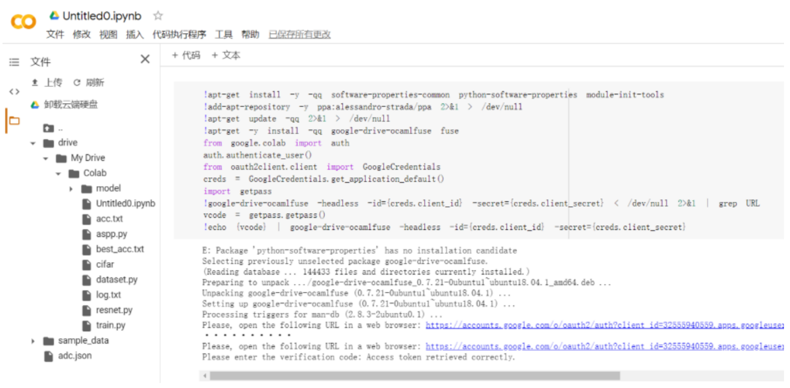
Experiment 2: Training Deep Neural Networks for CIFAR10

I. Experiment Introduction

In this experiment we need to learn how to train a deep neural network for dataset CIFAR10 and fulfill the task of classification. The given CIFAR10 dataset consists of 55000 training images and 11000 test images, each of which is 32*32 size with R, G, B channels. These images are classified in **12** classes: ***airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck, fish, flowers.***

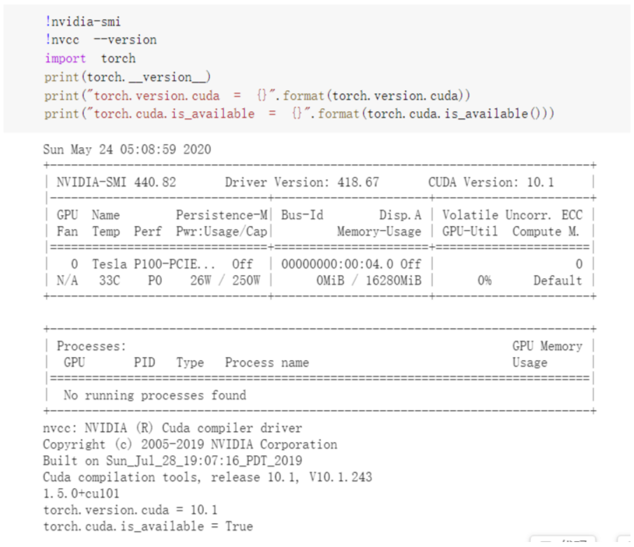
Since There is no available computing resource nearby for me, I choose to use CoLaboratory (CoLab) developed by Google. CoLab is very easy to use, but I can only have limited time for GPU resources. The interface and configuration code of CoLab are shown below.



```
!apt-get install -y -qq software-properties-common python-software-properties module-init-tools
!add-apt-repository -y ppa:alessandro-strada/ppa 2>&1 > /dev/null
!apt-get update -qq 2>&1 > /dev/null
!apt-get -y install -qq google-drive-ocamlfuse fuse
from google.colab import auth
auth.authenticate_user()
from oauth2client.client import GoogleCredentials
creds = GoogleCredentials.get_application_default()
import getpass
!google-drive-ocamlfuse -headless -id={creds.client_id} -secret={creds.client_secret} < /dev/null 2>&1 | grep URL
vcode = getpass.getpass()
!echo {vcode} | google-drive-ocamlfuse -headless -id={creds.client_id} -secret={creds.client_secret}
```

```
E: Package 'python-software-properties' has no installation candidate
Selecting previously unselected package google-drive-ocamlfuse.
(Reading database ... 144433 files and directories currently installed.)
Preparing to unpack .../google-drive-ocamlfuse_0.7.21-0ubuntu1~ubuntu18.04.1_amd64.deb ...
Unpacking google-drive-ocamlfuse (0.7.21-0ubuntu1~ubuntu18.04.1) ...
Setting up google-drive-ocamlfuse (0.7.21-0ubuntu1~ubuntu18.04.1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Please, open the following URL in a web browser: https://accounts.google.com/o/oauth2/auth?client_id=32555940559.apps.googleuse
··········
Please, open the following URL in a web browser: https://accounts.google.com/o/oauth2/auth?client_id=32555940559.apps.googleuse
Please enter the verification code: Access token retrieved correctly.
```

