# Traffic Sign Recognition

## 1. Summary

We proposed a convolution network for German traffic sign classification. It first uses “illumination network” to automatically learn a illumination correction method for preprocessing the input data. By using modified “dense-block”, which are concatenation of convolutions, we are able to “shortcut” activation from bottom to top. Our single network achieve 99.68% accuracy with just 27.0 millions MAC (multiply accumulation).

In order to prevent overfitting of data during training, we use dynamic data argumentation to create new data at the training epoch. This is achieved by randomly applying geometric and illumination transform to the original data.

*The code to the project can be found at:*
*[https://github.com/hengck23-udacity/udacity-driverless-car-nd-p2](https://github.com/hengck23-udacity/udacity-driverless-car-nd-p2)*

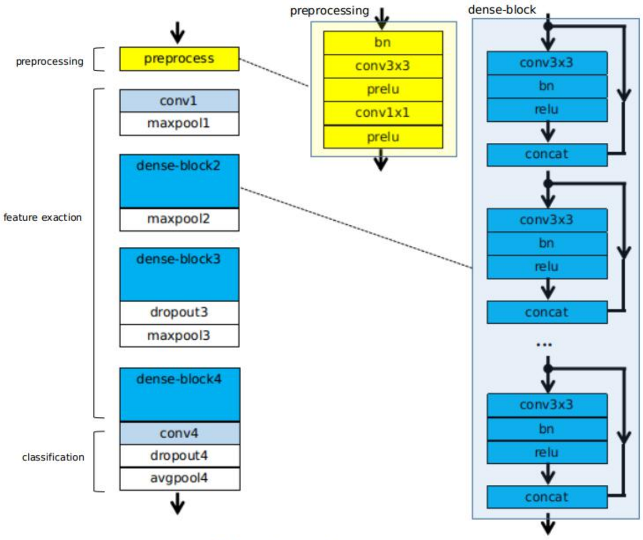



*Figure.6: Our network*

## 2. Data Set Summary & Exploration

**[Code]**

```
In [5]: train_images, train_labels = X_train, y_train

        #count
        #h = np.histogram(train_labels, bins=np.arange(num_class))

        #results image
        num_sample=10
        results_image = 255.*np.ones(shape=(num_class*height,(num_sample+2+22)*width, channel),dtype=np.float32)
```

**[Description]**

The data for the project consist of 32x32x3 rgb cropped image samples of various traffic sign. There are 43 target classes. There are 39,209 samples for training and 12,630 samples, see table.1.

| | | |
|---|---|---|
| **num of samples** | **train** | 39209 |
| | **test** | 12630 |
| **number of class** | | 43 |

*Table.1: Project data*

Figure.1 shows the train data characteristic. The first column is the label image. The second column is the mean image, followed by some example samples of each class. Next is the class label and class name. Finally, we give the sample counts for each class and the class distribution histogram.

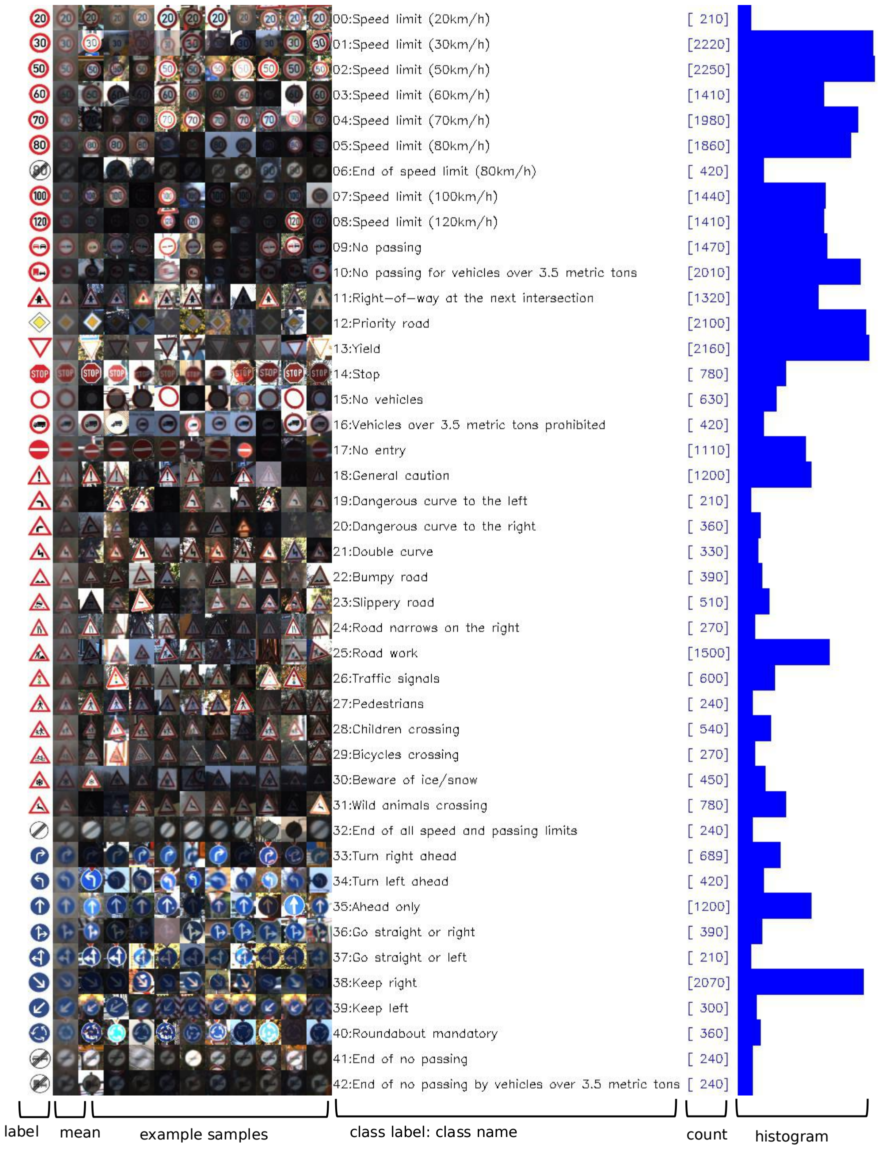



*Figure.1: Train samples characteristic*

We made the following observations:

1. The class distribution is imbalance, with the smallest class having about 200 samples, and the largest with about 2000 samples.

2. Within each class, samples exhibits variations. Brightness and contrast variations seems to be the most prominent. There are also rotation, scale and other geometric transform. Finally there are some blur and minor occlusions in few cases.

3. The mean images for each class represent the class cluster center. They are rather clear and well defined, meaning that the variations are “linear” in the image space. We hence expect the classification problem is “not too difficult”.

Next, Figure.2 shows the test data characteristic. Generally the test data has the same characteristic as the train. Their class distribution is the same. The class examples exhibit the same variations. Most importantly, their mean images are similar.

