**Is there a coating on that carbon fiber part?**

The goal is to see if an ML algorithm can determine if there is a coating on a piece of carbon fiber. Jamal made a model that runs on Google’s AutoML, but our goal is to make an app that runs on Google Glass or on an Android or some other system. AutoML is probably not the right choice here because:

* The model can not be dumped to a small device
* The AutoML “Prediction” web service will be way too slow with big images and network bandwidth.
* Glass will be looking around all the time and we only want to process images that contain the carbon fiber part that may or may not be coated.

I propose that the app function with the following logic: Do I see any carbon fiber? If yes, is it coated? So we need two models: One to identify a portion of the image that is completely carbon fiber. The second model will distinguish between coated and non-coated.

The plan is:

* Recreate Jamal’s work just to get familiar with AutoML
* Split a sample of Jamal’s large images into smaller tiles (512x512)
* Manually identify tiles that are all carbon, partially carbon, and no carbon.
* Train a model to distinguish between tiles that are 100% carbon fiber, have some carbon fiber, or have no carbon fiber. This will allow us to use AutoML to generate our training data by finding the fiber tiles for us.
* Split the rest of Jamal’s images into tiles
* Identify the tiles that are all carbon using the model that we just trained. These tiles will already be labeled as coated or not coated because we know which photo the tiles came from and we know if the photos are of coated or not coated fiber.
* Using this large set of all-carbon tiles, train another model to identify coated or not coated.

The resulting pipeline would take a series of large images, determine if there are one or more tiles that have nothing but carbon fiber in them, and then determine if the tile is coated or not coated. This logic may be able to be collapsed a bit so we have four categories of tiles: coated carbon, uncoated carbon, no carbon, some carbon. So, from a single image, if there are three tiles that are predicted to be “coated carbon” then we can stop the inference and declare that a coated part is in the scene.

**Jamal Mahoob wrote the following:**

Melonie (and Mike and Chris),

So I had some spare time tonight and wanted to see how this was going to work. The results are so good that I'm actually a bit skeptical. I wanted to try and take a crack at this with AutoML, our "ML Training as a Service" type of offering. If you're not familiar, its basically "ML Training for people who've only read about ML in the paper". We have it for various services, but the one I used was Vision (so it'd be AutoML Vision).

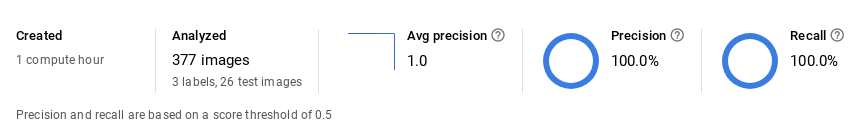
* Take a bunch of pictures
* Upload a bunch of pictures
* Label all of those pictures
* Hit the "Train" button and be wow'd

Separately from this I'll share with the group a link to a Google Drive folder containing the images I took. For as much ease-of-use as possible, I just used the normal camera on my Pixel 2 so that they'd sync somewhat easily straight to drive. Cleared off a space on my desk and tried to get it as well lit as possible. You'll notice a piece of white paper on the part, which is covering up an annotation present only on \*one\* of the parts (didn't want that to be a feature / trigger for the model) which I placed on the other piece even though it didn't have the annotation (that whole statement will make much more sense once you see the parts). From there they were all uploaded into AutoML Vision in batches, labeled as a group, and a model was trained.

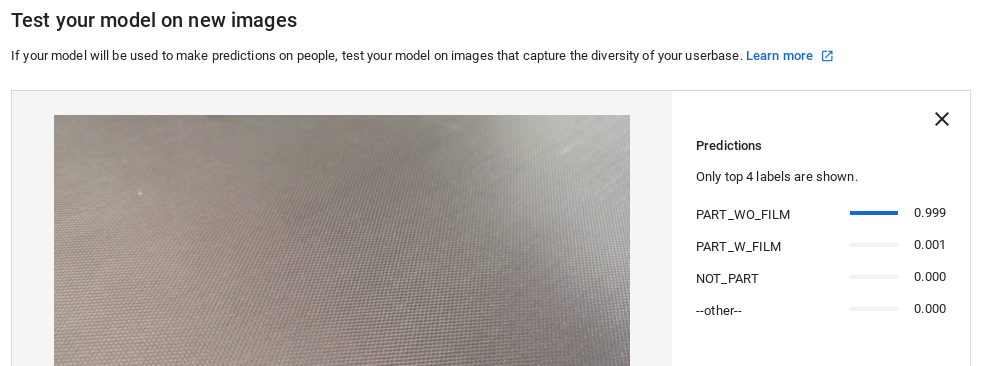
There are 3 label sets:

* PART\_W\_FILM
* PART\_WO\_FILM
* NOT\_PART
  + Things I had lying around my desk including an American Flag, a notebook with Homer Simpson on it, and my very VERY dirty old Nest Thermostat

You can probably see from this screenshot why I'm a bit skeptical of the results:



However I then took a bunch of \*additional\* pictures and uploaded them to test and, well, yeah, it seems to be working (sample size is obviously super small on this one).



At this point images could be uploaded against the model via a REST endpoint and integrated into an "app" of some sort.

