Night-time cloud classification with machine learning

Akin Kucukkurt

Abstract: Cloud classification with different types of machine learning algorithms as already been done many times. This paper aims to apply one of the methods used and present it in an easy to understand way as well as analyse the human judgment side of machine learning samples. Only cloud coverage and cirrus cloud coverage in 1/8's is predicted which is far simpler than the predictions made by other algorithms.(Summary of results)

Acknowledgements

I would like to offer my thanks to Hugh Jones and David Campbell for their help administrating me during this project. I especially thank David Campbell for providing the large classified data set. Finally I wish to thank my friend Orkan Mustafa for helping me with some programming issues.

1. Introduction

(What other papers have done),(Machine learning is only as good as the dataset),(Dealing with large amounts of data)

2. The Data Set

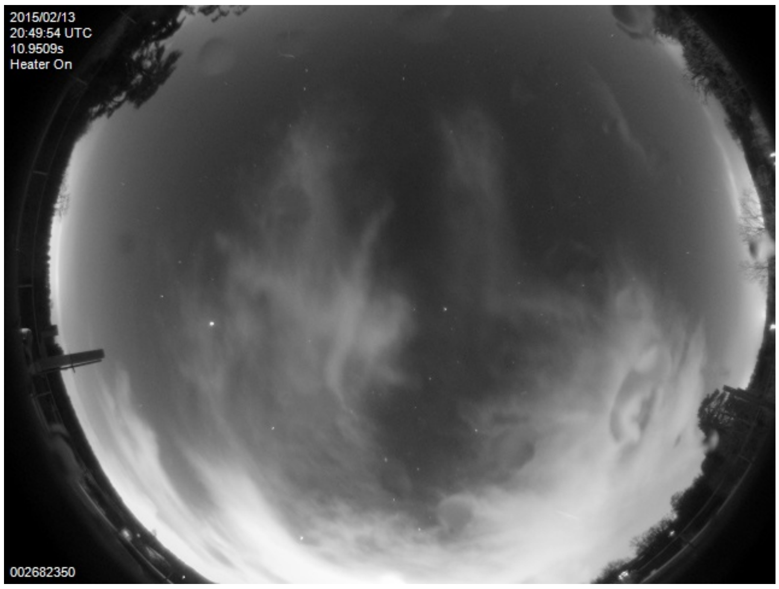
(Do I agree with the assigned values?),(How does the algorithm interpret this dataset?)(Cloud/Cirrus split)

3. The Method

(Summary of the method used)

3.1. Cutting and Transforming the image

The initial image from the sky camera looks like this.



It is important to crop this image to remove features that are not necessary for classification as they might interfere with the algorithm and it will lower computation time for this whole method which is already going to be a few hours due to the number of images I'm going to be using. The land features around the edge fit this description. There is also light pollution around the edges which may look like a cloud to the algorithm. The telescope domes at Bayfordbury cannot depress enough to view the sky towards the outer part of the images anyway so it can be cropped out. First I need to import some modules and define some variables.

```python
import numpy as np #Required for image manipulation as the image pixel values are stored in a numpy
    array, for easy array manipulation
import pandas as pd #Required for an easy way to read data from a csv file
from PIL import Image, ImageDraw #Required to draw circular mask on image as well as open images into
    arrays and save images from an array
import squircle #Required to turn circular image into a square

csvpath=r"C:\Users\Akin\Desktop\AllSky_classified\index_training.csv" #Put in the file path for
    csvfile containing image data in the form as is already there, if only converting one image just
    set this to any string
ipath=r"C:\Users\Akin\Desktop\AllSky_classified\train\50.jpg" #If you are converting one image put in
    the file path for that image in the form as is already there
folderimg=r"C:\Users\Akin\Desktop\AllSky_classified\train" #If you are converting multiple images ,
    put in the path for the folder containing the images in the form as is already shown
sname="result.png" #Put the name that you want your result file to be named to , adding a .jpg at the
    end as is shown. If converting multiple images set this to any string as it is ignored in favour
    of names from the csv file
fileindex="filename_index" #Put in string form the name of the header of the column of the csv file
    containing the names of your images , If doing a single image just put any string here
cdiam=480 #Put in the diameter in pixels for the circle size you want
```

