NYU TANDON SCHOOL OF ENGINEERING

Cats and dogs classification by CNN

Siyun Wang (sw3569)

Tong Liu (tl2142)

12/13/2017

New York University

NYU | TANDON SCHOOL OF ENGINEERING

Outline

Introduction

- **Goal:**
  Using convolutional neural network to recognize cats and dogs.

- **Data set:**
  We use dogs-vs-cats dataset, which is a very famous dataset offered by Kaggle. We downloaded all 25000 images in training dataset, and removed 500 images from cats and 500 images from dogs to validation dataset.

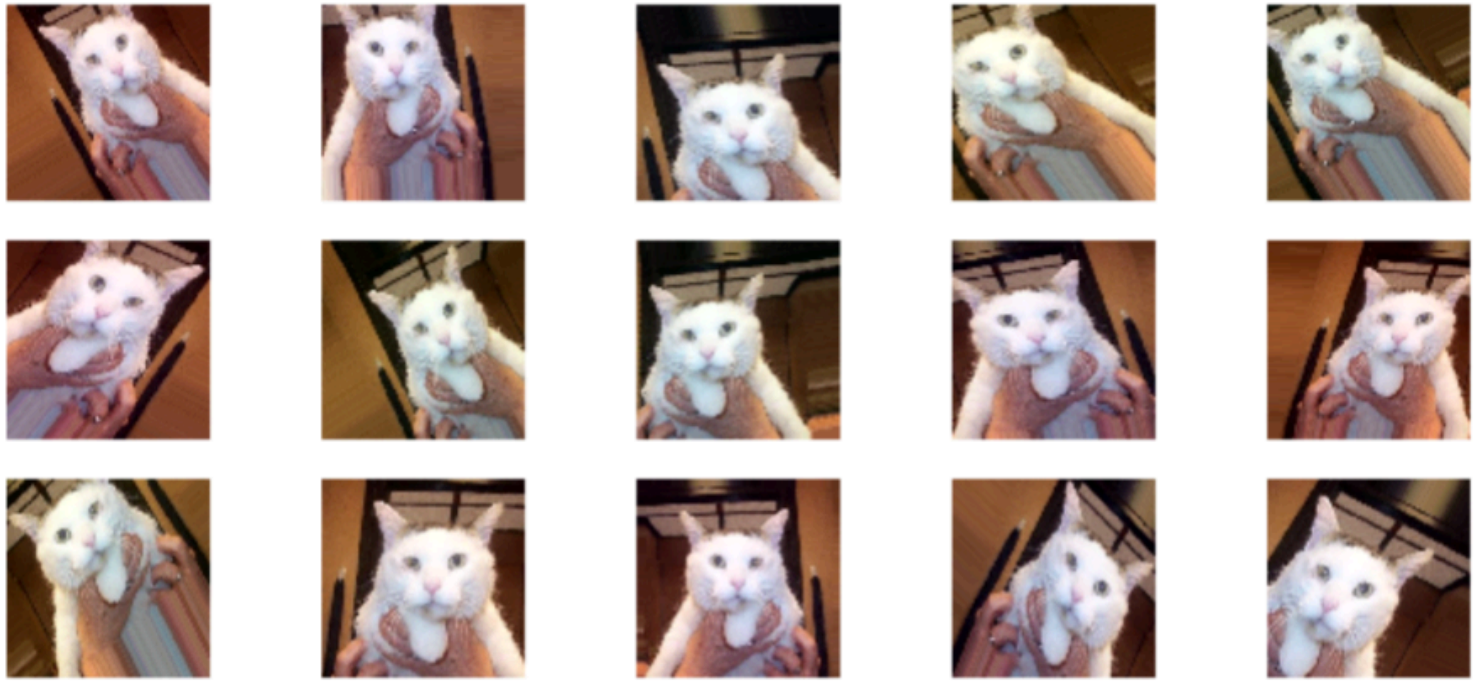
- **Model**
  Our project is based on the pre-trained network "Residual Network", and we modified the network so that the network could fit our project. ResNet50 is a built-in network in Keras, it contains 54 parameterized layers(53 convolutional + 1 fully connected for the softmax)

Part I : Data Processing

1. Downloaded the whole training dataset.
2. In this project, we used ImageDataGenerator to load images. In order to satisfy the requirement of this method, we separated the dataset into train/cats and train/dogs.
3. Then we created the validation dataset by randomly taking 500 images from train/cats and 500 images from train/dogs.
4. In order to know how does our data look like, we plotted one of the images. We loaded this image by load_img function.

