**Traffic Sign Classifier**

**Write up** 1st submit:August 10th, **2nd submit: August 11th** Kenta Kumazaki

**1. Purpose**

The goals / steps of this project are the following:

* Load the data set
* Explore, summarize and visualize the data set
* Design, train and test a model architecture
* Use the model to make predictions on new images
* Analyze the softmax probabilities of the new images
* Summarize the results with a written report

**2. Output**

* Make a pipeline that run deep learning to classify traffic signs.  
  I used the Workspace in Udacity for this project.
* Reflect on my work in a written report.

**3. Submission**

**(1) GitHub**

<https://github.com/kkumazaki/Self-Drivig-Car_Project3_Traffic-Sign-Classifier-Project.git>

**(2) Directory**

<folder: main>

* **Writeup\_of\_Lesson14.pdf**: This file
* **Traffic\_Sign\_Classifier.ipynb**: Pipeline file (Jupyter Notebook)
* **Traffic\_Sign\_Classifier.html**: Pipeline file (HTML)
* Image files are saved as following:

**<folder: Test\_pics>**

Additional Test images downloaded by the following site.

<https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/version/1>



**<folder: Test\_results>**

Modified Test images for Traffic Sign Classification.



**4. Reflection**

**(1)Description of my pipeline and results**

My pipeline consisted of 4 steps as following, including the preparation Step: 0.

Step 0: Load the Data

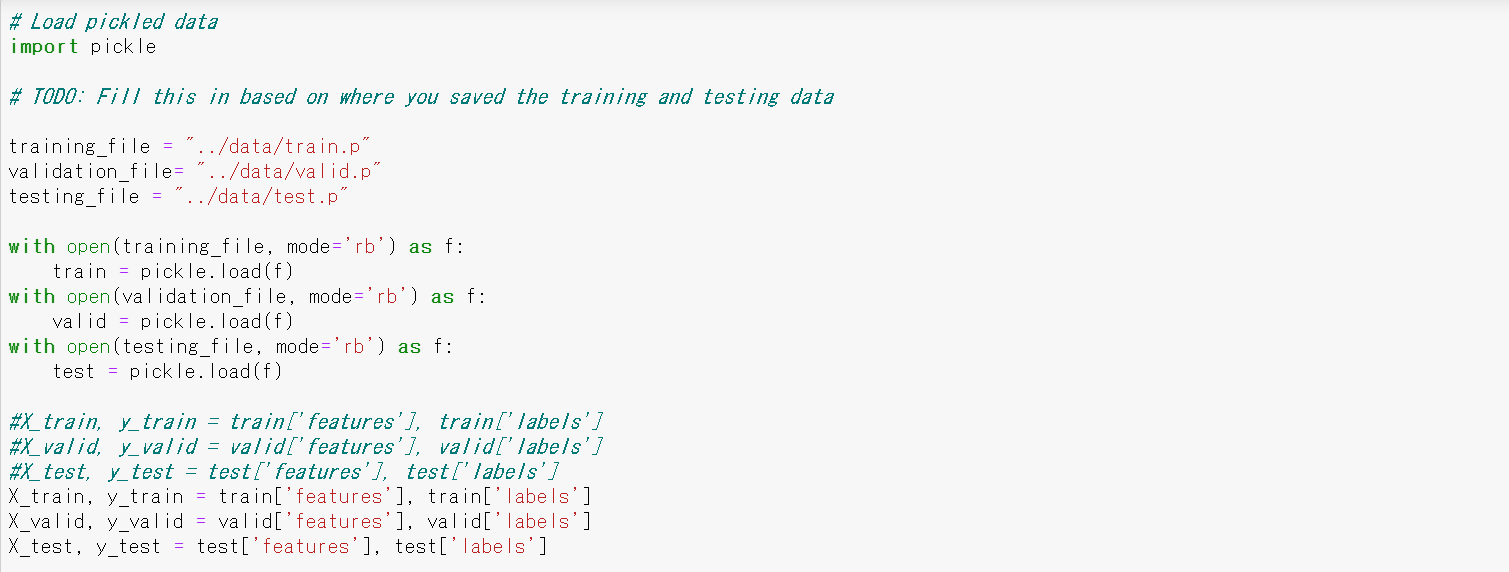
Step 1: Dataset Summary & Exploration

Step 2: Design and Test a Model Architecture

Step 3: Test a Model on New Images

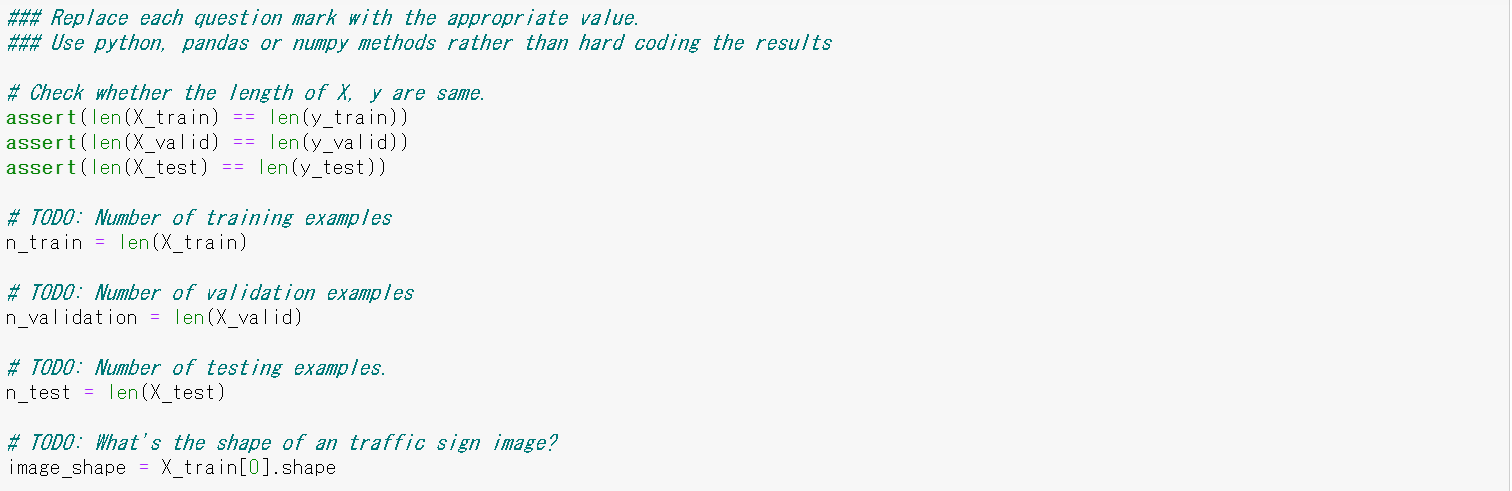
**Step 0: Load the Data**

Load the pre-uploaded datasets at the workspace.



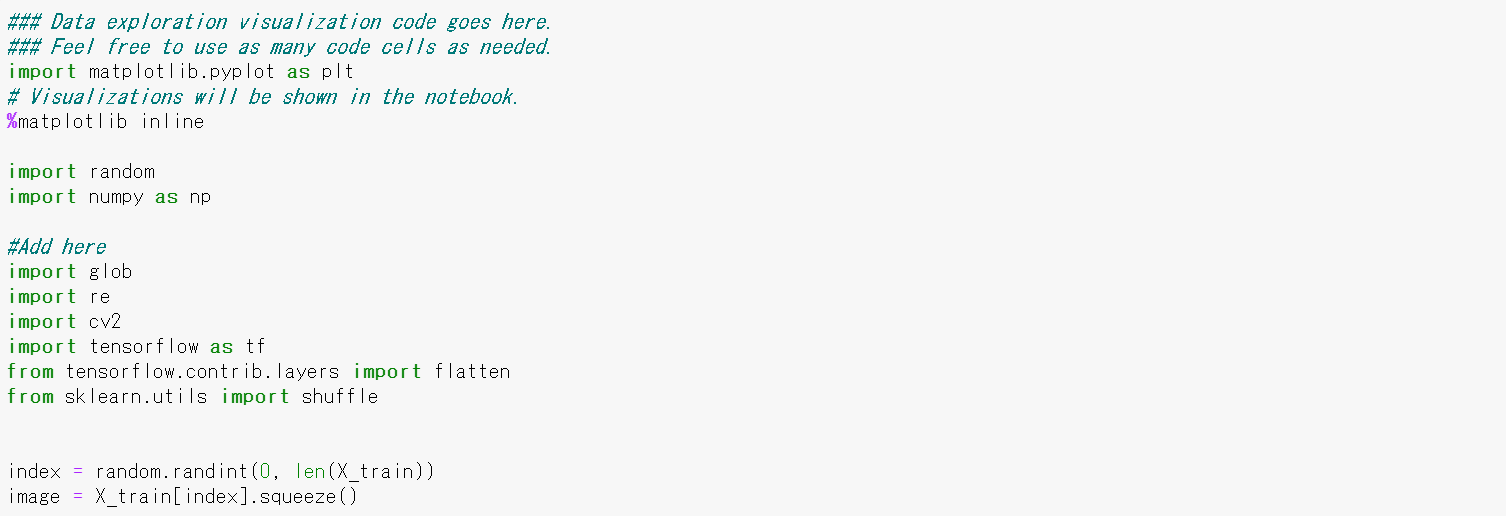
**Step 1: Dataset Summary & Exploration**

At first, I check the dimensions of each necessary data.