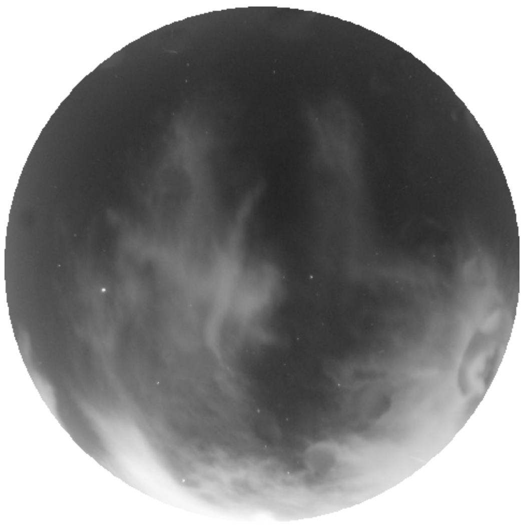
The first step is to then apply a circular mask to crop out the unwanted features.

```python
def Circlecrop(ipath,cdiam,sname): #This function creates and applies a circular mask to image in
    order to make a circular image
   img=Image.open(ipath)
   if cdiam>img.size[0] or cdiam>img.size[1]: #Checks if your circle diameter is larger than the
       width or height of the picture
      return ("Circle diameter bigger than picture width or height")
   else: #Calculation for finding the values needed to create a circle with the diameter that the
       user inputed
      diamc1=img.size[0]-cdiam
      diamc2=img.size[1]-cdiam
      a=int(diamc1/2)
      b=int(diamc2/2)
      c=img.size[0]-(a)
      d=img.size[1]-(b)
   #This creates the circular mask using the values calculated from above (a,b,c,d) and stacks it on
       top of the image and saves it
   npImage=np.array(img)
   alpha = Image.new('L', img.size,0) #Creating the alpha , at the moment its just a black
       rectangle/square
   draw = ImageDraw.Draw(alpha) #Drawing it
   draw.pieslice([a,b,c,d],0,360,fill=255) #Drawing a circle into the alpha mask so its got a circle
       gap on it.
   npAlpha=np.array(alpha)
   npImage=np.dstack((npImage,npAlpha)) #Stack alpha onto the image
   return Image.fromarray(npImage).save(sname) #Save the image , neccessary as further image
       manipulation does not work otherwise
```

The first part of the code defines the values needed to draw a circular mask with the input circular diameter. The second half actually creates and applies the mask. For each function in this code which modifies and creates a new image, I have to save and then load it again for the next function which increases computation time but the program can't process the new image otherwise or at the very least I couldn't make it work without it. I have to turn this image from a rectangular into a square which crops out some the empty space in the masked area and is necessary for the piece of code after this.

```
def Squarecrop(cdiam,sname): #Crops your masked image into a square with the width as the diameter of
    the circle
    npImage=Image.open(sname)
    width, height = npImage.size
    left = (width - cdiam)/2
    top = (height - cdiam)/2
    right = (width + cdiam)/2
    bottom = (height + cdiam)/2
    npImage=npImage.crop((left, top, right, bottom)) #Crops the image using the above information
    return (npImage.save(sname)) #Save the image , neccessary as further image manipulation does not
        work otherwise
```

The code calculates the values needed to crop the width of the rectangular image so that the image becomes a square. This is done such that none of the data is lost, only some masked area is lost. There is no mask area above the top and below the bottom. It then performs the crop using the crop function. The output after the first and second function looks like this. Since the mask area is white you can't tell the difference of the two outputs by eye.

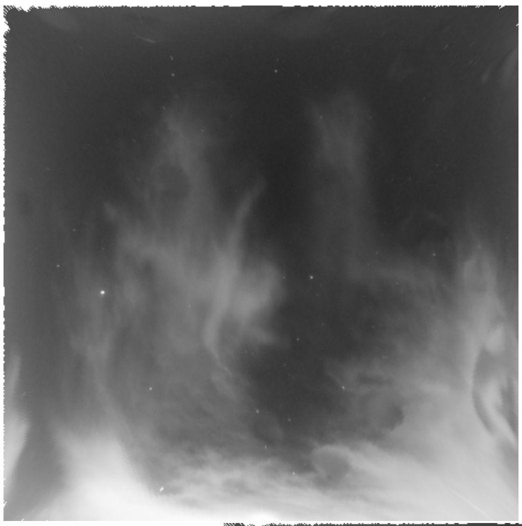
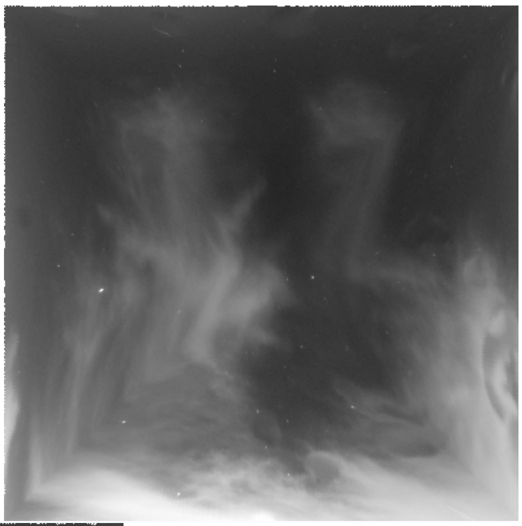