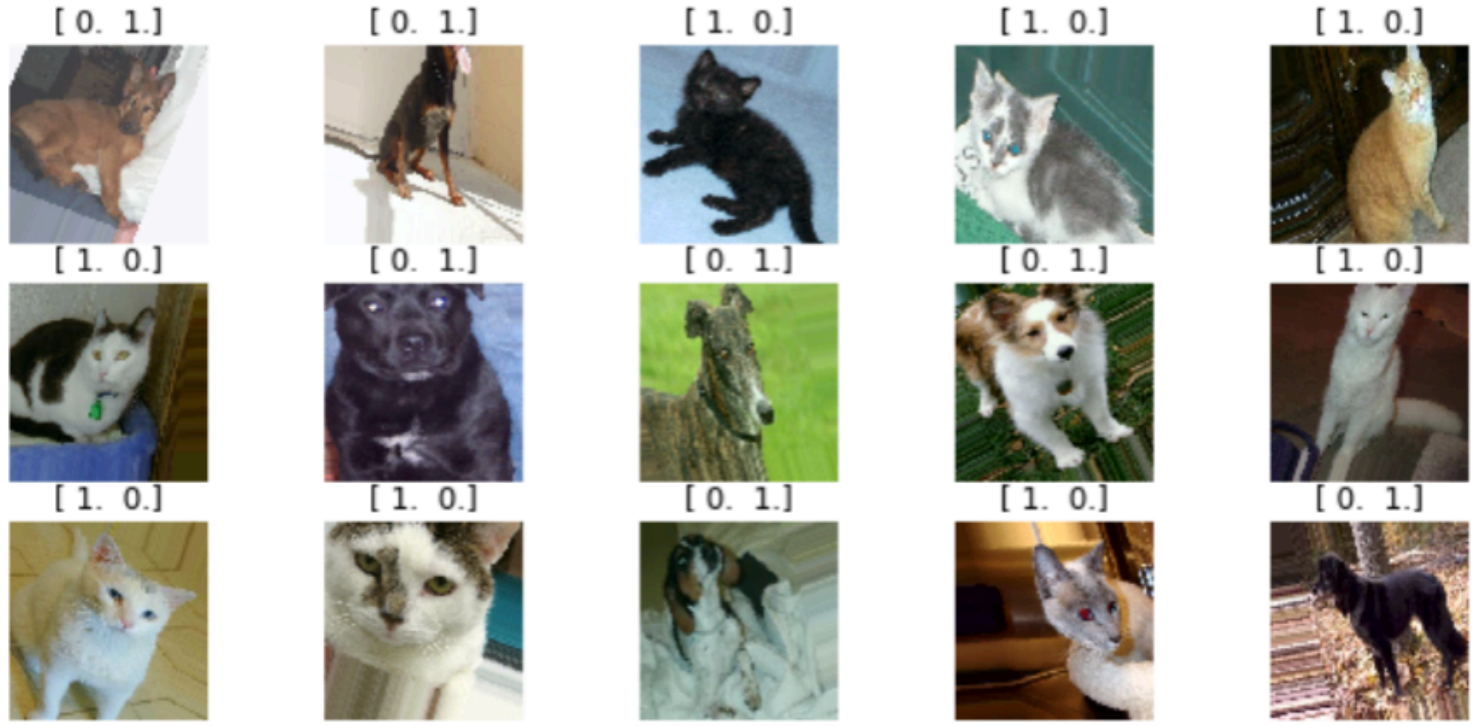
Part I : Data Processing

5. Then we used ImageDataGenerator and augmenting_datagen.flow to do data augmentation for individual image. Specifically, we flipped, rotated, shifted, zoomed the image. The image below shows the 15 augmented images.



