the learning rate included were those of 0.0001, 0.0005, 0.001, 0.005, 0.01 and 0.05 while those used for the momentum parameter were in the range of 0.1 to 1 with a step of 0.1. The combination which produced the best performing model was that having a learning rate of 0.01 and a momentum of 0.8

Data Augmentation Methods

Several techniques including random rolls, rotations, adding random noise, flipping the input data along either the x or y directions and random cropping were evaluated. The augmentation rate was set at 50%. The options which obtained a comparative or better mean AUC-PR were rolling the picture along the x-direction, flipping the picture left to right and rotating the image by a random amount. These were all included in the model and the combined methods produced a mean AUC-PR of 0.7988.

Data Augmentation Rate

The best data augmentation rate was also varied from 0.1 to 1 in steps of 0.1 to find the best possible rate. The best performing model with a mean AUC-PR of 0.8018 was that with an augmentation rate of 60%.

		Predicted	
		Yes	**No**
Labelled	**Yes**	1231	111
	No	2166	20844

Table 3.4: Confusion matrix for the architecture when trained and tested on data from all regions.

Dropout Position and Rate

Dropout was investigated next. It was trialled in three places, namely the convolutional base only, the fully-connected classifier only and throughout the model with dropout rates varying from 10% to 100% in steps of 10%. The model with the best AUC-PR, that of 0.8104, was that employing dropout with a rate of 10% throughout the model.

L2 Normalisation Position and Factor

L2 normalisation was also investigated. As with the previous optimisation, it was trailled in the same three places. The normalisation factors checked were 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5. The model that produced the best performance with a mean AUC-PR of 0.8128.

Batch Size

This final optimisation tested was of varying the batch size. Batch sizes of 8, 16, 64, 128, 256, 512, 1024 and 2048 were tested with the first option producing the best performing model with a mean AUC-PR of 0.8135.

Others

Other optimisations tested which did not produce a model with an improved performance included batch normalisation, varying the number of hidden layers and nodes and using different weight initialisation methods and activation functions.

3.2 Model Understanding and Justifications

TCDetect was evaluated against the test set described earlier. It was also investigated to understand how it generates its predictions, with the aim of demystifying the deep learning model. We present these results in this section.

3.2.1 Model Statistics

The model, after training, correctly classified 1231 (91.73%) of the 1342 cases having a TC and 20844 (90.59%) of the 23010 cases not having a TC. It misclassified 111 (8.27%) cases in which a TC was present and 2166 (9.41%) cases in which a TC was not present. These are summarised in the confusion matrix in Table 3.4.

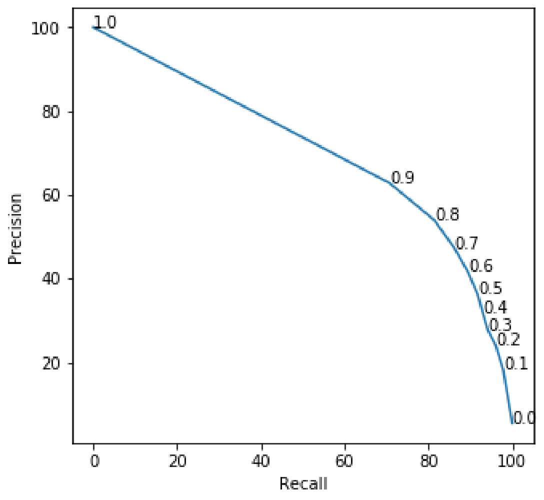


Figure 3.5: Precision-Recall curve for final architecture when trained and tested on data from all regions.

The above values lead to an accuracy of 90.65%, a recall rate of 91.73% and a precision rate of 36.24%. The use of AUC-PR as a metric to maximise when performing hyperparameter tuning resulted in a high recall rate and a sufficiently high precision rate for this model to be used as a data filtration technique.

These values could be further varied by changing the value which is the boundary between a positive and a negative prediction, currently 0.5. Figure 3.5 shows the AUC-PR curve for the model with the values at each point signifying the boundary at which the corresponding recall and precision rates are obtained.

3.2.2 Comparison with Standard Models

There are many existing deep learning architectures, so it is reasonable to ask "Would any of those do better than the architecture developed here?".

To test this, convolutional bases from a rich variety of standard model architectures were compared. These include: DenseNet121 (Huang et al. (2016)), DenseNet169 (Huang et al. (2016)), DenseNet201 (Huang et al. (2016)), InceptionResNetV2 (Szegedy et al. (2017)), InceptionV3 (Szegedy et al. (2016)), MobileNet (Howard et al. (2017)), MobileNetV2 (Sandler et al. (2018)), ResNet101 (He et al. (2016)), ResNet101V2 (He et al. (2016)), ResNet152 (He et al. (2016)), ResNet152V2 (He et al. (2016)), ResNet50 (He et al. (2016)), ResNet50V2 (He et al. (2016)), VGG16 (Simonyan and Zisserman (2014)), VGG19 (Simonyan and Zisserman (2014)) and Xception (Chollet (2017)), with weights obtained when training from the ImageNet dataset (Deng et al. (2009)), were added to the fully-connected classifier developed in this paper. The weights in the classifier were than trained on data from all regions with the presented model's learning rate, momentum value and L2 normalisation factor, while the convolutional base was kept frozen.

Given that these convolutional bases required inputs of at least 75 pixels by 75 pixels with 3 channels, some changes to the inputs were required. Firstly, as an input with only 3 channels is required for the most of the above architec-

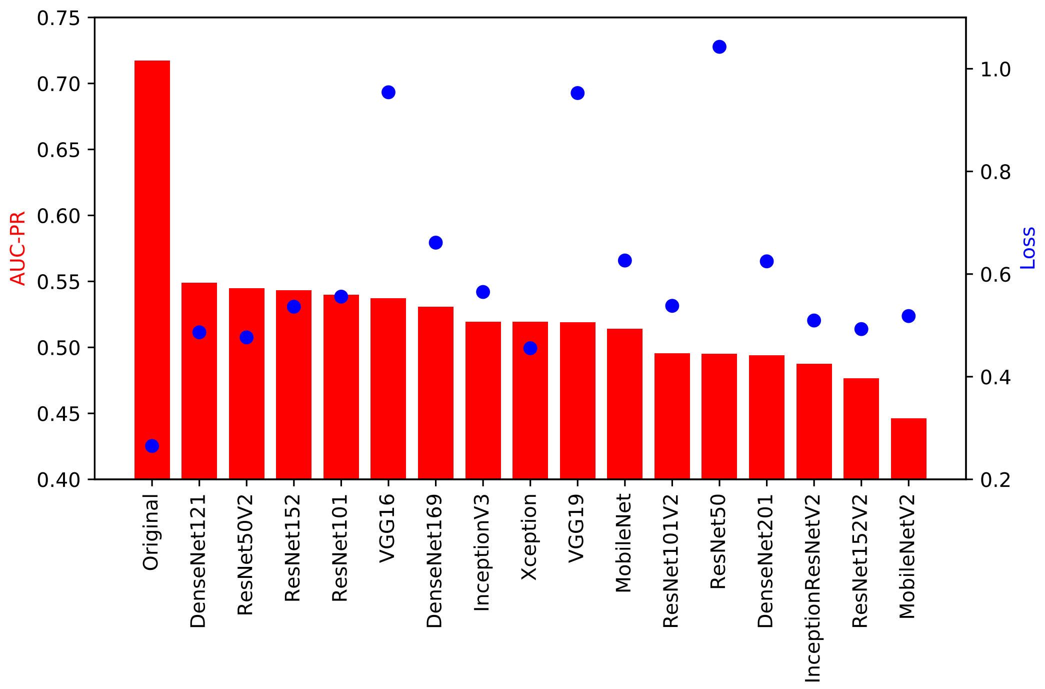


Figure 3.6: Test AUC-PR (bars) and test loss (points) for standard convolutional bases, pre-trained on the ImageNet database, attached to the fully-connected classifier developed in the presented model. The classifier was re-trained for each convolutional base with data from all regions.

tures, the fields retained were those of vorticity at 850hPa, vorticity at 600hPa and MSLP. This choice was made as these three fields were deemed the most important for the model being presented by tests detailed in Section 3.2.3. Secondly, the input size was extended five fold from 22x29 pixels to 110x145 pixels by interpolating any intermediate values.

Of the standard architectures tested (see figure 3.6 for all results), none managed to obtain better AUC-PR or loss values on the test set. Table 3.5 compares the complexity of some of these more standard models and their performance metrics to the model being presented here. All of the more standard models had far higher complexity in terms of the number of parameters used than the model being presented here, and the latter also outperforms the others in terms of AUC-PR, precision rate and loss. While recall for the model being presented here is not outperforming all of the other models, it is in the top third of the list.

3.2.3 Model Explainability

This section will go through some techniques used to make the model more explainable, with the aim of having more trust in its inferences. Some techniques used are feature importance, to determine which inputs influence the inferences most; checking whether the size of the dataset used affects the model's performance; and checking how the model performs over different regions of the globe.

Convolutional Base	Total Parameters	AUC-PR	Recall	Precision	Loss
Our Model	**3,789,977**	**0.7173**	92%	**36%**	**0.2650**
DenseNet121	8,620,865	0.5409	83%	25%	0.4865
DenseNet169	15,209,281	0.5307	93%	13%	0.6612
DenseNet201	21,821,601	0.4940	88%	20%	0.6248
InceptionResNet v2	55,526,881	0.4874	83%	20%	0.5095
Inception v3	23,386,145	0.5184	90%	18%	0.5651
MobileNet	4,812,225	0.5139	88%	17%	0.6264
MobileNet v2	5,545,281	0.4461	86%	18%	0.5182
ResNet101	47,911,553	0.5397	89%	17%	0.5560
ResNet101 v2	47,879,937	0.4955	88%	21%	0.5381
ResNet152	63,624,321	0.5430	84%	23%	0.5364
ResNet152 v2	63,585,025	0.4765	83%	27%	0.4928
ResNet50	28,841,089	0.4949	95%	11%	1.0428
ResNet50 v2	28,818,177	0.5447	89%	19%	0.4766
VGG16	15,511,617	0.5369	**97%**	11%	0.9542
VGG19	20,821,313	0.5187	95%	11%	0.9527

Table 3.5: Comparison of total parameters used and performance metrics for model being presented in this paper and similar models using more standard convolutional bases.

Feature Importance

One important aspect when building the model was to quantify each field's importance to the model's prediction. Two methods are employed for this: the Breimann method (Breiman (2001)) and the Lakshmanan method (Lakshmanan et al. (2015)).

The former involves permuting the data from one field across all the test cases and then re-testing the model with this modified dataset. A decrease in the model's performance is expected, with the most important field obtaining the largest decrease in performance.

The latter involves several steps: First, to permute the data as in the previous method for one field. Once the field with the most importance, i.e., the field which produces the largest decline in performance, is found, it is kept permuted, while the others fields are permuted individually. The next most important field is now found repeating the algorithm on the remaining fields. This process keeps on going until all the fields are permuted. The above methods were performed 30 times each and an average was taken to make sure of consistent and robust results.

Figure 3.7 shows the results of the Breiman and Lakshmanan methods for the trained model. The most important field was found to be that of vorticity at 850 hPa with the Breiman method showing a decrease in AUC-PR from 0.7173 to 0.0936. Then, the Breiman method shows that rest of the ranking for the most important field is as follows: MSLP, vorticity at 600hPa, vorticity at 700hPa and 10-metre wind speed. The Lakshmanan method shows a slightly different ranking, with MSLP demoted from being the second-most important field to the fourth most-important field. As before, not much should be read into this slight change as the difference in AUC-PR values is minimal.

Therefore, it can be concluded that the most important field is that of vorticity at 850hPa and the second-most is that of vorticity at 600hPa. Hence,

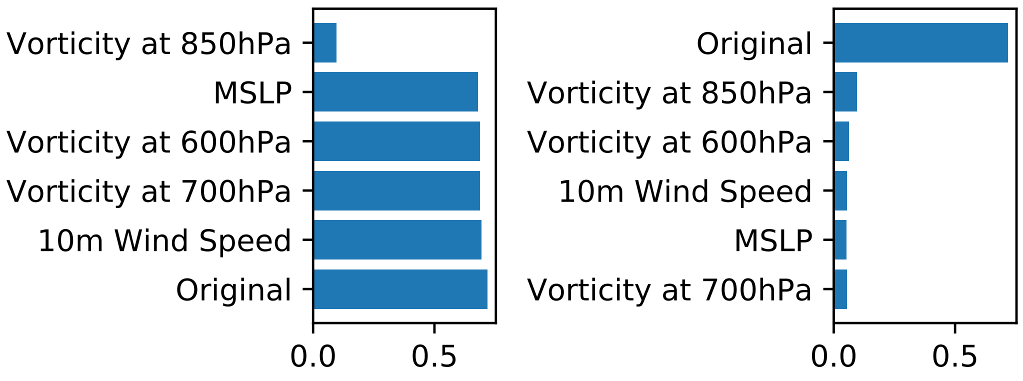


Figure 3.7: Feature Importance using the Breiman (left) and Lakshamanan (right) methods for model trained and tested on data from all regions.

it is possible that the model is using matching areas of high vorticity at 850hPa and 600hPa. Thus, the model apparently is checking for deep convection.

The importance of locality

Having seen which fields influence the model's inferences most, we checked how the developed model performs across different regions of the globe.

As the deep learning model being presented was developed, a number of optimisations were carried out (Appendix B). Due to time and computational constraints, the manual hyperparameter tuning process was performed on data from the Western Atlantic and Western Pacific (WAWP) regions. When doing this, two assumptions were made: that any change made to the architecture which caused an improvement in the model's performance resulted a similar improvement when the architecture was trained and tested on data from all regions; and that a model trained on data from the WAWP regions would generalise well when tested on data from all regions of the world.

The first and third columns of Table 3.6 show that the first assumption holds, although it can be seen that the magnitude of the improvements between the two models can vary. Also, as shown in Table 3.7, the architecture has similar performance when trained and tested only on data from the WAWP regions and when trained and tested on data from all regions.

However, the second assumption was found to not hold. The first and second columns of Table 3.6 show that a model trained on data from the WAWP regions decreased in performance considerably when tested on data from all regions. This is mirrored when using the final models, as shown in Table 3.7.

A reason for this is that the data from the WAWP regions differs from that of the whole world. Figure 3.8 shows the mean case for data originating only from the WAWP regions on the top row and the mean case for data originating from all regions on the bottom row, with each column corresponding to each vairable used, i.e. MSLP, 10-metre wind speed, vorticity at 850hPa, vorticity at 700hPa and vorticity at 600hPa. As can be seen, the two types of data differ, hence the model trained only on WAWP data might be trying to find a different pattern than that trained on data from all regions.

Step	Model trained and tested on WAWP	Model trained on WAWP tested on Whole World	Whole World Model
Choice of Data	0.5830	0.0660	0.4928
Early Stopping	0.7111	0.0815	0.5915
Normalisation	0.7469	0.1772	0.6790
Resolution	0.7908	0.3690	0.6794
Dataset Balancing	0.7721	0.2905	0.6856
Loss and Optimiser	0.7849	0.3946	0.6733
Learning Rate Momentum	0.7980	0.6149	0.6646
Data Augmentation	0.8038	0.4457	0.6901
Data Augmentation Rate	0.8035	0.6377	0.6759
Dropout Position Dropout Rate	0.8091	0.6076	0.6832
L2 Norm Position L2 Norm Rate	0.8128	0.5331	0.6955
Batch Size	0.8176	0.6315	0.6756

Table 3.6: Results when using validation data

Model	**Training Region**	**Evaluation Region**	**AUC-PR**
WAWP	WAWP	WAWP	0.7884
WAWP	WAWP	Global	0.6491
Global	Global	Global	0.7173

Table 3.7: Changes in AUC-PR with different training and tesing regions.

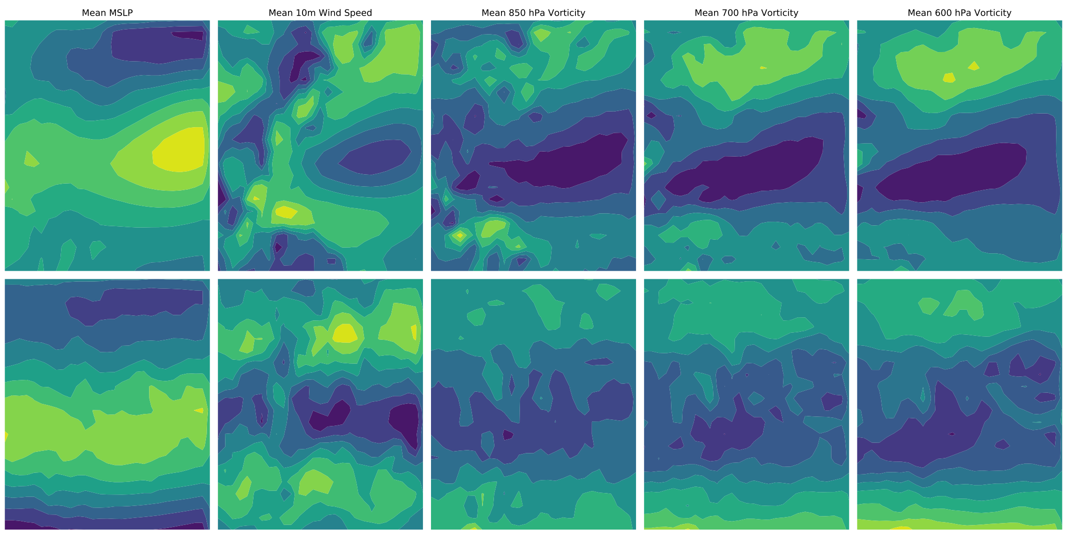


Figure 3.8: Mean Case for data originating only from the Western Atlantic and Western Pacific regions (top row) and for data originating from all regions (bottom row). Columns show MSLP (1st column), 10-metre wind speed (2nd column), vorticity at 850hPa (3rd column), vorticity at 700hPa (4th column) and vorticity at 600hPa (5th column).

Latitude	**60°S - 0°N**	**60°S - 0°N**	**60°S - 0°N**	**60°S - 0°N**	**0°N - 60°S**	**0°N - 60°S**	**0°N - 60°S**	**0°N - 60°S**
Longitude	**20°E - 100°E**	**100°E - 180°E**	**180°E - 260°E**	**260°E - 340°E**	**20°E - 100°E**	**100°E - 180°E**	**180°E - 260°E**	**260°E - 340°E**
Number of positive cases	72	400	267	250	115	113	26	0
Recall obtained by model trained on WAWP data	56.94%	90.75%	80.15%	90.80%	53.74%	58.41%	30.77%	N/A
Number of Positively Labelled cases correctly classified by model trained on WAWP data	41	363	214	227	115	66	8	N/A
Recall obtained by model trained on whole world data	88.89%	93.00%	99.25%	96.80%	80.37%	92.92%	42.31%	N/A
Number of Positively Labelled cases correctly classified by model trained on whole world data	64	372	265	242	172	105	11	N/A

Table 3.8: Evolution of accuracy during model development by basin (see text for explanation of rows).

Category	**1**	**2**	**3**	**4**	**5**
Recall	88%	92%	94%	95%	100%

Table 3.9: Recall based on TC category when using validation data

To further understand how the model trained on WAWP data differs from that trained on data from all regions, the results have been split by basin as shown in Table 3.8 to quantify any differences.

As expected, the model trained on WAWP data performs best on the Western Atlantic and Western Pacific regions, with a recall of 90.80% and 90.75% respectively. It also performs well in the Eastern Pacific region with a recall of 80.15%. However, all other regions do not surpass the recall rate of 60%.

On the other hand, when the model trained on data from all regions is used, all recall rates improve, some significantly. The Western Atlantic and Western Pacific regions improve their recall rates by 2.25% and 6% to 93% and 96.80% respectively. The most improved region is the one bounded by 100°E-180°E in the Southern Hemisphere, with its recall rate increasing by more than half from 58.41% to 92.92%. All but one region obtained a recall rate of at least 80%, with many surpassing a recall rate of 90%. The region that did not do well was that bounded by 180°E-260°E in the Southern Hemisphere, which obtained a recall rate of 42.31%. A possible reason for this to not perform as well as the other regions is that a smaller number of cases with TCs are available for this region, with only 26 in the test set.

Performance by Strength of Tropical Cyclone

A manual exploration of instances incorrectly classified by the deep learning model being presented here indicated that stronger tropical cyclones are picked up better. To prove this quantitatively, we analyze the results for the different cyclone categories. A good measure for this ability is the recall rate, as it shows the proportion of positively labelled cases, split by tropical cyclone category, being correctly classified. Table 3.9 shows very high recall for all categories, as defined in the Saffir-Simpsom scale and noted in the IBTrACS database, of tropical cyclones. The model has a recall rate of 88% for tropical cyclones of Category 1 (the weakest classified TCs) and a perfect recall rate for Category 5 (the strongest) tropical cyclones. As expected, an increasing trend of recall against category can be seen as higher category cyclones are easier to detect.

Furthermore, the TC strength of the cases predicted as having a TC present was investigated. As discussed earlier, the use case of this model is intended to be that of a data filtration method for TCs in meteorological data. Therefore, the standard definition of a TC, that of having sustained winds of 119 km/h, was used to differentiate between a non-TC depression and a TC.

This choice of boundary can still affect its use case. While the deep learning model will be looking for TCs, it is very difficult even for observers and scientists to discern between a TC and a storm slightly weaker than a TC, such as a storm still strengthening into a TC, based on the inputs given to the model. This therefore can put some ambiguity in the use of recall for the model and how it is used. Realistically, if some of the not-quite-TCs are predicted as TCs, it is not a bad thing, as the point of a data filtration method is to reduce the amount of data which is not of any interest - any data without any meteorological system in this case. Hence, the boundary of un/needed data shifts between the deep learning point of view and the application point of view.

It was found that only 506 out of the 3397 cases that obtained a positive inference from the model were cases that had no meteorological system present. The following shows the breakdown of the cases as labelled by IBTrACS:

- No meteorological system: 506
- Unknown: 2
- Post-tropical systems: 18
- Disturbances: 165
- Subtropical systems: 32
- Tropical Depressions: 348
- Tropical Storms: 1095
- Category 1 TCs: 426
- Category 2 TCs: 281
- Category 3 TCs: 243
- Category 4 TCs: 212
- Category 5 TCs: 69

This suggests that the deep learning model is picking up the required pattern needed, due to the high recall values for TCs of strength at least Category 1, but is mislabelling weaker features as TCs. As discussed earlier, while affecting the model's performance from a deep learning point of view, the inclusion of such storms is not a problem from an application point of view.

