Building a Dataset

You are going to be building a dataset using cityscapes images. You'll use this dataset to train a neural net that can confirm or deny the existence of a traffic light.

Step 1: Importing data from cityscapes

The images and labels you want to use can be found here:

Images
Cityscapes directory
-- leftImg8bit
-- -- train/test/val
-- -- -- names of German cities
-- -- -- -- *.png

Ground Truth
Cityscapes directory
-- gtFine
-- -- train/test/val
-- -- -- names of German cities
-- -- -- --*labelIds.png

Note that in the Ground Truth section the labels come in multiple forms. You should use the ones that end in "labelIds.png". And we will be creating a train dataset and a val (validation) dataset but you can ignore the "test" data.

You can read the images in a variety of ways (matplotlib._png.read_png_int(), scipy.misc.imread(), etc...) but the important thing is that **your images should be in uint8 format**. If you are importing them as floats, you need to convert them to uint8.

Step 2: For each example where a traffic light exists, use the labels to pick one pixel that falls on a traffic light and one random pixel that does not fall on a traffic light

It's very important when training a neural net to balance the dataset. If we made our dataset only from crops around random pixels, then only one out of a thousand would fall on a traffic light. Then when we tried to train our net, the first thing it would learn is that it should always guess "not a traffic light" because it will be right 99.9% of the time.

To prevent that we want to give it a dataset where half the examples are traffic lights and half are not. You can do this by taking a crop around one traffic light pixel and a crop around one non-traffic light pixel each time. Instead of one of each, you can also do five of each or ten, as long as you have the same number of traffic lights and non-traffic lights. This might be useful data or it might include lots of very similar examples depending on how many you take per example.

Step 3: Crop the images around your chosen pixels

We want to use **crops of 81x81x3**. I chose this size because it is big enough to see some of the scene around the traffic light pixel but it's also small enough that you can train the net on your cpus in a reasonable amount of time. This is something you can change later on if you like but I want everyone to first complete the whole project (including training the net) with this size and then you can experiment.

This crop function should work whether it is given a full pixel (which is what you'll get from the labels) or whether it is given a subpixel, like [23.4, 452.1] (which is what you'll get from the output of Part 1 of the project when you put it all together). In the case of a subpixel, rounding it to the nearest full pixel and cropping around that is a reasonable way to handle it in this case.

When cropping there is an interesting question you should consider: What do you do when the coordinates you want to crop around are too close to the edge of the image? You could in theory move the box so that it's not centered around the coordinates of the traffic light (or non-traffic light) but still includes it. Alternatively you could keep the given pixel in the center of the crop and fill in the part of the crop that lands outside the image with zeros. In this case, you should do the second option. The reason is that the question we really want our net to answer is not "Is there a traffic light somewhere in this crop?", it's "Does the center pixel fall on a traffic light?". From part 1 we will receive candidates for traffic lights not candidates for pixels that are *near* traffic lights. That's what we want to confirm or deny.

Step 4: Save your data

The format of our output, what we put in our data_dir, should be as follows:

Data_dir
-- train
-- -- data.bin, labels.bin

Data_dir
-- val
-- -- data.bin, labels.bin

The train data.bin file will be composed of images from the cityscapes "train" data and the val data.bin from the cityscapes "val" data.

The data.bin files should be binary files where images of the shape 81x81x3, *in uint8*, are saved one after the other. The labels.bin files are also binary and there, a single number -- 0 or 1, in uint8 -- should be saved for each image in data.bin **(make sure you preserve the order, i.e. if the fifth image in data.bin has a traffic light, the fifth number in labels.bin should be 1. This is very important)**.

You may find the numpy.array.tofile() function helpful for this.

Step 5: Make a function to display data from your dataset with the corresponding label

After you've built a dataset you want to check it to make sure everything is right. To do this you want to make a function that gets as input your data_dir and an index and then shows you the train image and its label at that index or the val image and label at that index. If your dataset is constructed correctly you should see that the label and the image match. If not you've got a problem somewhere.

Another useful way to check your dataset is by the size. The data.bin file should be 81x81x3 the size of the labels.bin file.