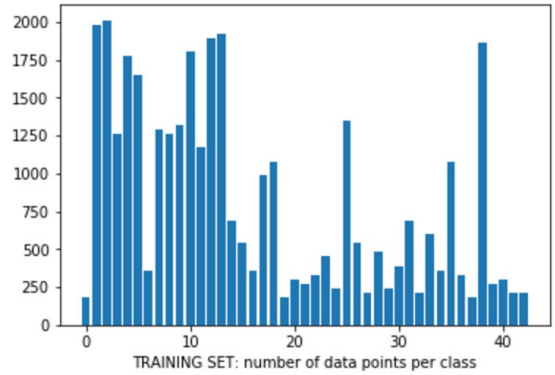
PROJECT 2: TRAFFIC SIGN CLASSIFIER USING DEEP LEARNING NEURAL NETWORK TECHNIQUES

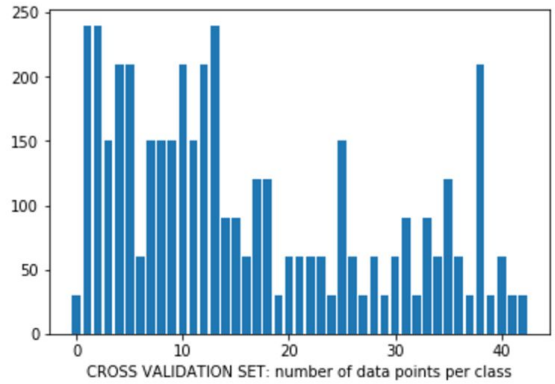
March 5, 2017 Shulamith Mithi Sevilla

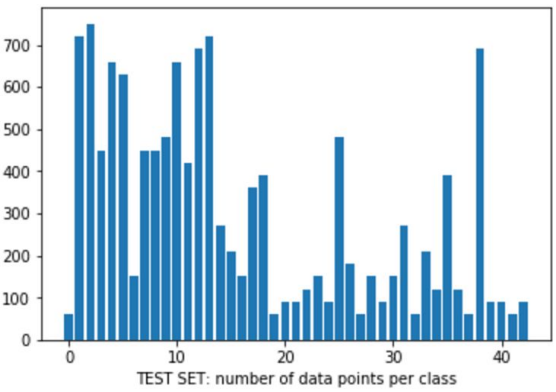
- Exploration of data
- Designing and testing the model architecture
- - preprocessing techniques
- - details about the neural network
- - details about training the network
- Test on new images from the internet
- Appendix Images, Recommendation and References

EXPLORATION OF DATA

HISTOGRAMS







SEE APPENDIX FOR
A: 450 random images from the training set
B. 15 random images per classification (43 classifications)

BASIC STATISTICS

TRAINING DATA SET SIZE	34799
CROSS VALIDATION DATA SET SIZE	4410
TEST DATA SET SIZE	12630
IMAGE SHAPE	32 x 32 x 3
NUMBER OF LABELS	43

TOTAL INSTANCES	END COUNT	LABEL #	NAME
210	210	41	End of no passing
690	900	31	Wild animals crossing
330	1230	36	Go straight or right
540	1770	26	Traffic signals
450	2220	23	Slippery road
1980	4200	1	Speed limit (30km/h)
300	4500	40	Roundabout mandatory
330	4830	22	Bumpy road
180	5010	37	Go straight or left
360	5370	16	Vehicles over 3.5 metric tons prohibited
1260	6630	3	Speed limit (60km/h)
180	6810	19	Dangerous curve to the left
1770	8580	4	Speed limit (70km/h)
1170	9750	11	Right-of-way at the next intersection
210	9960	42	End of no passing by vehicles over 3.5 metric tons
180	10140	0	Speed limit (20km/h)
210	10350	32	End of all speed and passing limits
210	10560	27	Pedestrians
240	10800	29	Bicycles crossing
240	11040	24	Road narrows on the right
1320	12360	9	No passing
1650	14010	5	Speed limit (80km/h)
1860	15870	38	Keep right
1260	17130	8	Speed limit (120km/h)
1800	18930	10	No passing for vehicles over 3.5 metric tons
1080	20010	35	Ahead only
360	20370	34	Turn left ahead
1080	21450	18	General caution
360	21810	6	End of speed limit (80km/h)
1920	23730	13	Yield
1290	25020	7	Speed limit (100km/h)
390	25410	30	Beware of ice/snow
270	25680	39	Keep left
270	25950	21	Double curve
300	26250	20	Dangerous curve to the right
599	26849	33	Turn right ahead
480	27329	28	Children crossing
1890	29219	12	Priority road
690	29909	14	Stop
540	30449	15	No vehicles
990	31439	17	No entry
2010	33449	2	Speed limit (50km/h)
1350	34799	25	Road work

