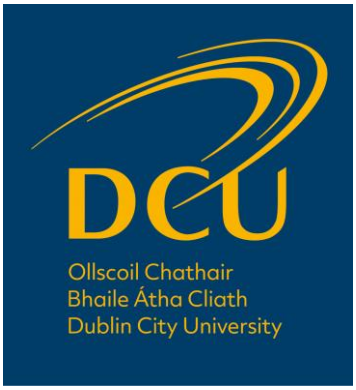



Final Year Project Report

# Shallow Learning for Meteor Identification from Images

Name: Fiachra Feehilly

Student No.: 18330276

Class: PHA4

Date: 29th March 2022

Supervisor: Dr. Jonathan Mackey, Dr. Masha Chernyakova

**Declaration**

| | |
|---|---|
| Name: | Fiachra Feehilly |
| Student ID Number: | 18330276 |
| Programme: | PHA4 |
| Module Code: | PS451 |
| Assignment Title: | Shallow Learning for Meteor Identification from Images |
| Submission Date: | 29/03/2022 |

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious.

I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy.

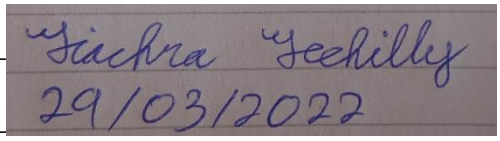
I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

I have used the DCU library referencing guidelines (available at: http://www.library.dcu.ie/LibraryGuides/Citing&ReferencingGuide/player.html) and/or the appropriate referencing system recommended in the assignment guidelines and/or programme documentation.

By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy (available at: http://www.dcu.ie/registry/examinations/index.shtml).

Name:__ 


Date:__

## Abstract

Research into meteors is becoming more significant as the field of astronomy continues to grow and more people get involved in astronomical research and as equipment and technology improve. The study of meteors can provide insight into the beginning of the formation of the solar system as they have remained mostly unchanged while existing in outer space. Being able to track asteroids, comets, meteors and other bodies that exist in space particularly near Earth is important but will also become more important as space exploration and satellites become more prevalent as global society grows. To observe and study meteors, different networks of camera stations have been setup previously. Most notably among those is the Global Meteor Network (GMN). To analyse all the footage recorded by these stations in the network, would require immense amounts of manpower and as such computer algorithms are utilised to reduce the amount of human input required. To this end the GMN use a meteor detection algorithm to pick out meteor detections from images. However this algorithm experiences many false positives as a result of airplanes, insects and clouds among other things. To reduce these false detections, deep learning was utilised. This has been spplied previously by Peter Gural tp data from another meteor detection network(CAMS) and it achieved high accuracy of ~99% of images correctly identified. The aims of this project thus were to attempt to replicate this level of success while using data solely from the GMN and also to investigate what affects the accuracy of the deep learning network. Impressive results of 98% accuracy were achieved by this project. Most significantly it was found that it was possible reduce the size of the neural network by ~950 times as compared to the neural network used by Peter Gural on the CAMS data. It was also found that there is a minimum detection image size required to achieve good results.

## Acknowledgements

I would like to firstly thank Dr. Jonathan Mackey for proposing this project and providing me with the opportunity to greatly expand my knowledge on deep learning, a topic which I hadn't dealt with before.

I would like to thank Dr. Denis Vida and the members of the Global Meteor Network for providing the data necessary for me to undertake such a project and for manually classifying each image, and also for their guidance on deep learning.

I would like to thank Dr. Masha Chernyakova and the other professors and educators in DCU for teaching me for the last four years and preparing me for this project and hopefully for my future career.

Finally I would like to thank my parents for supporting me throughout my entire academic journey.

# Contents

# Chapter 1: Introduction

## 1.1 Meteors and their significance

Meteors are defined as a piece of rock or other matter from outer space that produces a bright light as it travels through the atmosphere.[1] While they are pretty to look at, they also have considerable astronomical value. Before meteors burn up in the Earth's atmosphere they exist in the solar system as meteoroids, asteroids and comets. These objects when floating around in space do not undergo any major changes or experience any harsh conditions like they would on a planet such as Earth, with the exception of comets which melt slightly when close to the Sun. As a result of this they remain largely unchanged from when they were initially created. Many of these objects formed at an early stage in the formation of our Solar System, and as such their aspects, such as their composition give an insight into the conditions present at the early stages of the Solar System's formation. However investigating these objects in space is incredibly costly and difficult. Investigating meteors that have fallen on the Earth is much simpler and cost effective. Finding these meteors on Earth is still a difficult challenge.

These rocky objects also pose significant risk to space travel and satellites, both of which are industries that are likely to grow in the near future. Being able to keep track of these objects while they're in space could prevent major incidents and being able to track these objects as they fall to the Earth's surface would make recovery of them much simpler. To accomplish these goals, recording stations such as the Global Meteor Network (GMN)[2] were established. These recording stations use cameras to record the night sky and then use detection algorithms on the recorded footage to determine if a meteor was observed. The footage could then be used to determine the trajectory of the meteor and thus either their position in space or potentially the location where they touched down on Earth. The GMN was not the first network of recording station to be created, others such as the Cameras for All-sky Meteor Surveillance (CAMS) came first. The GMN is however currently the largest network of recording stations for meteor surveillance in the world. They utilise cheap modern camera equipment alongside open source software and post all their results publicly as soon as they have them. These aspects have made it incredibly popular with academics, amateur astronomers and members of the general public, allowing it to be easily and quickly expanded.

## 1.2 Project Aims

The GMN has to manage and analyse a large amount of data which will only increase as the GMN continues to grow. To reduce the human input required in the workflow of analysing these images, computer algorithms are already in place to pick out meteor detections in the data and perform automatic analysis. These detections algorithms are not perfect and currently experience a significant number of false positives that come as a result of objects in the night sky such as planes, insects and clouds among other things. The main aim of this project is to develop a deep learning neural network that is able to reduce the number of false positives that are being detected while still keeping as many true meteor detections as possible. Additional aims included investigating what aspects of the neural network or data pre-processing had the most significant impacts on the performance of the neural network.

# Chapter 2: Theory and Background

## 2.1 Basic concept of AI, Machine Learning and Deep Learning

Artificial intelligence began in the 1950s[3] when computer scientists began to wonder if it was possible to create programs complex enough that they could emulate human thinking and therefore be able to learn and adapt and thus achieve sentience. Artificial intelligence has not reached that point yet and is now used as blanket term used to describe many different techniques that are used in software in order to have computers perform complex tasks that humans are usually able to easily perform. Within AI is the subset of Machine Learning and within this then again is the subset of Shallow Learning and Deep Learning. The initial approach to AI was to write programmes with set rules that were known by the writers of the program and that once enough rules were added and the rules were complex enough then the programme would be able to solve the problem given to it without further input from the programmer. This was called symbolic AI.[4] This approach is quite limited as it requires everything to be accounted for by the programmer and will only ever be as good as the programmer's understanding of the problem that will be given to the AI to solve. Machine learning differs from this original approach in that the program is fed the data along with classifiers/identifiers/labels and will then try to determine the rules that govern the system.[5] The machine learning algorithm will attempt to find these rules by applying transformations to the data to try create a representation of the data that can be used to more easily identify the data[6]. A very basic example of this concept this is shown in Figure 1, while this sort of transformation does not necessarily take place in machine learning algorithms it clearly displays the concept of how transformations can make the data much easier to classify. The left hand side of Figure 1 illustrates a simple example of data, the input data is the x and y coordinates of the points and the classifiers will be the colour of the dot, either white or black. Looking at this example it is clear that there is linear division between the black and white dots as shown in the center image of Figure 1. The transformation applied in this case is an axes shift which then simplifies the dividing line between black and white to be x=0 so any dot with an x value greater than 0 is black and any dot with an x value less than 0 is white.

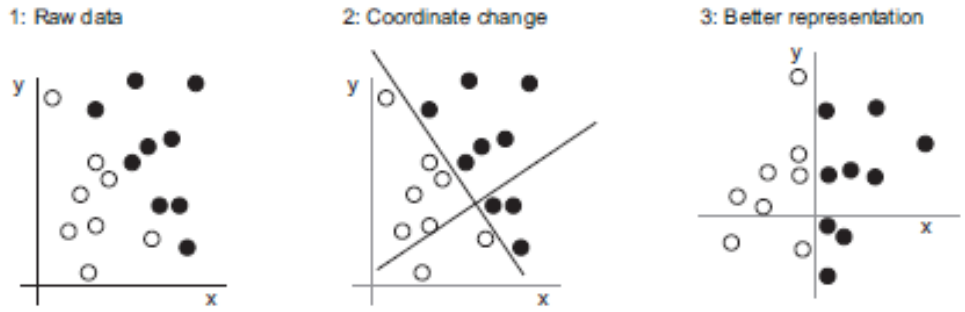



Figure 1 (Image representation of the actions of a simple data transformation [7])

Deep learning builds on machine learning by successively applying more and more transformations to create even more representations of the data which can be used to analyse the data. Each transformation is applied as a "layer" in the code and thus the deep learning algorithms are usually several layers deep and can even be 100s of layers deep. This is where the deep in deep learning comes from. These arrangements of layers in deep learning are typically called neural networks[8]. Figure 2 illustrates an example of a deep learning algorithm attempting to identify numbers from images, and some of the transformations it applies to the image to determine what it is.

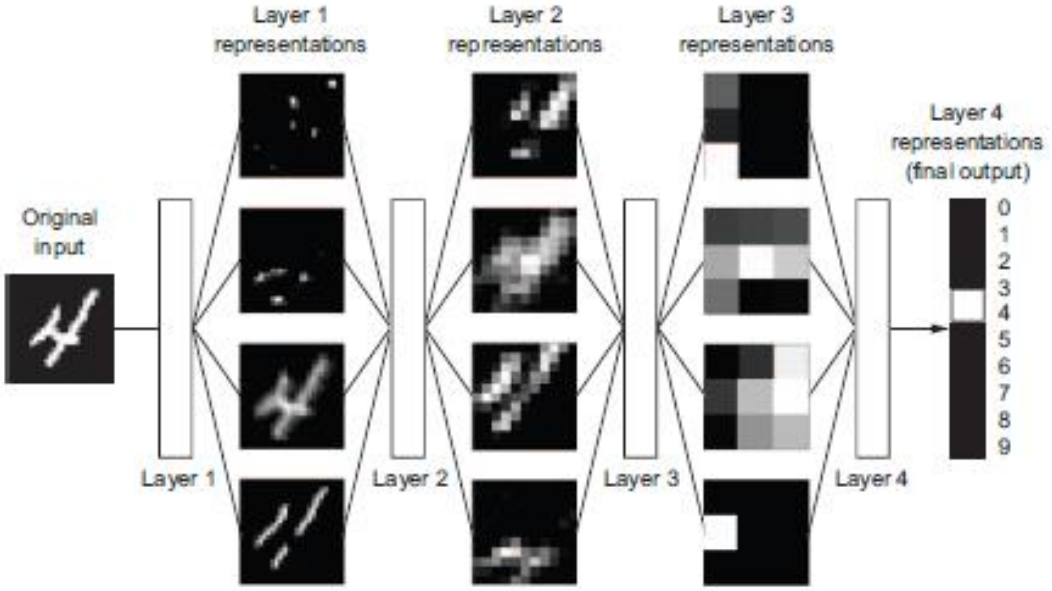



Figure 2(Basic deep learning algorithm and some of its transformations, as it attempts to identify a number from an image.[9])

Deep learning as a technique only rose to prominence in the 2010s. This is as a result of a couple of compounding factors. Firstly the hardware needed to run and compile a deep learning algorithm was previously not widely available. In the 2000s Graphics Processing Units (GPUs) started to become more widely available and were starting to be widely used even for recreational use like gaming. GPUs perform much better than conventional Central Processing Units (CPUs). This is because the deep learning algorithm consists of a large amount of simple tensor operations, while these operations can be quite simple the sheer number of them causes it to be computationally intensive for a CPU but as GPUs have a lot more cores than CPUs, they can do the tensor operations in parallel which reduces the overall time required. Additionally in the 2010s, GPUs became rapidly more powerful. Secondly in the 2010s, NVIDIA, one of the major companies producing GPUs released software libraries to allow graphics cards to be utilised in novel ways instead of just for graphics rendering, allowing them to be used for deep learning. Thirdly, as a result of the other two research into deep learning began and very quickly new techniques specific to deep learning were developed that drastically improved the accuracy of deep learning algorithms.[10] The terms Shallow Learning and Deep Learning may be used interchangeably in this report, as they make use of the same concepts and procedures. Shallow Learning simply refers to a neural network that is much smaller in size, in terms of neurons and layers, than typical Deep Learning neural networks.

## 2.2 Deep Learning Techniques

### 2.2.1 Learning Types

There are four main learning styles that are used with neural networks. These four styles are: supervised learning, unsupervised learning, self-supervised learning and reinforcement learning. Supervised learning is the most commonly used[11] and is the method that was used in this project, as such it is the only type that will be discussed in detail in this report. Supervised learning is the method that was described previously for how deep learning works. The neural network is fed the data along with classifiers for each piece of data. The neural network will then try and determine the best transformations to perform, to assign the correct classifier to each element of the data. This is well suited to problems that include: sequence generation, syntax tree prediction, object detection in images and image segmentation.

### 2.2.2 Dense Layers

The basic type of layer utilised in deep learning networks are called Dense layers. They consist of nodes that each perform operations on input data from the previous layer. Each layer in the neural network will consist of many hundreds or thousands if not more simple tensor operations, for example:

$$relu(dot(W, input) + b) \tag{2.1}$$

Equation 2.1 is an example of a function that could be part of a layer. In this example, the relu (Rectified Linear Unit) function outputs zero if the input is negative and outputs the input if the input is 0 or positive, next is the dot function which outputs the dot product of the input term, in this case W and input. In this example W and b are called weights and these are the parameters that the neural network will modify when training the network. The input term is the data that is fed into the neural network or the output from the previous layer. When beginning the training of a neural network these weights are set randomly but will be changed by the neural network as it trains. A dense layer would consist of many nodes where each node is an operation similar to Equation 2.1. In deep learning, dense layers and other layers are interconnected. Being interconnected means that the outputs from all the nodes in a previous layer are a single input into all the nodes of the next layer. As an example, imagine a neural network consisted of 3 Dense layers, and has one initial input and one output. The first Dense layer has 3 nodes, the second Dense layer has 4 nodes, and the final Dense layer has only 1 node as it produces only one output, the shape of this neural network is shown in Figure 3. The same input goes into each of the nodes in the first layer. The output of each of these nodes then act as 3 inputs going into each node in the second layer to produce 4 different outputs from layer 2. These 4 outputs from layer 2 are then 4 inputs for the final layer which produces a single output. This style of interconnectedness also exists with other layer types such as Convolution layers. The interconnectedness adds complexity to deep learning neural networks which can drastically improve their performance but also cause the size and complexity of the neural networks to expand rapidly.

