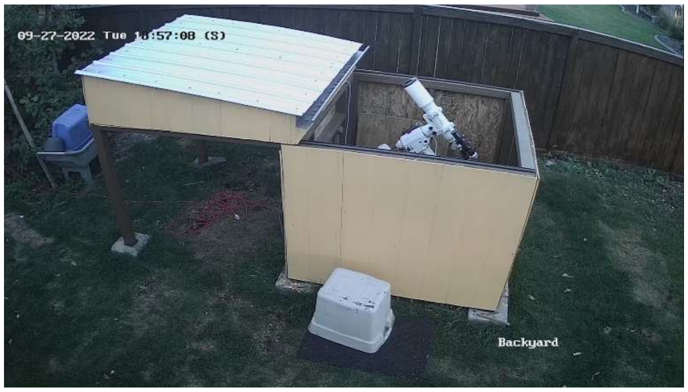
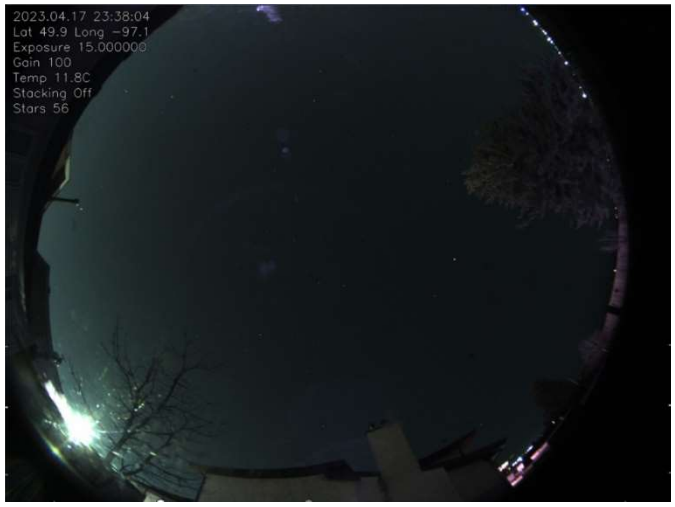
Detecting Clouds With Machine Learning

By Gord Tulloch

Automating my home observatory has been a lot of fun, and a lot of challenges as well! For new readers my home observatory is a 5x5 micro-observatory (i.e. no room to stand, just a building where the telescope can operate robotically) with a run off roof that is motorized with a gate opened, and is connected to a computer that can operate the motor that runs the roof on and off it's supports. A picture of the observatory is at right.



One of the big challenges I've faced is when should I open the roof? Obviously I can stick my head out of the house and look up to decide, but for an automated observatory, that's not very efficient.



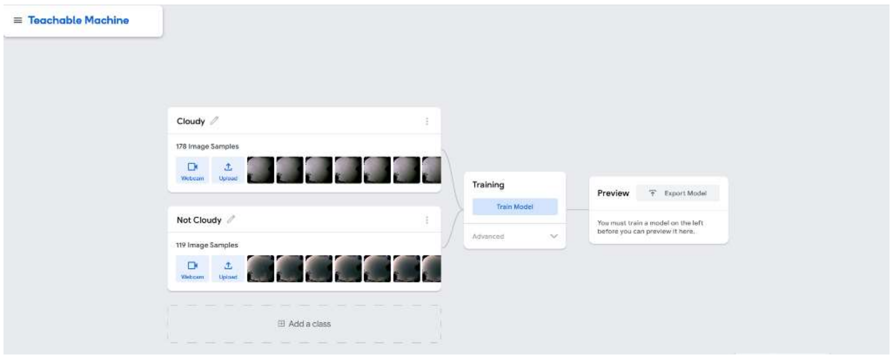
First of all I can automate the "sticking the head out of the house and looking up" part with an All Sky Camera, which is essentially a camera that points straight up, with a 180 degree field of view that shows me horizon to horizon. To create this camera I put a ZWO ASI224MC one shot color camera in a 4" PVC pipe with end caps, one of which had a hole cut into it to allow the camera to look up, with a plastic dome protecting it from the weather. To make sure the camera doesn't get blinded by dew or snow, a 12v heater ring is in place under the dome, and also computer controlled. The resulting image is above, showing a pretty much cloudless sky.

So the interesting thing is, computers have gotten pretty decent at looking at images and picking things out of them. I thought about a lot of different techniques for looking at images of the sky, and determining whether clouds were present or not. One way to do this is generate a starchart for the image based on where the stars are at the time of the image, and then superimpose that chart on top of the actual image and compare where there are matches. If there's a high number of matches then clearly the clouds are not covering the sky.

