Deep Learning with MATLAB and TensorFlow for shape recognition

Mentors: Dr. Trimble
Brennan Yamamoto
By: Blaine Furman (EE396)
Hunter Garrett (EE296)

Image Classifier Documentation

Introduction:

For the Kanaloa team project, we were presented with the task of image detecting shapes in real time for the Maritime Robotx competition for various reasons. These reasons include the detection for docking the boat, detecting waypoints and much more. Due to this, we decided to use some type of Neural Network to solve this issue. Our two, almost defacto choices to do this was MATLAB (R2018b to be exact) and TensorFlow with a custom made Neural Network. We chose these two out of the many other deep learning programs because either their customizability (in TensorFlow's case) or their (sort of) ease of use (like in MATLAB's case).

****NOTE: Tensorflow has been abandoned as we have decided to use Matlab for image recognition instead which makes everything we did for Tensorflow irrelevant at this point.**

TensorFlow:

Link to our multi-label classifier with color recognition that uses Keras:
https://www.pyimagesearch.com/2018/05/07/multi-label-classification-with-keras/

This site contains the code for the classifier that we are using as well as a thorough explanation of the code from the author.

Originally: This classifier categorized clothing and it's color

Note: Originally (when we first tested out the classifier with the clothing data) it was thought that the image training data was used because the accuracy was so high; however, this was not the case. We actually tested it out with clothing images that were not within the training dataset.
**Also, this classifier is confirmed to be performing classification and not regression.
**Run this on a GPU as it will make the training process SIGNIFICANTLY FASTER

To run the classifier (on Windows):
Enter command prompt (or conda prompt)
Then type: conda activate tfdeeplearning (or type: activate tfdeeplearning if using conda prompt)
This lets us run the classifier in Raymond's environment that he created with his config file.
In order to use Raymond's environment, follow his instructions on the Kanaloa Github under the Deep Learning Project Directory.
Link: https://github.com/riplaboratory/Kanaloa/tree/master/Projects/DeepLearning

Train the classifier by running the command:

```
python train.py --dataset dataset --model fashion.model --labelbin mlb.pickle
```

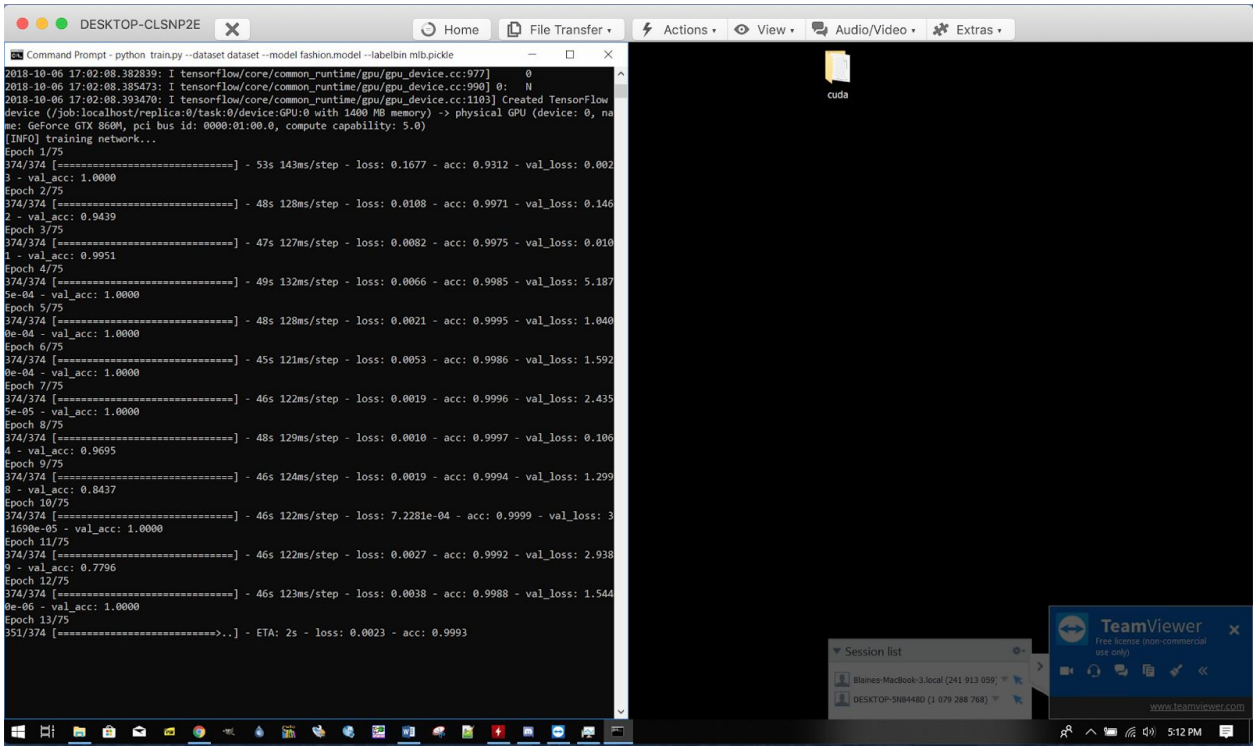
Here is a link by the same author of the blog post on command line arguments for reference:
https://www.pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/

Note:
You need to first navigate to the directory that the classifier is in before running this command.
In order to run this classifier, you need to install additional libraries that are missing in the anaconda environment that Raymond created with his config file.
These include: keras, imutils, and cv2
To install: Enter the tfdeeplearning environment using the command given earlier, and type:
pip install keras
pip install imutils
pip install opencv-python

We then replaced the clothing training images with a shape dataset from Kaggle:
https://www.kaggle.com/smeschke/four-shapes
This shape dataset contains 4 images: Circle, Square, Star, and Triangle
We threw each shape into separate folders and named them:
Circle_black
Square_black
Star_black
Triangle_black

This classifier automatically generates labels based on the names of the folders of the image data that we create. It's important to name the folders in this format, or else the classifier will not run and will be stuck in the training phase.

Here is a screenshot of retraining the classifier with shapes:



Training will take a while as this will loop 75 times (75 Epochs), although the number of Epochs can be adjusted to be less (although this will significantly reduce accuracy).

Results:

To test out the classifier, enter in this command:

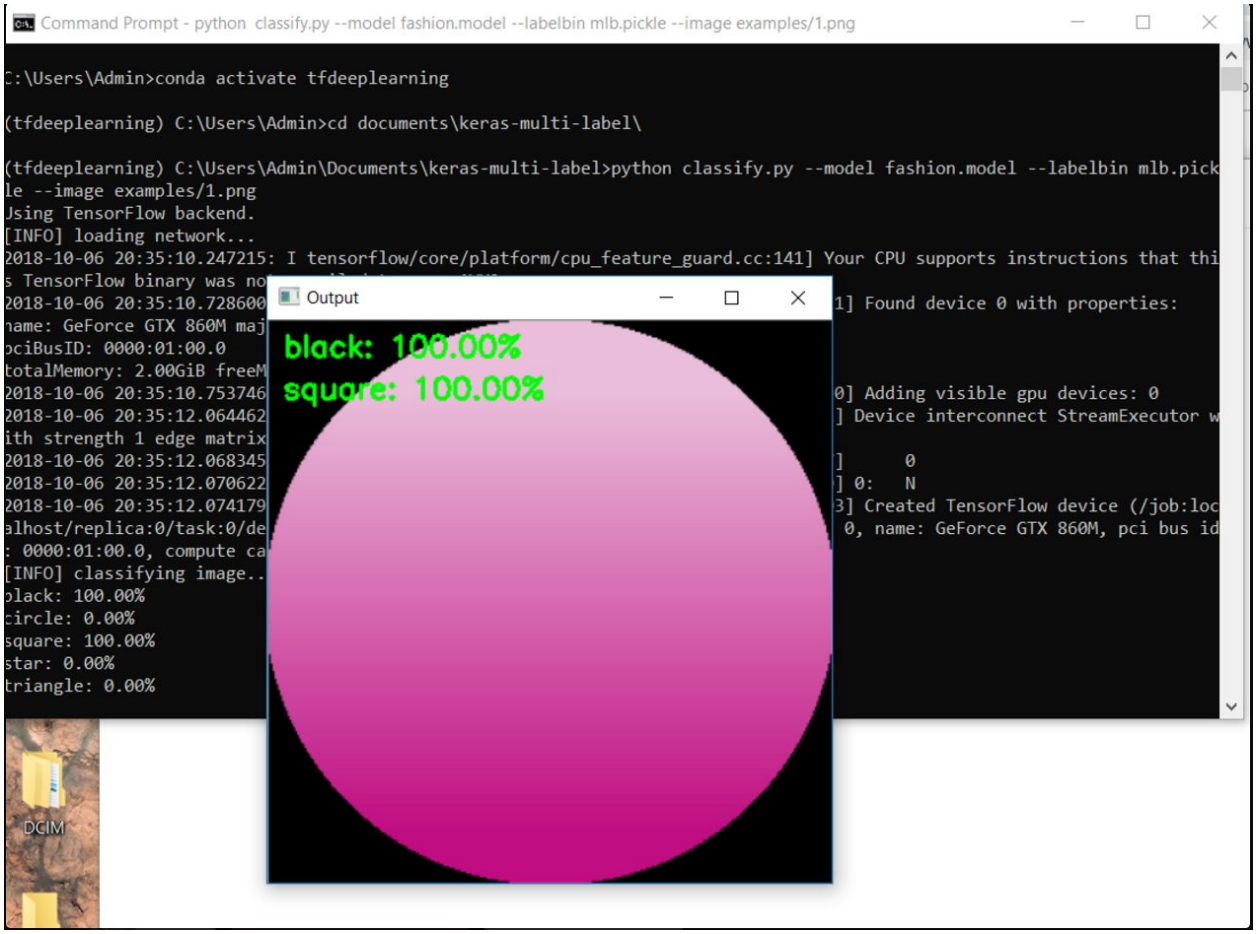
```
python classify.py --model fashion.model --labelbin mlb.pickle --image
examples/example_01.jpg
```

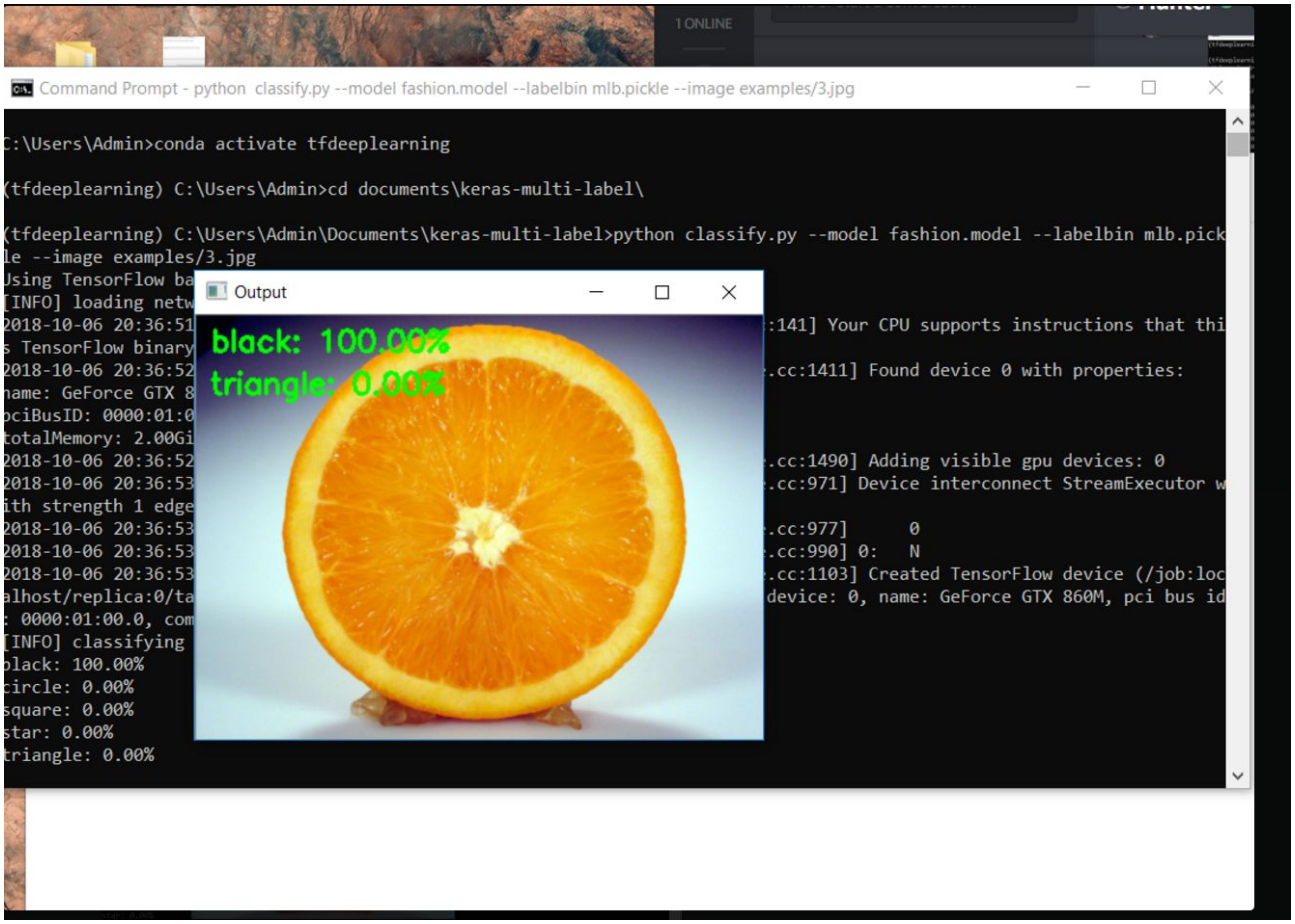
Replace the “examples/example_01.jpg” with the path to whatever image you want to feed in. For example, if we want to feed in an image called circle_01.jpg within the “examples” folder, we would use:

--image examples/circle_01.jpg

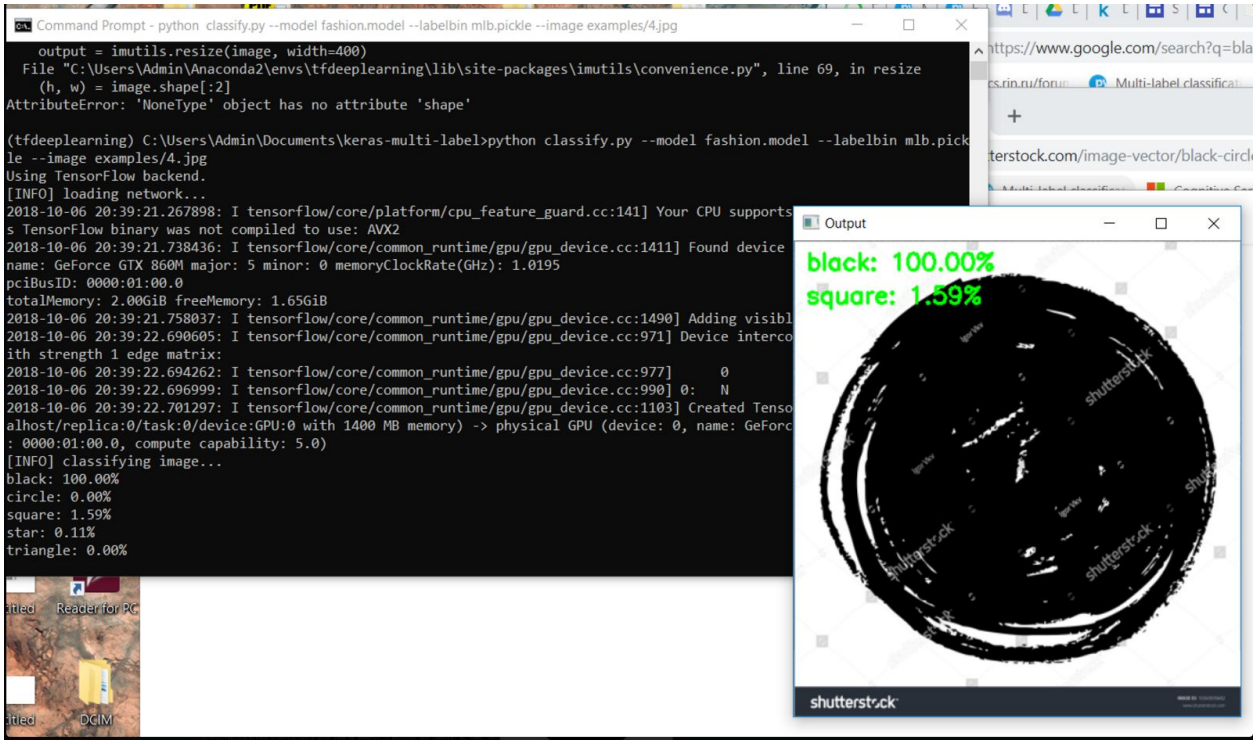
Note: If the classifier does not work at first, simply restart the classifier if this happens and it should work again.

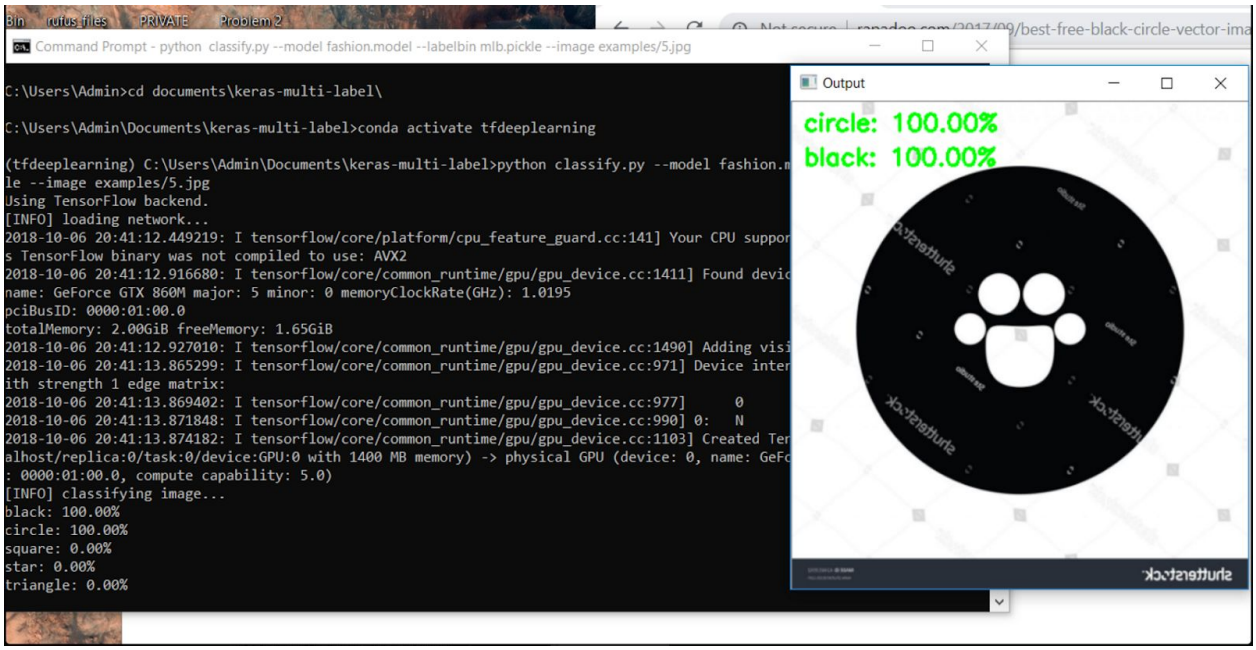
Here are some screenshots of the output result of random images that we have fed in (all of which are failures because they use terrible image data):