With all this said, I'm sure there's \*much\* more the NTConcepts team could do with their expertise. Especially in the realm of Google, my ML knowledge is fractional.

Keep a lookout for the Drive link with the images to follow, and good to meet everyone.

**--------------- End of Jamal’s writing -------------------**

Jamal also shared a Google Drive folder that contains all of the images that he took:

<https://drive.google.com/drive/folders/1VWScUD6qfI9i5NeC9uWylyeUuWnRHV8J>

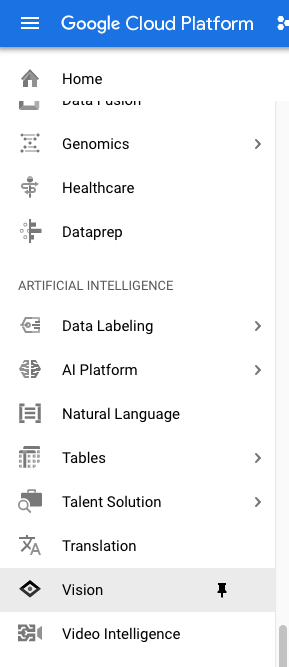
**Step 1: Recreate Jamal’s work**

I downloaded all of Jamal’s photos and put them into a zip file that can be accessed, along with everything else I will do on this project, in a [Google Drive folder](https://drive.google.com/drive/folders/16F7iZ3LFnUneJrwdFAh_FQsyQDya0CQD?usp=sharing). Anyone at NTC can access, add to, and edit stuff in this folder.

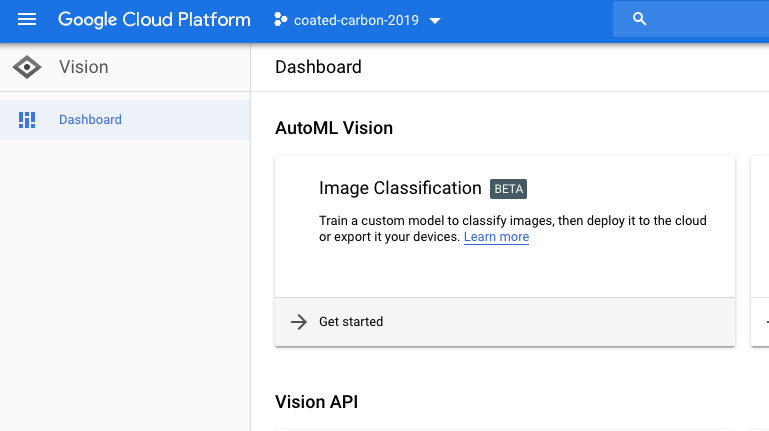
Next, I created an GCP project and enabled the AutoML API. This is the name of the project:



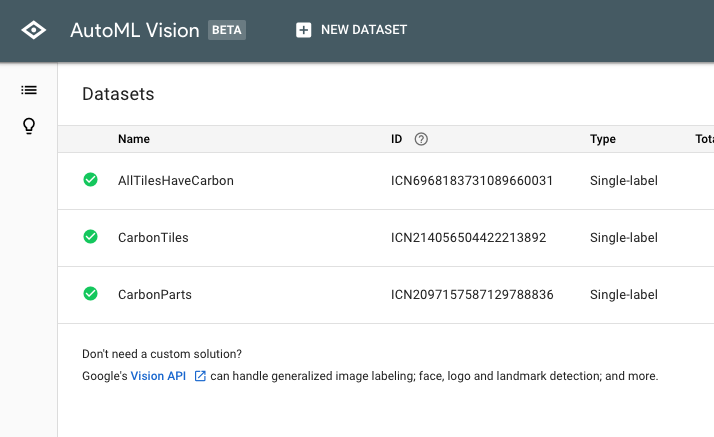
To get to the AutoML screen, drop down the menu bars on the far left and scroll way down to the “Vision” link:



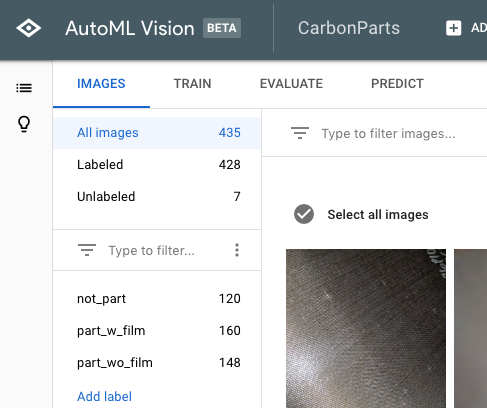
Once the “Vision” link is open, click the “Get Started” button on the AutoML Vision box:

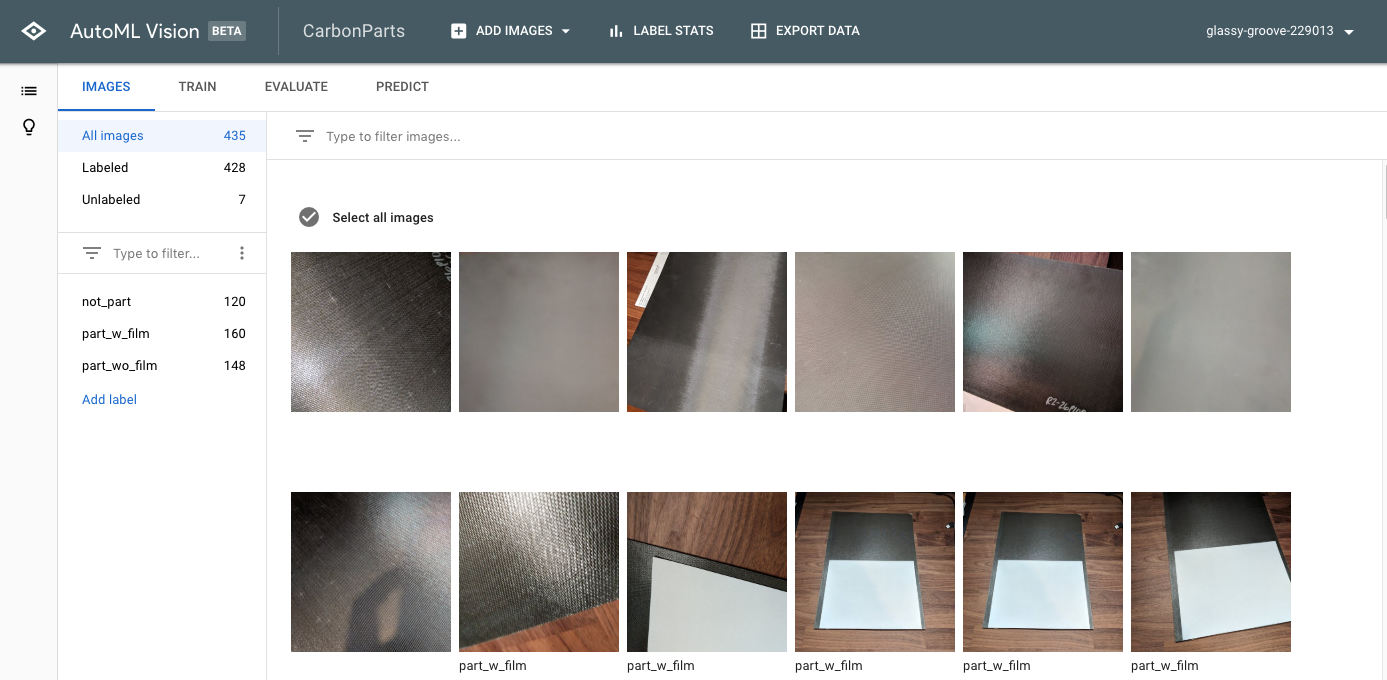


You should see a few different datasets. The “CarbonParts” data has all of Jamal’s original full frame images. The “CarbonTiles” dataset contains the same images but they are broken up into 512x512 tiles. The AllTilesHaveCarbon consists of nothing but tiles that have 100% carbon fiber coverage, and the tiles are categorized by “coated” and “not coated”.

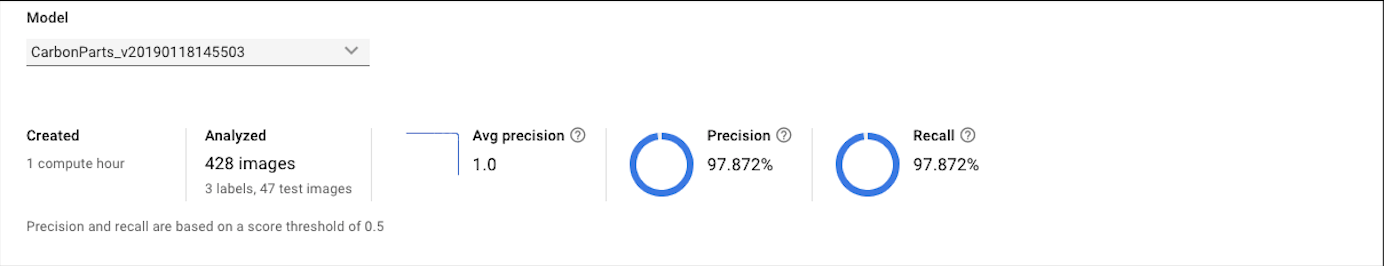


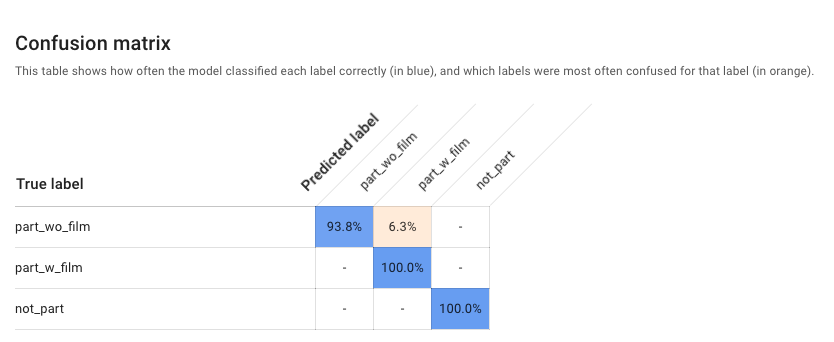
**CarbonParts dataset:** I uploaded the zip file and Google automatically assigned labels that correspond to the folder names. There were seven “unused” images and I removed them from the project, leaving me with three categories. I ran through the “train” step (see the “TRAIN” button at the top of the image below).

