Project 2: Udacity – Self-Driving Car
Convolutional Neural Networks (Update 1)
Graham Nekut

Submission
Submission Files
GN – All items for the model and new images have been saved and uploaded.

Dataset Exploration
Dataset Summary (Cell 2)

GN – Step 1 includes a summarization of the image size (32 x 32 x 3 to account for RGB channels), number of classifiers (43) and the size of the training (34799), validation (4410), and test (12630) data sets.

- Correction: Changed np.max() to len(np.unique()) to avoid using the assumption that the class identifiers are sequential.

Exploratory Visualization (Cell 3)
(Suggestion: To further explore the dataset visually, you could plot a couple examples from each class. Tip: check out numpy.where())

GN – Plotted the distribution of classes of all of the datasets (train, valid, and test) to ensure they were all representative of one another. I moved the image visualization to Step 2 preprocessing to allow for side by side comparison of color and greyscale images.

- Correction: Not sure how numpy.where() would help in this scenario but will explore it further. I do understand that getting an idea of what a larger set of the inputs look like will maybe help refine my model/parameters.

Design and Test a Model Architecture
Preprocessing (Cell 4)
(Suggestion: Preprocess to grayscale with Tensor Flow – GN: Now that I know I'll consider that for next time)

GN – Defined and used a function that will convert all images to greyscale (averaged each pixels RGB channels) and performed a normalization of the entire data set (scaled all images to [-1, 1] by taking the value of each greyscale pixel, subtracting the minimum image pixel value, multiplying by ratio of new to old value ranges, and adding that to the minimum of the new range values. Additionally, divided by the standard deviation to scale the new feature value to st. dev. from mean). While color can certainly help identify the various types of traffic signs, I used greyscale images to see how well a single channel image would perform before increasing the complexity and impacting training performance (ran on my CPU). The normalization helped center the image values around zero, which helped reduce the risk of weight becoming overly inflated in one dimension while under scaling in another. **Question:** Am I implementing the normalization correctly? I'm mainly concerned over the division of an image by it's standard deviation. When I did not have the SD division the model reliably performed over 90% accuracy in validation and testing, with it, it is in the mid-80s. Note that my submission has it commented out.

I chose to move the image visualizations to this section so I could look at the distribution of normalized pixels in a given image and to see the images side by side.

Model Architecture (Cell 5)

GN – I spent a *LOT* of time tweaking different filter and stride sizes (3x3 up to 9x9) to see the impact they would have on performance and if I could do any better than what the initial LeNet architecture had provided. Sadly, I was not able to and needed to push forward and complete the project. While the convolution and subsampling layers are important, there are many other hyper parameters and functions that I ended up tweaking to help refine my model. I am excited to learn more about the actual theory about convolutional networks and some of the methods people use to select different architectures. **Note:** I would have liked to implement an inception module to see how that impacted accuracy and will likely revisit that when time permits.

One item that I added to the model architecture was dropout after seeing that my model appeared to be overfitting the training data (divergence in training and validation data accuracy)

- Correction: My final model consisted of the following:
  - Layer 1 – Convolution [Input 32 x 32 x 1], [6 x Filter: 5 x 5], [Stride: 1 x 1], [Output: 28 x 28 x 6], followed by a rectified linear unit (ReLU) activation function (y = x for x > 0, otherwise 0)
  - Layer 2 – Max Pooling [Input 28 x 28 x 6], [Pool: 2 x 2], [Stride: 2 x 2], [Output: 14 x 14 x 6]
  - Layer 3 – Convolution [Input 14 x 14 x 6], [16 x Filter: 5 x 5], [Stride: 1 x 1], [Output: 10 x 10 x 16], followed by a ReLU

- Layer 4 – Max Pooling [Input 10 x 10 x 16], [Pool: 2 x 2], [Stride: 2 x 2], [Output: 5 x 5 x 16], followed by a dropout with a keep probability of 0.5 (meaning half of the values are set to zero and the others scaled by 2) to help prevent overfitting.
- Flatten the 5 x 5 x 16 convolution output to a single vector array.
- Layer 5-7 – [X: 1 x 400, 1 x 120, 1 x 84][W: 400 x 120, 120 x 84, 84 x 43], [b: 1 x 120, 1 x 84, 1 x 43]. Three fully connected neural networks (XW + b). These layers differ from convolutions as they have fully connected weights and inputs, greatly increasing the computational rigor needed to determine weights and it does not benefit from weight sharing that can often be useful to identify features at different depths.
- Note: Will provide a visualization of the network if needed but do fully grasp what the transformation at each stage looks like.