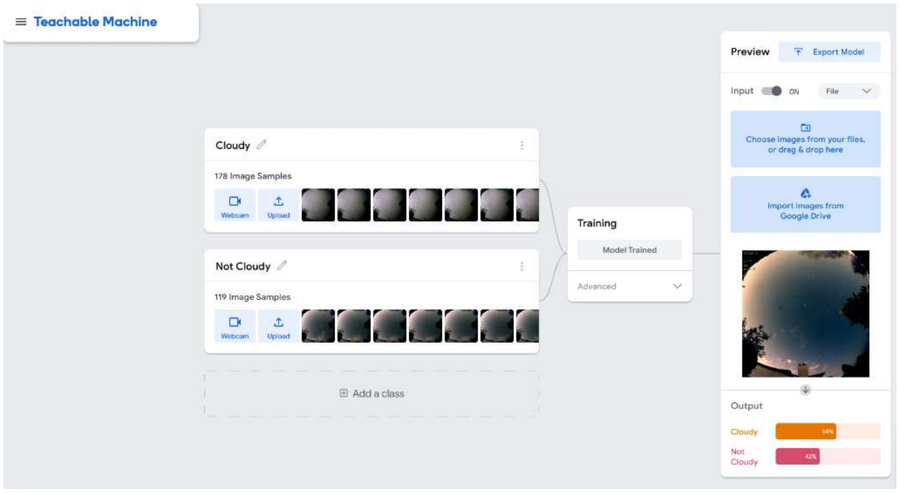
The other method is an artificial intelligence discipline known as Machine Learning. This space has really developed over the past few years, notably in the text processing area (thing ChatGPT) as well as in image processing and recognition. Google has released a variety of tools as well, including Teachable

Machine (TM)[1], which is what I have used for setting up my cloud detection system. TM allows you to create categories of images, and trains a Machine Learning model using a tool named Keras[2] that you can very simply query with some basic code. TM also provided the basic framework of a program in Python and I slightly modified it for my purposes. So work that was fairly complex to code in Python or similar languages is done in a simple web interface!

Training the Keras model on TM was very easy – first, I downloaded a nice evening of footage from my allsky camera that included both cloudy and clear skies. I used an online website that converted the MP4 video into a couple of hundred still images, then sorted those images into two folders on my disk, Cloudy and Not Cloudy. Any image that was more than 10% cloudy went into the Cloudy folder.

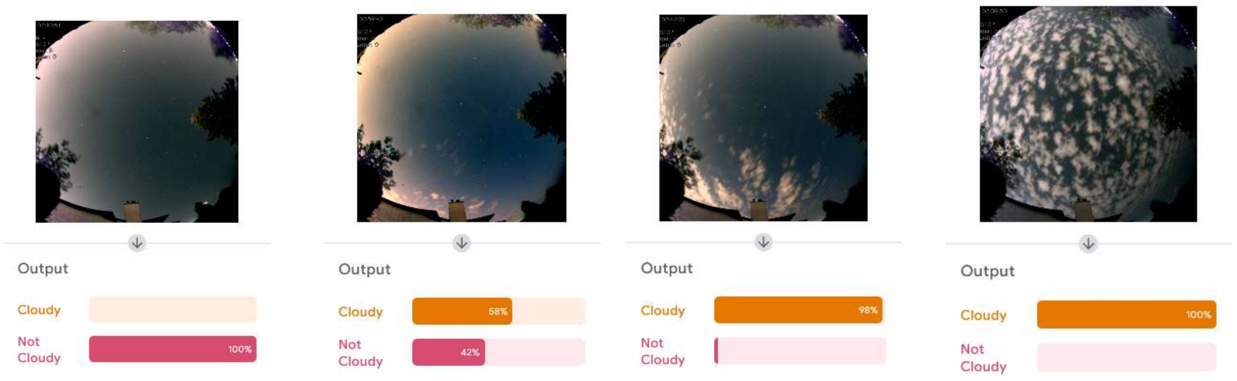


I uploaded each set of images via drag and drop to the TM web set to create my two classes of images, as seen above. Next I clicked Train Model, and a minute or two later, it presented me with a completed model I could export and download, as well as a test panel I could drop image files to and see what the model produced for these images.



[1] https://teachablemachine.withgoogle.com
[2] https://keras.io/

My test images produced the following results:



It should be noted that the percentages indicated don't indicate the percentage of cloud, but the confidence the model has in its decision.

