VE581 HW2 Report

Yu Cang 018370210001

2019-07-22

# Explore the dataset

The dataset placed into directory ‘./data’. Python package ‘pickle’ is used to import training set, validation set and testing set separately. There’re 34799 figures in the training set, 4410 figures in the validation set and 12630 figures in the testing set. Each figure within the dataset is a 32x32x3 RGB picture. In total, there’re 43 classes inside the dataset. The distribution of classes within each set is shown in Fig1. This part is done within section “**Summary of dataset**”.

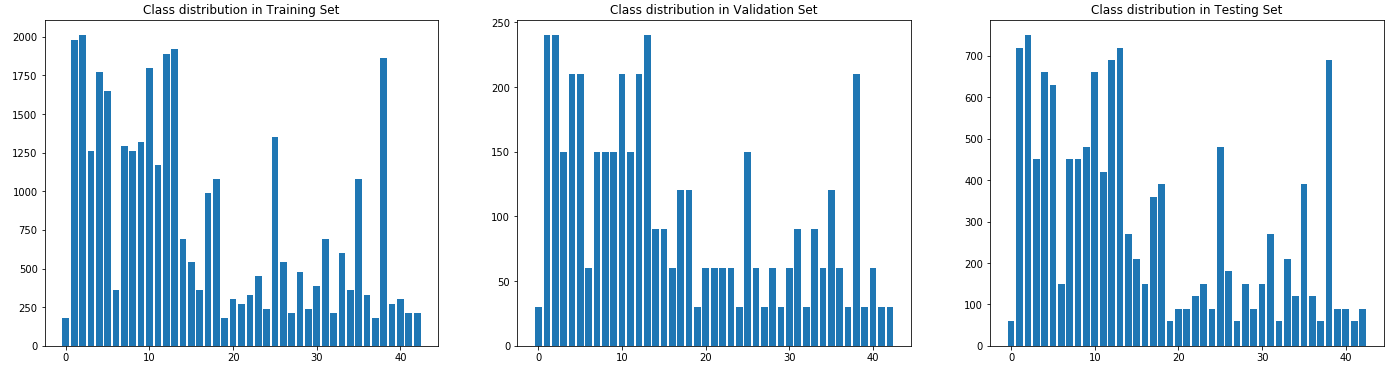


Fig 1. Distribution of classes within each set.

After loading the dataset, representative from each class is chosen randomly from the training set. The resulting gallery is shown in Fig2, and this part is done in section “**Exploration and Visualization**”.



Fig 2. Gallery of 43 representatives from each class.

# Design and Test on my classifier

## 2.1 Processing

It has been noticed that some of the figures are quite dark compared to others when exploring the dataset. Thus, a pre-processing procedure is necessary. In my code, the color converting strategy is taken.

As we want to add brightness to those dark figures, converting RGB figure into HSV space and tuning the ‘V’ channel is straightforward and useful. However, it should be noticed that not all figures need to be adjusted. A criterion is taken to judge if this figure it too dark or too bright. This is done by checking the maximum and minimum values in V-channel:

If the distance is less than 128, it’s grounded to believe that brightness in this figure is not well distributed. And it will be redistributed by taking a histogram equalization procedure. This part is done in section “**Process the dataset**”, where a helper function “*hist\_eq\_v*” is written to perform this task.

The effect of V-channel histogram equalization is shown in Fig9, where a typical image with index 2002 from the training set is chosen. It can be seen that the brightness is effectively improved, without introducing any other damages.

|  |  |
| --- | --- |
|  |  |
| Fig 9. Illustration of V-channel histogram equalization. Left is the figure before processing and right is the same figure after processing. | |

## 2.2 Model Architecture and Compiling

The architecture of my model is shown in Fig3.

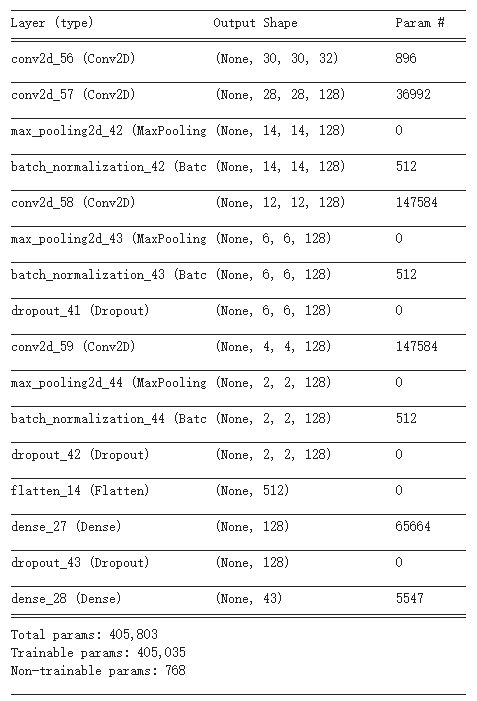


Fig 3. Model architecture.