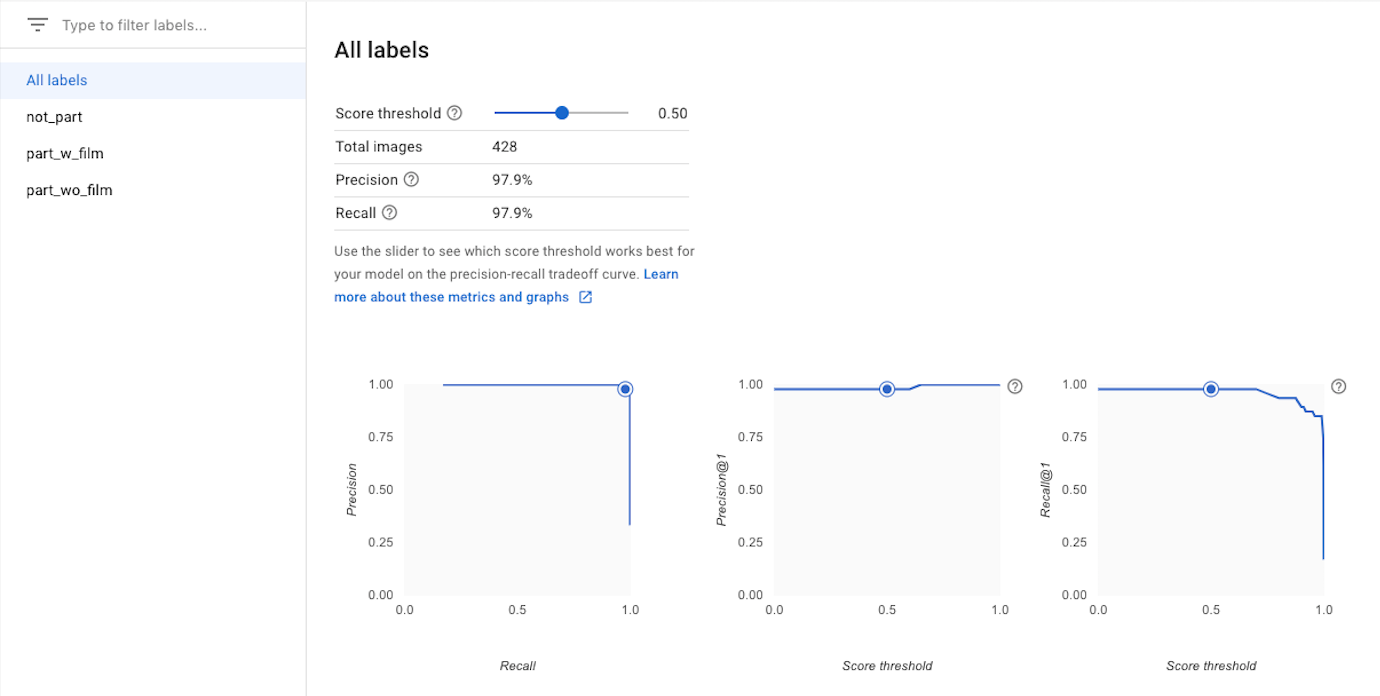


****

The results of the training are shown in the image below. My results were not 100% like Jamal’s, but they were close at 97.8%. I have six images where the model thought that there was a film when there actually wasn’t. Jamal used 337 images and I have 428, so I’m not sure what he excluded from his dataset. Anyway, I feel as if his results have been validated, so now we can move on. Interpreting the results is [explained by Google](https://cloud.google.com/vision/automl/docs/beginners-guide#evaluate).







**Step 2 - Tile some of the images**

I think that these steps are necessary for a full application. We need to isolate areas that are all carbon, and then decide if it is coated or not.

The image sizes for the Pixel images are: 3024 × 4032 or 4032 × 3024. Both numbers are divisible by 504, so we can have 8x6 or 6x8 tiles of 504 square.

*Note - Jamal says: not sure if this helps but some notes on the Glass (v3 / current) camera are:  
- 8MP  
- 720p  
  
Pixel 2 rear-facing camera I was using was leaps and bounds more than that. Unsure of the end resolution or if the glass hardware can accommodate differences (I know on a Pixel phone you can say "take a smaller / lower quality image", for instance).*

[According to techradar,](https://www.techradar.com/reviews/gadgets/google-glass-1152283/review/4) The Google Glass camera shoots 5-megapixel photos equivalent to that of the iPhone 4 camera and each picture has a 2528 x 1856 resolution.