Fig.1 CoLab Interface and configuration code



```
!nvidia-smi
!nvcc --version
import torch
print(torch.__version__)
print("torch.version.cuda = {}".format(torch.version.cuda))
print("torch.cuda.is_available = {}".format(torch.cuda.is_available()))
```

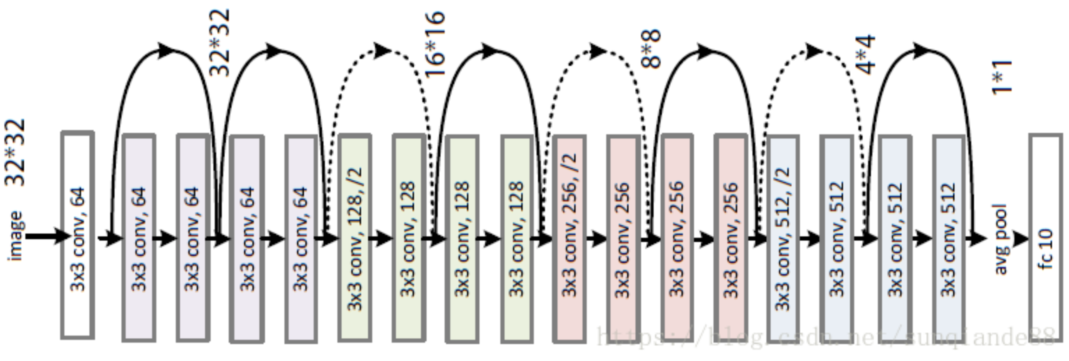
```
Sun May 24 05:08:59 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.82       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   33C    P0    26W / 250W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Sun_Jul_28_19:07:16_PDT_2019
Cuda compilation tools, release 10.1, V10.1.243
1.5.0+cu101
torch.version.cuda = 10.1
torch.cuda.is_available = True
```

Fig,2 Available GPU resource

II. Model Implementation

2.1 ResNet18

ResNet is first brought up by Kaiming He in 2016.[1] ResNet bring in the residual module so as to deal with the gradient disappearance and explosion problems. In this experiment I choose to implement ResNet18 because of limited computing resources. The structure of ResNet18 and the residual block are shown below.



Fig,3 The structure of ResNet18

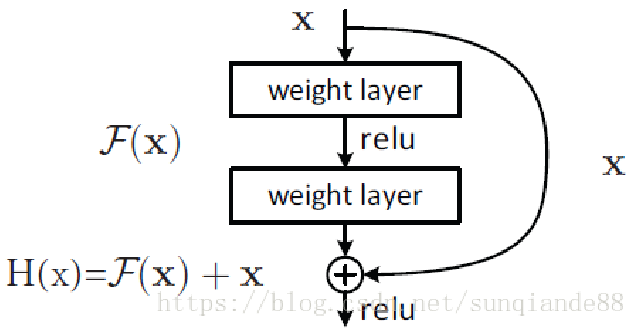
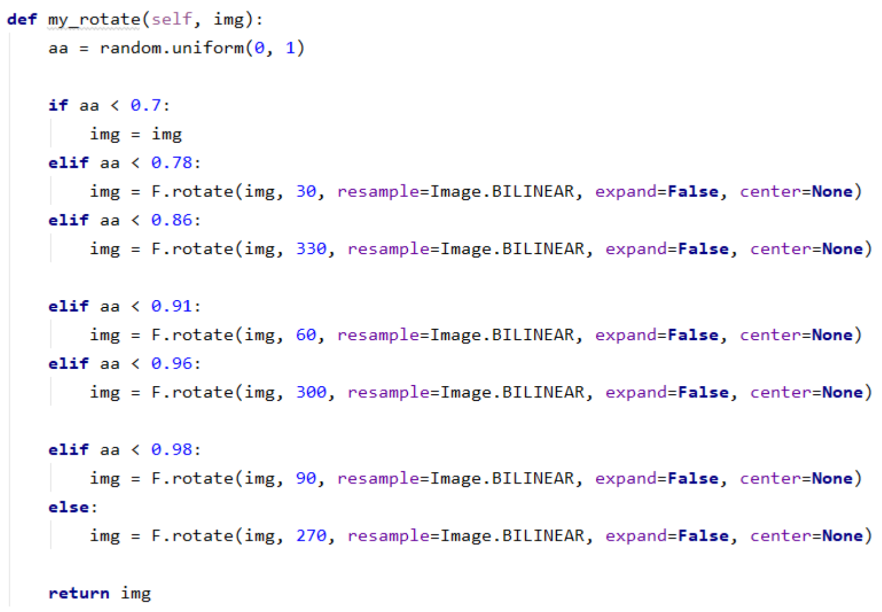