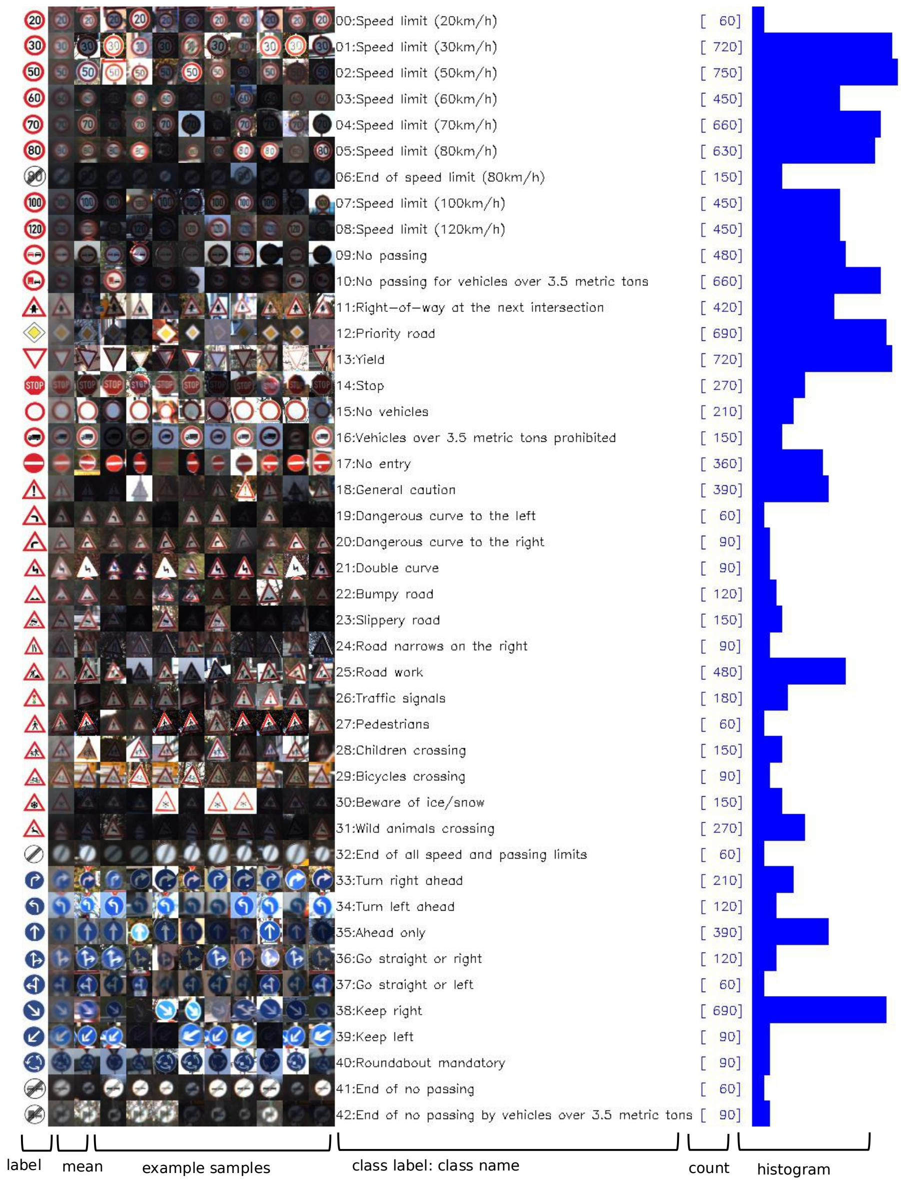



*Figure.2: Test samples characteristic*

# 3. Design and Test a Model Architecture

### 3.1. Setting up of testing, training and validation set

**[Code]**

```
In [10]: #prepare all data here
         classnames, X_train, y_train, X_test, y_test = load_data()

         train_images, train_labels,  valid_images, valid_labels = split_data(X_train, y_train)
         test_images, test_labels = X_test, y_test

         num_train = len(train_images)
         num_valid = len(valid_images)
         num_test  = len(test_images)
```

**[Description]**

Table.2 shows the setup of the data for testing , validating and training our convolution network. For test data, we use all the 12,630 test samples. For validation, we randomly select 3,000 samples from the original train samples.

| | | |
|---|---|---|
| **test set** | | 12630 |
| **validation set** | | 3000 |
| **train set** | **argumented samples per clas** | 20000 |
| | **number of class** | 43 |
| | **total** | 860000 |

*Table.2: Test, Train and validation data*

The remaining 39,209-3,000=36,209 train samples are used to create final augmented set of 86,0000 samples. By using data argumentation, we hope to solve the problems of insufficient train data and class imbalance. The steps of data argumentation are:

1. Flip the 36,209 train samples to create more train samples, see figure.3. This extends the train set to 62,187. We use the flipping method created by [*http://navoshta.com/traffic-signs-classification/*](http://navoshta.com/traffic-signs-classification/)

2. Resample the extended set, such that there are 20,000 samples per class. This creates a class-balanced set of 43x20,000=860,000 samples.

3. Randomly select 80% from the balanced set. These selected samples are perturbed by random geometric transform of rotation, scale, translation and perspective distortion. They are also perturbed by random illumination transform of brightness, contrast and saturation. Figure.4 shows the results of perturbed samples.

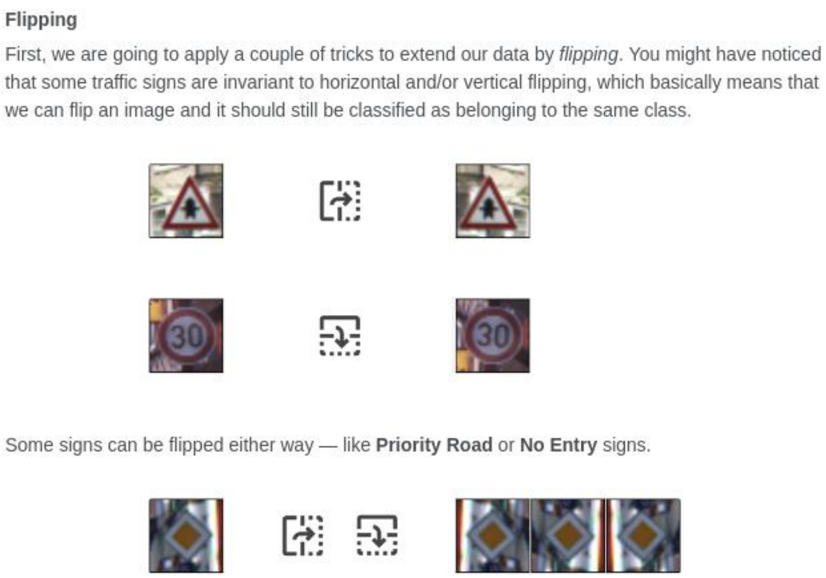



*Figure.4: Example of flipping (adopted from [http://navoshta.com/traffic-signs-classification](http://navoshta.com/traffic-signs-classification))*

