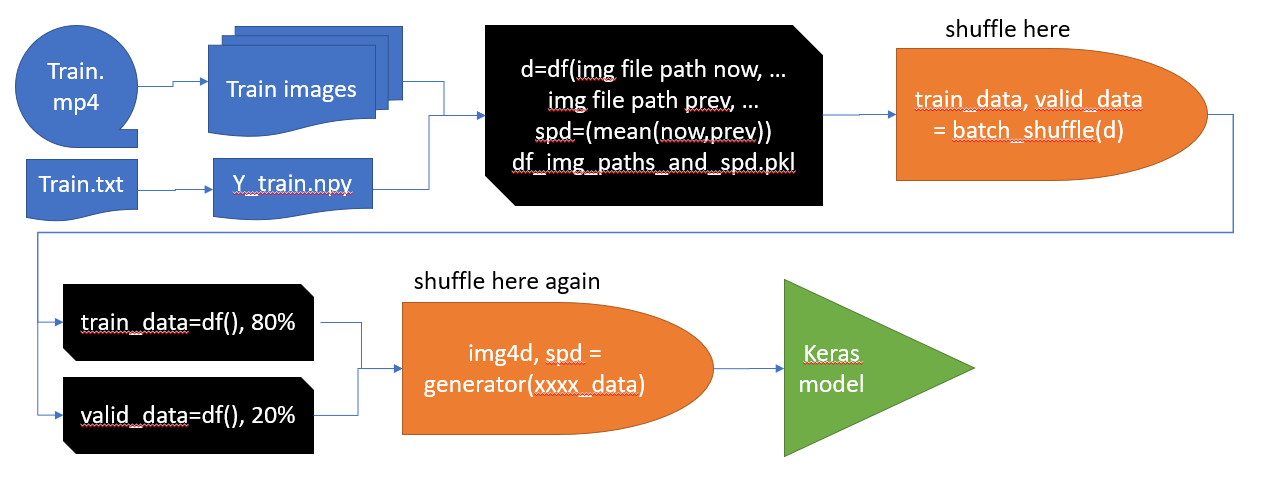
# Overview



generator does this:

1. shuffle list of now file path. prev flie patj, mean(spd now, spd prev)
2. load now and prev images
3. crop now and prev images
4. Canny edge detection for now and prev images (settings should probably be learnable)
5. optical flow between edges (settings should probably be learnable)

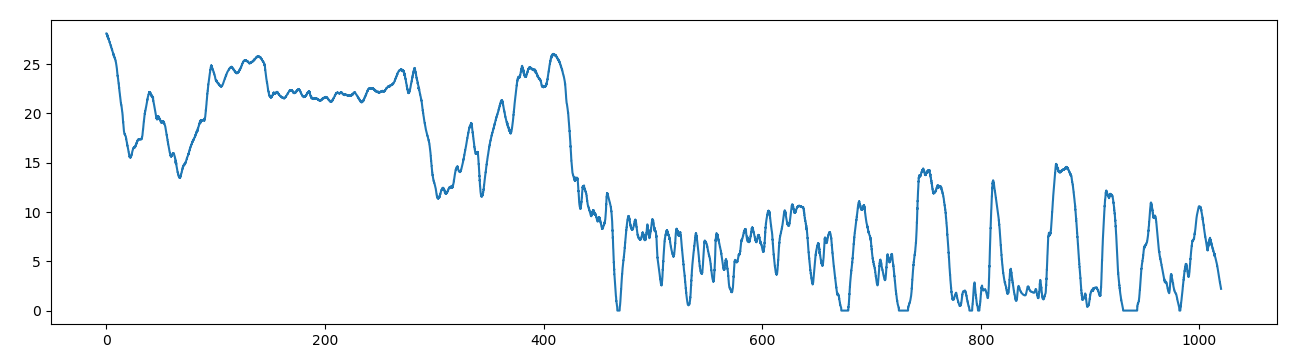
img4d= stack of optical flow output, where flow output is same size as copped image, stack is size batch\_size fed to generator

# Speed data

Watching the videos for train and test, I found that the distribution of data between the two is not identical. Seems like test has more instances of stopped car seeing cars drive across its view. This could be trouble for classic train>test scenario. My plan, view the test video, find sections which are very different from train, manually annotate, and train on that data, this is most important for the stopped car is fast cars crossing its view, so the 0mph cases, which are also easy to annotate.