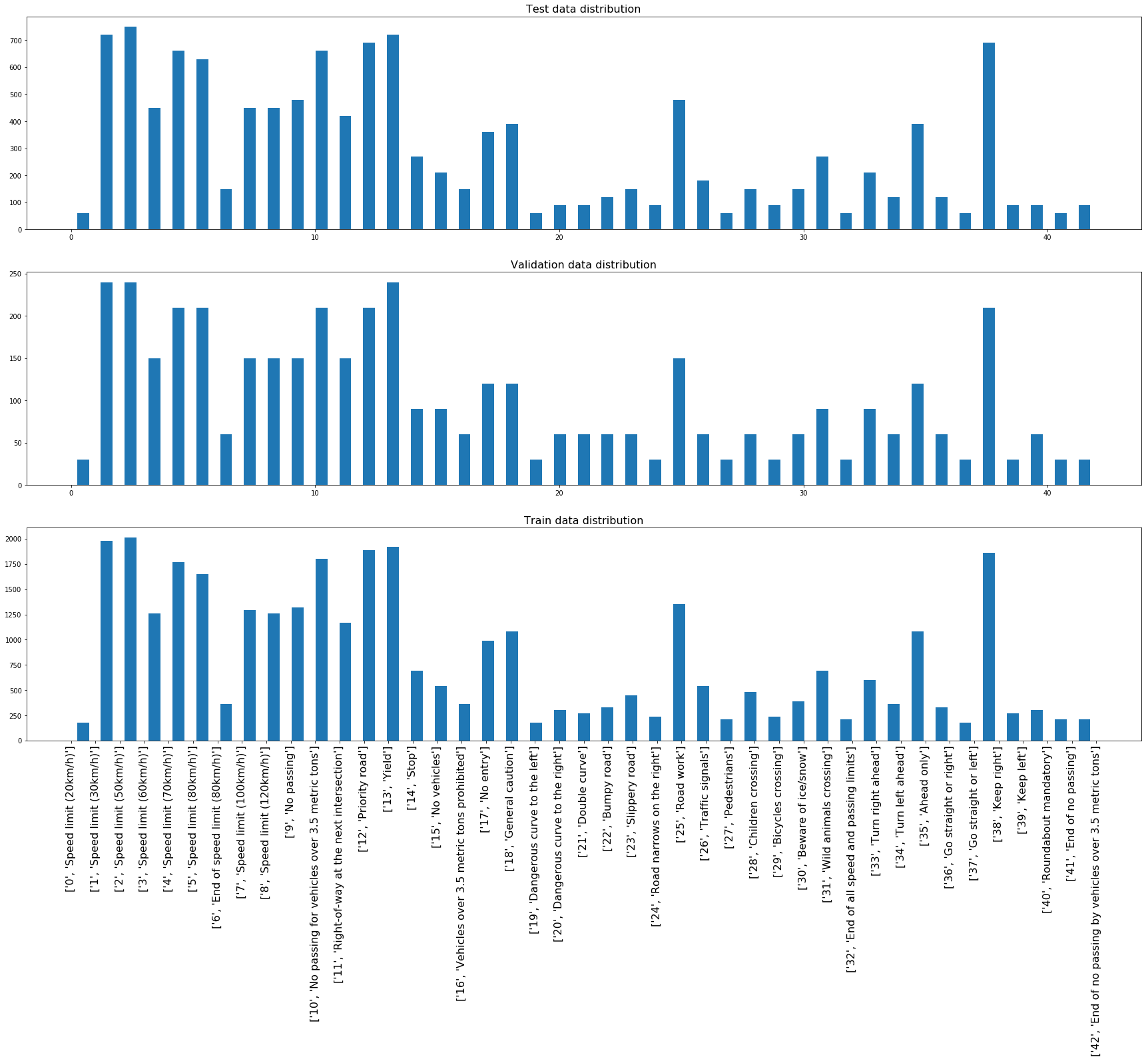
Next, I visualize the datasets with bar graphs.

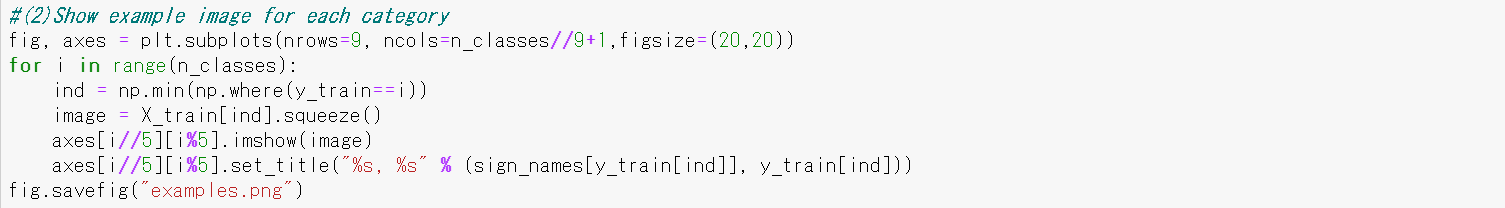




The distributions of Test Data, Validation Data and Tran Data are shown below.

There’s big disperse between each class (0 ~ 42), but there’s small disperse between each datasets,   
so it will be OK to proceed.





The following images are examples of Traffic Signs.



**Step 2: Design and Test a Model Architecture**

First in the Step2, I pre-process the Data Set.

I proceeded with the following steps:

[Check #1] I trained the model without any preprocess nor any change from the original model,

and the Validation Accuracy was just 89.2%. Target Accuracy is 93%, so it’s not good enough.

[Check #2] I made the images from color to gray scale, and normalized them.

Validation Accuracy became 92.5%, but it’s not good enough.

[Check #3] I added training data by rotating the original training data by +15/-15 degree.

The number of training data became triple from original, which is 104,397.

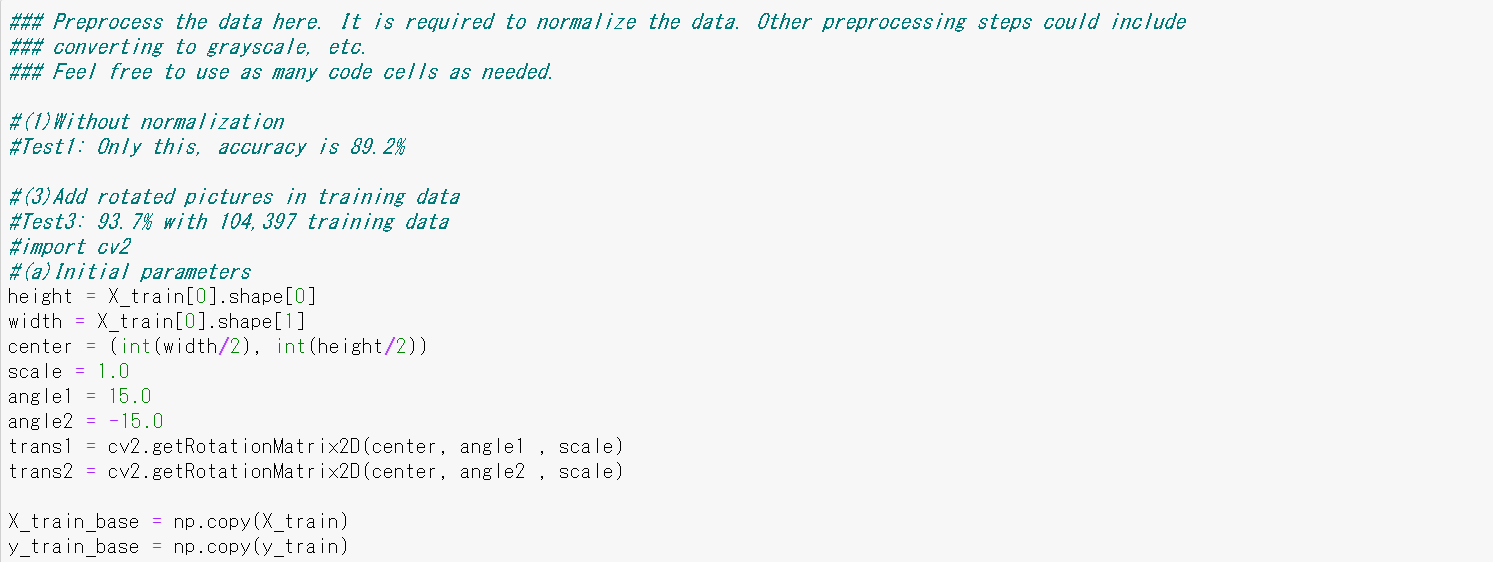
Validation Accuracy became 93.7%, but it’s better to make it more robust.

[Check #4] I added training data by zooming up/down the original training data by +2/-2 pixels.

