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

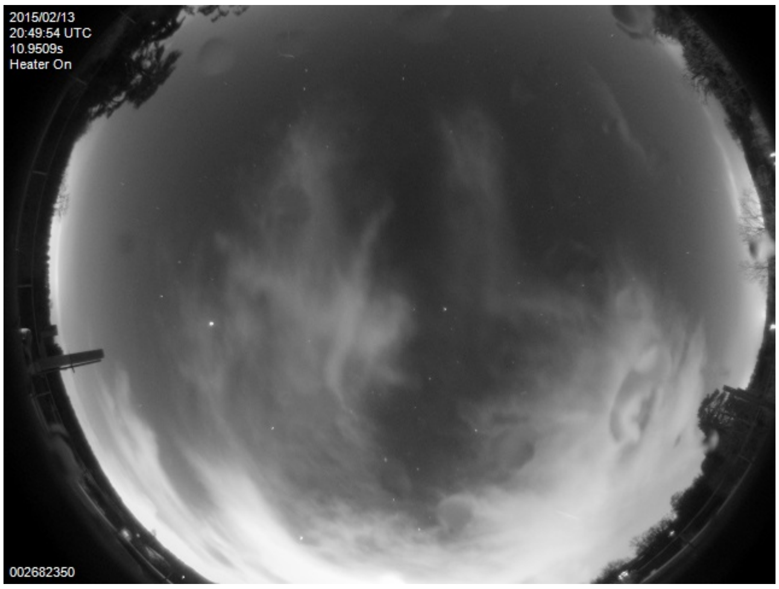
(Do I agree with the assigned values?),(How does the algorithm interpret this dataset?)(Cloud/Cirrus split)

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.

```
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
```

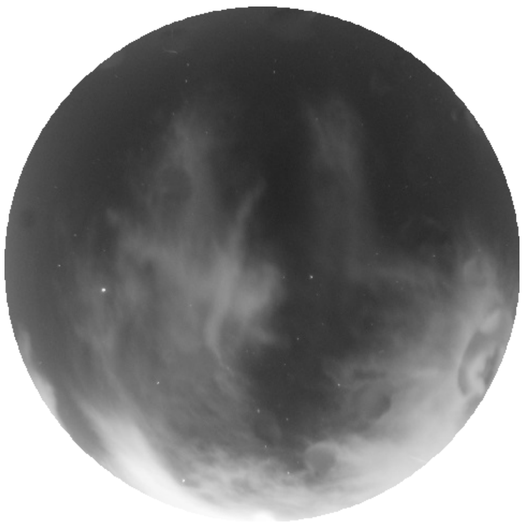
The first step is to then apply a circular mask to crop out the unwanted features.

```
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
```

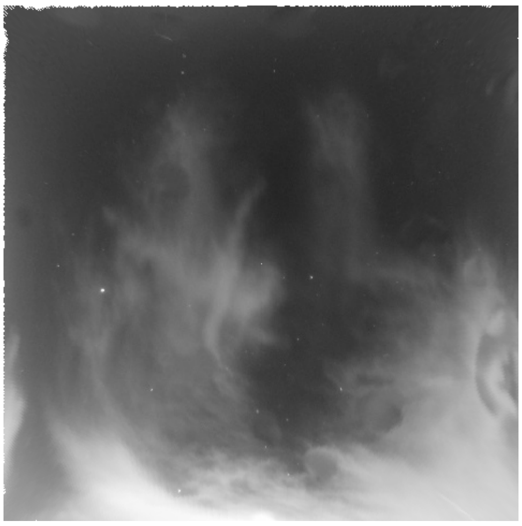
The code calculates the values needed to crop the width of the rectangular image so that the image becomes a square. This is done such that none of the data is lost, only some masked area is lost. There is no mask area above the top and below the bottom. It then performs the crop using the crop function. 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.



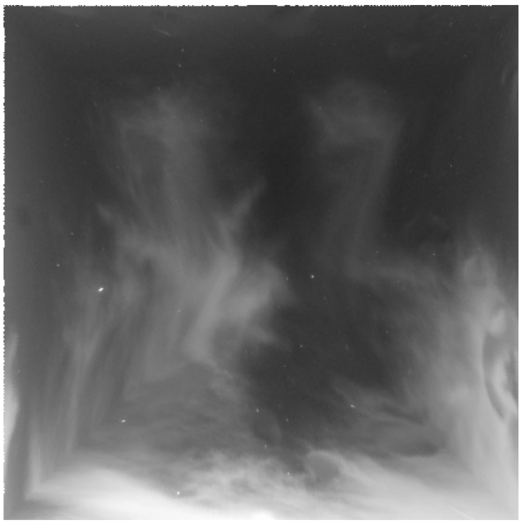
Whilst I could put this straight into the algorithm, The masked area is just empty space and the algorithm's decision trees will be simpler without having to take that and the circular nature of the actual data into account. Ideally I want to turn the actual circular data into a square. In order to do 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
```

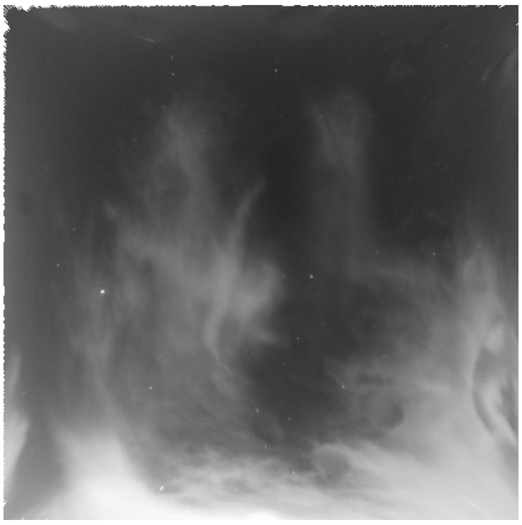
The 'fgs' method used is the Fernndez-Guasti squircle method which maps the points such that the area is the same and so it has very little deformation.



Another method is a simple stretch which doesn't preserve area and so deforms the image.



The images were classified according to their area is circular form and so it needs to preserved in order for the algorithm to learn properly. The last method is elliptical grid mapping which is a lot better but it isn't quite there.



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.

```
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

(What was the ideal result)(The parameters used)(Computation times)

4.1. Different Algorithms

(Effect of using different algorithm types)

4.2. Different Sample Sizes

(Effect of increasing the sample size)

4.3. Human Error

(How does human error effect the results)

Appendix A.

..