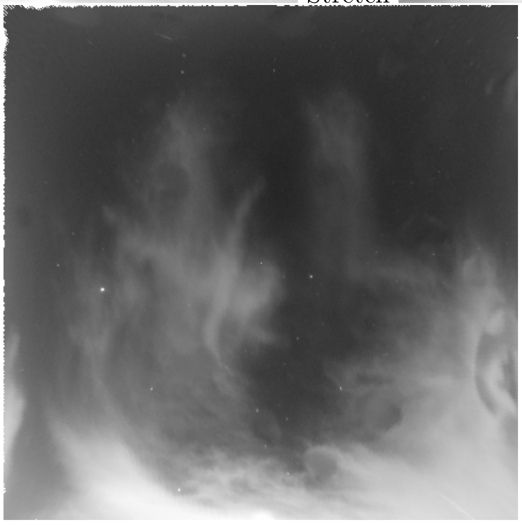
Whilst I could put this straight into the algorithm, The masked area is just empty space and the algorithm's decision trees will be simpler without having to take that and the circular nature of the actual data into account. Ideally I want to turn the actual circular data into a square. In order to do this I use the squircle function. This function only works when the masked image is a square hence the necessity of the second function.

```
def Converter(sname): #Converts your circle containing the circular image into a new square
    image = Image.open(sname)
    image = np.asarray(image)
    converted = squircle.to_square(image,'fgs') #Function which converts the circular image into a
        square
    Image.fromarray(converted).save(sname) #Save the final image
```

The 'FGS' method used is the Fernndez-Guasti squircle method which maps the points such that the area is the same and so it has very little deformation.

Another method is a simple stretch which doesn't preserve area and so deforms the image.The last method is elliptical grid mapping which is a lot better but I thought that FGS looked more accurate.The differences are minor but if you look carefully at some areas you can see them. The images were classified according to their area in circular form and so area needs to preserved in order for the algorithm to learn properly.

FGS  Stretch 

 Elliptical

There is an obvious issue with all three of these images, namely the white pixels around the edge. There are there because the masked image wasn't a perfect circle since it is made up of square pixels. I can get rid of them by recalling the my Squarecrop function and reducing the diameter by 10 which crops out the edge. Also to perform this whole process I need to run all the functions one by one, so I created a master function which would do just that and includes the final crop to get rid of the edge defect. It also runs the iterable version of the whole process for multiple images as well as non iterable for a single image as decided by the 'iterate' variable.

```python
def Master(csvpath,ipath,folderimg,sname,fileindex,cdiam,iterate): #This functions combines the
    functions above in order to give your converted image and iterates to do multiple images if you
    want it to
    if iterate==False: #For single image, uses the above function to create the picture
        Circlecrop(ipath,cdiam,sname)
        Squarecrop(cdiam,sname)
        Converter(sname)
        Squarecrop(cdiam-10,sname)
    elif iterate==True: #For multiple images , reads csv file , iterates through it and performs the
        functions to get the square picture for each iteration
        df = pd.read_csv (csvpath, header=0)
        col_a = list(df.eval(fileindex))
        for i in range(5316,13089):
            sname=str(i)+".png"
            ipath=folderimg+'\\'+str(i)+".jpg"
            Circlecrop(ipath,cdiam,sname)
            Squarecrop(cdiam,sname)
```

```
          Converter(sname)
          Squarecrop(cdiam-10,sname)
Master(csvpath,ipath,folderimg,sname,fileindex,cdiam,iterate)
```

3.2. Creating the Learning Array

(How the images were stored in pandas)(Describing the 2 dimensional array wanted by scikit)(How the pixel values were taken out)

3.3. Training the Algorithm

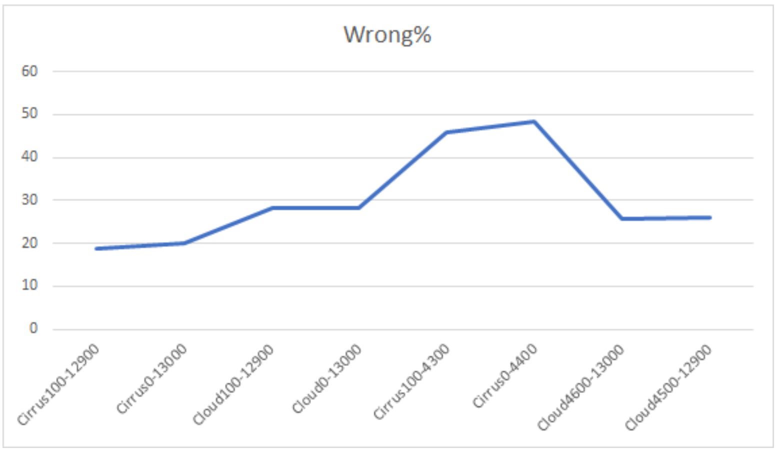
(Mod function)(Slicing and fitting to train algorithm)

3.4. Predicting values

(Mod function)(Slicing and Predicting values)(Analysis of those values)(Moon/other effects)

4. Analysis of Results

The initial and simplest results are as follows.



