This page talks about what I did at my internship. Although I cannot post the code I wrote during the internship, I can describe the general workflow.

The overall goal has been to create a neural network that would detect the specific type of motion from a live video stream, as an example using a dog wagging its tail in a certain specific fashion. My focus is on creating a neural network within the TensorFlow low-level API that would be able to identify this motion from a series of video screenshots. My network would then be combined with a video stream by means of OpenCV library, but this integration is not a part of my own work and I do not discuss it here. The work that I have completed so far has been to analyze the location and position of a dog’s tail from a single static image. I am planning to generalize my network to make use of the sequential nature of the screenshots by using Recursive Neural Net approach, but so far I have been using a simple architecture processing images one-by one.

There are three major steps to this work:

* creating the training dataset,
* programming the network architecture and feeding in the training data,
* training the neural network

Before I joined the project, the team had been trying each of these steps following online tutorials, and the results were not satisfactory. I have improved on each of these steps, as I describe below and illustrate with the files in this repository.

**Creating a training dataset.**

To train a neural net, it is necessary to have a dataset that the model can be trained on. The problem I encountered was the lack of a good dataset. A dataset that is useful for my purposes did not previously exist (apart from a small set created by other team members), as it requires for there to be both a picture of a dog as well as the angles and the (x, y) coordinates describing the tail’s location. Instead of creating this dataset manually, one-by-one, I created a script that would do it automatically. I did this by combining my knowledge of python and 3D modeling software (Autodesk Maya) to create a script in a program that combines these two ideas: Blender. I created a script in Blender that would generate these images. The images would all obviously have to be different, so variance needed to be created. The script does so by generating a dog in a random position, with its tail wagged in a random direction. The script also changes the way that dog itself looks, from the color of the dog to the amount of hair it has and the length of the tail. The lighting of the scene is also randomly varied. The script also contains optional features that can be turned on or off, such as adding a background picture, changing the floor, and the addition of more objects that could complicate the scene. The purpose of these features is to make sure my neural network is not confused by the presence of external interferences. The script then would render the resulting scene from varying angles, and generate the pictures together with the correct (x,y) coordinates of the beginning and the end of the tail, as well as the tail’s angle relative to the dog. The angle is the only thing controlled directly by my script, however the resulting generated images allow for the beginning and end coordinates to be determined automatically. This is done by generating a second image in which everything in the scene is made invisible except for the special marker objects placed at the beginning and end of the tail. That image is then put through an external script which finds those objects and extracts their coordinates. The key difference in this style of generating images is that, instead of having to manually find an image and create a file that keeps the angle and the (x,y) coordinates of the tail, this is all done all at once by the script. This allows me to increase the training dataset to an arbitrarily large size without investing an additional human effort. Another advantage of this approach (that I am not utilizing so far) is that it allows to create videos and for each timeframe of the video create a label of whether the tail is being static, or is being wagged vertically, or is being wagged horizontally. This can be used to train a recursive neural net. Doing this manually would be extremely time-consuming, but my approach resolves this problem.

In order to create a variance between the different images, the training images would then be cropped in various ways to account for variations in how a picture could be taken. (This is a standard step in training neural networks for visual recognition). Examples of these pictures can be seen in this repository.

**Programming the network architecture and feeding in the training data**

I used a pre-existing convolutional neural network model called Squeezenet, a very accurate yet compact model. The goal lied in trying to make squeezenet behave in the way that I wanted it too. I chose to use the features generated by squeezenet, but attach additional layers on top of the first 21 layers (out of 23) of Squeezenet .

Training in memory is, as the name implies, very memory intensive due to the large size of the dataset required to train a good model. This meant that I had to split up the process, extracting the features calculated by a specific layer of Squeezenet and saving them in the TensorFlow record (TFRecord) format. I then train my network by feeding the data from the TFRecord. This allows for me train only my sections of the neural network instead of altering the pretrained weights of Squeezenet.

**Training the neural network**

I made sure to use an architecture that allows for over-optimization while training on small datasets, which helps ensure that the model knows how to analyze specific variations within a dataset. I use the AdamOptimizer, which is the most advanced method, and varied hyperparameters to ensure efficient optimization. I train the model on my own GPU that I configured to work with TensorFlow.