Self-Driving Car Engineer Nanodegree Program

Table of Contents

Section 1, Dataset Exploration:

Data Set Summary:

The Data Set is a version of the widely available German Road Traffic Sign data set. Freely available for academic work and subject to much use in this field of study.

The Data Set consists of more than 50,000 images. Each image is 32 by 32 pixels in size with 3 channels for Red, Green and Blue colour information. It is divided into 3 sections the largest for Training the model, the smallest for Validation and the remaining for Testing at the final stage. A Pickle file format is used to stored the Data Set in a file.

Output from Cell 1 below shows the breakdown

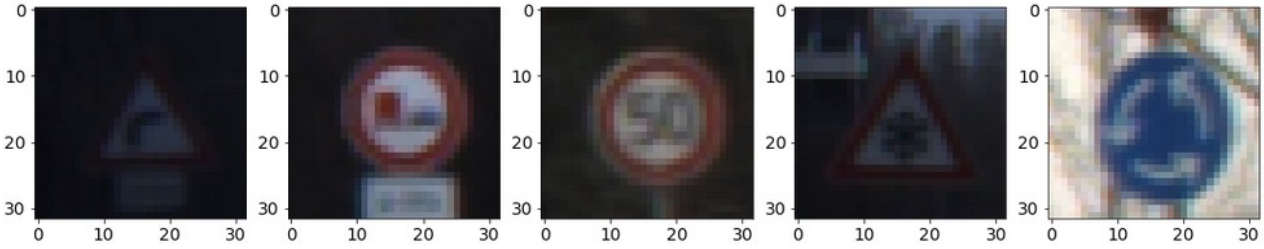
```
Image Shape: (32, 32, 3)

Training Set:   34799 samples
Validation Set: 4410 samples
Test Set:       12630 samples
```

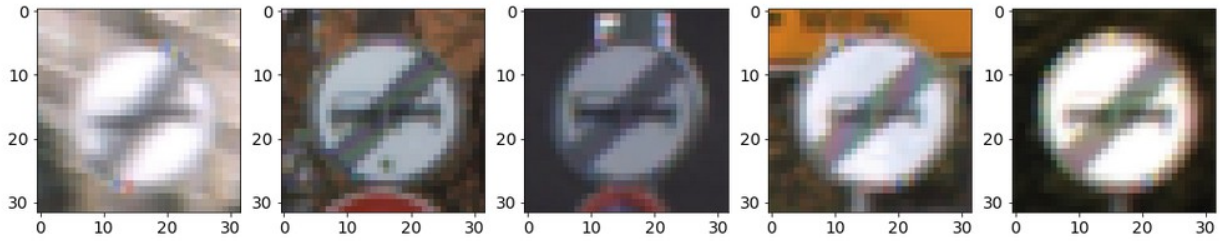
Data Labels are listed in a tuple in Cell 2. This was to allow for easy checking of the exact labels from a ready to read list as opposed to a stored file. It was necessary to refer to this on a many occasions to check results and find errors.

Exploratory Visualization:

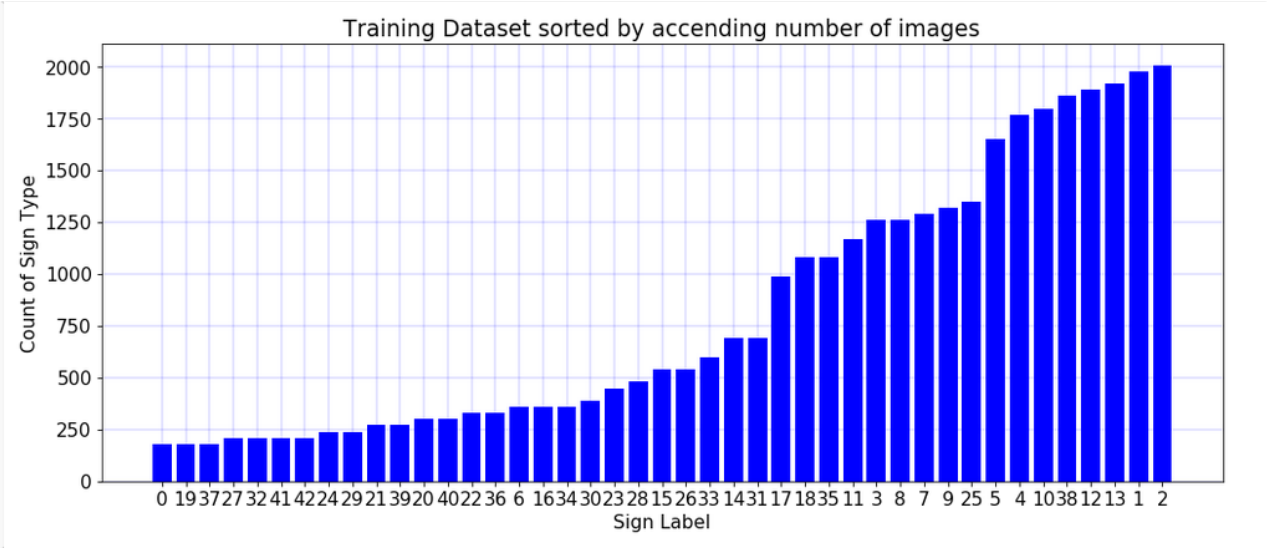
To give a more detailed and balanced representation of the Data Set view sample were taken and displayed in Cell 3. The first selection is a random selection of 5 different signs to give a representation of the whole Data Set and to illustrate the variation across the dataset.



The next figure is 5 samples of the same type. This is possible because the Data Set is not shuffled yet and similar signs are grouped together. I selected 5 examples of one sign to show the variation across one sign type. Note considerable variation in brightness, contracts and extraneous information across 5 samples of the same type.



It was noted that there seemed to be enormous variation within the data set but also the quantity of each Sign type may also vary and this may prove significant. A bar chart was created and sorted in ascending order to illustrate the total number of each sign, generated in Cell 4 and shown below.

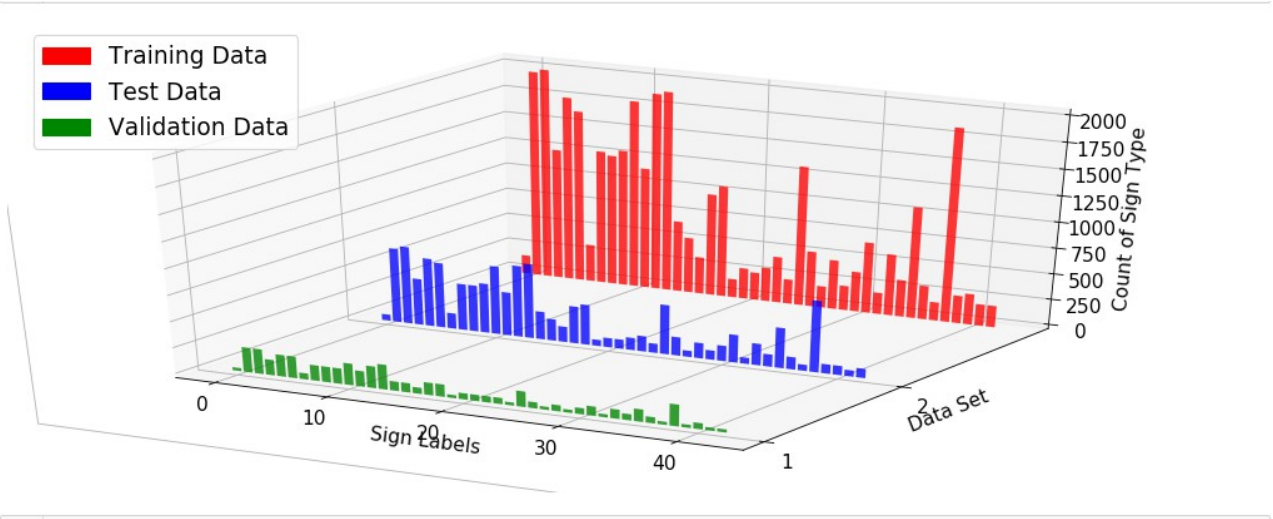


Comments on Diagram:

It is shown that approximately one quarter of the signs types with Data Set appear less than 200 times, approximately one quarter appear between 200 and 500 times, approximately one quarter between 500 and 1200 and approximately one quarter between 1200 and 2000 appearances.