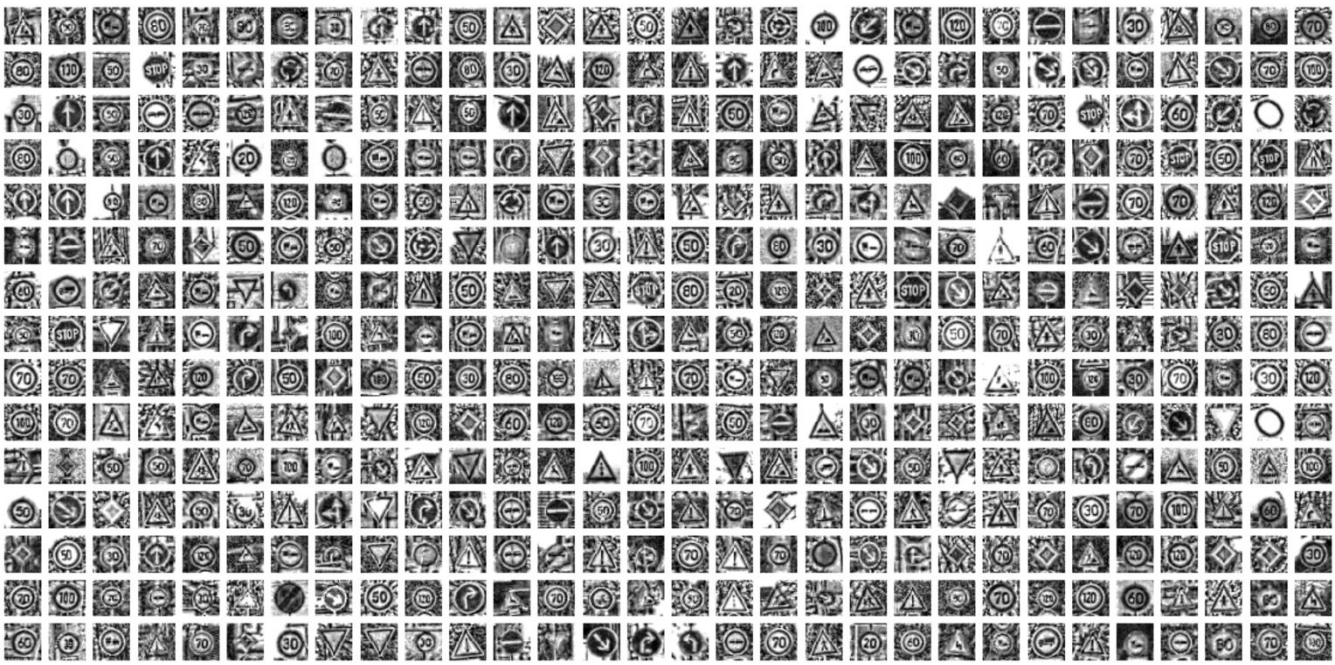
Datasets publicly available here –
http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset
https://d17h27t6h515a5.cloudfront.net/topher/2016/November/581faac4_traffic-signs-data/traffic-signs-data.zip

DESIGN AND TEST THE MODEL ARCHITECTURE

PREPROCESSING TECHNIQUES

- The images were first converted to grayscale. ***Sermanet/LeCunn's pape***r (http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf) talked about the lack of of increase in accuracy when color was used in classifying traffic signs. RGB color information was added information that would take longer to process by the neural network that would not increase accuracy so it is much better to just take this information out of the picture.
- A localized histogram equalization was applied because the images significantly differed in contrast and brightness.
- Values were scaled from [0, 255] to [0, 1]. We just did this to easily ballpark the intensity of the pixel. This step is actually not necessary at all. It just looks nice.
- This algorithm and code was derived from the preprocessing techniques by Navoshta in his article: http://navoshta.com/traffic-signs-classification/

450 RANDOM PREPROCESSED IMAGES FROM THE TRAINING SET



DETAILS ABOUT THE NEURAL NETWORK

I used a simple model of two convolution layers followed by two fully connected layers. The two convolution layers both have a ***relu*** followed by a ***maxpool***. The first fully-connected layer had a ***relu***, while the last layer did not. Below are more specifications of my neural network.