I installed “Image Slicer” from <https://image-slicer.readthedocs.io/en/latest/index.html>

I then grabbed about 12 images each from the PART\_W\_FILM and PART\_WO\_FILM and tiled them out.

for file in `ls \*.jpg`; do slice-image $file 48; done

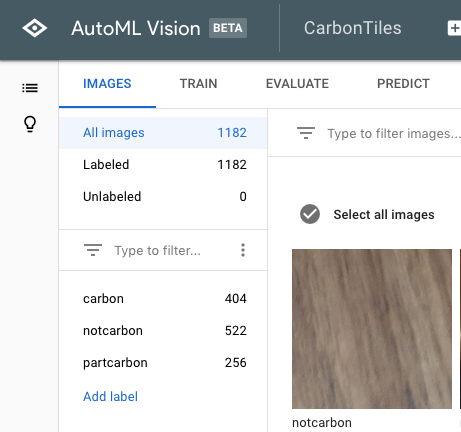
I manually took about 400 images that were all carbon fiber, with nothing else in the image, and put them into a “carbon” folder. I also selected images that had some carbon but not all carbon and put them into “somecarbon”. Finally, images with no carbon at all were put into “nocarbon”. I supplemented the “nocarbon” with tiles from Jamal’s NOT\_PART folder. Since “slice-image” dumps PNG files, I batch converted them to JPG [using the Finder app](http://osxdaily.com/2013/01/16/batch-image-conversion-mac-os-x-preview/) on the Mac.

Here are example tiles that will be fed into the model:

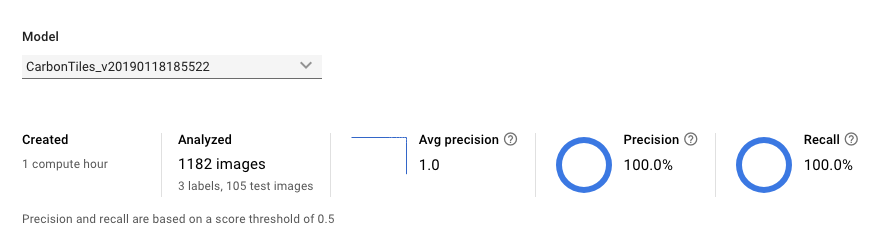
|  |  |  |
| --- | --- | --- |
|  |  |  |
| Carbon | Part Carbon | No Carbon |

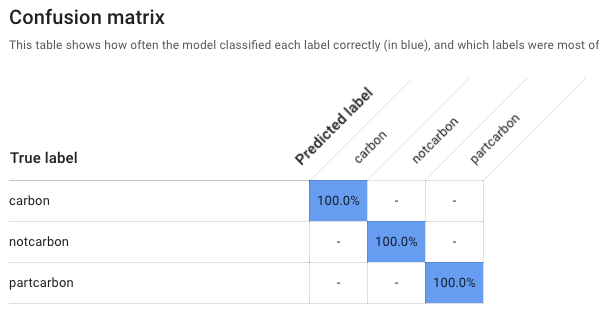
**Train a model to distinguish between tiles with all-carbon, some-carbon, no-carbon**

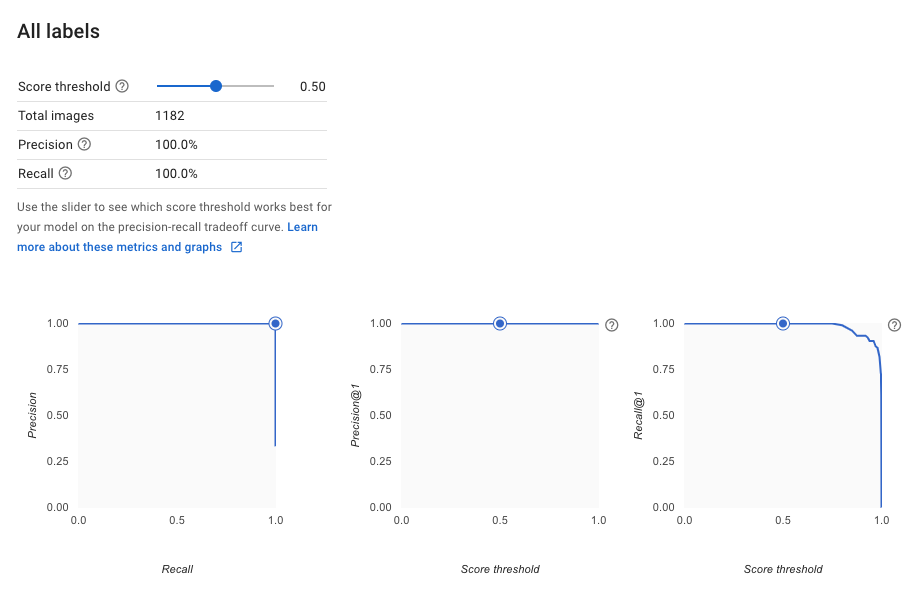
**CarbonTiles dataset:**



After all this, I uploaded all three “labeled” image sets to AutoML. The jpeg files are available on the Google Drive as “tiles.zip”. As you can see below, the model is very capable of determining which tiles are fully carbon, partly carbon and no carbon. The first chart below shows 100% for Precision and Recall but this is with a prediction threshold of 0.5. The graphs below show that these numbers are still solid up to thresholds exceeding 0.9.