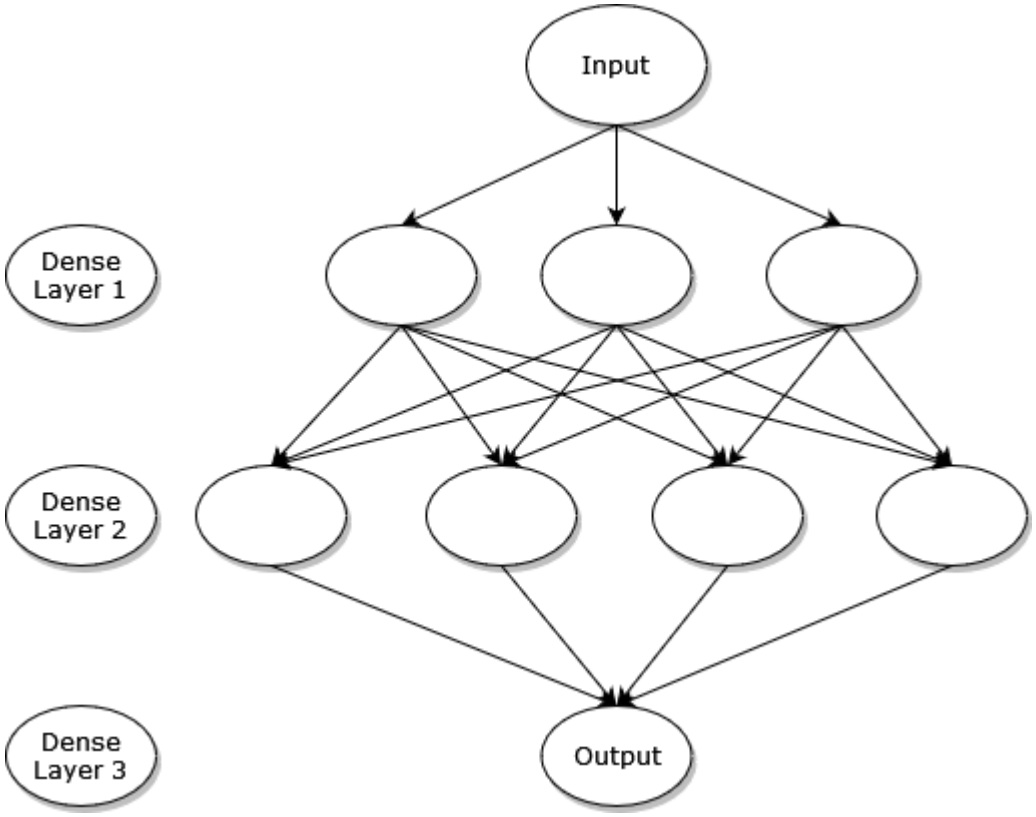



Figure 3 (Illustration of Dense interconnected layers in a deep learning neural network)

### 2.2.3 Training, Testing, Validating

As mentioned previously the neural network takes data and classifiers for input. The output is the data and the classifiers that they the network should be assigned to each data item. However the accuracy of the neural network is not guaranteed and as such its accuracy needs to be measured. To measure the network's accuracy the input data needs to be split into three sets: the training set, the testing set and the validation set, each set should be mutually exclusive. The training set is the data along with the classifiers that will be given to the neural network initially. The network will then use these to adjust the parameters of each layer in the network so that each item will be correctly identified. The purpose of the testing data set is to evaluate the network's performance of data that it did not train on, as the network may be able to identify correlations in the training set to identify its elements but these correlations may not exist in other data and thus the network will perform poorly on new data even though it performs extremely well on the training data, this is called overfitting[12]. To evaluate the network's performance it is given the testing set and also the classifiers assigned to each item in the data set, however this time the network will not adjust any parameters and will instead make predictions on what classifier should be assigned to each input. These predictions are then compared to the actual classifiers that each item and thus the accuracy rate of the network can be determined. If the network performs poorly on the testing set, adjustments can be made by hand to the structure of the layers, including the layer types, number or layers, order of layers etc. The validation set can be thought of as an additional testing set. As the network parameters are repeatedly manually changed to increase the network's performance on the testing set, this can lead to the network being unintentionally overfit to the testing set[13]. Thus the validation set exists as a way to check and thus prevent overfitting of the network to the testing set, however it is not always used.

### 2.2.4 Stochastic Gradient Descent

Training the neural network refers to the neural network changing its own parameters for the many tensor operations in each layer of the network. However determining the changes that are needed to improve the network's performance is not a simple problem. When determining the performance of the neural network a loss function is used which gives a numerical value to represent whatever metric is used to measure the performance, usually this would be the accuracy of the classification. For a binary classification where there are only two different classifications the accuracy would be the percentage of times that the neural network correctly classified the image. With the function given as an example as Equation 2.1, it would be easy to determine how the loss function is affected as W and b are changed using differential calculus, i.e. the derivative of the loss function with respect to W and b. Using this 2 variable derivative it is simple to calculate the values for W and b for when the derivative equals 0 which corresponds to the minimum value of the loss function and thus the maximum performance possible of the network. This concept can be applied in the same way to the whole network using the chain rule for differential calculus. So if all the weights of the network are known it will be a computationally simple process to compute the gradient (derivative of a tensor) even though the changing of a weight can affect many other functions in the layer and subsequent layers of the network. However determining the absolute minimum point of this gradient, as in where the gradient equals 0, would require solving for all the weights in the network which would be an N-degree polynomial, where N is the number of weights in the network which is usually greater than 1000 and can often reach into the millions. Stochastic Gradient Descent (SGD)[14] was created as a method for neural networks to determine the weights needed to achieve minimum loss functions and thus maximum performance. Typically a form of SGD called mini-batch SGD is used as it usually performs much better than traditional SGD as it is able to compute over an average rather than for each individual data input. To perform mini-SGD a small batch of training samples and their classifiers are used, the neural network will generate predictions and then the loss function is used to measure the accuracy of the predictions against the actual classifiers. The gradient of the loss function is determined with respect to the network's parameters. The parameters are then adjusted to go in the opposite direction of the gradient, thus going towards the minimum point of the loss function. The next batch of samples is then fed into the network and the same steps are repeated. This process of analysing a batch of samples, computing the gradient and then adjusting the parameters in the opposite direction of the gradient is called mini-batch GSD. The process is repeated until the loss function reaches its minimum point. While this may seem like a long process it is considerably faster than the trying to solve the N-polynomial equation as mentioned previously. Typically the neural network is trained over the entire available training dataset multiple times, each training iteration over the dataset is called an epoch. Increasing the number of training epochs that are used usually increases the accuracy of the neural network but training over an excessive number of epochs can cause overfitting.

### 2.2.5 Data Pre-processing

Data pre-processing refers to manipulating the data before feeding it into the deep learning algorithm. This could include simple operations, for example truncating unnecessary information, changing data types, co-ordinate transforms. Other operations specifically for images would include cropping the image, resizing the image, applying filters, rotating the images, etc. While the neural network will be applying transformations to the data is inputted, by deliberately doing specific transformations it can help guide the neural network towards a more favourable configuration that performs better. These transformations such as cropping or filters will also reduce the size of the input data and this can drastically reduce the computational costs of the neural network. For example, a greyscale image has a third of the information of an RGB colour image, the greyscale image will therefore be able to be processed faster and it also prevents the neural network from trying to find correlations between the colours in images if colour is not relevant and will thus have to find correlations in other aspects of the image. Overall data pre-processing will reduce the

complexity of the neural network and usually the simpler neural networks perform better than their complex counterparts.

### 2.2.6 Overfitting

Overfitting as mentioned previously is when the neural network starts to perform much better on the training data than it does on the testing data. This is as a result of the neural network tuning its itself to find correlations and patterns that only exist in the training dataset and does not exist in the testing dataset or any new, unseen data. As this is a known feature of neural networks there are several ways that have been developed to reduce overfitting. Of course one of the easiest ways of trying to improve the neural network is to simply give it more training data. However the easiest way to reduce overfitting without increasing the amount of training data is to reduce the number of parameters that the network has. If there are too many parameters that can be adjusted, the neural network can become too specific in the features that it is looking for and thus won't work well on new data[15]. Weight regularisation[16] is another method use to prevent overfitting. Weight regularisation works on a similar premise to Occam's razor, that often the simplest explanation is the best explanation. This is done by constraining the weights to have smaller values thus reducing the spread of the weights. Possibly one of the stranger method of improving the performance of a neural network is called dropout. Dropout is a method where some outputs from a layer are randomly "dropped" (set to 0) [17][18]. The dropout rate is typically set to between 0.2 and 0.5. While dropout seems strange, the creator Geoff Hinton gives the analogy of bank tellers being swapped around in order to prevent "conspiracies". By randomly dropping the outputs it prevents these "conspiracies" from forming, i.e. finding features that are specific to the training dataset.

### 2.2.7 Convnets

Deep learning specifically for images performs significantly better when an additional type of layer is added to the neural network. These layers are called convolution layers, as they use convolution operations. The convolution produces an output image from the input image by looking at the area around a single pixel in the image, usually a 3x3 or 5x5 area centered on this pixel and applying a mask/kernel, the output of the mask will be the new value in the output image which is at the same location as in the original image, as shown in Figure 4.

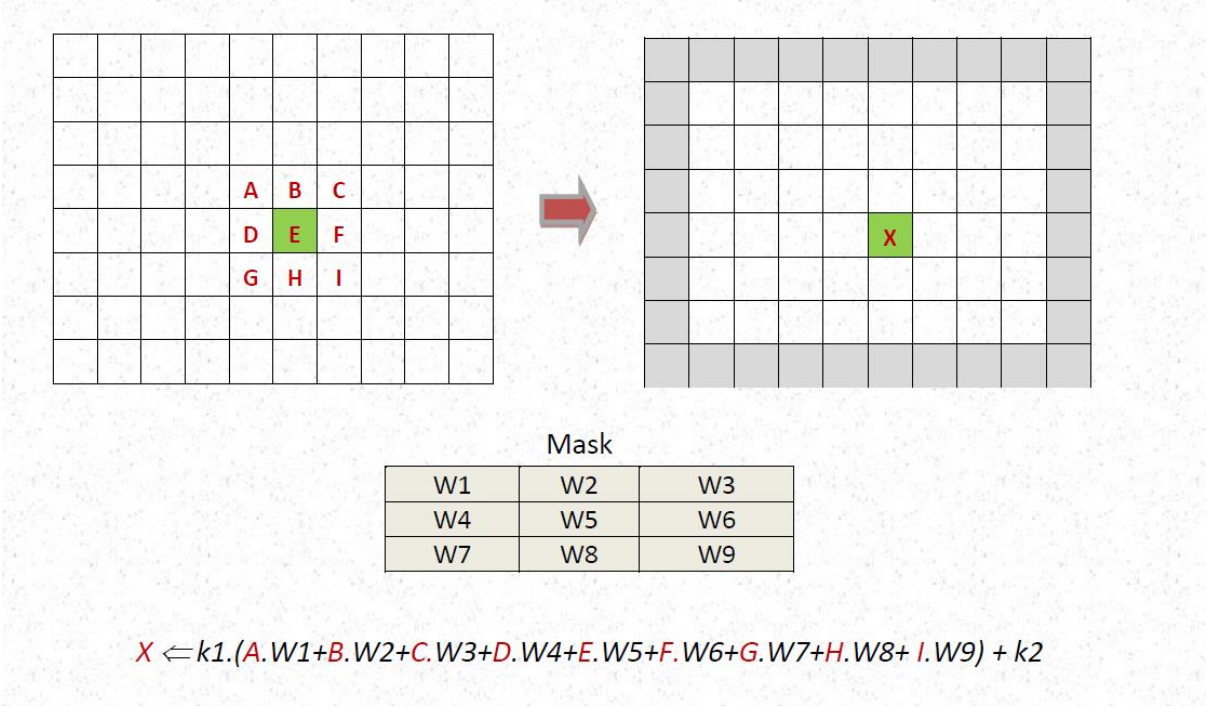


Figure 4(example of a convolution operation on an image[19])

The weights used in the masks/kernels of a convolution are initially set to random but will be adjusted by the neural network as it trains, in the same way the weights are dealt with for Dense layers. Convolution layers in the neural network allow local features in the data to be identified, by examining a pixel and its neighbouring pixels. In between each convolution layer there is typically a pooling layer which downsamples the image. By downsampling the image before applying another convolution layer, it in effectively zooms out and makes the everything in the image smaller in terms of the number of pixels it takes up. By doing this, larger features in the image get decreased in size so the 3x3 or 5x5 in the convolution layer will then be able to look at the larger features in the image thus meaning that each successive convolution layer in the neural network will be looking at larger and larger features in the image[20]. Dense layers are then typically placed at the end of the neural between after all the convolution and maxpool layers, a Flatten layer is also placed in between the final maxpool layer and the first Dense layer. The Flatten layer is placed there as the Dense layers are only able to analyse one dimensional tensors. The final layer in a neural network is typically a dense layer that uses an activation function like softmax or sigmoid to generate a single probability value, in the case of a single output neural network.

### 2.2.8 Analysing the performance of neural network

The most basic metric for analysing a neural network is its accuracy. The accuracy in this case would be the percentage of inputs into the network that are identified correctly. However the accuracy is not always a good metric for analysing the performance of a neural network. For example, if a neural network was differentiating between images of cats and dogs but the dataset consisted of 99% dog images and only 1% cat images and the neural network simply guessed dog for each image, it would have a 99% accuracy. While this level of bias in a dataset is severe, most datasets will not have an equal distribution and so the bias in the dataset needs to be accounted for when analysing the performance of a neural network. This is typically done using additional metrics such as confusion matrices, recall scores and precision scores. For a binary classification problem like cats vs dogs, or meteor vs non meteor a simple 2 by 2 confusion matrix will suffice. They take the form shown in Figure 5

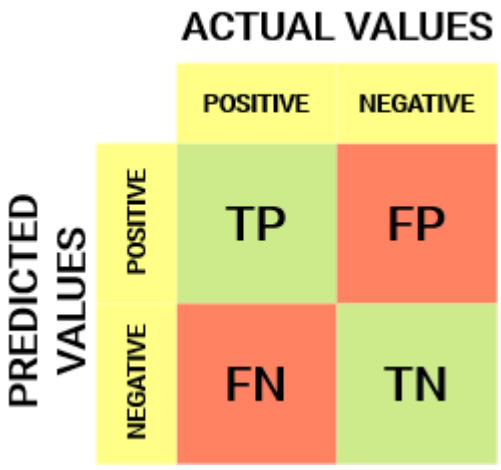



Figure 5 (Template of a confusion matrix used to analyse the performance of a neural network, TP stands for true positive, FP stands for false positive, FN stands for false negative and TN stands for true negative)

The upper left segment are what would be considered true positives so cases where the model correctly predicted it was a meteor and it actually was. The bottom right segment would then be considered the true negatives, which would be the number of non meteor detections that the network correctly identified as non meteors. The upper right segment is the false positive, the number of non meteors incorrectly classified as meteors by the network. The lower left segment represents the false negatives, the number of meteors incorrectly identified as non metoers by the network. For a well performing network the values the true predictions would be high and the number of false predictions would be low. The confusion matrix will show if the network has bias in either direction. Precision is the percentage of times a positive prediction is correct and is calculated using the formula shown in Equation 2.2.

$$Precision = \frac{Num.\ True\ Positives}{Num.\ True\ Positives + Num.\ False\ Positives}$$

Recall is the percentage of actual positives that are correctly identified and is calculated using the formuala shown in Equation 2.3.

$$Recall = \frac{Num.\ True\ Positives}{Num.\ True\ Positives + Num.\ False\ Negatives}$$

Using these additional metrics, one is able to analyse any bias that may be present in the neural network, that would be hidden by solely using the accuracy metric.