Another possible reason for this mislabelling of weaker TCs is that the deep learning model developed used data at a sixteenth of ERA-Interim's original resolution. This means that the model was using data with a resolution of around 2.8 degrees, or around 280km. This was chosen during the manual

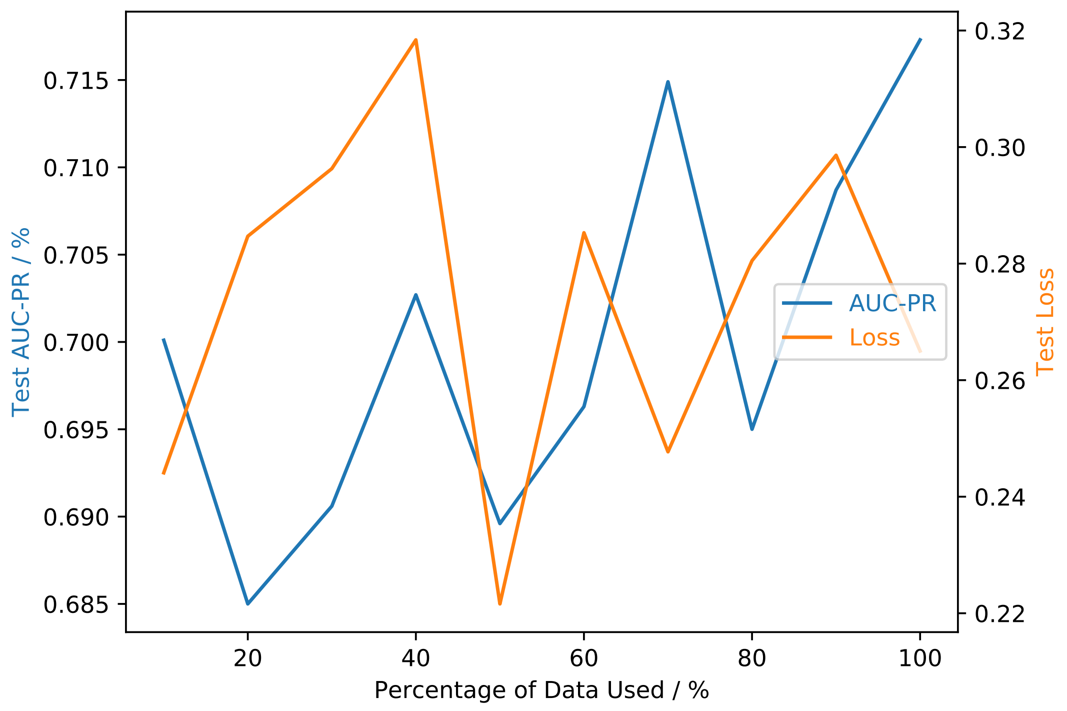


Figure 3.9: Test AUC-PR and Loss for model trained and tested on data from regions around the world.

hyperparameter search as detailed in the Appendix due to the model's metrics improving most at this resolution. The hypothesis as to why this occurred was that the coarsening of the data filtered out some of the noise present in the data while still preserving the structure of any system present. That being said, this might have had a larger impact on distorting the structure of lower-strength storms, which is possibly why the model struggles more with such storms.

Also, ERA-Interim was used for obtaining the training data as it was the most accessible and recent reanalysis data at the time that the study was started. We are aware of the more recent ERA5 dataset, which might produce better results due to the better data assimilation techniques used to produce the dataset, which might lead to a system's structure to be better defined.

Size of Dataset

The size of the dataset used was checked to see whether the dataset has been exploited to its fullest.

This was done by training the final architecture using varying amounts of data from the whole dataset. Data amounts used for training varied in steps of 10% from 10% to 100% of the training set.

Figure 3.9 shows the result when using data from regions across the world. The testing AUC-PR is increasing while the testing loss is mostly flat as more data is added. This shows that more data would probably improve the model's performance.

3.3 Chapter Summary

A Deep Learning method to identify the presence or absence of Tropical Cyclones in simulation data is presented. Trained on ERA-Interim data, the deep learning model obtained an AUC-PR of 0.7173 with an accuracy of 91% on a test set, which was made up of 24352 cases.

The model did not generalize well when training only on cases from the Western Pacific and Western Atlantic basins and testing on cases from the whole domain, however TCdetect, trained on data from all regions did generalise well.

As well as presenting the specific optimisations made to obtain the model (including dropout, early stopping, dataset balancing and different inputs), a selection of standard deep learning models are described and shown not to be able to outperform the model presented in this paper, while being more complex.

The possibility of obtaining a better model had more data been available was investigated, with some indications that more data would have helped.

While the training data was obtained from ERA-Interim, the ground truth used was IBTrACS, which introduces an element of uncertainty in interpreting the results - is an incorrect label (presence/absence) a consequence of the presence or absence of the TC in the ERA-Interim data? It is known that reanalysis data cannot resolve the full strength of storms, and so will likely undercount TCs, and hence depress the possible accuracy rates.

The impact of such issues on detectability is consistent with the result that this model is better at detecting stronger TCS then weaker TCs (weaker TCs will be more poorly represented in reanalysis data).

Future work includes attempting to improve TCDetect to better handle TCs of a low category potentially via ideas imported from other standard techniques or using different meteorological fields, as well as implementing an inference step using a version of the model in a full General Circulation Model to evaluate the pros and cons of avoiding data output.

Chapter 4

Evaluating the Deep Learning model

Tropical cyclones are large meteorological events that can leave devastating effects and are identified in simulations by the application of detection algorithms. Where the simulations are re-analysis, detection algorithms can be compared with observations recorded using the International Best Track Archive for Climate Stewardship (IBTrACS). In this work, a novel new detection algorithm based on deep learning is compared with a state of the art tracking system and observations to provide context for use in analysing climate simulations. The comparison utilises ERA-Interim data, and focuses on whether or not the tropical cyclone events are detected and whether the structure of the TCs detected or observed play a part in their relative performance. To perform the latter, the detected TCs location in space was also investigated. a A key part of the comparison is the recognition that ERA-Interim itself does not fully reflect the observations, and so no detection algorithm operating on ERA-Interim will fully recover the IBTrACS observations. However, for strong well-defined cyclone events, the two detection algorithms operating on reanalysis and the observations agree well, with comparable performance across all areas of the globe. Where events are detected by only one algorithm (or only in observations) they are the weakest events with around half the maximum vorticity seen in events detected by both algorithms and the observations. Furthermore, the events detected by both algorithms and the observations have the least amount of noise in their fields and have a clear centre of circulation.

Tropical Cyclones (TCs) are extreme weather events that can have a large effect on any environment. Such TCs can be and are detected and tracked in satellite data, numerical weather prediction (NWP) simulations, and longer simulations with global circulation models (GCMs) via automatic means.

Previous studies (see Chapter 2) have shown that the performance of various detection algorithms is comparable when addressing strong TCs, i.e. those that have obtained hurricane status according to the Saffir-Simpson scale. Some

This chapter is being prepared for publication; contributions - Daniel Galea: coding, data collection, writing, editing; Bryan Lawrence: writing, editing, direction show that the detection algorithms did not perform well when used on datasets other than that on which the algorithm was first devised.

In the previous chapter, we introduced a deep learning technique for detecting the presence or absence of a TC in a field of simulation data. Here we compare that technique with a version of TRACK (Hodges (1995), Hodges (1996), Hodges (1999)) applied to re-analysis data and compared with reality as recorded by the International Best Track Archive for Climate Stewardship (IBTrACS, Knapp et al. (2010), Knapp et al. (2018)) archive to understand some the limitations of the application of our technique for feature identification in simulation data.

4.1 Data and Methods

The goal of this chapter is to understand the characteristics and applicability of our deep learning cyclone detection method, TCDetect, when applied to simulations of the real world. Doing this requires going beyond the normal deep learning metrics, as there are additional complications for real world applications: both the observations (the ground truth) and the simulation data used as input to the deep-learning introduce detection biases.

In the real world IBTrACS provides the best source of ground truth. Initially developed by the National Oceanic and Atmospheric Administration (NOAA), it combines all the best-track data for TCs from all the official Tropical Cyclone Warning Centers, the WMO Regional Specialized Meteorological Centers (RSMCs), and other sources.

TRACK is a state-of-the-art automatic detection and tracking system for different types of atmospheric disturbances with considerable use since inception in Hodges (1995), Hodges (1996) and Hodges (1999). Here the TC tracking component is used as a gold-standard comparator against which to compare the results from the deep learning model.

We use TCDetect and TRACK applied to re-analysis data and compare them to each other and the IBTrACS dataset. Re-analysis data provides the best possible synthesized observations of meteorological variables; we choose to use the ERA-Interim product (Dee et al. (2011)). ERA-Interim utilises version CY31r2 of the European Centre for Medium-Range Weather Forecasts (ECMWF) numerical weather prediction system, the Integrated Forecasting System (IFS), together with assimilation of observations from 1979 through to 2019. The comparison is limited to the 25 months between the 1st of August 2017 until the end of August 2019 as earlier data is used in training the deep learning algorithm.

ERA-Interim data is produced at a spatial resolution of 79km, a temporal resolution of 6 hours and has 60 vertical levels up to 0.1hPa. Of the many parameters produced, only the mean-sea level pressure (MSLP), 10-metre wind speed and relative vorticity at 850hPa, 700hPa and 600hPa are used in this study.

4.1.1 IBTrACS

The IBTrACS dataset has information about reported storms, such as the storm centre in latitude and longitude, maximum surface wind speed, minimum sea level pressure and category.

While IBTrACS is the best available observational dataset, some inhomogeneity exists between each source as the contributing centres have differing observing systems and parametric approaches. Such observing systems can be limited in time and space, leading to the omission of systems not detected or an incomplete record of their evolution, particularly if they had limited or no human impact, or they were out of range of detection systems such as airborne missions.

4.1.2 The TCDetect Deep Learning Model

The TCDetect deep learning TC detection scheme was described in the previous chapter. It uses a deep learning scheme trained on ERA-Interim data which utilises mean sea-level pressure (MSLP), 10-metre wind speed, and vorticity at 850hPa, 700hPa and 600hPa; all coarsened to a sixteenth of ERA-Interim's native spatial resolution, resulting in an input resolution of approximately 320km.

These data were passed through a convolutional base connected to a fully-connected dense classifier trained to detect TCs labelled using IBTrACS. The system outputted a classifier value ranging between 0 and 1; a tropical cyclone is inferred to be present if the value is greater than 0.5, and absent if less than or equal to 0.5.

For the identification of tropical cyclones and using 0.5 as the boundary, when trained on ERA-Interim, the TCDetect algorithm obtained a recall rate of 92% with a precision rate of 36%. In practice, this means that the while most of the actual TCs were detected, many of the TCs identified were technically false negatives (i.e. not storms of strength 1 or greater on the Saffir-Simpson scale). However, as discussed in the previous chapter and further discussed below, most of these were actually meteorologically significant.

The recall rate and precision were calculated in terms of the application of the technique to ERA-Interim data, but the labels came from IBTrACS. It is reasonable then to ask "to what extent does the ability of ERA-Interim to reproduce the original storm strength and timing impact on these results"? We address this question by applying both T-TRACK and TCDetect to ERA-Interim, and comparing the results with the IBTrACS "ground truth-labels".

The TC centre is not given by TCDetect, so a way to extract it was needed. For this, the Gradient Class Activation Map technique (Grad-CAM, Selvaraju et al. (2017)) was used: for a given input, the output of the deep learning model is passed back through the model and together with gradient maximisation, produces a heatmap of the input areas used in a selected layer en route to the output. For TC location, we selected the first convolutional layer, and assumed the the TC central position in latitude and longitude is co-located with the maximum activation.

Because the heatmaps used for Grad-CAM were generated from the coarsened (320km resolution) data, the resulting TC centres were coarsely quantized and only poor quality comparisons were possible. To mitigate this effect, the

Grad-CAM centres (”interim centres”) were then passed through an additional refinement step to generate more accurate locations. A box with sides of 10 degrees in latitude and longitude was centred on the interim centres, and the original full resolution ERA-interim vorticity values at 850hPa, 700hPa and 600hPa were obtained and vertically averaged. The TC centre was assumed to be located at the position of the maximum in the absolute value of the averaged vorticity.

These TC centres were then used to make up TC tracks. Given that only one TC centre could be produced per region at any one timestep, a track was first defined as having TC centres which were present in consecutive timesteps in the same region. However, this produced many short (< 2 days) tracks. To try and fix this, tracks for a single region which had at most 2 days (8 timesteps) of no TC being detected and a separation distance of 20 degrees (geodesic) between the final TC centre from one track and the initial TC centre of the next track were joined to make up one track. This process was carried out until no more tracks could be joined. The separation distance criterion might intuitively seem to be too wide, but as will be shown below, TCDetect had some trouble with locating TC centres, so some buffer was built into this criterion.

4.1.3 TRACK

TRACK has four different stages: data preparation; segmentation; feature point detection and tracking.

In the first step, TRACK treats the data so that features of interest are easier to detect. This is done with the help of spectral filtering to only keep features which have spatial scales in the range of the features of interest. With regards to tropical cyclones, the features present in wavenumbers 5 to 63 are kept in the vertical average of vorticity between the heights of 850hPa and 600hPa.

During the segmentation stage, each point in each timestep of any data used is classified as a background or an object point, depending on whether the value for the vertical average of vorticity at 850hPa, 700hPa and 600hPa is above or below the threshold of 5×10^{-6} s^{-1}. The object points are then collected into objects.

Feature point detection then allocates a feature point to each object, representing its centre. This feature point could be selected as the centroid of the object, a local extrema or using some other technique, depending on the type of data used.

Finally, the tracking stage uses the feature points generated to minimise a constrained cost function to get the smoothest possible tracks.

The complete TRACK algorithm finds a range of cyclones, some of which may be TCs. The tracks produced can be processed to identify only TC tracks. Bengtsson, Hodges, and Esch (2007) summarise the necessary processing criteria:

- a lifetime of at least 2 days
- the initial point in the track must be in between the latitudes of 20°S and 20°N if over land or 30°S and 30°N if over an ocean

- a maximum in T63 vertically-averaged relative vorticity intensity at 850hPa over 5×10^{-6} s^{-1}
- a warm core check: a T63 vorticity maxima for each atmosphere level up to 250hPa and that the difference between the maxima at 850hPa and 250hPa is above a 5×10^{-6} s^{-1}
- the last two conditions holding for the last n timesteps, where n is a user-defined value

We refer to the set of tracks which conform to these criteria as the "truncated-TRACK" dataset or T-TRACK.

4.2 Results

The first question to consider is "To what extent do the two detection algorithms recover the TC events seen in the observations?". We can then ask "How well do the two algorithms (combined with ERA-Interim data) position the TCs in space?". Finally, we ask "To what extent does the detection success depend on the TC structure?"

4.2.1 Detection

Figure 4.1 shows the relationship between detections and observations for all the events during the period of interest. For these purposes, an event was counted when a TC (Cat-1 or greater on the Saffir-Simpson scale) was observed and/or detected in any timestep. TCs in different regions in the same timestep would give an event for each region in which a TC is seen. However, if multiple TCs are in the same region in the same timestep, this is considered to be one event.

In total there were 1342 such events in the IBTrACS data, and 4741 and 3397 detected by T-TRACK and TCDetect respectively (Figure 4.1a). The majority of the observed events were found by both detection algorithms, with TCDetect finding slightly more than T-TRACK. Relatively few (50) IBTrACS events were not found by one or other detection method, consistent with the expected high recall rates. However, more events were detected by one or both of T-TRACK and TCDetect than were present in the observations, which suggests many non-TC meteorological events were being incorrectly classified as TCs. This finding is discussed further below.

With an a priori expectation that IBTrACS may be undersampling TC events in the Southern hemisphere, the data was also split into hemispheres to investigate (Figure 4.1b/c). In terms of recall, that is the ability for IB-TrACS TCs to be detected in ERA-Interim, it can be seen (Table 4.1) that TCDetect is doing slightly better than T-TRACK in both hemispheres, and slightly more so in the North.

It is worth noting that the criteria used to supposedly screen TRACK to identify TCs are responsible for some of the "missing" detections. If TRACK alone is used, then the recall rate is much higher, reaching 96% globally, with 97/92% in the northern/southern hemispheres respectively, albeit with many more false positives.

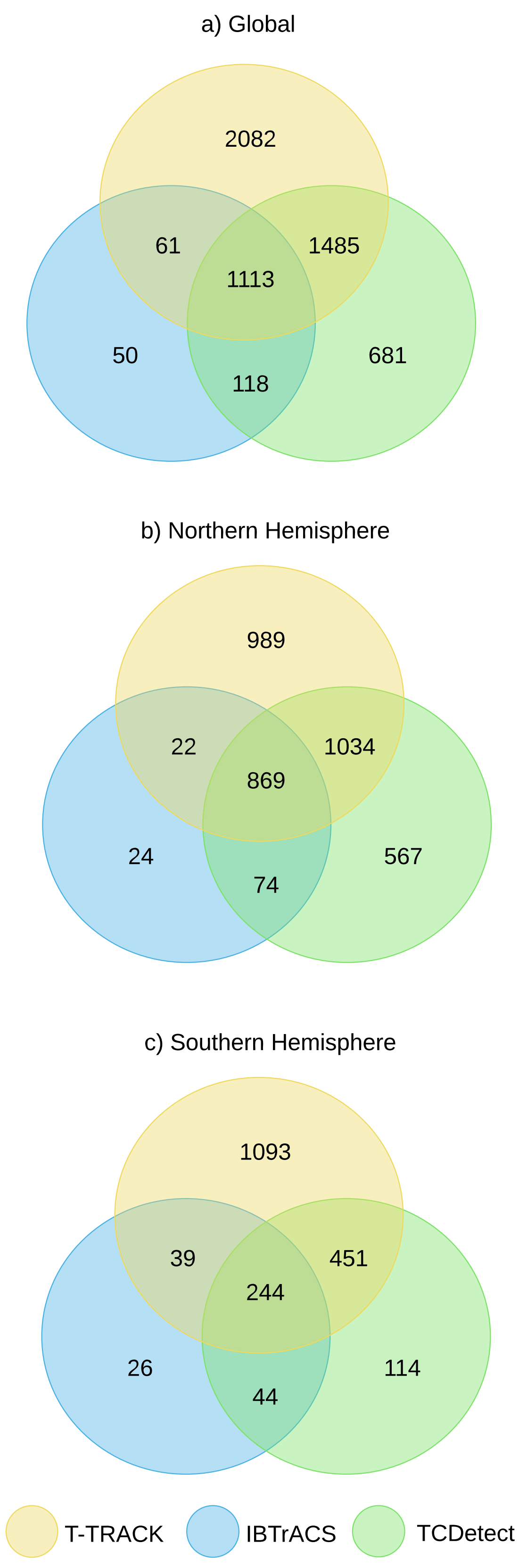


Figure 4.1: Events reported by observations (IBTrACS) and detected by T-TRACK and TCDetect applied to ERA-Interim data for (a) the whole globe, (b) the Northern Hemisphere and (c) the Southern Hemisphere.

Method	Global	NH	SH
T-TRACK	87%	90%	80%
Deep Learning	92%	95%	82%

Table 4.1: Percentage of IBTrACS TC events detected by T-TRACK and TCDetect applied to ERA-Interim data for all regions (global), the Northern Hemisphere (NH) and Southern Hemisphere (SH).

Of the 3397 cases in which TCDetect detects a TC, 681 cases, or around 20% are not observed or detected by T-TRACK, and similarly, of the 4741 cases in which T-TRACK detected the presence of a TC, 2082 cases, or around 44%, are not observed or detected by TCDetect. These "extra" events found by the detection algorithms require more investigation. Formally, they represent poor precision in the detection (a high proportion of false positives), but the significant overlap using two different techniques is interesting, and suggests the techniques are identifying things that are nearly TCs (just outside the tropics, or nearly TC-like in structure and strength, consistent with the results reported previously).

Thus far the analysis has considered timestep "events" since the algorithms (TRACK and TCDetect) are applied to one timestep after another - but in reality these steps form part of the life-cycle of a meteorological phenomenon, and it is that thinking that informs the criteria which distinguish T-TRACK from TRACK. These phenomena move along tracks and so we can consider track detection independently of event detection.

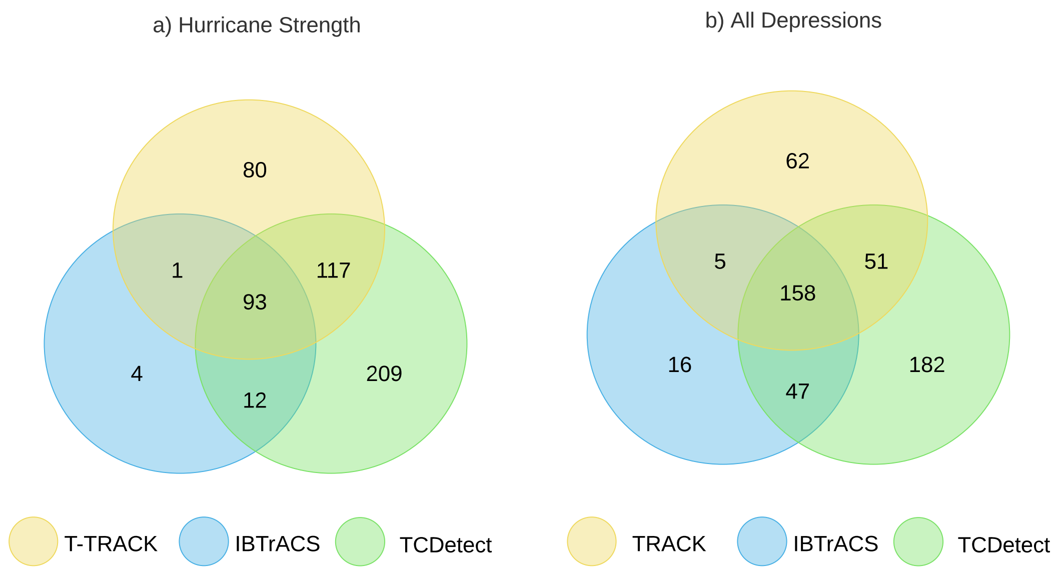


Figure 4.2: Tracks reported by observations (IBTrACS) and detected by T-TRACK and TCDetect applied to ERA-Interim data. Overlaps occur when they share a detection event at some point along the track in the same region at the same timestep. Tracks are matched for (a) only TCs (hurricane-strength) and (b) all depressions (i.e. a superset of a).

In terms of tracks, Figure 4.2a shows how many TC tracks match, whereby two tracks are matched across datasets if they share one or more detection events — in the same region at the same timestep. (Note that this means that a single track from one dataset can be matched to multiple tracks from another dataset if multiple TCs are detected in the second dataset.) Similarly, Figure 4.2b shows matching tracks where depression events were also considered.

