step 2

1.After spliting, the dataset becomes: training set - 99 images, validation set - 7 images, testing set - 27 images.

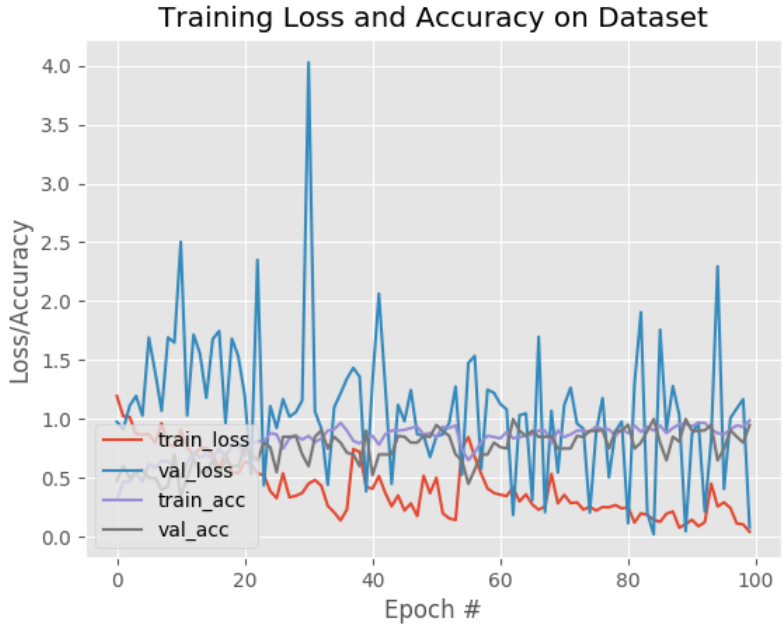
I used keras.preprocessing.image.ImageDataGenerator class to preprocess the data.
The source images are 'RGB' color format. They were changed to 'greyscale' color format for reducing training time. And the data value was rescaled to range 0 to 1.
The resolution was still 1024x768.

The training data was augmented when trained for avoiding overfitting. The images were randomly rotated, zoomed, flipped, or sheared. But the later experiments testified that this was useless. The validation and testing data were the original data without augmentation.

Other parameters:

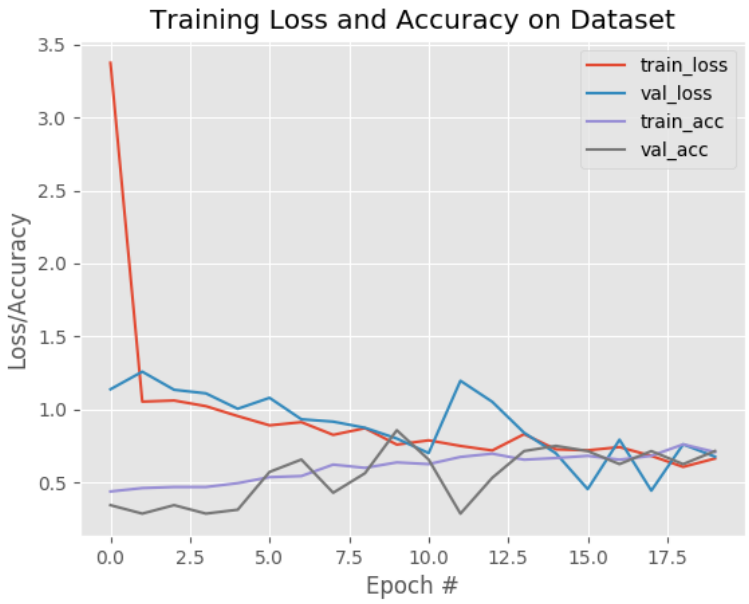
1. loss function: categorical_crossentropy;
2. optimizer: Adam;
3. activation: relu.