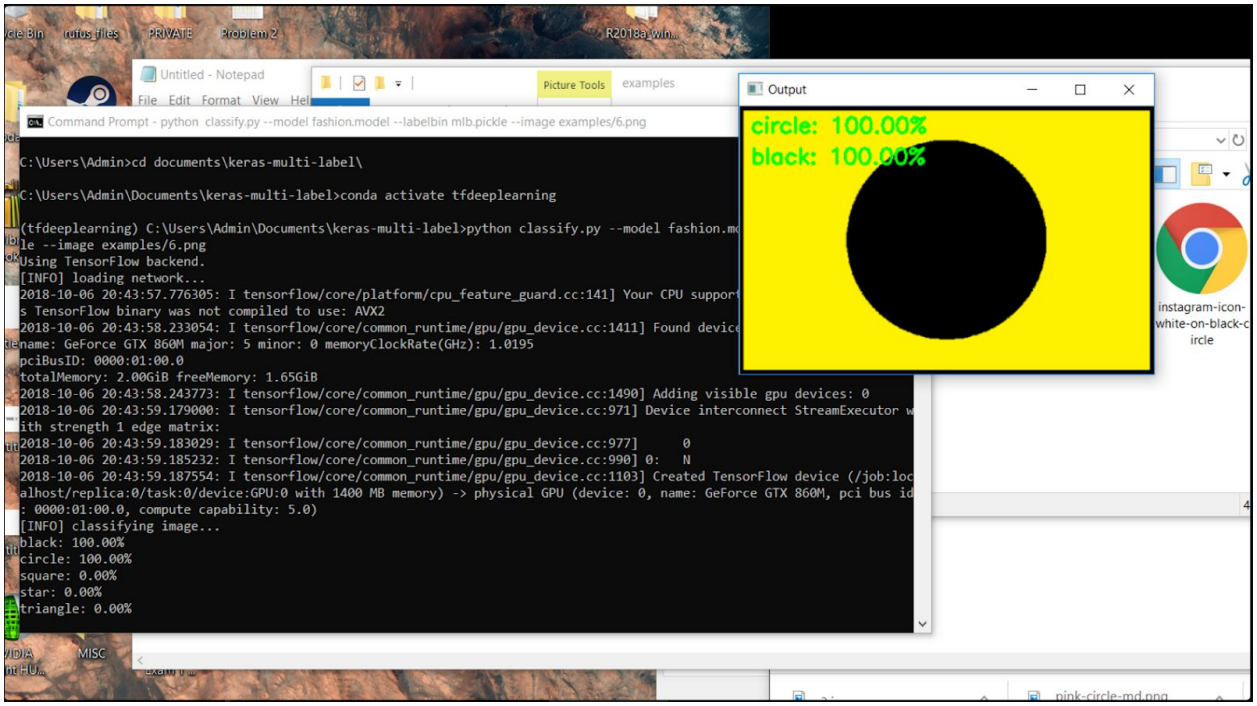

Command Prompt - python classify.py --model fashion.model --labelbin mlb.pickle --image examples/1.png
C:\Users\Admin>conda activate tfdeeplearning
(tfdeeplearning) C:\Users\Admin>cd documents\keras-multi-label\
(tfdeeplearning) C:\Users\Admin\Documents\keras-multi-label>python classify.py --model fashion.model --labelbin mlb.pickle --image examples/1.png
Using TensorFlow backend.
[INFO] loading network...
2018-10-06 20:35:10.247215: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that thi
s TensorFlow binary was no
2018-10-06 20:35:10.728600
name: GeForce GTX 860M maj
pciBusID: 0000:01:00.0
totalMemory: 2.00GiB freeM
2018-10-06 20:35:10.753746
2018-10-06 20:35:12.064462
ith strength 1 edge matrix
2018-10-06 20:35:12.068345
2018-10-06 20:35:12.070622
2018-10-06 20:35:12.074179
alhost/replica:0/task:0/de
: 0000:01:00.0, compute ca
[INFO] classifying image..
black: 100.00%
circle: 0.00%
square: 100.00%
star: 0.00%
triangle: 0.00%
1] Found device 0 with properties:
0] Adding visible gpu devices: 0
] Device interconnect StreamExecutor w
] 0
] 0: N
3] Created TensorFlow device (/job:loc
0, name: GeForce GTX 860M, pci bus id
Output
black: 100.00%
square: 100.00%
DCIM

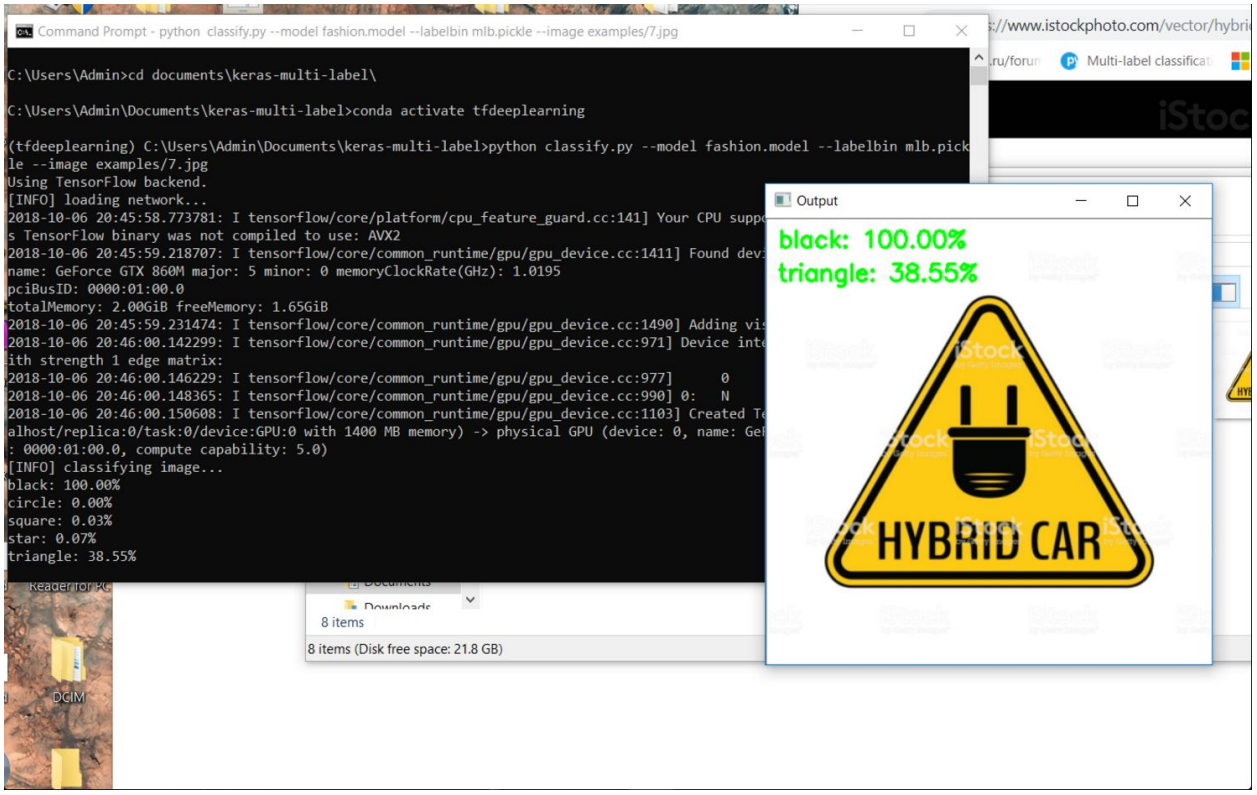


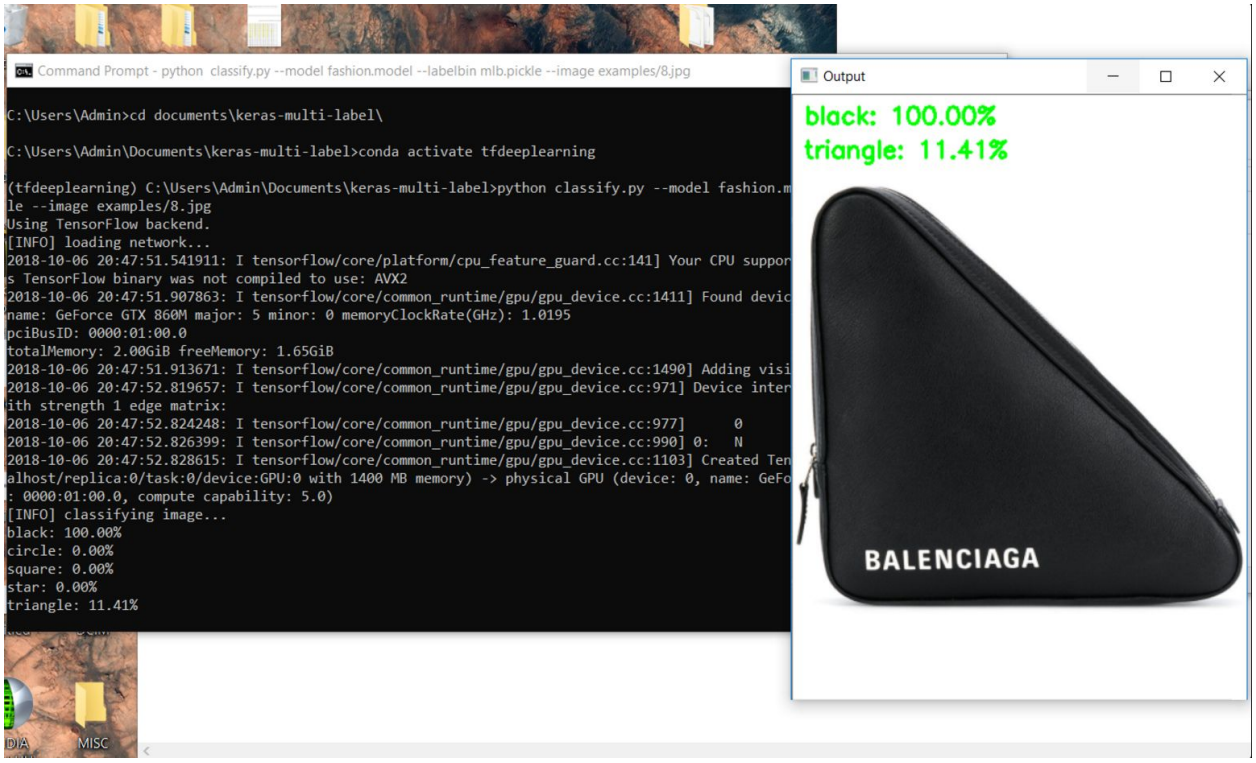
Here are the results of images that we have fed in from a slightly better dataset of random images that yield better results:

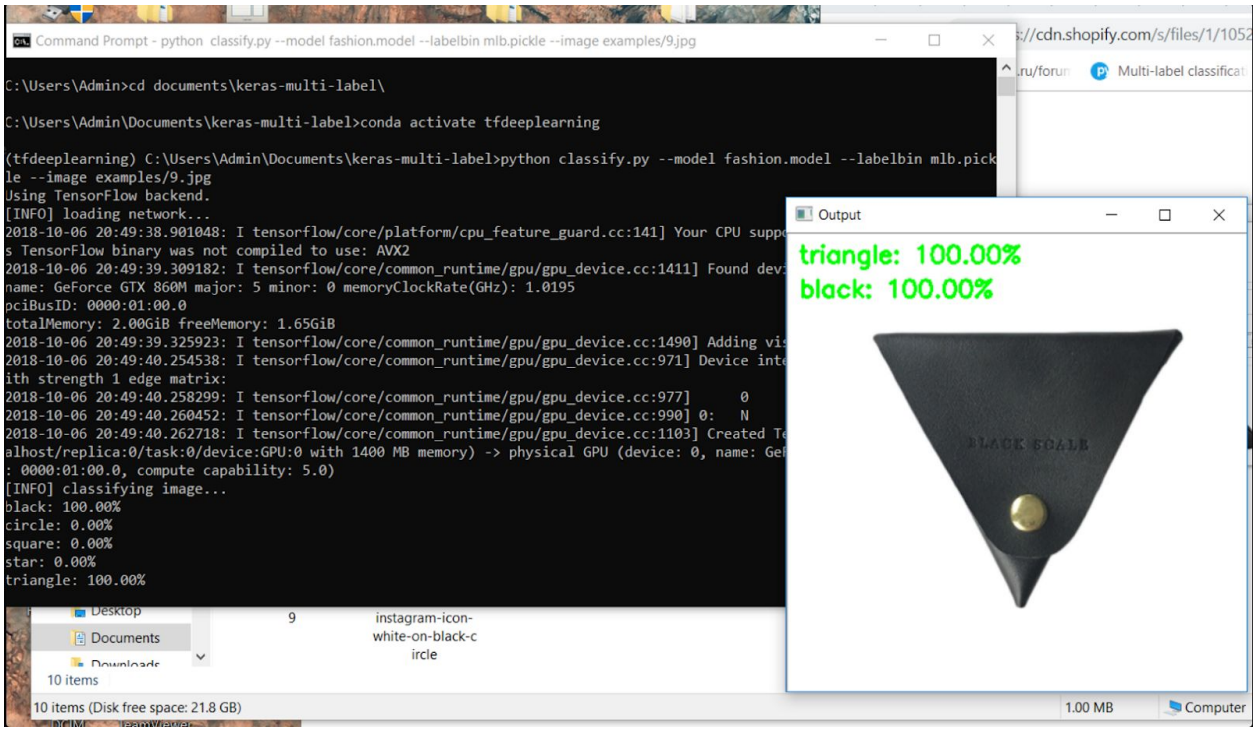

Command Prompt - python classify.py --model fashion.model --labelbin mlb.pickle --image examples/4.jpg
output = imutils.resize(image, width=400)
File "C:\Users\Admin\Anaconda2\envs\tfdeeplearning\lib\site-packages\imutils\convenience.py", line 69, in resize
(h, w) = image.shape[:2]
AttributeError: 'NoneType' object has no attribute 'shape'
(tfdeeplearning) C:\Users\Admin\Documents\keras-multi-label>python classify.py --model fashion.model --labelbin mlb.pickle --image examples/4.jpg
Using TensorFlow backend.
[INFO] loading network...
[INFO] classifying image...
black: 100.00%
circle: 0.00%
square: 1.59%
star: 0.11%
triangle: 0.00%
Output
black: 100.00%
square: 1.59%
Reader for PC
DCIM


Command Prompt - python classify.py --model fashion.model --labelbin mlb.pickle --image examples/5.jpg
C:\Users\Admin>cd documents\keras-multi-label\
C:\Users\Admin\Documents\keras-multi-label>conda activate tfdeeplearning
(tfdeeplearning) C:\Users\Admin\Documents\keras-multi-label>python classify.py --model fashion.model --image examples/5.jpg
Using TensorFlow backend.
[INFO] loading network...
[INFO] classifying image...
black: 100.00%
circle: 100.00%
square: 0.00%
star: 0.00%
triangle: 0.00%
Output
circle: 100.00%
black: 100.00%


Command Prompt - python classify.py --model fashion.model --labelbin mlb.pickle --image examples/6.png
C:\Users\Admin>cd documents\keras-multi-label\
C:\Users\Admin\Documents\keras-multi-label>conda activate tfdeeplearning
(tfdeeplearning) C:\Users\Admin\Documents\keras-multi-label>python classify.py --model fashion.model --labelbin mlb.pickle --image examples/6.png
Using TensorFlow backend.
[INFO] loading network...
2018-10-06 20:43:57.776305: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports TensorFlow binary was not compiled to use: AVX2
2018-10-06 20:43:58.233054: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1411] Found device name: GeForce GTX 860M major: 5 minor: 0 memoryClockRate(GHz): 1.0195
pciBusID: 0000:01:00.0
totalMemory: 2.00GiB freeMemory: 1.65GiB
2018-10-06 20:43:58.243773: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1490] Adding visible gpu devices: 0
2018-10-06 20:43:59.179000: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971] Device interconnect StreamExecutor with strength 1 edge matrix:
2018-10-06 20:43:59.183029: I tensorflow/core/common_runtime/gpu/gpu_device.cc:977] 0
2018-10-06 20:43:59.185232: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] 0: N
2018-10-06 20:43:59.187554: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1103] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1400 MB memory) -> physical GPU (device: 0, name: GeForce GTX 860M, pci bus id: 0000:01:00.0, compute capability: 5.0)
[INFO] classifying image...
black: 100.00%
circle: 100.00%
square: 0.00%
star: 0.00%
triangle: 0.00%
Output
circle: 100.00%
black: 100.00%
Untitled - Notepad
Picture Tools
examples
instagram-icon-white-on-black-circle