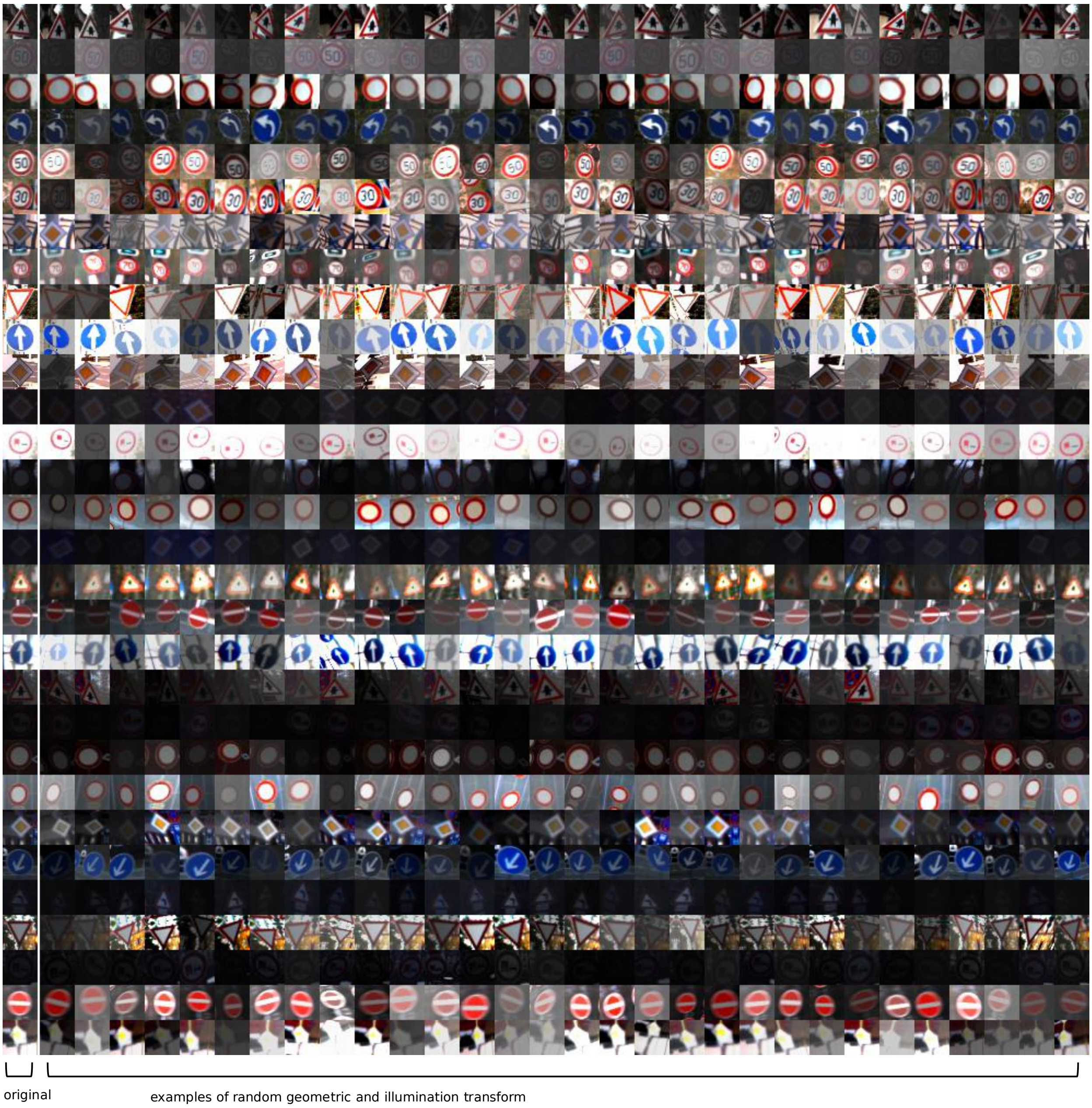



*Figure.5: Examples of random geometric and illumination transform*

**3.2. Setting up of data preprocessing**

**[Code]** **-** none-

**[Description]**

Popular data preprocessing methods include color transform, whitening and histogram equalization. We do not use such external “hand crafted” data preprocessing. Inspired by paper[1], our network includes data processing layers to act as "illumination transformation network". Just as “spatial transformer[2]” is being used to automatically correct affine transformation, our "illumination transformer" is used to correct illumination in the input. We will describe the implementation in the next Section3.3 and some of its results in Section3.6.

### 3.3. Setting up of network

**[Code]**

In [14]:
```python
# my densenet here!
#the inference part (without loss)

def DenseNet_3( input_shape=(1,1,1), output_shape = (1)):

    H, W, C   = input_shape
    num_class = output_shape
    input     = tf.placeholder(shape=[None, H, W, C], dtype=tf.float32, name='input')
```

**[Description]**

Our network is shown in Table.3 and Figure.6.

*for max pooling: 2x2/2 means
*for dropout: 0.9 means proportion kept

| | output | | | convolution | | | other* |
|---|---|---|---|---|---|---|---|
| | H | W | C | num | kernel | mac(M) | |
| input | 32 | 32 | 3 | | | | |
| preprocess | 32 | 32 | 8 | 8,8 | 3x3, 1x1 | 0.3 | |
| conv1 | 32 | 32 | 32 | 32 | 5x5 | 6.6 | |
| maxpool1 | 16 | 16 | 32 | | | | 2x2/2 |
| dense-block2 | 16 | 16 | 96 | concat: 4x16 | 3x3 | 8.3 | |
| maxpool2 | 8 | 8 | 96 | | | | 2x2/2 |
| dense-block3 | 8 | 8 | 192 | concat: 4x16 | 3x3 | 7.3 | |
| dropout3 | 8 | 8 | 192 | | | | 0.9 |
| maxpool3 | 4 | 4 | 192 | | | | 2x2/2 |
| dense-block4 | 4 | 4 | 320 | concat: 4x16 | 3x3 | 4.4 | |
| conv4 | 4 | 4 | 43 | | 1x1 | 0.2 | |
| dropout4 | 4 | 4 | 43 | | | | 0.9 |
| avgpool4 | 1 | 1 | 43 | | | | 4x4/4 |
| | | | | | total | 27.0 | |

