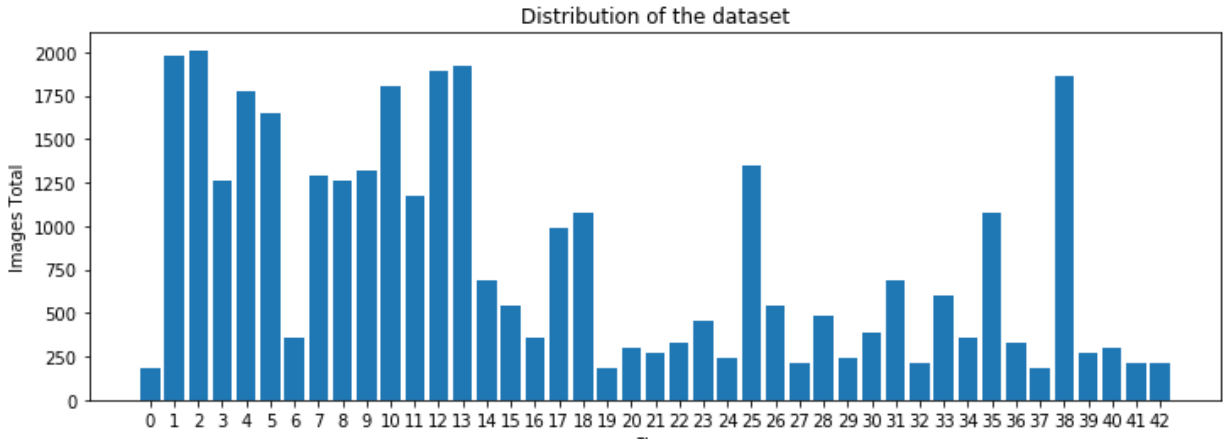
1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

I used the matplotlib.pyplot and numpy libraries to summarize and visualize dataset:

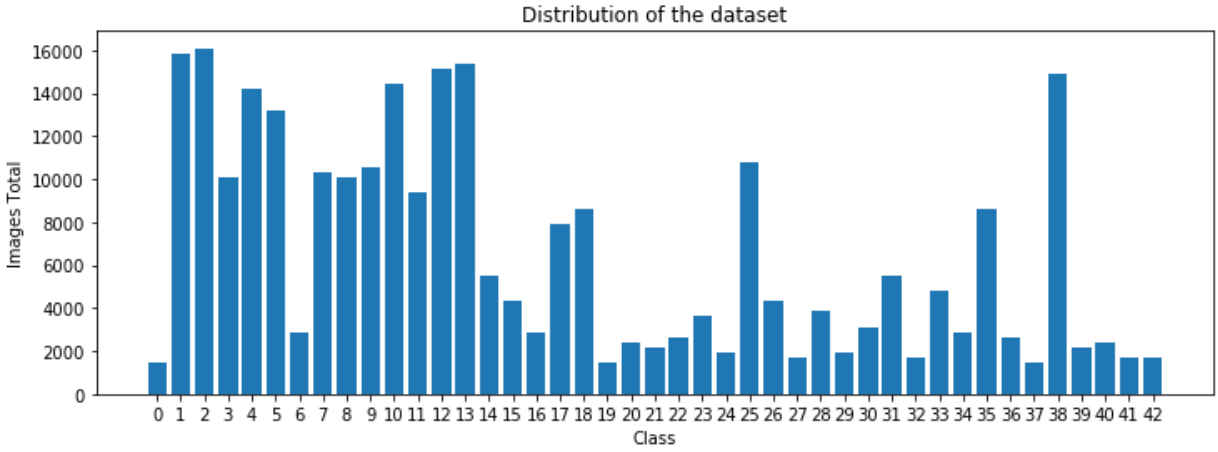
- The size of training set is – 35000 samples
- The size of the validation set is – 4410 samples
- The size of test set is - 12631
- The shape of a traffic sign image is – 32x32x3
- The number of unique classes/labels in the data set is – 43

2. Include an exploratory visualization of the dataset.

Visualization of original dataset distribution:



Distribution after adding generated examples:



Design and Test a Model Architecture

1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques?

Original idea was to increase value of edges, brighten up the image and generally enhance and brighten things up, may be grayscale using openCV and other libraries. However, experiments show that architecture of choice does not work with any preprocessing.

Any combination of mentioned techniques above made the model to converge on about 80%-90%. As the result they were dropped (I left commented out code in Jupiter Notebook of the project).

In the end, I used Keras preprocessing function to generate pictures with minor shifts in height and width, 5-degree rotation and slight zoom. These pictures were used to significantly increase the size of dataset, while leaving data distribution intact.

2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

I went with semi-supervised GAN to ~~play with it and justify money spent on deep learning nanodegree~~ enhance learning process of the network with generated data and help model to much better generalize to novel examples. With GAN we can train much deeper net with less data, due to the fact that generator can bring some useful noise that prevent overfitting (otherwise network just memorize training set).

However, we need to have enough data to train generator. That is the reason behind huge amount of generated images from Keras.