## 2.3 Deep Learning on Meteor Images

The idea of applying deep learning to identify meteors from images has been done previously, most notably by Peter Gural who managed to achieve an accuracy of >99% using a convolutional neural network[21]. This work using data from the CAMS(Cameras for All-sky Meteor Surveillance), which consists of cameras placed across several different countries which provides a wide array of data samples to be used to train the neural network. The neural network has a structure as shown in Figure 6.

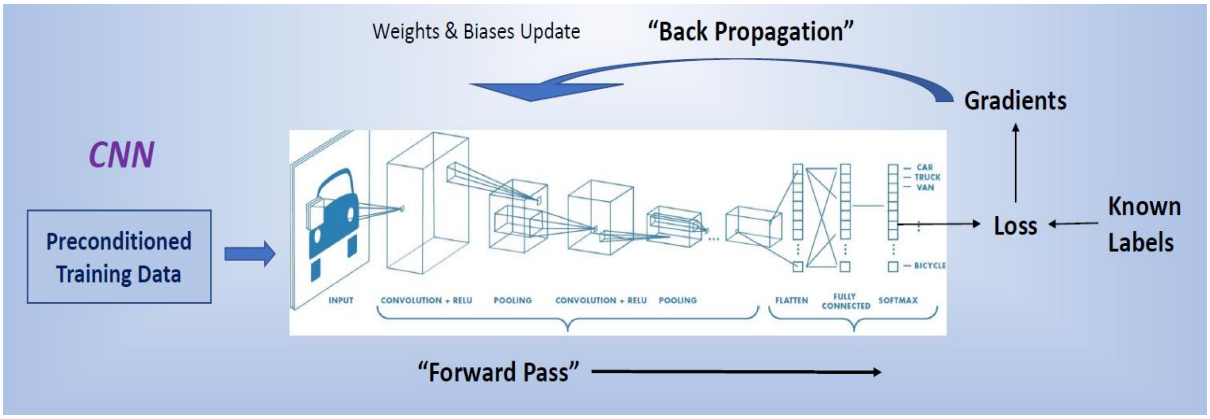



Figure 6 (Structure of neural network used by Peter Gural on CAMS network data[22])

Peter Gural's work also states that are several factors that do not affect the accuracy of the neural network. Firstly using RGB(Red, Green, Blue) colour images does not have a benefit over using greyscale images and so greyscale images were used to reduce the amount of computation needed[23]. Secondly utilising time dependent data provides little difference as compared to using

time independent data[24]. Peter Gural also applied this convolution neural network to data from another meteor camera network called GMN(Global Meteor Network) which is the network from which provided the data for use in this project, and achieved a 91% accuracy even without retraining the network[25]. This shows that the neural network that had been developed is able to be applied to other data sets coming from different hardware sets and still retain a good performance.
Another project that attempted to use deep learning neural networks to identify images from the GMN data was done in 2021 by Esther Gavin. This used a very similar structure to the network as shown in Figure 6, with only a difference in the last couple layers and different size layers. Esther Gavin's neural network was able to achieve an accuracy of 85%[26]. While this is lower than the accuracy of Peter Gural's neural network, that is likely as a result that Esther Gavin trained a new neural network on the GMN dataset which was smaller than the CAMS dataset. However the 85% accuracy is a testament to the capabilities of deep learning to perform without overly complicated input from users.

# Chapter 3: Methodology

## 3.1 GMN data format

The dataset provided by the GMN consisted of two main directories, confirmed meteor detections and rejected meteor detections i.e. true and false positive detections. Within these two directories there were many subdirectories from nights where detections occurred. The GMN utilises a compression method that had been devised for use in the CAMS meteor detection system by Peter Gural[27] to store 256 frames of video as 4 frames within a fits file.[28] More conventional compression methods are not suitable for this task as they cause loss of pixel details in particular for faint pixels which are how many meteors appear on recorded footage. The GMN compression method preserves image quality while sill reducing the size of files that are uploaded to their central servers. The fits file can also be used to somewhat reconstruct the original video. Most of the GMN camera stations record at 25fps (frames per second) and so each batch of 256 frames is 10.24 seconds of footage and the time interval between frames is 0.04 seconds. The 4 images in the fits file are labelled as maxpixel, avgpixel, maxframe and stdev. Only the maxpixel, avgpixel and maxframe images were used and so only these images will be discussed. Figures 7, 8, 9 show the maxpixel, avgpixel, and maxframe image respectively, from a single fits file.

Figure 7 (Maxpixel image from a fits file showing a Confirmed meteor detection in the upper left)

Figure 8 (Avgpixel image from a fits file showing a Confirmed meteor detection in the upper left)

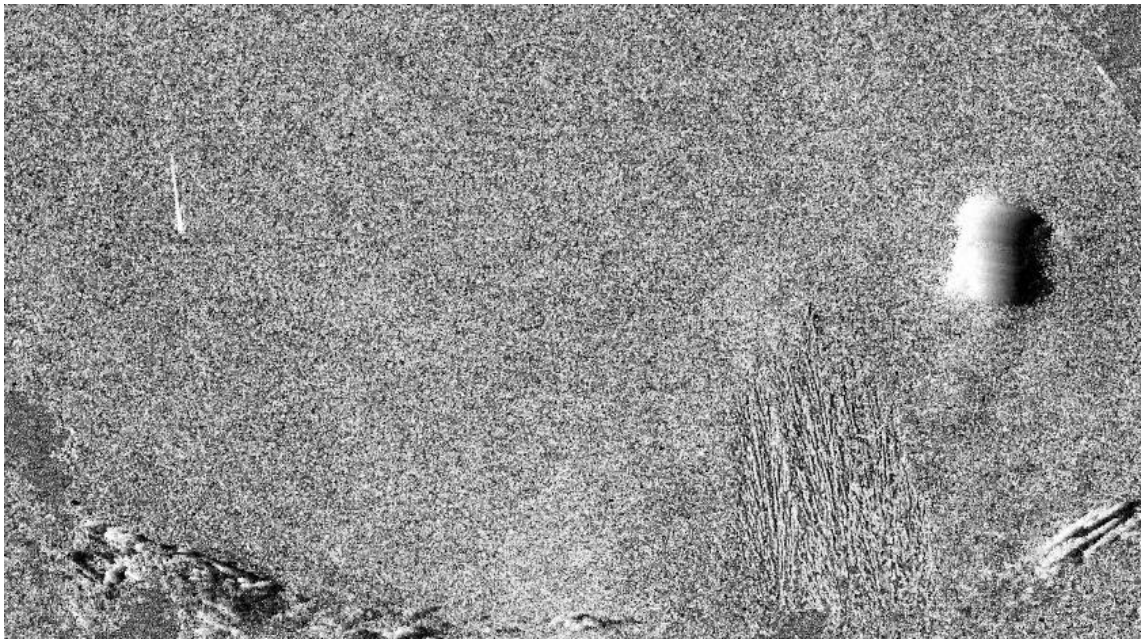


Figure 9 (Maxframe image from a fits file showing a Confirmed meteor detection in the upper left)

The maxpixel image is a max intensity image so each pixel in the image is the max value that pixel experienced over the 256 frames. The avgpixel is the average intensity image so each pixel in the image is the average value that pixel experienced over the 256 frames. In the maxframe image each pixel has a value from 0-255 and it corresponds to the frame number when that particular pixel experienced its max intensity. As an example, if a pixel at location (20,20) in the image experienced a flash of intense light for 0.04s(1 frame) that maxed out the sensor 10 frames after the camera began recording that particular batch of frames and no light the rest of the time. In the maxpixel image this

pixel would have an intensity value of 255, the max value read out from the camera. In the avgpixel image this particular pixel would have a value of ~1, the average value of 255 zeroes and a single 255. The maxframe pixel at (20,20) would have a 10 as that is the frame where the max value occurred. As the GMN utilises their own detection algorithm, the machine learning algorithm is intended to come after this step in the processing pipeline and as such all information from the detection algorithm can be utilised to pre-process the image data before inputting it into the machine learning algorithm. The GMN algorithm output a txt file for each fits file. This txt file contains information about each detection in the fits file including the pixel location in the image where the detection occurs and also the frame numbers in which the detection occurs.

## 3.2 Pre-processing the dataset

All pre-processing of the dataset was done with code that utilised functions from the following python libraries: tensorflow, numpy, astropy, RMS(GMN code library) and inbuilt python libraries.

Several different datasets were created from the initial GMN dataset to investigate the effects of different forms of pre-processing on the performance of the machine learning algorithm.

The first dataset was created as follows. The first step to pre-processing these images was to subtract the avgpixel image from the maxpixel image. This reduces unwanted noise and additional information like stars and the background in the image as it means that the only bright pixels in the subsequent image are those where movement occurred. Then using the information from the txt files, the images were cropped spatially to a rectangular shape which is the size of the detections and the time/frame information was used to time slice the image and remove any bright pixels that did not occur at the same time/frame as the detection, to further reduce noise in the images. When this was done initially it was noted when qualitatively looking at the resultant images that the tails of many meteors were cut off as they were not bright enough to be registered by the initial GMN detection algorithm as illustrated in Figure 10. As such it was decided to increase the size of the cropping rectangle to include a “border” of +20 pixels on each side of the detection. If the detection occurred at the edge or corner of the image additional rows or columns of black pixels were added to keep the detection centered in the image and have the required border on each side. These final images were then made square by adding rows or columns of black pixels as needed, while keeping the detection centered in the image. The purpose of making the images square is that the machine learning algorithm needs all input images to be the same size and so the images will be stretched at a later stage but by making the images square it ensures the detection is centered in the image and will not asymmetrically distorted when stretched/shrunk to the required input size.

This procedure was used to generate several datasets with different size “borders”, of sizes: +5 pixels, +10 pixels, +20 pixels, +30 pixels.

The above procedure was also used to generate datasets from the maxframe images. As the maxframe images encode the time information, they were used to investigate if including the time information would affect the performance of the machine learning algorithm. The detections were cropped in the same way as mentioned previously with borders of +20 pixels. With the maxframe dataset was generated without using the time slicing method was not used. Examples of how these images appear after pre-processing are shown in Figures 11, 12 and 13

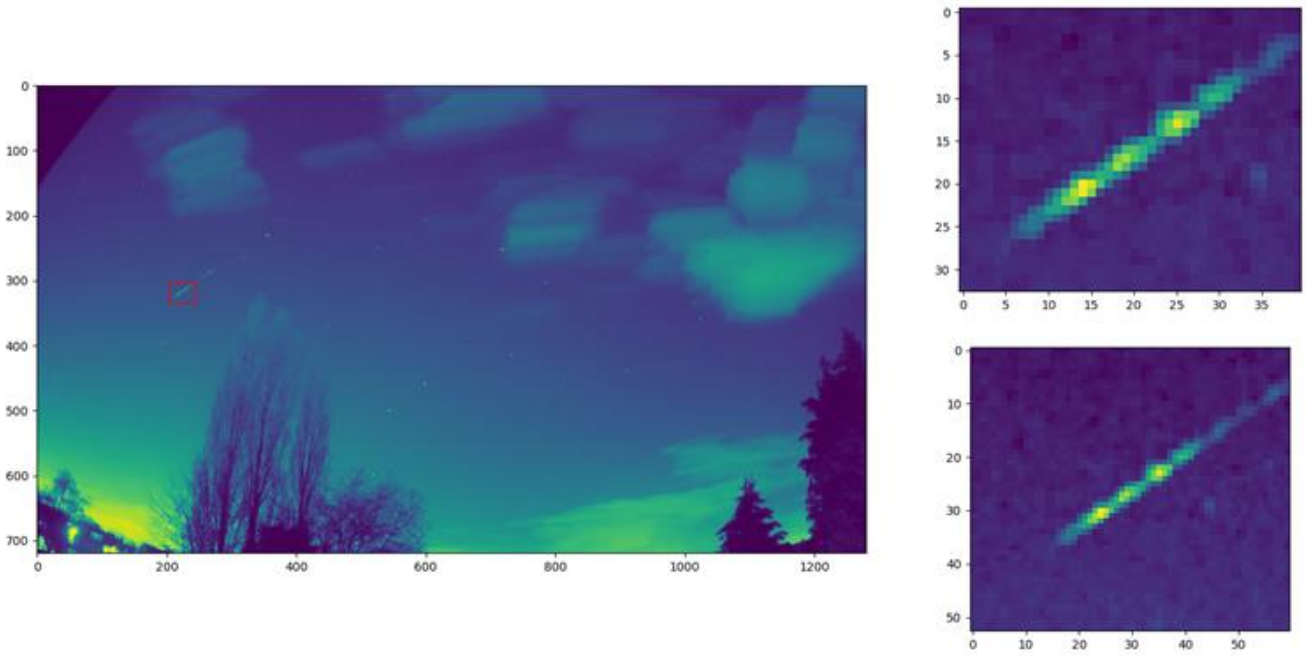


Figure 10 (red rectangle in image on the left is the exact cropping rectangle based on the information from the detection txt files, the image in the upper right is an enlarged version of this rectangle with +10 pixels on each side also included. The image in the lower right is an enlarged version of this rectangle with +20 pixels on each side also included. It should be noted that the entire meteor trail is not inside the red box)