Command Prompt - python classify.py --model fashion.model --labelbin mlb.pickle --image examples/7.jpg
C:\Users\Admin>cd documents\keras-multi-label\
C:\Users\Admin\Documents\keras-multi-label>conda activate tfdeeplearning
(tfdeeplearning) C:\Users\Admin\Documents\keras-multi-label>python classify.py --model fashion.model --labelbin mlb.pickle --image examples/7.jpg
Using TensorFlow backend.
[INFO] loading network...
2018-10-06 20:45:58.773781: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports TensorFlow binary was not compiled to use: AVX2
2018-10-06 20:45:59.218707: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1411] Found device name: GeForce GTX 860M major: 5 minor: 0 memoryClockRate(GHz): 1.0195
pciBusID: 0000:01:00.0
totalMemory: 2.00GiB freeMemory: 1.65GiB
2018-10-06 20:45:59.231474: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1490] Adding visible gpu devices: 0
2018-10-06 20:46:00.142299: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971] Device interconnect StreamExecutor with strength 1 edge matrix:
2018-10-06 20:46:00.146229: I tensorflow/core/common_runtime/gpu/gpu_device.cc:977] 0
2018-10-06 20:46:00.148365: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] 0: N
2018-10-06 20:46:00.150608: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1103] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 1400 MB memory) -> physical GPU (device: 0, name: GeForce GTX 860M, pci bus id: 0000:01:00.0, compute capability: 5.0)
[INFO] classifying image...
black: 100.00%
circle: 0.00%
square: 0.03%
star: 0.07%
triangle: 38.55%
Output
black: 100.00%
triangle: 38.55%
HYBRID CAR
8 items
8 items (Disk free space: 21.8 GB)
DCIM

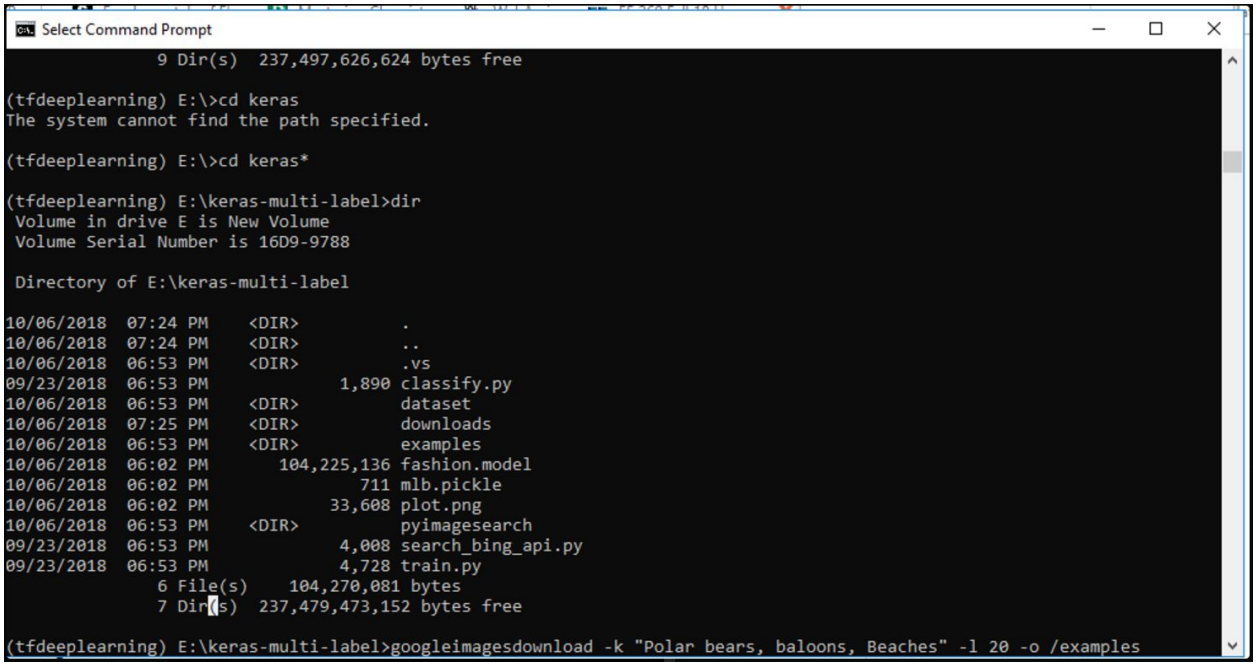




**As of right now, we tested this classifier on things that it wasn't trained for since we had trained it on a dataset from Kaggle with only solid black squares, circles, triangles, and stars.

Important Note: This classifier must be trained on to recognize each shape and color. So, in this case we have only trained it to recognize the color black since that is the color of all of our shapes within this test data set. However, if we feed it in an image of a shape of a different color, the classifier doesn't know what to do and it ends up classifying an image as the wrong color and shape. This most likely is due to the fact that we did not train it to recognize multiple colors, which causes it to become confused. When the image contains a white background, and a shape and color that the classifier is trained on, it works fine. However, once an image is fed in with color in the background, the classifier becomes completely wrong and gets confused. If we use a much better dataset that is more related to the actual images that we will need to classify, the classifier program should perform as expected (although we have yet to prove this since we need better image data).
In the case of the example image dataset with the clothing, each image contained multiple colors and even people with a white background, with the focal point being the clothing and its color. However, it was accurate. Our dataset that we used seems too simple given that each image only contains one shape in a solid black color with a blank white background. This is most likely the reason why the classifier is currently not performing as expected when feeding it in a random image of a colored shape, and why it gets both the color and the shape completely wrong.

Image Data Collection:
Here is how we got image data online through google images (via breaking the google terms of service):
https://github.com/hardikvasa/google-images-download
This Github repository lists the complete instructions to download and scrape images off of google images using the Windows command prompt command line.
Important Note:
You need to specify which directory you want to save the pictures to by typing: -o /[whatever directory you want to save the images to] at the end (-o is output directory)
If you don't do this, finding where the images downloaded to will be a pain.
You can even choose which format to save the images as (i.e. jpg, png, etc.), and specify how many images you want to download (the number right before the -o).
Here is a screenshot of the code to run in command prompt to scrape images:



In this screenshot, we scraped images of polar bears, balloons, and beaches. This will obviously be replaced by the topic of whatever images you want to scrape (in this case, red squares, blue triangles, yellow circles, etc.)

We downloaded about 1000 images of each possible shape and color to use as (poor quality) testing data (for now) until we get ahold of a better and more accurate dataset that is similar to what we need.
This dataset contains about 16000 images which includes:
1000 red circles
1000 red squares
1000 red triangles
1000 red crosses
1000 green circles
1000 green squares
1000 green triangles
1000 green crosses
1000 blue circles
1000 blue squares
1000 blue triangles
1000 blue crosses
1000 yellow circles
1000 yellow squares
1000 yellow triangles

1000 yellow crosses

Note: This dataset is not a well controlled set, as we scraped off these images from google images using the command line to get as much data as possible. So, this data will not yield accurate results with this classifier, especially since our classifier was not trained on this particular dataset as mentioned earlier. Once we get ahold of a much better dataset, we will be able to have a much better idea of how this classifier truly performs. We could also manually go through all 16000 images and filter out whatever is not relevant to our shape categories, although this will take a very long time.