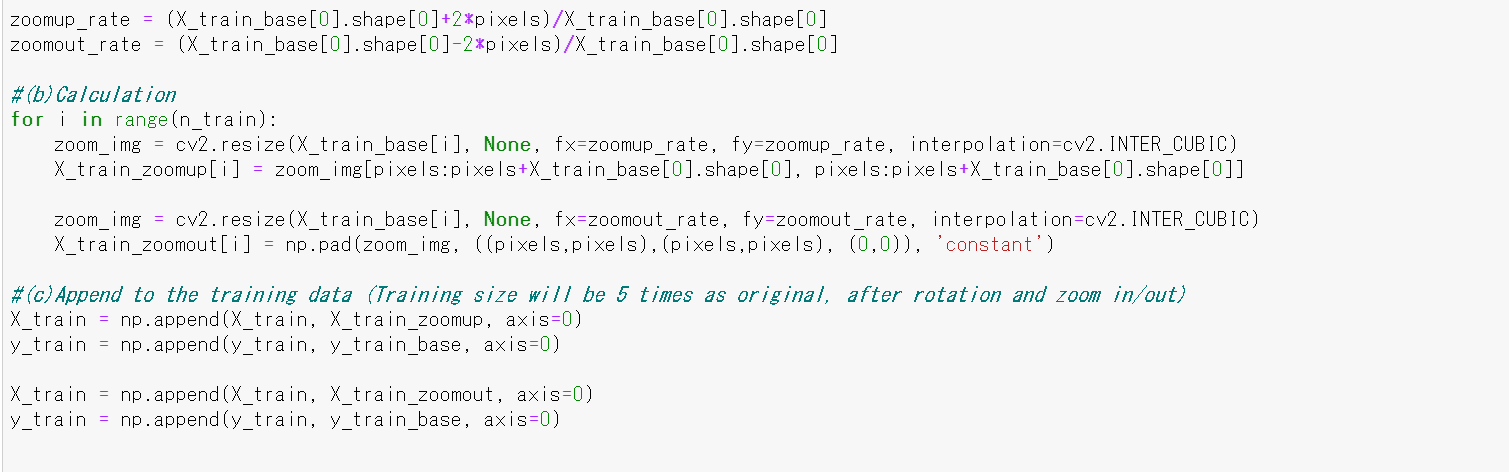
The number of training data became five times from original, which is 173,995.

Validation Accuracy became 94.3%, but it’s not good enough because Test Accuracy was 92.4%.

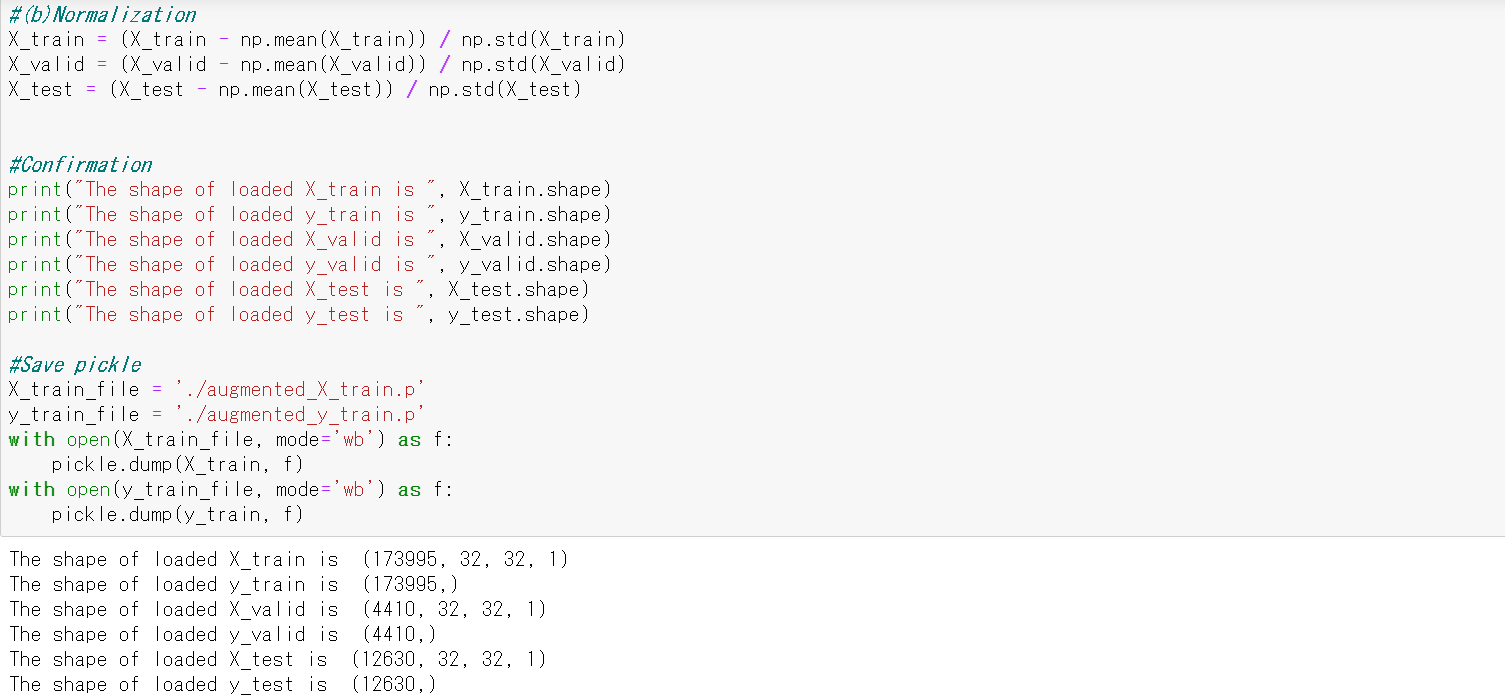
I found that it’s better to change the parameters of the model to get better results.