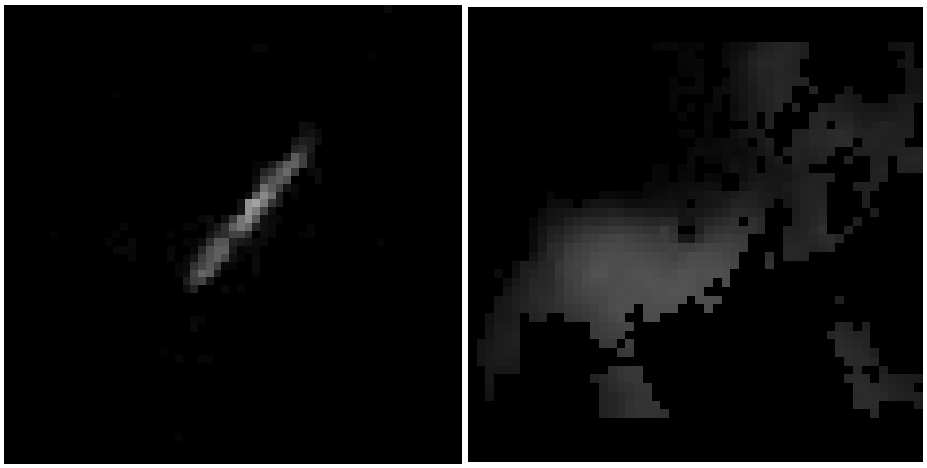


Figure 11 (Timesliced movement image +20pixels each side, Confirmed Meteor detection on left and rejected meteor detection on right)

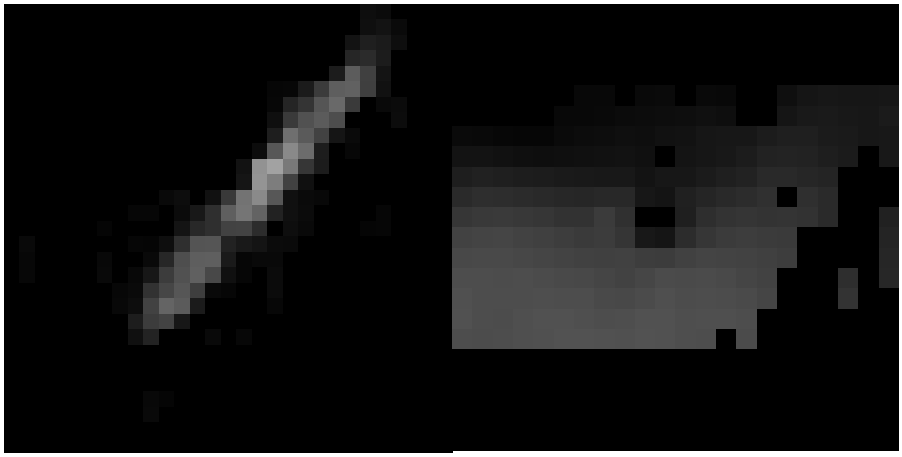


Figure 12 (Timesliced movement image +5pixels each side, confirmed meteor detection on left and rejected meteor detection on right)

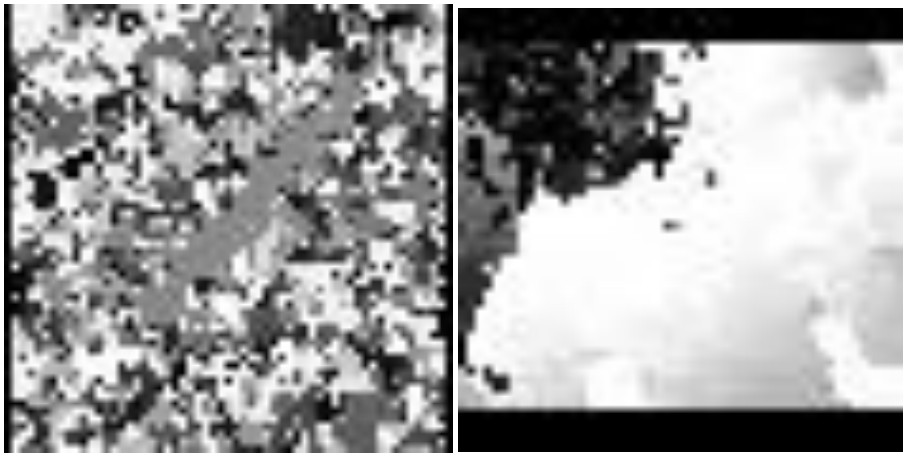


Figure 13 (Non timesliced maxframe image +20 pixels each side, confirmed meteor detection on left and rejected meteor detection on right)

Figures 11, 12 and 13 show the same confirmed detection and rejected detection and showcase the differences in appearance that the different preprocessing steps cause. It should be noted that these images have been stretched to appear the same size, which can be noticed in particular when comparing the rejected images in Figures 11 and 12. The rejected image in Figure 12 only shows the very small region at the centre of the rejected image in Figure 11. The black dots/segments that are particularly noticeable in the rejected image in Figure 11 as well as the noise in the confirmed image in Figure 13 are compression artifacts from the compression that the GMN uses.

When training the networks, additional preprocessing steps were taken to make use of the functionality and increased performance of the tensorflow library. These steps loaded the png images in batches of 32 images. The images were stretched or shrunk to be 128 by 128 pixels using a bilinear interpolation method. For each image the png image was converted into a Numpy style array. The intensity value range of each image was rescaled from 0-255 to 0-1 by converting the intensity values to floats and dividing each pixel value by 255. The intensity values were then centered about 0 by subtracting the mean intensity of the image from each pixel intensity in the image. The intensity distribution was then standardised by dividing each intensity value in the image by the standard deviation of the image. This step causes some images to then have intensity values with a magnitude greater than 1 so the images were rescaled again to be between -1 and 1 by dividing each pixel intensity by the max pixel intensity in the image. The image datasets were then

split into training and validation datasets, with 80% of the total images being put in the training datasets and the remaining 20% in the validation datasets with roughly equal proportion of confirmed meteor detections to rejected meteor detections in each dataset.

## 3.3 Neural Network Structure

The neural network code was implemented using the Keras framework for deep learning neural networks. The structure of neural network used was very similar to the structure used by Peter Gural with some small changes. The overall structure of the first neural network used can be seen in Figure 14. The main differences between the Gural network structure and the network used in this project are that a sigmoid activation function was used instead of a softmax function in the final layer of the network, the loss function was changed to inbuilt keras binary crossentropy function, and the optimizer function was changed from the SGD optimizer function to the Adam[29] optimizer function.

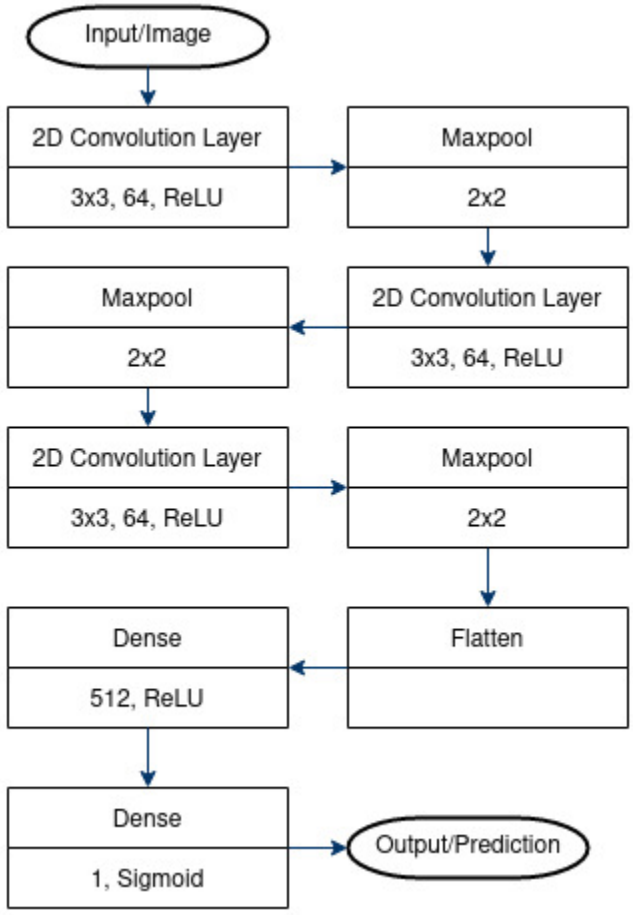



Figure 14 (Overall structure of neural network)

This same network structure was used to train different neural networks on the different previously generated png datasets. A similar structure was also devised to have both the cropped movement image and a cropped maxframe image as the input. The overall structure of this style of network is shown in Figure 15.

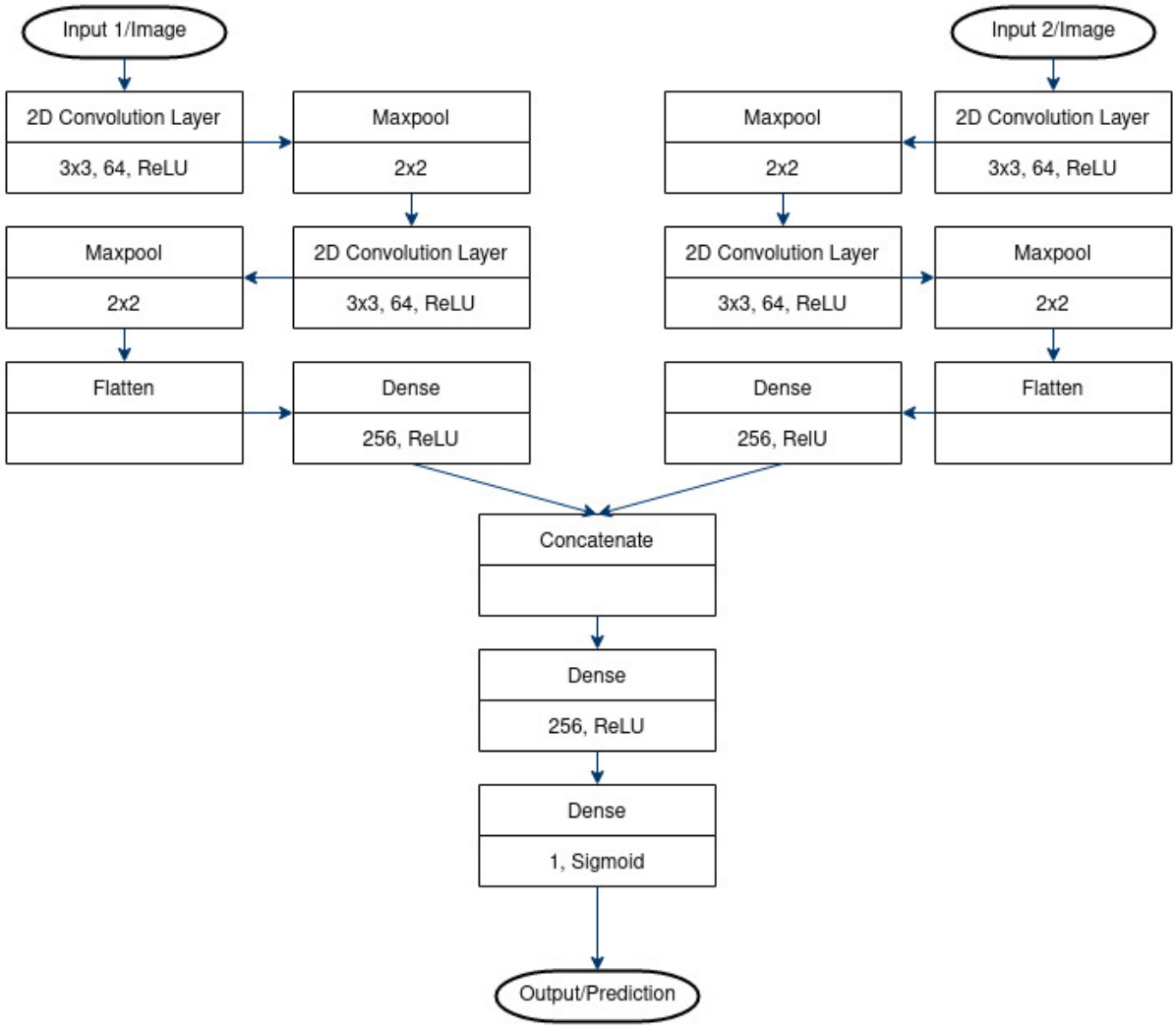



Figure 15 (Overall structure of two input neural network)

While those were the initial structures used, other changes were also made to investigate what effect if any they would have on the performance of the neural network, including changing the output size of layers(in essence changing the number of parameters each layer has), adding and removing convolution, maxpool or dense layers, changing the activation functions of layers, changing kernel size for the convolution layers, changing the stride length for the maxpooling layers and for the two input network changing how the method for how the two inputs are combined.

## 3.4 Augmenting the data

As the dataset provided by the GMN had more confirmed detections than rejected detections, data augmentation was used to generate some additional datasets in order to investigate the effects of trying to artificially reduce this bias. This was done by simply flipping the png images. Horizontally flipped (left to right) copies of the confirmed pngs were created. The same was then done to generate additional rejected pngs. Copies of these rejected pngs were then flipped vertically(top to bottom) were then generated). This doubles the size of the confirmed detections dataset and quadruples the rejected detections dataset drastically reducing the bias in the dataset while also increasing the overall size of the dataset used for training and validation.

# Chapter 4: Results, Analysis & Discussion

## 4.1 Results and Analysis

A total of 67 different variations of the neural network were tried, however only 9 had significant importance due to either their structure, overall performance or dataset used for training and validation, as such only these 9 are detailed in this report. The details of these different neural networks are listed in Table 2. Several different datasets were generated from the initial dataset provided to investigate the effect of any pre-processing steps on the performance of the neural network. The details of these different datasets are listed in Table 1. Each dataset entry in Table 1 is assigned a name such as Dataset 1, Dataset 2 etc. and that is how the datasets will be referred to in the rest of the report. It should be noted that for datasets 2 and 3, that ~8% of the detections had been listed as both confirmed and rejected meteor detections, which explains why the accuracy of any of the neural networks trained on these datasets did not achieve higher accuracies. Datasets 4, 5, 6 and 7 were rectified so that these images were listed as either confirmed or rejected and not both.

The timeslice column in Table 1 refers to using the information from the maxframe image to remove any pixels that did not occur at the same time as the detection as it was listed in the txt file. The image type used column refers to two different types of images used. The first type is the movement image, which is the image generated as described in the methodology section where the avgpixel image is subtracted from the maxpixel image to create an image that contains only pixels that relate to movement. The second image type is the maxframe image of the detection. Both image types are cropped to the desired size around the detection as noted in the txt files. The augmentation column refers to whether the dataset has been artificially enlarged by flipping the images as described previously.