Test Sample	Wrong%
Cirrus100-12900	18.859375
Cirrus0-13000	19.92307692
Cloud100-12900	28.21875
Cloud0-13000	28.27692308
Cirrus100-4300	45.80952381
Cirrus0-4400	48.27272727
Cloud4600-13000	25.83333333
Cloud4500-12900	25.85714286

The test samples describe which algorithm was used which respond to the 2 columns for the classified values in the data-set.It also describes the range of images that were tested which are a series of 100 images for every interval of 200. So the images that each one was trained on is the opposite series-e.g., tested on 100-12900 means it was trained on 0-13000. The wrong% column describes the percentage of images the algorithm got wrong. This isn't a complete measure as there is no value for the accuracy of the algorithm per image, this is simply the overall accuracy. It is rather misleading and doesn't tell you much how or what the algorithm has learned. However I can use the information about the different subsets that the algorithm has performed on to make some meaningful predictions from this graph.

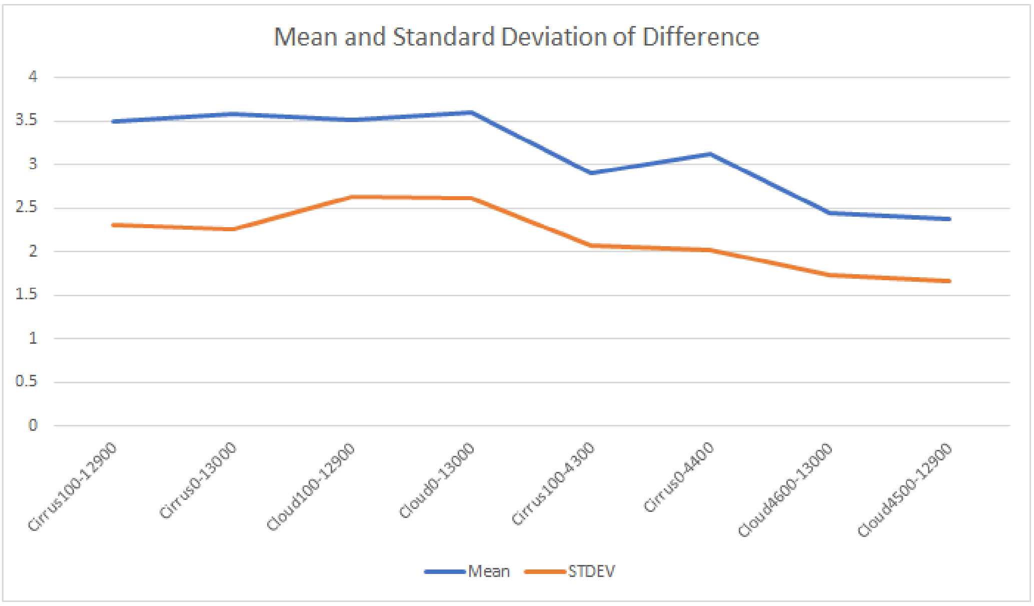
The Cirrus algorithm has a low accuracy when only the images that were in the Cirrus subset are used. Using what I know of this subset I can say that there isn't an obvious systematic issue with it and so the error is mostly due to measurement error which presents itself as over-fitting as the algorithm learns the error instead getting the general relation. Over-fitting essentially means the signal to noise ratio is low. The signal is the brightness of a cloud and so the total cloud/brightness coverage which means both subsets, Cloud and Cirrus, are looking for the same signal. The noise is the variation of the images in the subset which should be lower for the cirrus as subset is it only has images of one type of cloud. I will analyse how true this actually is later on.

Using what I know of the subset and these results I can conclude that the low signal to noise ratio is due to lower number of images. There is a significant difference between the Cirrus100-4300 and the Cirrus0-4400 results, this could be due the differences in the subset they were trained and tested on as well as the difference in the number of images they were trained on as Cirrus100-4300 is trained on 100 more images. The fact that Cloud4600-13000 and Cloud4500-12900 are around twice as accurate and have around twice as much images suggest that the number of images is the dominant factor with the difference in the number of images to accuracy ratio due to the lower variation in the Cirrus subset which decreases noise as well as the differences between the 0-4400 subset and the 100-4300 subset which increases noise.

Although Cirrus100-12900 and Cirrus0-13000 have the best results , this is misleading. The Cirrus subset has mostly zeros and in the Cirrus column the Cloud subset has zero for every image which means that these two algorithms have been trained to label zero so they don't have any practical use.

For the Cloud algorithms those two that were trained only on the Cloud subset have the best result. No doubt that adding the Cirrus subset confused the algorithm as they are all rated zero for Cloud result column. Though the difference between the two is small considering that Cloud0-13000 and Cloud100-12900 have 50% more images that are all labelled wrongly as zero. Around half the Cirrus subset is classified as zero anyway so this helps mitigate the effect of adding the incorrectly labelled subset.

A simple representation of the difference data between the algorithm and the classification looks like this which is a meaningful representation of how much each algorithm has learnt.