Questions arise as to will there be a correlation between the accuracy of the models ability to predict signs and the frequency those signs appear in the Data Set. Can the Dataset be expanded by replication or copying data and can augmentation of data be useful? What if every sample appeared 2000 times? These questions are answered later in the report.

The next question to consider was did the variation in the Data Set extend into the Validation and Test sets. Cell 5 generates a 3D chart to put specially made bar charts together for comparison. Note the standard *Mathplotlib* library does not facilitate this. The 3D plot as shown is for bar charts not histograms. Using Histograms on this chart type will create the histograms emerging from the side or horizontally and the representation does not work so a work around was created. 3 histograms had to be made and then converted into bar charts and then plotted to represent the data as shown.



Comments on Diagram:

The diagram clearly shows that the variation in the Training Set is similar to the Test and Validation variations. This is shown by similar peaks in the data for corresponding labels. The relative size of the data sets are also clear.

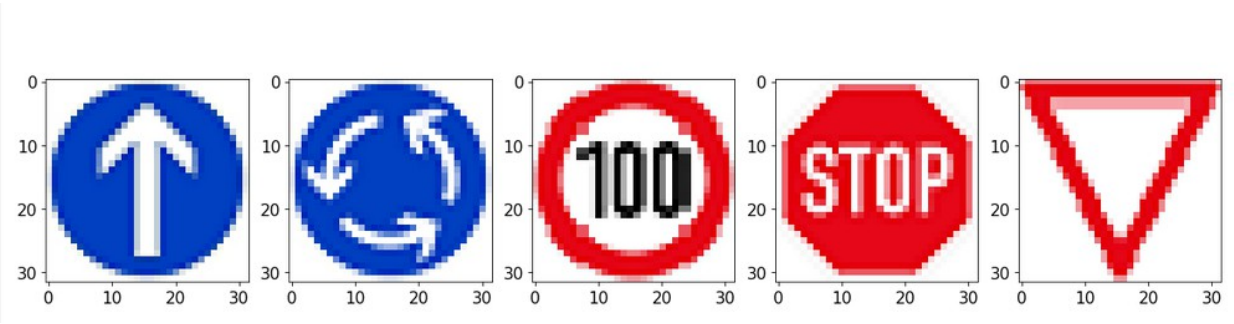
Extra Signs:

A second data set was also created. This was a sample of German Road Traffic Signs found on the internet. These images are used later in the project but loaded towards the start to facilitate preprocessing and to have all the loading and preprocessing together for clarity.

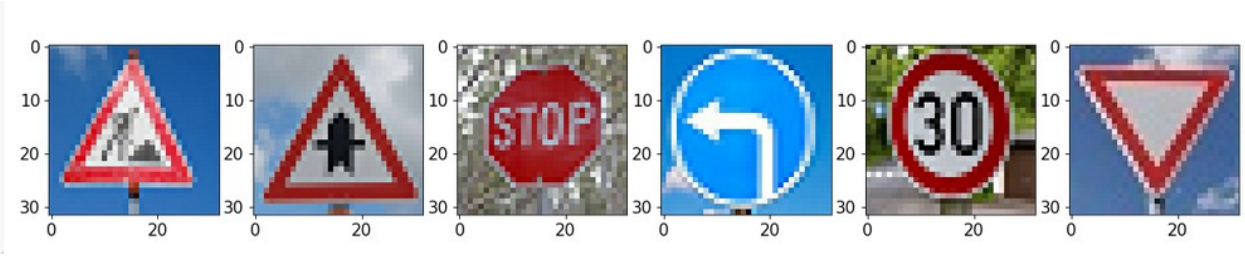
The signs are loaded in individually in Cell 6. This was to allow names and labels to be checked as there are many similar data labels and it turned out some were in fact incorrectly assigned and affecting result. " Priority Road "and "Right of way at next junction" were confused at one point

Two types of extra signs were chosen the first were diagrams of signs taken from Wikipedia. This choice was to see if the model could correctly classify diagrams when being trained on pictures. Would the simplicity of the diagrams help or hinder classification ? Would "real" be recognised but diagram equivalents not? The other signs were normal pictures of German road signs resized for the project.

