- Dataset Summary:
    - Here I used the csv library and some fundamental python to get a feel for the data that was provided for this project. I used the csv library to read the signnames.csv file that was given to create a dictionary that mapped all the class values to their string counterpart. This all gave me the result of:
        - Number of training examples: 34799
        - Number of testing examples: 12630
        - Image data shape: (32, 32, 3)
        - Number of classes: 43
- Data Visualization:
    - To visualize the data I plotted out a few different images with their titles. This allowed me to see what the actual images themselves looked like before any normalization. This also added the benefit of testing out the dictionary that I had created in the previous step.
- Preprocessing the data:
    - Here I decided to normalize the pixel values of each image to be between 0 and 1. This would simplify the images and allow the network to train better than if the pixel values were larger. I also experimented with the difference between color and grayscale images in relation to how the network would over perform. I decided to keep the images in their color format because it seemed to be positively effecting the validation accuracy of the network.
- Model Architecture:
    - At first I started with an architecture that was similar to the one given in the lessons but it didn't seem to perform as well as I wanted it to. This led me to making the model bigger and deeper. I padded the images using 'SAME' padding which allowed me to add another layer. My layers ended up going 32=>64=>128 for the convolutional layers, and then a flatten layer of 2048. The flattened layer was then connected to a 256 node fully-connected layer and then to a 128 node fully-connected layer, which then led to the output layer of 43. This seemed to all benefit my network well.
- Model Training:
    - Training the model is what took me the longest. I ended finding that my validation accuracy seemed to be best with a learning rate of 0.001 and a batch size of 128. I used the Adam optimizer and never tried a different one, so perhaps that could be a good way to see if I could get better performance using a different optimizer. The epochs didn't seem to have as big of an effect on the accuracy of the model. I noticed that my model would train in roughly 10 epochs and then hover for the most part after that, with only small improvements up until 20 epochs, this led me to settle at the 15 epoch mark, since I figured the extra time and computation required for 5 more epochs wasn't very efficient
- Solution:
    - I ended up getting to the 93% mark which was asked of the project. My results vary a bit towards the last few epochs of my network from around 93.5 all the way upwards of 95.5.

    - When finally deciding on my model and hyper-parameters I fed the test images through the network and ended up with an accuracy of 93.53%, which I found satisfying and a good sign that my model was doing well
- New Images:
    - I found new images online and tested my network on them with some mixed results. Looking back, I believe that a few of these images would most likely be rather difficult for the network to recognize. Specifically because two of the images seemed to be somewhat on an angle that might confuse the network. I also don't believe that images like this would be taken if it was by a driving car since all the images of signs from the perspective of a car would most likely be a bit more facing of the car. I also think that two of my images could have been cropped to be closer and to get rid of some of the noise around the image. For example, the last image that I used was a 70kmh speed limit sign, however, the sign only takes up about 50% of the images and there is a clear tree line on the bottom half of the image.
- Performance of images:
    - Like I said, the performance on these new images could have been better. It would jump around between 40% and 60% on the five images that I used.
- Model Certainty:
    - The models certainty was a bit uplifting. My model predicted the first two images correctly. The 4^{th} image was one of the ones that I talked about previously that was taken on an angle. The model thought that this sign was a dangerous curve to the right sign instead of the animal crossing sign that it was. And when looking at this, it is understandable why it would believe that. My model didn't not have the correct prediction in its top 5 however. As for the 5^{th} image, it classified wrongly, but the correct prediction was its second guess. So given these observations I believe that the 40% accuracy that my model achieved is a bit pessimistic, and given some better images and perhaps some image cropping, the accuracy might rise to be 80% or so.