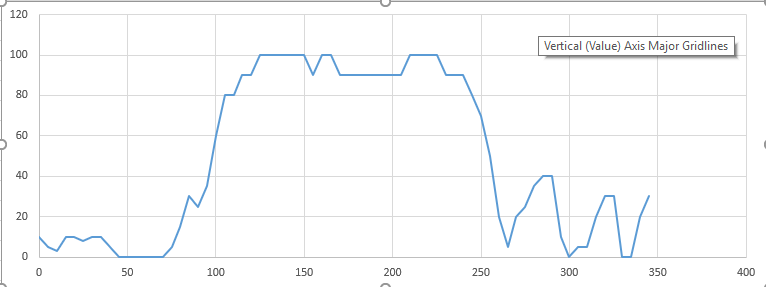
## Train

MPH



## Test

Several trials of watching the video and writing down the speeds estimated. Km/h



the

# data preparation

batch shuffling from <https://chatbotslife.com/autonomous-vehicle-speed-estimation-from-dashboard-cam-ca96c24120e4>

# Top level model

A regressor that looks at 2 sequential images and predicts the ground speed of the vehicle taking the images.

Initial concept: gray scaleCrop>edge detection>concatenate (previous frame, current frame)>2D Conv>(maybe some stuff like pooling or other>(maybe more conv)>dense layer>speed value.

Then after some quick search for reasonable 2dConv net architectures as a starting point, I found “Vehicle Motion Detection using CNN”, by Yaqi Zhang, which led me to cv2.calcOpticalFlowFarneback, which I will use in place of the edge detection.

Concept 2: grey scale>crop>edge detection>cv2.calcOpticalFlowFarneback>2dConv> dense layer>speed value. This is very simple but I gotta start somewhere.

I’ll setup the keras model as follows. Custom layer>standard layer>standard layer

Concept 3: grey scale>crop>edge detection>cv2.calcOpticalFlowFarneback>2dConv>batch norm> dense layer>speed value. Where: generator>standard layer with batch norm>standard layer

# Custom layer

## Cropping frames



The bottom of the image is the dash, it’s a waste of compute resources to run that through a train and test workflow, so crop the bottom off.

Trial and error led to imgc0 = img[0:350, 0:639],



The top of the image is typically full of sky or other stuff we don’t need.

imgc0 = img[100:350, 0:639]



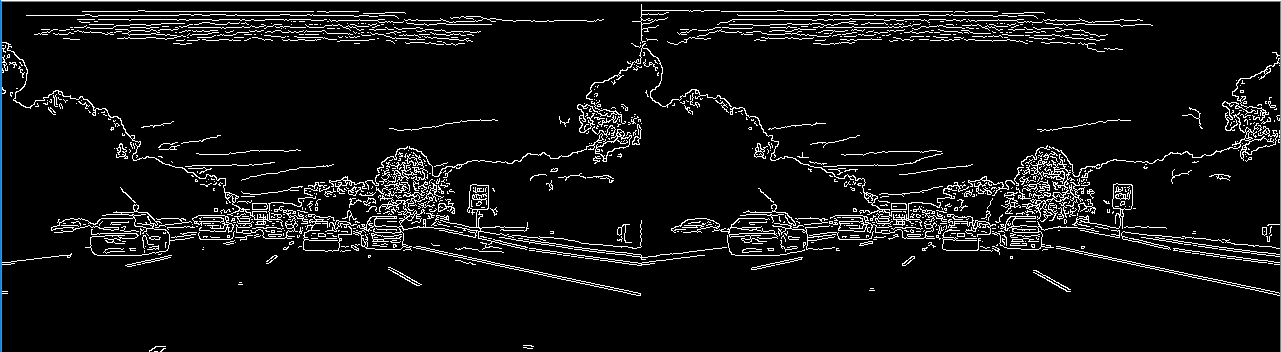
We’ll try this.

## Edge detection

### Canny

2 parameters. Could be hyperparameters, but I think with a bit of trial and human review, I can get a reasonable starting point.

Using 20,50. We’re getting sky, which is useless.