The five diagrams signs from Wikipedia are shown below:



The 6 signs actual selected from the internet are shown below:



Section 2, Design and Test a Model Architecture:

Preprocessing:

Most of the work in this project focused upon prepossessing because there was so much variation in the dataset it seemed likely this might be an issue and also to address the issue of over fitting augmentation and modification seemed likely to be necessary.

Without any adjustments or preprocessing the LeNet architecture can achieve appropriately 89% accuracy. A variety of preprocessing techniques were used some upon suggestion or upon precedent established in papers and some out of availability of pre-existing frameworks eg, OpenCV.

Gray Scale:

The Data Set was in colour and one relatively straight forward process was to convert to grayscale and this achieve an improvement of approximate 2%. This is done in Cell 7 in the function *RGBToGrayScale.*

Zero Center and Zero Centre and Normalise:

Zero centring means to adjust the data to vary around the zero point. This was explained in the online classes as better for the model. Normalising is the process of making the data vary within a limited range such as implemented by this formula `(pixel - 128)/ 128` . Zero Centring and Normalising were implemented separately and then together to see if it made a difference to processing. Zero Centering on its own produced results around 5% and zerocentering and norm,ailisng produced results peaking at about 92% . Result of better than 93% were obtained without these steps but including grayscale conversion. See Cell 7 for the functions.

Also in Cell 7 preprocessed data was saved and if already in existence loaded to skip the time consuming preprocessing if possible.

Also in Cell 7 are test displays and progress outputs to monitor progress.

Cell 8 has two special functions *DownSelect* and *capAt* . These functions were used to select groups of sign types from the data and also to experiment with capping the number of individual signs at arbitrary levels to see what would happen. Using downSelecting it was possible to break up the training data and validation data into quarters as previously described in Cell 4. The lowest occurring quarter achieved 80% accuracy the next quarter 84% accuracy the next quarter 96% accuracy and the highest quarter 98% accuracy. This firmly established the importance of quantity of sample in the training the model. Also it established some targets, could the lowest occurring sign types achieve 98% with augmentation? The answer proved to be no but better than 94% was achieved.

Cell 9 is where downselect was used with the commented out code showing how the various labels were selected from the overall data.

Cell 10 is where the augmentation begins. What happens here is that the data is send into a function called *increaseByCopy* and it takes the current count of a given sign from a generated histogram of the data then calculates how many more copies are need to bring the total up to a given target eg 2000, in fact all sign types were copied until there was 2000 examples of each type.

Cell 11 is where the data that had been increased by simply copying until each sign type numbered 2000, now the new copied images are modified to change them significantly enough so that the model does not over train be seeming the same thing repeatedly but at the same time the altered images have to resemble the originals to some extent. Various techniques were used such as randomly picking which technique to use them applying rotation, sharpen or affine transformation to alter the image. This improved the accuracy to greater than 94% Of note rotation seemed to be too much at 90 degrees but result with only + or – ten degrees were actually about 2% worse so significant changes to the images was actually important.

The next 2 cells 12 and 13 called and used the above functions and displayed some test results for experimentation.

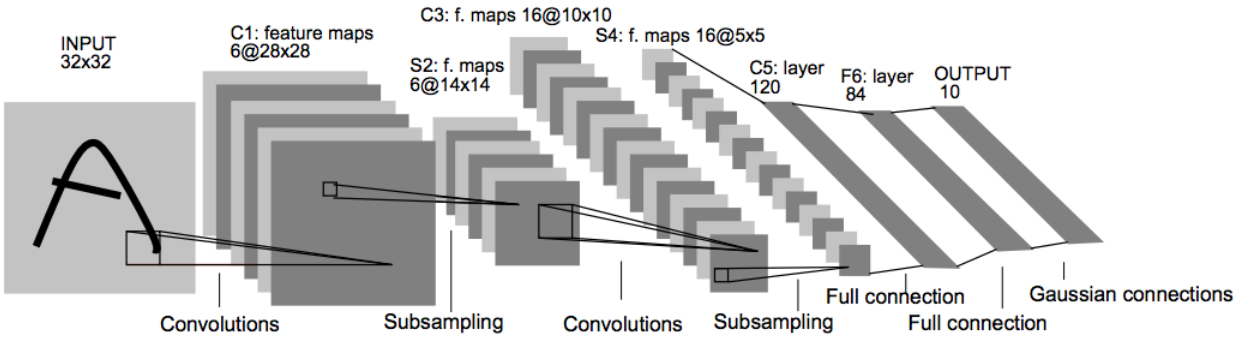
Further detail about augmenting and modifying the data in Section 5.