Table 1 (Details of each dataset used for the different neural networks)

| Name | Num Confirmed Detections | Num. Rejected Detections | Image Type Used | Border size (+ pixels each side) | Timesliced | Augmented |
|---|---|---|---|---|---|---|
| Dataset 1 | 56,709 | 15,780 | Movement | 20 | Yes | No |
| Dataset 2 | 57,407 | 31,275 | Movement | 20 | Yes | No |
| Dataset 3 | 58,691 | 27,730 | Maxframe | 20 | No | No |
| Dataset 4 | 58,690 | 25,985 | Movement | 20 | Yes | No |
| Dataset 5 | 117,380 | 103,940 | Movement | 20 | Yes | Yes |
| Dataset 6 | 117,380 | 103,940 | Movement | 5 | Yes | Yes |
| Dataset 7 | 117,380 | 103,490 | Movement | 10 | Yes | Yes |
| Dataset 8 | 117,380 | 103,490 | Movement | 30 | Yes | Yes |

As several different neural networks were tested, the performance of each neural network, details of their structure and the dataset used for training are summarised in Table 2. The performance of the neural networks is given as the average fractional accuracy of the training accuracy and validation accuracy of the network before overfitting occurs. The training and validation accuracy are self reported by the keras software and is the fraction of images correctly identified in the training and validation dataset respectively by the neural network. The networks were considered to have begun overfitting once the validation accuracy began to either decrease or stay constant over several training epochs, while the training accuracy continued to increase. The neural networks are

named as CNN 1, CNN 2 etc. where CNN stands for Convolutional Neural Network. The structure is laid out as the list of layers in the sequence they occur. For convolution layers the numbers inside the brackets refer to the output size of the layer and the 2D size of the kernel used, e.g. (64, 3, 3) is a layer that has 64 outputs and uses a kernel of size 3x3. For the maxpooling layers the size of the strides used are indicated in the brackets. For the dense layers the number in the brackets is the number of outputs the layer has. The activation function of each layer is also listed at the end of each line

Table 2 (Details of different neural networks and their performance)

| Name | Structure | Dataset Used | Accuracy |
|---|---|---|---|
| CNN 1 | Input(128,128)<br>Conv2D(64, 3,3), relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Flatten<br>Dense(512) relu<br>Dense(1) sigmoid | Dataset 1 | 0.9775 |
| CNN 2 | Input(128,128)<br>Conv2D(64, 3,3), relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Flatten<br>Dense(512) relu<br>Dense(1) sigmoid | Dataset 2 | 0.8995 |
| CNN 3 | Input(128,128)<br>Conv2D(64, 13,13), relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Flatten<br>Dense(512) relu<br>Dense(1) sigmoid | Dataset 3 | 0.9395 |
| CNN 4 | Input(128,128)<br>Conv2D(64, 3,3), relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Flatten<br>Dense(512) relu<br>Dense(1) sigmoid | Dataset 4 | 0.9883 |

Table 2 (continued)

| Name | Structure | | Dataset Used | Accuracy |
|---|---|---|---|---|
| CNN 5 | Input_1 (128,128)<br>Conv2D(64, 3,3), relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Flatten<br>Dense(256) relu | Input_2 (128,128)<br>Conv2D(64, 3,3), relu<br>Maxpool(2,2)<br>Conv2D(64, 3,3) relu<br>Maxpool(2,2)<br>Flatten<br>Dense(256) relu | Dataset 3 & 4 | 0.9695 |
| | Concatenate<br>Dense(256) relu<br>Dense(1) sigmoid | | | |
| CNN 6 | Input(32,32)<br>Conv2D(16, 5,5), relu<br>Maxpool(5,5)<br>Flatten<br>Dense(16) relu<br>Dense(1) sigmoid | | Dataset 5 | 0.9820 |
| CNN 7 | Input(32,32)<br>Conv2D(16, 5,5), relu<br>Maxpool(5,5)<br>Flatten<br>Dense(16) relu<br>Dense(1) sigmoid | | Dataset 6 | 0.5300 |
| CNN 8 | Input(32,32)<br>Conv2D(16, 5,5), relu<br>Maxpool(5,5)<br>Flatten<br>Dense(16) relu<br>Dense(1) sigmoid | | Datset 7 | 0.9811 |
| CNN 9 | Input(32,32)<br>Conv2D(16, 5,5), relu<br>Maxpool(5,5)<br>Flatten<br>Dense(16) relu<br>Dense(1) sigmoid | | Dataset 8 | 0.9805 |

It was found that the size of the border around the image made a significant difference and that there exists a minimum threshold that exists at which the accuracy of the neural network begins to improve. This threshold exists somewhere between 5 and 10 pixels on each side of the detection, this can be seen by comparing the accuracy of CNN 7 and CNN 8 in Table 2. Increasing the size of the border above 10 pixels does not seem to yield an increase in accuracy as seen by comparing CNN 6, 8 and 9 in Table 2. Using the movement frame images as input yielded consistently and significantly better accuracy compared to the neural networks that used the maxframe images. This is clearly shown by the accuracy of CNN 3 and CNN 4 in Table 2. When using both the movement frame and maxframe as input into a single neural network, CNN5, it performed better than the maxframe input

networks, CNN3, but worse than the movement frame input networks, CNN 4. It was also discovered that the neural network could be significantly reduced in size and capacity and still achieve a very similar level of accuracy, as seen by comparing CNN 4 and CNN 6. CNN 4 had ~6,500,000 trainable parameters while CNN 6 had ~6,850 trainable parameters, a nearly 950 times decrease in size while still maintaining an ~ 98% accuracy rate. When comparing the code in Appendix 1 and Appendix 2 for CNN 1 and CNN 6 listed in Table 2, it should be noted that there are differences in the preprocessing steps. The input data for CNN 1 is only normalised to be between 0 and 1, it is not centered on 0 between -1 and 1, the distribution in the images is also not standardised like it is for the data used for CNN 6. However the accuracy of both networks is nearly the same and as such it appears that not all the preprocessing steps are necessary. Reducing the amount of preprocessing done could improve computational speed but likely not significantly as the biggest computational cos occurs from all the operations performed by the neural network itself. It is important to note that the same preprocessing steps must be used on images being evaluated, as the ones used for training the neural network.

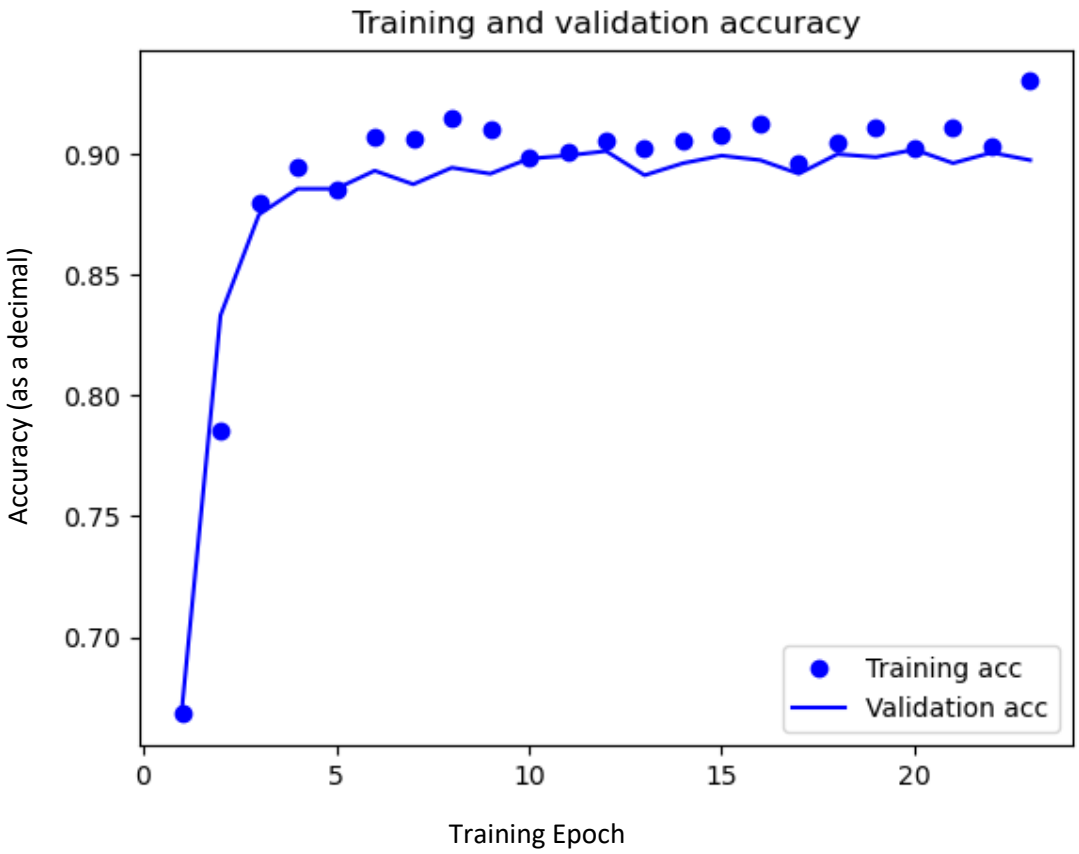



Figure 16 (performance of CNN 2)

CNN 2 only achieved an accuracy of ~90%, which is a result of the dataset used for this neural network containing images that were labelled as both a confirmed and rejected detections. However it displays a good example of a well fit network as there is no overfitting or underfitting, which is evident from how closely the training and validation accuracy follow each other as shown in Figure 16.

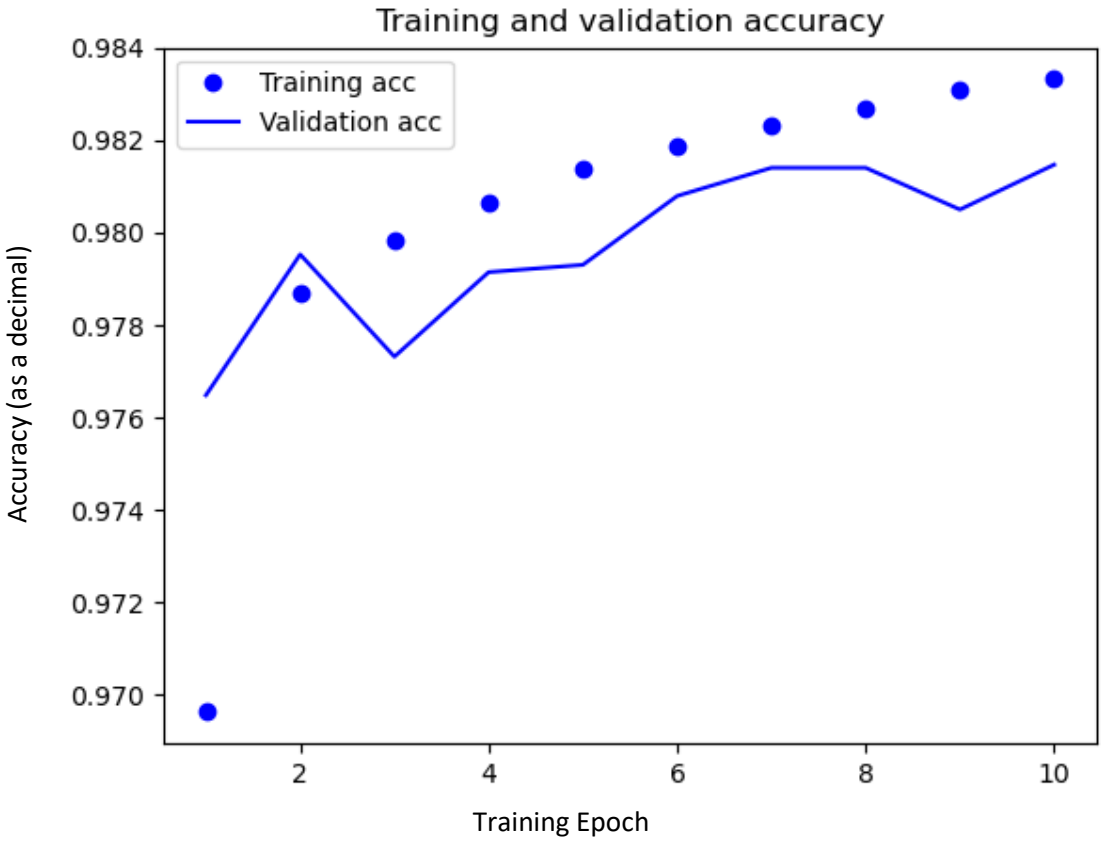



Figure 17 (performance of CNN 6)

CNN 6 overall performed quite well achieving an accuracy of ~98%. While the training accuracy is slightly higher than the validation accuracy about 0.2% this is to be expected and both the training and validation accuracy are increasing at a similar rate as the neural network completed more training epochs as shown in Figure 17. This clearly demonstrates that this neural network is not overfitting to the training data.

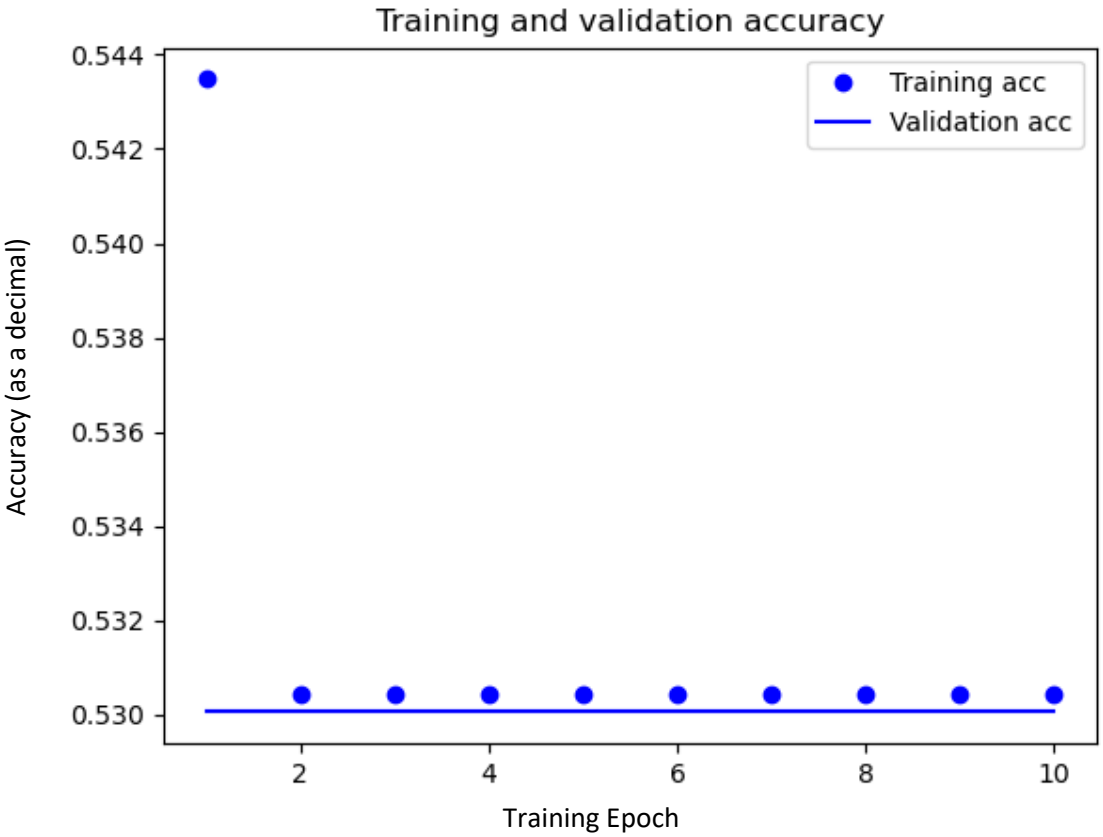



Figure 18 (performance of CNN 7)

The consistent lack of change in accuracy of CNN 7 as shown in Figure 18 indicates that the neural network has failed to learn anything, in this case for an unknown reason.