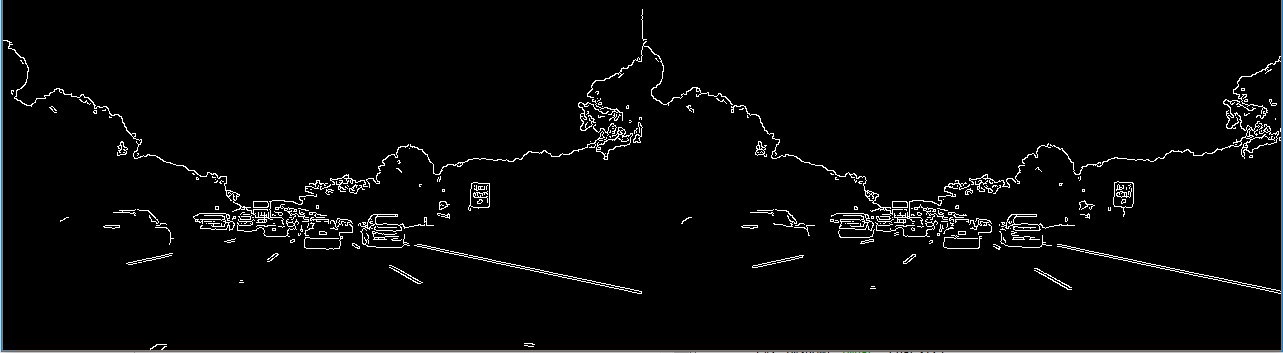
Using 50,100. Not bad, getting road side features.



75,150. Happy medium, lets start with this.



100,200. Not bad either, but less road side features, might be a good thing?



200,400. Not enough stuff left. If lane lines were always present, this might do, but I can rely on that.



I’ll start with 75, 150.

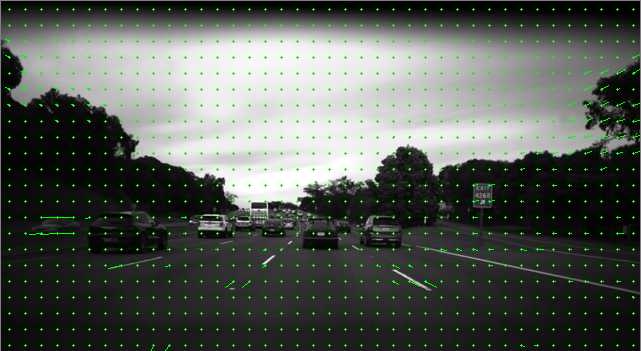
Might need to adjust these parameters based on image lighting or estimated speed, or maybe even edge pixel density (i.e. I make a target “white pixel density” of say 3%, and adjust parameters to approach this).

### HED

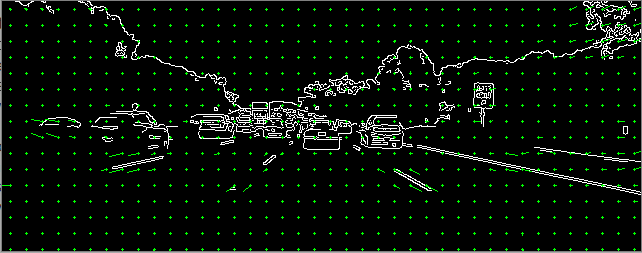
Toolchain not installed yet.

## Optical Flow

Default settings from opencv example opt\_flow.py



Maybe this also works well with images of canny edges? Looks like it does



## Developing the functions which make up the custom keras layer

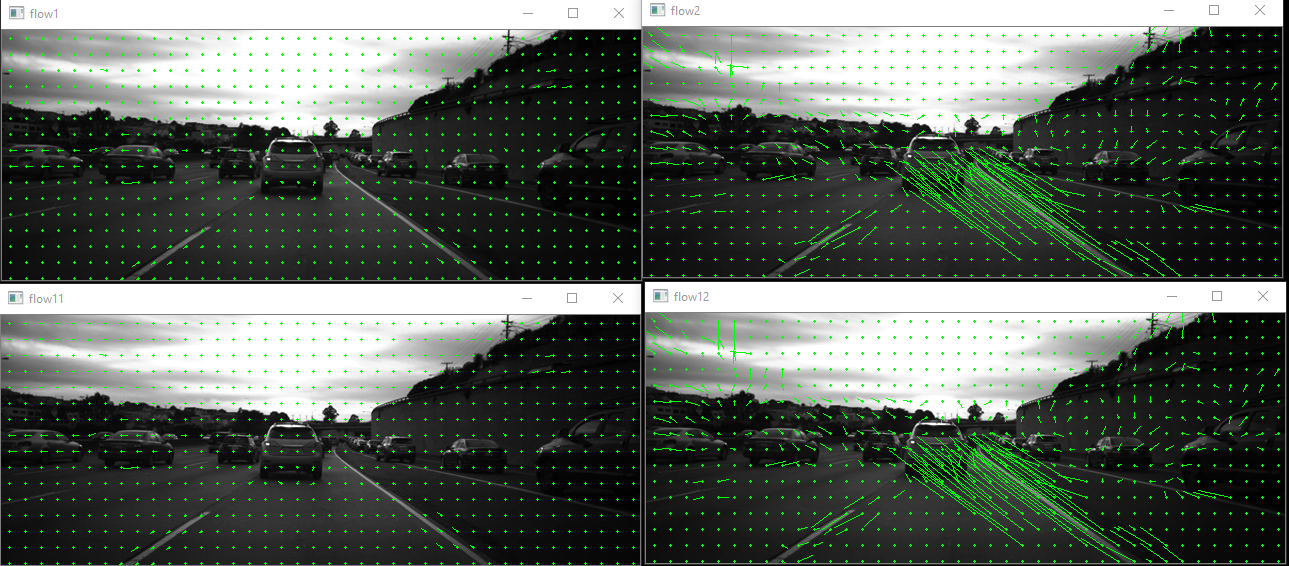
Stack looks like this:

* CustomLayer
  + image\_tensor\_func
    - cv2\_preprocess\_tnsr\_fcn

I have test function for cv2\_preprocess\_tnsr\_fcn. These were last seen here: 51c3423ccdb6a99f883b4bb6dd8f81e3e938f4db

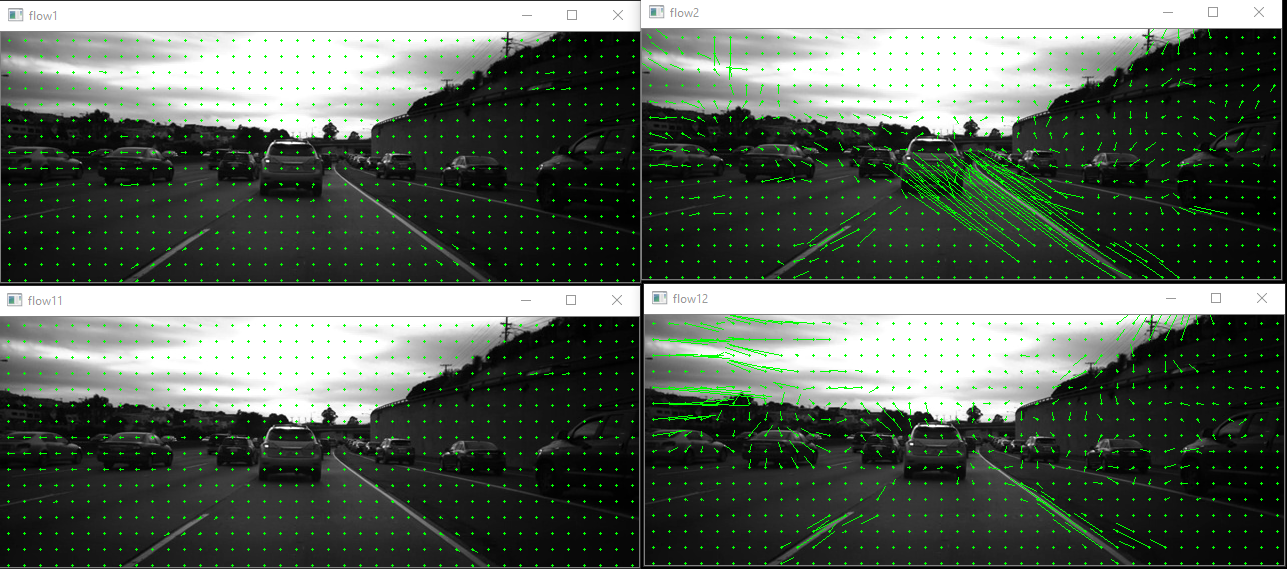
I then removed cv2\_preprocess\_old and this section of main, 3] DEEVELOPING PREPROCESSING TECHNIQUE at: f38c3085a5165b5100f35159aa1b94882cc9b8f1

At this point, I’m getting this output:



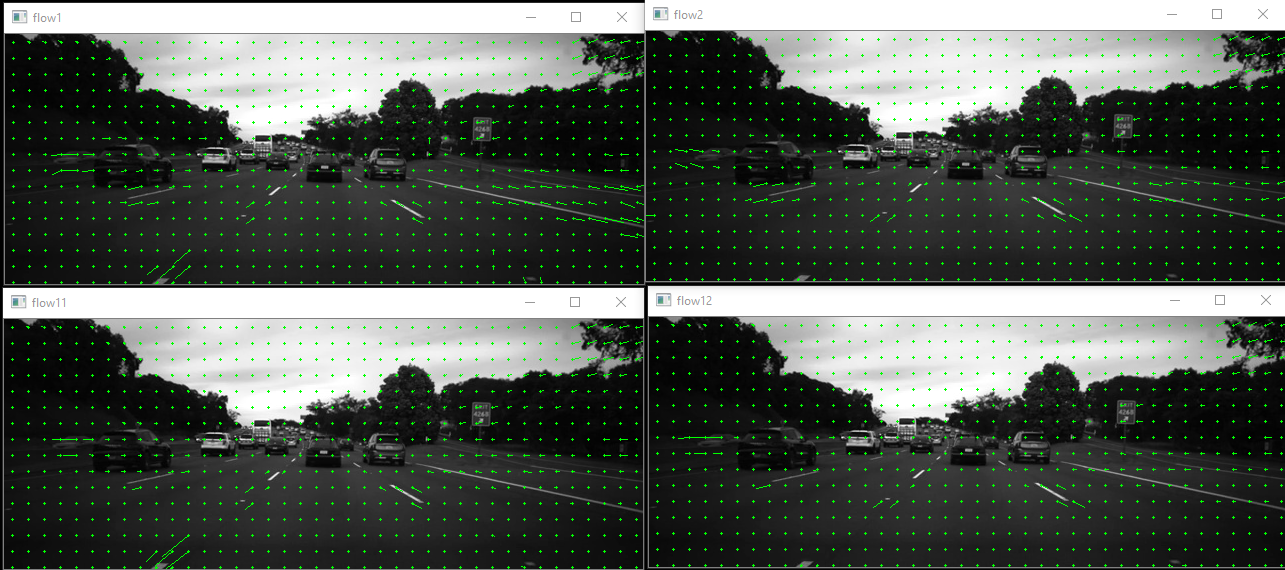
This doesn’t exude confidence that the preprocessing pipeline is working as desired, but at least the 2 versions of it are doing the same thing, so at least I have some control of what’s happening.

Try removing the Canny Edge step from the function creating the lower row of images.



I think it’s fair to say that the lower set without edge detection upstream of flow detection is better, well for these 2 image sets at least. Could be that the edge detection was poorly tuned for these image sets. Whats more troubling is that the flow in each column is very different. The left column is frame10898.jpg and frame10899.jpg, while the right column is frame10899.jpg, frame109.jpg. so there is the cause of the issue. Need to figure out why the original dataframe is returning some rows with sequential images, and some rows with non-sequential images.

Fixed that.



Now, we can see that its fair to say the upper row shows more instance of speed relative optical flow. Lets start with this.

709459ef7ca053433b51cecdef1aae40b042d760

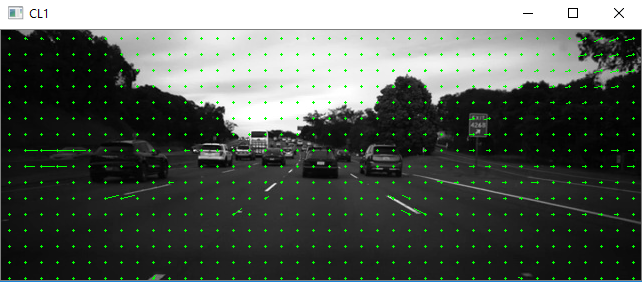
Tests with custom layer resulted in input output size mismatch. Input expected size =[dim1,dim2,dim3,dim4]:

* dim1=sequences of images or images sets [1:inf]
* dim2 = image num pixels height, 250 for this project
* dim3 = image num pixels width, 639 for this project
* dim4 = image depth/layers(e.g. RBG, hsv, optical flow) in this project, the 2 layers will be grey scale image now, and grey scale image previous.

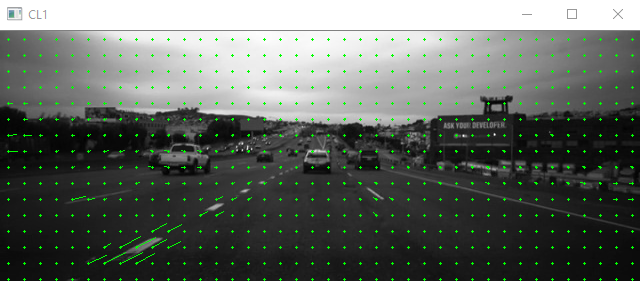
Input expected type = float64.

Clearly the dataframe doesn’t have this shape or type. Solution: make a generator which produces a input of this shape and type from the dataframe.

