**Scheduler.py & Helpers.py**

**amandamt1224@gmail.com**

**Set Up**

* What you need:
  + scheduler.py
  + helpers.py
  + imutils.py
  + “.caffemodel”
* You should make a directory called "output" in your working directory
  + This is where the functions will write out all of the coordinates into their own separate CSV files
* Make sure that the locations of the Caffe files are correct
  + These are located at the top of scheduler.py
  + You shouldn't need to change the locations in scheduler.py if your image will produce more than one batch (the default batch size is 4000) but it's probably good practice to change both of them if you change either
* Usage:
  + "python scheduler.py
  + –i [image name]
  + -b [batch size (default is 4000)]
  + -p [number of processes to use]
  + -t [threshold for deciding whether a prediction indicates a car or not (default is .8)]
  + -s [step size in between frames (default is 8)]
* Your output should be:
  + One marked up image called "output.jpg"
  + One CSV file containing all of the coordinates above the threshold called "coords.csv"
  + Your output directory should contain one CSV for each child process you had.

**Pro Tips**

* **If you are working with a color image model:**
  + In the coordsToFrames function in helpers.py, **comment out** (line 68). It reads: "window = window[:, :, np.newaxis]"
  + You also have to make sure that you aren’t converting the color (line 24)
  + Large images (like the Utah picture) will run into problems if you use the default batch size with a color model. The frames take up a lot more memory in color than in grayscale. I suggest using batch sizes of 2000 for color.
  + The code on GitHub is currently set up to run in color
* If you’re working with images about 2000x2000 and smaller you might want to stick to around 4 processes. More than that tends to be overpowered and actually make the program go slower.
* Calculate number of frames your image will produce:
  + ((width/(STEP\_SIZE))-(FRAME\_SIZE/STEP\_SIZE) -1) \* ((height/(STEP\_SIZE)-(FRAME\_SIZE/STEP\_SIZE) -1)
  + That, combined with the batch size you chose (or if you just used the default –> 4000) should help you get a feel for how many processes you should use

**C++ Code**

**Set Up**

* What you need:
  + scheduler.cpp
  + scheduler.h (probably not needed….)
  + classifier.cpp
  + classifier.h
  + Makefile
* I made sure to note where I found some of the code I borrowed

**Notes**

* Modifications I made to classifier.cpp:
  + The classify function used to return a vector of floats. This is useful if you’re classifying more than one type of thing (e.g. pools and cars) but I changed to just one float
  + The classify function would make a call to “Argmax”. This would help the function make a vector of all the prediction values for all of the things being classified. In our case it would just be “car” and “not car” so I commented out the calls to “Argmax”. If you choose to use it, you should pass in “2” as the value of parameter “N” in the “Classify” function. If you use a value that is higher than the number of things you’re classifying you’ll likely seg fault. If you leave the code the way it is, then what you put as “N” is irrelevant.
* The classifier code needs to be modified more to accommodate batches. Right now I’m just iteratively passing one frame at a time.
* GPU mode seems to be working!
* I tested the code that makes the centroid coordinates against my python code and they seem to work exactly the same. I used a few different images (including the Utah picture) so I feel pretty confident.
* I also checked the code that breaks the frames up into batches. I know it’s a pretty simple function, but at least now we can be 99% sure that any discrepancies we find aren’t caused by silly things
* The prediction values ended up being pretty different in my first few runs….
  + For some reason the values are a lot lower in C++ than in python
  + I ran both on cars.jpg and outputted the prediction values (I’ll call the files c\_output.txt and python\_output.txt)

**Basic Step-by-Step**

1. Read in the image (convert to grayscale if necessary)
2. Calculate the number of frames that your image will have
3. Make an array with all of the coordinates of all of the center points of all of the frames
4. If the number of frames is less than or equal to the given batch size then don’t spin off extra processes and just proceed to Step 7
5. Otherwise make a task queue and spin off the additional processes
6. Break job up into batches, and push the tasks onto the task queue.
   1. Task = (int start, int end) where start and end refer to start and end indices
   2. “Process all the frames from coordsArray[start] until coordsArray[end] exclusive”
7. Make the classifier and set GPU mode
   1. If there is one process GPU mode will always be turned on
   2. If there are multiple processes only the first four processes are told to turn on GPU mode (this number can be changed on line 78 of helpers.py
8. Turn the batch into an array of image frames
   1. Really easy to do as long as every process has access to the original image
9. Take the frame array and put it into the prediction function
   1. In C++ steps 8 and 9 are all done in the “coordsToFrames” function
10. Write the predictions out
    1. In C++ I’m currently just writing dots out to copy of the image
    2. In python I’m currently writing out to CSV’s then using those to write back to the image
    3. Key difference is: in python all of predictions get written out and a separate function goes through and (depending on the threshold value) decides where dots should be placed

Command to get GPU stats:

nvidia-smi -q -g 0 -d UTILIZATION -l

Let me know if you have any questions!!

-Amanda