As has been addressed in class, Adam optimizer is the first choice. The loss function is chosen as ‘sparse\_categorical\_crossentropy’, with ‘accuracy’ being the metrics.

## Training process

Batch size is set to 128, and 8 epochs are used. Validation set is also used in every epoch. The training process is recorded as Fig4.

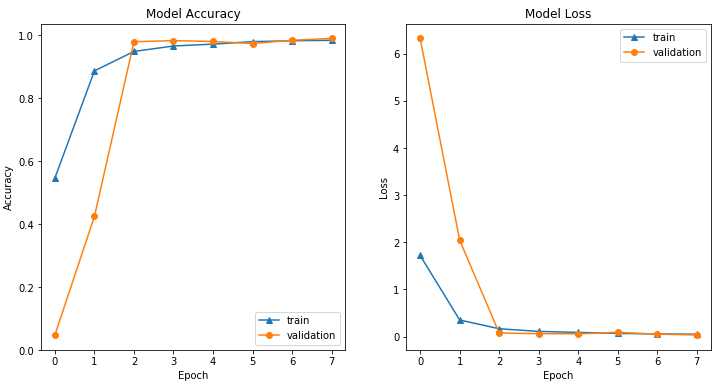


Fig 4. Accuracy and loss history of the model.

After tuning parameters, mainly depth in convolution layers, the accuracy on training set and validation set reach to 98.37% and 98.62% respectively. Details are shown in Fig5.

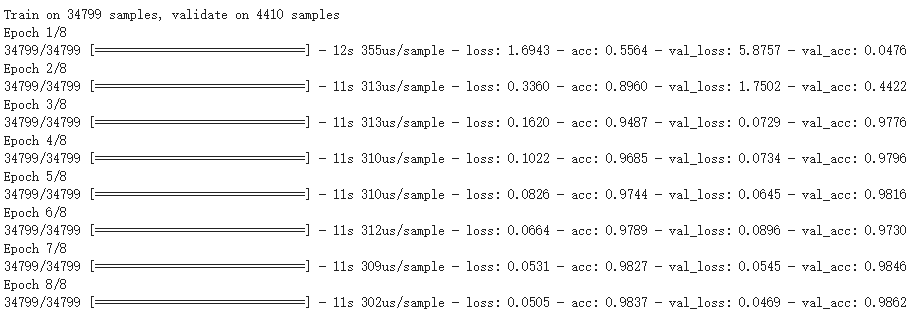


Fig 5. Details of accuracy and loss history

## 2.4 Evaluation and prediction on testing set

First, we evaluate the model using the testing set through KERAS API, which is quite convenient. The accuracy on testing set is 97.14%, which agrees well with results on training set and validation set. Related code and details are listed in Fig6.

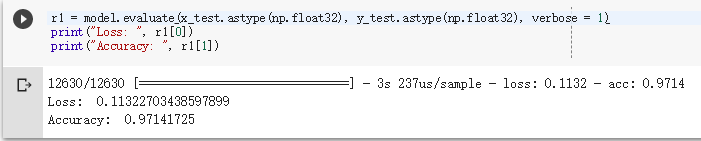


Fig 6. Evaluation code and detailed results.

Then, detailed analysis is performed with the help of confusion matrix, which illustrates the distribution of errors clearly. As shown in Fig7, it is observed that there’re clustering at the top-left of the matrix, which means the model is more likely to fail when identifying speed limits. This part is done in section “**Analyze testing accuracy**”.

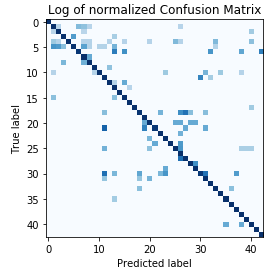


Fig 7. Confusion matrix on testing set.

Finally, we use this model to make predictions on 10 figures, which are randomly chosen from the testing set. Results are shown in Fig8. It can be seen that the model works quite well. This part is done in section “**Predicting on 10 figures randomly chosen from testing dataset**”.

Additionally, if the judgement is 0 for certain figure, the figure, as well as corresponding prediction and actual labels will be displayed.

