Problem3_Report

February 15, 2019

Team member (alphabetical order): Jin Dong, Liheng Ma, Maximilien Le Clei.
Github Link: https://github.com/djdongjin/IFT6135-Assignment.

0.1 Environment Setting and Data Preprocessing

To reproduce the result, please put the dataset into a folder of google drive whose path is saved in *data_path*, in our case, data_path = '/content/gdrive/My Drive/Datasets/dogcat'

We load the dataset and move files into the virtual machine environment of colab to speed up image loading.

We divided the dataset into four folders:

- trainset: the original training dataset which will be used to re-train the final model after parameter tuning.
- testset: the original test dataset used to generate predictions.
- train, valid: the training dataset and validation dataset used to train and select hyperparameters. They are splitted from trainset as a split ratio.

We defined four data iterators as below, given a batch size. We also have different data augmentation pipelines for train dataset and test dataset, since test dataset should keep unchanged.

0.2 1. Model Architecture

Our model is based on VGG-11 with some modifications, since it is designed for ImageNet with size of 224x224, but our data size is 64x64. We deleted the last vgg block because too many pooling layers are included which reduced the image size to 2x2. Instead, we add an extra convolutional layer in the second vgg blocks to maintain the number of convolutional layers.

The final archtecture and corresponding hyperparameters (e.g. kernel size, padding) we used is as below. It includes four vgg blocks, each of which has one or two convolutional layers followed by a MaxPooling layer. At the end, we added three dense layers to produce the final results.

The number of parameters is: $32 * 3 * 3 + 64 * 3 * 3 * 2 + 128 * 3 * 3 * 2 + 256 * 3 * 3 * 2 + 4097 * 1024 + 1025 * 512 + 513 * 2 = 4729506$.

We also tried using larger dense layers (2048, 1024 for each hidden layer), but didn't get improvements on validation performance.

```
In [0]: # vgg11
```

```
Out[0]: Sequential(
          (0): VGGBlock(
            (net): Sequential(
              (0): Conv2D(3 -> 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (1): MaxPool2D(size=(2, 2), stride=(2, 2), padding=(0, 0), ceil_mode=False)
            )
          )
          (1): VGGBlock(
            (net): Sequential(
              (0): Conv2D(32 -> 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (1): Conv2D(64 -> 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (2): MaxPool2D(size=(2, 2), stride=(2, 2), padding=(0, 0), ceil_mode=False)
            )
          )
          (2): VGGBlock(
            (net): Sequential(
              (0): Conv2D(64 -> 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (1): Conv2D(128 -> 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (2): MaxPool2D(size=(2, 2), stride=(2, 2), padding=(0, 0), ceil_mode=False)
            )
          )
          (3): VGGBlock(
            (net): Sequential(
              (0): Conv2D(128 -> 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (1): Conv2D(256 -> 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
              (2): MaxPool2D(size=(2, 2), stride=(2, 2), padding=(0, 0), ceil_mode=False)
            )
          )
          (4): Dense(4096 -> 1024, Activation(relu))
          (5): Dense(1024 -> 512, Activation(relu))
          (6): Dense(512 -> 2, linear)
        )
```

0.3 2. Experiments

Dataset

We split the original training set into a training set and a validation set with a ratio of 0.8. Meanwhile, we used several data augmentation techniques to expand the dataset, including:

- Random crop the image and resize to the original size: RandomResizedCrop.
- Random flip left or right the image: RandomFlipLeftRight.
- Random change the color properties of the image: RandomColorJitter, RandomLighting.

We also tried to normalize the image but the performance didn't change too much. We also only did augmentation on training set to ensure the validation accuracy is sound.

Results

We finally achieved **0.8682** accuracy on training set and **0.8464** accuracy on validation set, with hyperparameter setting mentioned below. After retraining, the highest score we achieved on Kaggle is **88%**.

We plot the accuracy (1-error) and loss curves on training set and validation set. We found that the learning speed is slow. Compared with SGD, a more powerful optimizier, such as Adam, should be helpful and faster to converge to a minimum point.

BatchNorm should be helpful to improve the validation performance, since it can be seen as a normalization over dataset and can let the model learn more steady.

A more advanced architecture with more normalization techniques is also helpful. For example, we did experiments on ResNet. Without BatchNorm, its performance is bad. But after adding BatchNorm, it's performance on validation set exceeded our model, achieved 91%. (Since we didn't implemented BatchNorm by hand, we didn't use this model for any submission. Details can be found in appendix.)

Dropout should not be very helpful because from the curve we can see that the model is not overfitted, which is the main issue that Dropout aims to solve.