Fig.4 Residual block

2.2 Data Augmentation

Usually, we need to do data augmentation before training. This procedure can add the ‘nonlinearity’, thus make our network more robust to datasets. In PyTorch libraries, there are 22 *transforms* (augmentation methods) including 4 classes: crop, flip and rotation, image transform, operation for above transforms.

In this experiment, I implemented *HorizontalFlip()* in model 2 and *HorizontalFlip()*, *VerticalFlip()*, self-defined Rotation in model 2, 3, 4. In the rotation transform, I set rotation angles including 0°, 30°, 330°, 60°, 300°, 90°, 270° with different possibilities respectively.



```python
def my_rotate(self, img):
    aa = random.uniform(0, 1)

    if aa < 0.7:
        img = img
    elif aa < 0.78:
        img = F.rotate(img, 30, resample=Image.BILINEAR, expand=False, center=None)
    elif aa < 0.86:
        img = F.rotate(img, 330, resample=Image.BILINEAR, expand=False, center=None)

    elif aa < 0.91:
        img = F.rotate(img, 60, resample=Image.BILINEAR, expand=False, center=None)
    elif aa < 0.96:
        img = F.rotate(img, 300, resample=Image.BILINEAR, expand=False, center=None)

    elif aa < 0.98:
        img = F.rotate(img, 90, resample=Image.BILINEAR, expand=False, center=None)
    else:
        img = F.rotate(img, 270, resample=Image.BILINEAR, expand=False, center=None)

    return img
```

Fig.5 Self-defined rotation transform in model 2

2.3 CBAM

Convolutional Block Attention Module (CBAM)[2] is a attention module including both spatial attention and channel attention. This module generates a weight map which we can use to multiply with the feature map and get the needed attention effect. The structure of CBAM applied in ResNet and code are shown below.

