Untitled

February 3, 2019

1 Data Set Summary & Exploration

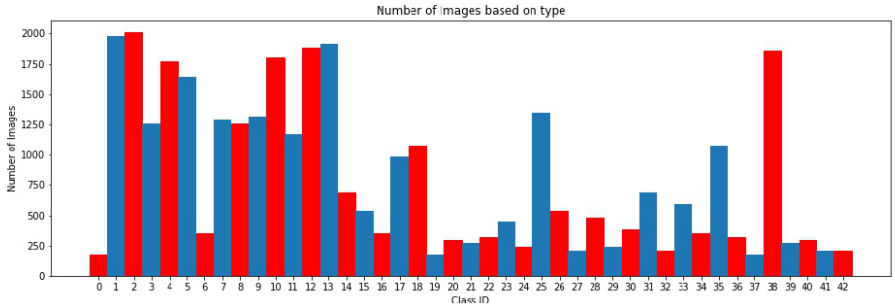
1.0.1 1. As the dataset was in `pickel`(.p) format like `train.p`, So, I used pickel to read the file and used `set` method to get unique labels and `len` to get size of data and numpy method `shape` to get shape of image

- The Size of training set is `34799` images.
- The size of the validation set is `4410` images.
- The size of test set is `12630` images.
- The shape of a traffic sign image is `32x32x3`.
- The number of unique classes/labels in the data set is `43`

1.0.2 2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data ...

```
In [4]: import matplotlib.pyplot as plt
        f = plt.figure(figsize=(20, 10))
        plt.axis('off')
        plt.imshow(plt.imread('n_images_vs_class_id.jpg'));
```



2 Design and Test a Model Architecture

2.1 1. Prerocessing Dataset

I am dividing each image by `255` to normalize it as by dividing an image by 255 it helps in removing distortions and shadows in a image.

I also tried to use `Batch Normalization` as first step in model `without preprocessing` and it was also giving same result like normalizing an image.

I even tried `(image -128)/128` but I was not getting better accuracy with this in my model.

And normalization of image does not change its appearance, it just changes its scale from 0-255 to 0-1 as you can see below

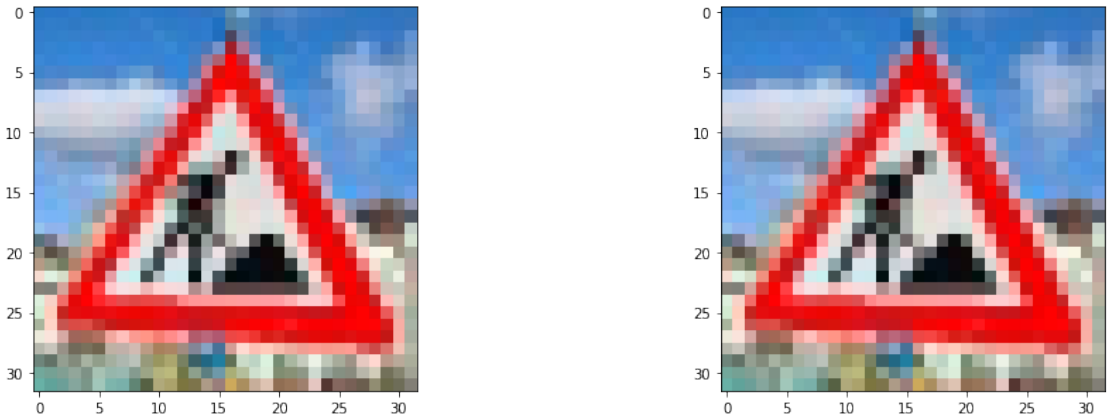
```
In [24]: !ls
```

```
CODEOWNERS                    README.md                          Untitled.ipynb
examples                    set_git.sh                         valid.p
internet_images                   signnames.csv                        visualize_cnn.png
LICENSE                           test.p                           write up.pdf
n_images_vs_class_id.jpg  Traffic_Sign_Classifier.ipynb  writeup_template.md
normalized_yield.jpg          train.p
```

```
In [27]: f = plt.figure(figsize=(16, 5))

         plt.subplot(1,2,1)
         plt.imshow(plt.imread('internet_images/road_work.jpg'))
         plt.subplot(1,2,2)
         plt.imshow(plt.imread('normalized_yield.jpg'))
```

```
Out[27]: <matplotlib.image.AxesImage at 0x7f0c8c41ec50>
```