*Table.3: Our network*

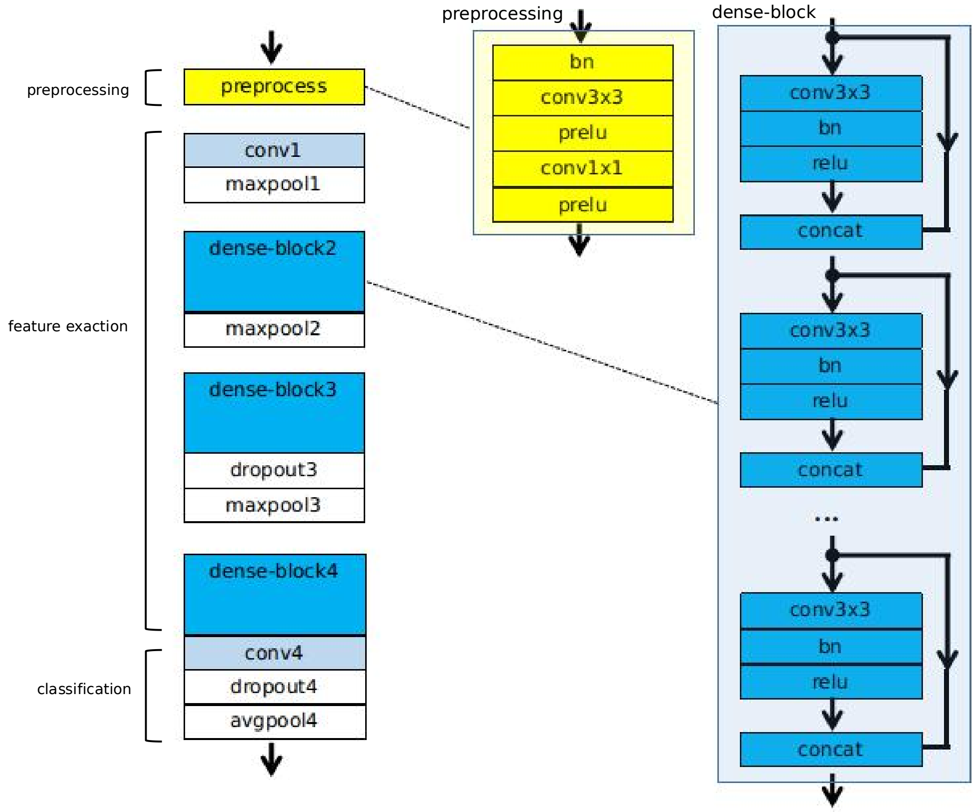



*Figure.6: Our network*

Our network has total of 27.0 million MACs (multiply-accumulation ops). It has 3 main component:

1. Preprocessing

In ”preprocess”, we first use batch normalization bn layer to standardize the 32x32x3 input to standard normal distribution. We next use 8 conv3x3 later followed by 8 conv1x1, with learnable parametric parametric relu as activation.

2. Feature extraction

In ”conv1”, we use 32 conv5x5, followed by bn and relu activation. Inspired by the work of [3], we next use 3 modified dense-blocks. Each dense-block has 4 concatenation of N conv3x3, followed by bn and relu activation. N =16,24,32 for “dense-block2,3,4” respectively. Max poolings are inserted in between to reduce the activation maps by half successively. The output of the feature extraction is 4x4x320 from “dense-block4”.

Note that there are two differences of our dense-block, compared to that of [3]:

- we are using conv-bn-relu-concat, while [3] is using bn-relu-conv-concat
- we do not use dropout within the dense-block. Dropout is only applied after the block if required. In [3], dropout is applied within block, at the conv before concatenation

3. Classification

Lastly, in “conv4”, the 4x4x320 feature is feed to 43 conv1x1, followed by bn and relu activation. Dropout is used. Global average pooling is applied to the feature to give a logit vector of 1x1x43.

### 3.4. Setting up of solver

**[Code]**

```
In [17]: #solver
epoch_log  = 2
max_run    = 9
batch_size = 128  #256  #96  384  #128
steps = (0, 3, 6, 8)
rates = (0.1, 0.01,  0.001, 0.0001)

learning_rate = tf.placeholder(tf.float32, shape=[])
solver = tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.9)
solver_step = solver.minimize(loss)
```

**[Description]**

Our loss function consist of cross entropy loss for classification. L2 regularization loss for the weights is added, with regularization factor=0.0005. We use stochastic gradient descent sgd solver for loss optimization. Batch size is 128. Momentum is set at 0.9.

In order to create an “infinite” number of train samples to prevent overfitting, we dynamically create new argument samples during the training epoch. We think that since the train samples is always changing, it is more difficult for the network to fit the train data. However, we have to be careful that the data change cannot be too big, which else will results in “jumps” in the training loss curve.

Our strategy for dynamic data argumentation is shown in Figure.7. We first divide the epoch into runs. For each run, we select 20% from a fix pool of extended samples, and perturbed the remaining 80% to create new train samples. We then run E epoch of sgd on these train samples. From our experiments, R=9 and E=24 seems to work the best, giving total training epoch of about 9x24 = 216.

Finally, the learning rate is stepped at 0.1, 0.01. 0.001, 0.0001 at the 0,3,6,8 of the runs.

```
▷ Given train samples
▷ Create extended train samples by flipping (steps.1 of Section.3.1)
▷ For run = 1:R
        ▷ Generate new argument samples (steps.2 and 3 of Section.3.1), with
          "new samples = 20% of extended samples + 80% of perturbed samples"
        ▷ For epoch = 1:E
              ▷ Do sgd gradient descent using new samples
```

*Figure.6: Our strategy for dynamic data argumentation*

### 3.5. Training, validation and testing results

We achieve a good results 99.68% accuracy and cross entropy loss 0.012501 on the test set. Table.4 shows the different results. Figure.7 and 8 shows the loss curve and accuracy curve on the train and validation sets. We note that the validation results are better because it does not contain perturbed samples.

| train (batch) | | valid | | test | |
|---|---|---|---|---|---|
| loss | acc | loss | acc | loss | acc |
| 0.00142 | 1.00000 | 0.00009 | 1.00000 | 0.01250 | 0.99683 |