The majority (96%) of IBTrACS tracks, whether depressions or hurricanes, match tracks identified by at least one of the two detection algorithms. Similar to the events, TCDetect matched to more IBTrACS tracks than T-TRACK, but a majority (88% of hurricanes) of the matched IBTrACS tracks were with both detection algorithms. Also, there were many hurricane tracks that matched between T-TRACK and TCDetect, but not with IBTrACS. This could be evidence of TC-like structures being picked up by the detection algorithms which either had not strengthened to hurricane strength or were non-tropical systems - an argument supported by the increased number of three-way matches seen when including all depression tracks (4.2b), and our earlier analysis for TCDetect and IBTrACS alone. The most unmatched tracks come from TCDetect, again consistent with results from the previous chapter, were due to many non-meteorological false positives. However, it is encouraging that most of the tracks either produced by TRACK or given in IBTrACS are being matched by tracks produced by TCDetect.

With this life-cycle matching in mind, we revisit event matches (Figure 4.1a) by allowing matches between any class of depression — Figure 4.3. After doing this, it can be seen that only 17 hurricane-strength events were left unmatched. Also, 80 of the events that were detected by TCDetect and present in IBTrACS are now detected by TRACK as well. Similarly, a large number of those events detected by both TCDetect and T-TRACK (1485 to 330) have now migrated into the region where they are detected by both algorithms and are present in IBTrACS, albeit as depressions and not hurricane-strength TCs. This value (1494) also shows that TCDetect is able to detect a large number of other depressions, other than just hurricane-strength TCs.

To further understand the differences between the two methods and the observations, the tracks from both detection algorithms and the observations for hurricane-strength TCs were matched using the following criteria. These were similar to those used by Hodges, Cobb, and Vidale (2017):

- the mean separation distance between all overlapping points between tracks is less than 5^o (geodesic)
- the tracks need to overlap for at least 10% of the base track's lifetime
- the track with the least mean separation distance is chosen if multiple matching tracks exist

These constraints remove any of the unmatched TC tracks, but events can still not match, since they may fall on part of a track where those events were not detected/observed by another method.

After these criteria were applied, the TC events from the remaining tracks were again split by which method picked the events up. The matches between detection and observations now includes fewer TC events (compare Figure 4.4 and Figure4.1a). The number of cases with the presence of a TC as given by

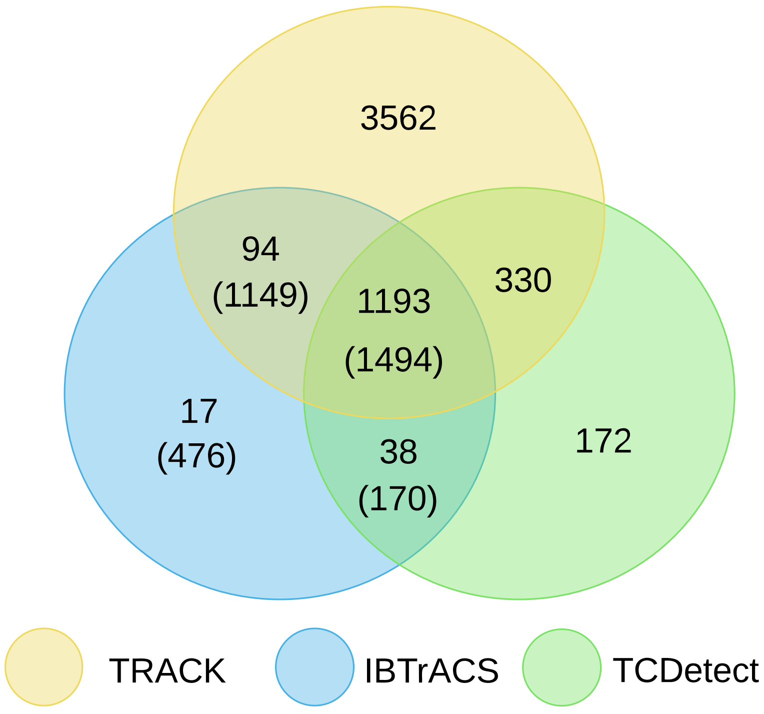


Figure 4.3: Events detected by TRACK, TCDetect and reported by IBTrACS. All meteorological systems are included from IBTrACS and TRACK, not just category 1 and higher systems. Events present in IBTrACS (blue area) were split into TCs of hurricane status (non-bracketed; defined as true positives for TCDetect) and other depressions (bracketed values; defined as false positives for TCDetect and T-TRACK).

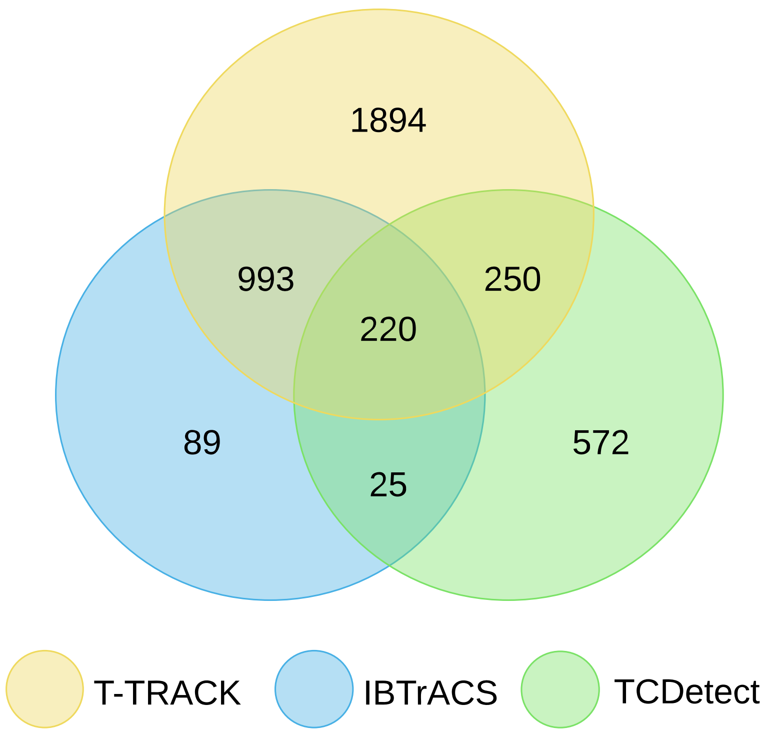


Figure 4.4: Events detected by TRACK, TCDetect and/or reported by IBTrACS which fall on matching tracks, defined by applying constraints similar to those of Hodges, Cobb, and Vidale (2017).

IBTrACS decreases from 1342 to 1327. The same occurs for those given by T-TRACK (4741 to 3357) and TCDetect (3397 to 1067).

As expected, there is only a small change in the total number of IBTrACS events, as the vast majority of TC tracks from IBTrACS were picked up by at least one of the detection methods. Cases from the deep learning model suffered the greatest decrease due to the TC centres generated by the deep learning model not being quite in the right place, thus exceeding the 5° (geodesic) criterion.

4.2.2 Location

The question as to how well TCDetect locates TC centres given the matching technique is now addressed in more detail. Figure 4.5 shows the location of the events reported using each technique following the matching technique discussed above.

The IBTrACS data is here considered to be the ground truth. It shows that most TCs are found in a few well-defined regions:

- close to the eastern shores of the North American continent and further out to the middle of the Atlantic
- to the west of the North American continent and in the middle of the Pacific ocean
- to the east of Asia, over the Western Pacific ocean

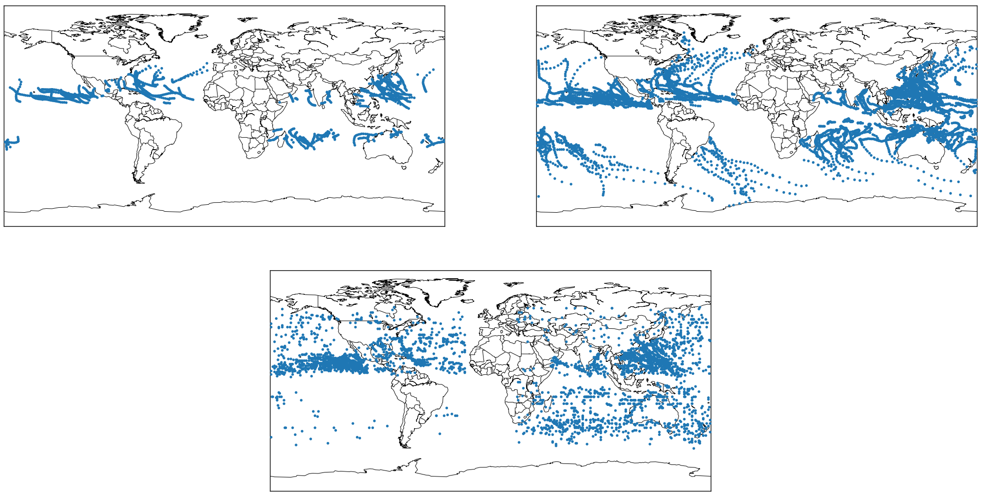


Figure 4.5: Position of each Tropical Cyclone event center as given by IBTrACS (top-left); T-TRACK (top-right) and the deep learning model (bottom).

- over the middle of the Indian ocean and to the north of Australia

In comparison, T-TRACK shows a larger number of events and longer tracks, some extending well into the sub-tropics, suggesting that the constraints applied to TRACK to filter out non-TC storms are not optimal. There are also more TC centres present in the Southern Hemisphere than IBTrACS, especially the Central Southern Pacific ocean. The locations off the eastern coast of the South American continent, which are non-existent in IBTrACS, could point to the use of re-analysis data and tracking algorithms providing better ground truth in observation poor regions of the globe.

The locations reported by TCDetect are positioned mostly in the right regions, but some centres are located well inland or well into the subtropics, where TCs are not expected. Also, the centres over the Indian ocean are more spread out than those found in IBTrACS or T-TRACK. It is clear that the geolocation part of the algorithm is not working as well as the detection algorithm — consistent with the way the deep learning model was developed (it was trained for detection, not location).

This is further confirmed when looking at the TC frequencies generated by each of the detection algorithms and the observational dataset, as shown in Figure 4.6. This shows that while T-TRACK and TCDetect detect more TC tracks, they still follow the same intra-annual variability as given by the observations in IBTrACS. This is especially seen in the panels showing each region separately. Regions in the Northern Hemisphere show an uptick in TC frequencies in the months between July and October, while TC frequencies increase in the months between December and June for regions in the Southern Hemisphere.

We can address this more quantitatively with spatial correlation (Figure 4.7). These show all the matched TC events within 10 ^{o} between TC centres as given by the different sources. The correlations between TC centres given by IBTrACS and T-TRACK (both for a two-way and a three-way match) show a

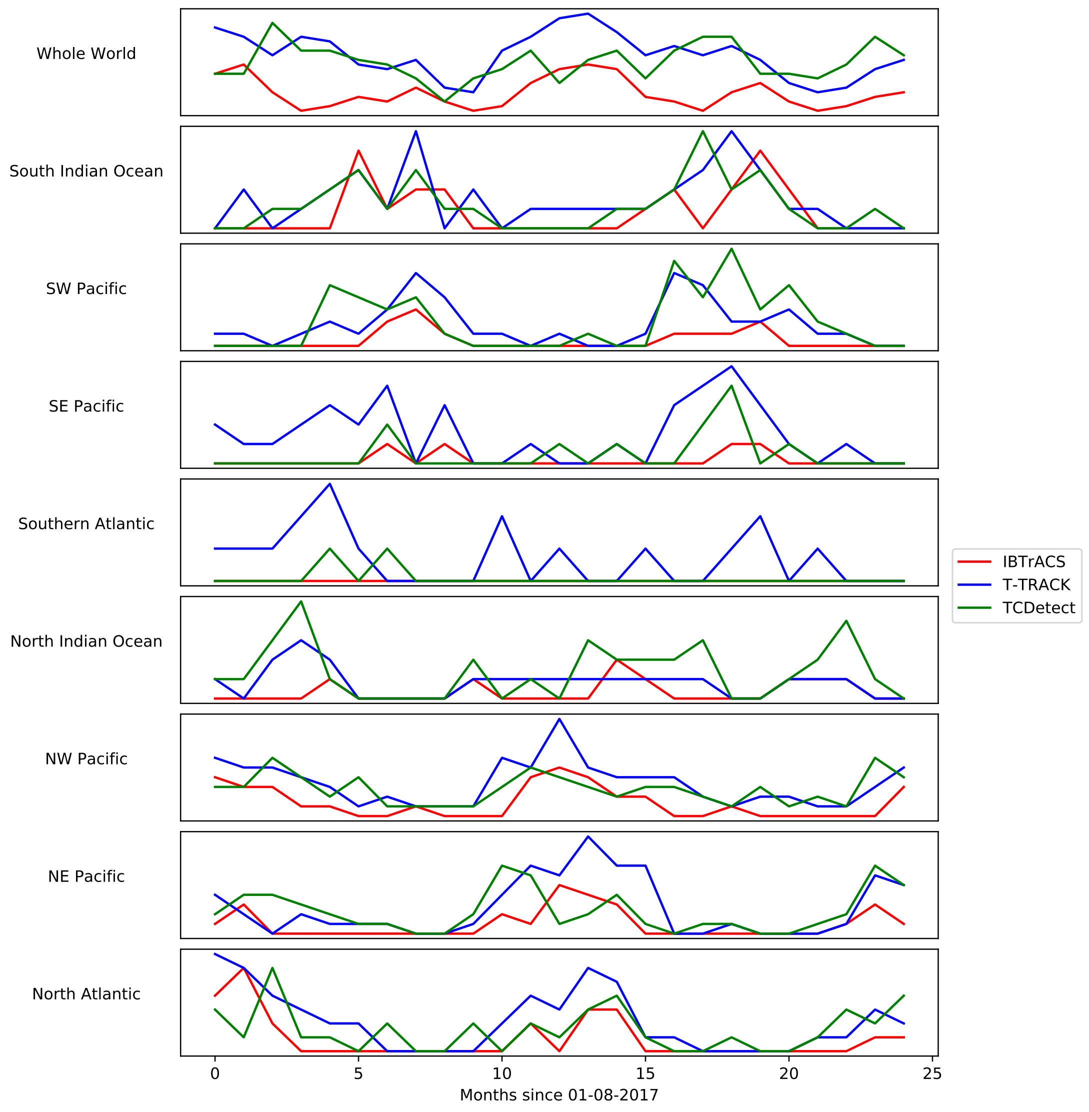


Figure 4.6: TC frequency, i.e. number of TC tracks present in a month, as given by IBTrACS, T-TRACK and TCDetect for each of the regions used by TCDetect as shown in Figure 3.1.

tight grouping and a good correlation, but more scatter is seen in the two-way matches involving TCDetect. This could point to the fact that TCDetect may be producing TC centres in the wrong place but at the right time.

There were some matches which had a difference between TC centres greater than 10 ° (not shown) and were considered not well located. The worst case was for matches between T-TRACK and TCDetect where 57 out of 250 were not well co-located (the other mismatches occurred in 11/220 for the three-way match, with only 2/25 mismatches for IBTrACS and TCDetect). This points to TCDetect not locating TC centres well, but it can also point to an error in the way tracks for TCDetect were created, possibly due to erroneously joining two tracks together, which were in fact two separate TCs.

Figure 4.8 shows the distribution of all TC cases by latitude as generated from both detection algorithms and the observational dataset. While the peak of the distributions for both detection algorithms in both hemispheres is biased equatorwards (with respect to IBTrACS observations) the two detection algo-

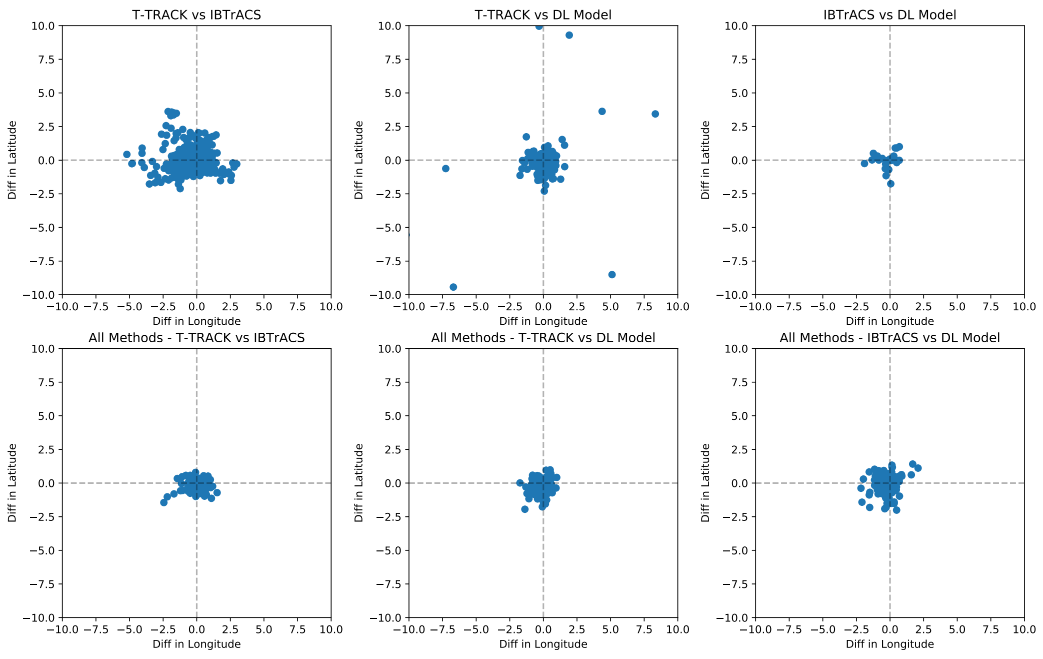


Figure 4.7: Spatial correlation of the overlapping regions shown in Figure 4.4, i.e. for matches with constraints applied. Top row pairwise matches showing pairwise correlation. Bottom row, matches in all three methods, but still pairwise correlations.

rithms broadly agree. However, the distribution for the deep learning based algorithm shows two peaks in the Southern Hemisphere: one at around 10°S and a peak at around 40°S. The first peak matches up well with that from T-TRACK. The second is consistent with the southern bias in positions seen in the Indian Ocean and the excess of detections in and around the Tasman Sea.

4.2.3 Structure

It is feasible that the physical structure of cyclones in terms of their representation in ERA-Interim might affect the results presented here. To investigate this we created composites of the events presented in Figure 4.1 using the ERA-interim data. For each method the composites were created by averaging boxes with sides 30°, centered on the reported TC centre. For cases in which the TC was detected by T-TRACK, the TC centre used was that as given by T-TRACK. Of the remaining cases, if the TC was present in IBTrACS, the centre used was that as given by IBTrACS, and for those TCs that were only detected by the deep learning model, the TC centre used was that as derived from the deep learning model, with the help of the Grad-CAM technique.

The data fields examined were those used as input to the deep learning algorithm: mean sea level pressure (MSLP), 10-m wind speed, and the magnitude of vorticity at 850, 700 and 650 hPa.

The composite case for TCs detected by all three methods shows a fairly symmetric low pressure area with a minimum of around 998 hPa. It also shows a wind field with the maximum wind speed of around 13.5 m s^{-1} in the top-right quadrant of the TC and a clear eye. Finally, vorticity is very concentric

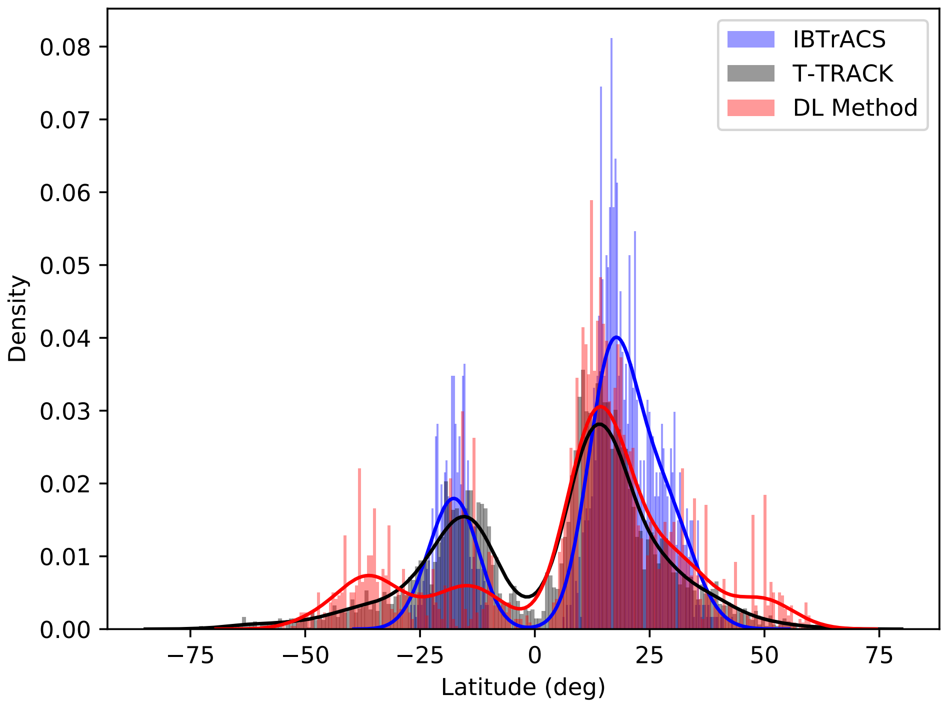


Figure 4.8: Density plots of TC centre latitude as given by IBTrACS (blue), T-TRACK (black) and the deep learning based algorithm (red).

with very little noise with highs of 0.00024 s^{-1}, 0.00021 s^{-1} and 0.000175 s^{-1} at the 850hPa, 700hPa and 600hPa levels respectively. All the features and magnitudes are similar for composites in both hemispheres.

The picture is similar with some subtle differences for the composite cases of TCs detected by two of the three detection methods. MSLP fields for these cases have slightly wider low centres and all have a weaker low with a central pressure no lower than 1000hPa. The wind speed field is similar. All cases show more noise in the composite, especially in the composite case derived from TCs detected by T-TRACK and IBTrACS but not the deep learning model but this is somewhat expected as relatively few TCs are present only in IBTrACS when compared to the other composites. Also, maximum wind speeds are weaker and do not exceed 10.4 m s^{-1}. The vorticity fields show a similar situation where all vorticity centres are wider and those at 850hPa and 700hPa have their maximum magnitude between a third and a half that of the composite case of TCs detected by all detection methods.

When examining these composites when split up by hemisphere, one thing of note emerges. It is seen that both MSLP and wind speed fields have a tighter center of circulation for the composite case coming from cases from the Northern Hemisphere than from those originating in the Southern Hemisphere.