Then I got the following result



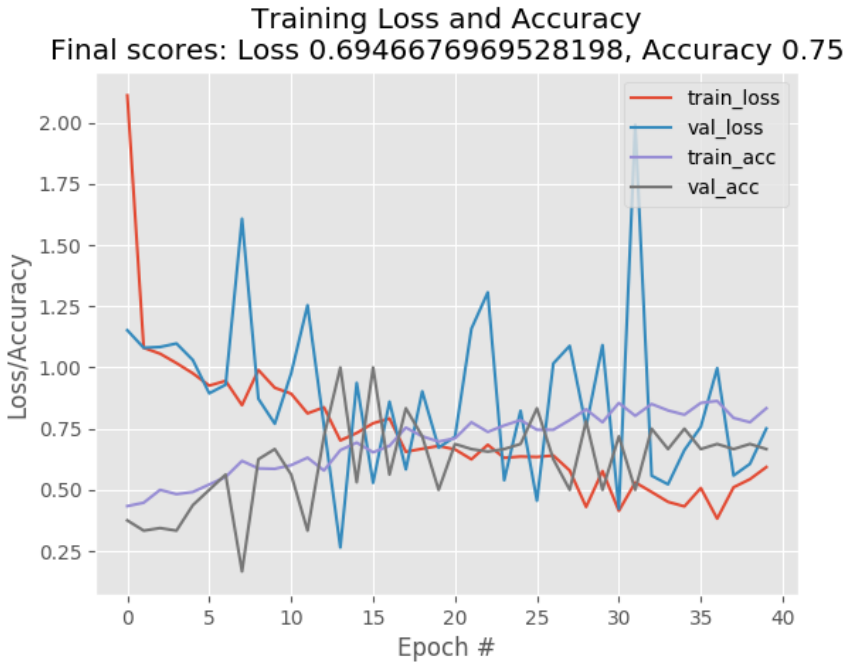
The accuracy was high at the end. But the validation loss was not stable. I thought the reason was that validation dataset was not enough. So I generated more data using my upsample.py script. The dataset were expanded by ImageDataGenerator. Then the data were randomly split into training, testing and validation set (99=>595, 27=>159, 7=>39).

However, the training speed became too slow. So I decreased the dataset size. (training: 299, testing: 159, validation: 39). For accelerating, the epochs were set to 20. The result seemed not too bad as following. I guessed that the result will become better by more epochs.

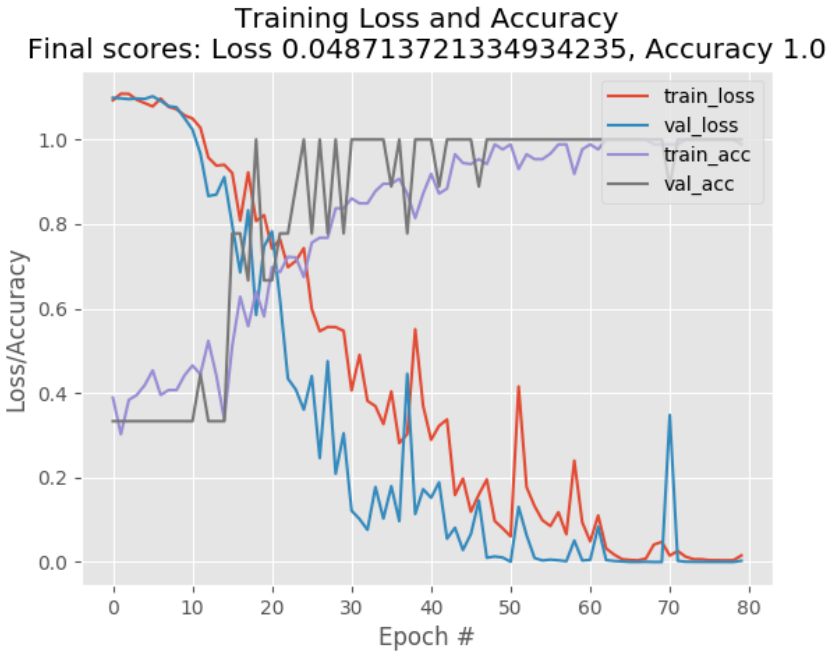


But training high resolution images really cost a lot of time.

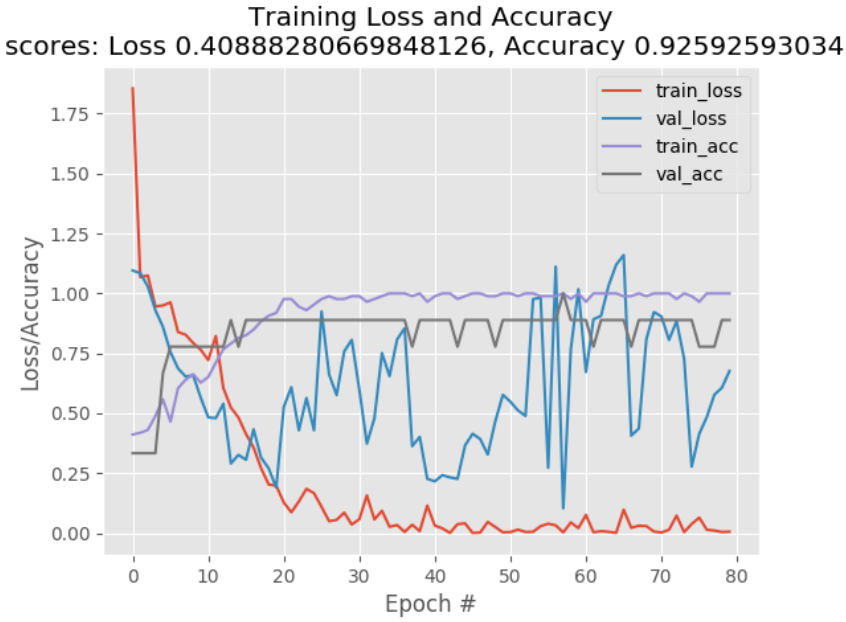
2. For accelerating, I decided to reduce the resolution of images on training to 720x540. The result was as following. But it still cost too much time to train.