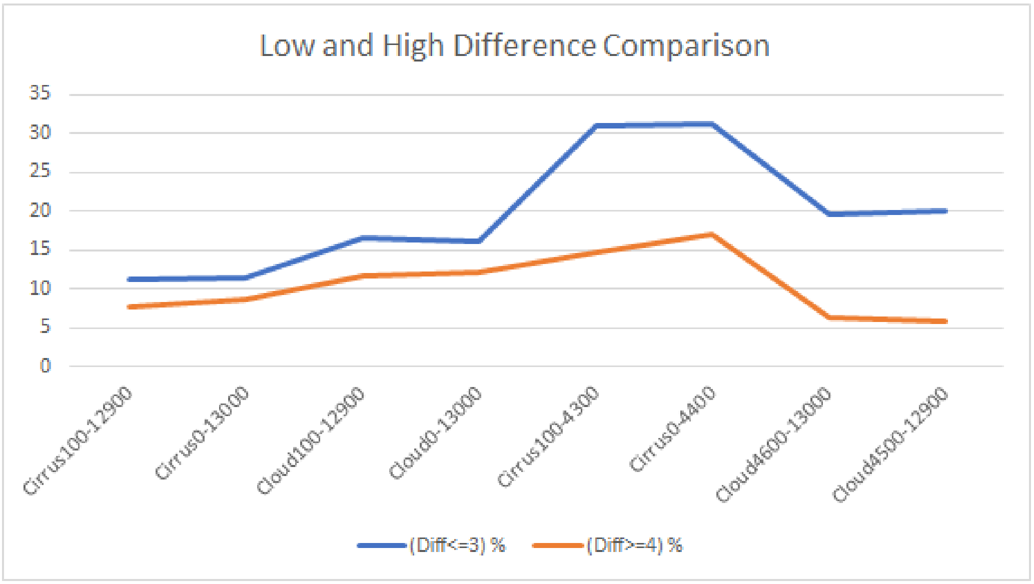


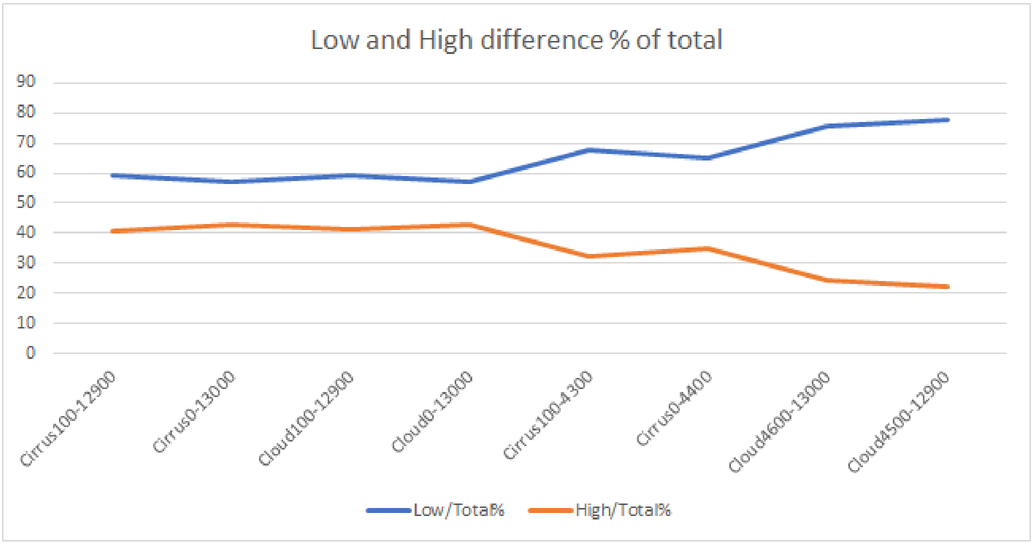
Both the Cloud algorithm and the Cirrus algorithm had around the same mean when performing over the whole data-set though the Cirrus algorithm has a slightly lower standard deviation which means it was a bit more consistent but not by much. Even though the Cirrus algorithm has the most initial accuracy we can see from here the effect of the issue that I already mentioned , it's good at classifying zeros but when the image has a value above zero it struggles the most with giving a value near the classified value. The fact that both the Cloud algorithm and the Cirrus algorithm have the same mean and a similar standard deviation is interesting as it would suggest that they both performed similarly even though they dealt with significantly different classifications and data. When I look at a more detailed analysis of the difference later on, there is a significant difference visible between them.

Cirrus 100-4300 and Cirrus0-4400 actually performs better even though it has the lowest initial accuracy. It suffers from over-fitting which means that it has a hard time getting the exact value but the algorithm seems to have learned better as it gets closer to the real value. This makes sense as it performed with properly classified values instead of just zeros. There is slight difference between them which has discussed above I concluded to be from the fact that Cirrus100-4300 was trained on 100 more images which makes it a bit better.

Cloud 4600-13000 and Cloud 4500-12900 has the best results. It has the lowest mean but the standard deviation is slight closer to the mean so it was a bit less consistent which might be due to the increased variation in the Cloud subset. The low mean is consistent with the fact that it performs on a properly classified subset and has the most number of images. The difference between these two and the Cirrus only algorithms before it isn't as much as the difference in initial accuracy. Even though the Cloud algorithm is twice as accurate overall it hasn't learnt twice as much which means that the Cirrus subset has a systematic issue that is causing it to be less accurate overall which isn't something I was able to predict in the first part of this section. I will analyse where this comes from later on.