Finally, the composites for TCs detected by only one of the detection methods show some differences from the composite for the TCs detected by all three methods.

As a general note, it is noticeable that wind speed values in the Northern

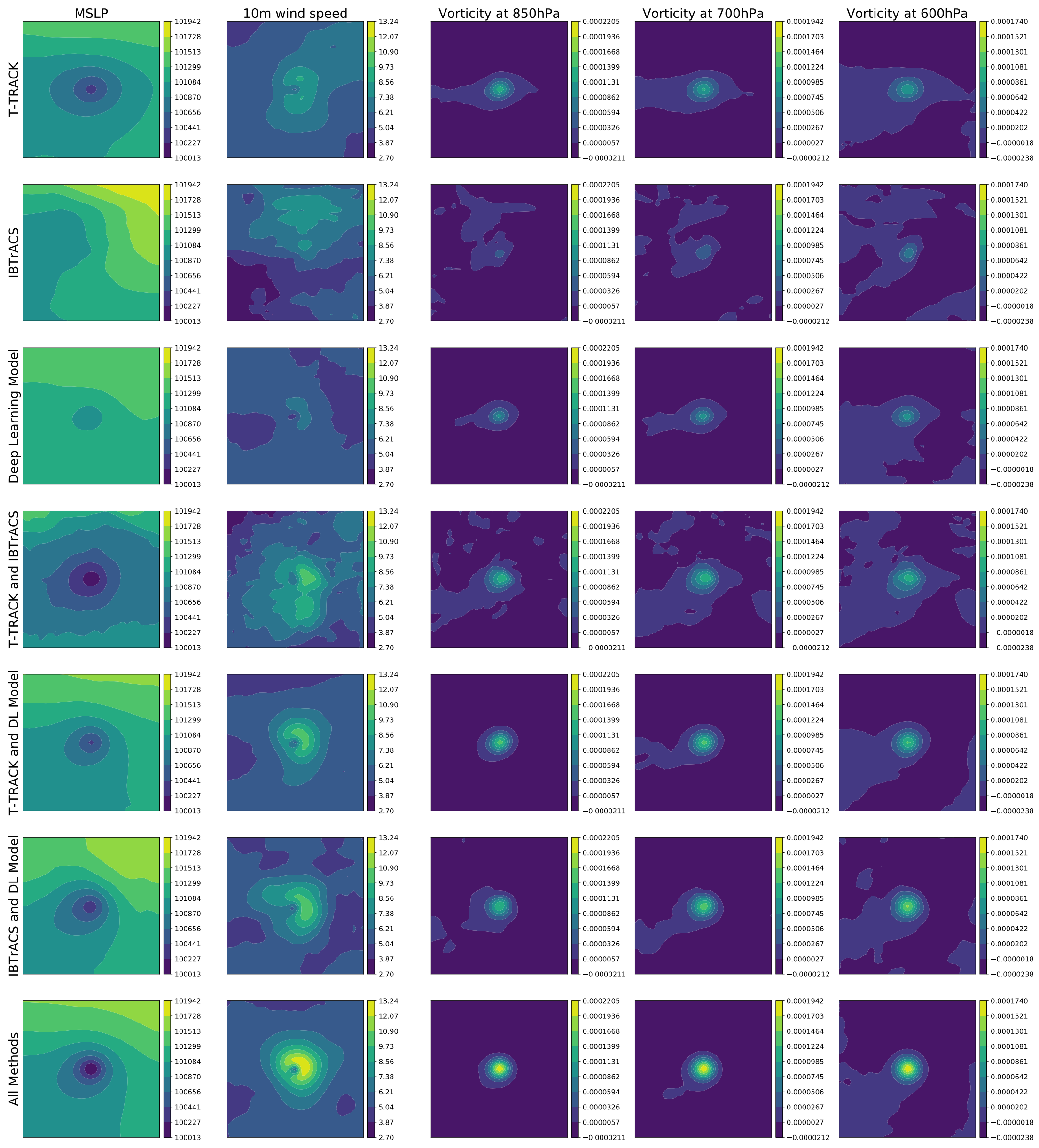


Figure 4.9: Composite view of Northern Hemisphere events by detection algorithm or observations which pick up the TC. Total number of cases used to produce each composite can be obtained from 4.1. Columns correspond to the variables used: MSLP (first column), 10-metre wind speed (second column), vorticity at 850hPa (third column), vorticity at 700hPa (fourth column) and vorticity at 600hPa (fifth column).

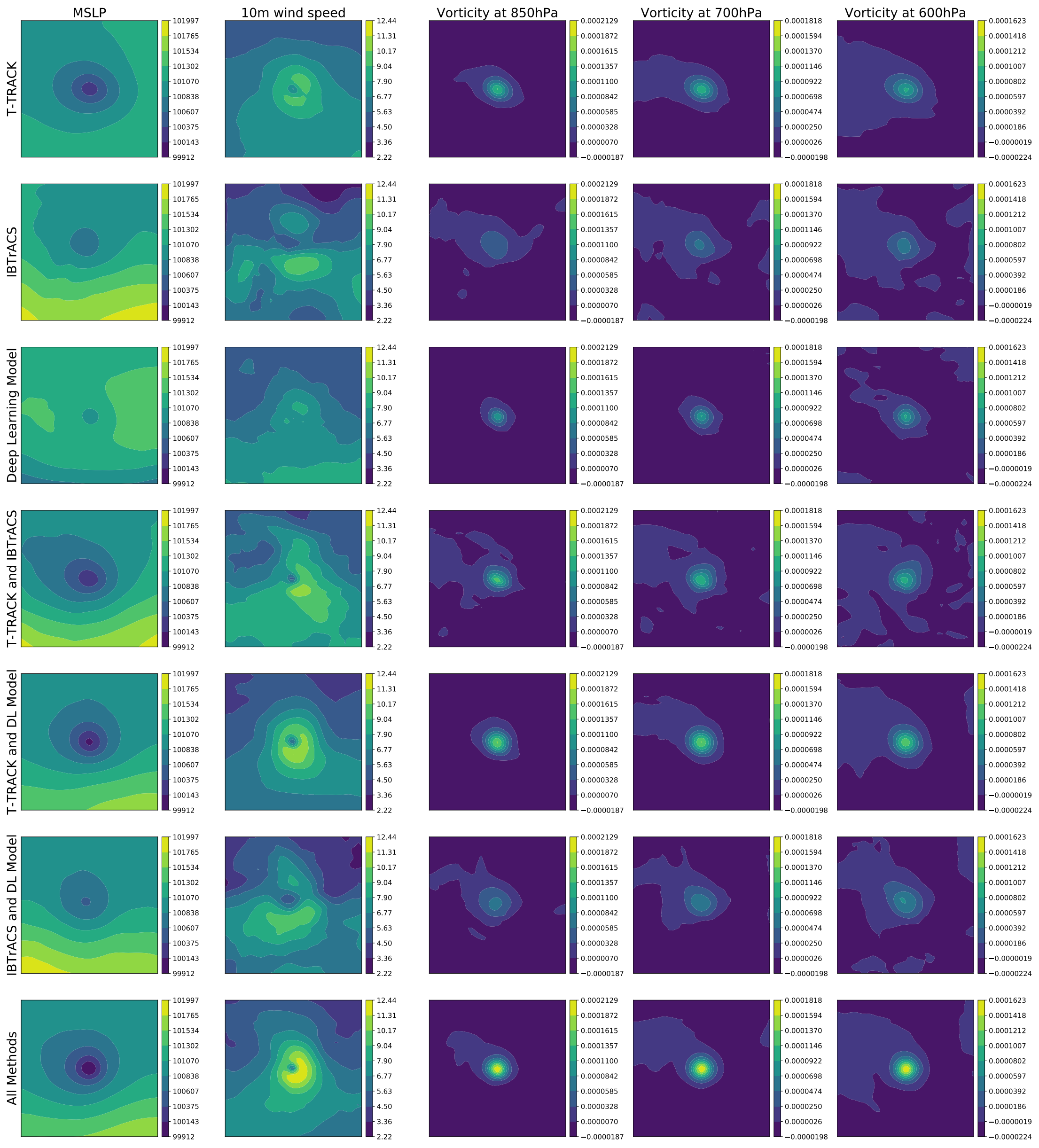


Figure 4.10: Composite view of the Southern Hemisphere cases (rows and columns as described in Figure 4.9) - but the sign of vorticity has been reversed for ease of comparison).

Hemisphere in cases detected by only one of the two methods or present only in the observational data are weaker than those in the Southern Hemisphere.

The composite for TCs present only in IBTrACS shows a low pressure with a considerably higher minimum pressure of 1008hPa. The maximum wind speed is also down to 8.4 m s^{-1}, and does not show a clear eye at the centre of the composite. The vorticity fields show wider but much shallower centres, with the maximum vorticity around half an order of magnitude than that of the composite for TCs detected by all three methods. Considerable noise is also present outside the vorticity centres, but this is somewhat expected as relatively few TCs are detected by IBTrACS only when compared to the other composites.

When split up by hemisphere, these composite cases show some differences. First, the MSLP field in the composite for the Northern Hemisphere cases shows a wave structure rather than a well-defined low. The wind sped field also shows a lack of a centre. The vorticity fields do show clear centers but have considerable noise present.

The composite for cases originating in the Southern Hemisphere shows a much more organised situation. A clear, but wide, low pressure centre is noted, as well as a centre in the wind speed field. The vorticity fields also have well-defined but not concentric centres but there is also a considerable amount of noise present on the outskirts of the centres.

When examining the composite case for TCs detected by the deep learning model only, a concentric centre is observed in the MSLP field with a minimum MSLP of around 1009hPa. A clear centre is also seen in the wind speed and vorticity fields as well. The maximum wind speed is around 7.2 m s^{-1} and the magnitude of the vorticity fields is around half that of the composite case for TCs detected by all three methods.

This situation does not change much when the composite is split by hemisphere. The one difference is that the composite for the Southern Hemisphere shows a relatively shallow area of low pressure in the MSLP field when compared to the composite for TCs detected by all three methods.

Finally, the composite for TCs detected only by T-TRACK is very similar to that of TCs detected by all three detection methods. The only differences are that the magnitudes for voriticty in the former are about half that of the latter. This does not change when the TCs are split by hemisphere.

From the above analysis, it could be concluded that the TCs detected by all three detection methods are the strongest and most well-defined in the data. Furthermore, those detected by two of the methods are weaker, usually with a lack of a clear area of maximum wind speed and somewhat less organised. Finally, those TCs detected by only one detection method are even weaker, with the most noticeable decrease in strength in the vorticity fields.

4.2.4 DL Retraining

From the composite cases, it was seen that three-way matches were the strongest TCs, while the weakest TCs were picked up by only one of the detection methods or were present in the observations. The lead us to check how the model performed on cases with TCs of differing strengths and whether is was struggling with differentiating between TCs and meteorological systems of lesser strength.

Class	Positive Inference	Negative Inference
No meteorological system	506	19253
Unknown	2	30
Post-tropical systems	18	47
Disturbances	165	337
Subtropical systems	32	51
Tropical Depressions	348	625
Tropical Storms	1095	501
Category 1 TCs	426	58
Category 2 TCs	281	26
Category 3 TCs	243	15
Category 4 TCs	212	12
Category 5 TCs	69	0

Table 4.2: Split of cases by storm type (rows) as given by IBTrACS given a positive inference (second column) or a negative inference (third column) by TCDetect. For example, of the 19759 cases which had no meteorological system, TCDetect classified 506 as having a TC present (i.e. false positives). Also of the 484 cases in which a Category 1 TC was the strongest system present, 426 were classified as having a TC (i.e. true positives).

Table 4.2 shows how cases split by the prediction from the deep learning model are split by storm type as given by IBTrACS.

First, the cases predicted as having a TC present were examined. It was found that only 506 out of 3397 cases were cases that had no meteorological system present.

This suggests that the deep learning model is picking up the required pattern needed, due to the high recall values for TCs of strength at least Category 1, but is struggling to distinguish between such TCs and weaker systems. This is not unexpected, as even humans can struggle to get the system's strength right, as it usually depends on wind speeds inside the system and such direct observations are usually hard to come by.

This is further confirmed when checking the cases in which the deep learning model did not detect a TC presence.

This shows that the vast majority of cases with no TCs are being classified as such, i.e. true negatives. Some cases with TCs present are being misclassified (false negatives) with a greater portion of lower category TCs are being misclassified than higher category TCs.

4.3 Chapter Summary

In this study, two automatic detection methods for TCs, namely T-TRACK and a deep learning based algorithm were compared to an observational dataset for TCs, the International Best Track Archive for Climate Stewardship (IBTrACS) database, to discern how they compared when detecting TCs.

A priori we might have expected that the events recorded by IBTrACS would be stronger in the observations than in the reanalysis (Strachan et al.

(2013) Hodges, Cobb, and Vidale (2017), and that some events in the southern hemisphere would be omitted by the observations (Hodges, Cobb, and Vidale (2017)). T-TRACK and the deep learning algorithm found more events overall, with both finding more in the Indian ocean, while the deep learning based algorithm found more over land.

The positions of detected cyclones differs from the observations. The T-TRACK algorithm finds a distribution skewed to higher latitudes, but with the peak at a lower latitude. These differences can be explained by noting that when matching detected cyclones with observed cyclones little difference is observed, and that T-TRACK is finding more cyclones than were observed.

While both the deep learning algorithm and T-TRACK found more (presumably real) cyclones in the southern hemisphere, for the deep learning the matching of detected and observed cyclones was not as good as for T-TRACK. The latitudinal distribution of cyclones found by deep learning also shows a considerable number of cases at around 30°S. These were mostly over the Indian ocean and, although poorly located, were still present in the observations. However, the deep learning model was not trained to find TC centres - rather only to find fields with TCs present - so this represents a failure of (or biases in) the assumption that the centre of the TC is co-located with the centre of the activation points reported by the deep learning.

Those TCs found by both detection methods and observed in IBTrACS were the strongest and most well-defined. Those detected by any two of T-TRACK, IBTrACS and the deep learning were weaker and had more disorganised fields, and those detected by only one of the methods were the weakest storms present and had considerable noise in their fields.

Finally, it was found that most of the false positives generated by the deep learning based algorithm had a TC which did not have hurricane status, therefore it was concluded that the model was picking up the right pattern but was possibly struggling to define the cut-off point between a lower-end TC and a Tropical Storm.

Further work is based on integrating the developed deep learning model into a GCM model for it to make inferences during a model run and questions relating to model and resolution compatibility are to be further investigated.

Chapter 5

Implementation of Deep Learning Model in the UK Met Office's Unified Model

Machine learning, especially deep learning, has been experiencing a period of increased interest as of late. It has also become popular in the meteorological community. However, integrating a deep learning model into a climate model is not trivial. Here, we detail how a deep learning model was integrated into the UK Met Office's Unified Model for the application of an online filtering method for tropical cyclones. We also address how adaptable the method is to data from different sources and horizontal resolutions. We show that the method can be used "as is" for data coming from different simulated climates. For data originating from different simulations and reanalysis, retraining of the DL model is needed if going from higher to lower resolution data, but is found to perform well otherwise.

A deep learning model, TCDetect, has been created to detect the presence of tropical cyclones in model data. Its performance has also been verified against a state-of-the-art detection and tracking algorithm and compared to observational data. A filtering method can use TCDetect to only save TC relevant data to disk. In this chapter, we apply this method in the Met Office's Unified Model (UM).

The process by which the deep learning model, trained in Python, was included in the FORTRAN-based UM is shown in Section 5.2. The expected reduction in data is discussed in Section 5.3.3. The ability for the method to be applied on data from different simulations and reanalysis is explored in Section 5.3.1. TCDetect inference was inserted into a model and used in a range of simulations. These different simulations, at different horizontal resolutions and for different climates constitute different sources. Finally, the method's computational performance is detailed in Section 5.4.

This chapter is being prepared for publication; contributions - Daniel Galea: coding, data collection, writing, editing; Bryan Lawrence: writing, editing, direction; Jeff Cole: direction with UM coding; Simon Wilson: help with setting up UM runs

5.1 The Met Office Unified Model

The deep learning model developed here, TCDetect, was used during a climate simulation. Due to the expertise available, this was chosen to be the GA7.0 UM11 AMIP (Atmospheric Model Intercomparison Project) version of the UK Met Office's Unified Model.

The original workflow internal to the UM is shown in Figure 5.1. It starts by initialising various variables and arrays which are required to store any calculated prognostic and diagnostic variables in memory. It then loads the starting conditions. The number of timesteps required is calculated and a loop is used to process the required timesteps.

A timestep is computed by executing various physics schemes with potential data writing interspersed between different schemes. First, local variables used in the timestep are initialised. Then a physics scheme is run and any required output variables are written. Another physics scheme is then executed and any further required prognostic variables are written. The diffusion scheme is run next with potentially more data writing of prognostic variables following. The dynamics solver is subsequently executed and any further required variables are written. Penultimately, the aerosol modelling scheme is run and is followed by the writing out of any needed variables. Finally, the incremental analysis unit (IAU) is performed. The IAU is a method by which a variable is changed across several timesteps. Changing a field at each timestep introduces noise to gravity waves, so the IAU introduces the changes across a series of timesteps to help lessen these. Also, any end of timestep diagnostics are calculated and any final required variables are written.

After all of the timesteps required are computed, the Unified Model finishes the simulation by performing some cleanup routines before exiting.

For this study, the UM was used to produce two different simulations, each at different resolutions. The first was a standard AMIP simulation of 20.5 years at a horizontal resolution of N96 ($\approx$135km). It was run on 8 nodes of the ARCHER supercomputer, with each node having 24 processing elements. The domain was decomposed into 12 East-West processes and 8 North-South processes with 2 OpenMP threads, where OpenMP is a library which enables shared memory parallelisation. The second was a 3 year simulation at a horizontal resolution of N512 ($\approx$25 km). It was run on 104 nodes of the ARCHER2 supercomputer. While each node has 128 processing elements, the nodes were depopulated to 50 processes per node to avoid out-of-memory errors produced by the UM. The domain was decomposed into 36 East-West processes and 36 North-South processes with 4 OpenMP threads.

Data from these runs is used to quantify how much data would not be saved to disk had the method been implemented fully. These datasets will also help understand how the method performs with one data source in the training dataset and another one for the testing dataset and also how the method performs when using data having different native horizontal resolutions.

The computational performance of the method is also calculated in shorter, seven-simulated-days runs at both resolutions.

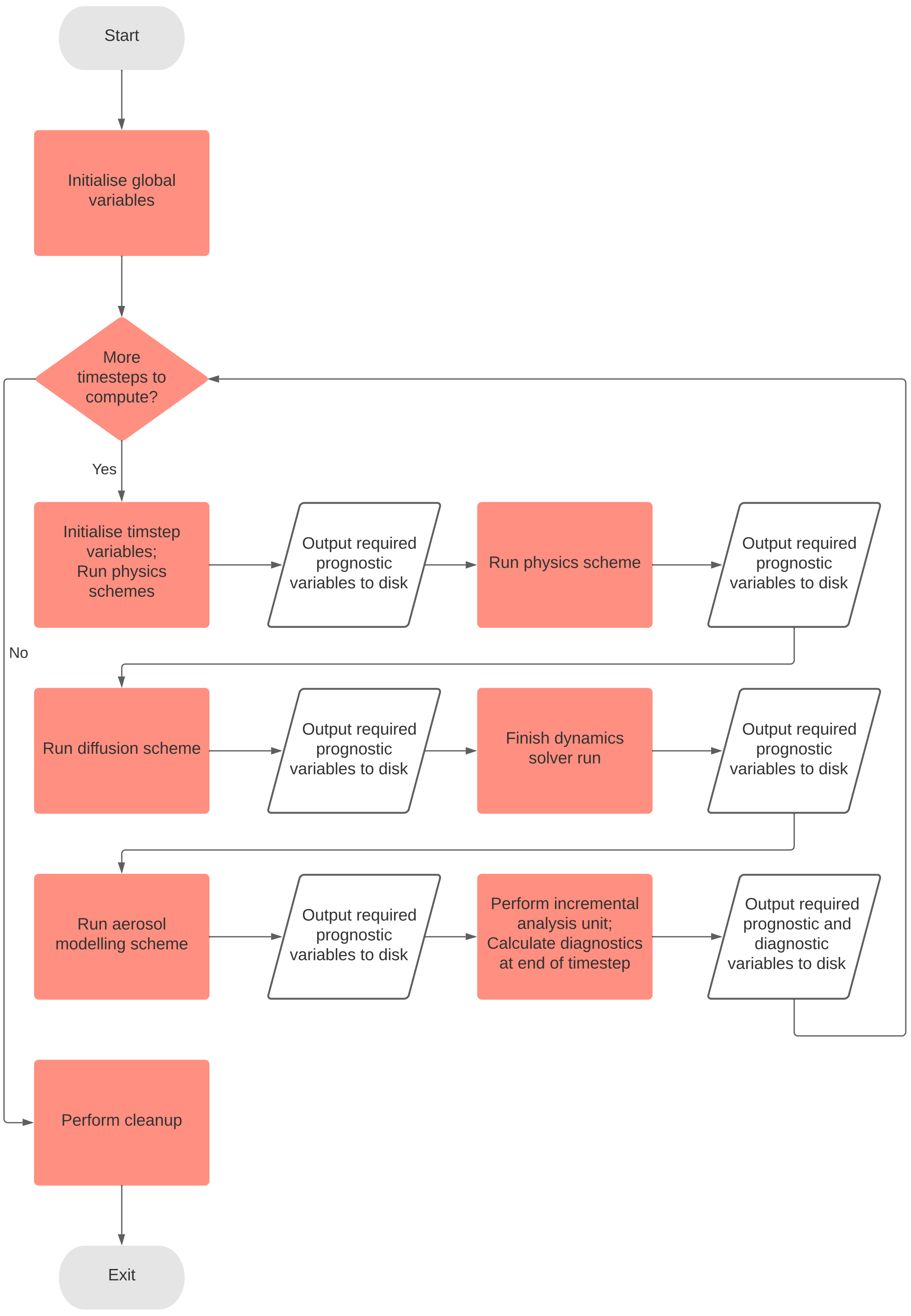

Start
Initialise global variables
More timesteps to compute?
Yes
No
Initialise timstep variables; Run physics schemes
Output required prognostic variables to disk
Run physics scheme
Output required prognostic variables to disk
Run diffusion scheme
Output required prognostic variables to disk
Finish dynamics solver run
Output required prognostic variables to disk
Run aerosol modelling scheme
Output required prognostic variables to disk
Perform incremental analysis unit; Calculate diagnostics at end of timestep
Output required prognostic and diagnostic variables to disk
Perform cleanup
Exit

Figure 5.1: The original workflow of the UK Met Office's Unified Model to carry out a full climate simulation run. Objects in white are data saving steps in the UM.