The TM page also suggested some Python code that I could download and use to run images against the model. I copied that program to my Linux computer in the observatory where my indi-allsky software runs as mlCloudDetect.py, and created a folder to hold the python program as well as the Keras model.

In the mlCloudDetect folder I created a new Python virtual environment so I could load new libraries and versions and not mess up other Python programs, and install keras and tensorflow, which is required by keras. The installs will take a while.

```
cd ~/mlCloudDetect
python3 -m venv .
source bin/activate
pip install --upgrade pip
pip install keras
pip install tensorflow
pip install pillow
```

Running the program is done using the python interpreter. The output looks like this:

```
$ python3 ./mlCloudDetector.py

2023-05-26 10:01:28.029132: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers
on your machine, GPU will not be used.
2023-05-26 10:01:28.084529: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers
on your machine, GPU will not be used.
2023-05-26 10:01:28.084964: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow
binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the
appropriate compiler flags.
2023-05-26 10:01:28.974177: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT
Warning: Could not find TensorRT
1/1 [==============================] - 1s 709ms/step
Class: Cloudy
Confidence Score: 0.99982363
```

You can suppress all of the warnings by running the program with all of the extra stuff redirected to the bit bucket:

```
$ python3 ./mlCloudDetect.py 2>&-
```

Every minute on my server, a crontab[3] entry runs a little program modified from one provided with indi-allsky called makelatest.php that extracts the name of the latest image file produced from the allsky camera, and allows the cron command line to save it to a file called latest.jpg. The mlCloudDetect.py program runs in a continuous loop, waking up every minute to analyze the latest.jpg image and determine the fit into the Cloudy/Not Cloudy class. The program then writes a file that contains this information. I set up indi-allsky to load that file as additional information for the images it produces, and I now had cloud information to look at on my allskycamera images and videos.

Once I validate the program is working as expected next step is to integrate it into my observatory weather monitor and use it as a parameter for whether (no pun intended) to roll back my roof and do some automated observing. Watch this space for more!

Python code for the mlCloudDetect.py program is on the following page.

Special Thanks to Ignacio Diaz Bobilla[4] and utnuc[5] on Cloudy Nights for their work on this topic, and particularly to utnuc for pointing me towards Teachable Machine! I'd already started an online course in Keras and Tensorflow and envisioned doing a lot of coding to get this working, but TM made it easy!

[3] Crontab is a configuration file for the cron facility all Linux systems have that allows you to run things at a certain time of frequency. See https://www.adminschoice.com/crontab-quick-reference for more info.
[4] http://www.pampaskies.com/gallery3/Equipment/All-Sky-Camera-with-Sky-Condition-Detection
[5] https://www.cloudynights.com/topic/823536-cloud-detection-for-all-sky-cameras/

```
#!/bin/python3
from keras.models import load_model  # TensorFlow is required for Keras to work
from PIL import Image, ImageOps      # Install pillow instead of PIL
import numpy as np                   # Numpy provides numeric processing
import time                          # Sleep
from pysolar.solar import *          # Solar altitude calcs
import datetime                      # What time is it for solar altitude

# Set up lat and long so sun altitude can be calc'd
latitude=49.9
longitude=-97.1

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the model
model = load_model("keras_model.h5", compile=False)

# Load the labels
class_names = open("labels.txt", "r").readlines()

# Create the array of the right shape to feed into the keras model
# The 'length' or number of images you can put into the array is
# determined by the first position in the shape tuple, in this case 1
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

while True:
        # If the sun is up don't bother running the model
        date = datetime.datetime.now(datetime.timezone.utc)
        if (get_altitude(latitude, longitude, date) > -6.0):
                print("Daytime skipping")
                f = open("status.txt","w")
                f.write("Daytime")
                f.close()
                time.sleep(60)
                continue

        # Load the image from the AllSkyCam
        image = Image.open("latest.jpg").convert("RGB")

        # resizing the image to be at least 224x224 and then cropping from the center
        size = (224, 224)
        image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)

        # turn the image into a numpy array
        image_array = np.asarray(image)

        # Normalize the image
        normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1

        # Load the image into the array
        data[0] = normalized_image_array

        # Predicts the model
        prediction = model.predict(data)
        index = np.argmax(prediction)
        class_name = class_names[index]
        confidence_score = prediction[0][index]

        # Save prediction and confidence score to file
        f=open("status.txt","w")
        f.write(class_name[2:].replace('\n', '')+" ("+confidence_score.astype('str')+")")
        f.close

        # Sleep for 60 secs
        time.sleep(60)
```