A more detailed representation is this.

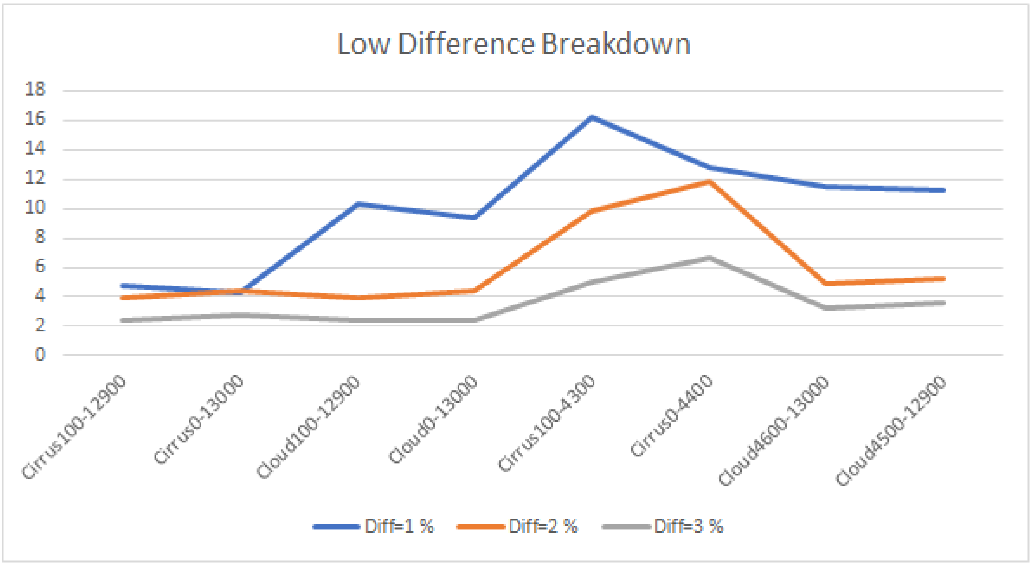




The first graph represents the percentage of predictions that had a difference of 0-3 from the classified as the top line and the bottom represents the predictions that had a difference of 4-8. Over-fitting will cause the algorithm to make small mistakes so the low difference line can be approximated as the size of the over-fit error. The high differences are due to a systematic error as it approximately only occurs when the classification has a large error which suggests a mistake. These two can have some overlap but the boundaries I have chosen are approximately where each is respectively dominant. In the case of the two algorithms working on the entire data-set though the systematic error effects the low difference range as well which means the first four points on this graph is misleading, as it suggest the dominant issue was over-fit when it really wasn't. For that reason I won't be analysing them. An easy way to relate the two lines is to look at what percentage of the total error they make up which leads to the second graph.

The graph helps confirm what I predicted about over-fit being the main issue. Cirrus100-4300 has a bit more over-fit error than Cirrus0-4400 which is good as it confirms the effect of having more images as I predicted in the first part of this section. These two have more systematic error than Cloud4600-13000 and Cloud4500-12900 which confirms what I concluded above that the Cirrus subset has a higher systematic error than the Cloud subset.

I can get some more info by looking at the specifics of the low difference.



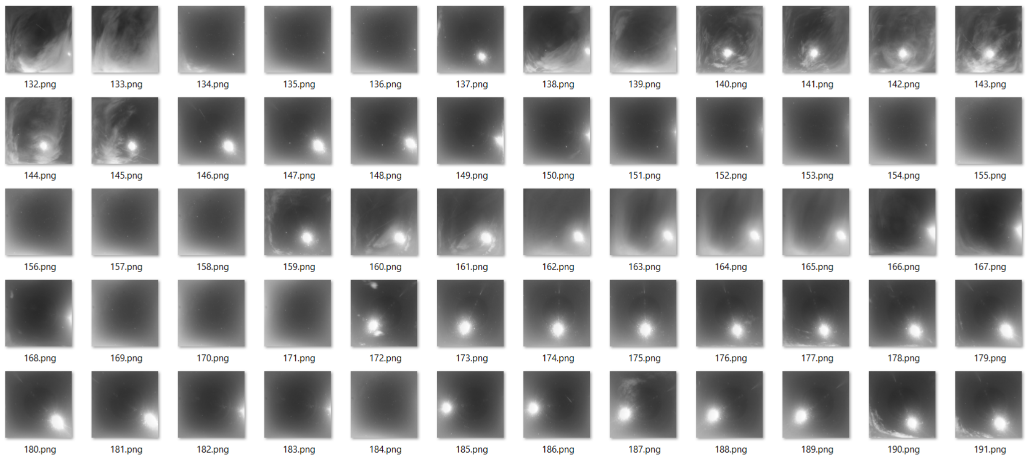
Cirrus100-12900 and Cirrus0-13000 stands out with an abnormal distribution due to it's flawed nature and there is finally a difference between these two and Cloud100-12900 and Cloud0-13000 which has a normal looking distribution which shows that it isn't as flawed.