Part I : Data Processing

6. We also looked into the performance of augmentation for different images in training dataset. The figure below shows the augmented images with their label. We can find out that cats have label [1 0], while dogs have label [0 1].

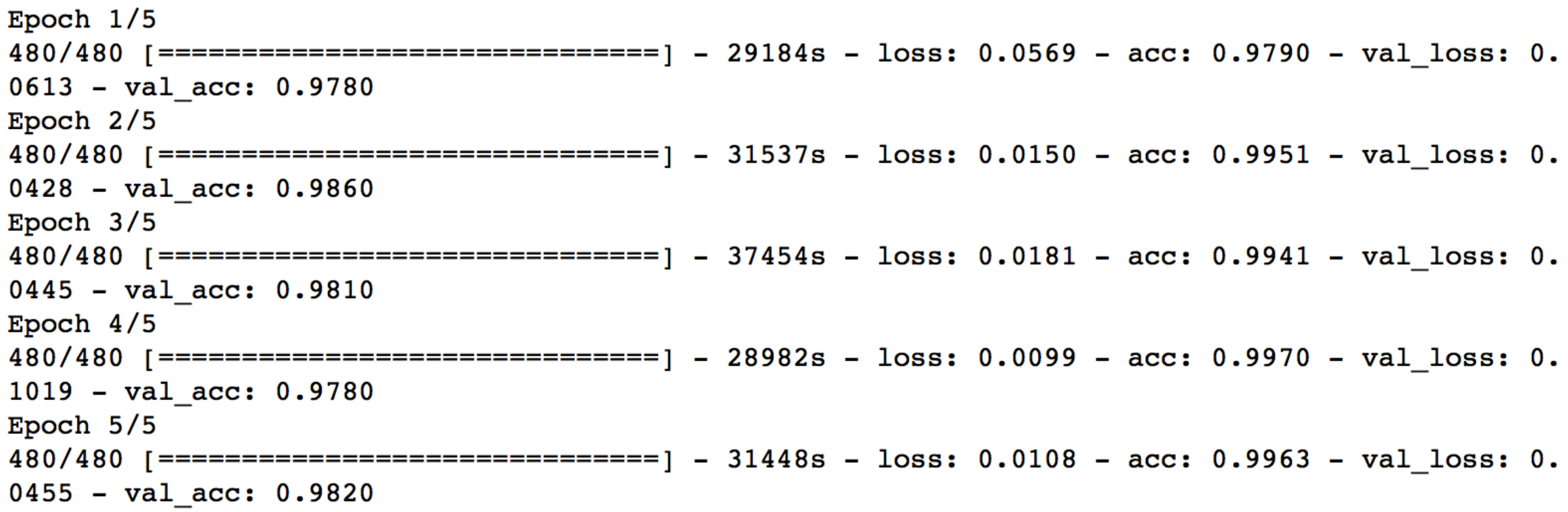


Part II : Realization

1. Loaded the pre-trained ResNet50 network, this network has 54 parameterized layers which includes 53 convolutional layers and 1 fully connected layer for soft max.
2. Removed the last layer, which is a fully connected layer for multiclass analysis.
3. Added a fully connected layer, in this layer the input dimension is 2048 while the output dimension is 1. We used ‘sigmoid’ function as the activation function so that the output of our network will be a probability.
4. Used ‘valgen.flow_from_directory’ function to load training dataset and validation dataset.
5. Complied the network use ‘Adam’ optimizer with learning rate ‘1e-4’, set loss to ‘binary_crossentropy’ and metrics to ‘accuracy’.
6. Set batch size to 50. Fit the data, and let the process run 5 epochs, each epoch has 480 iterations.