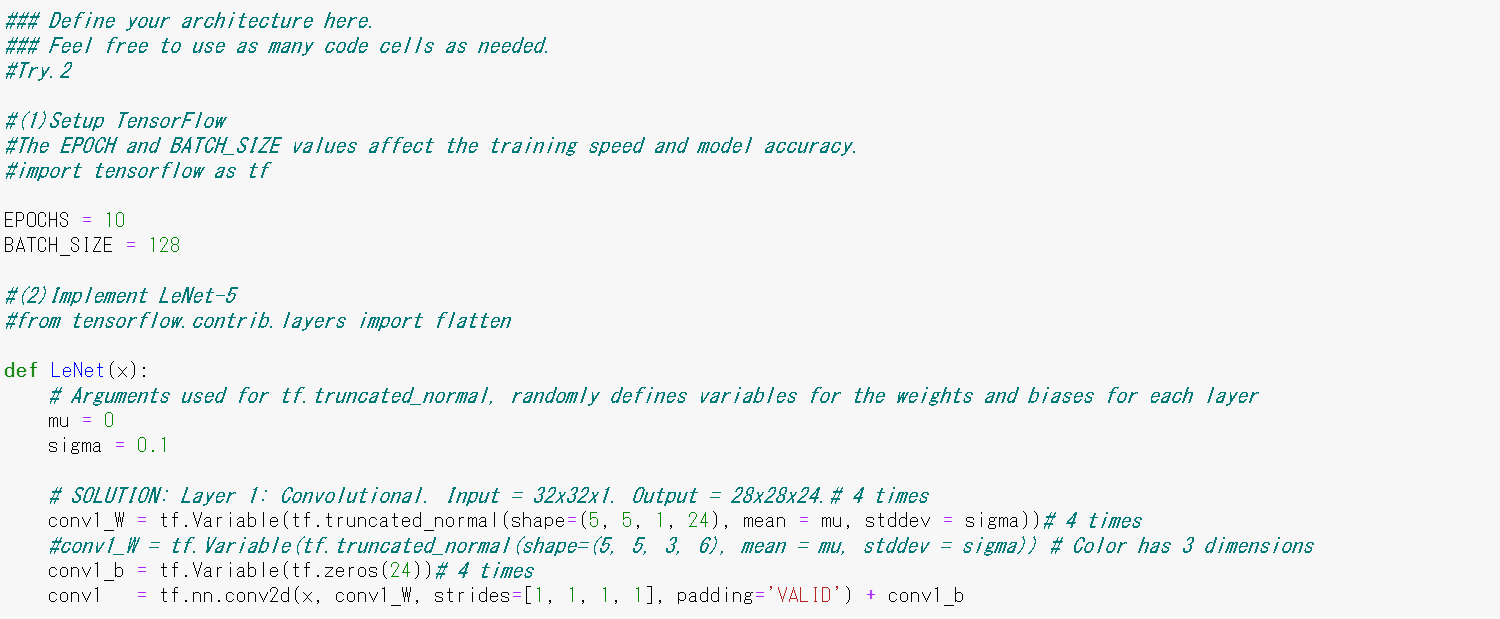


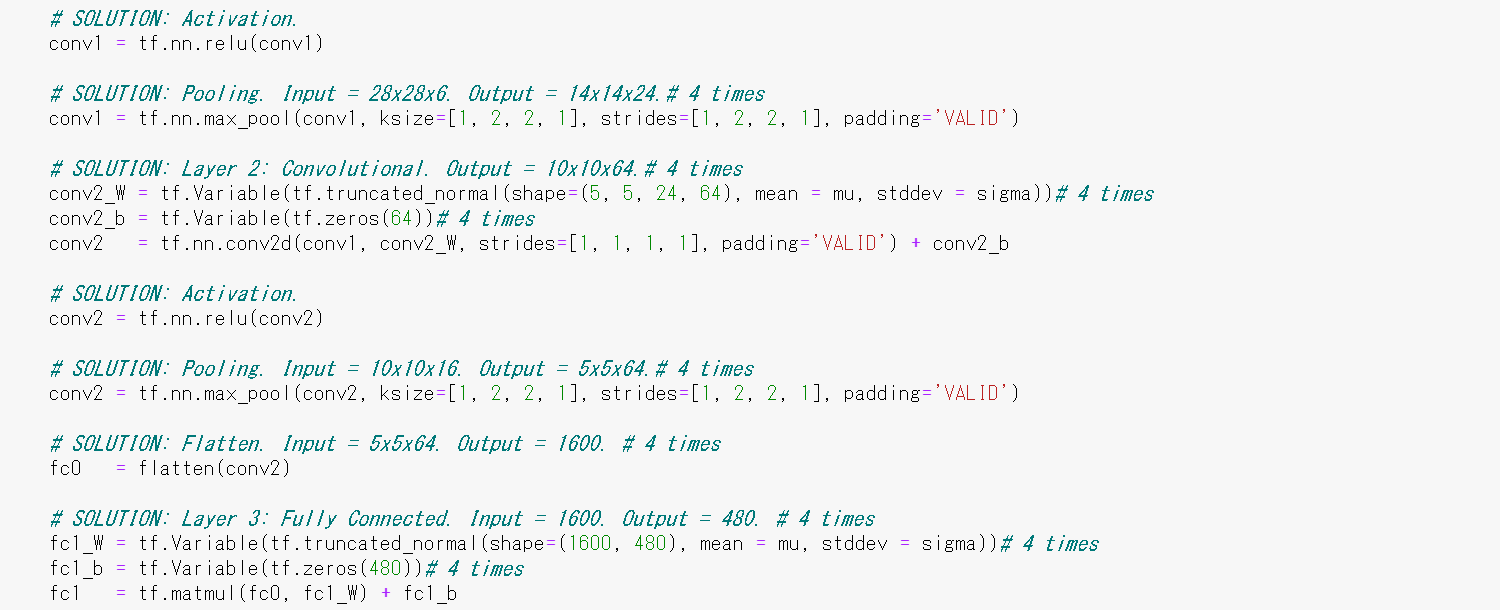


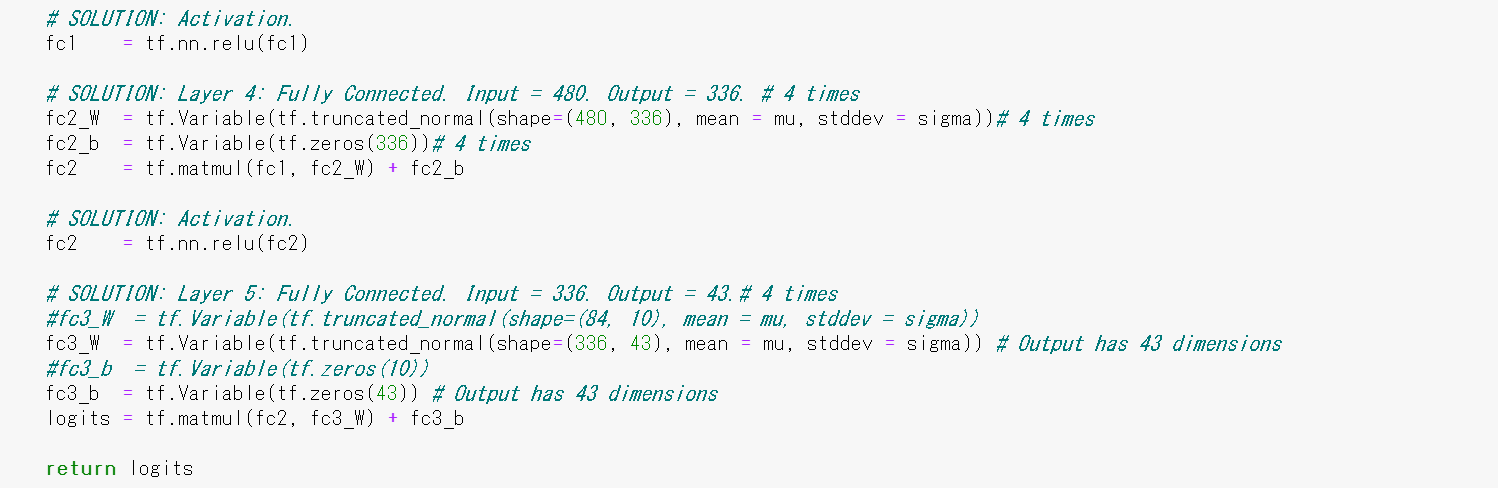
The following code shows the final model of this project.

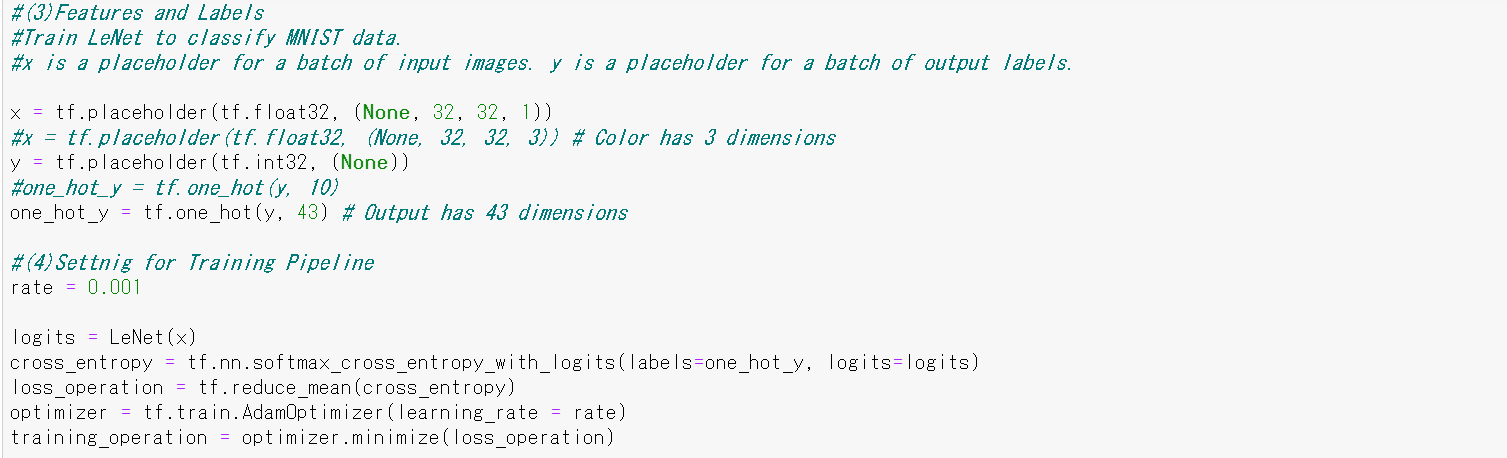
[Check #5] The original LeNet has output dimension: 10 (0, 1, …, 8, 9), but our project has output dimension 43.

It’s almost 4 times bigger than original one, so I made the depth of the hidden layers 4 times bigger   
than original.

