Computer Vision Assignment 3 - Traffic Sign Detection

Lizi Chen
lc3397@nyu.edu
https://drive.google.com/drive/folders/0B-2-l_M-RD3RUlNaWmdrUUdmNGs?usp=sharing

Final Submission:

37	↑5	Lizi Chen	0.97926	6

97.926%

Stats:

Epochs: 100
Models: A Modified version of *Cifar* Sequential to take `48-by-48` re-sized images for all.
Layers:

```
local model = nn.Sequential()
                                                --3*48*48
model:add(nn.SpatialConvolution(3,27,8,8))      --27*41*41
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2,2,2,2))        --27*20*20

model:add(nn.SpatialConvolution(27,128,5,5))    --128*16*16
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2,2,2,2))        --128*8*8

model:add(nn.View(8192))                        --128*8*8
model:add(nn.Dropout(0.5))

model:add(nn.Linear(8192, 320))
model:add(nn.Tanh())

model:add(nn.Dropout(0.5))

model:add(nn.Linear(320, 43))

return model
```

Links:

Please visit the link for all code files, cvs files, logs, and t7 trained models.
https://drive.google.com/drive/folders/0B-2-l_M-RD3RUlNaWmdrUUdmNGs?usp=sharing
An updated version of this report file is included in the Google Drive above.

References:

Most of the parameters tweaking and various models constructions are referenced directly from the following resources:

Pierre Sermanet, Yann LeCun
Traffic Sign Recognition with Multi-Scale Convolutional Networks
J. Stallkamp, M. Schlipsing, J. Salmen, C.Igel
Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition
Dan Ciresan, Ueli Meir, Jurgen Schmidhuber
Multi-column Deep Neural Networks for Image Classification
Online resources and tutorials of many for Image Classification.
Clement Farabet Code http://www.clement.farabet.net/code.html for colorspace preprocessing

Data Preprocessing:

Read in train and test images:

```
function trainSamples(dataset, idx)
    a = dataset[idx]
    classId, track, file = a[9], a[1], a[2]
    file = string.format("%05d/%05d_%05d.ppm", classId, track, file)
    return transformInput(image.load(DATA_PATH .. '/train_images/'..file),r)
end

for i=1,dataset.train.size do
    dataset.train.data[i] =  trainSamples(trainData, i)
    dataset.train.label[i][1] = trainLabels(trainData, i)
end
```

Use **transformInput** to change the dimension of an image into desired weight and height, 48

```
function transformInput(image,a)
      croppedImg = image.crop(image,a[5],a[6],a[7],a[8])
      return image.scale(croppedImg, WIDTH,HEIGHT)
end
```

Use `image.rgb2yuv(dataset.train.data[n])` to convert **RGB -> YUV**

```
for n = 1, dataset.train.size do
        dataset.train.data[i] = image.rgb2yuv(dataset.train.data[n])
end
```

Normalize y, u, v channels by using **:mean()** and **:std()** functions as shown below

mean = dataset.train.data[{ {}, current, {}, {} }] : mean()

std = dataset.train.data[{ {}, current, {}, {} }] : std()

Training, Testing and Kaggle Results:

Use a simple Java program to compare the test label retrieved from German Traffic Sign Benchmark website when doing sample training of epochs less than 100. Tweak model parameters to find faster convergence curve.

I modified the model based on its original given version; which I call *Cifar* here, has already reached 0.9451 accuracy. I have some general ideas about what parameters can be changed to test so as to have a better result accuracy. My initial approach is to change the parameters first. Such as substitute a Convolution of 108 features in the first layer and a 200 features in the second layer to the ones in *Cifar*. This 108-first-200-second model is suggested in Yann LeCun's paper; as given in the *Reference* part above. However, LeCun used a Multi-Scale one that uses nn.Concat() function that I was not introduced in the very beginning. The following table shows some work that I have improved based on the original *Cifar* model.