Part III : Results

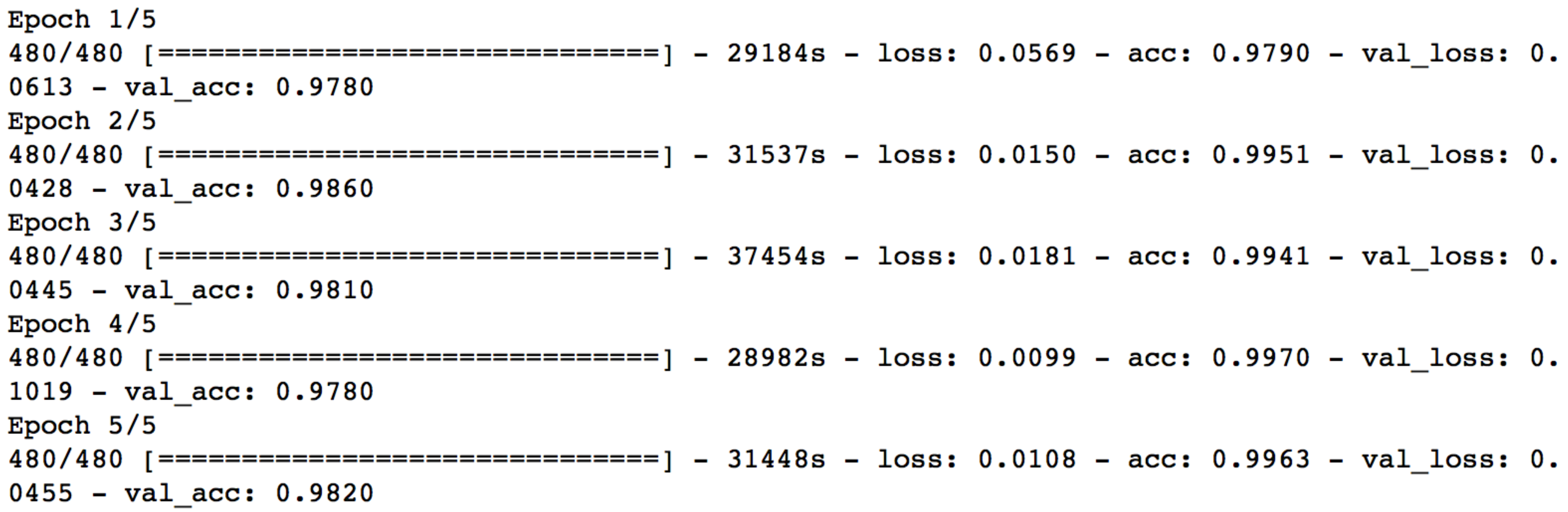
1. The image below shows the results. From the results we can see that the loss is high in the first epoch, then it goes down.



```
Epoch 1/5
480/480 [==============================] - 29184s - loss: 0.0569 - acc: 0.9790 - val_loss: 0.
0613 - val_acc: 0.9780
Epoch 2/5
480/480 [==============================] - 31537s - loss: 0.0150 - acc: 0.9951 - val_loss: 0.
0428 - val_acc: 0.9860
Epoch 3/5
480/480 [==============================] - 37454s - loss: 0.0181 - acc: 0.9941 - val_loss: 0.
0445 - val_acc: 0.9810
Epoch 4/5
480/480 [==============================] - 28982s - loss: 0.0099 - acc: 0.9970 - val_loss: 0.
1019 - val_acc: 0.9780
Epoch 5/5
480/480 [==============================] - 31448s - loss: 0.0108 - acc: 0.9963 - val_loss: 0.
0455 - val_acc: 0.9820
```