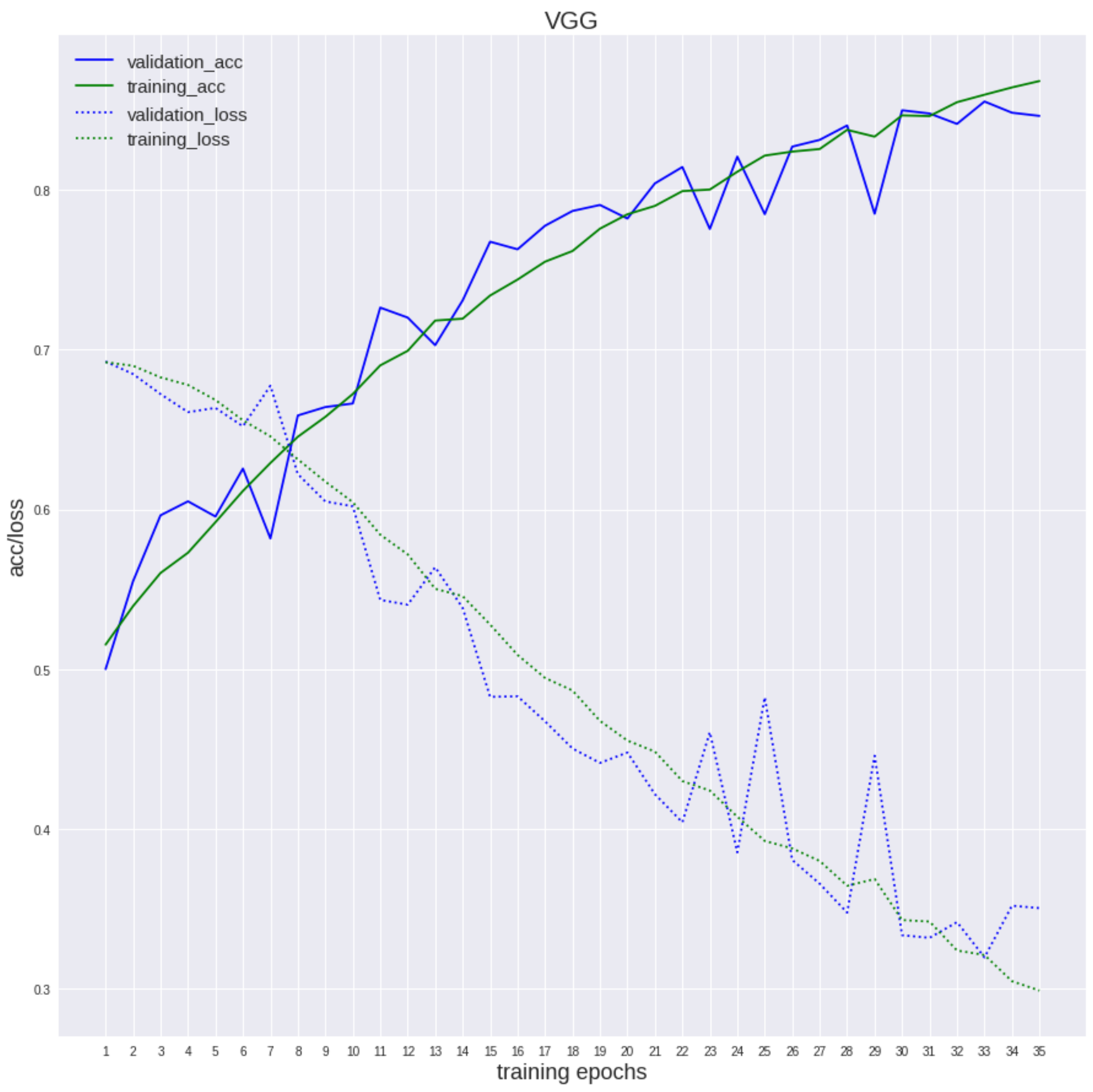
After tuning hyperparameters, we retrained the model useing the whole training set and generated several submissions. The model could achieve **93%** accuracy on validation set whereas on the test set, it only achieved **88%**. The reason is that when retraining, the model has more data so that its accuracy becomes higher (also, the validation set is also included in the training set when retraining). But the test set is new and never be seen by the model, so that the accuracy becomes lower again.

0.3.1 Generating Results

The hyperparameter setting we used for submission is as below:

- Epochs: 35
- Batch size: 128
- Learning rate: 0.1
- Model architecture: (1, 32), (2, 64), (2, 128), (2, 256), 1024, 512, 2. ((i, j) means a vgg block where i is denoted as number of convolutional layers and j is denoted as number of channels; the last three layers are dense layers.)

```
In [0]: loss_acc_plot(content=["valid_acc","train_acc","valid_loss","train_loss"],
                      valid_loss=valid_loss, train_loss=train_loss,
                      valid_acc=valid_accuracy,train_acc=train_accuracy)
```



0.3.2 Hyperparameter Tuning

All of settings are based on the setting used for generating results and only the corresponding hyperparameter is different:

- Batch size: (128, 256), validation accuracy: (0.8464, 0.7571)
- Learning rate: (0.1, 0.05, 0.005), validation accuracy: (0.8464, 0.8304, 0.6393)
- Epochs: (15, 25, 35), validation accuracy: (0.7676, 0.7849, 0.8464)

0.4 Appendix: ResNet Implementation

We also implemented and tested a ResNet on the dataset. With BatchNorm, it can achieve higher accuracy with less epochs. But when BatchNorm is dropped out, its accuracy is pretty low. Considering that BatchNorm need to be implemented by hand, we only used this part as an experiment and didn't use it to submit results on kaggle.