5.2 Implementation

In the previous section, Figure 5.1 showed the original workflow of the UM. In this section, Figure 5.2 shows the amended workflow that uses the developed deep learning model in the timesteps previous to those ones which write out analysis data. New processes are given in green.

A C++ header-only package, frugally-deep (Hermann (2020)), is utilised as an intermediate layer to be able to use the deep learning model, trained in Python, as an inference engine in the FORTRAN-based UM. An explanation on how this package works is contained in Appendix A. A FORTRAN module utilising this C++ package was written in which subroutines to load and use the deep learning model as well as perform any data manipulation needed were created. The SPHEREPACK package (Adams and Swarztrauber (1999)) was also used to perform the calculations needed for spherical filtering of data.

The deep learning model, stored in a JSON file when saved in Python, is loaded by the process controlling the simulation before any timesteps are processed. The spherical coefficients needed for spherical filtering at the resolution of ERA-Interim are also calculated and stored by the same process. The spherical coefficients are calculated once at the initialisation stage and are used in each timestep to calculate the spherical harmonics of the data. The required flags on which the data-writing decision are determined are also initialised and set to false.

The first set of timesteps before the timestep which is meant to write out analysis data is then calculated. At this point, the deep learning model is utilised to check whether it detects a tropical cyclone in any of the regions in the timestep just calculated.

The required data - the 10-metre wind speed, MSLP and vorticity at 850hPa, 700hPa and 600hPa fields - are gathered from their host processing elements onto that controlling the simulation. All others are forced to wait until the method is run fully.

The collected data is then resized to ERA-Interim resolution using simple linear interpolation if necessary. Spherical filtering is then applied such that wavenumbers 5 to 106 are kept for the MSLP and 10-metre wind fields and wavenumbers 1 to 63 are kept for all vorticity fields.

At this point, the global image is split up into the eight regions of Figure 3.1 and passed onto the deep learning model for inference making. The regions are passed to the deep learning model serially and the inferences are collected. If TCDetect infers the presence of a TC in any of the regions, the required flags for data-writing are set to true so that data from any of the required regions in the next timestep is written out to disk. At this point, all CPUs are released to calculate the next timestep. This continues until all timesteps are calculated and the climate model exits.

It should be noted that the method is aimed at reducing the size of certain data outputs. Any climate model produces multiple types of data which are outputted at various different intervals. The UM outputs three different types of data: dumps, means and analysis data. Dumps are when a model checkpoints its execution by writing out all of its data to disk, so that if anything happens in the rest of its execution, the model can be restarted at these checkpoints

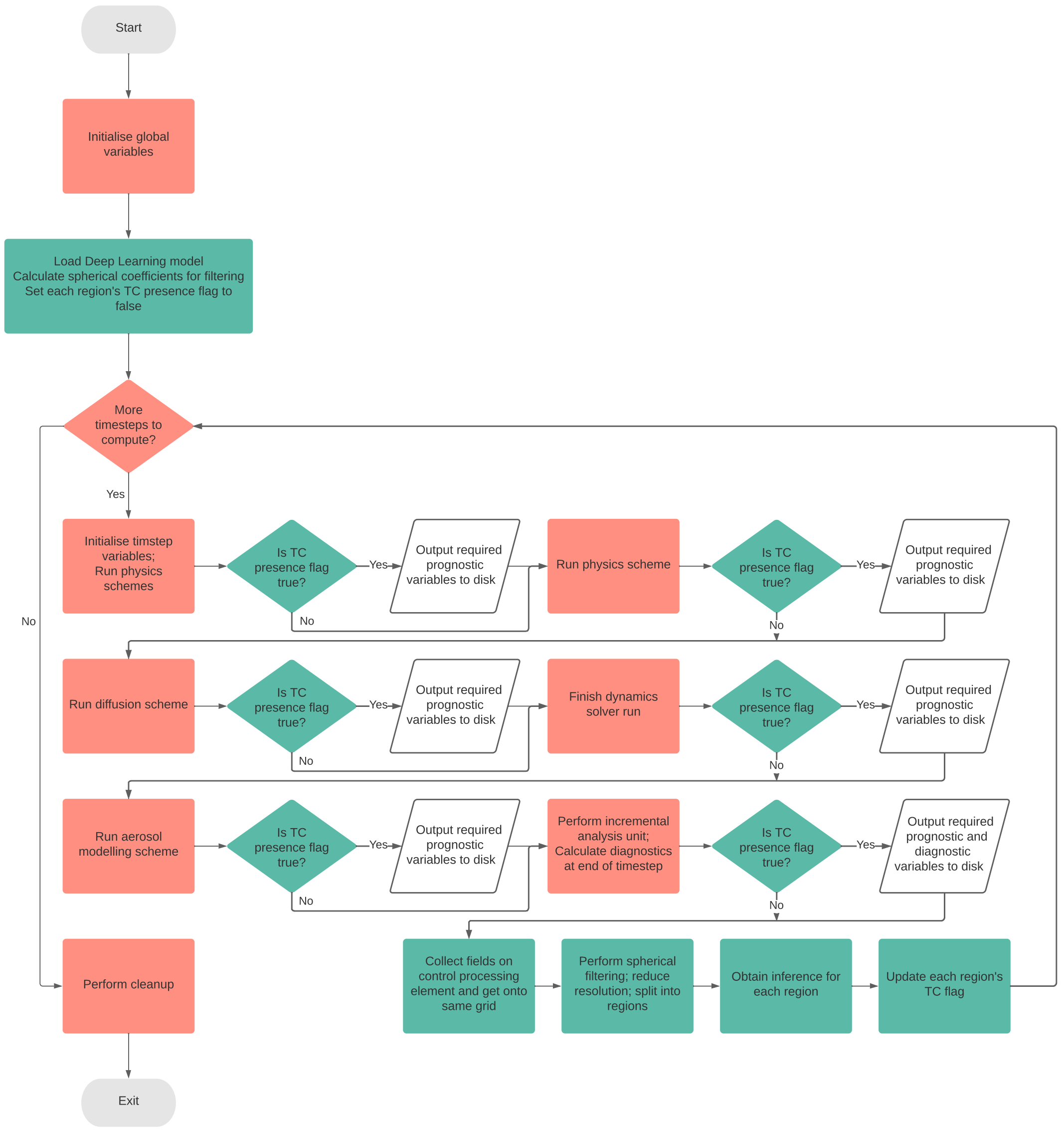

Start
Initialise global variables
Load Deep Learning model
Calculate spherical coefficients for filtering
Set each region's TC presence flag to false
More timesteps to compute?
Yes
No
Initialise timstep variables; Run physics schemes
Is TC presence flag true?
Yes
No
Output required prognostic variables to disk
Run physics scheme
Is TC presence flag true?
Yes
No
Output required prognostic variables to disk
Run diffusion scheme
Is TC presence flag true?
Yes
No
Output required prognostic variables to disk
Finish dynamics solver run
Is TC presence flag true?
Yes
No
Output required prognostic variables to disk
Run aerosol modelling scheme
Is TC presence flag true?
Yes
No
Output required prognostic variables to disk
Perform incremental analysis unit; Calculate diagnostics at end of timestep
Is TC presence flag true?
Yes
No
Output required prognostic and diagnostic variables to disk
Perform cleanup
Collect fields on control processing element and get onto same grid
Perform spherical filtering; reduce resolution; split into regions
Obtain inference for each region
Update each region's TC flag
Exit

Figure 5.2: The UM workflow for a full climate simulation run when using the deep learning model inference to decide whether to write out data to disk. Red objects show process present in the original workflow shown in Figure 5.1. White Objects are data saving routines. Green object show the extra processes needed to use the deep learning model in the UM.

instead of having to start from the beginning. The next type of data is mean data. This is when climatological means for the simulation are outputted to disk. These are used when studying trends of simulated phenomena at a coarse temporal resolution, usually monthly or seasonally. The final type of data is analysis data. This data is when the state of the simulated system is outputted for further analysis after the simulation has finished executing. Most of this data is produced for every few simulated hours and is useful when studying how a certain simulated event evolves over time. Our method is specifically aimed at this last type of data. In most cases, analysis data is outputted every six simulated hours and thus, this filtering method is run at the same temporal frequency. Therefore, in the N96 and N512 simulations used here, this method is run every 18 and 36 timesteps respectively.

(For the purposes of the following sections, the result of the presence of a TC is output to the run logs and no changes to the data writing functions were made. This was so that the data could be collected and used in further investigation.)

5.3 Results and Discussion

This section will present the results and discuss the adaptability of the method across different data sources. It will also discuss the effect of changing labelling techniques as well as the the amount of data reduction achieved when applying the method described above.

Six different data sources are used: the original data from ERA-Interim with labels obtained from IBTrACS, the same data but with labels obtained from T-TRACK and data from the N96 and N512 runs of the UM detailed in Section in 5.1. The final two datasets that were used were obtained from the CMIP6 project (Eyring et al. (2016)). The first, termed Hist1950, uses the model output from ensemble member r1i1p1f1 of a run of Natural Environment Research Council (NERC) HadGEM3-GC31-HH model from the hist-1950 set of simulations (Met Office Hadley Centre (2020)). This is a coupled atmosphere-ocean model using historical temperature forcings. The second, termed Future, uses the model output from ensemble member r1i1p1f1 of a run of the same model from the highres-future set of experiments (Centre ()), where the modelled climate is forced to be as close to the CMIP5 RCP8.5 scenario.

Our ERA-Interim testing period is 25 months, while it is 36 months and 12 months respectively for the UM N96 and N512 simulations. The Hist1950 dataset is 13 years long and the Future dataset is 6 years long. These represent around 20% of the available data, the other 80% of which was used to train the different variants of TCDetect. Table 5.1 summarises the different datasets used in this section and includes their horizontal resolutions and length in simulated years.

As the deep learning model underpins the whole method, we will be discussing performance of the method in terms of the deep learning model's recall rate.

We should note that TCDetect and the method as a whole will not obtain a perfect recall rate. As discussed before, no detection algorithm can detect all of the TCs present. For example, TRACK detects 97% of observed TCs.

Dataset	**Labelling Method**	**Horizontal Resolution**	**Length of Testing Dataset / simulated years**
ERA-Interim	IBTrACS	∼79 km	2
ERA-Interim	T-TRACK	∼79 km	2
UM N96	T-TRACK	∼135 km	3
UM N512	T-TRACK	∼25 km	1
Hist1950	T-TRACK	∼25 km	13
Future	T-TRACK	∼25 km	6

Table 5.1: Datasets used for exploring the adaptability of the method across different data.

From the Venn diagram of Figure 4.1a, T-TRACK correctly classified 87% of the regions shown in 3.1 in which IBTrACS observed a TC. Also, the other detection methods discussed in Section 2.5 were shown to need to have their thresholds tuned to the datasets used, inferring that even they cannot detect all the TCs present. IBTrACS itself is not perfect as discussed in Section 4.1.1. With this in mind, the performance of our method is discussed below.

5.3.1 Method Adaptability

An important aspect of this method being presented it how adaptable and reliable it is to different data sources. The following attempts to understand this.

TCDetect, as a deep learning model, is made up of two parts - its architecture, i.e. how each layer is arranged to make up the final model; and its weights, which are the values used to make an inference. To quantify the ability of the model to be applied across different data sources, we keep the same underlying architecture as TCDetect but retrain the model on each data source to come up with different sets of weights to make up new variants of TCDetect. TCDetect is the original model as trained on ERA-Interim data with labels obtained from IBTrACS. TCDetect-TRACK is the model when trained on the same data but using labels derived from T-TRACK. TCDetect-N96 and TCDetect-N512 refer to models trained on N96 and N512 UM data respectively, with labels acquired from T-TRACK. Finally, TCDetect-CC and TCDetect-FC are models trained on the Hist1950 and Future datasets from the CMIP6 HighResMIP experiment.

The recall rate for each of these models for each of the datasets available is tabulated in Table 5.2. Table 5.3 shows the mean and standard deviation of the monthly recall rate for each model as tested on each dataset.

TCDetect obtained a recall rate of 92% on the ERA-Interim dataset which was labelled using IBTrACS. However, in the following work, we will be utilising data from simulations using the UM. For this, IBTrACS is not available. Therefore, the first step was to retrain the model using T-TRACK labels.

The version of TCDetect trained with labelling obtained from T-TRACK is now termed TCDetect-TRACK. As seen from Table 5.2, this obtained a recall rate of 85% when it is tested on data with IBTrACS labels, compared to the 87% obtained from T-TRACK, and 59% when tested on data with T-TRACK labels. T-TRACK is known to contain some systems which are not TCs, so it

		Detection Method					
Data Source	**Label Source**	**TCDetect**	**TCDetect -TRACK**	**TCDetect -N96**	**TCDetect -N512**	**TCDetect -CC**	**TCDetect -FC**
ERA-Interim	IBTrACS	92%	85%	97%	91%	93%	89%
ERA-Interim	T-TRACK	59%	62%	76%	70%	73%	62%
UM N96	T-TRACK	34%	41%	78%	50%	62%	44%
UM N512	T-TRACK	58%	58%	71%	72%	77%	60%
Hist1950	T-TRACK	60%	58%	73%	73%	77%	64%
Future	T-TRACK	62%	62%	74%	74%	79%	66%

Table 5.2: Recall rates when each variant of TCDetect was tested on combinations of different data and labelling sources.

		Detection Method					
Data Source	**Label Source**	**TCDetect**	**TCDetect -TRACK**	**TCDetect -N96**	**TCDetect -N512**	**TCDetect -CC**	**TCDetect -FC**
ERA-Interim	IBTrACS	$\bar{x}$: 88% σ : 13%	$\bar{x}$: 84% σ : 17%	$\bar{x}$: 95% σ : 9%	$\bar{x}$: 87% σ : 19%	$\bar{x}$: 92% σ : 18%	$\bar{x}$: 87% σ : 16%
ERA-Interim	T-TRACK	$\bar{x}$: 51% σ : 24%	$\bar{x}$: 59% σ : 16%	$\bar{x}$: 71% σ : 17%	$\bar{x}$: 64% σ : 19%	$\bar{x}$: 69% σ : 16%	$\bar{x}$: 56% σ : 20%
UM N96	T-TRACK	$\bar{x}$: 33% σ : 16%	$\bar{x}$: 38% σ : 15%	$\bar{x}$: 76% σ : 13%	$\bar{x}$: 49% σ : 14%	$\bar{x}$: 59% σ : 15%	$\bar{x}$: 42% σ : 15%
UM N512	T-TRACK	$\bar{x}$: 58% σ : 14%	$\bar{x}$: 59% σ : 13%	$\bar{x}$: 71% σ : 13%	$\bar{x}$: 72% σ : 11%	$\bar{x}$: 77% σ : 9%	$\bar{x}$: 60% σ : 13%
Hist1950	T-TRACK	$\bar{x}$: 57% σ : 13%	$\bar{x}$: 72% σ : 11%	$\bar{x}$: 72% σ : 13%	$\bar{x}$: 77% σ : 10%	$\bar{x}$: 63% σ : 14%	$\bar{x}$: 60% σ : 17%
Future	T-TRACK	$\bar{x}$: 60% σ : 17%	$\bar{x}$: 60% σ : 15%	$\bar{x}$: 73% σ : 14%	$\bar{x}$: 73% σ : 14%	$\bar{x}$: 78% σ : 12%	$\bar{x}$: 65% σ : 16%

Table 5.3: Monthly mean and standard deviation recall rates when each variant of TCDetect was tested on each different data source.

is not surprising that the performance of the model decreased. This could be due to the different types of systems present in the labelling system that may be forcing the deep learning model to try to learn two different patterns. We attempt to see if this is the case by splitting the cases in the test dataset by the highest category of any TCs present as given by IBTrACS, with results shown in Table 5.4. When compared to TCDetect's results in Table 4.2, TCDetect-TRACK produces more than double the true false positives (positive inference; no meteorological system) than TCDetect, while its recall of category-strength storms decreases.

Now that we have a model trained with T-TRACK labelling, we can use it to determine how adaptable the method is to data from different sources.

Starting with the original ERA-Interim data, the original model, TCDetect, obtained a recall rate of 92% while TCDetect-TRACK obtained a recall rate of 85%. On the other hand, most of those models trained on UM data obtained recall rates close to or better than TCDetect, with the notable exception being TCDetect-FC, the model trained on a future climate. These numbers start to indicate that differences are to be expected in the model's performance when applying it to data from different sources and different climates.

When considering ERA-Interim data with T-TRACK labels, a clear stratification can be noted. Those models trained on UM data, with the same exception as before in TCDetect-FC, obtain a higher recall rate than those trained on ERA-Interim data, although all recall rates are more 20% lower in each case. This is, in fact seen when testing with the rest of the available datasets. This points to the possibility that models trained on UM data continue to perform

Class	Positive Inference	Negative Inference
No meteorological system	1393	18366
Unknown	5	27
Post-tropical systems	20	45
Disturbances	111	391
Subtropical systems	40	43
Tropical Depressions	320	653
Tropical Storms	1049	547
Category 1 TCs	395	89
Category 2 TCs	252	55
Category 3 TCs	224	34
Category 4 TCs	208	16
Category 5 TCs	68	1

Table 5.4: Split of cases by storm type (rows) as given by IBTrACS given a positive inference (second column) or a negative inference (third column) by TCDetect-TRACK.

well when used on ERA-Interim data, but not vice-versa.

These values are reflected in the mean and standard deviation of the monthly recall rate. The standard deviations reported when the models are tested on UM data are smaller than when tested on ERA-Interim data. This could point that the models perform more consistently across all timesteps, although some caution needs to be taken due to the smaller testing dataset in the latter case. It should also be noted that the standard deviation values are still large, showing that the differences in the mean monthly recall may not be representative, however Figure 5.3 shows that these differences are representative.

Furthermore, the model trained on the dataset considered obtains the best or second best recall rate. This is expected as from the previous discussion in Chapter 2.5, classical algorithms performed best when they were tuned to the dataset being considered. There is no reason to expect a deep learning algorithm to differ from this behaviour. Once again, these findings are consistent with the numbers reported for the mean daily recall.

It was also noted that all models except one, TCDetect-N96, performed better on UM N512 data than UM N96 data. For TCDetect-N96, performance on N512 data was worse. These findings could point to the model being applicable to higher resolutions, albeit with a slight drop in performance, but not vice-versa. To be able to go from low resolution to a higher resolution, a simple retraining of the model would capture some of the original model's performance back, but not all of it.

The different variants of TCDetect were tested on the two datasets intended to infer the adaptability of the method to different climates, Hist1950 and Future. The recall rates across the two datasets as well as the mean monthly recall rates and their standard deviations were very close. This shows that the method is found to work in future climates if an acceptable recall rate is obtained on data from the current climate, at least for detecting TCs.

Finally, we touch on the reliability of the method, i.e. the ability of the method to produce similar performance across the dataset. To check this, we show the monthly recall rate from the TCDetect, TCDetect-TRACK, TCDetect-N96 and TCDetect-N512 models across the UM N96 testing dataset (Figure 5.3). This shows a high variability with one clear drop in performance across all models around May 2006. This occurs when there are the fewest TCs, and most of which come from the South Atlantic and South Eastern Pacific regions (Figure 5.4). These are the two regions with which TCDetect struggles the most due to a lack of observed TCs from IBTrACS in the region. From Table 5.3, this variability is highest when testing on data from ERA-Interim with labels from IBTrACS. This decreases slightly when using T-TRACK labels and decreases further when using UM data.

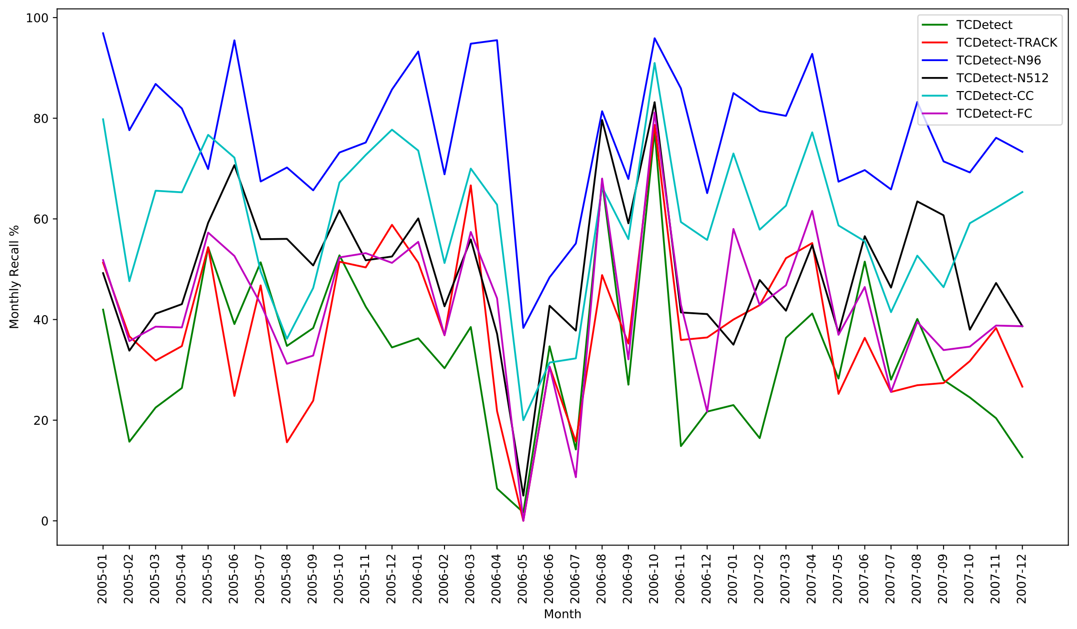


Figure 5.3: Monthly ecall rate from the TCDetect, TCDetect-TRACK, TCDetect-N96, TCDetect-N512, TCDetect-CC and TCDetect-FC models across the UM N96 testing dataset.