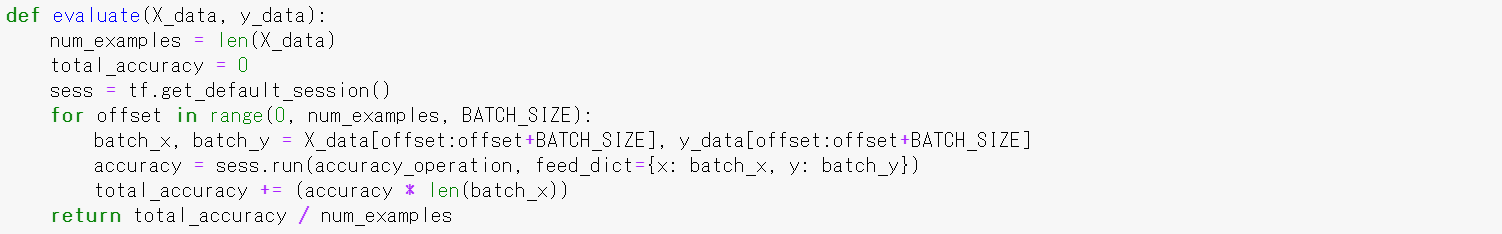


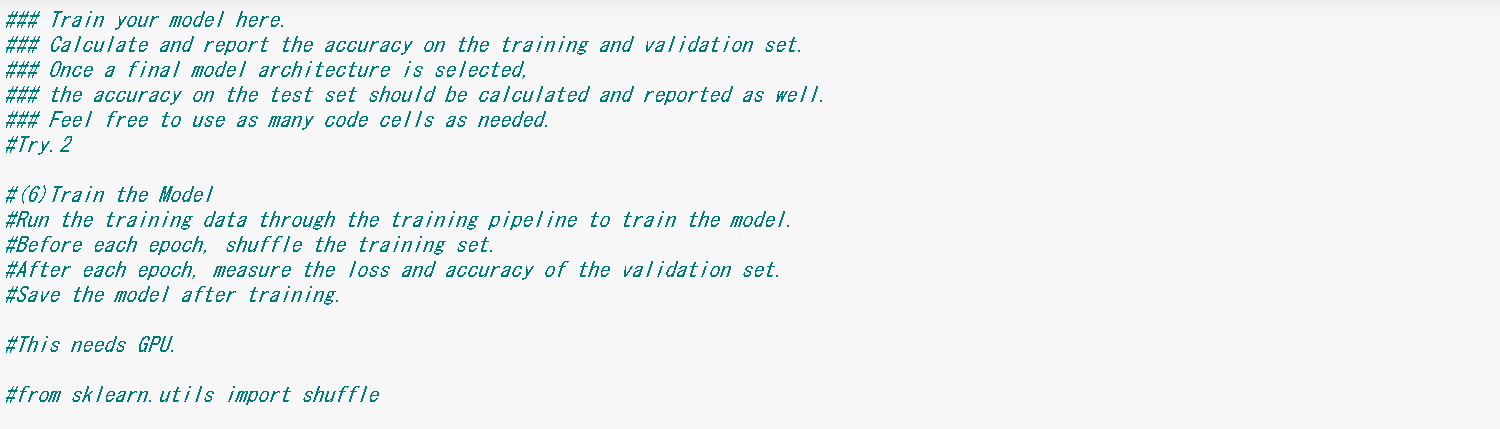


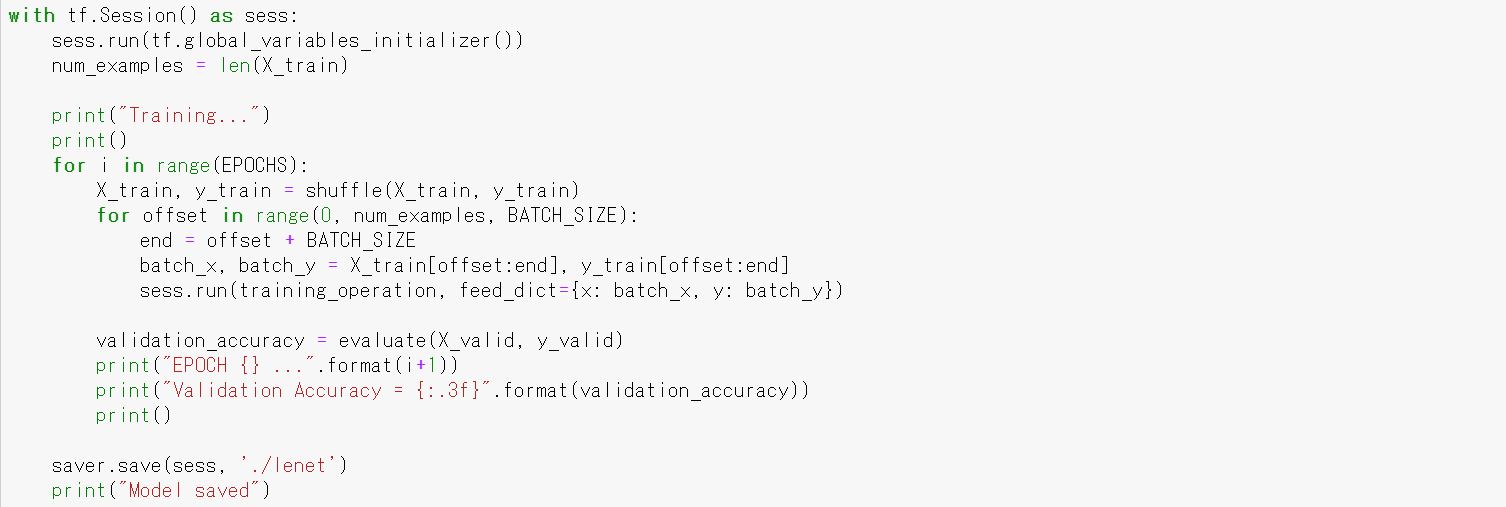


****

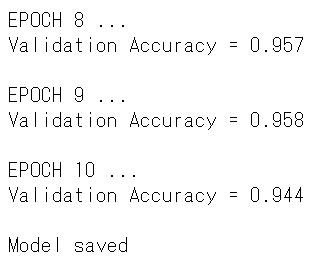
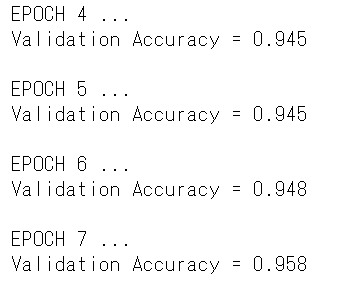
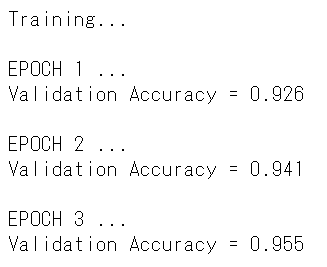
The source code of training is shown below.

****

****

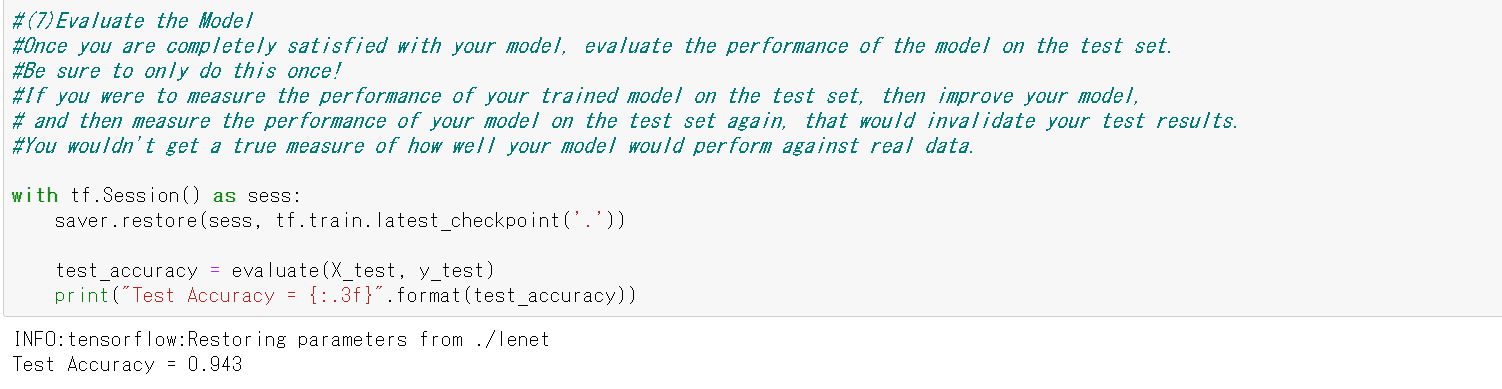
****

The results of Validation Accuracy is shown below. Final Validation Accuracy is **94.4%**, which is better than the Target Accuracy: 93%. It can have better result by Early Termination at **EPOCH 7~9**, but it may have better results with other Test Data, so I proceed by taking this trained model.

****

I evaluated the Model with Test Data.

Test Accuracy is 94.3%, so it passes the Target Accuracy.

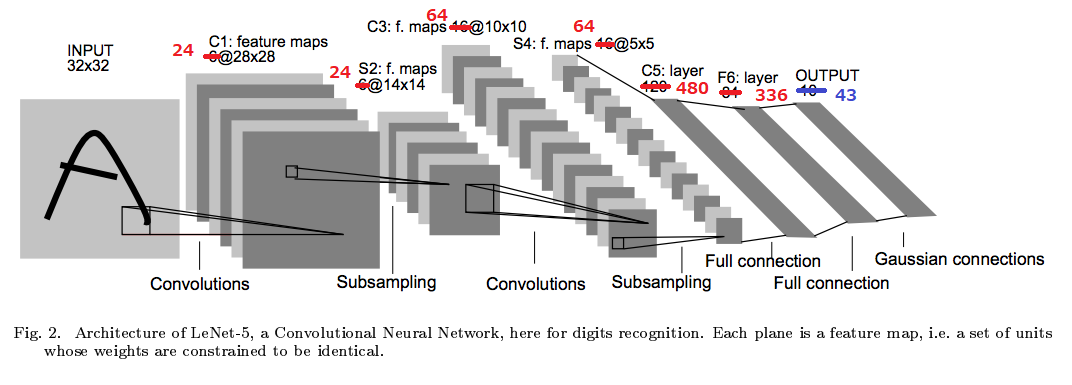
****

**<Additional page at 2nd submission, August 11th>**

I got 3 feedbacks at the 1st submission, so I added this and next page with 2nd submission.

**(1) Details of the characteristics and qualities of the architecture**

I describe my model architecture in detail as below. The original is LeNet-5, and I modified dimensions.



|  |  |
| --- | --- |
| **Layer** | **Description** |
| Input | 32x32x1 Grayscale image |
| C1: Convolution | 1x1 stride, valid padding, output = 28x28x24 |
| Activation function: RELU |  |
| S2: Max pooling | 2x2 stride, valid padding, output = 14x14x24 |
| C3: Convolution | 1x1 stride, valid padding, output = 10x10x64 |
| Activation function: RELU |  |
| S4: Max pooling | 2x2 stride, valid padding, output = 5x5x64 |
| C5: Fully Connected | output = 480 |
| Activation function: RELU |  |
| F6: Fully Connected | output = 336 |
| Activation function: RELU |  |
| Output: Softmax | output = 43 |

\*: For each Convolution and Fully connected, I used the same parameter: Mean=0, stddev=0.1

**(2) How the model was trained**

To train this model, I used the “**Adam Optimizer**”.

At first, I used the same parameters as the Lesson shown in below.

* Batch size: 128
* Number of epochs: 10
* Learning rate: 0.001

**(3) Approach to find a solution**

**(a) Choice of architecture**

As shown in the previous page, I chose LeNet-5 as the original model.

In the paper it was used to classify characters, but I though it can be used to classify Traffic Sign images by the following reasons:

* LeNet-5 is the basic Convolutional Neural Network, which was referred and arranged by many people to classify more complex images.
* It deals with input images with 32x32 size, which is good size for Traffic Sign images.