Cirrus100-4300 is significantly better than Cirrus0-4400 which is a repeat result. These 2 results however seem to be inflated in comparison to Cloud4600-13000 and Cloud4500-12900 which again is most likely due to having a lower number of images leading more over-fit which is represented in this graph.

In conclusion the best algorithm seems to be Cloud4500-12900 which has the most number of images and even though it suffer from increased variety , the large amount of images seems to make up for it. Training over the whole data-set when it isn't defined for both parameters predictably leads to failure. I also can't really measure the error from David's classification from these results as it's effects essentially blends in. The Cirrus subset seems to suffer from increased systematic error well as over-fit error.

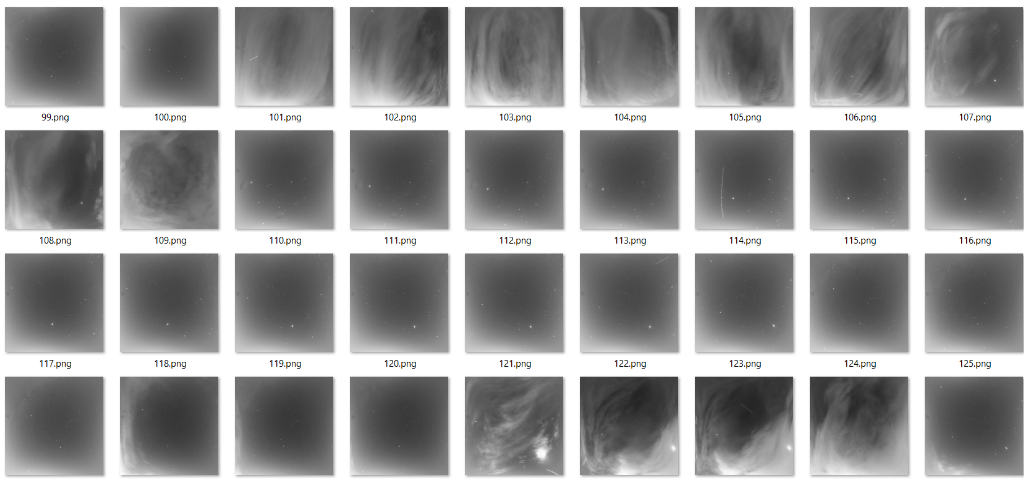
4.1. Image Analysis

I can find an explanation for the systematic error by analysing the image sets. First of all though, it's important to mention to algorithm had no problem dealing with the moon, when looking through the list of images that the algorithm got wrong, virtually none of them were of the moon.



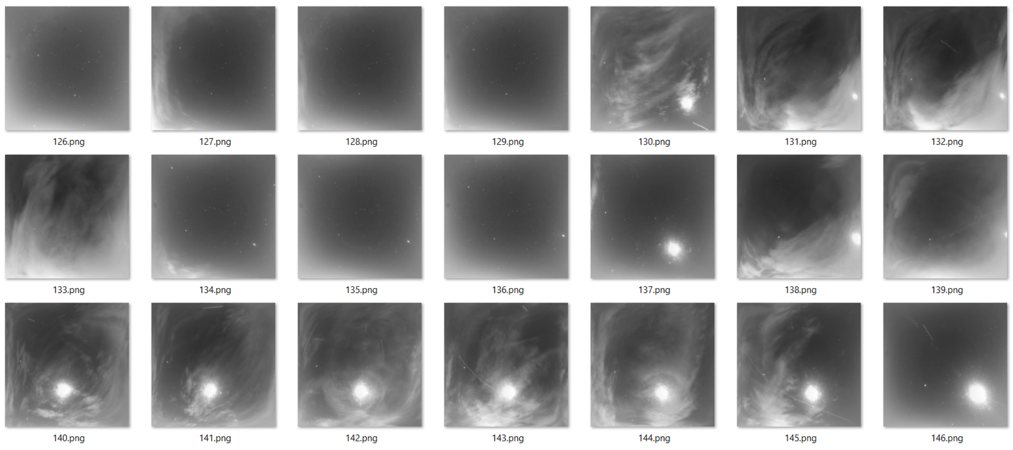
The explanation for this is that the moon is much brighter than any cloud so the algorithms learn to ignore it.

The algorithms, especially the Cirrus only algorithm can't tell the difference between light pollution and clouds.