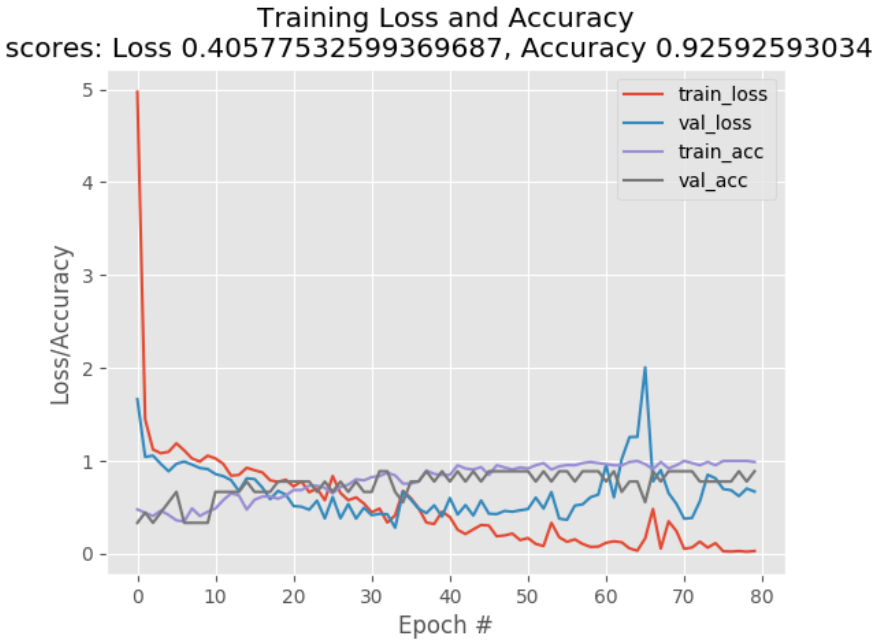
3. Then I changed the resolution of the images to 120x80. And deleted all the generated images. So this time I just used the original data to train and test without any data augmentation.
Resolution: 120x80, Training set: 95, Testing set:29, Validation set: 9.
Following is the result. The performance became very good.



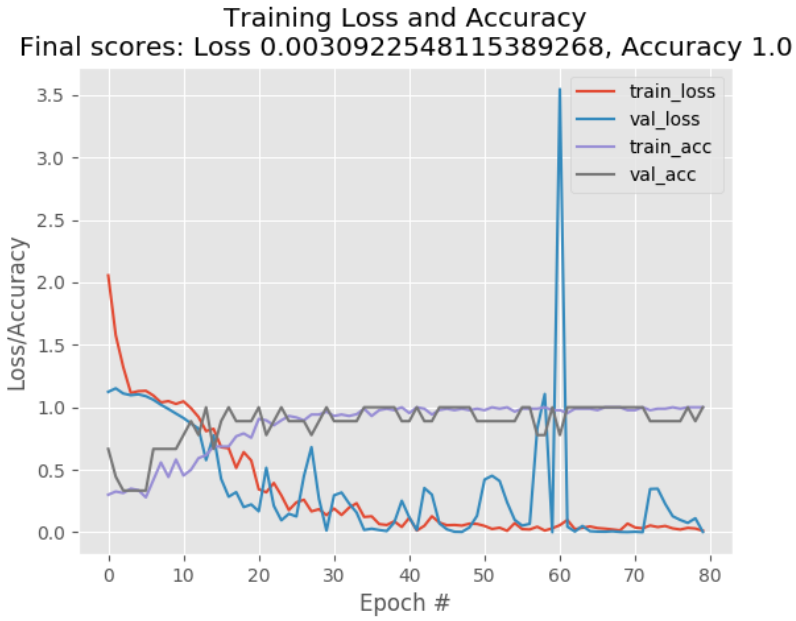
4. Without max-pooling.
resolution : 120x80.
The absence of max-pooling layers was worse than the presence of max-pooling layers. And the val loss became unstable. And the validation accuracy could not reach 1.0. It was interesting. So the max-pooling was very necessary.