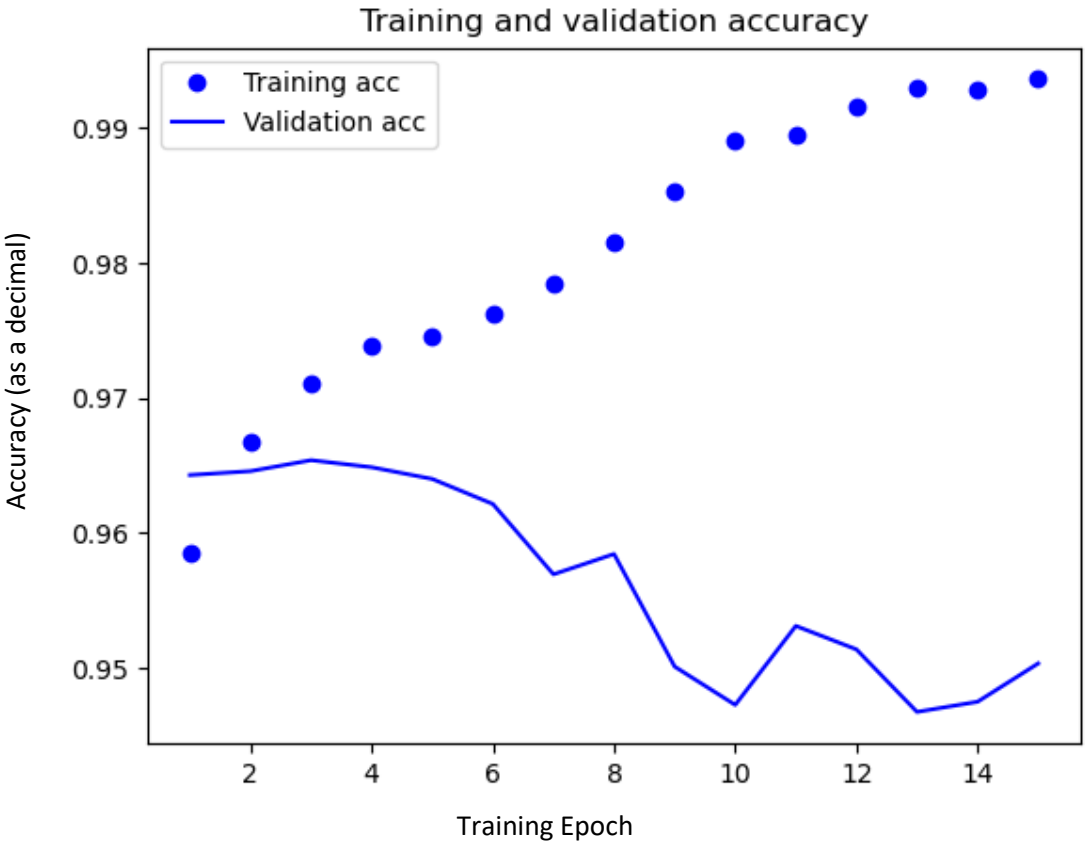



Figure 19 (performance of CNN 5)

It's from Figure 19 that using both the maxframe and movement as two inputs does not improve the accuracy of the neural network. Changing the parameters of CNN 5, including the number of layers, size of layers, combination method(Concatenate vs Average, etc.) yielded no improvement in the accuracy of the neural network.

Table 3 (Confusion Matrix for CNN 6)

| | Actual Positive | Actual Negative |
|---|---|---|
| Predicted Positive | 116262 | 2959 |
| Predicted Negative | 1118 | 100981 |

Table 4 (Recall and Precision of CNN 6)

| | |
|---|---|
| Precision | 0.9905 |
| Recall | 0.9752 |

The values for precision and recall in Table 4 are calculated using the equations 2.2 and 2.3 respectively using the numbers in Table 3. As both recall and precision for CNN 6 are extremely high as shown in Table 4 it indicates, that the neural network does not suffer from any bias in the dataset.

## 4.2 Discussion

A major flaw that occurred within this project is that the datasets used should've been split into three separate datasets, one for training, one for validating and one for testing. For this project the dataset was only split into two, one for training and one for validating. As changes were made to the structure of the neural networks, the additional testing dataset would have been used to test if the neural network had been inadvertently optimised to work on just the data in the training and validation datasets. Other methods apart from splitting the dataset into three could have also been implemented but they were not. Further testing on new data would be required to ensure that neural network has not been inadvertently optimised for the training and validation datasets

While the highest accuracy achieved out of any of the networks is about 98%, the accuracy could potentially be improved. There may be a small amount images in the dataset that are incorrectly labelled. This could be easily investigated by manually checking any images that are being incorrectly classified by the neural network and any images that are falling in the mid-range of the network's predictions, the 0.4—0.6 range.

If given more time for further experimentation, it may be possible to shrink the neural network even further and still achieve similar accuracy levels. If the network is shrank sufficiently it may also become small enough that some manual analysis could be done on the neural network and a simpler algorithm devised from it. In particular the convolution masks devised by the Convolution Layers may provide insight into what features are being used by the neural network to differentiate between the images.

Being able to reduce the size of the neural network used is incredibly useful for the GMN. This is because the GMN uses small recording stations that use Rasperry Pi computers to perform some of the initial data processing including meteor detection. This is done to reduce the amount of data that needs to be transmitted to the central server and also the amount of work that needs to be done by the central server. By reducing the size of the neural network sufficiently it may be able to run efficiently and fast enough to be used on the Raspberry Pi computers.

The exact reason for CNN 7's failure is unknown. The consistent lack of change in accuracy indicates a computational or mathematical error occurring somewhere, for example there may be a division by zero occurring somewhere. Some of the images with the +5 pixel border are incredibly small and these may be causing the issue. Some of these small images may be of clouds and as such they may contain only 1 or 2 different pixel values in the whole image. If the image only has 1 pixel intensity throughout the entire image, the standard deviation will be 0 and thus when preprocessing the image there will be a divide by 0. This could be avoided by checking the standard deviation of each image before dividing it by its standard deviation during the standardisation step. Another possible explanation is that some of the rejected detections may have coordinates in the detections txt file that are either completely horizontal or vertical. When these are cropped from the image they would be a narrow bright line of pixels and then black pixels would be added around the detection to make it square, making it appear as a bright line in the centre of the image exactly like a meteor(confirmed detection) would thus causing immense confusion for the neural network. This second explanation is less likely. CNN 2 had images that were listed as both confirmed and rejected detections but the accuracy of the neural network was changing over each epoch as seen in Figure 16, instead of staying constant like in Figure 18.

Including the time information, as the maxframe images, hampered the performance of CNN 3 and CNN 5 as compared to any of the other neural networks. This likely occurred as the images were

made square by adding columns and rows of black pixels to the image as needed. For the movement frames this made sense as many of the images were already black around the edges. When using the maxframe images, these images appeared quite noisy around the edges as a result of the compression used and adding these black columns and rows drastically changed the composition of these images.

# Chapter 5: Conclusion

In conclusion it was found that it was possible to achieve nearly the same level of accuracy on the GMN data that had been achieved previously on the CAMS data, 98% and 99.7% respectively. Significantly these results were achieved using a neural network that that was 950 times smaller than the neural network used on the CAMS dataset. It was also discovered that an initial minimum image size was required before stretching the image in order for the neural network to be able to perform correctly. Other aspects of the Peter Gural report were also reconfirmed in that including the time information of the detections as part of the input did not significantly improve the accuracy of the neural network and in this case even made it worse.

Additional experimentation could be performed to investigate if the neural networks could be made even smaller while still maintaining the same level of accuracy. This includes investigating if the pre-processing steps are the cause for the poor performance of CNN 7 which used the images with the smaller borders. Investigation of the operations of the small neural networks like CNN 6, in particular the convolution masks used in the Convolution layer, could yield insight into what features the neural network is using to differentiate between the images of meteors and non-meteors. This could then be used to devise a simpler algorithm to take the place of a more computationally costly neural network.

To implement any of these Convolution Neural Networks into the GMN data processing pipeline would require testing to see if the neural network is small enough to be able to be computed on the Raspberry Pis that are used. Testing would also be required to see if the neural network maintains its accuracy when used on data that comes from GMN camera stations that did not have data included in the training dataset. Before implementing the neural network into the processing pipeline the training dataset should also be checked for any detections that have been misclassified. While doing this for the nearly 90,000 detections is a large task, this could be reduced to ~2000 detections by simply looking at the original non-augmented images that are being misclassified by the desired neural network.

## References


1. https://dictionary.cambridge.org/dictionary/english/meteor
2. https://globalmeteornetwork.org/
3. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 4
4. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 4
5. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 5
6. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 6-8
7. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 7
8. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 8
9. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 9
10. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 20-22
11. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 94
12. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 30
13. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 97
14. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 48-49
15. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 104
16. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 107
17. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 109
18. Nitish Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research,* 15 (2014), 1929-1958
19. EE425 – Image Processing & Analysis, Baseline IPA Techniques, © 2021 Paul F Whelan. Paul F. Whelan (2019), "Image Processing & Analysis - EE425", Vision Systems Group, Dublin City University, 2019.
20. François Chollet, Deep Learning with Python, (Manning Publications Co., 2018), p. 127-129
21. Peter S. Gural, "Deep learning algorithms applied to the classification of video meteor detections", *Monthly Notices of the Royal Astronomical Society,* 489 (2019), 5109-5118 (1)
22. Peter S. Gural, "Using Deep Learning to Automate Meteor Confirmation in the CAMS Image Processing Pipeline", UWO Weekly Meeting CS 795 Project for Spring 2019, powerpoint presentation
23. Peter S. Gural, "Deep learning algorithms applied to the classification of video meteor detections", *Monthly Notices of the Royal Astronomical Society,* 489 (2019), 5109-5118 (7)
24. Peter S. Gural, "Deep learning algorithms applied to the classification of video meteor detections", *Monthly Notices of the Royal Astronomical Society,* 489 (2019), 5109-5118 (7)
25. Peter S. Gural, "Applying Deep Learning to RMS Meteor Classification – A First Look", GMN meeting February 2021, powerpoint presentation
26. Esther Gavin, "Convolutional Neural Network Model to Classify Meteor Images", *(Undergraduate Thesis, Technological University Dublin, 2021),* p. 3
27. Gural P. S. (2011). "The California All-sky Meteor Surveillance (CAMS) System", *In Proceedings of the International Meteor Conference, Armagh, Northern Ireland*, *IMO,* (16-19 September, 2010), p. 28-31.
28. Vida, D., Zubović, D., Šegon, D., Gural, P., & Cupec, R., "Open-source meteor detection software for low-cost single-board computers", *In Proceedings of the International Meteor Conference (IMC2016), Egmond, The Netherlands,* (2016), p. 2-5
29. Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization", *Published as a conference paper at ICLR 2015,* (2015)

**Appendix 1 – Risk Assessment**

No risk assessment required as it was a purely computational project

**Appendix 2 – Code**

All code used as well of graphs of results of neural networks and some additional results is available at https://github.com/fiachraf/meteorml However the most important pieces of code are listed below