Part III : Results

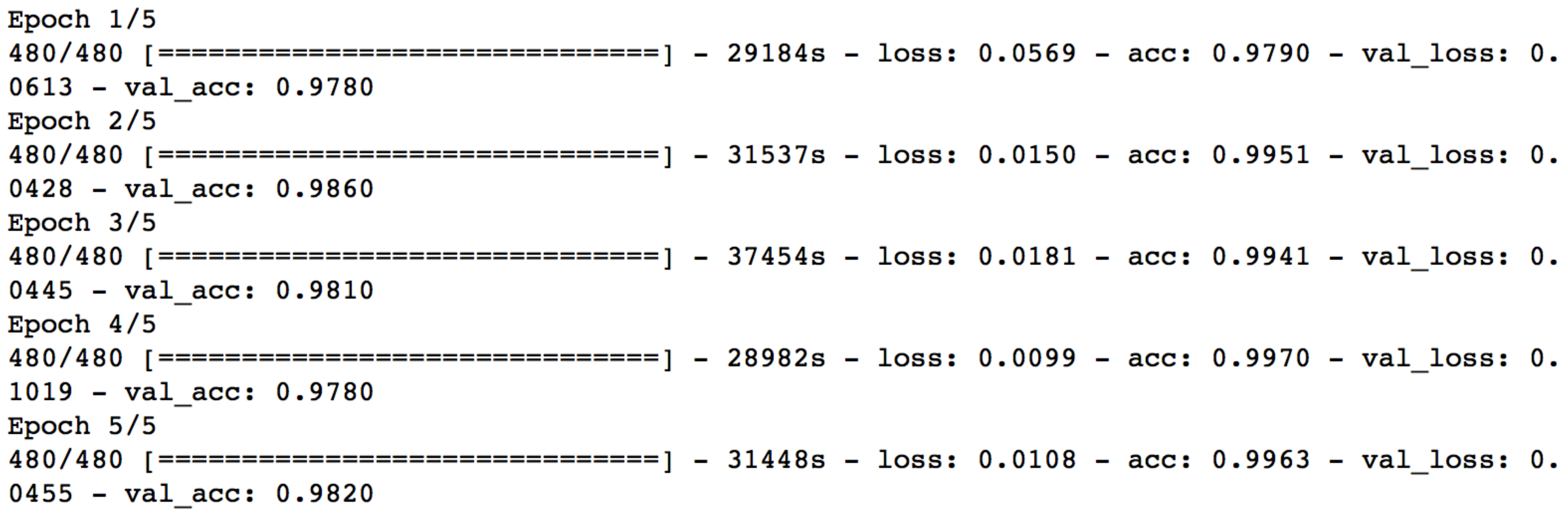
2. The accuracy for the training set is about 97% at first epoch, while the accuracy in the last four epochs are over 99% but have some vibration. That means the program goes stable after 2 epochs, so it is not necessary to run more epochs.



```
Epoch 1/5
480/480 [==============================] - 29184s - loss: 0.0569 - acc: 0.9790 - val_loss: 0.
0613 - val_acc: 0.9780
Epoch 2/5
480/480 [==============================] - 31537s - loss: 0.0150 - acc: 0.9951 - val_loss: 0.
0428 - val_acc: 0.9860
Epoch 3/5
480/480 [==============================] - 37454s - loss: 0.0181 - acc: 0.9941 - val_loss: 0.
0445 - val_acc: 0.9810
Epoch 4/5
480/480 [==============================] - 28982s - loss: 0.0099 - acc: 0.9970 - val_loss: 0.
1019 - val_acc: 0.9780
Epoch 5/5
480/480 [==============================] - 31448s - loss: 0.0108 - acc: 0.9963 - val_loss: 0.
0455 - val_acc: 0.9820
```

Part III : Results

3. The validation accuracy goes fast in the first epoch, but it seems that there is not much difference between each epoch.



```
Epoch 1/5
480/480 [==============================] - 29184s - loss: 0.0569 - acc: 0.9790 - val_loss: 0.
0613 - val_acc: 0.9780
Epoch 2/5
480/480 [==============================] - 31537s - loss: 0.0150 - acc: 0.9951 - val_loss: 0.
0428 - val_acc: 0.9860
Epoch 3/5
480/480 [==============================] - 37454s - loss: 0.0181 - acc: 0.9941 - val_loss: 0.
0445 - val_acc: 0.9810
Epoch 4/5
480/480 [==============================] - 28982s - loss: 0.0099 - acc: 0.9970 - val_loss: 0.
1019 - val_acc: 0.9780
Epoch 5/5
480/480 [==============================] - 31448s - loss: 0.0108 - acc: 0.9963 - val_loss: 0.
0455 - val_acc: 0.9820
```

Thanks!