Section 3, Model Architecture:

The model architecture used was the LeNet classifier. As it turned out it was possible to achieve the target of better than 93% using preprocessing and augmentation. The only addition made to LeNEt was a break out statement to cut off the training process at an arbitrary point eg > 94% this ensured the model would be saved at the highest value that had been seen in practice runs. Generally the last epoch may not have been the highest accuracy so this was added.

Drop out and inception could have been added but time did not allow and the target accuracy had already been achieved. Adding to LeNet will become a portfolio project.

The LeNet architecture is shown in the illustration below, source Yan LeCun



The Model of LeNet-5 is describe below, source LeNet-Lab Udacity

Input

The LeNet architecture accepts a 32x32xC image as input, where C is the number of color channels. Since MNIST images are grayscale, C is 1 in this case.

Architecture

Layer 1: Convolutional. The output shape should be 28x28x6.

Activation. Your choice of activation function.

Pooling. The output shape should be 14x14x6.

Layer 2: Convolutional. The output shape should be 10x10x16.

Activation. Your choice of activation function.

Pooling. The output shape should be 5x5x16.

Flatten. Flatten the output shape of the final pooling layer such that it's 1D instead of 3D. The easiest way to do is by using `tf.contrib.layers.flatten`, which is already imported for you.

Layer 3: Fully Connected. This should have 120 outputs.

Activation. Your choice of activation function.

Layer 4: Fully Connected. This should have 84 outputs.

Activation. Your choice of activation function.

Layer 5: Fully Connected (Logits). This should have 10 outputs.

Model Training:

There were a number of parameters to adjust. Epoch, Batch size, learning rate and also adjusting the size of the sample as a basis of comparison.

Epochs:

Epoch levels used were 10 , 50 , 100 and 200. Generally there was a noticeable difference between 10 and 50 Epochs and usually the best figures were obtained between the 50 Epoch and 100. Increasing beyond 100 usually gave a small improvement of maybe 1% and 200 Epochs often did not improve the figures.

Interesting observations around Epochs is that the accuracy of testing the model against the Extra Signs often was better around 50 to 100 and higher levels like 200 often did worse. This is likely because of over training not giving good answers on the “diagram” signs. Also with low Epochs of 10 or 50 the probabilities of choice were less than 1.0 and the second probability was useful after 100 epoch the probabilities were often 1.0 even if the guess was incorrect. This suggests higher epoch counts can be counter productive.

Batch Sizes:

Varying this number did not seem to make a significant difference.

Learning Rate:

The default rate of 0.001 was tested against 0.0005 and 0.0001 . There was no consistent improvement but it could be seen that the model was improving at a slower pace but did not get to a noticeable better place. Improvements of less than 0.5% can be hard to consistently reproduce and could be put down to chance so the default level was kept as it seemed faster and equally good.

Different data sets:

For comparison the data set was split into 4 quarters in ascending order of number of signs. Theses training subsets were tested against similarly reduced validation subsets. In testing the lowest frequently occurring group achieved 80%, the second lowest 84% the second highest 96% and the highest 98%. This also demonstrated that the LeNet architecture was capable of very high accuracy with good data inputted.

Solution Approach:

The solution was arrived at by starting with the LeNEt architecture and gradually improving the whole process. The sequence of improving is very close to the sequence that the code is presented in the Notebook. That is to say the code at the very top of the page was done first and almost in sequence until the end. After each idea was tried it was tested. No further ideas were tested because the target had been reached.

Preprocessing got a lot of effort because the effort was rewarded. Starting with Colour to GrayScale was recommended and relatively straight forward. Normalising and Zero Centring proved counter productive or neutral..

Augmenting the data was also part of preprocessing and discussed in its own section. Augmenting and modifying the augmented data was the key to getting the results from around 92-93% to 94-95%.