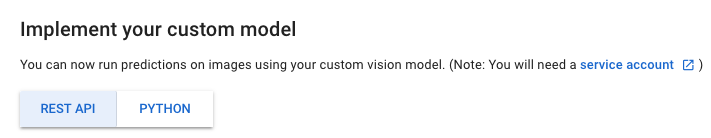
**Step 3 - Create a training set of tiles with carbon only, both coated and uncoated**

First, I split all of Jamal’s large images into tiles and then batch convert to JPEG. This required the installation of the “convert” command on my mac which is part of the Imagemagick package. I used “brew install imagemagick” to install. Here is the shell script for splitting and converting.

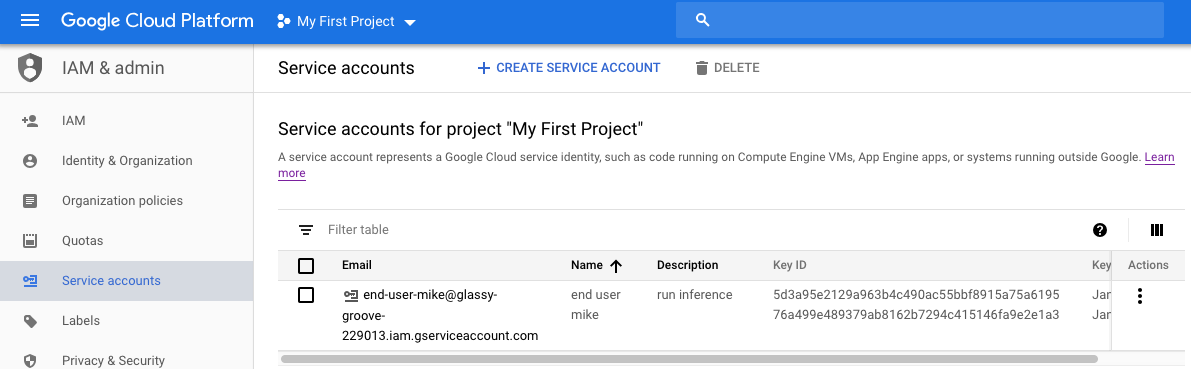
|  |
| --- |
| cd TilesWithFilm  for file in `find ../PART\_W\_FILM -name "\*.jpg"`  do  slice-image $file 48  done  ls | grep png > filelist.txt  for file in `cat filelist.txt`  do  base=`echo $file | sed 's/.png//'`  convert $base.png $base.jpg  rm $base.png  done |

Next, I needed to find the tiles that are all-carbon from the thousands of tiles that were generated. I used the AutoML model that I just trained to do this. Overall, this was a very painful process. Beyond the usual difficulty in dealing with OAuth tokens, AutoML is in beta and the docs for the restful endpoint and for the Python access were incomplete. Also, the end-user authentication is not enabled on this API yet, so a service account was necessary. I tried using the restful endpoint and “curl” as many ways as I could think of, but with no success. I experimented with the example python program but it did not work. Finally, I was able to sprinkle the correct fairy dust onto the process and I got the python to work.

Google text on the “PREDICT” page of the AutoML model:



So, you will need to set up a service account. Here is what it looks like when you are done:



The service account must have “run inference” permission for the project. You will need to download the json file that contains the keys for this user. The example python code that is included on the PREDICT page is not functional. Below is the python program that I used to run inference on all of the tiles. You can see where the “glassy-groove-229013-76a499e48937.json” credentials file is loaded as the SERVIDE\_ACCOUNT\_FILE.

|  |
| --- |
| import sys  import os  from google.cloud import automl\_v1beta1  from google.cloud.automl\_v1beta1.proto import service\_pb2  from google.oauth2 import service\_account  SERVICE\_ACCOUNT\_FILE = '/Users/mikeevanoff/CarbonParts/glassy-groove-229013-76a499e48937.json'  creds = service\_account.Credentials.from\_service\_account\_file(SERVICE\_ACCOUNT\_FILE)  project\_id = "glassy-groove-229013"  model\_id = "ICN18736000583343382"  def get\_prediction(content, project\_id, model\_id):  prediction\_client = automl\_v1beta1.PredictionServiceClient(credentials=creds)  name = 'projects/{}/locations/us-central1/models/{}'.format(project\_id, model\_id)  payload = {'image': {'image\_bytes': content }}  params = {}  request = prediction\_client.predict(name, payload, params)  return request # waits till request is returned  if \_\_name\_\_ == '\_\_main\_\_':  path = '.'  outfile=open('BatchOutput.txt','w')  files = os.listdir(path)  for image in files:  print(image)  if image.endswith('.jpg'):  ff=open(image, 'rb')  content = ff.read()  pred = get\_prediction(content, project\_id, model\_id)  print(pred)    try:  outfile.write("%s, %f, %s\n" % (image,  Pred.payload[0].classification.score,  str(pred.payload[0].display\_name)))  except:  print("Error occurred processing ", image) |

The end result of the processing is a file that looks like:

IMG\_20190115\_192025\_02\_01.jpg, 0.835266, partcarbon

IMG\_20190115\_192001\_1\_05\_02.jpg, 0.991857, partcarbon

IMG\_20190115\_192055\_06\_02.jpg, 0.999901, carbon

IMG\_20190115\_191928\_1\_02\_07.jpg, 0.999980, notcarbon

IMG\_20190115\_191934\_02\_05.jpg, 0.999576, carbon

IMG\_20190115\_191934\_1\_02\_06.jpg, 0.999355, carbon

Now, we can generate a large training set of coated and uncoated tiles by searching for tiles that are 99% certain to be all carbon.