*Table.4: Training, validation and testing results*

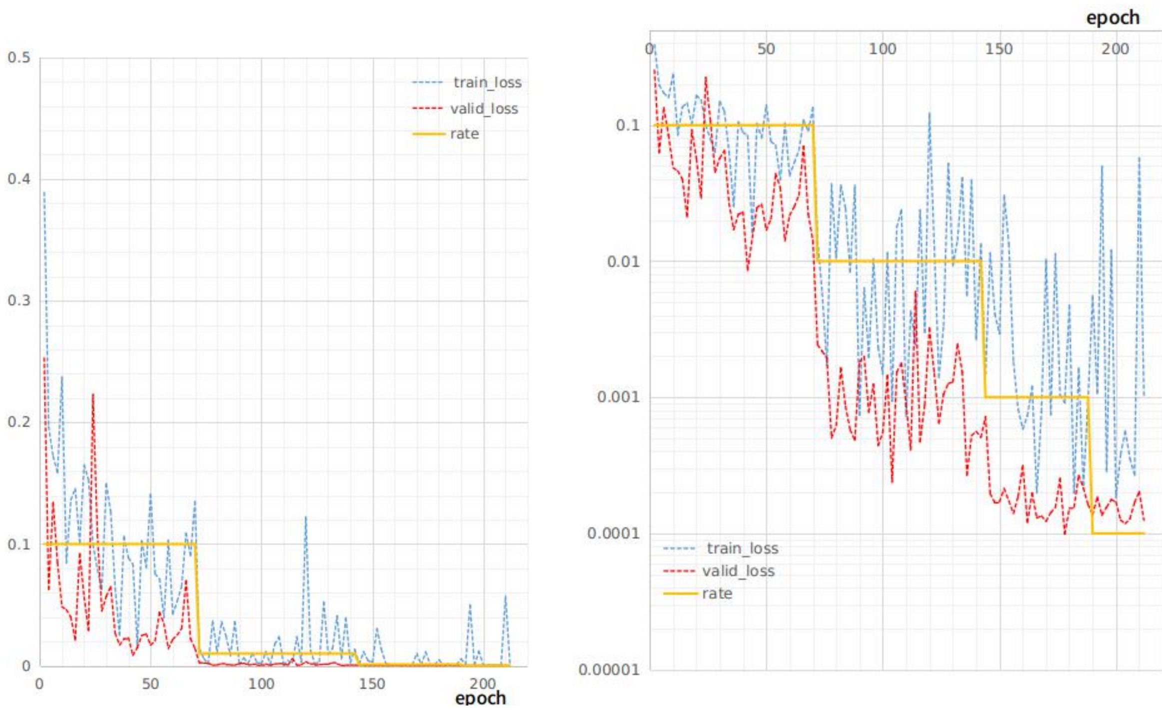



*Figure.7: Training, validation loss (right curve is the same left curve in log scale)*

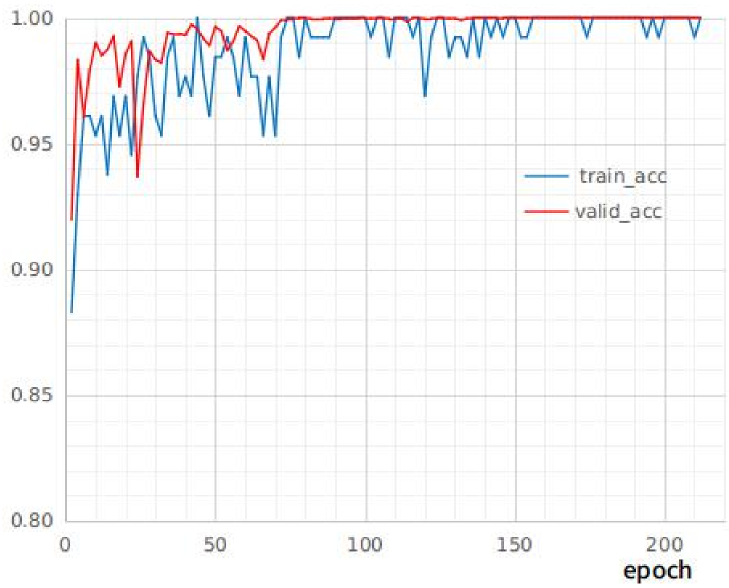
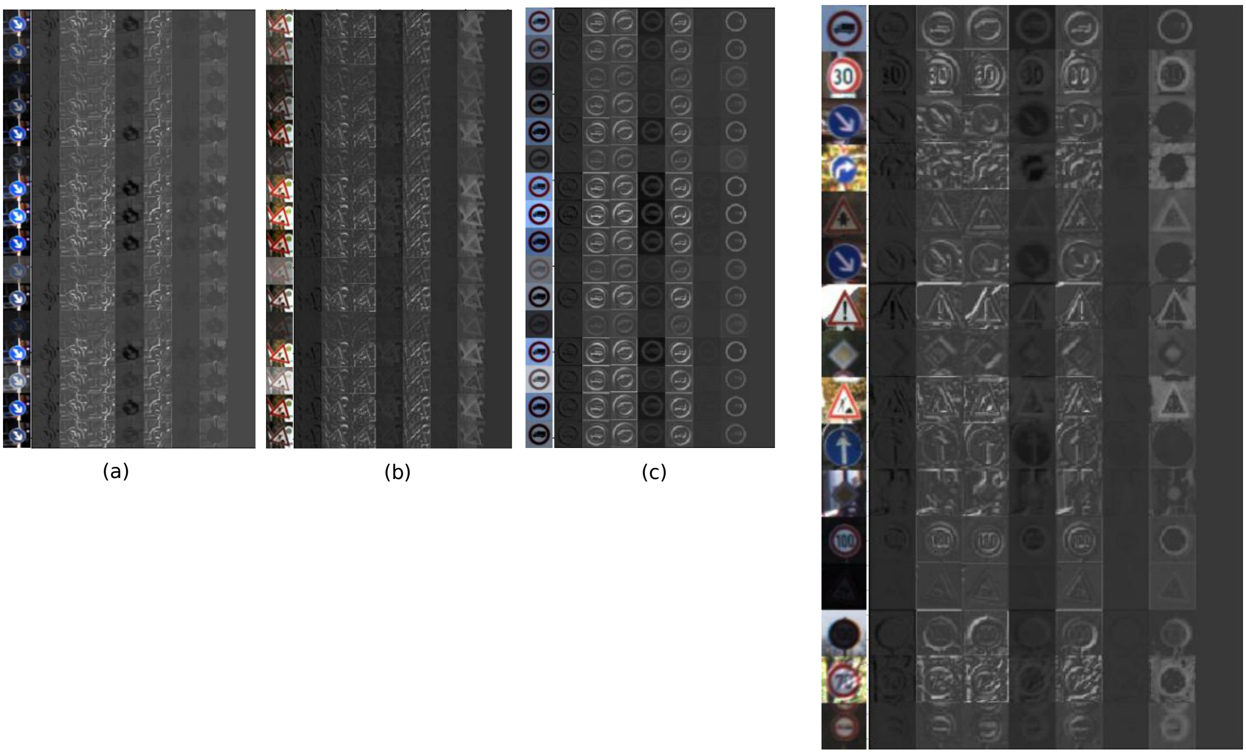



*Figure.8: Training, validation accuracy*

### 3.6. Discussion of results

<u>Results of prepossessing</u>

Figure.9 shows the 32x32x8 activation of the preprocessing layer of in Table.3. In (a),(b) and (c), input samples are synthetically generated from our random illumination perturbation, described in section3.1. (d) are samples from the test set. It can be seen that some of the 8 output channels are sensitive to blue and red colors and invariant to brightness. It is also observed that the few channels has weak activation, meaning that they may be remove for further complexity reduction.



(a) (b) (c) (d)

*Figure.9: Results of preprocessing on synthetic samples (a),(b),(c) and test samples (d)*

Results of wrong prediction

Figure.10 shows the wrong prediction on test set. There are 40 (0.32%) wrong predictions, possibly due to:

- occlusion
- artifacts on sign (e.g. white paint under the red non-entry sign)
- very small or blurred
- uneven illumination due to sunlight and shadow

Figure.11 shows the expected predicted probability on the test set. Note that this is not the confusion matrix. Given a test sample x with true label y_hat, the network predicts p_1 ...p_c ...p_43 probability for each class. Then:

*"expected predicted probability[y_hat, c] = Mean { p_c(x) }, over all x with true label y_hat"*

The figure is to be read row wise. Each row indexes y_hat and each column indexes c. (a) is a contrast enhanced version of (b) be better readability. Note that the results of Figure.11 supports the results for Figure.10. For example, "stop sign" as "no-entry sign" is one of the most wrongly predicted results.