Network --> Discriminator – is implemented mostly according with original DCGAN paper but in order to make it work here I made it deeper

Layer	Description
Layer 1	
Input	32x32x3 RGB image
Dropout	0.2
Convolution 5x5	stride 2, same padding, outputs 32x32x64
Batch normalization	
Leaky RELU	
Layer 2	

Layer	Description
Convolution 5x5	stride 2, same padding, outputs 64x64x64
Batch normalization	
Leaky RELU	
Layer 3	
Convolution 5x5	stride 2, same padding, outputs 128x128x64
Batch normalization	
Leaky RELU	
Dropout	
Layer 4	
Convolution 5x5	stride 1, same padding, outputs 256x256x128
Batch normalization	
Leaky RELU	
Layer 5	
Convolution 5x5	stride 1, same padding, outputs 256x256x128
Batch normalization	
Leaky RELU	
Dropout	
Layer 5	
Convolution 5x5	stride 1, same padding, outputs 256x256x128
Batch normalization	
Leaky RELU	
Layer 6	
Convolution 5x5	stride 1, same padding, outputs 256x256x128
Batch normalization	
Leaky RELU	

Layer	Description
Dropout	
Layer 7	
Convolution 3x3	stride 1, valid padding, outputs 256x256x128
Leaky RELU	
Average Pool	
Layer 8	
Fully Connected	Outputs 43 classes for Softmax
Layer 9	
Softmax	Predictions

Generator:

Layer	Description
Input vector Z - 100	
Layer 1	
Fully Connected	outputs 4x4x3072
Batch normalization	
Leaky RELU	
Layer 2	
Convolution 5x5	stride 2, same padding, outputs 8x8x128
Batch normalization	
Leaky RELU	
Layer 3	
Convolution 5x5	stride 2, same padding, outputs 16x16x64
Batch normalization	
Leaky RELU	

Layer	Description
Layer 4	
Convolution 5x5	stride 1, same padding, outputs 16x16x32
Batch normalization	
Leaky RELU	
Dropout	
Layer 5	
Convolution 5x5	stride 1, same padding, outputs 16x16x32
Batch normalization	
Leaky RELU	
Layer 6	
Convolution 5x5	stride 1, same padding, outputs 16x16x32
Batch normalization	
Leaky RELU	
Dropout	
Layer 7	
Convolution 5x5	stride 2, same padding, outputs 32x32x3
Batch normalization	
Leaky RELU	
Layer 8	
Tan H	

3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

I used Adam optimizer, which pretty much standard practice here. Learning rate was pretty high 0.003, but it was shrinking by multiple of 0.99 each epoch.

Batch size 128. That was pretty much maximum for generator or it had issues generating something believable.

Number of epoch is large. It is GAN, that generates noise and with time can get even better results (with this dataset maximum I saw was 95.3% on validation and about the same on test, however I did not save that).

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93.

The whole point for me here was to evaluate how good GAN is for this type of tasks. Approach was to train on large amount of epoch while GAN does not allow model to converge after simply memorizing dataset. Allowing getting better generalization.

My final model results were:

1) Training – 99.36%
2) Testing – 92.25%
3) Validation – 93.46%

I had troubles with a generator to converge into something sensible to train the network. Generating mora data with Keras proved useful, but if I try to add generation of new images into the model during training so new epochs have different data sets did not prove useful. In fact, model tend to converge on about 80% during most of the efforts.

If a well-known architecture was chosen – ImageNet would be more than enough. It well tested on standard dataset and produce desired results with no issues.

Some additional questions were asked by Udacity's reviewer.

- What was the first architecture that was tried and why was it chosen?

No idea how that is unclear. Semi-supervised GAN to enhance training process

- What were some problems with the initial architecture?

Had to go a bit deeper with both generator and discriminator.

- How was the architecture adjusted and why was it adjusted?

Added 2 layers to generator to get more sensible images. As the result of deeper architecture dropout was added.

With discriminator, just 1 layer was added to balance out generator.

- Which parameters were tuned? How were they adjusted and why?

Learning rate, additional generated images size, batch size. It is important to get network a good start so generator converge early.

- What are some of the important design choices and why were they chosen?

All-important choices are described in original DC GAN paper, referenced above.

Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

I took 5 pictures of roadwork signs. They are in triangle, pretty popular shape in dataset. And with all twists lets see how can model predict new pictures.

One sign is base line, another have yellow color and turned, other is very dark, other I blurred a lot and last one have other signs in the picture.

Model was wrong only once when I deliberately smudged the middle. All sharp photos I gave were identified correctly.

```
Prediction is -  CORRECT
  ID_S   NAME OF THE SIGN
   25          Road work
   22         Bumpy road
   26    Traffic signals
   29  Bicycles crossing
   18    General caution
```

```
Prediction is -  CORRECT
  ID_S                        NAME OF THE SIGN
```

```
   25                            Road work
   30                     Beware of ice/snow
   31                   Wild animals crossing
   11  Right-of-way at the next intersection
   26                         Traffic signals
```

```
Prediction is -  CORRECT
  ID_S                         NAME OF THE SIGN
   25                                Road work
   11  Right-of-way at the next intersection
    6            End of speed limit (80km/h)
   26                         Traffic signals
   18                         General caution
```

```
Prediction is -  WRONG
  ID_S                         NAME OF THE SIGN
   11  Right-of-way at the next intersection
   25                                Road work
    6            End of speed limit (80km/h)
   26                         Traffic signals
   28                       Children crossing
```

```
Prediction is -  CORRECT
  ID_S                         NAME OF THE SIGN
   25                                Road work
   11  Right-of-way at the next intersection
   18                         General caution
   26                         Traffic signals
   24              Road narrows on the right
```

Udacity reviewer asked me to add here that final accuracy is 60%, since discriminator got one wrong and that implies overfitting of the data, since training testing and validation accuracies are much higher.

That is completely ridiculous thing to say since we compare dataset of 35000 examples and 5 and using percent correct to compare them. That is totally wrong thing to say, but I guess I have to mention it.