Section 4, Test a Model on New Images:

Acquiring New Images:

A number of images were acquired for different source, the first was images from the internet, the second was images specifically from Wikipedia that were in fact diagrams of German road sign.

Performance on New images:

It was observed that a perfect score could be obtained after 200 epochs in the current configuration however this did not happen every time training was conducted that is to say it might get 100% but if the model was cleared and training done again it might not get 100% so there seems to be an element of randomness to the process. Also sometimes lower epochs produced better results and similarly a value of 93% after 100 epochs seemed to be sometime better than 93% after 200 Epochs. This suggests that the training accuracy value expressed in % is not telling the whole story. In particular the Diagram images and sign types that were less frequent caused difficulty until the count of each sign was increased and the signs were modified that gave the best results.

The picture below shows the final result at 200 Epochs. Frustratingly Epoch 199 was 94% and Epoch 200 was 93%. The same run of the program gave 95.4% at 134 see picture second below.

```
EPOCH 199 ...
Validation Accuracy = 0.940

EPOCH 200 ...
Validation Accuracy = 0.933

Model saved
```

```
21]:
1  with tf.Session() as sess:
2      saver.restore(sess, tf.train.latest_checkpoint('.'))
3
4      test_accuracy = evaluate(X_test, y_test)
5      print("Test Accuracy = {:.3f}".format(test_accuracy))
```

```
Test Accuracy = 0.919
```

There was a piece of code added but not used in this case, to cut off the Epochs at a predetermined level using that method 94% could be achieved in around 100 Epochs usually the peak was around 150 Epochs and the final result is a fair representation of most of the systems typical performance. Keeping a running track of the best Epoch by saving the highest so far could be added to improve performance taking the best out of 200 should consistently give 95% or better.

Predict the Sign Type for Each Image

```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))

    test_accuracy = evaluate(extraSigns, extraSignsLabels)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
```

```
Test Accuracy = 0.818
```

The result on the new images was disappoint it was 81% but on occasion it has got 100%. Generally it was noted that with very low Epoch counts around 20 performance would be bad , around 100 it was best and around 200 performance was noticeable below peak. This is an example of over training. Checking the performance against the new data was kept in a separate Cell 22 to allow it to be run independently of testing the model on the test data set which could only be done once.

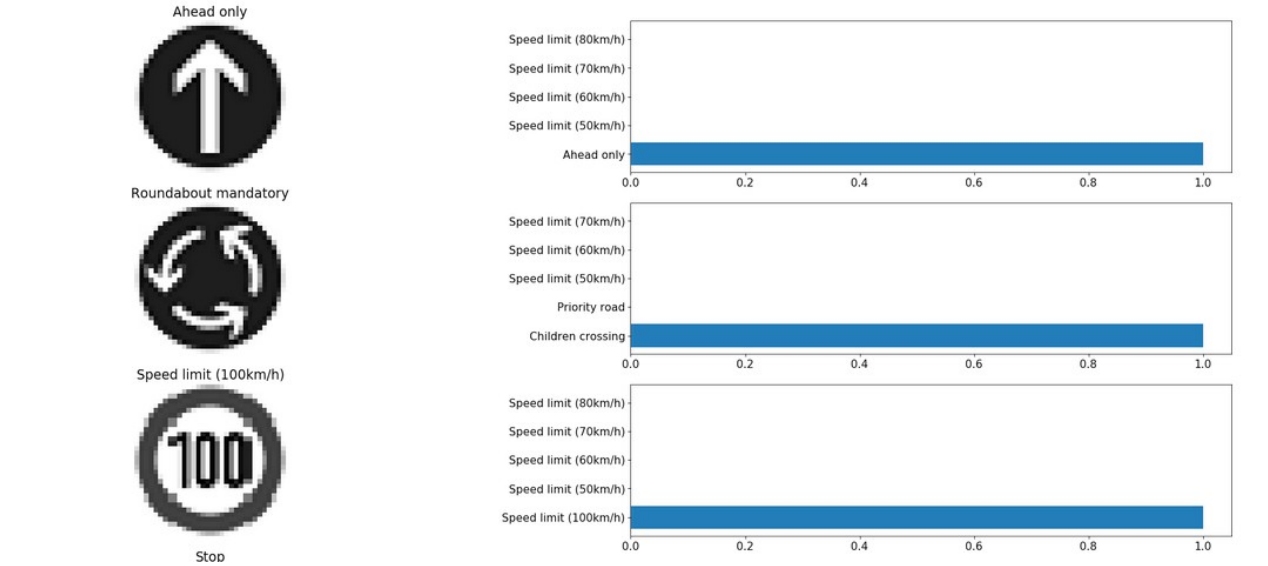
Model Certainty- Softmax Probabilities:

Visualisation of Softmax Probabilities

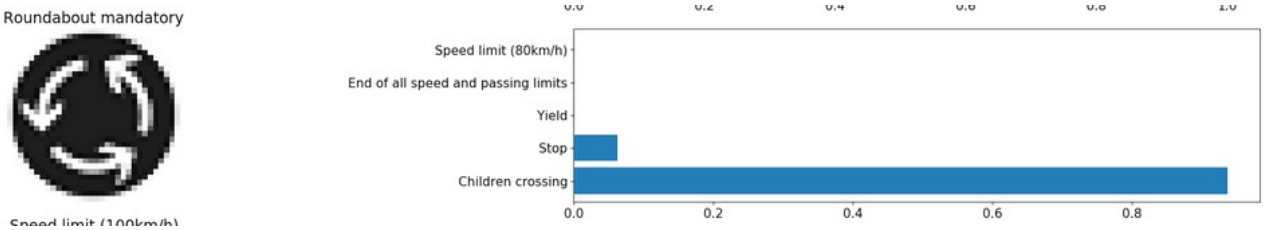
The Softmax probabilities were generated in Cell 26. Sample output shown below. Note worthy here is a characteristic of the Softmax function. This was discovered while trying to understand the second or third guess by the model. Softmax function will return the top K (5) values but if the first value is 1.0 then the next 4 values are actually the next 4 values in sequence from the location of the first value. In other word softmax with just return the next 4 values it finds. This explains why the second or third guess at 0% probability are really nothing to do with the sign. This effect can be seen for all the signs and is also true if softmax reaches 1.0 probability with more than once guess. ie if the first guess is 70% and the second guess is 30% the next 3 results are just the 3 results that were in sequence in the array from the second result. This can be checked by looking at the Labels data and you can see if it picked label number 10 then next one would be the label right next to it etc.

In [26]:
```
plt.figure(figsize=(30, 50)) #16,21

for i in range(11):
    plt.subplot(12, 2, 2*i+1)
    plt.imshow(extraSigns[i].squeeze(), cmap="gray")
    #plt.title(i)
    plt.title(labels[int(extraSignsLabels[i])+2])
    plt.axis('off')
    plt.subplot(12, 2, 2*i+2)
    plt.barh(np.arange(1, 6, 1), signs_top_5.values[i, :])
    #labs=[signs_class[j] for j in signs_top_5.indices[i]]
    labs=[labels[j+2] for j in signs_top_5.indices[i]]
    plt.yticks(np.arange(1, 6, 1), labs)
plt.show()
```



Another noteworthy point is that to see second and third most probable choices a lower epoch was needed. To see multiple choices with various degrees of probability. The below picture was generated with epoch count of 20. Also note that both guess are incorrect at this epoch of 20.



Section 5, Stand Out section:

Augment the training Data:

At the beginning of the project as soon as the DataSet was represented ascending order bar chart it was obvious there were large differences in the numbers of some sign types in the data with values ranging from around 200 to near 2000 with half the data set over 1000. The obvious question that arises from this fact was what if any effect did this variation have on performance.. That is to say was it true to say the signs that appeared less frequently in the dataset were less likely to be correctly identified?

In order to address this question the Data Set was sub sampled. Using the ascending ordered bar chart as a guide 4 subsets of 10 sign types corresponding to the 10 least frequently occurring sign types , the 10 next least frequently, the next 10 least frequently and finally the top ten most frequently occurring. 3 Sign types were left out to create 10 equal sized subsets for the sake of consistency.

The code in Cell 9 is where the subsets are created and you can see the commented out code where the subsets were selected. This code can be used to recreate the subsets if needed. This code was commented out to prevent the running of the overall model becoming to long as the running the model on 4 subsets and the original set and the extra signs is a lot of data and computational effort. Validation Data and test data was also sampled into subsets.