2.2 2. Model Architecture

2.2.1 In my model, I am using three type of blocks which I have created.

1. Resnet block
2. Simple Convolution
3. Convolution

2.2.2 1. Resnet Block: It is based on Resnet architecture but here I have modified it which looks like this

Layer	Description
Input	
Convolution 1x1	1x1 stride, same padding,initializing weight using `He Normal` outputs 32x32x64, L2 Regularizer
Convolution 1x1	1x1 stride, same padding,initializing weight using `He Normal` outputs 32x32x64, L2 Regularizer
Batch Normalization	
Add Input to Batch Normalization output	
Leaky RELU	

2.3 2. Simple Convolution block: It looks something like this

Layer	Description
Input	32x32x3 RGB image
Convolution fxf(f=filter)	sxs stride, same padding, `He initializer`
Batch Normalization	
Leaky Relu	

2.4 3. Conv Block: It looks similar to simple convolution block only difference is here we are adding two extra layers which are shown below

Layer	Description
Input	
Convolution fxf(f=filter)	sxs stride, same padding, `He initializer`
Batch Normalization	
Leaky Relu	
Convolution 1x1	stride=1x1, `He normal` Initializer
Batch Normalization	

2.5 Full model Archictecture

Layer	Description
`Simple Convolution` output_channels = 64	kernel_size=3, stride=2, Input_image=(nx32x32x3), output=(nx15x15x 64)
`Resnet Block` output_channels = 64	output=(nx15x15x64)
`Convolution Block` output_channels= 128	kernel_size=3, stride=2, output=(nx7x7x128)
`Resnet block` output_channel=128	output = (nx7x7x128)
`Convolution Block` output_channel=256	kernel_size=1, stride = 1, output=(nx7x7x256)
`Resnet Block` output_channel=256	output = (nx7x7x256)
`Convolution Block` output_channel = 512	kernel_size= 3, stride=2, output=(nx3x3x512)
`Resnet Block` output_channel=512	output=(nx3x3x512)
`Convolution Block`, output_channel = 1024	kernel_size=3, stride=2, output=(nx1x1x1024)
`Resnet Block` output_channel = 1024	output = (nx1x1x1024)
`Simple Convolution Block` output_filters=128	kernel_size = 1, stride=1, output=(nx1x1x128)
`Simple Convolution Block` output_filters=128	kernel_size=1, stride=1, output = (nx1x1x128)
Global Average Pooling	output=(nx128)
BatchNormalization	
Dropout	p=0.25
Dense	activation = relu
BatchNormalization	
Dropout	p=0.5
Dense	activation = Softmax

3 3. Training

1. To Train the model first I converted the `y` values (Labels) to one hot encoding.
2. To compile the model I am using `Adam` optimizer, `categorical_crossentropy` loss and `accuracy` Matrix.
3. Using batch size of 32 and 20 epochs, But I am also using callback function which stops the training when it reaches 97% validation accuracy as sometime it bounces back to 96% validation score.