Alternatives to data collection: We are considering using a Raspberry Pie to crawl around the web and collect approximately 20000 images of shapes to make data collection easier for us. It sounds ridiculous, but it might actually work. Update: We have attempted this, and it was not time efficient at all to set up or run.

Here is a link to a blog post for easily creating a deep learning dataset using Google images with Javascript:
https://www.pyimagesearch.com/2017/12/04/how-to-create-a-deep-learning-dataset-using-google-images/

Problem: When testing the classifier on our new image dataset with images we scraped off of google, it won't run and returns an error. This is most likely due to the fact that some of the images that we downloaded got corrupted or didn't fully download, which is breaking that data set.

Something to Consider:
The author mentions that there are multi-label classifiers and multi-class classifiers. Multi-label classifiers can look at one particular class and predict multiple labels from it. Multi-class classifiers can work with two or more class labels in the dataset.

Here is a link to a stackexchange post describing the difference between a multiclass and multilabel classifier:
https://stats.stackexchange.com/questions/11859/what-is-the-difference-between-multiclass-and-multilabel-problem

Here is a link to an article describing multiclass classification using a color example:
https://shapeofdata.wordpress.com/2013/06/04/multi-class-classification/

Important Note: Multi-label classifiers focus on one data point (a single shape in this case), and the possible labels associated with that data point. Multi-class classifiers can classify more than two classes (multiple objects, or shapes in our case), but they are locked to only one label. Basically, multi-label classifiers such as this one appear to only be able to recognize one shape at a time, but can also do color recognition. Multi-class classifiers seem to have the ability to classify multiple shapes where each shape is one class; however, it does not appear to be able to do color recognition.
This means that we need to use a multi-class classifier instead and either find a completely new classifier and abandon this one, or find an example of a multi-class classifier and implement its multi-class functionality into this multi-label classifier if it is at all possible.

Notes by the Author on the Classifier:
The author mentions that fashion.model file is a binary HDF5 file generated by Keras, and that it includes the model architecture and the weights for each layer. This file is only meant to be read, so it probably cannot be renamed.

The author also suggests to use a multi-class classifier instead of a multi-label classifier for shape recognition.

The author also mentions that a multi-class classifier is used when there are two or more labels in a dataset. The author also states that this model is trained to predict only one of these labels, and that the softmax output is often used for this task since it returns probabilities that can be more easily read. In the case of multi-label classification, the goal is to train a model which predicts multiple labels where each data point has one or more class labels.

Machine Learning Methods:

Here are two links to Quora posts that explain the difference between Support Vector Machines and Convolutional Neural Networks as well as various types of machine learning methods:
https://www.quora.com/What-is-the-difference-between-deep-learning-and-SVM
https://www.quora.com/What-is-the-difference-between-CNN-and-a-support-vector-machine

Original Goal for Tensorflow:
The original plan was to get more images, refine the dataset, and retrain the network on this dataset and test it out. We also needed to see how it handles multiple shapes in an image and figure out where to go from there. **However, all of the data that we collected needed to be converted and re-formatted to a uniform standard such as JPG's so that it could be processed by the classifier.** We would also need to find a multi-class classifier and use that instead, or modify this classifier to make it into a multi-classifier to give it the ability to classify

multiple images at once. Once this is done, we would need to factor in segmentation to be able to locate where a shape is within the image and ultimately integrate this with color recognition.

Multi-class classifier:
Here is a link to a blog post that has a basic example of a multi-class classifier:
https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/
The comments section of this post also contains useful questions, information, code, solutions to problems, and links to the author's other posts. This seems like a good starting point to get something up and running for the multi-label classifier. However, there were not many examples at all of multi-class classifiers. Most (pretty much all) of the examples of multi-classifiers that we have found are all multi-label classifiers.

Segmentation:
Here is a link to a blog post that explains how to find shapes in images using python and openCV:
https://www.pyimagesearch.com/2014/10/20/finding-shapes-images-using-python-opencv/
This doesn't give the exact position of each shape in the image, but it does locate and identify all of the shapes. This could be a starting point for what we need to be able to integrate segmentation into our classifier. We could follow this example and try to implement the code and see what happens and if it works or not.

This example code does:
Recognize black shapes within an image
If two or more figures overlap, they are treated as one object
Detect and draw contours (boundaries) around each of the black shapes
Count the number of black shapes

In our case we could probably adjust the code to look for red, green, blue, and yellow shapes instead of black shapes.

When training it initially, it says "known incorrect sRGB profile." This could be due to the fact that openCV uses BGR instead of the standard RGB (different order of red and blue).

Retraining the classifier with bag file data:
We re-trained the classifier with image data from the bag files using Raymond's script, and it does accept the image data and can be trained with it. However, as of right now the classifier is unable to properly classify multiple shapes, or classify even a single shape. Since we re-trained the classifier using the image data from the bag file, it created labels based on the names of the

files which partly explains why it is classifying poorly. It always classifies an image as “All” and “image” according to the current default file naming convention, and it cannot differentiate between multiple or single shapes like it should. This will require some editing to the file names and labels so that the classifier can actually produce a reasonable output. This performance could mainly be due to the fact that the bag files contain really bad image data which isn’t ideal for training a classifier. Some images also had the shapes too far off in the distance which makes it difficult to correctly classify each shape that it sees. We definitely need to retrain it on a much better dataset to get better results, or could retrain it with a different controlled dataset and test it out with a bag file image to see how it performs in that particular case. This current bag file dataset definitely needs to be filtered as well to yield much more accurate results by manually removing bad or negative images.

Sripts to retrain and test the classifier through Windows Command prompt:

Re-train:
python train.py --dataset dataset --model fashion.model --labelbin mlb.pickle

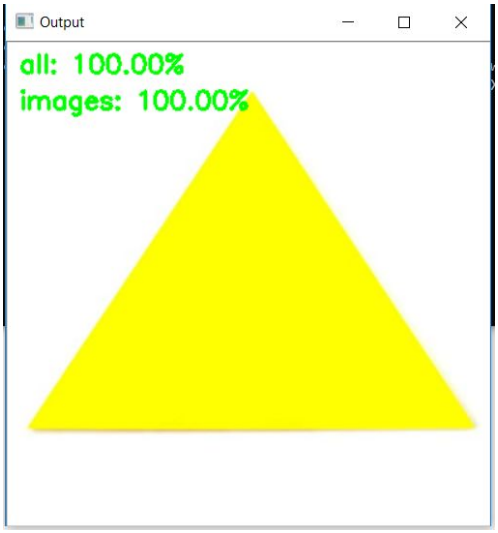
Navigate to environment to use classifier:
Note: This will vary depending on how your path to the classifier is set up on your machine and where you saved the classifier to. In the case of this machine, it was saved into documents, and is under keras. Once we navigate to the proper directory, we can then launch the environment.

cd Documents/keras*
conda activate tfdeeplearning

Test:
python classify.py --model fashion.model --labelbin mlb.pickle --image examples/13.jpg

Note: The code after the “--image” will vary depending on what image you want to feed into the classifier and which folder the image resides in.