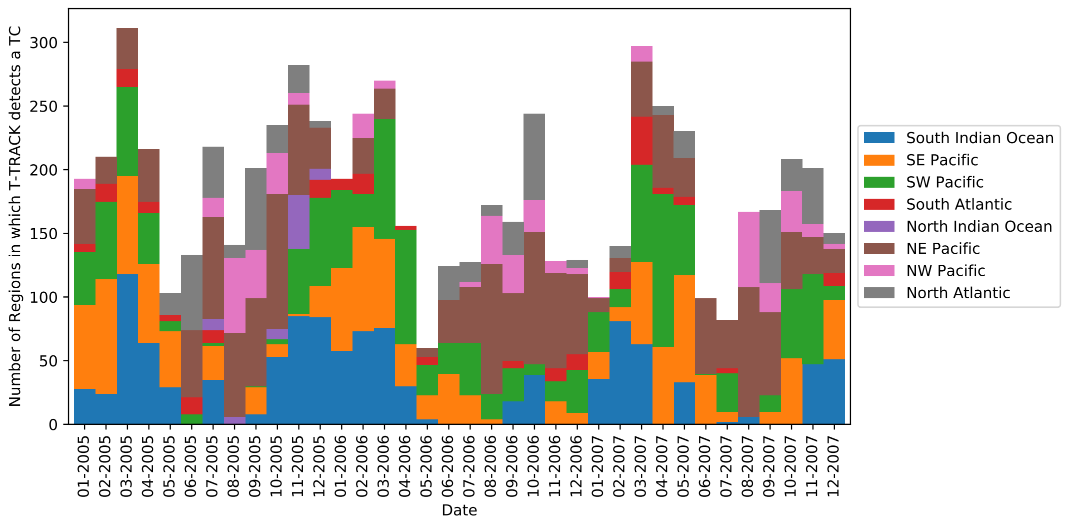


Figure 5.4: Monthly number of regions having a TC detected by T-TRACK across the UM N96 testing dataset.

5.3.2 The Effect of Different Labelling

One aspect which might be important to understand is that of the effect of changing the labelling source. In the previous section, we noted that IBTrACS, the labelling source of TCDetect, is not available for UM simulations. Hence, the labelling source was switched to T-TRACK. While the change was done due to necessity, it is important to investigate how that effected the performance of the deep learning model.

To do this, the analysis of Chapter 4 was repeated with TCDetect-TRACK and compared to that done for TCDetect.

Firstly, Figure 5.5 shows how these two models compare to each other in terms of the regions in which a TC was detected by the deep learning models, T-TRACK and observed in IBTrACS. Figure 5.5i repeats Figure 4.1 for ease of comparison.

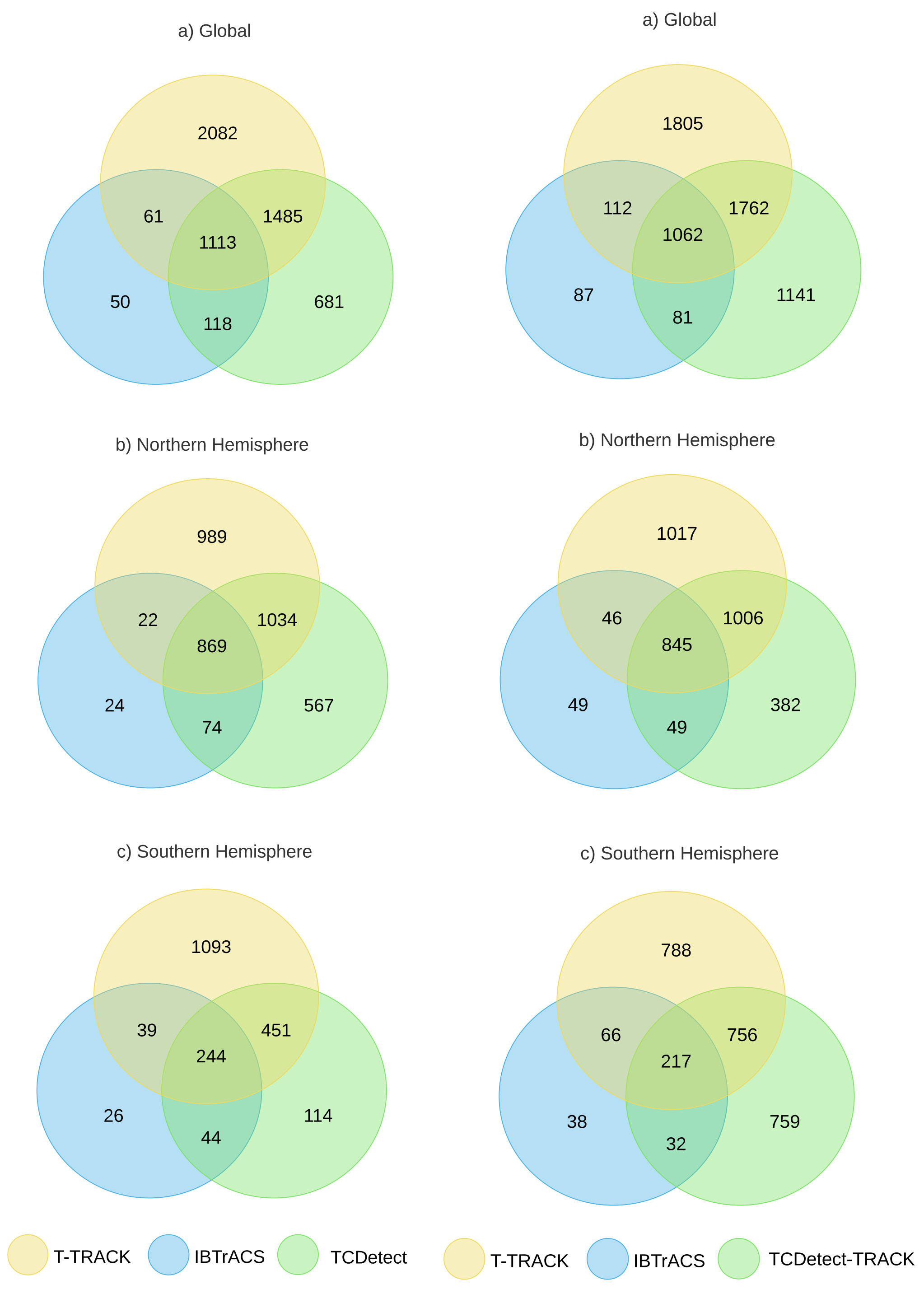
It could be noted that TCDetect-TRACK detects the presence of a TC in more regions than TCDetect. This is expected as T-TRACK detects more TCs than IBTrACS, so the model trained with T-TRACK labels is expected to mirror the labelling more closely. However, the proportion of the regions in which a TC has only been detected by the deep learning model grows (20% by TCDetect vs 28% by TCDetect-TRACK). This increase points to a larger number of false positives produced, as shown in Table 5.4.

Despite the increase in the total number of regions in which TCDetect-TRACK detected a TC, the number in the Northern Hemisphere decrease slightly while the number in the Southern Hemisphere increase considerably. The latter aligns to the knowledge that IBTrACS underestimates the TC count in the Southern Hemisphere. This is mirrored in the number of matches between the deep learning model and T-TRACK in both hemispheres.

TCDetect-TRACK detected more of the TCs also detected by T-TRACK than TCDetect did (intersection of yellow and green areas). This is expected due to the labelling used when training TCDetect-TRACK as this model is expected to perform closer to the new labelling. However, fewer regions in which IBTrACS observed a TC were detected by TCDetect-TRACK (intersection of blue and green regions). This, and the knowledge that T-TRACK produces tracks which are too long, i.e. containing systems which are not hurricane-strength TCs, could be pointing to a possibility that the labelling is making the model learn different patterns for a positive inference during training.

This is more evident in Figures 5.6 and 5.7, which show the composite TCs of all cases by Venn diagram region (5.5i) when using TCDetect-TRACK. When compared to the similar plots when using TCDetect in Figures 4.9 and 4.10, it is seen that only the row showing the composite TC of the cases detected only by TCDetect-TRACK changes considerably. This composite is now much more similar to the expected structure of a TC, albeit one which is not at hurricane-strength due to the width of the MSLP field and lack of a clear center in the wind speed field. This points to TCDetect-TRACK detecting systems which are valid meteorological systems but do not have the strength of a TC.

Given that T-TRACK produces overly long tracks, we would expect TCDetect-TRACK to produce TC centres at seemingly wrong positions as we would expect more TC centres to reach the extra-tropics. Figure 5.8 shows just this. There are more TC centres in the wrong place, i.e. over land, and more points at



(i) TCDetect　　　(ii) TCDetect-TRACK

Figure 5.5: Events reported by observations (IBTrACS) and detected by T-TRACK and (i) TCDetect and (ii) TCDetect-TRACK, applied to ERA-Interim data for (a) the whole globe, (b) the Northern Hemisphere and (c) the Southern Hemisphere.

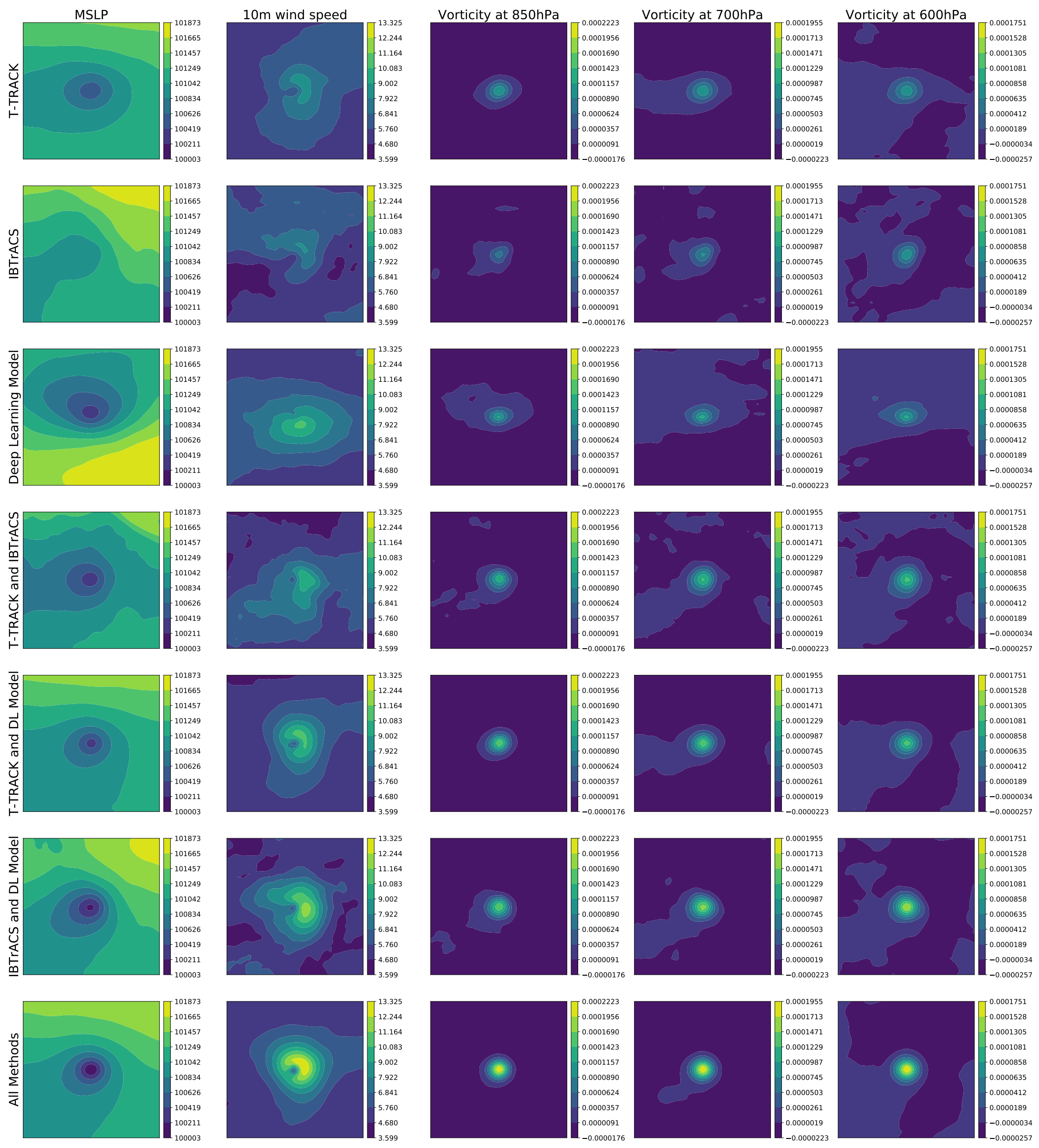


Figure 5.6: Composite view of Northern Hemisphere events by detection algorithm or observations which pick up the TC. Total number of cases used to produce each composite can be obtained from 5.5ii. Columns correspond to the variables used: MSLP (first column), 10-metre wind speed (second column), vorticity at 850hPa (third column), vorticity at 700hPa (fourth column) and vorticity at 600hPa (fifth column).

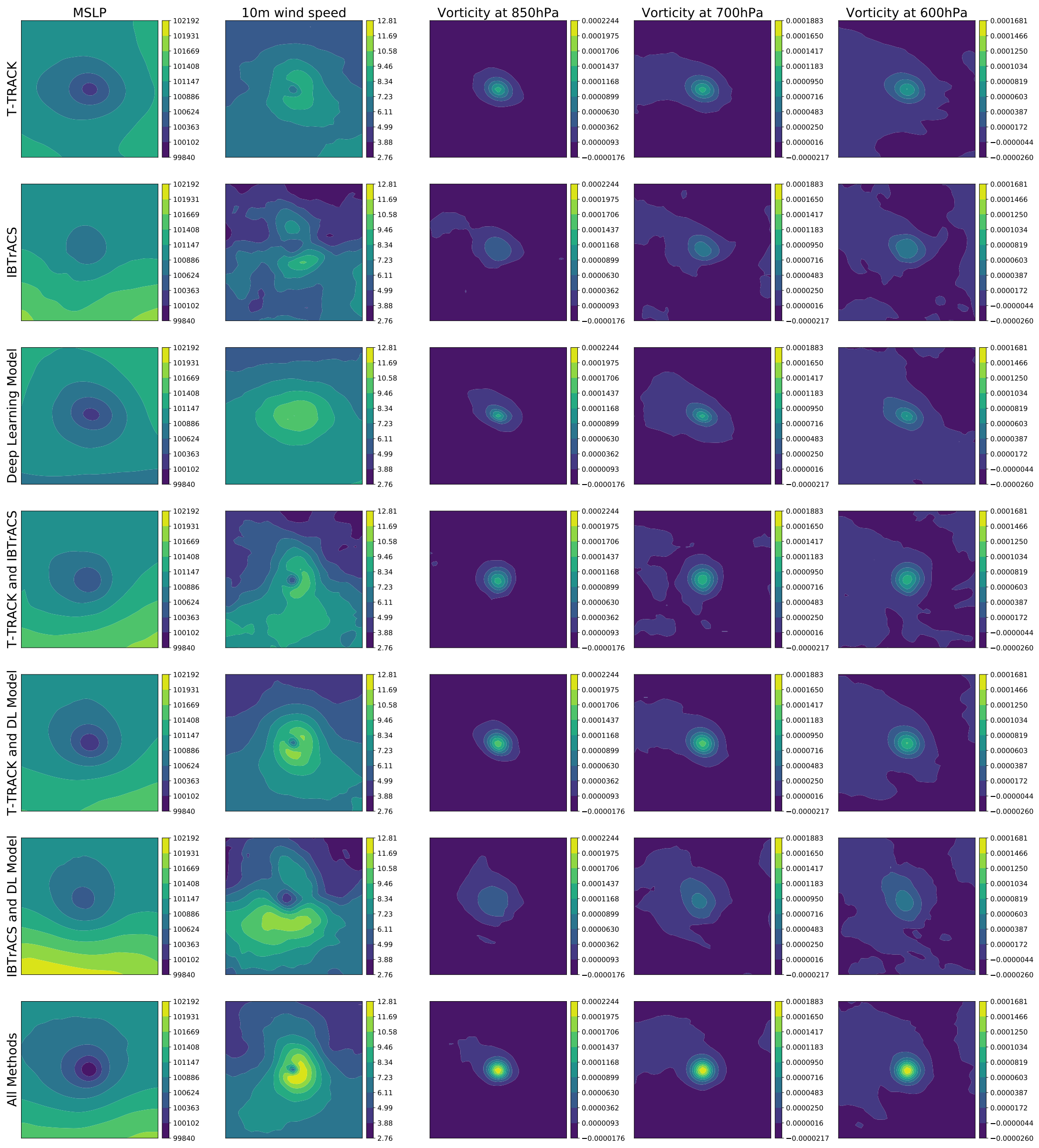


Figure 5.7: Composite view of the Southern Hemisphere cases (rows and columns as described in Figure 5.6) - but the sign of vorticity has been reversed for ease of comparison).

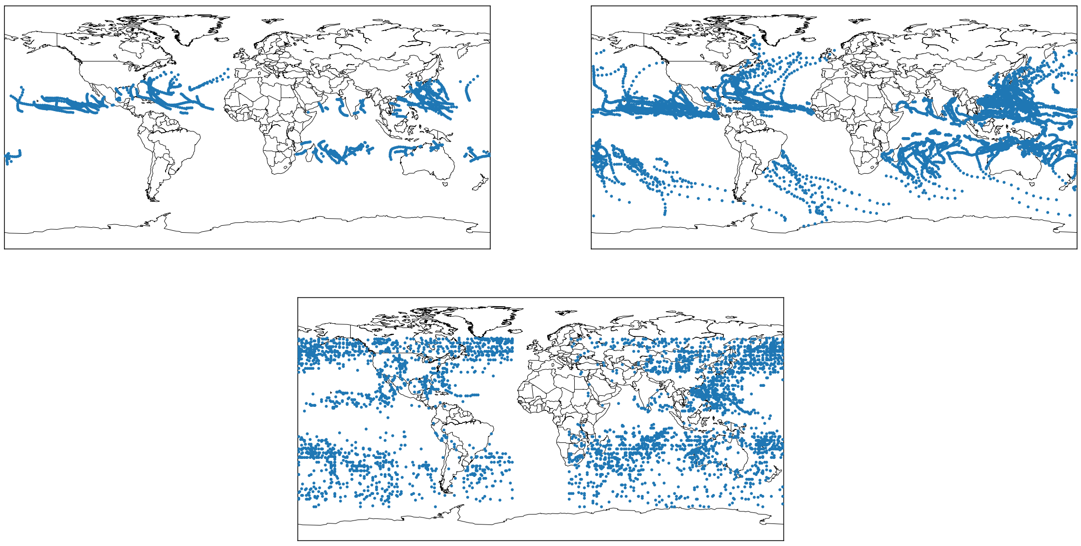


Figure 5.8: Position of each Tropical Cyclone event center as given by IBTrACS (top-left); T-TRACK (top-right) and the TCDetect-TRACK (bottom).

the extra-tropics, especially in the Northern Hemisphere. Also, more TC centres are generated over the Southern Hemisphere, as expected from the increase in regions in which TCDetect-TRACK detected a TC in the Southern Hemisphere. The latter behaviour also mirrors the T-TRACK labelling more closely - expected due to the change in labelling.

One complicating point is that only one TC centre can be produced in each region in which the deep learning model detects the presence of a TC. This is due to how the Grad-CAM technique (Selvaraju et al. (2017)) was used in this study. It was assumed that the the activation point which produced the highest value from Grad-CAM was to be assigned as a TC centre. This was as having Grad-CAM produce multiple TC centres would need a threshold above which any points would be a TC and thus this would introduce subjectivity. However, with the change of labelling, TCDetect-TRACK effectively changes its detection criteria to detect most tropical systems. This change would make the assumption to not necessarily hold as many cases exist where multiple tropical systems are present in the region, with Grad-CAM possibly assigning the highest activation to a region which is still a tropical system, but not a hurricane-strength TC.

This could be seen in the specific example shown in Figure 5.9. This shows a region in which two tropical systems can easily be identified in the region's data (top row) as areas of low values in the MSLP field and matching areas of high values in the other fields in the middle of the region. The area in the top half of the region is a tropical storm and the one in the bottom half is a Category 5 TC, however, both are present in the T-TRACK dataset. The first panel on the bottom row shows the output of Grad-CAM when using TCDetect with this region. The area of the highest activation (yellow area), and the TC position given by Grad-CAM (red dot) are loosely near the Category 5 TC. However, when doing this with TCDetect-TRACK in the second panel, the area of highest activation and the TC position given by Grad-CAM is nearer the top of the region, much closer to the tropical storm. Thus, a TC centre

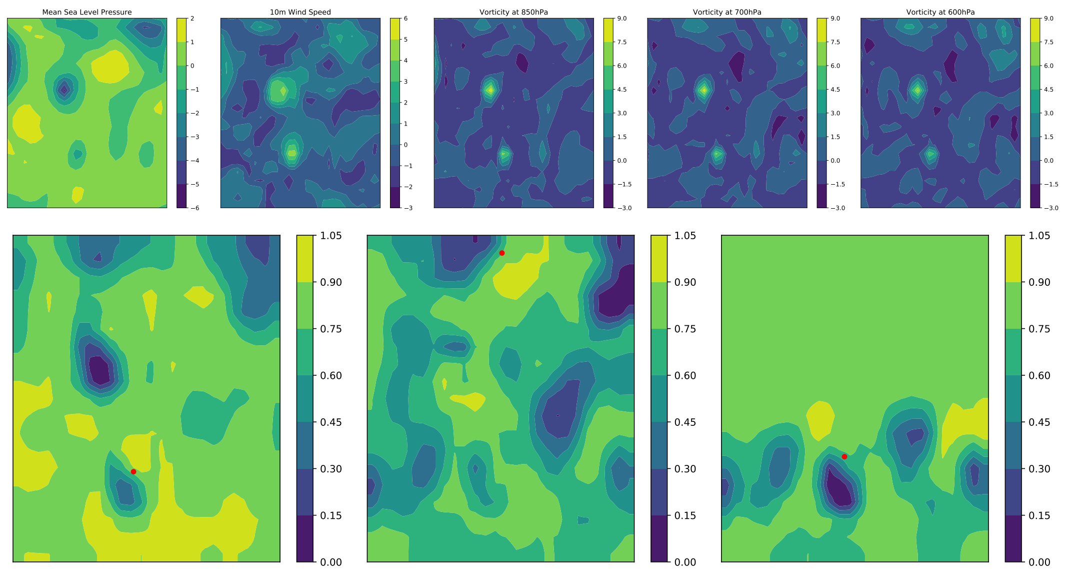


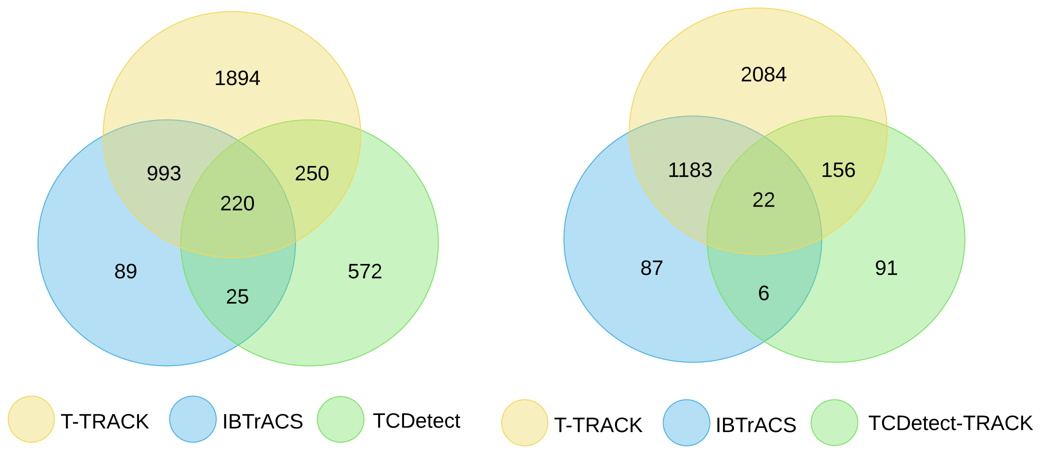
Figure 5.9: Example of region having a multiple tropical systems present, only one of which is a hurricane-strength TC (top row). Grad-CAM outputs (bottom row) for the example from TCDetect (left panel), TCDetect-TRACK (middle panel) and TCDetect-TRACK with top half of the example taken out (right panel). TC positions as given by Grad-CAM shown by red dots.

which is too Northerly is produced. To make sure that TCDetect-TRACK could still detect the Category 5 TC, the top half of the region was set to the mean of the field, effectively blanking out that part of the field, and Grad-CAM now puts the area of highest activation closer and the TC position generated close to the Category 5 TC as expected, shown the the third panel. This shows that TCDetect-TRACK is performing as expected, but the use of Grad-CAM is problematic.