## Network Design Considerations

We now outline the considerations that lead to the final design of our solution.

1. Establishing baseline performance:

We first do some initial experiments on LeNet and Vgg, see "Appendix.A Additional Experiments on LeNet, Vgg". Note that we do not do a lot fine tuning or extensive hyper-parameter search in these early experiments. We have the following test accuracy:

LeNet Only (MAC = 83.6 M): 89.80%
LeNet+Flip+Resample: 93.18%
LeNet+Flip+Resample+Argumentation: 97.61%
LeNet+Flip+Resample+Argumentation+ dropout: 95.56%
LeNet+Flip+Resample+Argumentation+ dropout + bn: 97.99%

We conjuncture that baseline performance of basic LeNet is about 98%. For Vgg, we can get 98.52% at MAC = 15.5M. These early experiments shows that:

Generating new data is necessary to improve performance
However, the LeNet network structure has limited accuracy and efficiency. It seems that Vgg can gives better results with less MAC .

Hence we decide to abandon LeNet and design a better structure.

2. Designing a Densenet

From the results of the state-of-art, see "Appendix B : State-of-art performances on traffic sign dataset", we note that multi-scale feature is important. For example, spatial transformer network corrects the scale and Sermanet's multi-scale CNN use skip connections to combine features of low and high scales.

Hence, we choose to use Densenet [3] because it by concatenating conv layers, it can “shortcut” low-scale features to the top. Further, Densenet has shown better performance then other architecture like Vgg, resNet or inception googleNet.

However, we find that Densenet however overfits very easily and results is sensitive to dropout, maybe due to the fact that our problem is small and easy. In fact our initial densenet actually performs worse than Vgg. To reduce overfitting, we make the adjustments:

- Use small number of conv filters
- Remove dropout inside the dense-block (we think due to the small input size of 32x32, the dropout can get magnified by concatenation). Use dropout outside the dense-block instead.

The network design is very much a trial and error process.


*Figure.10: 40(0.32%) wrong predicted test samples*

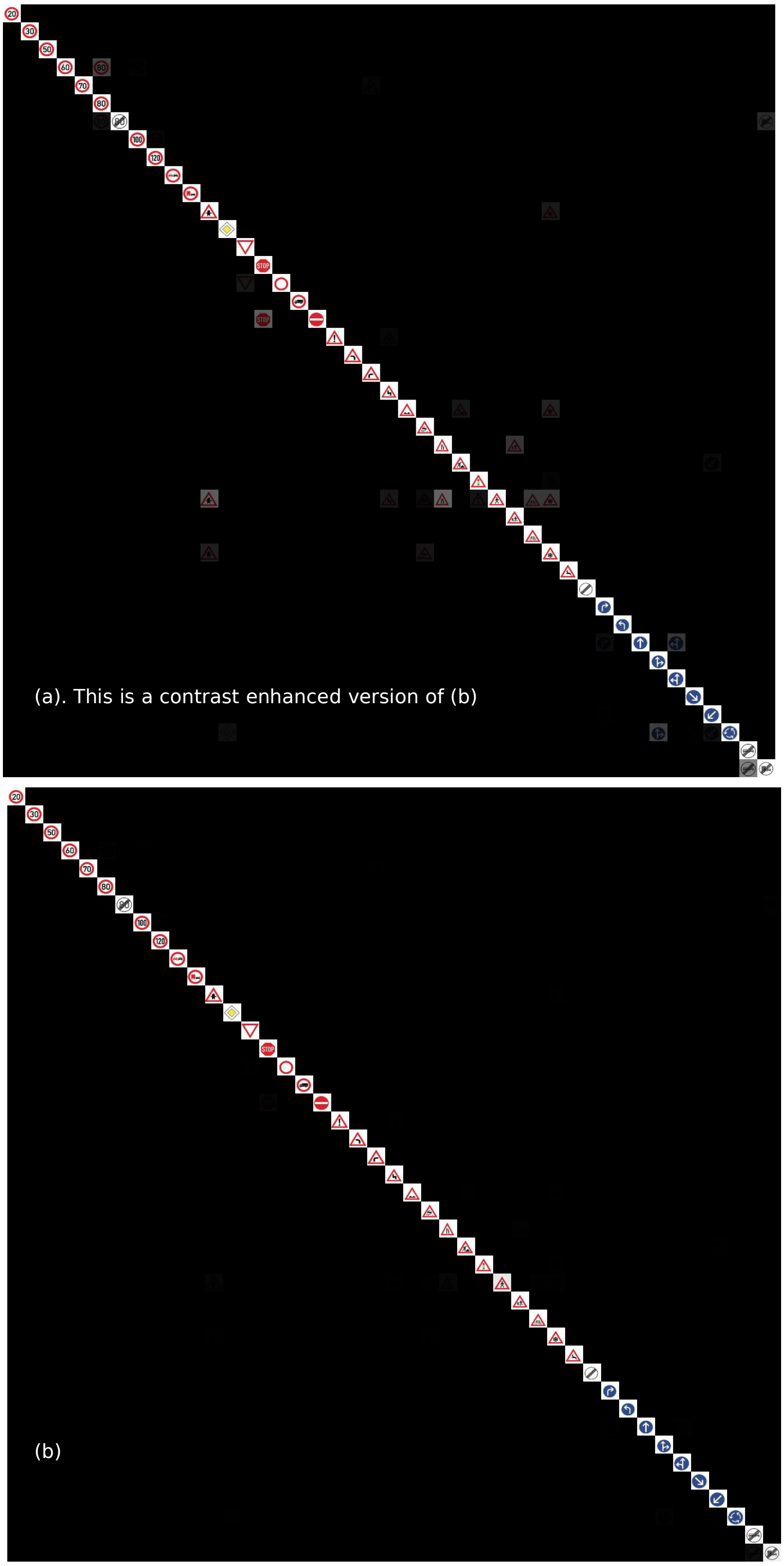



*Figure.11: Expected predicted probability (not confusion matrix)*

## 4. Testing on New Images

**[Code]**

```
In [20]: ### Load the images and plot them here.
         ### Feel free to use as many code cells as needed.

         test_files=['0004.jpg',   #normal
                     '0000.jpg',   #normal
                     '0007.jpg',   #occluded with snow
                     '0006.jpg',   #small
                     '0005.jpg',   #not in class
                     ]
         test_rois =[(54,180,125,260),(160,430,207,469),(181,32,321,142),(226,65,242,78 ),(388,408,700,676)]
         test_label=[13,38,2,49,-1]
```

**[Description]**

Figure.12 shows some test image containing German traffic sign we downloaded from the internet. The roi regions are marked by hand, cropped and resized to 32x32 and input to the trained network from Section.2.