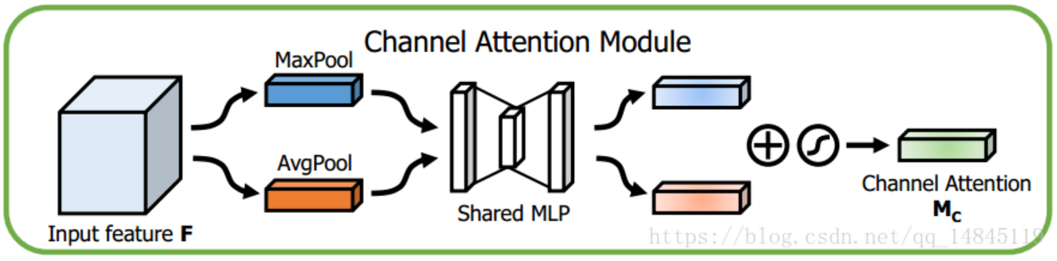


Fig.6 Channel attention module

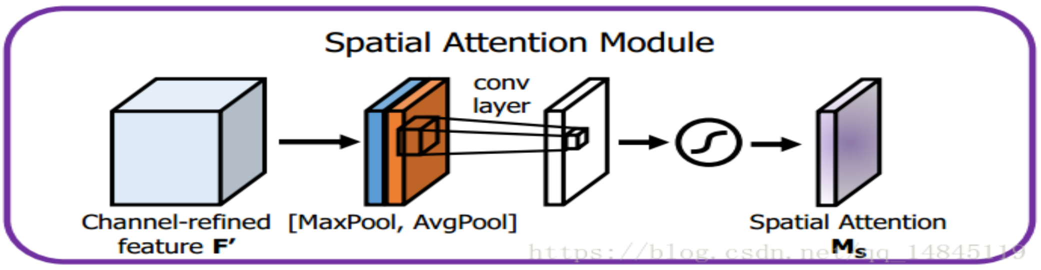
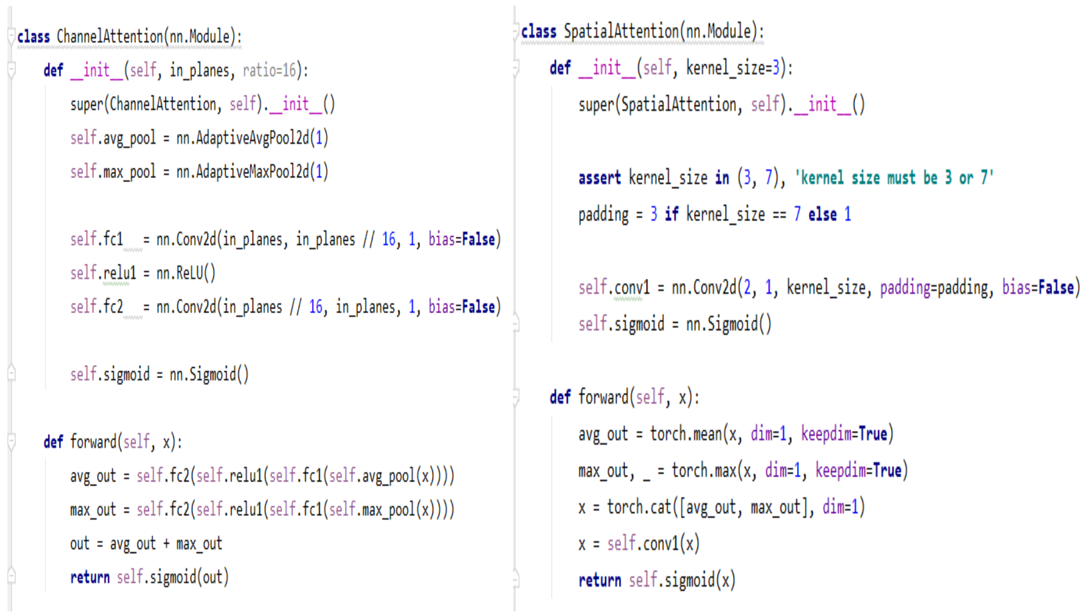


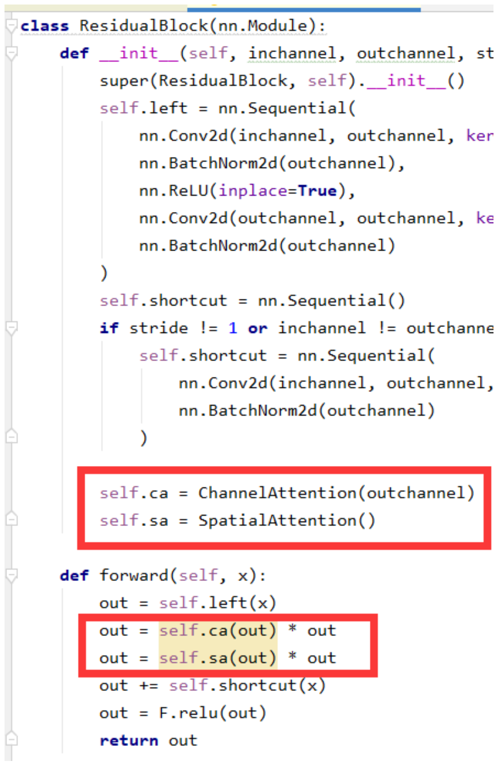
Fig.7 Spatial attention module



```
class ChannelAttention(nn.Module):
    def __init__(self, in_planes, ratio=16):
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)

        self.fc1   = nn.Conv2d(in_planes, in_planes // 16, 1, bias=False)
        self.relu1 = nn.ReLU()
        self.fc2   = nn.Conv2d(in_planes // 16, in_planes, 1, bias=False)

        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.fc2(self.relu1(self.fc1(self.avg_pool(x))))
        max_out = self.fc2(self.relu1(self.fc1(self.max_pool(x))))
        out = avg_out + max_out
        return self.sigmoid(out)
```

```
class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=3):
        super(SpatialAttention, self).__init__()

        assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
        padding = 3 if kernel_size == 7 else 1

        self.conv1 = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = torch.mean(x, dim=1, keepdim=True)
        max_out, _ = torch.max(x, dim=1, keepdim=True)
        x = torch.cat([avg_out, max_out], dim=1)
        x = self.conv1(x)
        return self.sigmoid(x)
```