**(b) Approach and results**

To check which item works well, I ran training every time for each item.

|  |  |  |  |
| --- | --- | --- | --- |
| Check item | Accuracy | | Comments |
| Validation | Test |
| 1.Default | 89.2% | - | I used LeNet-5 for Traffic Sign images without any change at first, and Validation Accuracy is much less than 93%. |
| 2.Grayscale and  Normalization | 92.5% | - | When I saw the examples of Traffic Sign images in page 4, there are many images whose color are dark and difficult to identify only by color. On the contrary, shape of image looks important to classify each sign.  I applied Grayscale and Normalization to Traffic Sign images and accuracy improved a lot as I assumed.  However, it’s not good enough. |
| 3.Rotate  (+15/-15 degree) | 93.7% | - | Same as above, there are some images who lean toward left or right. I increased training data by rotating the original train images, and accuracy improved. |
| 4.Zoom  (+2/-2 pixels) | 94.3% | 92.4% | Same as above, there are some images whose size is bigger or smaller. I increased training data by zooming the original train images, and accuracy improved.  However, Test Accuracy is less than 93%, so I thought that there’s over fitting with training data and validation data.  I figured out that I need to change something with the Model to get the similar amount of Test Accuracy. |
| 5.Change depth  of hidden layers | **94.4%** | **94.3%** | Output classes of LeNet-5 is 10 (0, 1, …, 8, 9), but output classes of Traffic Sign images is 43. It’s almost 4 times bigger than LeNet-5, so I figured out that I need deeper data in the hidden layers.  I made depth 4 times bigger than LeNet-5 as shown in the previous page, then both Validation Accuracy / Test Accuracy achieved the target accuracy: more than 93%. |

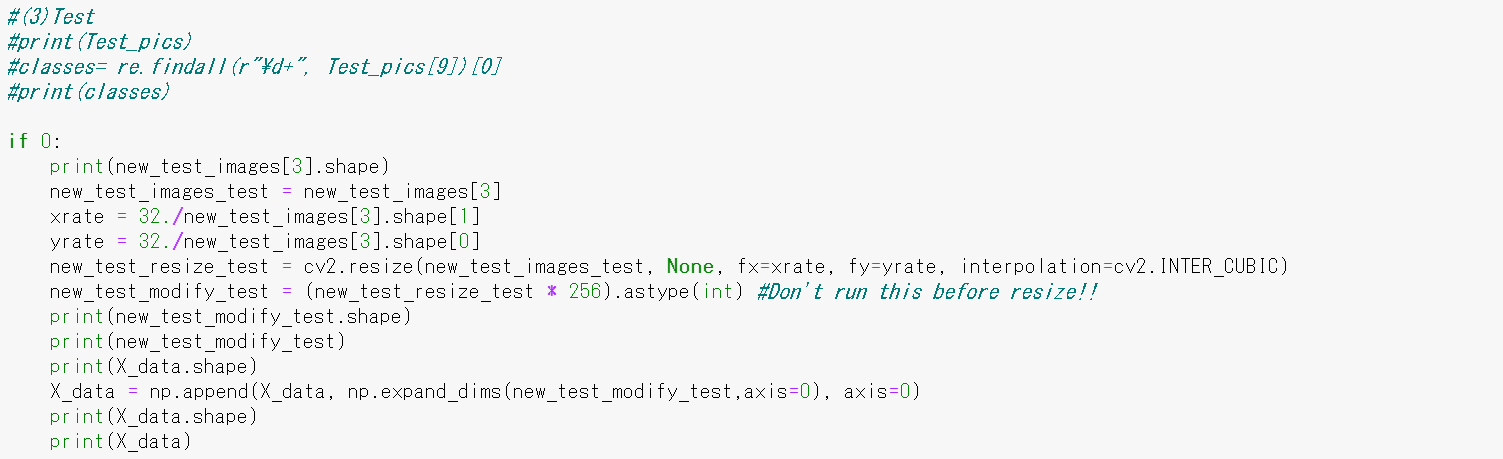
**Step 3: Test a Model on New Images**

I downloaded **10 new Test Images** by the following Website, and evaluated the Model with them.

<https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/version/1>

These images have different image size and data type is float, so I modified them into good data type with the Model. The size of Test Data by new images is **10(image numbers)x32x32(size)x3(RGB colors)**.









I chose 10 images. **Some of them are challenging for classification**.



Leaning

Blurred &

Shaded

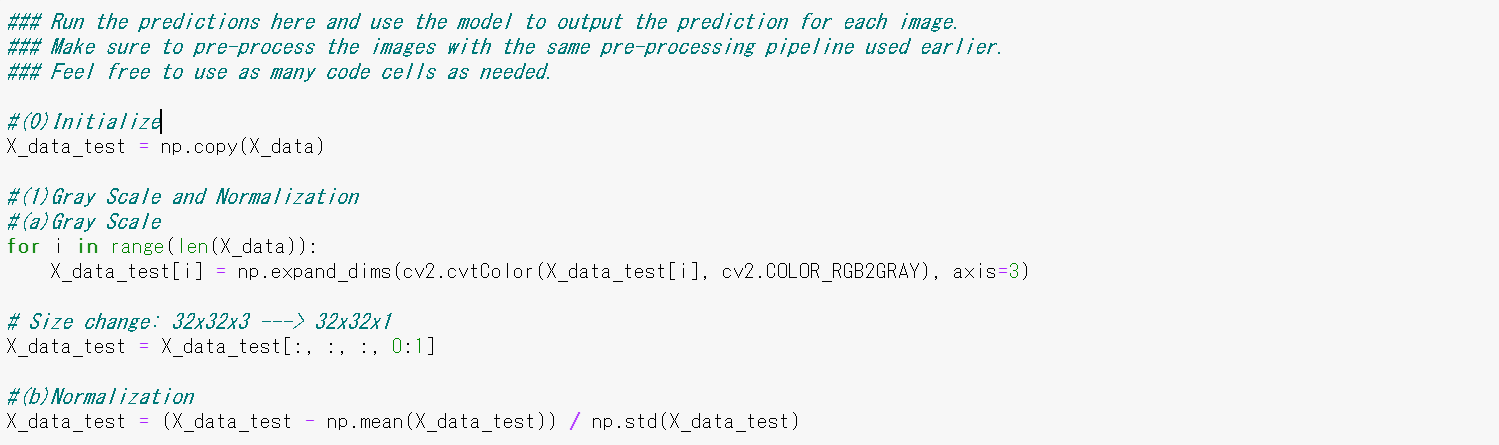
Blurred

Dark

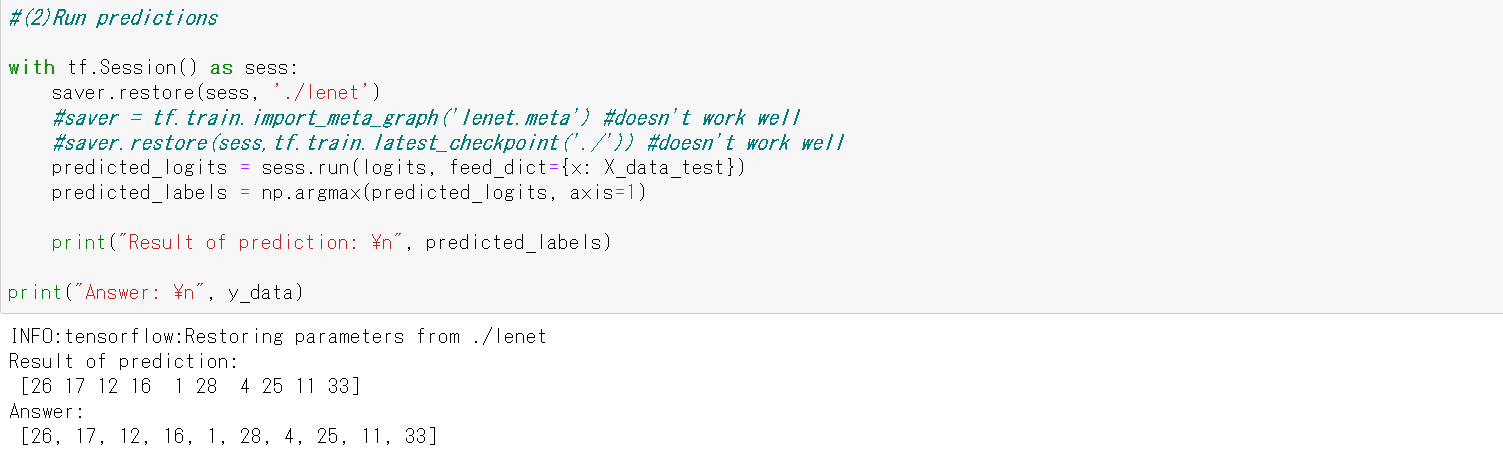
color

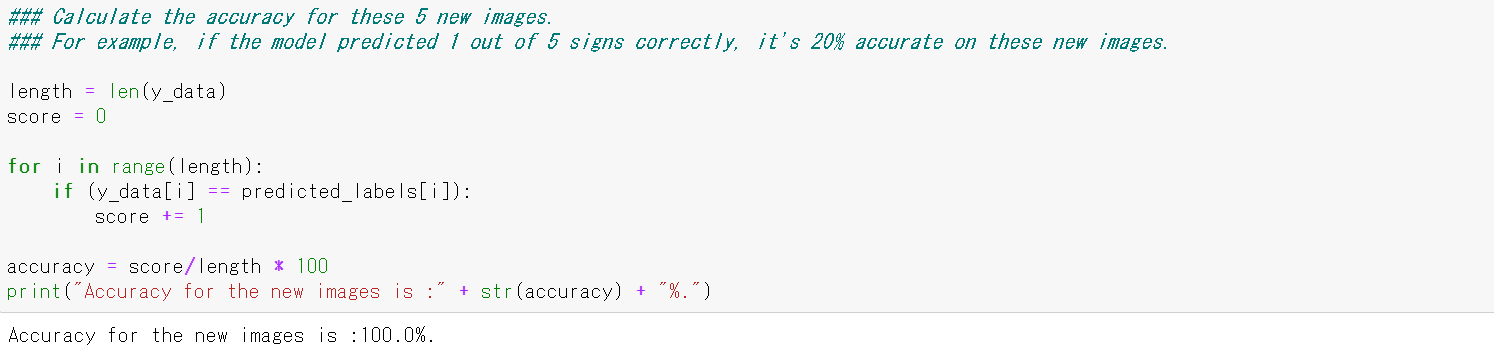
Crushed shape

I adapted Gray scale and Normalization to the new 10 images same as other image data, and I ran the prediction of the Sign Type for each image.