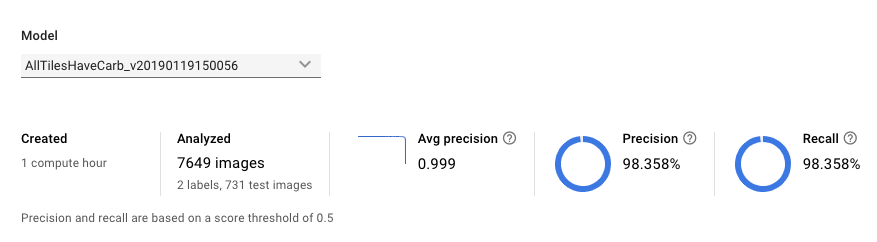
egrep -v '(not|part)' InferenceWithFilm.txt | grep '0\.99' | awk -F, '{print $1}' > CarbonTilesWithFilm.txt

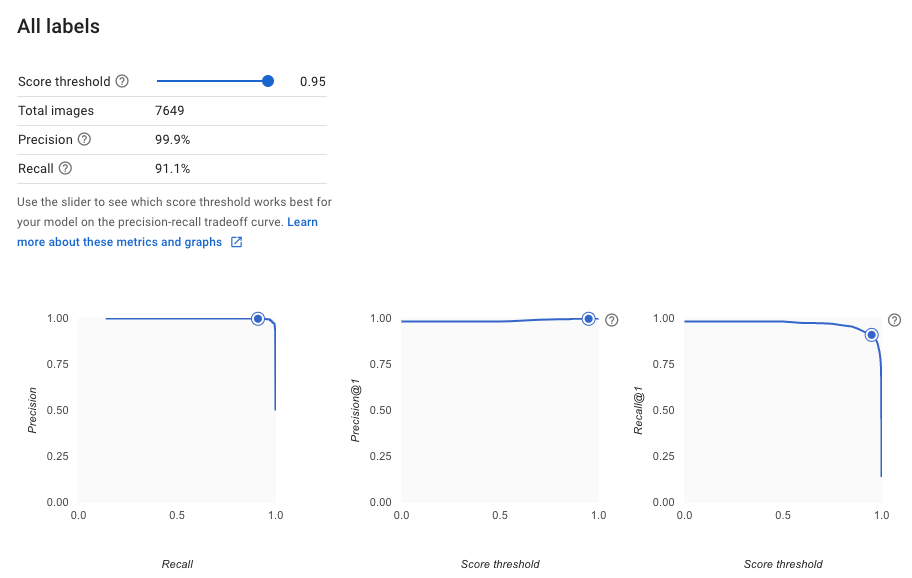
egrep -v '(not|part)' InferenceNoFilm.txt | grep '0\.99' | awk -F, '{print $1}' > CarbonTilesNoFilm.txt

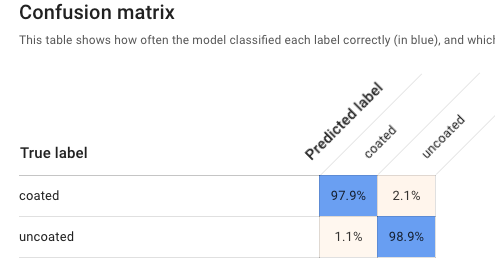
**AllTilesHaveCarbon dataset:** Here is the training dataset in AutoML

****

Here are the training results:







The danger of these models is that they are not well trained on “not carbon” items. Here are three images of grey carpeting and the final model is 75-85% sure that these are carbon fiber. There will be some false positives until sufficient examples of “almost carbon” are found and added to the training set.

|  |  |  |
| --- | --- | --- |
|  |  |  |

The value of additional training can be seen below. The top prediction of 99.9% is from the first model where I was recreating Jamal’s work. The bottom estimate of 88.7% comes from the third model which was trained on multiple carbon-only tiles. Neither result is good, but the added training gives a much better result.

|  |
| --- |
|  |
|  |

**Processing real-time data**

It will be extremely inefficient to pass each entire frame out to an ML model. The Pixel camera’s main image is being split into a 7x7 grid of smaller tiles by the split-image process. We can specify that at least two adjacent tiles of the ten diagonal tiles in the yellow part of the image must be all carbon before the image is a candidate for full inference. This will ensure that the camera is looking directly at the carbon piece rather than looking elsewhere. In this case, ten tiles will be processed for a non-detection rather than 49, which will speed the processing by a factor of five.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Once we have determined that an image contains sufficient carbon tiles, we can determine if the tiles are coated or not.