Fig.8 Code of CBAM

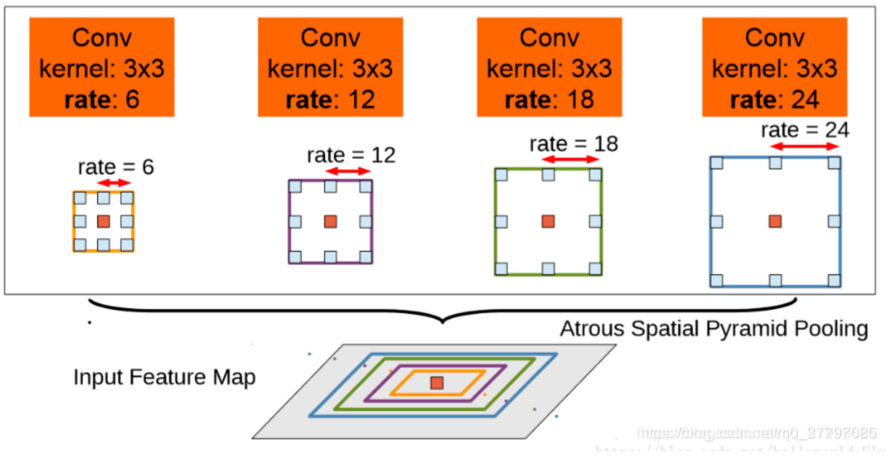


```
class ResidualBlock(nn.Module):
    def __init__(self, inchannel, outchannel, st
        super(ResidualBlock, self).__init__()
        self.left = nn.Sequential(
            nn.Conv2d(inchannel, outchannel, ker
            nn.BatchNorm2d(outchannel),
            nn.ReLU(inplace=True),
            nn.Conv2d(outchannel, outchannel, ke
            nn.BatchNorm2d(outchannel)
        )
        self.shortcut = nn.Sequential()
        if stride != 1 or inchannel != outchanne
            self.shortcut = nn.Sequential(
                nn.Conv2d(inchannel, outchannel,
                nn.BatchNorm2d(outchannel)
            )

        self.ca = ChannelAttention(outchannel)
        self.sa = SpatialAttention()

    def forward(self, x):
        out = self.left(x)
        out = self.ca(out) * out
        out = self.sa(out) * out
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

Fig.9 Implementing CBAM in Residual module

2.4 ASPP