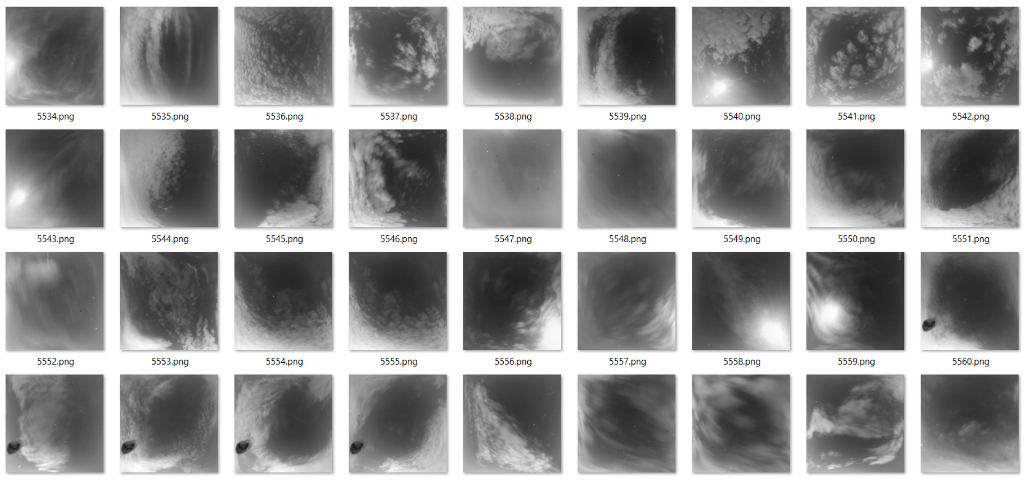
The information the algorithm has is the gamma value of each pixel and light pollution has the same effect as clouds, Cirrus especially struggles because they tend to have low brightness and are more likely to send the same signal as light pollution. The algorithm did properly predict some of these images but it was very inconsistent.

There is another problem with the Cirrus set, the algorithm struggled to identify thin cirrus clouds.

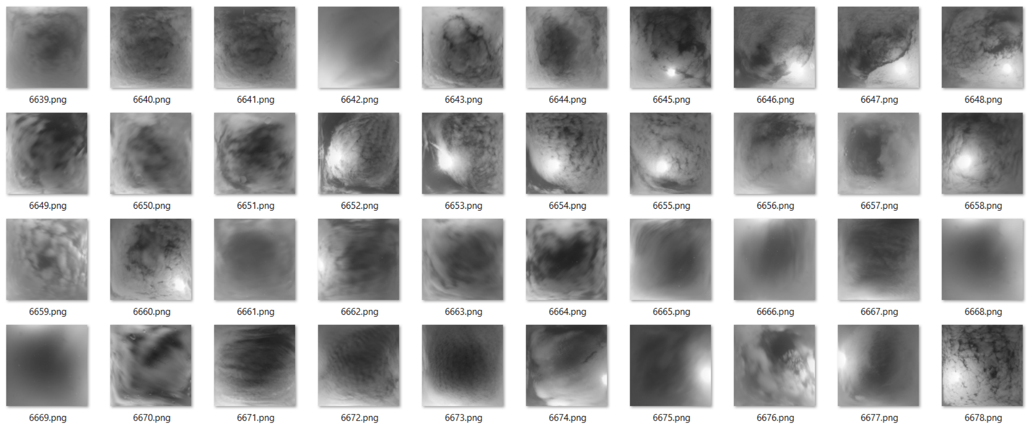


Including thick Cirrus clouds like 101.png and David labelled very light clouds as zero which seems to have confused algorithm as it's essentially getting two separate signals. Couple that with the issue discussed before and the fact that it has less images, I can see why the Cirrus algorithm had a large systematic error and the lowest overall accuracy. The entire reason of having a cirrus category was to identify a type of cloud that doesn't interfere much with telescopic observations but thick cirrus clouds do have a large interference so they should actually not be included in this category

Earlier I predicted that the Cloud subset should have more variety and therefore suffers from it's effects. It has any type of cloud inside it, including cirrus.



Every Cloud image is classified as zero in the Cirrus column and with this type of variety that kinda looks likes the Cirrus set, I can put another reason on why Cirrus0-13000 and Cirrus100-12900 was a failure. However the images of cirrus clouds in this subset are fairly thick and bright, despite the variety in cloud types and shapes, the Cloud subset has clouds that are of similar brightness which means this subset doesn't suffer from the issue mentioned above despite seemingly having more variety. Also whilst about half of the Cirrus set is classified as zero, the cloud subset is about equal numbers of images for each classification with the '8' class being about twice as big as the rest. You can see more evidence for these reasons with this image.



Whilst I can now tell why the Cirrus subset was more flawed than the Cloud subset, this image analysis allows me to see what was wrong with the classification system overall and how I can improve the algorithms. The classification system needs to both become more simple. Having a specific measure in 1/8's for cloud coverage complicates the signal and introduces more measurement error. Most images won't have an exact multiple of this coverage and classification by human eye introduces a large error, the best way to reduce this is to simplify the classification system by reducing the number of classes. For example doing 1/4's or even 1/3's and the aim of this algorithm is to tell whether or not the sky is cloudy for telescopic observations which you can still do as you can define new boundaries. The issues with the Cirrus subset can be resolved by only including thin low brightness cirrus clouds as well including more images that don't have the classification of '0'. Of course increasing the overall number of images will also improve the algorithm and whilst there is a point of diminishing returns, I don't believe this data-set reached that point.

4.2. Comparing results

(Compare results with other papers.)

Appendix A.

..