Model Id	Model Name	Pre-processing	Input Image size	Model Detail (Partially shown in the lua files under ./models directory)	Epoch	Test Results Difference	Kaggle Result	Machine
1	Original cifar	N/A	32*32	**Conv[16(5*5)]** - **Tanh** - MaxPo(2,2) - **Conv[128(5*5)]**- **Tanh** - MaxPo(2,2) - View-Linear-**Tanh**-Linear	***6 (Test Only)***	7873	**0.9451*	local
2	Original cifar	N/A	32*32	**Conv[16(5*5)]** - **Tanh** - MaxPo(2,2) - **Conv[128(5*5)]**- **Tanh** - MaxPo(2,2) - View-Linear-**Tanh**-Linear	30	1309	Ignore	local
3	Cifar 2	RGB -> YUV	32*32	**Conv[108(7*7)]** - **Tanh** - MaxPo(2,2) - **Conv[200(5*5)]** - **Tanh** - MaxPo(2,2) - View - **Dropout(0.5)**-Linear-**ReLU**-**Dropout(0.5)**-Linear	***6 (Test Only)***	4837	Ignore	local
4	Cifar 2	RGB -> YUV	32*32	**Conv[108(7*7)]** - **Tanh** - MaxPo(2,2) - **Conv[200(5*5)]** - **Tanh** - MaxPo(2,2) - View - **Dropout(0.5)**-Linear-**ReLU**-**Dropout(0.5)**-Linear	35	923	0.92431	local
5	Cifar 2	RGB -> YUV	32*32	**Conv[108(7*7)]** - **Tanh** - MaxPo(2,2) - **Conv[200(5*5)]** - **Tanh** - MaxPo(2,2) - View - **Dropout(0.5)**-Linear-**ReLU**-**Dropout(0.5)**-Linear	60	774	0.93777	AWS 16 read-in threads c3.2xLarge
6	Cifar 2	RGB -> YUV	32*32	**Conv[108(7*7)]** - **Tanh** - MaxPo(2,2) - **Conv[200(5*5)]** - **Tanh** - MaxPo(2,2) - **Conv[250(3*3)]** -View - **Dropout(0.5)**-Linear-**Tanh**-**Dropout(0.5)**-Linear	60	579	0.95451	AWS 16 read-in threads c3.2xLarge

Model id 2 suggested that by running 30 epochs; compared to a 6-epochs run, we can have an almost 6-fold accuracy improve. Therefore, I ran 6 to 10 epochs just to test and compare, and run 30 to 60 epochs for models that are most likely to have an assertive improvement. By this approach, I can save time to run more models and collect data.

Starting from Model id 3, I followed the suggestion given by Yann LeCun's paper, where he converted RGB to YUV for a better result. The result came out better when comparing to the original *Cifar* model, although I did not run more than 60 epochs because the convergence curve had very minimal change. Also, The non linearity Tanh had a better outcome than ReLU.

7	Conv48-1	RGB -> YUV + Data Resample	**48*48**	**Conv[27(8*8)]** - **Tanh** - MaxPo(2,2) - **Conv[128(5*5)]** - **Tanh** - MaxPo(2,2) - View - Linear - **ReLU** - **Dropout(0.5)** - Linear	***6 (Test Only)***	1121	Ignore	local
8	Conv48-1'	RGB -> YUV + Data Resample	**48*48**	**Conv[27(8*8)]** - **Tanh** - MaxPo(2,2) - **Conv[128(5*5)]** - **Tanh** - MaxPo(2,2) - View - **Dropout(0.5)** -Linear - **Tanh** - **Dropout(0.5)** - Linear	120	301	0.97926	AWS 16 read-in threads c3.2xLarge
9	Conv48-1'	RGB -> YUV + Data Resample	**48*48**	**Conv[27(8*8)]** - **Tanh** - MaxPo(2,2) - **Conv[128(5*5)]** - **Tanh** - MaxPo(2,2) - View - **Dropout(0.5)** -Linear - **Tanh** - **Dropout(0.5)** - Linear	***6 (Test Only)***	962	Ignore	local

The paper from Sermanet suggested using 48pixel-by-48pixel as input image. Thus, starting from Model id 7, I started pre-processing the data to crop to 48-by-48, which itself has dramatically improved the result accuracy.