Atrous Spatial Pyramid Pooling (ASPP)[3] implements different convolution block with same kernel but different rate. By doing so, ASPP can capture information from multi scales in the feature map and can help our network classify objects with different size in the dataset. The structure of ASPP is shown below.



Fig,10 Structure of ASPP

However, in this experiment we are given images with size of 32*32, which is not a very large scale, so I slightly changed the code of ASPP. The ASPP here used is more like SPP. Please note that.

III. Results and Evaluation

In this experiment I implemented 4 models: basic ResNet18 model, ResNet18 with augmented data, ResNet18 with augmented data and CBAM, ResNet18 with augmented data and CBAM and ASPP. The final result comparison is listed below.

Table.1 Test comparison of 4 models

Model	Basic	Data Augment(DA)	DA&CBAM	DA&CBAM &ASPP
Procedures (Epoch = 300)	*HorizontalFlip()*	*HorizontalFlip() VerticalFlip (p=0.1) RandomRotate (0.7, 0.08,0.08,0.05, 0.05,0.02,0.02)*	*HorizontalFlip() VerticalFlip (p=0.1) RandomRotate (0.89, 0.03,0.03,0.02, 0.02,0.01,0.01)*	*HorizontalFlip() VerticalFlip (p=0.1) RandomRotate (0.89, 0.03,0.03,0.02, 0.02,0.01,0.01)*