**Final Approach and results**

I wrote a python program that creates, trains and saves a model. It is currently about 98% accurate. All scripts are found in [this drive folder](https://drive.google.com/drive/folders/16F7iZ3LFnUneJrwdFAh_FQsyQDya0CQD?usp=sharing). The steps involved were:

1. Download the images that Jamal took
2. Rotate them so they are all horizontal, then crop from 4032x3024 to 2560x2560. This gets the center of the photo and makes it easy to split the images into 100 256x256 tiles. Steps one and two are done using ***MakeTiles.sh*** found [in this Drive folder](https://drive.google.com/drive/folders/16F7iZ3LFnUneJrwdFAh_FQsyQDya0CQD?usp=sharing).
3. Manually identify tiles that match the following four categories and put them into directories whose names match the categories:
   1. Coated
   2. Uncoated
   3. Notcarbon
   4. Partcarbon
4. Run ***IntegratedTrain.py*** - this is a python/Keras ML program that trains a model to recognize these four classes. It runs from scratch with no pre-trained model needed. After about 4 hours of training, it is approximately 99% accurate in its predictions. We now have a model that can take a tile and identify if it is coated carbon, uncoated carbon, not carbon at all or partially carbon.
5. The program “***AreYouCarbon.py***” (*still under development*) can read tiles and tell you if they are carbon or not and, if so, if it is coated.
6. At this point, we can demonstrate the ability to identify tiles that are coated carbon. Now, we need more information about the application.
   1. Will it be run completely on Glass or will Glass send the images to a main server.
   2. What ML inference frameworks, if any, are available for Glass
   3. Is Glass powerful enough to run inference and, if so, at what rate?

**A few details:**

Here is the shape of the model for now. Since it achieves such high accuracy as-is, we can probably make it leaner if we need speed or memory.

|  |
| --- |
| **Layer (type) Output Shape Param #**  =================================================================  conv2d\_1 (Conv2D) (None, 125, 125, 16) 2368  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_2 (Conv2D) (None, 125, 125, 32) 4640  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_1 (Activation) (None, 125, 125, 32) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  batch\_normalization\_1 (Batch (None, 125, 125, 32) 128  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_3 (Conv2D) (None, 63, 63, 32) 9248  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_2 (Activation) (None, 63, 63, 32) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  batch\_normalization\_2 (Batch (None, 63, 63, 32) 128  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_1 (Dropout) (None, 63, 63, 32) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_4 (Conv2D) (None, 63, 63, 64) 18496  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_3 (Activation) (None, 63, 63, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  batch\_normalization\_3 (Batch (None, 63, 63, 64) 256  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_5 (Conv2D) (None, 32, 32, 64) 36928  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_4 (Activation) (None, 32, 32, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  batch\_normalization\_4 (Batch (None, 32, 32, 64) 256  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_2 (Dropout) (None, 32, 32, 64) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_6 (Conv2D) (None, 32, 32, 128) 73856  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_5 (Activation) (None, 32, 32, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  batch\_normalization\_5 (Batch (None, 32, 32, 128) 512  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  conv2d\_7 (Conv2D) (None, 16, 16, 128) 147584  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_6 (Activation) (None, 16, 16, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  batch\_normalization\_6 (Batch (None, 16, 16, 128) 512  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_3 (Dropout) (None, 16, 16, 128) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  flatten\_1 (Flatten) (None, 32768) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_1 (Dense) (None, 512) 16777728  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_7 (Activation) (None, 512) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  batch\_normalization\_7 (Batch (None, 512) 2048  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dropout\_4 (Dropout) (None, 512) 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  dense\_2 (Dense) (None, 4) 2052  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  activation\_8 (Activation) (None, 4) 0  =================================================================  **Total params: 17,076,740**  **Trainable params: 17,074,820**  **Non-trainable params: 1,920** |

The saved parameters for the trained model are available in the Drive folder - look for IntegratedTrain.h5

Classification Report

|  |
| --- |
| precision recall f1-score support  coated 0.99 0.92 0.95 500  notcarbon 0.95 0.53 0.68 307  partcarbon 0.81 0.95 0.87 282  uncoated 0.78 0.97 0.87 500  micro avg 0.86 0.86 0.86 1589  macro avg 0.88 0.84 0.84 1589  weighted avg 0.88 0.86 0.86 1589 |

Training output

|  |
| --- |
| Epoch 1- loss: 1.5718 - acc: 0.6636 - val\_loss: 1.7066 - val\_acc: 0.6822  Epoch 2- loss: 1.2607 - acc: 0.7360 - val\_loss: 1.5058 - val\_acc: 0.7218  Epoch 3- loss: 1.1489 - acc: 0.7643 - val\_loss: 1.7218 - val\_acc: 0.6388  Epoch 4- loss: 1.0653 - acc: 0.7884 - val\_loss: 1.0268 - val\_acc: 0.7892  Epoch 5- loss: 1.0029 - acc: 0.8127 - val\_loss: 1.3196 - val\_acc: 0.7369  **.**  **.**  **.**  Epoch 496- loss: 0.1479 - acc: 0.9837 - val\_loss: 0.2371 - val\_acc: 0.9459  Epoch 497- loss: 0.1507 - acc: 0.9816 - val\_loss: 0.2238 - val\_acc: 0.9641  Epoch 498- loss: 0.1580 - acc: 0.9793 - val\_loss: 0.1651 - val\_acc: 0.9767  Epoch 499- loss: 0.1579 - acc: 0.9827 - val\_loss: 0.1720 - val\_acc: 0.9805  Epoch 500- loss: 0.1532 - acc: 0.9793 - val\_loss: 0.6613 - val\_acc: 0.8647 |