Also, by adding Dropout(0.5) in the later layers has improved the training time as well as results.

By lowering down the number of features in early layer to save more space and gradually increase the number of features in the second layer, we received a better result in model #8. Within only 120 epochs training, it reaches 0.97926.

10	IDSIA	RGB -> YUV + Data Resample	**48*48**	**Conv[100(7*7)]** - **ReLU** - MaxPo(2,2):***ceil*** - **Conv[150(4*4)]** - **ReLU** - MaxPo(2,2):***ceil*** - **Conv[250(4*4)]** - **ReLU** - MaxPo(2,2):***ceil*** - View - **Dropout(0.5)** - Linear - **Dropout(0.5)** - Linear	***6 (Test Only)***	1554	Ignore	AWS 16 read-in threads c3.2xLarge
11	IDSIA-2	RGB - > YUV + Data Resample	**48*48**	**Conv[100(7*7)]** - **ReLU** - MaxPo(2,2):***ceil*** - **Conv[150(4*4)]** - **ReLU** - MaxPo(2,2):***ceil*** - **Conv[250(4*4)]** - **ReLU** - MaxPo(2,2):***ceil*** - View - **Dropout(0.5)** - Linear - **ReLU** - **Dropout(0.5)** - Linear	***6 (Test Only)***	2975	Ignore	AWS 32 read-in threads c4.4xLarge
12	IDSIA-3	RGB -> YUV + Data Resample	**48*48**	**Conv[100(7*7)]** - **Tanh** - MaxPo(2,2) - **Conv[150(4*4)]** - **Tanh** - MaxPo(2,2) - **Conv[250(4*4)]** - **Tanh** - MaxPo(2,2) - View - **Dropout(0.5)** - Linear - **Tanh** - **Dropout(0.5)** - Linear	24	555	0.95605	AWS 32 read-in threads c4.4xLarge
13	Conv48-3	RGB -> YUV + Data Resample	**48*48**	ConvNet + SpatialBatchNormalization + Maxpooling:Ceil	25	600+	Ignore	AWS 16 read-in threads c3.2xLarge
14	IDSIA-3	RGB -> YUV + Data Resample	**48*48**	**Conv[100(7*7)]** - **Tanh** - MaxPo(2,2) - **Conv[150(4*4)]** - **Tanh** - MaxPo(2,2) - **Conv[250(4*4)]** - **Tanh** - MaxPo(2,2) - View - **Dropout(0.5)** - Linear - **Tanh** - **Dropout(0.5)** - Linear *(Same as ID_12)*	**40**	471	0.96270	AWS 32 read-in threads c4.4xLarge

Same outcome as previous testing, using ReLU somehow lowers the accuracy. Perhaps only in the early training stage. This will be studied afterwards. Models #10, #11, and #12 are inspired by IDSIA team which I read from previous referenced papers. By adding extra convolution layer increments the