**Crop_and_convert.py – python script for indexing through provided dataset and creating png files of the detections in the fits files using information from the provided text files.**

```
from astropy.io import fits
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import math
import numpy as np
import os
import sys
import fiachra_python_logger as logger
import datetime
from PIL import Image
import traceback
import blackfill
from pathlib import Path
from time import sleep, perf_counter

path_root = Path(__file__).parents[1]
print(f"path_root: {path_root}/RMS")
sys.path.insert(1, str(path_root) + "/RMS")
#sys path needs to be changed depending on machine or just have RMS added properly as a package
# sys.path.insert(1, "/home/fiachra/atom_projects/meteorml/RMS")
from RMS.Formats import FFfile
from RMS.Formats import FTPdetectinfo
#file paths handled by input from user
#to get current working directory
#home_dir = os.getcwd()
#log command: (custom logger, just wanted something simple that I could look back on afterwards,
could easily be changed out or removed)
#log(log_file_name="log.csv", log_priority, log_type, logger_call_no, details,
log_time=datetime.datetime.now()):
print("need to change script to choose image type you want produced, detect only frames,
maxframe or maxframe time sliced?")
cwd = input("directory path which has ConfirmedFiles and RejectedFiles folders: ")  #Meteor_Files
directory path e.g. ~/Downloads/Meteor_Files
time_start = perf_counter()
os.chdir(cwd)
```

```
output_dir = input("output directory name for the pngs e.g. 20210104_pngs : ")
home_dir = os.getcwd()  #Meteor_Files directory
print(home_dir)
#gets a list of all the files available and then later as each file is analysed it will be removed from the list. At the end of the script any .fits files left in the list
fits_not_analysed = blackfill.search_dirs("", chosen_dir=home_dir, file_or_folder="file", search_subdirs=True)
top_directories = os.listdir()  #should be ConfirmedFiles and RejectedFiles folders amd Empty directories too
print(f"top_directories: {top_directories}")
#maybe add line for input to confirm that user has inputted the correct directory, otherwise get them to cancel the script
#creating new directories for the png versions of ConfirmedFiles and RejectedFiles
if output_dir not in top_directories:
    try:
        os.mkdir(output_dir)
    except:
        pass
    # may cause erros that it doesn't check if ConfirmedFiles_png or RejectedFiles_png directories exist
    try:
        os.mkdir(output_dir + "/" + "ConfirmedFiles_png")
    except:
        pass
    try:
        os.mkdir(output_dir + "/" + "RejectedFiles_png")
    except:
        pass
fits_dont_exist = []
# fits_not_analysed = []
#go through the folders in the top_directories and then only go through the ConfirmedFiles and RejectedFiles folders
for dir_name in top_directories:
    os.chdir(home_dir)
    top_dirs = os.getcwd()
    try:
        if dir_name == "ConfirmedFiles" or dir_name == "RejectedFiles" and os.path.isdir(dir_name) == True:    #if not directory containing empty directories
            os.chdir(home_dir +"/" + dir_name)    #now in either ConfirmedFiles or RejectedFiles
            Con_or_Rej_dir = os.getcwd()
            print("cwd1=", Con_or_Rej_dir)
            detection_folder_list = os.listdir() #should be BE0001....5, BE0001....6, etc.
            try:
                for night_dir in detection_folder_list:   #night_dir is a folder which is a single night of detections from an operating station
                    try:
                        print("\n")
                        # print("cwd2", cwd)
                        os.chdir(home_dir + "/" + dir_name + "/" + night_dir)
                        cwd = os.getcwd()
```

```
                print("cwd3", cwd)
                files_list = os.listdir()   #individual files in the detection folder, e.g. BE0001....fits,
FTPdetectinfo....txt etc.
                #list of fits files that are not anlaysed as they dont appear in the FTPdetectinfo file
                # all files are added to the fits_not_analysed list so that they can be removed as they
are analysed
                # for element in files_list:
                #     fits_not_analysed.append((element, cwd)) #this is the fastest method to make an
actual copy, other methods exist for different scenarios when the list objects aren't strings
                #elements from list are removed later
                #as some folders have multiple FTPdetectinfo files, need to sort through them and pick
correct one
                FTPdetect_list = []
                for item in files_list:
                   #search through files for FTPdetectinfo file, might be multiple files or file named
differently
                   if item.find("FTPdetectinfo") != -1:
                      FTPdetect_list.append(item)
                if len(FTPdetect_list) == 0:
                   print("no FTPdetectinfo file found")
                   sleep(2)
                   logger.log(home_dir + "/log.csv", log_priority = "High", log_type = "FTPdetectinfo File
not found", logger_call_no = 1, details = f"failed to find FTPdetectinfo file in dir: {cwd}")
                else:
                   chosen_FTP_file = ""
                   print("\n")
                   for index, file in enumerate(FTPdetect_list):
                      # if an FTPdetectinfo file name in the folder matches the naming scheme (2 letters
at the start, nothing like pre-confiramtion appended to the end) then it will automatically choose
that FTPdetectinfo_ file
                      if file == "FTPdetectinfo_" + night_dir + ".txt":
                         chosen_FTP_file = file
                         print(f"13, FTPdetect_list: {FTPdetect_list}")
                         print(f"27, chosen_FTP_file: {chosen_FTP_file}")
                   #if no FTPdetectinfo auto chosen then, print options available and have the user
choose the correct one
                   if chosen_FTP_file == "":
                      print("\n choose best FTPdetectinfo file")
                      for possible_file_index, possible_file in enumerate(FTPdetect_list):
                         print(f"file_name: {possible_file}, index: {possible_file_index}")
                      #take first input and if bad input (empty, not a number, out of range) is inputted
then prompt user again
                      while True:
                         try:
                            chosen_FTP_file_index = int(input("please enter index of FTPdetectinfo file to
be used to pick out meteors from the image: "))
                            if chosen_FTP_file_index >= 0 and chosen_FTP_file_index <
len(FTPdetect_list):
                               chosen_FTP_file = FTPdetect_list[chosen_FTP_file_index]
                               break
```

```
                    except Exception as error_4:
                        print("non number value entered, try again")
                #read data from FTPdetectinfo file
                FTP_file = FTPdetectinfo.readFTPdetectinfo(cwd, chosen_FTP_file)
                #loop through each image entry in the FTPdetectinfo file and analyse each image
                for detection_entry in FTP_file:
                    fits_file_name = detection_entry[0]
                    meteor_num = detection_entry[2]
                    if len((find_fits_file :=  blackfill.search_dirs(fits_file_name, chosen_dir=cwd,
file_or_folder="file", exact_match=True, search_subdirs=False))) == 1:
                        # print("test2")
                        square_crop_image = blackfill.crop_detections(detection_entry, cwd)
                        try:
                            fits_not_analysed.remove((cwd, fits_file_name))
                            #put in try statement as, if the same fits is analysed multiple times because of
multiple detections in the same image
                        except:
                            pass
                        #save the Numpy array as a png using PIL
                        im = Image.fromarray(square_crop_image)
                        im = im.convert("L")    #converts to grescale
                        im.save(home_dir + "/" + output_dir + "/" + dir_name + "_png" + "/" +
fits_file_name[:-5] + "_" + str(int(meteor_num)) + ".png")
                    elif len((find_fits_file := blackfill.search_dirs(fits_file_name, chosen_dir=home_dir,
file_or_folder="file", exact_match=True, search_subdirs=True))) == 1:
                        square_crop_image = blackfill.crop_detections(detection_entry,
find_fits_file[0][0])
                        try:
                            fits_not_analysed.remove((find_fits_file[0][0], find_fits_file[0][1]))
                            #put in try statement as, if the same fits is analysed multiple times because of
multiple detections in the same image
                        except:
                            pass
                        # save the Numpy array as a png using PIL
                        im = Image.fromarray(square_crop_image)
                        im = im.convert("L")    #converts to grescale
                        im.save(home_dir + "/" + output_dir + "/" + dir_name + "_png" + "/" +
fits_file_name[:-5] + "_" + str(int(meteor_num)) + ".png")
                    else:
                        fits_dont_exist.append((fits_file_name, cwd))
                    #don't think is necessary and will only fill up the log file, would be necessary if
script doesn't finish and needs to be run again but would need to add additional code to have the
script pick up where it had left off
                    #logging files that have been analysed
                    # logger.log(home_dir + "/log.csv", log_priority = "Low", log_type = "File
processed", logger_call_no = 3, details = f"processed file:{fits_file_name}  in dir :{cwd}")
                #if the file isn't found in the cwd then search for the file in other directories
                    #logging any unanticipated errors that may occur
                    # except Exception as error_5:
                    #     logger.log(home_dir + "/log.csv", log_priority="High", log_type="file analysis
failure", logger_call_no=10, details= traceback.format_exc())
```

```
                    #     print(traceback.format_exc())
                except Exception as error_3:
                    logger.log(home_dir + "/log.csv", log_priority = "High", log_type = "Unknown error",
logger_call_no = 9, details = f"Error: {traceback.format_exc()} - occured in dir: {cwd}")
                    print(f"error_3: {traceback.format_exc()}")
            except Exception as error_2:
                logger.log(home_dir + "/log.csv", log_priority = "High", log_type = "Unknown error",
logger_call_no = 8, details = f"Error: {traceback.format_exc()} - occured in dir: {cwd}")
                print(f"error_2: {traceback.format_exc()}")
        except Exception as error_1:
            logger.log(home_dir + "/log.csv", log_priority = "High", log_type = "Unknown error",
logger_call_no = 7, details = f"Error: {error_1} - occured in dir: {cwd}")
            print(f"error_1: {traceback.format_exc()}")
for item in fits_not_analysed:
    if item[1][-4:] == "fits":
        logger.log(home_dir + "/log.csv", log_priority = "Medium", log_type = "File not analysed",
logger_call_no = 4, details = f"this file: {item[1]} in dir: {item[0]} was not analysed")
for item in fits_dont_exist:
    logger.log(home_dir + "/log.csv", log_priority = "Medium", log_type = "FITS File not found",
logger_call_no = 5, details = f"failed to find FITS file: {item[0]} in dir: {item[1]}")
print("Cropping Script Finished")
logger.log(home_dir + "/log.csv", log_priority = "High", log_type = "Script finished", logger_call_no =
6, details = f"Script has finished")
time_end = perf_counter()
print(f"time: {time_end - time_start}")
```