Example output:



MATLAB R2018b portion:

Moving over to MATLAB to attempt Shape recognition, we started with installing the R2018b version of MATLAB, which gave us the newest version of the Computer Vision Workspace to Image Label and Train/Test our data. The data we used was real images for MATLAB, which consisted of many different locations within the UHM Campus, Coconut Island, and a few more places (including a well lit night time environment whose dataset is in a different training set).

Displayed below are two of the images that we took:

To collect these images, we used the same camera that is going to be used in the competition, the Logitech C920 USB webcam (which was outfitted with a waterproof housing and requires a computer in order to take photos). Using this camera, we tried to vary our dataset with different scenery, lighting, shapes, and angles, and distances. With these images, we saved them into one working directory on our computer following up our next step in MATLAB R2018b. Our goal was to get as many images as we could in a short amount of time (preferably 5000), but we only got up to about 2000 images.

So, after we collected over 2000 images in these many different locations, we needed to feed our data into MATLAB to train and test with. In this part, we used the Image Processing and Computer Vision "Image Labeller" applet to be able to label most of the images we collected (while keeping some of them on the side for testing if our network was actually valid).

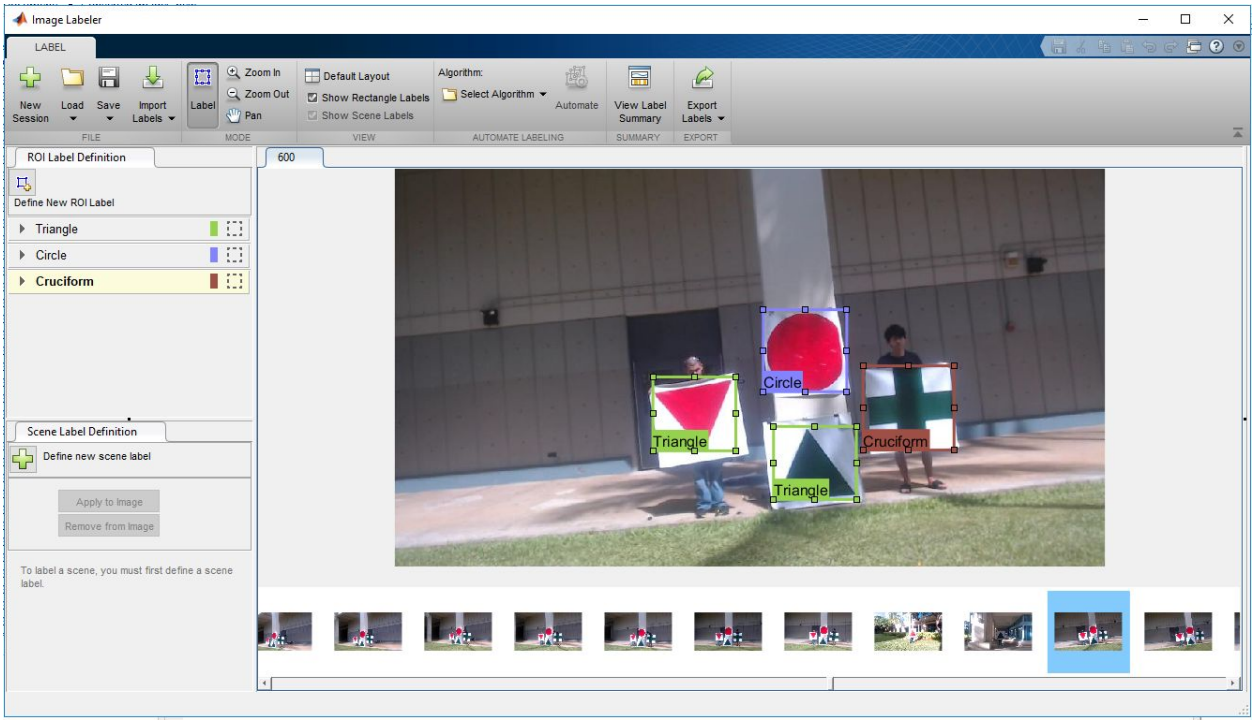
****Important Notes:** In order to be able to label each image and train the classifier, images taken by the USB cameras must be downscaled to 960x540 resolution from the native 1920x1080 resolution. If these images are not downscaled to this exact size, you will not be able to feed them into MATLAB and label them to train the classifier. This downscaling is taken care of by a MATLAB script.

**When dealing with images taken from the WAMV simulation, these images are of size 800x800. Therefore, these images need to be cropped to be of size 800x450 and then upscaled

by 1.2 times to get them to be of size 960x540. This is necessary because the MATLAB classifier requires images to be of a 16:9 ratio.

While labeling, we used the labels for the three images from the competition (the circle, triangle, cruciform) and labeled each image manually one by one.

While labeling the images, the applet looked like this:



Within each image, we needed to label the shapes in the image and also label multiple shapes (while not taking into account the color, due to scan the code already being pretty robust and well established for that purpose even though MATLAB could also be used for it). This was one of the benefits to using MATLAB instead of TensorFlow because the ability to recognize multiple shapes in TensorFlow needed an advanced custom neural network configuration. Another advantage to MATLAB was that it could pinpoint the exact location of each shape within an image due to the labelling feature used for training. Tensorflow multi-class classifiers did not have the ability to easily implement this as this “segmentation” implementation would involve modifying the code extensively in a complicated manner which requires a thorough understanding of every line and function in the code.

**For a little bit of background information, MATLAB saves these images as a table for you in your workspace.

After this, we needed to save our image labelling session using our compiled image dataset and then export this as a ground truth file. This will be outputted as the file GroundTruth.mat and would then be loaded into our workspace to begin training. This allows us to run the TrainRCNN script. This script leverages the RCNN (Regional Convolution Neural Network) part of the Computer Vision Applet and trains all images with either CPU or GPU computational power. For our case, we leveraged the CUDA acceleration (do to my personal computer having an NVidia GPU), which made training much faster than what it previously was with just a CPU (Saving us the time equivalent to a couple of days for training the network with just one dataset).

****Note:** Use a GPU for training, since training on a CPU with a dataset this large results in MATLAB crashing every time shortly after starting training and going through the epochs.

This is an instance of us training a new dataset in MATLAB R2018b where the bounding boxes are being extracted:

```
>> m20181005_trainRCNN
*************************************************************************
Training an R-CNN Object Detector for the following object classes:

* Circle
* Cruciform
* Triangle

--> Extracting region proposals from 1978 training images...K>>
```

And after this, we would get epoch image with an accuracy in percentage form and are shown the time elapsed (which would usually be a few hours for one training session).

After this, we would get a workspace on the side filled up with many different parts (which needs to be saved in one .MAT file to save for later) and then exported to another machine.

We currently still need to test our data against the other images that we have but have full confidence in MATLAB giving us more robust results.

Conclusion:

Overall, we learned that MATLAB's robustness of the image labeler and RCNN (Neural Network) single handedly beats TensorFlow by quite a large margin. Overall, the ability to classify multiple shapes in MATLAB without having to know exactly how the Neural Network works is a benefit in itself. In general, we will continue to work towards adding more images to the dataset in the weeks to come and implement this dataset into the Unix shell of the main computer of the WAM-V (Intel NUC running Ubuntu).