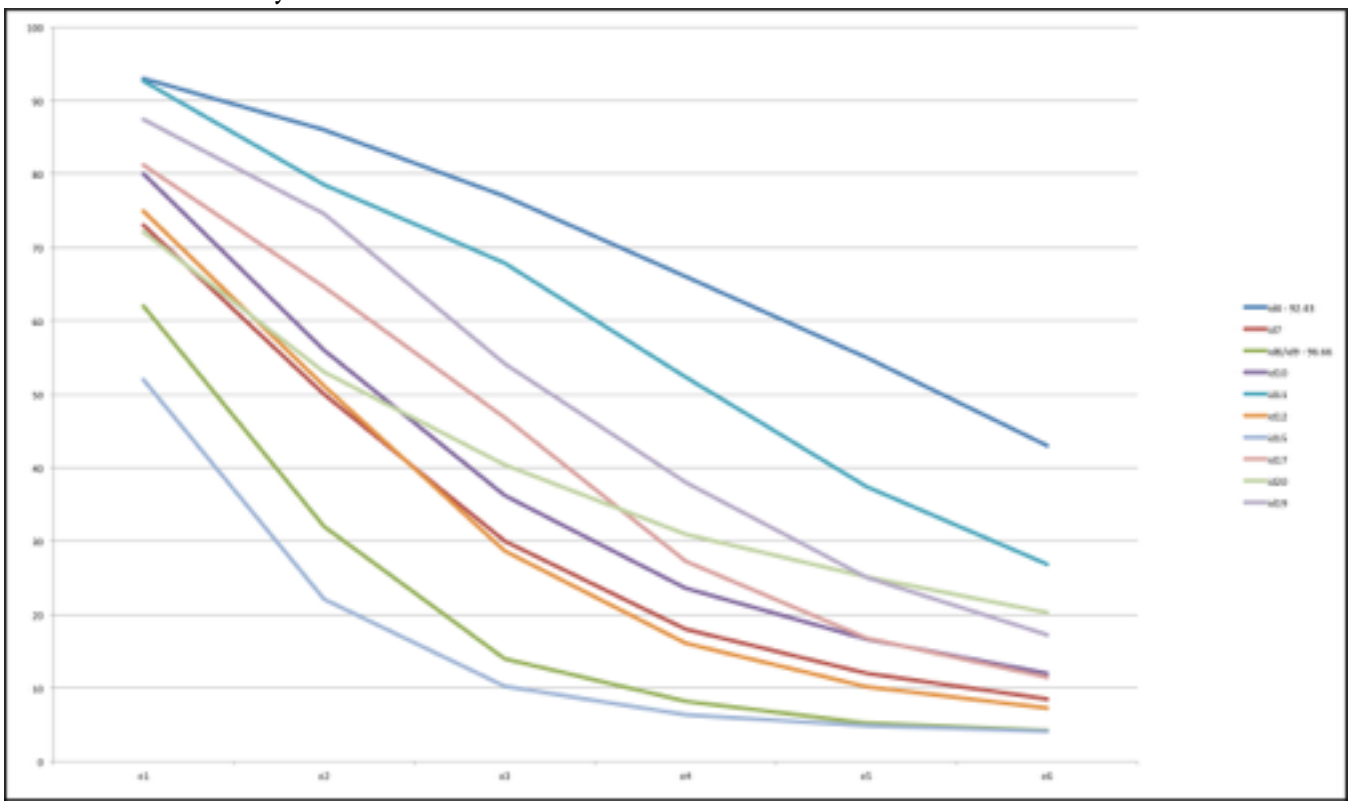
training time; however, the result was pleasant - within 24 epochs, the accuracy has already reaches 0.95605; within 40 epochs, the accuracy is 0.96270. Further training work may result in a better result above 0.979260.

As there is still room to improve, I tried to use SpatialContrastiveNormalization for both 48-by-48 cropped image and 32-by-32 cropped image. The early training result turned out to be non-optimistic compared to simple *Cifar* model. Thus, further work will be left to be done when time permits.

I also tried to use a (prahaps misunderstood version of) Yann LeCun's Multi-Scale model; as shown in Model #19. It used nn.Concat() for connecting two parallel nn.Sequential(). However, the result was not pleasing, which misunderstanding must have been involved. Further study is needed.