1st layer - Convolution	**2nd layer - Convolution**	**3rd layer - Fully Connected** (Wx + B)	**4th layer - Fully Connected** (Wx + B)
Filter_size = 5 x 5 Depth = 64 Stride = 1, padding = same Maxpool = 2x2 With Relu, No Dropout *Output Size = 32 x 32 x depth of 64. The 32 x 32 in the input size which is the same size of the processed image, since the stride is one and the padding is the same.*	Filter_size = 5 x 5 Depth = 32 Stride = 1, padding = same Maxpool = 2x2 With Relu, No Dropout *Output size = 16 x 16 x depth of 32. 16 x 16 is the reduced matrix when the the output of the last layer is max-pooled (2x2)*	(Flattened) With Relu With Dropout prob = 0.75 *Output_size = 256* *This output size was chosen arbitrarily and was tweaked to go higher and lower based on the performance of the model when trained*	Dropout prob = 0.75 (no relu) *Output_size = 43* This is the number of classifications/labels to choose from. There are 43 types of traffic signs in our data sets.

- The output of this network is in logits, so we feed this to a ***softmax*** function to get the probability. We then feed that probability to a ***cross entropy*** function to convert it to a ***one-hot encoding*** format.
- This architecture is a modified version of what was provided as an example in one of the lectures: *https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/convolutional_network.py*
- The parameters chosen here such as the **filter size, filter depth, output size,** are largely based on trial and error. I decided to use a 5x5 filter size because a 5x5 images seems like a good enough size to notice simple patterns like basic curves and shapes which are the basic building blocks to understanding the contents of an image.
- A **padding** was chosen to be the **'same'** to keep the computation of the output matrix shape simple.

- The **maxpool** is used to prevent overfitting and decrease the size of the output (faster computations). A 2X2 maxpool with a stride of 1 as recommended in the lectures.
- **Dropouts** are added in the fully-connected layers to help prevent overfitting, reduce the reliance to any particular weight.
- **Relu** functions are included to introduce nonlinearity to the system, and weed out the unnecessary negative values.

DETAILS ABOUT HOW THE MODEL WAS TRAINED

A **learning rate of 0.00005**, **epoch of 180**, and **batch size of 256** was chosen. The **adam optimizer** was chosen as recommended in the lectures. I started with a higher learning rate but validation accuracy was going back and forth so I lowered it. I started with the batch size of 512 but it was taking longer to produce the results of each epoch without actually improving the results so I rolled it down to half . I started with higher epoch but the validation accuracy was not improving up to a certain point which seems like a waste of computational power and might actually be overfitting the training data. Lowering the epoch has the risk of not being able to reach a good cross validation accuracy which means we are under fitting the data. The parameters are chosen largely from intuition and trial and error and also how the training was behaving based on the parameters. I stopped training when the validation accuracy was more than 95%, and the test accuracy was around 95% as well. To see how the model performed with the validation set, no dropouts are made, the dropouts are just randomly placed when training to prevent the model from relying too much on certain features, and reduce the likelihood of overfitting.

It is important to note that the data set was unbalanced but I think this is okay because that means you are likely to see the traffic signs that has a higher data size in real life as well, so it's okay that we are a little bit biased towards that. But it would be nice to compare a model trained with a balanced data set and see which performed better.

TESTING THE MODEL ON NEW IMAGES

I got 10 random german traffic sign images from the internet, and cropped them. I got 10 out of 10 signs predicted correctly. I think this is because I subconsciously chose the easiest ones to classify. All of them got at least 97% or more confidence except for the 2nd to the last image. Which is **Go straight or right.** This got a less than 71% confidence (and got 17% confidence that it was seeing a **yield** which was weird, because they really look different. It would be much understandable if it was mistaken to be an **ahead** sign. I think the reason for this is that the image was not cropped properly (It is leaning towards the right). Factors that might give difficulties to classifications include the brightness, contrast, of the image, the angle of the traffic sign, shadows, and physical obstruction.