| file | true label | crop 32x32x3 | orginal image | difficulty |
|---|---|---|---|---|
| 0004.jpg | 13:Yleid |  |  | Easy. |
| 0000.jpg | 38:Keep right |  |  | Easy, but sign is off centered in the crop. |
| 0007.jpg | 03:Speed limit (60km/h) |  |  | Difficult. Sign is occluded by snow. However, human can still recognize it. |
| 0006.jpg | 40:Roundabout mandatory |  |  | Moderate. However, human may have difficulty in reconizing it in the orginal image because of the small size |
| 0005.jpg | 14: Stop |  |  | Difficult? "ARRET" is stop sign in french. This sign is not in the training sample |

*Figure.12: Test images from the internet*

Figure.13 shows the prediction results on the cropped images. Except for image 0007.jpg, we have the rest correct. This is 80% correct. This is lower than the 99.68% of the test set. But the performance may be acceptable, since the wrong prediction is due to heavy occlusion.

| file | true label | crop 32x32x3 | top-5 predictions (proability label:classname) | comments |
|---|---|---|---|---|
| 0004.jpg | 13:Yleid |  | top0: 1.000000 13:Yield<br>top1: 0.000000 26:Traffic signals<br>top2: 0.000000 24:Road narrows on the right<br>top3: 0.000000 00:Speed limit (20km/h)<br>top4: 0.000000 14:Stop | Correct. Confidence is very high |
| 0000.jpg | 38:Keep right |  | top0: 1.000000 38:Keep right<br>top1: 0.000000 34:Turn left ahead<br>top2: 0.000000 05:Speed limit (80km/h)<br>top3: 0.000000 30:Beware of ice/snow<br>top4: 0.000000 36:Go straight or right | Correct. Confidence is very high |
| 0007.jpg | 03:Speed limit (60km/h) |  | top0: 0.352064 28:Children crossing<br>top1: 0.315537 29:Bicycles crossing<br>top2: 0.148701 02:Speed limit (50km/h)<br>top3: 0.037797 19:Dangerous curve to the left<br>top4: 0.025367 40:Roundabout mandatory | Wrong. Confidence is not high at all. The true label is not within top-5.<br>However, the top-3 prediction is speed limit 50 km/h which somehow close to the true sign. |
| 0006.jpg | 40:Roundabout mandatory |  | top0: 0.938421 40:Roundabout mandatory<br>top1: 0.056301 33:Turn right ahead<br>top2: 0.004292 34:Turn left ahead<br>top3: 0.000149 38:Keep right<br>top4: 0.000134 05:Speed limit (80km/h) | Correct. Confidence is very high |
| 0005.jpg | 14: Stop |  | top0: 0.999497 14:Stop<br>top1: 0.000480 29:Bicycles crossing<br>top2: 0.000013 07:Speed limit (100km/h)<br>top3: 0.000003 33:Turn right ahead<br>top4: 0.000002 08:Speed limit (120km/h) | Correct. Confidence is "surprisingly" high. I would expect a correct prediction due to the uquie hexagon shape, but I though the confidence is too high. |


*Figure.13: Prediction of the test images from the internet*

# Appendix A : Additional Experiments on LeNet, Vgg

| ***Lenet*** | | | | | | | |
|---|---|---|---|---|---|---|---|
| | **output** | | | **convolution** | | | |
| | **H** | **W** | **C** | **num** | **kernel** | **mac(M)** | others |
| input | 32 | 32 | 3 | | | | |
| conv | 32 | 32 | 108 | 108 | 5x5 | 8.3 | |
| maxpool | 16 | 16 | 108 | | | | 2x2/2 |
| conv | 16 | 16 | 108 | 108 | 5x5 | 74.6 | |
| maxpool | 8 | 8 | 108 | | | | 2x2/2 |
| dense | | | 100 | 100 | | 0.7 | |
| (droupout) | | | | | | | 0.5 |
| dense | | | 100 | 100 | | | |
| (droupout) | | | | | | | 0.5 |
| dense | | | 43 | 43 | | | |
| | | | | | **total** | 83.6 | |

| ***Vgg*** | | | | | | | |
|---|---|---|---|---|---|---|---|
| | **output** | | | **convolution** | | | |
| | **H** | **W** | **C** | **num** | **kernel** | **mac(M)** | others |
| input | 32 | 32 | 3 | | | | |
| conv | 32 | 32 | 64 | 64 | 5x5 | 4.9 | |
| maxpool | 16 | 16 | 64 | | | | 2x2/2 |
| conv | 16 | 16 | 32 | 32 | 1x1 | 0.5 | |
| conv | 16 | 16 | 32 | 32 | 3x3 | 2.4 | |
| conv | 16 | 16 | 64 | 64 | 1x1 | 0.5 | |
| maxpool | 8 | 8 | 64 | | | | 2x2/2 |
| conv | 8 | 8 | 64 | 64 | 1x1 | 0.3 | |
| conv | 8 | 8 | 64 | 64 | 3x3 | 2.4 | |
| conv | 8 | 8 | 128 | 128 | 1x1 | 0.5 | |
| conv | 8 | 8 | 64 | 64 | 1x1 | 0.5 | |
| conv | 8 | 8 | 64 | 64 | 3x3 | 2.4 | |
| conv | 8 | 8 | 128 | 128 | 1x1 | 0.5 | |
| maxpool | 4 | 4 | 128 | | | | 2x2/2 |
| dense | | | 256 | 256 | | 0.5 | |
| dense | | | 256 | 256 | | 0.1 | |
| dense | | | 43 | 43 | | 0 | |
| | | | | | **total** | 15.5 | |

*Figure.A.1: LetNet and Vgg network used in additional experiments*

| **comparing preprossing** | | | | **argumentation** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | **flip + resample** | **perturbance** | | **train (batch)** | | **valid** | | **test** | |
| | | **MAC (M)** | **preprocess** | | **geometric** | **illumination** | **loss** | **acc** | **loss** | **acc** | **loss** | **acc** |
| Lenet_2 | lenet + dropout + bn | 83.6 | rgb=(rgb-128)/128 | ● | ● | | 0.1274 | 0.9531 | 0.0024 | 0.9997 | 0.0852 | 0.9779 |
| Lenet_3 | lenet + dropout + bn + whiten | | | ● | ● | | 0.1448 | 0.9609 | 0.0129 | 0.9997 | 0.0959 | 0.9698 |