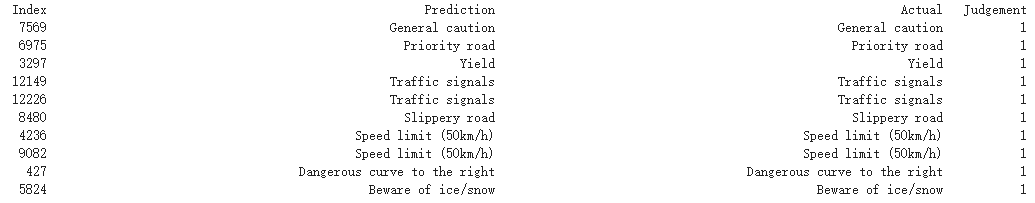


Fig 8. Prediction results on 10 random figures from testing set.

# Predictions on new images

## 3.1 Pre-processing

Images downloaded from Internet need to be adjusted to 32x32x3 in order to be used by the model. Python package “PIL” is used to resize these figures, which are shown in Fig10.

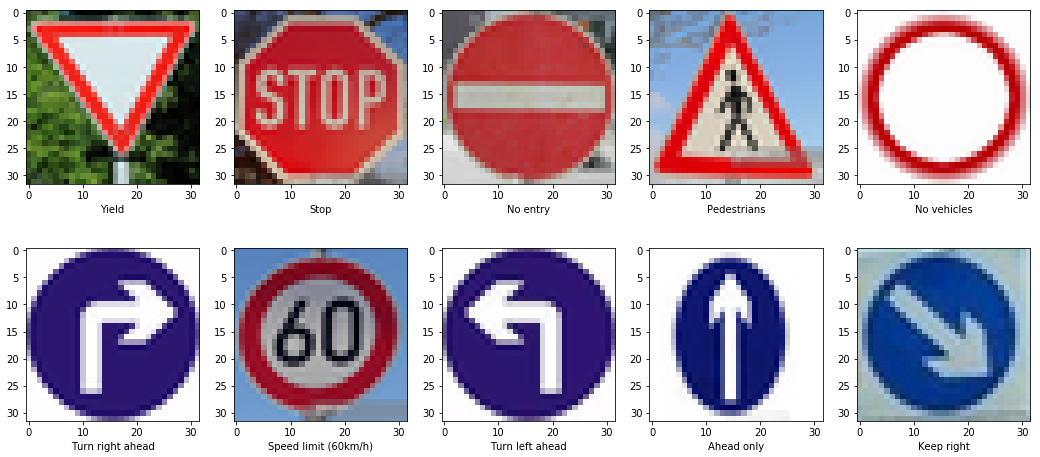


Fig 10. Resized 10 internet images used for prediction.

## 3.2 Predictions

Following similar procedure carried on testing set, the prediction on these 10 images is done in section “**Predict on downloaded images**”. The prediction results are shown in Fig11.

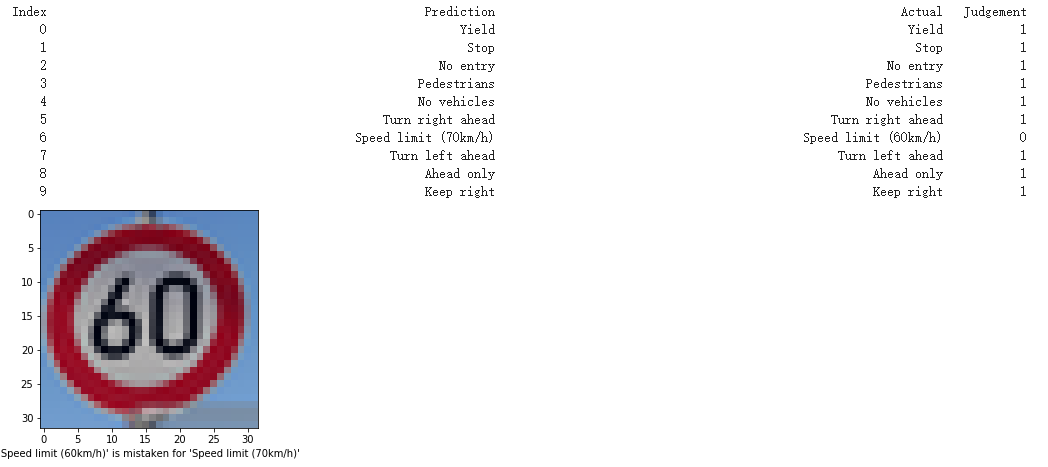


Fig 11. Prediction result on 10 downloaded figures.