These sub-optimal TC positions from TCDetect-TRACK should mean that while it may be detecting a tropical system in a region, matching it with TCs from the other two sources using the constraints in Section 4.2.1 becomes harder, mostly due to the constraint requiring the mean separation distance between the overlapping parts of two tracks to be within 5 degrees (geodesic).

Figure 5.10ii shows the Venn diagrams of matching TCs after the constraints detailed in Section 4.2.1 are applied. This shows a large reduction in the number of regions detected by TCDetect-TRACK which were matched to a TC from either T-TRACK or IBTrACS or both when compared to what was reported using TCDetect. This reduction is mostly down to the TC centres produced by Grad-CAM being in the wrong place. Despite this, unsurprisingly the largest number of matches is between TCDetect-TRACK and T-TRACK.

The analysis above shows that the effect of changing the labels from those derived from IBTrACS to those derived from T-TRACK is not insignificant, mainly due to these labels effectively changing what the models are being trained to do. As such, TCDetect-TRACK is not expected to perform as well as TCDetect, as the architecture was developed with IBTrACS labelling. Despite this, it is still skillful at detecting hurricane strength TCs, as they are a subset of tropical cyclones.



(i) TCDetect (ii) TCDetect-TRACK

Figure 5.10: Events detected by TRACK, (i) TCDetect and (ii) TCDetect-TRACK and/or reported by IBTrACS which fall on matching tracks, defined by applying constraints similar to those of Hodges, Cobb, and Vidale (2017).

5.3.3 Region Filtering

The method presented above was initially designed to act as a filtering mechanism to be used during a climate model run. As such, it is important to measure the amount of data filtered.

For this and the following sections, data from the UM runs detailed in Section 5.1 was processed to get it in a form that can be used to obtain the deep learning model's inference, with the process shown in Figure 5.11.

Data output from UM runs is formatted as .pp files with timesteps grouped monthly. These are then converted to netCDF files and the required fields - MSLP, 10-metre wind speed, and vorticity at 850hPA, 700hPa and 600hPa - are collected into their own netCDF files. TRACK and T-TRACK are run using these and output detected TCs along with a timestamp in a text file.

The netCDF files are also processed for use by the deep learning model, by resizing each timestep to the resolution used by ERA-Interim (≈79km). After, any spherical filtering is performed and each timestep's data is split into the eight regions shown in Figure 3.1. The data from each region is saved to disk as a .npz file, with the presence of a TC according to T-TRACK included in the file's path. The presence of a TC in the region is obtained by processing the TC centres obtained by T-TRACK and checking whether a TC was reported in the timestamp and region being processed.

The method was initially intended to be used to save data from whole timesteps based on whether a TC was detected by TCDetect. However, it was found that a TC was detected in the majority of the timesteps. This would have made the method less useful as the data reduction would have been minimal. T-TRACK also showed this, as seen in Table 5.5, especially for non-ERA-Interim data. As such, the method was changed to only output data from regions in which a TC was detected, so that regional climate modelling could use this data as boundary conditions for further analysis.

As shown in Figure 4.6 and discussed in Section 4.2.2, TCDetect overesti-

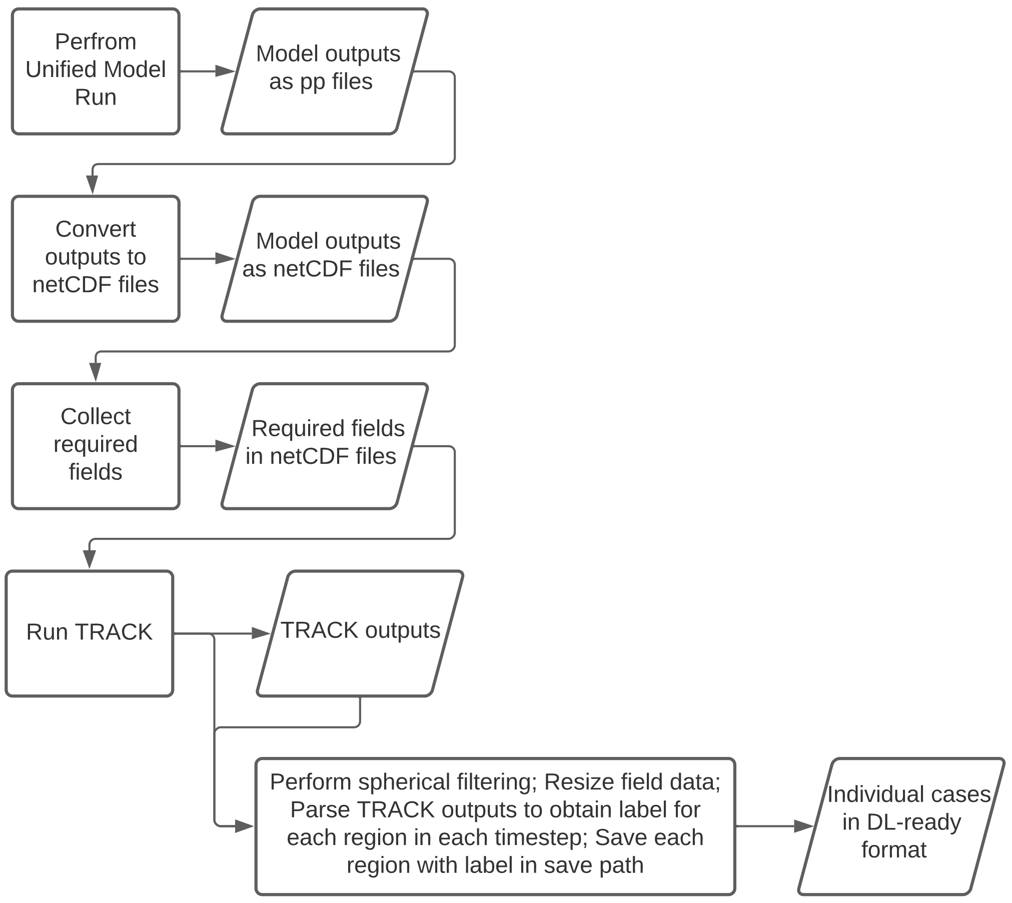

Perfrom Unified Model Run
Model outputs as pp files
Convert outputs to netCDF files
Model outputs as netCDF files
Collect required fields
Required fields in netCDF files
Run TRACK
TRACK outputs
Perform spherical filtering; Resize field data; Parse TRACK outputs to obtain label for each region in each timestep; Save each region with label in save path
Individual cases in DL-ready format

Figure 5.11: Data preparation workflow for producing training data for DL model from Unified Model outputs.

mates the number of TCs but is able to capture intra-annual variability when detecting TCs. We aim to use this behaviour to formulate the filtering method to only save high resolution data from regions which have a TC present.

Table 5.6 shows the number of instances of regions having a TC according to IBTrACS, T-TRACK and TCDetect for ERA-Interim data from the 1st July 2017 to 31st August 2019. As expected, both T-TRACK and TCDetect overestimate these instances with respect to IBTrACS. The latter observed a TC in 6% of all of these regions (3044 timesteps multipled by eight regions), while T-TRACK and TCDetect detected a TC in 19% and 14% of such regions. This means that if the filtering method was used there would be a reduction of 86% of the original data size.

A similar exercise was performed on data from the N96 simulation run of the UM, with results shown in Table 5.7. As IBTrACS data is not available for this dataset, it was not included. Also, to remove the effect of the deep learning model being tested on data which has a different horizontal resolution than that it is being tested on, the architecture of TCDetect was retrained on UM N96 data to obtain TCDetect-N96. When testing with this model, the results show that T-TRACK detected a TC in 5799 (17%) of 34560 regions, while TCDetect-N96 detected a TC in 9749 (28%) of the regions. The latter would represent a 72% reduction in the data saved to disk.

	IBTrACS	**T-TRACK**
ERA-Interim	36%	70%
UM N96	N/A	84%
UM N512	N/A	99%
Hist1950	N/A	97%
Future	N/A	98%

Table 5.5: Percentage of timesteps in which IBTrACS or T-TRACK detect a TC for each dataset.

	IBTrACS	**T-TRACK**	**TCDetect**
North Indian	72 (2%)	277 (9%)	239 (8%)
North Western Pacific	400 (13%)	1222 (40%)	933 (31%)
North Eastern Pacific	267 (9%)	712 (23%)	875 (29%)
North Atlantic	250 (8%)	703 (23%)	497 (16%)
South Indian	214 (7%)	646 (21%)	406 (13%)
South Western Pacific	113 (4%)	740 (24%)	399 (13%)
South Eastern Pacific	26 (1%)	322 (11%)	35 (1%)
South Atlantic	0 (0%)	119 (4%)	13 (0%)

Table 5.6: Number of instances of regions having a TC according to IBTrACS (first column), T-TRACK (second column) and TCDetect (third column) for ERA-Interim data from the 1st July 2017 to 31st August 2019. Each region has a total of 3044 timesteps and the percentage in brackets shows the percentage of the total number of timesteps which have a TC present.

Finally, the exercise was repeated on data from the N512 simulation run of the UM. Retraining of the original TCDetect architecture on this data was done, with TCDetect-N512 being obtained. Results of the number of regions in which a TC was detected by both T-TRACK and TCDetect-N512 are shown in Table 5.8. The results show that T-TRACK detected a TC in 2633 (23%) of 11480 regions, while TCDetect detected a TC in 3097 (27%) of the regions. The latter would represent a 73% reduction in the data saved to disk.

Therefore, the method as presented here would help to filter out the majority of data when data containing only TCs is required. For high resolution (N512) data, this is a 73% reduction. Given the large size of such high resolution data, this reduction represents a significant reduction in data.

	T-TRACK	**TCDetect-N96**
North Indian	74 (2%)	105 (2%)
North Western Pacific	1674 (39%)	2617 (61%)
North Eastern Pacific	442 (11%)	689 (16%)
North Atlantic	559 (13%)	999 (23%)
South Indian	913 (21%)	1596 (37%)
South Western Pacific	874 (20%)	1436 (33%)
South Eastern Pacific	1096 (25%)	1945 (45%)
South Atlantic	167 (4%)	362 (8%)

Table 5.7: Number of instances of regions having a TC according to T-TRACK (first column) and TCDetect (second column) for three simulated years of test data from the N96 simulation of the UM. Each region has a total of 4320 instances and the percentage in brackets shows the percentage of the total number of instances which have a TC present.

	T-TRACK	**TCDetect-N512**
North Indian	157 (11%)	197 (14%)
North Western Pacific	956 (67%)	920 (64%)
North Eastern Pacific	524 (37%)	714 (50%)
North Atlantic	431 (30%)	683 (48%)
South Indian	234 (16%)	552 (38%)
South Western Pacific	154 (11%)	374 (26%)
South Eastern Pacific	149 (10%)	370 (26%)
South Atlantic	28 (2%)	97 (7%)

Table 5.8: Number of instances of regions having a TC according to T-TRACK (first column) and TCDetect (second column) for the simulated year of test data from the N512 simulation of the UM. Each region has a total of 1435 instances and the percentage in brackets shows the percentage of the total number of instances which have a TC present.

5.4 Computational Performance

One important aspect of the method presented is that should be as lightweight as possible, i.e. it should not slow down the execution of the simulation excessively. Two seven-simulated-days runs at both N96 and N512 resolutions were carried out where the mean CPU time of the method as a whole and various parts of the method were timed, with results tabulated in Table 5.9. The model used a timestep length of 20 simulated minutes for the run at a resolution of N96, giving a total of 504 timesteps. The model run at a resolution of N512 used a timestep length of 10 simulated minutes, giving a total of 1008 timesteps.

The results obtained are put in a way that would show that the method would not be computationally viable if applied for every timestep, but would be so if applied to analysis data written out very number of timesteps, usually every six simulated hours.

In the N96 simulation, the simulation for a whole timestep took, on average, 6.15 seconds to compute, with 4.96 seconds used for the filtering method. This

		Mean CPU time (seconds)	
Function	**Times Applied**	**N96**	**N512**
Collect data on central pe	1	4.64×10^{-4}	2.99×10^{-3}
Interplate MSLP field to B-grid	1	3.15×10^{-4}	0.35
Resizing field to ERA-Interim resolution	5	9.1×10^{-3}	3.67×10^{-2}
Calculate vorticity	3	1.16×10^{-3}	2.28×10^{-3}
Perform spherical filtering on a field	5	0.68	1.36
Perform standardisation on a field	5	4.13×10^{-5}	6.64×10^{-5}
Prepare a region's data in DL format	8	1.63×10^{-5}	3.66×10^{-5}
Obtain DL inference for a region	8	0.19	0.37
Full data filtration method		4.96	10.22
Full Timestep		6.15	15.64

Table 5.9: Timings for UM runs including the filtering method at horizontal resolutions of N96 and N512.

shows that around 80% of the total timestep duration is taken up by the filtering method being presented. Similarly, in the N512 simulation, a whole timestep simulation takes, on average, 15.64 seconds, with 10.22 seconds, or around 65%, dedicated to the filtering method. In both cases, the filtering method is slowing down the simulation considerably.

Table 5.9 also shows a breakdown of execution times of various sections of the method. Only one section of the code, that interpolating the MSLP field from an Arakawa C-grid to an Arakawa B-grid, is majorly affected by the change in resolution. In the N96 resolution simulation, this took 3.15×10^{-4} seconds to complete, while it took 0.35 seconds to complete, an increase of more than 1000%, in the N512 resolution simulation. This is due to the larger amount of points that need to be interpolated.

Two sections of code, those that perform spherical filtering of a field and that obtains the inference from the DL model take up the majority of the time in both simulations. The former takes 0.68 seconds and 1.36 seconds in the N96 and N512 resolution simulations respectively. The latter takes 0.19 seconds and 0.37 seconds in the N96 and N512 resolution simulations respectively.

The code that performs the spherical filtering is called on each of the five fields while that which obtains the inference from the deep learning model is called on each of the eight regions shown in Figure 3.1. Currently, these are done serially, hence a large chunk of the time required for the whole timestep, is taken up by these two computationally expensive sections of code. On the other hand, had these been done in parallel and assuming ideal speedup, the filtering method would take 0.91 seconds for the N96 resolution simulation and 2.19 seconds for the N512 resolution simulation. These represent a reduction of around 80% for the time required to execute the filtering method. Also, had the method been executed in such a way that an inference from each method is obtained at the same time, i.e. utilising many CPUs to run in parallel, it would take around 43% of a timestep's execution time (neglecting any time needed to scatter the data to each required CPU, but this should be at the same order of collecting the data on the central CPU, which is insignificant) in the N96 resolution simulation and around 29% of a timestep's execution in the N512 resolution simulation, resulting in a much more lightweight filtering method.

All of this would mean that the method being presented is not lightweight if applied at each model timestep as it slows the execution of the simulation down considerably. However, it might be more acceptable at even higher resolutions than currently tested. The cost of the method has been shown to decrease from 43% of the total timestep execution time at N96 resolution to 29% at N512. Hence, we would expect this to decrease further, mainly due to a timestep taking longer to compute at higher resolutions. Also, the importance of the method would increase with a higher resolution simulation as more data would be produced.

However, when applying this method to analysis data which is only outputted every certain amount of timesteps, the computation cost decreases. Assuming that our method is to applied every six simulated hours and in the N96 simulation, the timestep used by the model is that of 20 simulated minues, our method would be used every 18 timesteps, so it would add 4.96 seconds of computational time for every 21.42 seconds of runtime, representing a 23% slowdown in computation. The N512 simulation uses a timestep of 10 simulated minutes, so our method would add 10.22 seconds per 195 seconds of runtime, slowing down the computation by around 5%.

These show that the method if applied in this way does not slow down the computation of the simulation significantly, especially for those run at higher resolution.

5.5 Chapter Summary

TCDetect, a deep learning model developed to detect the presence of a Tropical Cyclone in meteorological data, was incorporated into a fully-fledged method that could be used by the Met Office's Unified Model (UM) to decide whether to save analysis data to disk depending on the inferred presence of a Tropical Cyclone in a certain region.

The amended UM algorithm to include this method was detailed and shown to require minimal changes to the original UM. This included how a C++ intermediate layer was used to load the deep learning model trained in Python to be used by the FORTRAN-based UM.

It was shown that both TCDetect and T-TRACK, a state of the art detection and tracking algorithm, detected a TC in most analysis timesteps. This would have made the reduction in data negligible. Hence, the method was changed to only save data from a certain region of the globe when a TC was detected in that region. This data would serve as boundary conditions for regional climate modelling which is performed to study the adaptation of the Earth system to a changing climate. The method as now intended reduces the data saved to disk considerably.

The adaptability of the method to operate on data with different sources was also discussed. Multiple sources of data were used for this but having only IBTrACS labelling available for one, the change of labelling from IBTrACS to T-TRACK was shown to be significant. Given that T-TRACK still contains various types of tropical systems, not just hurricane-force systems, the performance of a variant of TCDetect trained with this labelling was shown to be less than that of TCDetect, possibly due to the change of labelling effectively changing what the deep learning model is being asked to detect. Despite this, it was shown that a deep learning model trained on UM data could be applied on ERA-Interim data, but not vice-versa. It was also shown that a model trained on the data that reflected the testing data performed best or second best. It was also found that a model can be applied to data of a higher resolution, albeit with a slight drop in performance, but not vice-versa. It was finally shown that a model trained on data from the current climate an be applied to data from a future climate, at least when TCs are considered.

Finally, the computational expense of the method was discussed. It was shown that the method applied to the output of analysis data, which only happens after every period of simulated time, slows down the computation of the simulation by 23% in a N96 simulation using the UM, and only by 5% in a N512 simulation. Given that this method is more likely to be applied to higher resolution data, i.e. to the N512 simulation, the computational cost of the method is sufficient for it to be considered a lightweight method.

Chapter 6

Conclusions

It is known that current climates models produce large amounts of data. As higher resolutions are used, the volume of this data starts to increase greatly. Such volumes of data are difficult to store and even more difficult to analyse, especially if focusing on only one or a few phenomena.

Hence, a data filtration method to filter out any unwanted data was created and is being presented in this study, with hurricane-strength Tropical Cyclones (TCs) being targeted to show whether this method works. One novel aspect that we wanted this method to have is that it could be used during a climate simulation to decide at the time of writing out data whether to write it out, rather than the current method of writing out all of the data and removing any unnecessary data afterwards.

For this method, some way of detecting the presence of TCs in the data needed to be found. There are multiple algorithms already developed that do this, but most would either not be able to perform their function on single timesteps of data, or would take too long to return a decision, so the runtime of the climate model would be impacted.

Hence, to make the method as lightweight as possible, it was decided that a deep learning model, trained to detected the presence of a TC in model data, would be developed for this purpose. The model developed was termed TCDetect.

The first part of the study goes into how TCDetect was created and evaluated. ERA-Interim data and IBTrACS-derived labels were used to train and evaluate the deep learning model. It obtained a recall rate, i.e. its ability of detecting TCs, of 92% and a precision rate, i.e. its ability to not create false positives, of 36%. This shows that it has a high skill in detecting the presence of TCs but could do better in identifying data in which no TC is present.

Various aspects of TCDetect were also investigated to understand of the inferences it was making. First, the most important feature to the model was found to be vorticity at 850hPa. This was expected as all TCs manually checked had a clear signal in that field. Also, many of the classical algorithms use that field for their purpose. Next, it was shown that TCDetect worked similarly in all regions of the globe, which is important if it was to be used in a method covering the globe. It was also shown that TCDetect picked out all Category 5 TCs in the test dataset with more inferior performance for weaker TCs. This was expected as Category 5 TCs are the most well formed and easily recognisable systems, with weaker systems not being so clear to spot. It was also shown that most of the false positives given by TCDetect had meteorological systems present, showing that it might have had some problem in distinguishing hurricane-strength TCs from non-hurricane ones. However, it was shown that the location of the TCs given by TCDetect, via the Grad-CAM technique, was not very accurate. This was expected TCDetect was not trained for this function. Finally, TCDetect was compared to other more standard deep learning models to show that it performed better than them and that it was worthwhile developing a new model for the specific task at hand.

The study continued by comparing TCDetect to a state-of-the-art TC detection and tracking algorithm, T-TRACK, and an observational dataset, IBTrACS. It was shown that TCDetect picks up most of the TCs also observed by IBTrACS and detected by T-TRACK. Also, T-TRACK and TCDetect agreed on a considerable amount of others, and together with the understanding that IBTrACS is not a perfect observational dataset, it was concluded that these were possible TCs that were missed by IBTrACS. The tracks of the TCs detected by both algorithms and seen in the observations were compared and it was shown that TCDetect matched the majority of these tracks.

Composite TCs were created for all types of matches between the two algorithms and IBTrACS. It was shown that the strongest TCs, in terms of maximum wind speeds, minimum mean sea level pressures or maximum vorticity, were found by all of IBTrACS, T-TRACK and TCDetect. Slightly weaker ones generated matches by two of the two methods or IBTrACS, while the weakest TCs were only found by one of the methods or the observational dataset.

The positions of the TC centres as given by both algorithms and IBTrACS were also investigated and it was shown that TCDetect mostly produced TC centres in the expected areas of the globe. The relative positions of TC centres were also compared and most TC centres given by TCDetect were within 2.5 degrees (geodesic) of their respective one as given by T-TRACK and/or IBTrACS.