Made a prototype naive generator, with a test feature for batch\_size==2, and ran test on custom layer. Cv.Canny function was giving error. The below image is without Canny edge dection.



Below image produced using generator prototype with batch\_size=3



281a2ffdc64987dc2e6ea1946363ea7d9f8a1640

This layer should NOT accept img4d and iterate through it. It should accept img3d (2 sequential raw images from the dash camera), since:

1. that’s what it would get in real life if it was to be used
2. during training validation, testing, keras will be fed a vector of inputs, and a vector of targets. The inputs are image pairs, and the targets are speeds.

So remove the iteration layer. c4b19802af631b62a13074a3817959667e5bb3b6

Need to normalize the optical flow for the net. Move all custom layer code in to the generator.

Since the input is only 1 feature type, optical flow, we may not need to normalize it. Remove the normalization in the generator at first to see if the net works acceptably without it.

Removed generator normalization @ dd94c4a7bb56fe1dea5241ed0830ff46e897ee0b

Re-introduce Canny edge detection @ f6cb37cf2d129d0298648b7532097c637de2c156

### TODO

#### Need to fix error from cv.canny in cv2\_preprocess\_tnsr\_fcn:

Traceback (most recent call last):

File "<ipython-input-17-ce082881944b>", line 1, in <module>

runfile('C:/git/comma-speed-challenge/csc00.py', wdir='C:/git/comma-speed-challenge')

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\spyder\_kernels\customize\spydercustomize.py", line 827, in runfile

execfile(filename, namespace)

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\spyder\_kernels\customize\spydercustomize.py", line 110, in execfile

exec(compile(f.read(), filename, 'exec'), namespace)

File "C:/git/comma-speed-challenge/csc00.py", line 205, in <module>

b = model.predict(img4d)

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\engine\training.py", line 1167, in predict

steps=steps)

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\engine\training\_arrays.py", line 294, in predict\_loop

batch\_outs = f(ins\_batch)

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\backend\tensorflow\_backend.py", line 2666, in \_\_call\_\_

return self.\_call(inputs)

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\backend\tensorflow\_backend.py", line 2636, in \_call

fetched = self.\_callable\_fn(\*array\_vals)

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\client\session.py", line 1382, in \_\_call\_\_

run\_metadata\_ptr)

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\framework\errors\_impl.py", line 519, in \_\_exit\_\_

c\_api.TF\_GetCode(self.status.status))

UnknownError: error: OpenCV(4.1.0) C:\projects\opencv-python\opencv\modules\imgproc\src\canny.cpp:949: error: (-215:Assertion failed) \_src.depth() == CV\_8U in function 'cv::Canny'

Traceback (most recent call last):

File "C:\Users\albert.mathews\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\ops\script\_ops.py", line 206, in \_\_call\_\_

ret = func(\*args)

File "C:/git/comma-speed-challenge/csc00.py", line 109, in image\_tensor\_func\_o

rimg3d = cv2\_preprocess\_tnsr\_fcn(img3d )

File "C:/git/comma-speed-challenge/csc00.py", line 96, in cv2\_preprocess\_tnsr\_fcn

now=cv2.Canny(now,75,150)

cv2.error: OpenCV(4.1.0) C:\projects\opencv-python\opencv\modules\imgproc\src\canny.cpp:949: error: (-215:Assertion failed) \_src.depth() == CV\_8U in function 'cv::Canny'

[[Node: custom\_3/cvOpt = PyFuncStateless[Tin=[DT\_FLOAT], Tout=[DT\_FLOAT], token="pyfunc\_3", \_device="/job:localhost/replica:0/task:0/device:CPU:0"](\_arg\_input\_4\_0\_0)]]

# Keras model development

baseline keras model working with train generator @ 0d98a244e1129b1f51180179ee19d8e766d9cd43

create train and validation dataframes by shuffling all data

pass to generator

pass generator to keras model

@ d98669b44802e16e75609a2e5f83a7cb21036513

# TODO

1. create proper keras generators (maybe inspire from <https://chatbotslife.com/autonomous-vehicle-speed-estimation-from-dashboard-cam-ca96c24120e4>)
   1. train
   2. validate
2. create keras conv layer(s)
3. setup keras early stopping callback (maybe inspire from <https://chatbotslife.com/autonomous-vehicle-speed-estimation-from-dashboard-cam-ca96c24120e4>)
4. identify hypers
5. pull in the zero speed frames from the test data since that is easily human annotated.