15	Conv48-1''	RGB -> YUV + Data Resample + SpatialContrastiveNormalization	**48*48**	Conv48-1 = 2maxPo l view (128*8*8)	***6 (Test Only)***	1015	Ignore	AWS 16 read-in threads C4.8xLarge
17	Conv48-1	RGB - > YUV + Data Resample + SpatialContrastiveNormalization	**48*48**	Conv(3,27,8,8)MaxPo(3,3,2,2)Conv(27,128,5,5)MaxPo(3,3,2,2)Conv(128,250,4,4)MaxPo(2,2,2,2)View(100)Dropout(0.5)Linear(1000,320) Dropout(0.5)Linear(320,43)	***6 (Test Only)***	1722	Ignore	AWS 16 read-in threads C4.8xLarge
18	Cifar - 3	RGB -> YUV	32*32	Add SpatialBatchNormalization after each Convolutions in Cifar Model	***6 (Test Only)***	1841	Ignore	AWS 16 read-in threads C4.8xLarge
19	Multiscale - 1	RGB - > YUV + Data Resample	32*32	Conv(3,108,5,5) - Tanh - MaxPo(2,2) Branch - 1: Sequential: Conv(108, 108, 5, 5) - Tanh - MaxPo(2,2) - Vector Branch - 2: Vector nn.Concat(2):add(Branch - 1 and Branch - 2) Linear - Tanh - Dropout(0.5) - Linear	***6 (Test Only)***	2499	Ignore	
20	Cifar - 3'	RGB - > YUV + Data Resample	32*32	Add SpatialBatchNormalization after each Convolutions in Cifar Model + Data Resampling	40	545	0.95684	AWS 16 read-in threads C4.8xLarge

A simple early stage training convergence curve diagram is shown as below, which I used to decide which model should be run with more epochs and what parameters and design improve or decrease the accuracy.



Takeaways and Further Improvement:

The work is not perfect here; however there are a lot to take away from this assignment.

1. Somehow Tanh performs better than ReLU in the test cases.
2. Knowing to use Cuda with GPU on HPC/AWS can greatly reduce the waiting time.
3. Error curve does not necessarily proportional to the final result. For example, the Model #19 has a curve that has a steep error lowering curve; however, its outcome was not good.
4. Result accuracy does not improve much after 30 epochs for thin convolutional network, but a multi-layer convNet has the potential to keep lowering error rate for many further epochs.
5. Importance of Data Pre-processing is greater than model tweaking. Cropping the image size from 32-by-32 to 48-by-48 dramatically improve the overall accuracy. Changing the RGB to YUV also improve the accuracy, even though with minor difference. Data Resampling needs a better approach to take a leap in the overall pre-processing phase.
6. Process Speed Optimization - Multi-Thread data reading and multi-thread processing by using more CPU core has dropped my initial training time from 12 minutes per epoch to 5 minutes per one.
7. Training model # 14, which has the model of a 16 layers as shown below.

```
local model = nn.Sequential()

model:add(nn.SpatialConvolution(3,100,7,7))      -- 1
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2,2,2,2))         -- 2
```

```
model:add(nn.SpatialConvolution(100,150,4,4))    -- 3
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2,2,2,2))         -- 4

model:add(nn.SpatialConvolution(150,250,4,4))    -- 5
model:add(nn.Tanh())
model:add(nn.SpatialMaxPooling(2,2,2,2))         -- 6

model:add(nn.View(2250))  -- 250*3*3
model:add(nn.Dropout(0.5))

model:add(nn.Linear(2250, 300))                  -- 7
model:add(nn.Tanh())
model:add(nn.Dropout(0.5))

model:add(nn.Linear(300, 43))                    -- 8

return model
```

This model potentially can have better results as it only had 40 epochs and its accuracy is greater than 96%.

A multi-scale model from Yann LeCun's file is still running on AWS, result will be included in the Google Drive.

To see an updated version of this file, please visit the Public Google Drive Folder:

https://drive.google.com/drive/folders/0B-2-l_M-RD3RUlNaWmdrUUdmNGs?usp=sharing

A cleaner version of code will be uploaded to my github:

https://github.com/lizichen/german_traffic_sign_recognition

Thanks,
Lizi Chen
10 Dec 2016