Running training and validation on the subsets showed the accuracy of 80%, 84%, 96% and 98% could be achieved for the lowest to highest occurring subsets respectively. This firmly established the total number of signs in the datasets as being a major factor in the accuracy achieved in validation.

With the frequency of occurrence of each sign established as being important. The next task was to increase the occurrence of each sign within the data set. With some experimentation it was decided to make every sign have the same frequency of occurrence of 2000 as this was the maximum number in the dataset.

A function to increase the dataset is in Cell 10. This cell takes the occurrence of each sign type and subtracts it from a target in this case 2000. It then goes through the dataset (before shuffling has occurred) and copies each group of signs into a new array repeatedly until the total number is 2000 of each type. The current number is divided into 2000 and a loop iterates that many times with a remainder value (mod, remainder) number used to select the remainder giving an exact 2000 for each sign.

The Data Set was thus augmented so all signs had 2000 examples but much of the data , approximately 50% was now not original data but simply copied data.

This new data set achieved approximately 94 % after 100 Epochs which was some way short of the 98% achieved by the top 10 occurring signs on their own but an improvement on the result without augmentation of about 1-2%

Concluding that copying the data on it own was not as good as 2000 original samples per sign. The dataset was modified as well as increased.

Cell 11 is where random augmentation and modification occurs. This Cell takes each sign type, skips the original data and modify the new data. As the data is still not shuffled it was possible to step through the data in 2000 intervals and use the original data counts to skip the original data easily. Each sign (image) is taken individually and randomly modified to be Rotated, Sharpened or Translated, with a random factor within the modifying functions also. This creates enough variation in the new data to improve the performance and help reduce over-training.

Using this new data set that has been both augments and then modified performance was improved from around 93% to a peak of over 95%

Noteworthy is that in the rotation transformation originally used a possible rotation range of plus or minus 5 degrees but using plus or minus 90 degree worked better which as a surprise.

Analyse new Image Performance in more detail:

The performance of the classifier on the new (extra) images was interesting for a few reasons. Firstly the images that were in fact diagrams had similar success rate to those that were real images and this has the implication that diagrams of signs may be a useful way of teaching classifiers about signs when no Data Sets are available. The idea of training the model only on diagrams and especially diagrams that are modified will make an interesting portfolio project for the near future.

The success rate on the new or extra signs actually was better on epoch around 100 than around 200. Over training is likely the reason. In fact 100% was observed with epochs of 100.

Also noteworthy is the fact that the system is often 100% certain about a sign but in fact not correct. Usually the sign is a reasonable misclassification such as 80 kmph being misclassified as 60kmph but not always. Also as mentioned previously when the probability is 100% the second guess is actually just the next sign in the array and not of any significance in terms of probability. This can create results where the fist one is wrong and the rest are irrelevant.

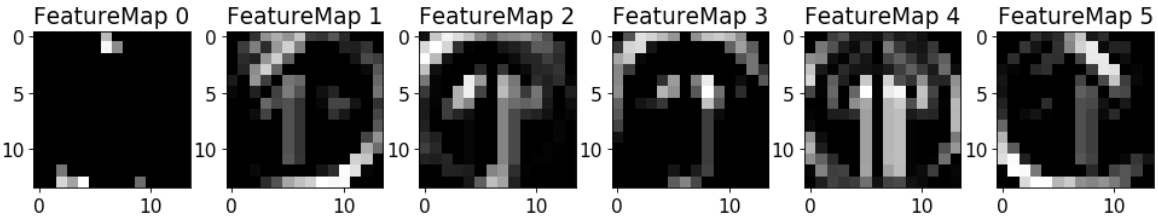
Visualise layer of neutral network:

Below is a picture of the final model after 200 epochs

```
28]:
with tf.Session() as sess:

    saver.restore(sess, tf.train.latest_checkpoint('.'))
    print("conv1 : First layer")
    outputFeatureMap(extraSigns, conv1)
```

```
conv1 : First layer
```



Below is a picture of the final model after 20 epochs, the similarity of FeaturesMap 4 and 5 for each Epoch count is remarkable and supports the idea that much of the learning happens in a relatively low epoch count.

```
conv1 : First layer
```