The final part of the study put together the final data filtration method and showed how it could be used in a running simulation by the Met Office's Unified Model (MetUM). The original high-level algorithm currently used by the MetUM and how it was changed to include the method being presented was detailed. This included how the TCDetect, trained in Python, was used in the FORTRAN codebase using a C++ layer. The amount of data reduction was calculated for multiple scenarios.

The adaptability of the method to operate on data with different sources was also discussed. It was shown that a deep learning model trained on UM data could be applied on ERA-Interim data, but not vice-versa. It was also shown that a model trained on the data that reflected the testing data performed best or second best. It was also found that a model can be applied to data of a higher resolution, albeit with a slight drop in performance, but not vice-versa. It was finally shown that a model trained on data from the current climate an be applied to data from a future climate, at least when TCs are considered.

While the study produced encouraging results for this type of method to be used, some improvements could have been made. Some are:

- Use different data: The input data used was obtained from ERA-Interim.

ERA5, which represents TCs better, would have possibly been a better choice, but it was only available after the start of the study and it would have been too time-consuming to re-run all of the hyperparameter tuning on the new dataset. Also, only five fields were used in creating the deep learning model. While an informed decision was made on this choice from those fields used for the classical TC detection and tracking algorithms, other different fields might have improved the deep learning model's performance. One such field is temperature. This was used in most non-ML TC tracking algorithms as it helps out filter out extra-tropical cyclones due to a difference in temperature in the centres of TCs and extra-tropical systems.

- Automatic Hyperparameter Tuning: Manual hyperparameter tuning was utilised in this study, mainly because of the time and computational needs for developing the expertise needed for this, but it would have almost certainly produced a better performing deep learning model.
- Using a different deep learning architecture: The one presented was created in early stages of the study when deep learning expertise was still being developed. A better architecture would have possibly been to utilise a U-Net, where the input data is attempted to be compressed and restored, and then use the bottleneck layer of the architecture to produce an inference on the presence of a TC.
- Specify the problem differently: Again, due to a lack of deep learning expertise at the start of the study, the problem was specified as detecting the presence of a TC in a region of the globe. A better problem specification would have been to locate all TCs in a whole timestep, similar to other previously done work. This would have made the method even more lightweight and the method could have also generated some TC statistics, which are usually generated after a climate simulation has been completed. Meteorological verification of the model would also have been easier to do as Grad-CAM would not have been needed. Another avenue to explore would be to change the model to detect all tropical systems. This would have made the model easier to compare to TRACK.
- Optimise the filtering method: The presented method is executed wholly in series. However, it was shown that some parts, namely those obtaining the inference from the deep learning model, could be executed in parallel to lessen the time needed for the method to execute. Also, the current method first interpolates the data from the original model's resolution to that of ERA-Interim before performing spherical filtering. This was done to test that the FORTRAN code was obtaining the right data manipulations, but the interpolating step is redundant if the spherical filtering is performed on the initial data. This will also have the effect of lessening the method's execution time.

Some unanswered questions also remain. Some of these include:

- How does TCDetect compare to other algorithms besides T-TRACK? Can any differences be explained to inform a better choice of inputs fields?

- Can the performance of the method be improved by using a multi-step process? Given that most Tropical Cyclones start their lifetime in the Tropics, could a two-step process be used where any detection is performed on the Tropics only and some time restrictions are then applied to any other TCs detected in the extra-tropics. This should reduce the number of false positives generated.

- Can a better indication of the performance of the method for different climates be obtained? Currently, only one ensemble member is used to come to the conclusions made, but using more ensemble members would increase the size of the testing dataset, making any conclusions more significant.

Finally, while some future avenues to extend this study exist, the obvious one would be to apply the method to detect different types of phenomena and check whether similar performance can be obtained. This would add credence to the method being present and show that it is not only applicable to TCs. Also, the deep learning model underpinning the method could be improved to obtain better performance.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

[2] John C. Adams and Paul N. Swarztrauber. "SPHEREPACK 3.0: A Model Development Facility". In: *Monthly Weather Review* 127.8 (1999), pp. 1872–1878. DOI: `10.1175/1520-0493(1999)127<1872:SAMDF>2.0.CO;2`. URL: `https://journals.ametsoc.org/view/journals/mwre/127/8/1520-0493_1999_127_1872_samdf_2.0.co_2.xml`.

[3] L. Bengtsson, K. I. Hodges, and M. Esch. "Tropical cyclones in a T159 resolution global climate model: comparison with observations and re-analyses". In: *Tellus A: Dynamic Meteorology and Oceanography* 59.4 (2007), pp. 396–416. DOI: `10.1111/j.1600-0870.2007.00236.x`. eprint: `https://doi.org/10.1111/j.1600-0870.2007.00236.x`. URL: `https://doi.org/10.1111/j.1600-0870.2007.00236.x`.

[4] Bernhard Boser, Isabelle Guyon, and Vladimir Vapnik. "A Training Algorithm for Optimal Margin Classifier". In: *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory* 5 (Aug. 1996). DOI: `10.1145/130385.130401`.

[5] Leo Breiman. "Random forests". In: *Mach. Learn.* 45.1 (2001), pp. 5–32. ISSN: 08856125. DOI: `10.1023/A:1010933404324`. URL: `https://link.springer.com/article/10.1023/A:1010933404324`.

[6] Suzana J. Camargo and Stephen E. Zebiak. "Improving the detection and tracking of tropical cyclones in atmospheric general circulation models". In: *Weather Forecast.* 17.6 (2002), pp. 1152–1162. ISSN: 08828156. DOI: `10.1175/1520-0434(2002)017<1152:ITDATO>2.0.CO;2`.

[7] John P. Cangialosi, Andrew S. Latto, and Robbie Berg. *Hurricane Irma (AL112017).* Tech. rep. `https://www.scirp.org/(S(351jmbntvnsjt1aadkposzje))/reference/ReferencesPapers.aspx?ReferenceID=2635550`, as accessed on 23/03/2021. National Oceanic and Atmospheric Administration (NOAA), 2018, p. 111.

[8] Met Office Hadley Centre. "WCRP CMIP6: Met Office Hadley Centre (MOHC) HadGEM3-GC31-HH model output for the "highres-future" experiment". In: ().

[9] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*. Vol. 2017-January. Institute of Electrical and Electronics Engineers Inc., 2017, pp. 1800–1807. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.195. arXiv: 1610.02357. URL: https://arxiv.org/abs/1610.02357v3.

[10] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. arXiv: 1511.07289 [cs.LG].

[11] Alexis Conneau et al. "Very deep convolutional networks for text classification". In: *arXiv preprint arXiv:1606.01781* (2016).

[12] D. P. Dee et al. "The ERA-Interim reanalysis: configuration and performance of the data assimilation system". In: *Q. J. R. Meteorol. Soc.* 137.656 (2011), pp. 553–597. ISSN: 00359009. DOI: 10.1002/qj.828. URL: http://doi.wiley.com/10.1002/qj.828.

[13] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[14] Timothy Dozat. "Incorporating Nesterov Momentum into Adam". In: 2016.

[15] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

[16] V. Eyring et al. "Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization". In: *Geoscientific Model Development* 9.5 (2016), pp. 1937–1958. DOI: 10.5194/gmd-9-1937-2016. URL: https://gmd.copernicus.org/articles/9/1937/2016/.

[17] C.F. Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. Carl Friedrich Gauss Werke. Hamburgi sumptibus Frid. Perthes et I.H.Besser, 1809. URL: https://books.google.com.mt/books?id=ORUOAAAAQAAJ.

[18] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[20] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* Vol. 2016-December. IEEE Computer Society, 2016, pp. 770–778. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385. URL: http://image-net.org/challenges/LSVRC/2015/.

[21] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV).* 2015.

[22] Kaiming He et al. “Identity mappings in deep residual networks”. In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics).* Vol. 9908 LNCS. Springer Verlag, 2016, pp. 630–645. ISBN: 9783319464923. DOI: 10.1007/978-3-319-46493-0_38. arXiv: 1603.05027. URL: http://arxiv.org/abs/1603.05027.

[23] Tobias Hermann. *frugally-deep.* https://github.com/Dobiasd/frugally-deep. 2020.

[24] K. I. Hodges. “Adaptive Constraints for Feature Tracking”. In: *Monthly Weather Review* 127.6 (1999), pp. 1362–1373. DOI: 10.1175/1520-0493(1999)127<1362:ACFFT>2.0.CO;2. eprint: https://doi.org/10.1175/1520-0493(1999)127<1362:ACFFT>2.0.CO;2. URL: https://doi.org/10.1175/1520-0493(1999)127<1362:ACFFT>2.0.CO;2.

[25] K. I. Hodges. “Feature Tracking on the Unit Sphere”. In: *Monthly Weather Review* 123.12 (1995), pp. 3458–3465. DOI: 10.1175/1520-0493(1995)123<3458:FTOTUS>2.0.CO;2. eprint: https://doi.org/10.1175/1520-0493(1995)123<3458:FTOTUS>2.0.CO;2. URL: https://doi.org/10.1175/1520-0493(1995)123<3458:FTOTUS>2.0.CO;2.

[26] K. I. Hodges. “Spherical Nonparametric Estimators Applied to the UGAMP Model Integration for AMIP”. In: *Monthly Weather Review* 124.12 (1996), pp. 2914–2932. DOI: 10.1175/1520-0493(1996)124<2914:SNEATT>2.0.CO;2. eprint: https://doi.org/10.1175/1520-0493(1996)124<2914:SNEATT>2.0.CO;2. URL: https://doi.org/10.1175/1520-0493(1996)124<2914:SNEATT>2.0.CO;2.

[27] Kevin Hodges, Alison Cobb, and Pier Luigi Vidale. “How Well Are Tropical Cyclones Represented in Reanalysis Datasets?” In: *Journal of Climate* 30.14 (2017), pp. 5243 –5264. DOI: 10.1175/JCLI-D-16-0557.1. URL: https://journals.ametsoc.org/view/journals/clim/30/14/jcli-d-16-0557.1.xml.

[28] M Horn, K Walsh, and A Ballinger. *Detection of tropical cyclones using a phenomenon-based cyclone tracking scheme.* Tech. rep. US CLIVAR, 2013. URL: https://usclivar.org/sites/default/files/documents/2014/2013HurricaneReportFinal.v3_0_1.pdf.

[29] Michael Horn et al. “Tracking Scheme Dependence of Simulated Tropical Cyclone Response to Idealized Climate Simulations”. In: *Journal of Climate* 27.24 (2014), pp. 9197 –9213. DOI: 10.1175/JCLI-D-14-00200.1. URL: https://journals.ametsoc.org/view/journals/clim/27/24/jcli-d-14-00200.1.xml.

[30] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[31] Gao Huang et al. "Densely Connected Convolutional Networks". In: *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017* 2017-January (2016), pp. 2261–2269. arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[32] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[33] Günter Klambauer et al. "Self-Normalizing Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/5d44ee6f2c3f71b73125876103c8f6c4-Paper.pdf.

[34] S. Kleppek et al. "Tropical cyclones in ERA-40: A detection and tracking method". In: *Geophys. Res. Lett.* 35.10 (2008). ISSN: 00948276. DOI: 10.1029/2008GL033880.

[35] Kenneth R. Knapp et al. *International Best Track Archive for Climate Stewardship (IBTrACS) Project, Version 4*. 2018. URL: https://data.nodc.noaa.gov/cgi-bin/iso?id=gov.noaa.ncdc:C01552 (visited on 12/04/2019).

[36] Kenneth R. Knapp et al. "The international best track archive for climate stewardship (IBTrACS)". In: *Bull. Am. Meteorol. Soc.* 91.3 (2010), pp. 363–376. ISSN: 00030007. DOI: 10.1175/2009BAMS2755.1.

[37] Christina Kumler-Bonfanti et al. "Tropical and Extratropical Cyclone Detection Using Deep Learning". In: *Journal of Applied Meteorology and Climatology* 59 (Dec. 2020), pp. 1971–1985. DOI: 10.1175/JAMC-D-20-0117.1.

[38] Thorsten Kurth et al. "Deep Learning at 15PF: Supervised and Semi-Supervised Classification for Scientific Data". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. ISBN: 9781450351140. DOI: 10.1145/3126908.3126916. URL: https://doi.org/10.1145/3126908.3126916.

[39] Valliappa Lakshmanan et al. "Which polarimetric variables are important for weather/no-weather discrimination?" In: *J. Atmos. Ocean. Technol.* 32.6 (2015), pp. 1209–1223. ISSN: 15200426. DOI: 10.1175/JTECH-D-13-00205.1. URL: http://journals.ametsoc.org/jtech/article-pdf/32/6/1209/3376431/jtech-d-13-00205{_}1.pdf.

[40] Bryan Lawrence et al. "The JASMIN super-data-cluster". In: *ArXiv* abs/1204.3553 (2012).

[41] Yann A LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[42] AM Legendre. *New methods for the determination of the orbits of comets [microform] / by AM Legendre*. French. F. Didot Paris, 1805, viii, 80 p., [1] leaf of plates:

[43] Y. Liu et al. "Application of Deep Convolutional Neural Networks for Detecting Extreme Weather in Climate Datasets". In: *ArXiv* abs/1605.01156 (2016).

[44] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[45] Warren Mcculloch and Walter Pitts. "A Logical Calculus of Ideas Immanent in Nervous Activity". In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147.

[46] Met Office Hadley Centre. "WCRP CMIP6: Met Office Hadley Centre (MOHC) HadGEM3-GC31-HH model output for the "hist-1950" experiment". In: (2020). URL: `https://catalogue.ceda.ac.uk/uuid/56fd51e7a5854128bffb82302fb6e513` (visited on 11/23/2021).

[47] Ryo Mizuta et al. "Climate Simulations Using MRI-AGCM3.2 with 20-km Grid". In: *J. Meteor. Soc. Japan* 90A (2012), pp. 233–258. DOI: `10.2151/jmsj.2012-A12`.

[48] Mayur Mudigonda et al. "Segmenting and Tracking Extreme Climate Events using Neural Networks". In: *31st Conf. Neural Inf. Process. Syst.* (2017), pp. 1–5. URL: `https://dl4physicalsciences.github.io/files/nips{\_}dlps{\_}2017{\_}20.pdf`.

[49] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, 807–814. ISBN: 9781605589077.

[50] Y. E. Nesterov. "A method for solving the convex programming problem with convergence rate $O(1/k^2)$". In: *Dokl. Akad. Nauk SSSR* 269 (1983), pp. 543–547. URL: `https://ci.nii.ac.jp/naid/10029946121/en/`.

[51] Kazutoshi Onogi et al. "The JRA-25 Reanalysis". In: *Journal of the Meteorological Society of Japan. Ser. II* 85.3 (2007), pp. 369–432. DOI: `10.2151/jmsj.85.369`.

[52] N. Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66.

[53] Norman A. Phillips. "The general circulation of the atmosphere: A numerical experiment". In: *Quarterly Journal of the Royal Meteorological Society* 82.352 (1956), pp. 123–164. DOI: `https://doi.org/10.1002/qj.49708235202`. eprint: `https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/qj.49708235202`. URL: `https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49708235202`.

[54] Prabhat et al. "TECA: A parallel toolkit for extreme climate analysis". In: *Procedia Comput. Sci.* Vol. 9. Elsevier B.V., 2012, pp. 866–876. DOI: `10.1016/j.procs.2012.04.093`.

[55] Prabhat et al. "TECA: Petascale Pattern Recognition for Climate Science". In: *Computer Analysis of Images and Patterns*. Ed. by George Azzopardi and Nicolai Petkov. Cham: Springer International Publishing, 2015, pp. 426–436. ISBN: 978-3-319-23117-4.

[56] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1 (1999), pp. 145–151. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/S0893-6080(98)00116-6`. URL: `https://www.sciencedirect.com/science/article/pii/S0893608098001166`.

[57] Evan Racah et al. "ExtremeWeather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events". In: *NIPS*. 2017.

[58] Lewis F. Richardson. *Weather Prediction by Numerical Process*. Cambridge University Press, 1922.

[59] Malcolm J Roberts et al. "Tropical Cyclones in the UPSCALE Ensemble of High-Resolution Global Climate Models". In: *J. Clim.* 28.2 (2015), pp. 574–596. DOI: `10.1175/JCLI-D-14-00131.1`. URL: `https://doi.org/10.1175/JCLI-D-14-00131.1`.

[60] E Roeckner et al. "The atmospheric general circulation model ECHAM 5. PART I: model description". In: *Max Planck Institute for Meteorology Report* 349 (Jan. 2003).

[61] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[62] S. Saha. "The NCEP Climate Forecast System version 2". In: *J. Climate* 27 (2014), pp. 2185–2208.

[63] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* IEEE Computer Society, 2018, pp. 4510–4520. ISBN: 9781538664209. DOI: `10.1109/CVPR.2018.00474`. arXiv: `1801.04381`. URL: `http://arxiv.org/abs/1801.04381`.

[64] Benjamin A. Schenkel and Robert E. Hart. "An Examination of Tropical Cyclone Position, Intensity, and Intensity Life Cycle within Atmospheric Reanalysis Datasets". In: *Journal of Climate* 25.10 (2012), pp. 3453 –3475. DOI: `10.1175/2011JCLI4208.1`. URL: `https://journals.ametsoc.org/view/journals/clim/25/10/2011jcli4208.1.xml`.

[65] G. A. Schmidt. "Configuration and assessment of the GISS ModelE2 contributions to the CMIP5 archive". In: *J. Adv. Model. Earth Syst.* 6 (2014), pp. 141–184.

[66] R. R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 618–626. DOI: `10.1109/ICCV.2017.74`.

[67] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv 1409.1556* (2014).

[68] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[69] Jane Strachan et al. “Investigating Global Tropical Cyclone Activity with a Hierarchy of AGCMs: The Role of Model Resolution”. In: *Journal of Climate* 26.1 (2013), pp. 133 –152. DOI: 10.1175/JCLI-D-12-00012.1. URL: https://journals.ametsoc.org/view/journals/clim/26/1/jcli-d-12-00012.1.xml.

[70] Richard S. Sutton. “Two Problems with Backpropagation and Other Steepest-Descent Learning Procedures for Networks”. In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum, 1986.

[71] Christian Szegedy et al. “Inception-v4, inception-ResNet and the impact of residual connections on learning”. In: *31st AAAI Conf. Artif. Intell. AAAI 2017.* AAAI press, 2017, pp. 4278–4284. arXiv: 1602.07261. URL: https://arxiv.org/abs/1602.07261v2.

[72] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* Vol. 2016-December. IEEE Computer Society, 2016, pp. 2818–2826. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.308. arXiv: 1512.00567. URL: https://arxiv.org/abs/1512.00567v3.

[73] P. A. Ullrich and C. M. Zarzycki. “TempestExtremes: a framework for scale-insensitive pointwise feature tracking on unstructured grids”. In: *Geoscientific Model Development* 10.3 (2017), pp. 1069–1090. DOI: 10.5194/gmd-10-1069-2017. URL: https://gmd.copernicus.org/articles/10/1069/2017/.

[74] S. M. Uppala et al. “The ERA-40 re-analysis”. In: *Quarterly Journal of the Royal Meteorological Society* 131.612 (2005), pp. 2961–3012. DOI: https://doi.org/10.1256/qj.04.176. eprint: https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1256/qj.04.176. URL: https://rmets.onlinelibrary.wiley.com/doi/abs/10.1256/qj.04.176.

[75] F. Vitart, J. L. Anderson, and W. F. Stern. “Simulation of Interannual Variability of Tropical Storm Frequency in an Ensemble of GCM Integrations”. In: *Journal of Climate* 10.4 (1997), pp. 745–760. DOI: 10.1175/1520-0442(1997)010<0745:SOIVOT>2.0.CO;2. eprint: https://doi.org/10.1175/1520-0442(1997)010<0745:SOIVOT>2.0.CO;2. URL: https://doi.org/10.1175/1520-0442(1997)010<0745:SOIVOT>2.0.CO;2.

[76] K. Walsh et al. “Objectively determined resolution-dependent threshold criteria for the detection of tropical cyclones in climate models and reanalyses”. In: *J. Climate* 20 (2007), pp. 2307–2314.

[77] Colin M. Zarzycki and Paul A. Ullrich. “Assessing sensitivities in algorithmic detection of tropical cyclones in climate data”. In: *Geophysical Research Letters* 44.2 (2017), pp. 1141–1149. DOI: `https://doi.org/10.1002/2016GL071606`. eprint: `https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1002/2016GL071606`. URL: `https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2016GL071606`.

[78] Ming Zhao et al. “Simulations of Global Hurricane Climatology, Interannual Variability, and Response to Global Warming Using a 50-km Resolution GCM”. In: *Journal of Climate* 22.24 (2009), pp. 6653 –6678. DOI: `10.1175/2009JCLI3049.1`. URL: `https://journals.ametsoc.org/view/journals/clim/22/24/2009jcli3049.1.xml`.

Appendix A

The `frugally-deep` package

As explained in the Chapter 5, the package `frugally-deep` is used to be able to use the deep learning model trained in Python in the FORTRAN-based UM. In this appendix, we will attempt to show the inner workings for this package and how it was used in the UM implementation discussed in Chapter 5.

The package is a header-only library written in C++ except for a script which saves the trained model to disk which is written in Python. The aim of this package is to use Tensorflow trained deep learning models in C or C++ code, without having to configure Tensorflow on the system being used. As such, `frugally-deep` can only produce inferences using the trained model.

First, an important Python script is included in the package which converts and saves the trained model to disk in a readable format for `frugally-deep`. It requires a trained model that has been saved to an HDF5 file via Tensorflow's `model.save()` routine. This model is loaded by the converter and the converted model is saved as a as a JavaScript Object Notation (JSON) file which has six entries. The first is the architecture, i.e. the layer types and other hyperparameters, of the model in JSON format. The second entry is the image data format, i.e. whether the data that will be put into the model will have the channels index as the first or the last dimension of any array used. The next two entries are the expected input and output array shapes. The penultimate entry contains all of the model's weights in ASCII format and the final entry has the model's hash stored.

In C++ code, `frugally-deep` loads the model via its own `model.load_model()` routine. This uses the json dependency to read in the json file and constructs the model by loading the weights and architecture into vectors. This C++ code is called from the UM's FORTRAN code and returns a pointer to the loaded deep learning model to be stored and used in the FORTRAN code.

After this, the data needed to produce the inference is constructed. The preprocessed data is obtained from the UM and is given as a three dimensional array. This array is passed onto a C++ routine and used to construct `frugally-deep` tensors. These tensors are then passed onto the constructed model for an inference to be made via `frugally-deep`'s `model.predict()` which return another `frugally-deep` tensor. The inference is extracted from this tensor and returned to the UM's FORTRAN code, where it is handled and processed to get the needed decision on whether to save data to disk or not.