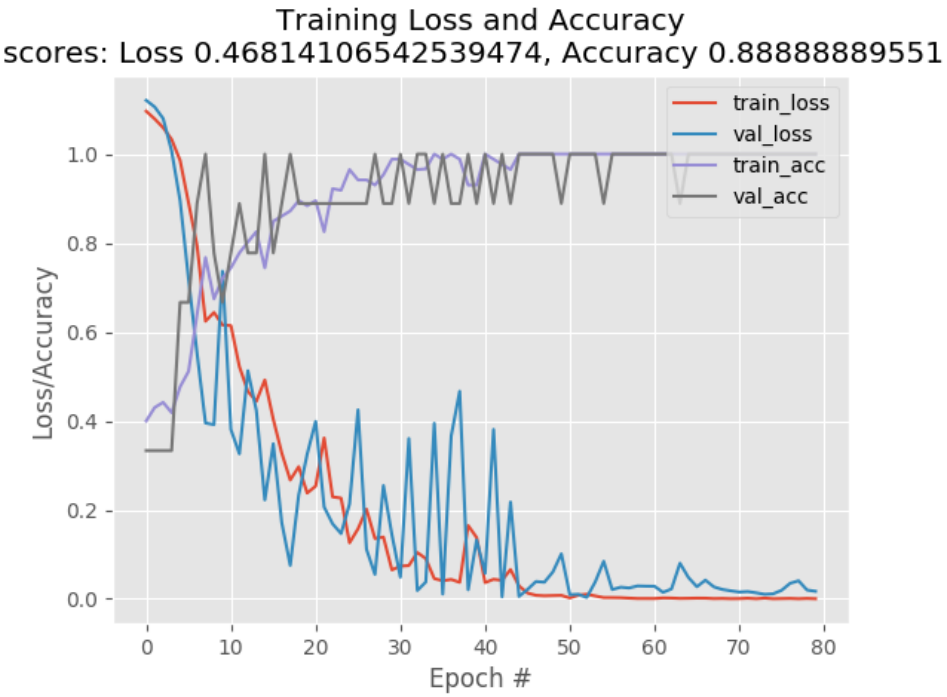
5. Without normalization
The result without normalization was worse than the result with normalization. And the validation loss started to increase after 55 epochs. But the testing result was good. It was really strange. The normalization was necessary.



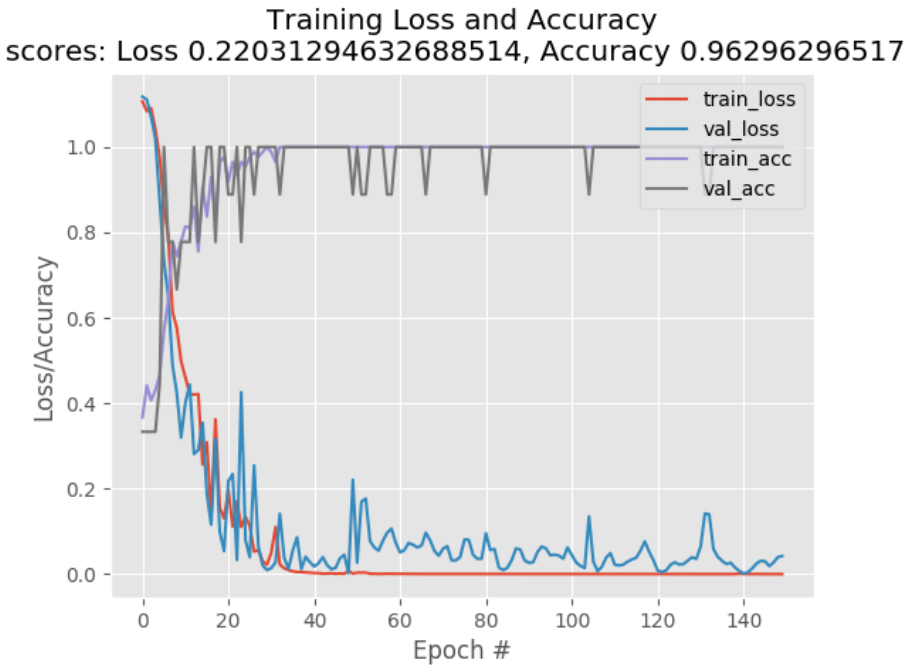
6. Less convolutional layers.
There was a pulse around 60th epoch. The validation loss was not stable. The result was worse than the result of 3. So I thought enough convolutional layers were necessary. But the testing result was 1.0. It was perfect.



7. Less general neural network layers.
This model must be very good. The validation loss and accuracy became stable after 50 epochs. The result was nearly perfect. So I thought more neural network in CNN model was not necessary.



8. Finally I got my perfect model.



Summary: In this problem, first, we must reduce the resolution of the images data for accelerating and high performance. Then the data augmentation was not necessary in this problem. Enough convolutional layers, max-pooling layer, and data normalization were good for better performance. Too many general neural network layers were not necessary.

step 1.
The last example:
Test loss: 0.02713118914191291
Test accuracy: 0.9915

Image with low resolution would likely get better performance.