**blackfill.py – python script that contains functions used in the crop_and_convert.py script**

```
import numpy as np
import math
import os
import sys
import traceback
from pathlib import Path
import matplotlib.pyplot as plt
import matplotlib.patches as patches

path_root = Path(__file__).parents[1]
#print(f"path_root: {path_root}/RMS")
sys.path.insert(1, str(path_root) + "/RMS")

from RMS.Formats import FFfile
from RMS.Formats import FTPdetectinfo

def add_zeros_row(image, top_or_bottom, num_rows_to_add):
    """ adds rows of zeros to either the top or bottom of the numpy array
        if performance is important the same effect can be achieved using numpy slicing which is faster
    """
    image_shape = np.shape(image)
    #shape returns (num_rows, num_cols)
    num_rows = image_shape[0]
```

```
    num_cols = image_shape[1]

    zero_rows = np.zeros((num_rows_to_add, num_cols))

    if top_or_bottom == "top":
        new_image = np.vstack((zero_rows, image))
        return new_image
    elif top_or_bottom == "bottom":
        new_image = np.vstack((image, zero_rows))
        return new_image
    #return None which will cause an error if invalid inputs have been used
    return


def add_zeros_col(image, left_or_right, num__cols_to_add):
    """ adds columns of zeros to either the left or right of the numpy array
        if performance is important the same effect can be achieved using numpy slicing which is faster
    """
    image_shape = np.shape(image)
    #shape returns (num_rows, num_cols)
    num_rows = image_shape[0]
    num_cols = image_shape[1]
    zero_cols = np.zeros((num_rows, num__cols_to_add))


    if left_or_right == "left":
        new_image = np.hstack((zero_cols, image))
        return new_image
    elif left_or_right == "right":
        new_image = np.hstack((image, zero_cols))
        return new_image
    #return None which will cause an error if invalid inputs have been used
    return


def blackfill(image, leftover_top=0, leftover_bottom=0, leftover_left=0, leftover_right=0):
    """ will make the image square by adding rows or columns of black pixels to the image, as the
image needs to be square to be fed into a Convolutional Neural Network(CNN)

    As I am giving the cropped images +20 pixels on all sides from the detection square edges, any
meteors that occur on the edge of the frame in the fits file will not be centered so I am using the
leftover terms, to add columns or rows for the edges that were cut off so that the meteor is
centered in the image. This might badly affect the data and thus the CNN so it might need to be
changed. However I think it might help as the added space around the meteor I hope will make sure
that it includes the entire meteor trail which can hopefully help differentiate true detections from
false detections

    When squaring the image it will also keep the meteor detection in roughly the center

    Resizing of the images to be all the same size will be done in another script using the
keras.preprocessing module
```

```
    Could also do squaring of images using keras.preprocessing module I just thought doing it this
might yield better results as it wont be stretched or distored in a potentially non-uniform way
    """

    if leftover_top > 0:
        image = add_zeros_row(image, "top", leftover_top)
    if leftover_bottom > 0:
        image = add_zeros_row(image, "bottom", leftover_bottom)
    if leftover_left > 0:
        image = add_zeros_col(image, "left", leftover_left)
    if leftover_right > 0:
        image = add_zeros_col(image, "right", leftover_right)

    if np.shape(image)[0] == np.shape(image)[1]:
        new_image = image[:,:]
        return new_image

    if np.shape(image)[0] < np.shape(image)[1]:
        rows_needed = np.shape(image)[1] - np.shape(image)[0]
        image = add_zeros_row(image, "top", math.floor(rows_needed/2))
        image = add_zeros_row(image, "bottom", math.ceil(rows_needed/2))
        new_image = image[:,:]
        return new_image

    if np.shape(image)[1] < np.shape(image)[0]:
        cols_needed = np.shape(image)[0] - np.shape(image)[1]
        image = add_zeros_col(image, "left", math.floor(cols_needed/2))
        image = add_zeros_col(image, "right", math.ceil(cols_needed/2))
        new_image = image[:,:]
        return new_image

    return


def search_dirs(search_term, chosen_dir, file_or_folder="", exact_match=False,
search_subdirs=False): #extension=""):
    """
    searches directory for the search term and returns a list of objects containing the search term
    chosen_dir is the directory to search
    file_or_folder designates if it should search for files or folders, by default it will search for both
    exact_match designates if only exact matches should be found
    search_subdirs designates if it should recursively search subdirectories
    extension designates if it should search for only files with the matching extension

    Returns list of tuples, index 0 is the directory where the file is located and index 1 is the name of
the file
    """
    matching_entries = []

    if search_subdirs == True:
        for root, dirs, files in os.walk(chosen_dir):
```

```
            if file_or_folder == "" or file_or_folder == "folder":
                for dir in dirs:
                    if exact_match == True:
                        if dir == search_term:
                            matching_entries.append((root, dir))
                    elif exact_match == False:
                        if dir.find(search_term) != -1:
                            matching_entries.append((root, dir))
            if file_or_folder == "" or file_or_folder == "file":
                for file in files:
                    if exact_match == True:
                        if file == search_term:
                            matching_entries.append((root, file))
                    elif exact_match == False:
                        if file.find(search_term) != -1:
                            matching_entries.append((root, file))


    elif search_subdirs == False:
        contents_list = os.listdir(chosen_dir)
        for item in contents_list:
            if exact_match == True:
                if item == search_term:
                    if file_or_folder == "folder":
                        if os.path.isdir(os.path.join(chosen_dir, item)) == True:
                            matching_entries.append(item)
                    elif file_or_folder == "file":
                        if os.path.isfile(os.path.join(chosen_dir, item)) == True:
                            matching_entries.append(item)
                    elif file_or_folder == "":
                        matching_entries.append(item)

            elif exact_match == False:
                if item.find(search_term) != -1:
                    if file_or_folder == "folder":
                        if os.path.isdir(os.path.join(chosen_dir, item)) == True:
                            matching_entries.append((os.getcwd(), item))
                    elif file_or_folder == "file":
                        if os.path.isfile(os.path.join(chosen_dir, item)) == True:
                            matching_entries.append((os.getcwd(), item))
                    elif file_or_folder == "":
                        matching_entries.append((os.getcwd(), item))

    return matching_entries


def crop_detections(detection_info, fits_dir):
    """
    crops the detection from the fits file using the information provided from the FTPdetectinfo files
```

```
    detection_info is a single element of the list returned by the
RMS.RMS.Formats.FTPdetectinfo.readFTPdetectinfo() function. This list contains only information on
a single detection
    fits_dir is the the directory where the fits file is located

    returns the cropped image as a Numpy array
    """

    fits_file_name = detection_info[0]
    meteor_num = detection_info[2]
    num_segments = detection_info[3]
    first_frame_info = detection_info[11][0]
    first_frame_no = first_frame_info[1]
    last_frame_info = detection_info[11][-1]
    last_frame_no = last_frame_info[1]

    try:
        #read the fits_file
        # print(f"fits_dir: {fits_dir}\nfits_file_name: {fits_file_name}")
        fits_file = FFfile.read(fits_dir, fits_file_name, fmt="fits")
        #image array with background set to 0 so detections stand out more
        #TODO inlcude code to use mask for the camera, currently masks not available on the data
given to me, Fiachra Feehilly (2021)
        detect_only = fits_file.maxpixel - fits_file.avepixel
        #set image to only include frames where detection occurs, reduces likelihood that there will
then be multiple detections in the same cropped image
        detect_only_frames = FFfile.selectFFFrames(detect_only, fits_file, first_frame_no,
last_frame_no)

        #get size of the image
        row_size = detect_only_frames.shape[0]
        col_size = detect_only_frames.shape[1]

        #side 1, 2 are the left and right sides but still need to determine which is which
        # left side will be the lesser value as the value represents column number
        side_1 = first_frame_info[2]
        side_2 = last_frame_info[2]
        if side_1 > side_2:
            right_side = math.ceil(side_1) + 1 #rounds up and adds 1 to deal with Python slicing so that it
includes everything rather than cutting off the last column
            left_side = math.floor(side_2)
        else:
            left_side = math.floor(side_1)
            right_side = math.ceil(side_2) + 1
        #side 3 and 4 are the top and bottom sides but still need to determine which is which
        # bottom side will be the higher value as the value represents the row number
        side_3 = first_frame_info[3]
        side_4 = last_frame_info[3]
        if side_3 > side_4:
            bottom_side = math.ceil(side_3) + 1
            top_side = math.floor(side_4)
```

```
        else:
            top_side = math.floor(side_3)
            bottom_side = math.ceil(side_4) + 1

        #add some space around the meteor detection so that its not touching the edges
        #leftover terms need to be set to 0 outside if statements otherwise they wont be set if there's
nothing left over which will cause an error with the blackfill.blackfill() line
        left_side = left_side - 30
        leftover_left = 0
        if left_side < 0:
            #this will be used later to determine how to fill in the rest of the image to make it square but
also have the meteor centered in the image
            leftover_left = 0 - left_side
            left_side = 0

        right_side = right_side + 30
        leftover_right = 0
        if right_side > col_size:
            leftover_right = right_side - col_size
            right_side = col_size

        top_side = top_side - 30
        leftover_top = 0
        if top_side < 0:
            leftover_top = 0 - top_side
            top_side = 0

        bottom_side = bottom_side + 30
        leftover_bottom = 0
        if bottom_side > row_size:
            leftover_bottom = bottom_side - row_size
            bottom_side = row_size


        #get cropped image of the meteor detection
        #first index set is for row selection, second index set is for column selection
        crop_image = detect_only_frames[top_side:bottom_side, left_side:right_side]
        square_crop_image = blackfill(crop_image, leftover_top, leftover_bottom, leftover_left,
leftover_right)
        #square_crop_image = crop_image
        # #this bit is only needed to plot the image for visual demonstrations
        # #-----------------------------------------------------------------
        # #create plot of meteor_image and crop_image
        # # has full image displayed on left side and then the can do two cropped images displayed on
the right side
        # fig, axd = plt.subplot_mosaic([['big_image', 'big_image', 'crop_image'],
        #                                ['big_image', 'big_image', 'small_image_2']],
        #                               )
        #
        # axd['big_image'].imshow(detect_only_frames)
        # axd['crop_image'].imshow(crop_image)
```

```
        # axd['small_image_2'].imshow(square_crop_image)
        #
        # # Create a Rectangle patch Rectangle((left column, top row), column width, row height,
linewidth, edgecolor, facecolor)
        # rect = patches.Rectangle((left_side, top_side), (right_side - left_side), (bottom_side -
top_side), linewidth=1, edgecolor='r', facecolor='none')
        #
        # # Add the patch to the big image
        # axd['big_image'].add_patch(rect)
        #
        # # change the size of the figure
        # fig.set_size_inches(18.5, 10.5)
        #
        # # display the plot
        # plt.show()
        #
        # #----------------------------------------------------------------------

        return square_crop_image

    except Exception as error:
        print(f"error: {traceback.format_exc()}")
        return None

def crop_detections_maxframe(detection_info, fits_dir, time_slice=False):
    """
    crops the detection from the fits file maxpixel image using the spatial information provided from
the FTPdetectinfo files
    detection_info is a single element of the list returned by the
RMS.RMS.Formats.FTPdetectinfo.readFTPdetectinfo() function. This list contains only information on
a single detection
    fits_dir is the the directory where the fits file is located
    time_slice says whether to set all maxpixel pixels to 0 if they're not in one of the time frames. Set
false to do no time slicing E.g for a meteor detection in frames 50 to 60, time_slice = False just
includes all the pixels in the spatial bounding box, time_slice = True will set all values < 50 and values
>60 to 0.
    returns the cropped image as a Numpy array
    """

    #print("should change blackfill to a gaussian noise fill or solid fill of average intensity")
    fits_file_name = detection_info[0]
    meteor_num = detection_info[2]
    num_segments = detection_info[3]
    first_frame_info = detection_info[11][0]
    first_frame_no = first_frame_info[1]
    last_frame_info = detection_info[11][-1]
    last_frame_no = last_frame_info[1]

    try:
        #read the fits_file
        # print(f"fits_dir: {fits_dir}\nfits_file_name: {fits_file_name}")
```

```
    fits_file = FFfile.read(fits_dir, fits_file_name, fmt="fits")
    #image array with background set to 0 so detections stand out more
    #TODO inlcude code to use mask for the camera, currently masks not available on the data given to me, Fiachra Feehilly (2021)
    max_frame_image = fits_file.maxframe
    if time_slice == True:
        #find way to set values in numpy array to 0 if below or above certain value
        #set image to only include frames where detection occurs, reduces likelihood that there will then be multiple detections in the same cropped image
        max_frame_image = FFfile.selectFFFrames(max_frame_image, fits_file, first_frame_no, last_frame_no)

    #get size of the image
    row_size = max_frame_image.shape[0]
    col_size = max_frame_image.shape[1]

    #side 1, 2 are the left and right sides but still need to determine which is which
    # left side will be the lesser value as the value represents column number
    side_1 = first_frame_info[2]
    side_2 = last_frame_info[2]
    if side_1 > side_2:
        right_side = math.ceil(side_1) + 1 #rounds up and adds 1 to deal with Python slicing so that it includes everything rather than cutting off the last column
        left_side = math.floor(side_2)
    else:
        left_side = math.floor(side_1)
        right_side = math.ceil(side_2) + 1
    #side 3 and 4 are the top and bottom sides but still need to determine which is which
    # bottom side will be the higher value as the value represents the row number
    side_3 = first_frame_info[3]
    side_4 = last_frame_info[3]
    if side_3 > side_4:
        bottom_side = math.ceil(side_3) + 1
        top_side = math.floor(side_4)
    else:
        top_side = math.floor(side_3)
        bottom_side = math.ceil(side_4) + 1

    #add some space around the meteor detection so that its not touching the edges
    #leftover terms need to be set to 0 outside if statements otherwise they wont be set if there's nothing left over which will cause an error with the blackfill.blackfill() line
    left_side = left_side - 20
    leftover_left = 0
    if left_side < 0:
        #this will be used later to determine how to fill in the rest of the image to make it square but also have the meteor centered in the image
        leftover_left = 0 - left_side
        left_side = 0

    right_side = right_side + 20
    leftover_right = 0
```

```
    if right_side > col_size:
        leftover_right = right_side - col_size
        right_side = col_size

    top_side = top_side - 20
    leftover_top = 0
    if top_side < 0:
        leftover_top = 0 - top_side
        top_side = 0

    bottom_side = bottom_side + 20
    leftover_bottom = 0
    if bottom_side > row_size:
        leftover_bottom = bottom_side - row_size
        bottom_side = row_size


    #get cropped image of the meteor detection
    #first index set is for row selection, second index set is for column selection
    crop_image = max_frame_image[top_side:bottom_side, left_side:right_side]
    square_crop_image = blackfill(crop_image, leftover_top, leftover_bottom, leftover_left,
leftover_right)
    #square_crop_image = crop_image
    # #this bit is only needed to plot the image for visual demonstrations
    # #-----------------------------------------------------------------
    # #create plot of meteor_image and crop_image
    # # has full image displayed on left side and then the can do two cropped images displayed on
the right side
    # fig, axd = plt.subplot_mosaic([['big_image', 'big_image', 'crop_image'],
    #                                ['big_image', 'big_image', 'small_image_2']],
    #                               )
    #
    # axd['big_image'].imshow(max_frame_image)
    # axd['crop_image'].imshow(crop_image)
    # axd['small_image_2'].imshow(square_crop_image)
    #
    # # Create a Rectangle patch Rectangle((left column, top row), column width, row height,
linewidth, edgecolor, facecolor)
    # rect = patches.Rectangle((left_side, top_side), (right_side - left_side), (bottom_side -
top_side), linewidth=1, edgecolor='r', facecolor='none')
    #
    # # Add the patch to the big image
    # axd['big_image'].add_patch(rect)
    #
    # # change the size of the figure
    # fig.set_size_inches(18.5, 10.5)
    #
    # # display the plot
    # plt.show()
    #
    # #-----------------------------------------------------------------
```

```
        return square_crop_image

    except Exception as error:
        print(f"error: {traceback.format_exc()}")
        return None
```

**CNN_script_20220220_4.py – code used to create Keras neural network**

```
#ensures that tensorflow is used as the backend
import tensorflow as tf
#tf.compact.v1.keras.backend.set_session()
tf.config.threading.set_intra_op_parallelism_threads(16)
tf.config.threading.set_inter_op_parallelism_threads(16)

#from keras.backend.tensorflow_backend import set_session
#import  tf.keras.backend.tensorflow_backend as K
#taken from Francis Chollet - Deep Learning with Python, starting page 147
#import keras
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import models

#noramlisaton functions, can be done using keras preprocessinglayers which are present in newer versions of tensorflow, I am using tensorflow 2.4.1 I had a reson for this specific verions but I just can't remember
def normalise_me(image, label):
    image = tf.cast(image/255., tf.float32)
    return image, label

def center_me(image, label):
    #image is really a batch of images and so axis=[1,2,3] and keepdims=True is needed so that it gets the mean and standard deviation for each individula image rather than across the batch
    mean_1 = tf.math.reduce_mean(image, axis=[1,2,3], keepdims=True)
    image = image - mean_1
    return image, label

def standardise_me(image, label):
    std_val = tf.math.reduce_std(image, axis=[1,2,3], keepdims=True)
    image = image / std_val
    return image, label

def rescale_2(image, label):
    max_val =tf.math.reduce_max(tf.math.abs(image), axis=[1,2,3], keepdims=True)
    image = image / max_val
    return image, label


#keras preprocessing, will resize images etc.
# tf.keras.preprocessing.image_dataset_from_directory(
```

```
#     directory,
#     labels="inferred",
#     label_mode="int",
#     class_names=None,
#     color_mode="rgb",
#     batch_size=32,
#     image_size=(256, 256),
#     shuffle=True,
#     seed=None,
#     validation_split=None,
#     subset=None,
#     interpolation="bilinear",
#     follow_links=False,
#     crop_to_aspect_ratio=False,
#     **kwargs
# )
data_dir = input("directory that contains solely Confirmed and Rejected folders, labelled as 1 and 0 respectively: ")
output_model_name = input("model name that you want to give it e.g. meteorml_20220201 (do not include the .5): ")


#keras preprocessing, will resize images etc. create training and validation datasets
train_data = keras.preprocessing.image_dataset_from_directory(
    data_dir,
    labels="inferred",
    label_mode="binary",
    class_names=None,
    color_mode="grayscale",
    batch_size=32,
    image_size=(32, 32),
    shuffle=True,
    seed=169,
    validation_split=0.2,
    subset="training",
    interpolation="bilinear",
)


#need to create new datasets like this as it doesn't seem to work properly if the dataset is modified in place for some reason
train_norm = train_data.map(normalise_me)
train_cent = train_norm.map(center_me)
train_stan = train_cent.map(standardise_me)
train_resc2 = train_stan.map(rescale_2)


val_data = keras.preprocessing.image_dataset_from_directory(
    data_dir,
    labels="inferred",
    label_mode="binary",
```

```
    class_names=None,
    color_mode="grayscale",
    batch_size=32,
    image_size=(32, 32),
    shuffle=True,
    seed=169,
    validation_split=0.2,
    subset="validation",
    interpolation="bilinear",
)
val_norm = val_data.map(normalise_me)
val_cent = val_norm.map(center_me)
val_stan = val_cent.map(standardise_me)
val_resc2 = val_stan.map(rescale_2)

#sample netowrk, first layer has to include an argument for input shape
# model = models.Sequential()
# model.add(layers.Conv2D(32, (3, 3), activation="relu", input_shape=(150, 150, 3)))
# model.add(layers.MaxPooling2D((2,2)))
# model.add(layers.Conv2D(64, (3,3), activation="relu"))
# model.add(layers.MaxPooling2D((2,2)))
# model.add(layers.Conv2D(128, (3,3), activation="relu"))
# model.add(layers.MaxPooling2D((2,2)))
# model.add(layers.Conv2D(128, (3,3), activation="relu"))
# model.add(layers.MaxPooling2D((2,2)))
# model.add(layers.Flatten())
# model.add(layers.Dense(512, activation="relu"))
# model.add(layers.Dense(1, activation="sigmoid"))

#prints a summary of the model
#model.summary

#import matplotlib.pyplot as plt
#
#for image_1_batch, label_batch in train_cent:
#    print("test")
#    print(tf.shape(image_1_batch), tf.shape(image_2_batch), tf.shape(label_batch))
#    for image_1, image_2, label in zip(image_1_batch, image_2_batch, label_batch):
#        # print(tf.shape(image_1), tf.shape(image_2), tf.shape(label))
#        print(f"label: {label}")
#        plt.imshow(image_1)
#        plt.show()
#        plt.imshow(image_2)
#        plt.show()
#similar recreation of model used by Peter S. Gural as shown in Table 3 in his paper:
# Deep learning algorithms applied to the classification of video meteor detections, Peter S. Gural,
doi:10.1093/mnras/stz2456
model = models.Sequential()
model.add(layers.Conv2D(16, (5, 5), activation="relu", input_shape=(32,32,1)))
model.add(layers.MaxPooling2D((5,5)))
#model.add(layers.Conv2D(64, (3,3), activation="relu"))
```

```
#model.add(layers.MaxPooling2D((2,2)))
#model.add(layers.Conv2D(64, (3,3), activation="relu"))
#model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation="relu"))
model.add(layers.Dense(1, activation="sigmoid"))


model.summary()

#compile the network
from keras import optimizers

model.compile(loss="binary_crossentropy", optimizer=optimizers.Adam(), metrics=["acc"])



#fit the model to the data
#steps_per_epoch is the number of training steps the code runs before beginning a new epoch, exclude this line to run over the whole dataset for each epoch
history = model.fit(
train_resc2,
# steps_per_epoch=36,
epochs=10,
validation_data=val_resc2)
# validation_steps=36)

model.save(output_model_name + ".h5")


import matplotlib.pyplot as plt

acc = history.history["acc"]
val_acc = history.history["val_acc"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

epochs = range(1, len(acc) +1)

plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.legend()
plt.savefig("accuracy" + output_model_name + ".png")
plt.figure()

plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.savefig("loss" + output_model_name + ".png")
```

```
plt.show()
```