It can be seen that the model fails on the 6-th figure, which is actually “Speed limit of 60 Km/h” but confused to be “Speed limit of 70 Km/h”. However, the overall performance is acceptable, and it indicates that the model can be generalized to real applications.

## 3.3 Softmax probabilities output

The prediction details of each figure can be obtained by calling ‘*model.predict*’, which returns probabilities of each class on given figure. Sorting all the probabilities in descending order, top 5 values of each figure can be obtained, and the result is shown in Fig12. This part is done in section “**Top 5 softmax probabilities**”.

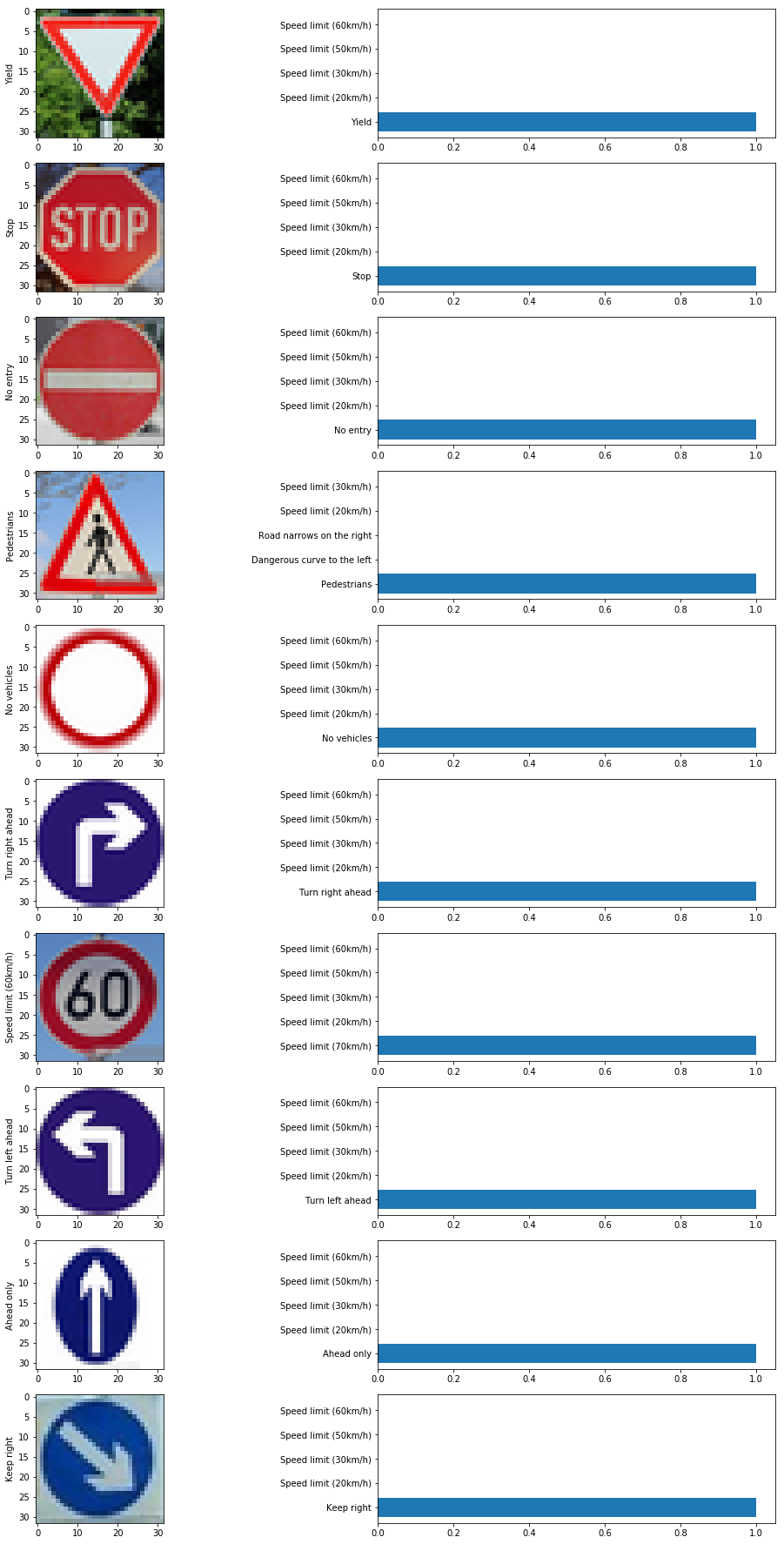


Fig 12. Top 5 softmax probabilities of each downloaded figure.