Model Training (Cell 6) - The submission describes how the model was trained by discussing what optimizer was used, batch size, number of epochs and values for hyper-parameters.

GN – [Optimizer: Cross-entropy with softmax fed into the Adam algorithm (need to read more about it) with TensorFlow], [batch size: Tried 64, 128, and 256 but ended up sticking with 128 because it seemed like the largest my machine could handle], [Epochs: Used 10 for most of the training runs as that seemed to do be where I found the accuracy plateau], [Learning rate: Tried LR decay at a rate of 1/x, 1/sqrt(x), and 1/x^2 where x varied based upon what epoch the iteration was on.][**Note:** I would have like to implement momentum]

- Correction: Based upon feedback I have implemented a control structure that will end the EPOCH iterations if the average difference of the last N training accuracy calculations is less than P percentage point. Initial tests with N=3, P=1 seemed to be strict, but adequate. Could consider reducing N or increasing P to alter behavior to be more relaxed. Initial tests showed to only need four iterations, instead of the total number of 15 maximum set at a hyper-parameter. This helps with reducing overfitting and time wasted.

Solution Approach - The submission describes the approach to finding a solution. Accuracy on the validation set is 0.93 or greater.

GN – Unsure of what this is asking. The approach I used was the same backbone that was used in the Tensor Flow lab and what was taught in the classes. The use of LeNet architecture, average cross-entropy softmax to determine the 'distance' and loss from the probabilistic output and one-hot encoded classifiers, and the Adam algorithm optimizer (need more insight) were the key steps in finding the solution. Over many of the runs I approached 90% accuracy on the validation data but seemed to oscillate between the 85 and 95.

- Correction: After refining my training model but limiting the number of EPOCHS based upon the changes described above, I was able to better accuracy that could likely be attributed to overfitting in previous model setups.

Test a Model on New Images

Acquiring New Images (Step 3, Cell 1) - The submission includes five new German Traffic signs found on the web, and the images are visualized. Discussion is made as to any particular qualities of the images or traffic signs in the images that may be of interest, such as whether they would be difficult for the model to classify.

GN – Finding images online was fairly simple however many of them had watermarks so I was unable to use most of them. After downloading, I scaled all images to be 32 x 32 pixels. All of the images have different perspectives (angled, head-on, etc.) and background/foreground qualities so I thought it would be a good collect to test how well the model performed.

Performance on New Images (Step 3, Cell 2-3) - The submission documents the performance of the model when tested on the captured images. The performance on the new images is compared to the accuracy results of the test set.
(Suggestion - For increased robustness, I suggest you use the image augmentation technique to rebalance the number of classes and to expose your model to a wider variety of image qualities. Check out [this article](#) for a great explanation with examples.)

GN – Depending on the hyper parameters and which run of the model I was performing, I averaged 80% accuracy on the five images, sometime being as low as 40%. Granted, I believe that more than five images are needed to verify, but given that the project already included a substantial test set, I found this to be more of just an interesting exercise.

- Correction: Definitely like the idea of altering the images to make sure the model is robust enough. Many of the images I included were fairly clear with good contrast and color. In addition to editing the photos it would be nice to use a larger collection of images, save them to a directory with the class identifier as the file name. I can then iterate over all objects in the folder and compare its file name to the predicted class. This would prevent any hardcoding of y_new in the code.

Model Certainty (Step 3, Cell 4) - Softmax Probabilities - The top five softmax probabilities of the predictions on the captured images are outputted. The submission discusses how certain or uncertain the model is of its predictions.

GN – With this submission, the model certainty for the five images I found online were interesting. The roundabout (image 0, classifier 40) and the do not enter (image 3, class 17) signs were 100% certain, while the speed limit 60 (image 4, class 10) was categorized as 'No passing for vehicles over 3.5 metric tons' but had a certainty of 96%. This is not too surprising since there are many signs with similar shape, a red outline and text in the middle. The slippery road (image 1, class 23) and turn left ahead (image 2, class 34) were lower certainty (42 and 56%, respectively), almost a guess when looking at the second closest certainty (25 and 40%, respectively) and is likely a result of having very few of those in the data set in proportion to the others (see histograms in step one showing the distributions of classes in each training, validation, and test sets.

- Correction/Suggestion: If I had more than five sample images a visualization would definitely help to identify the probabilities a little easier. I will leave it out of this analysis for the time being.
- **CORRECTION:** After refining my model based upon EPOCH suggestion, I achieved a greater prediction accuracy (usually in the range of 80%), with an approximate photo classification probability range between 60-100%.