4 4. My approach to reach the solution

4.0.1 My final model results were:

- training set accuracy of `98.72` %

- Validation set accuracy of `97.17` %
- Test set accuracy of `95.94` %

4.0.2 The steps I followed to get upto the outputs are:

1. I started with LeNet Architecture which was taught in class with tensorflow but the accuracy was very poor So, I thought of applying some different architecture.
2. And while I was checking tensorflow documentation they were explaining using keras, So, I thought of using keras and after checking keras documentation and some pre designed architectures, I learnt using Keras and decided to use keras as our next project is also in keras and even tensorflow documentation mentioned keras.
3. So, I googled and found many architectures for Image recogntion like Vgg16, AlexNet, ResNet, ResNext, DenseNet and manymore.
4. Then, I tried small part of vgg and I got nice results without even preprocessing the image or converting to grayscale.
5. And it improved further more when I added more layers including the ones with 1x1 convolutions and got better results.
6. Then I read somewhere that using `leaky relu` after convolution and batch normalization we can get better score.
7. And it easily crossed the 93 % accuracy But in project 1st video udacity founder has challenged to get minimum of 97% score, So I started modifying model to get better accuracy.
8. And thought to apply `Resnet` but as image size is 32x32x3 so, I used only kernel_size of (1,1) as well as stride of (1, 1), Then I got above 95% accuracy.
9. But to get more better accuracy, I started searching more how to get more accuracy, I tried many ways and find out with `He initializer` and added `Dropouts` to reduce the chance of `overfitting` and increasing validation scoreand after that I was getting around between 96 to around 97.8.
10. But while traing it was going beyond 97% accuracy and again falls around 96.5%, So I thought of stopping the training then I found callbacks in keras documentation and as well as on stackoverflow.

5 Test a Model on New Images

5.1 Here are five German traffic signs that I found on the web:

```
In [12]: import glob
         img_internet = glob.glob('internet_images/*.jpg')
         img_internet = [plt.imread(i) for i in img_internet]

In [13]: f = plt.figure(figsize=(20, 16))
         for i in range(5):
             plt.subplot(1, 5, i+1)
             plt.imshow(img_internet[i])
             plt.axis('off')
             plt.subplots_adjust(top = 0.99, bottom=0.01, hspace=0.5, wspace=0.4)
```

Image	Prediction
General Caution	General Caution
Wild Animal Crossing	No Passing
Turn Right Ahead	Turn Right Ahead
Yield	Yield
Road Work	Road Work

The model was able to correctly guess `4 of the 5` traffic signs, which gives an `accuracy of 80%`. And also the wrong one `Wild Animal Crossing` appears similar to `No Passing` after `normalization`.

5.2 Image wise analysis for each image

5.2.1 1. Image of `general caution`

Model is sure that the image is of `General Caution` as the probability is 0.99.The Top five softmax probabilities are

Probability	Prediction
0.99	General Caution
0.31e-05	Pedestrians
0.18e-05	Traffic signals
0.65e-06	Go straight or left
0.30e-06	Bumpy road

5.2.2 2. Image of `animal crossing`

This one it has predicted wrong but if we see the image of `animal crossing` and `No passing` after normalization it appears to be similar. And the top five softmax probabilities are:

Probability	Prediction
0.94	No Passing
0.047	Dangerous curve to the right
0.17e-02	No passing for vehicles over 3.5 metric tons
0.12e-02	Slippery road
0.63e-03	Dangerous curve to the left

5.2.3 3. Image of `Turn right ahead`

This one is also correctly predicted by model with a probabily of 0.83 and is sure that it is image of `Turn right ahead` sign.And the top five softmax probabilities for this image are:

Probability	Prediction
0.83	Turn right ahead
0.09	Speed limit (80km/h)
0.03	Speed limit (100km/h)
0.9e-02	Keep left
0.7e-02	Priority road

5.2.4 4. Image of `Yield`

Model is sure that the image is of `Yield` as the probability is 0.99.The Top five softmax probabilities for this image are:

Probability	Prediction
0.99	Yield
0.12e-05	Road work
0.7e-06	Speed limit (60km/h)
0.39e-06	Keep left
0.35e-06	End of no passing by vehicles over 3.5 metric

5.2.5 5. Image of `Road work`

Model is sure that the image is of `Road work` as the probability is 0.99.The Top five softmax probabilities for this image are:

Probability	Prediction
0.88	Road work
0.12	Bumpy road
0.89e-04	No passing for vehicles over 3.5 metric tons
0.54e-04	Ahead only
0.1e-04	Dangerous curve to the right