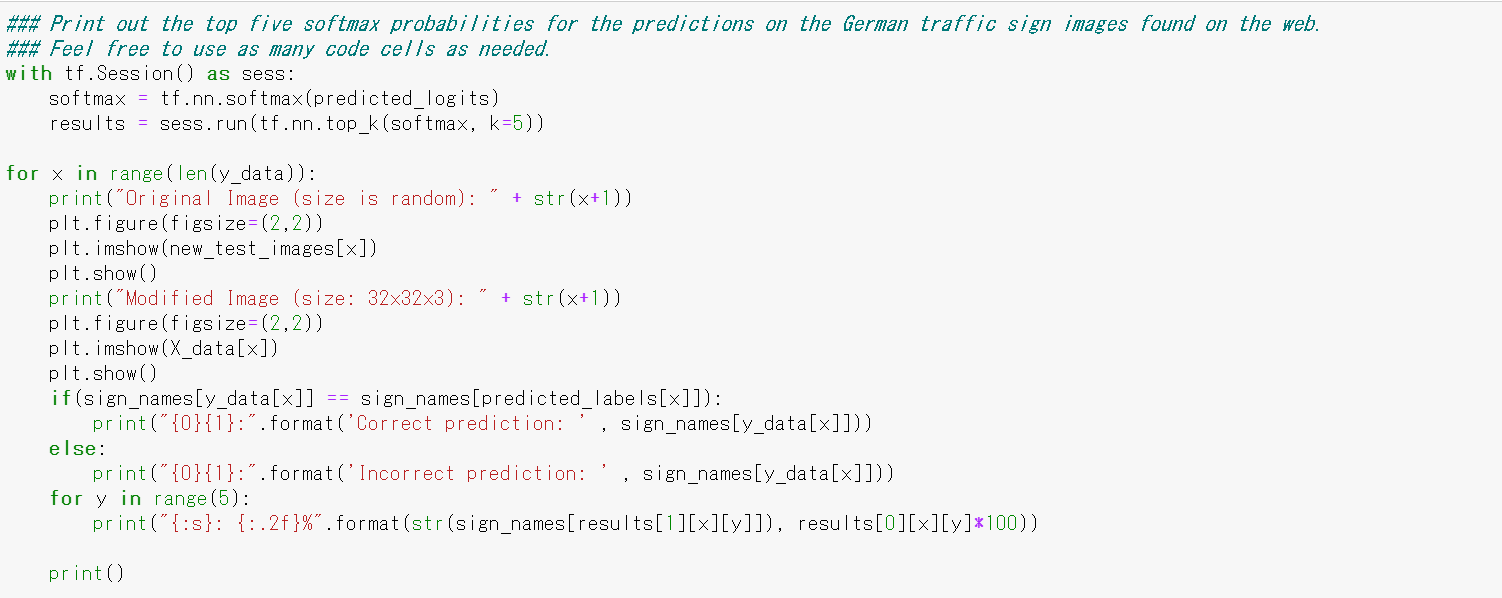


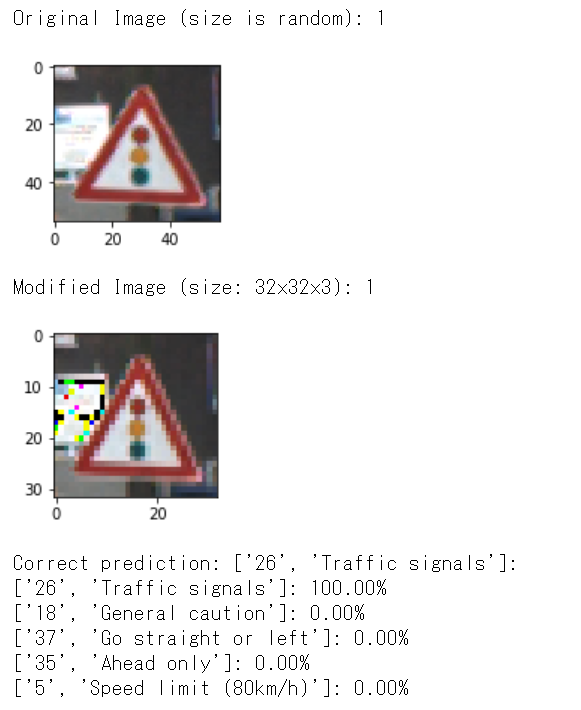
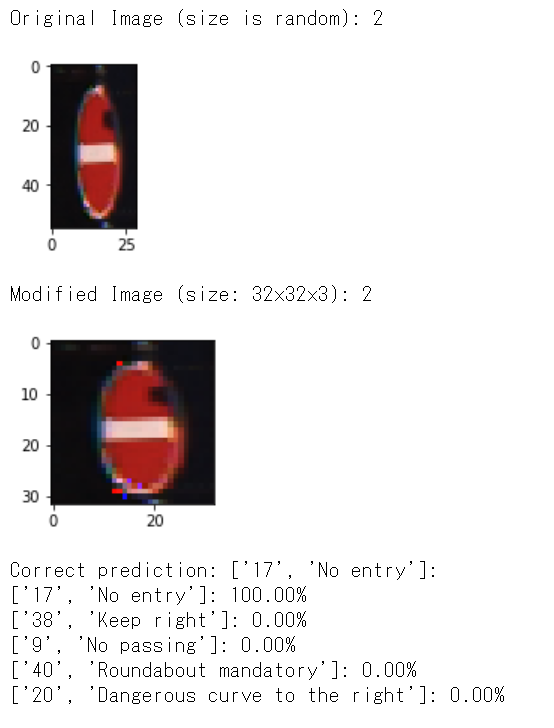
The result of prediction matched the labeled answer **100%**, so my Model is well trained for new images as well.

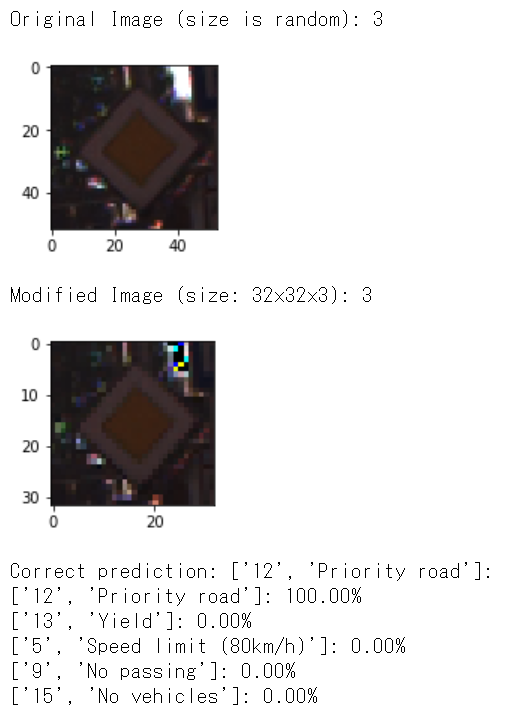
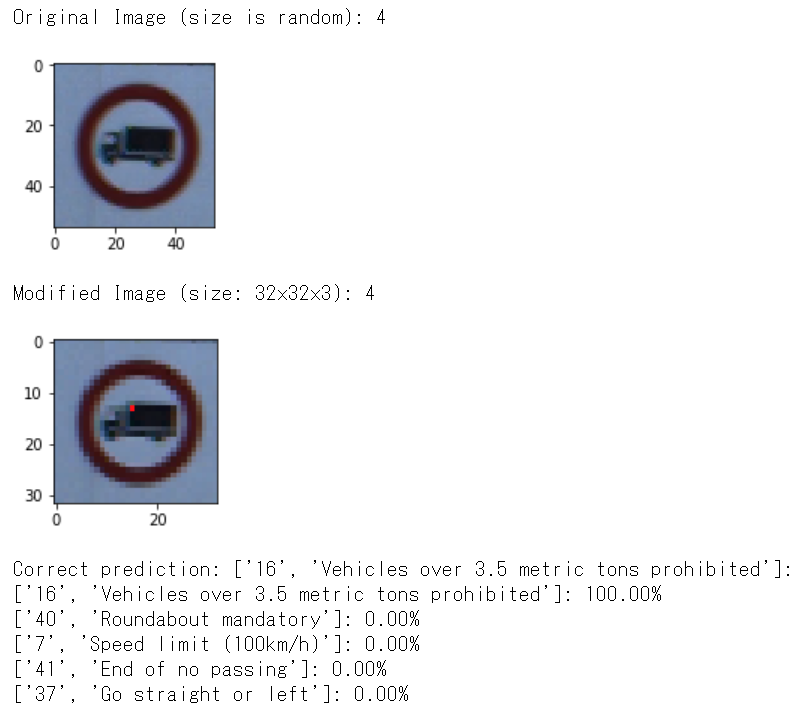