| **comparing augmentation** | | | | **argumentation** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | **flip + resample** | **perturbance** | | **train (batch)** | | **valid** | | **test** | |
| | | **MAC (M)** | **preprocess** | | **geometric** | **illumination** | **loss** | **acc** | **loss** | **acc** | **loss** | **acc** |
| Lenet_0 | lenet | 83.6 | rgb=(rgb-128)/128 | | | | 0.0227 | 1.0000 | 0.0957 | 0.9770 | 0.5318 | 0.8980 |
| | | | | ● | | | 0.0000 | 1.0000 | 0.0669 | 0.9903 | 0.6316 | 0.9318 |
| | | | | ● | ● | | 0.0218 | 0.9922 | 0.0052 | 0.9990 | 0.2713 | 0.9561 |
| | | | | ● | | ● | 0.0372 | 0.9922 | 0.0729 | 0.9913 | 0.5969 | 0.9512 |
| | | | | ● | ● | ● | 0.1615 | 0.9531 | 0.0043 | 0.9980 | 0.1338 | 0.9761 |
| Lenet_1 | lenet + dropout | | | ● | ● | | 0.2380 | 0.9375 | 0.0273 | 0.9933 | 0.1723 | 0.9682 |
| | | | | ● | ● | ● | 0.5629 | 0.8594 | 0.0512 | 0.9867 | 0.1621 | 0.9556 |
| Lenet_2 | lenet + dropout + bn | | | ● | ● | | 0.1274 | 0.9531 | 0.0024 | 0.9997 | 0.0852 | 0.9779 |
| | | | | ● | | ● | 0.0504 | 0.9844 | 0.0060 | 0.9983 | 0.0832 | 0.9817 |
| | | | | ● | ● | ● | 0.2666 | 0.9297 | 0.0037 | 0.9987 | 0.0730 | 0.9799 |
| Vgg_0 | vgg + bn | 15.5 | | ● | ● | | 0.0002 | 1.0000 | 0.0022 | 0.9993 | 0.1143 | 0.9721 |
| | | | | ● | | ● | 0.0332 | 0.9922 | 0.0112 | 0.9970 | 0.1818 | 0.9599 |
| | | | | ● | ● | ● | 0.1109 | 0.9688 | 0.0027 | 0.9997 | 0.0624 | 0.9852 |

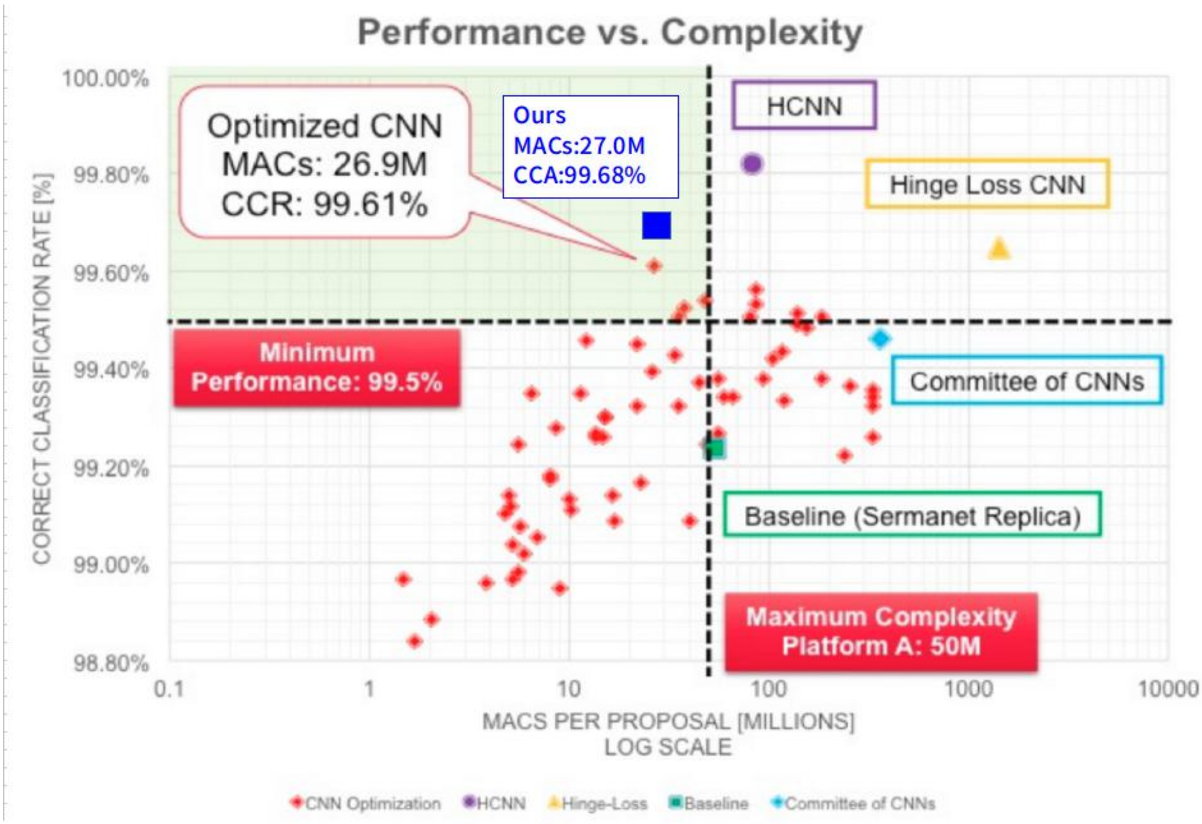
*Figure.A.2: Training and validation results in additional experiments*

# Appendix B : State-of-art performances on traffic sign dataset

| | test accuracy |
|---|---|
| Industrial performance/Cadence hierarchical CNN [3] | 99.82 |
| illumination Transformer + DenseNet (Ours) | 99.68 |
| 2 Spatial Transformer Networks[8] | 99.67 |
| Hinge Loss CNN[5] | 99.65 |
| Spatial Transformer Networks[4] | 99.61 |
| IDSIA Committee of CNNs[2] | 99.46 |
| Alex modified multi-scale CNN [6] | 99.33 |
| Vivek modified Vgg CNN [7] | 99.10 |
| Sermanet multi-scale CNN [1] | 99.17 |
| Human performance[1] | 98.84 |

[1] "Traffic Sign Recognition with Multi-Scale Convolutional Networks" - Pierre Sermanet and Yann LeCun, IJCNN 2011
[2] “Multi-column deep neural network for traffic sign classification” - D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, NN 2012.
[3] "Is Bigger CNN Better?" - Samer Hijazi - eNNS2016
https://ip.cadence.com/uploads/presentations/1345PM_ENNS_v10_Samer_Hijazi.pdf
[4] http://torch.ch/blog/2015/09/07/spatial_transformers.html
[5] "Traffic Sign Recognition With Hinge Loss Trained Convolutional Neural Networks" - Junqi Jin, Kun Fu, Changshui Zhang, IEEE ITS 2014
[6] http://navoshta.com/traffic-signs-classification/
[7] https://medium.com/@vivek.yadav/improved-performance-of-deep-learning-neural-network-models-on-traffic-sign-classification-using-6355346da2dc#.2c5sgna28
[8] https://github.com/Moodstocks/gtsrb.torch

*Figure.B.1: Comparing performances (accuracy)*




https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/archive/2016/07/13/how-to-optimize-your-cnn

*Figure.B.2: Comparing performances (MACs and accuracy)*

## References

[1] "Systematic evaluation of CNN advances on the ImageNet"-Dmytro Mishkin, Nikolay Sergievskiy, Jiri Matas, Arxiv 2016

[2] "Spatial Transformer Networks" - Max Jaderberg, Karen Simonyan, Andrew Zisserman, Koray Kavukcuoglu, Arxiv 2015

[3] "Densely Connected Convolutional Networks" - Gao Huang, Zhuang Liu, Kilian Weinberger, Laurens van der Maaten, Arxiv 2016