To strengthen the predictions of this neural network, I think we should feed it more data. Some of the classes were represented far more than others. The lack of balance in a data is likely to result in a bias towards classes with more data points. We can generate more "fake: data points for less represented classes by applying small but random translational and rotational shifts as well as shearing and warping to existing images in the training set.

```
image_name = ['0-20speed','1-30speed', '12-priority-road','13-yield', '14-stop','17-no-entry', '18-general-caution',
              '3-60speed', '36-go-straight-right', '40-roundabout-mandatory']

own_set_y = np.array([0, 1, 12, 13, 14, 17, 18, 3, 36, 40])


 [ 0  1 12 13 14 17 18  3 36 40] <-predictions
 [ 0  1 12 13 14 17 18  3 36 40] <-actual
[[ 0  1 12 13 14 17 18  3 36 40]
 [ 1  0 41 12  5  9 26 28 13 37]
 [32  2 35 35  1 13  4 40 28 12]
 [35  5 15 15 40 35 11 35 35 11]
 [ 6  7 40  5 33 32 24 11 12  7]]
(top  5 predictions above) for each image

probability for top 5 predictions for each image:
0 [  9.81733441e-01   1.79047305e-02   1.56996073e-04   8.96898564e-05  8.69674623e-05]
1 [  9.99999881e-01   8.71776535e-08   3.14505613e-08   1.04039453e-08 3.02087932e-09]
2 [  1.00000000e+00   1.31352940e-09   1.00492659e-09   6.52618570e-10 4.45015802e-10]
3 [  9.99999881e-01   1.35349822e-07   5.87961971e-11   3.51375491e-11 3.08744592e-11]
4 [  9.99447763e-01   3.10564443e-04   1.51801534e-04   3.46596680e-05 1.87184160e-05]
5 [  1.00000000e+00   1.87969557e-10   6.72472078e-11   2.32838523e-11 5.54580036e-12]
6 [  1.00000000e+00   2.92547107e-08   2.58097432e-09   1.73857773e-09 1.08491138e-09]
7 [  9.96503234e-01   7.56855297e-04   5.36789594e-04   4.64317447e-04 4.07741842e-04]
8 [ 0.71430153  0.17930424  0.04936121  0.04407945  0.00309171]
9 [  1.00000000e+00   1.05749176e-09   8.63611738e-10   2.26828875e-10 1.02200789e-10]

Test Accuracy = 1.000
```

The test set consists of 12630 data points. At roughly 0.945 accuracy, this means that the model incorrectly predicted as much as 700 images. The validation set consists of 4410 data points at a 94.5% validation accuracy which we ended up with, the model incorrectly predicted a little less than 200 images on the validation set when it was being trained.

On the ten cropped images found on the web, since it has correctly predicted each one of them. This means that the with the new signs the prediction accuracy is 100% which is even better than cross validation accuracy of around 95.5% and a test accuracy of around 94.5%. It's a good sign that the model performs well on real-world data. However, calculating the accuracy on these 10 German traffic sign images found on the web might not give a comprehensive overview of how well the model is performing. The model does not seem to be overfitting, but I think we should test on a larger data set of new images and see if it passes the test - 5 or 10 data points don't seem to be a enough good set size for me to confidently say that that our model is generalizing well to real world data. I noticed that the images I have chosen don't have shadows or obstructing objects, and weren't particularly warped or distorted, and was centered in the middle quite nicely, and I specifically looked for german traffic signs. It's also reasonable to assume that if most real-world data were all as easily distinguishable as the 10 images chosen that the accuracy would be as high.

APPENDIX

RECOMMENDATIONS TO IMPROVE THIS DESIGN

- To strengthen the predictions of this convolutional neural network, I think we should feed it more data. Some of the classes were represented far more than others. The lack of balance in the training data set results in a bias towards classes with more data points. We can generate "fake" data points for less represented classes by applying small but random translational and rotational shifts as well as shearing and warping to existing images in the training set.
- Preprocessing the data can be made more faster by using better localized histogram equalization techniques and also no longer normalizing the values to be floats within the range of 0 to 1. Using integers between 0, 255 might be sufficient.
- Visualizing the network weights can also help in designing the architecture. Visualize them by plotting the weights filters of the convolutional layers as grayscale images
- Check the data points which are incorrectly predicted by the system and try to analyze this information
- Experiment with hyperparameters and other architectures.Try fiddling with the filter size of the convolutional layers as well as its output/output depth, you can also fiddle with the output size of the fully connected layers and the dropout probability.
- Use L2 regulation and early stopper techniques to prevent overfitting. Also play around with different types of optimizer function.
-

OTHER REFERENCES

http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf
https://github.com/jeremy-shannon/CarND-Traffic-Sign-Classifier-Project
https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/convolutional_network.py

https://github.com/vxy10/p2-TrafficSigns
https://github.com/paramaggarwal/CarND-Traffic-Sign-Classifier-Project/blob/master/Traffic_Sign_Classifier.ipynb
https://github.com/MehdiSv/TrafficSignsRecognition/blob/master/final_Traffic_Signs_Recognition.ipynb
https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project/blob/master/Traffic_Sign_Classifier.ipynb
https://github.com/navoshta/traffic-signs
https://github.com/hengck23-udacity/udacity-driverless-car-nd-p2

450 RANDOM IMAGES FROM THE TRAINING SET

10 RANDOM IMAGES PER CLASSIFICATIONS (43 CLASSIFICATIONS)

(See the png files from the Traffic_Sign_ClassifierMD folder output_16_1.png to output_16_43.png)