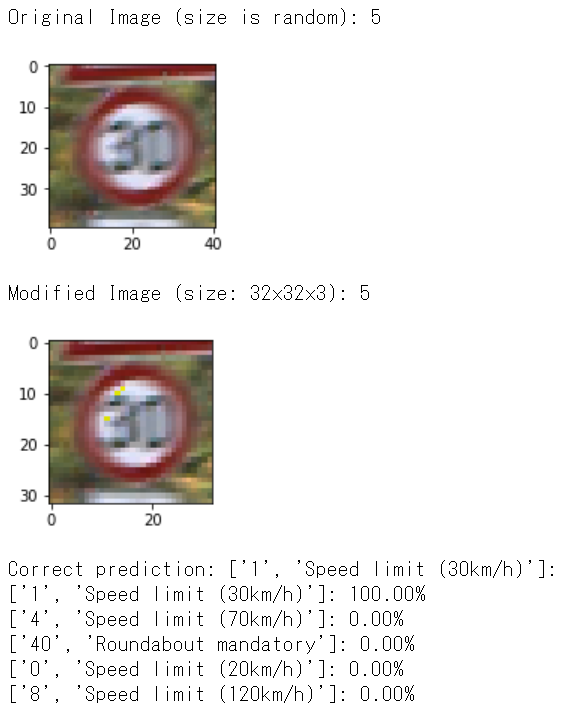
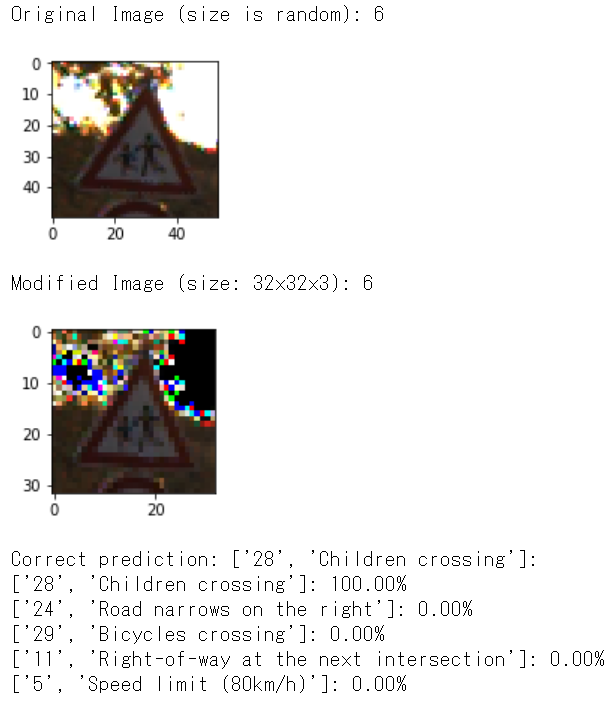


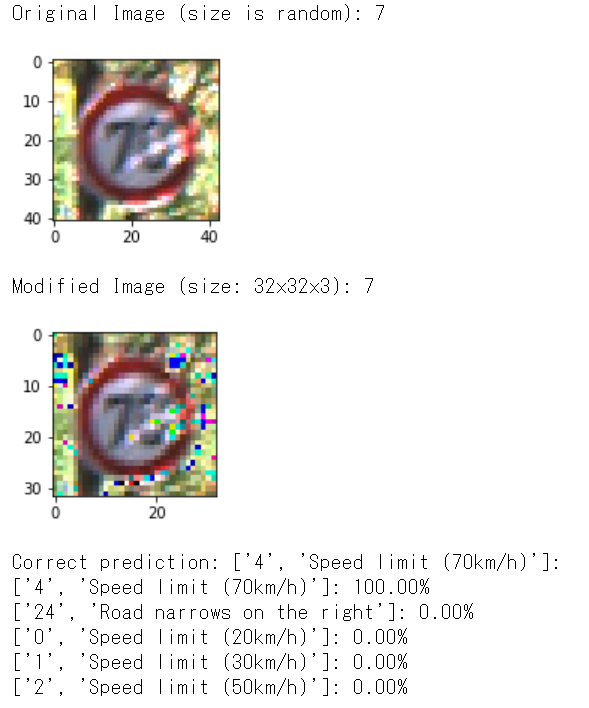
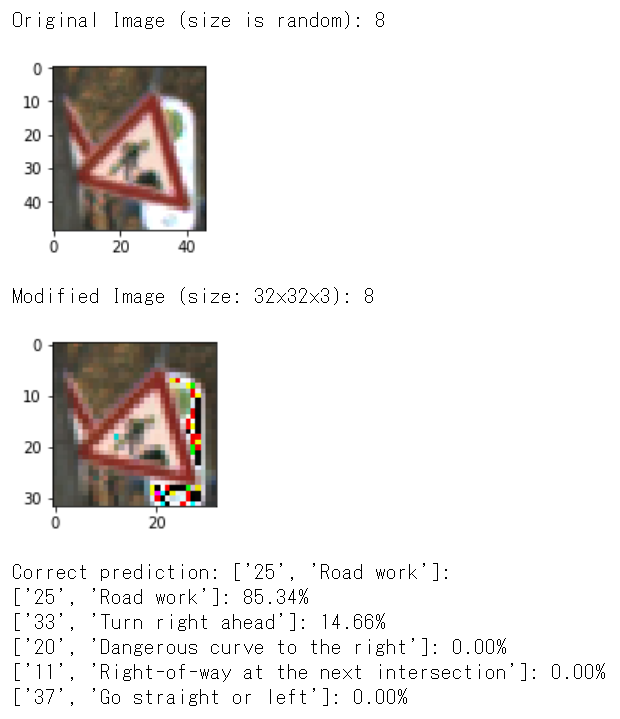
Finally, I show the Output Top 5 Softmax Probabilities for each image found on the Web site.

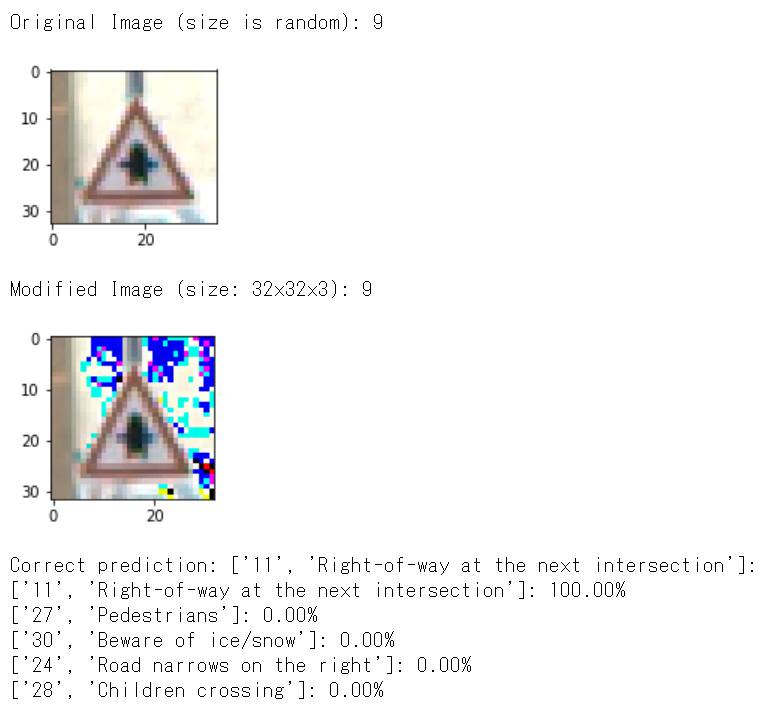
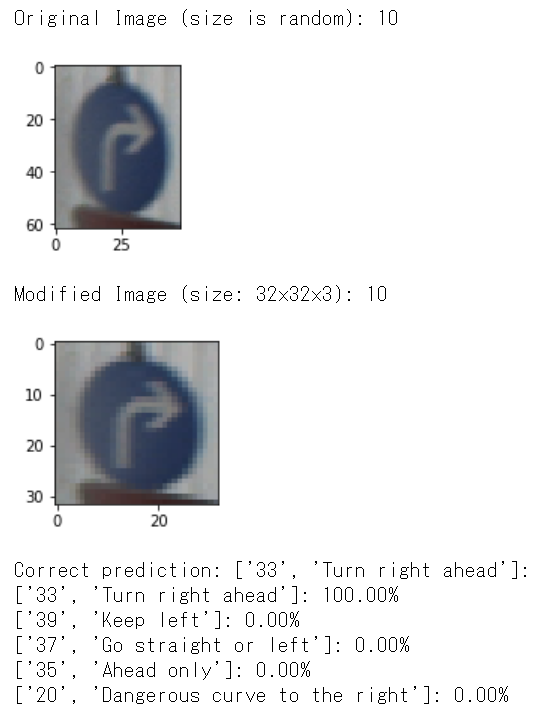
****

** **

** **

** **

**** ****

**** ****

**1st prediction is 85.3% and 2nd prediction is 14.7% with No.8**, but **mostly the 1st prediction is 100%.**