Best test accuracy	95.000%	93.409%	94.709%	94.936%
Best test epoch	286	249	232	281

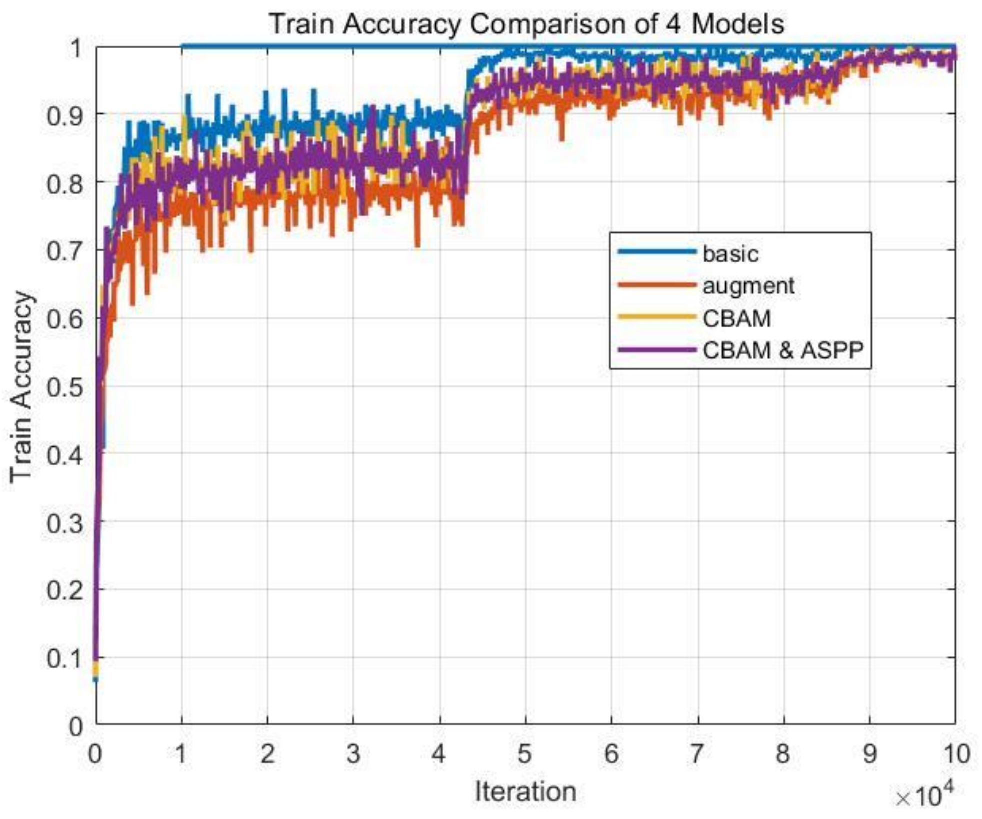


Fig.11 Train accuracy

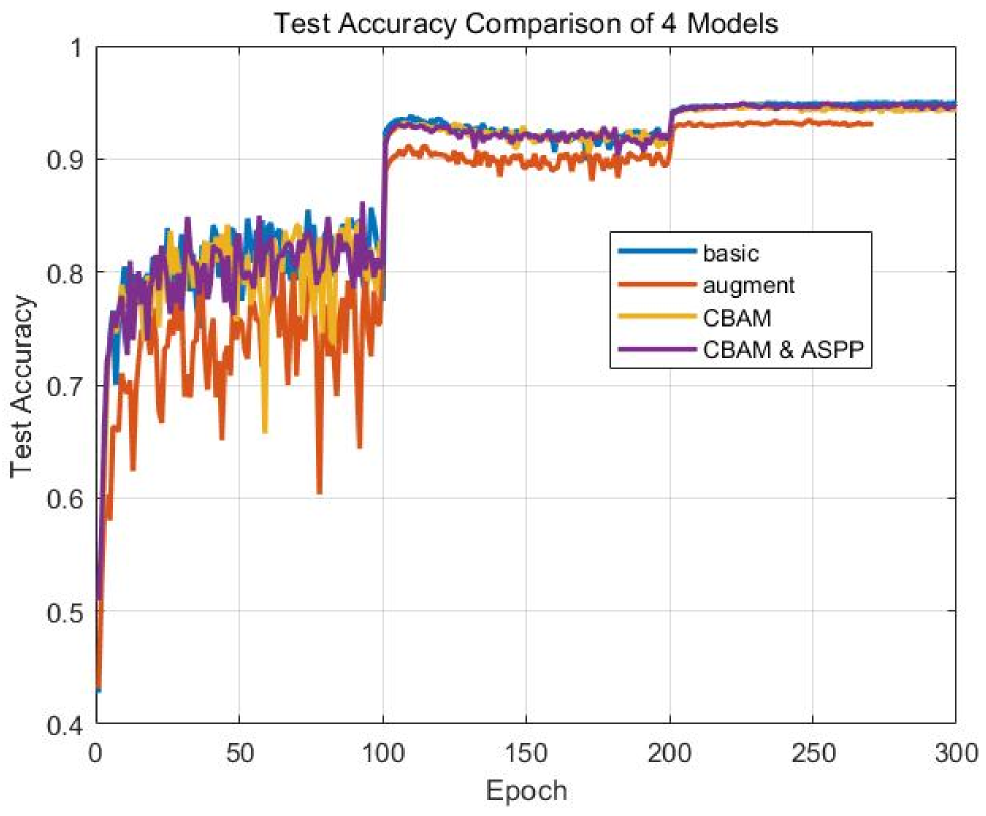


Fig.12 Test accuracy

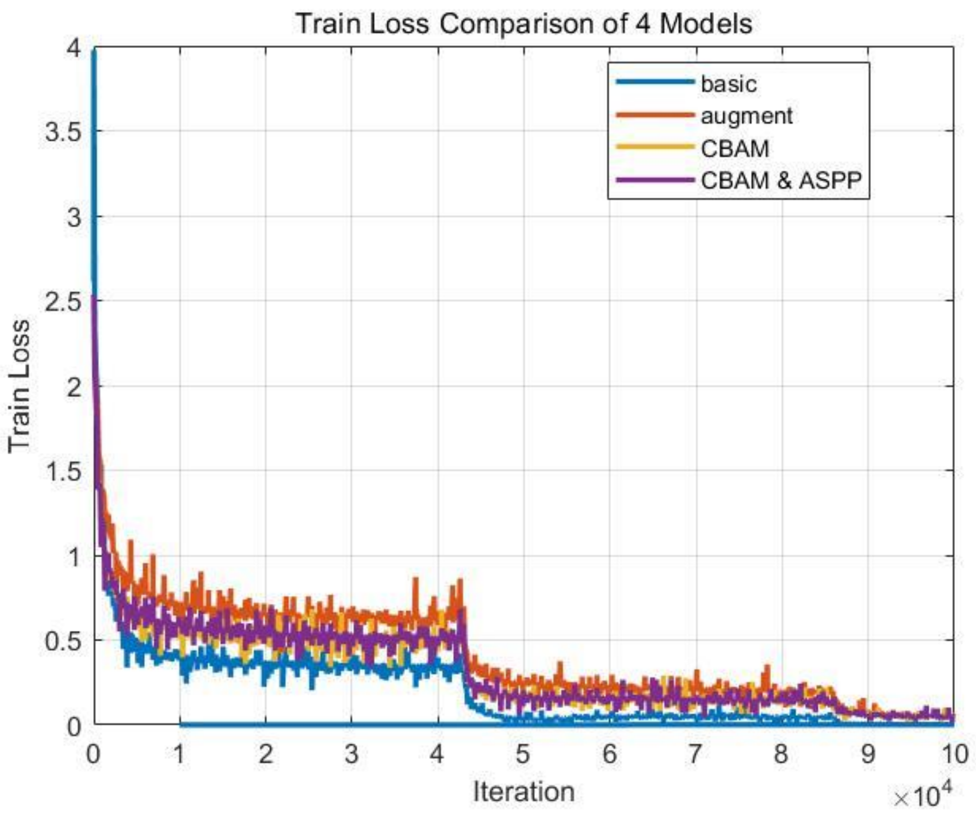


Fig.13 Training Loss

From above pictures we can find very sharp changes in both accuracies and loss. This is because I set learning rate as 0.1 in epoch 1-100, 0.01 in epoch 101- 200, 0.001 in epoch 201-300. This decaying learning rate trick helps our model converge to the optimal object more easily because as the training procedure proceeds, it gets closer to the optimal point. If the learning rate is still very large, our model might leap over this point and turn to worse point somewhere else.

Meanwhile, it's not surprising that sometimes, adding these modules would not make the final performance better. Because, to make it very simple, all these modules add up the 'nonlinearity' for the features and thus make our neural network more robust theoretically. But if the dataset is not that complicated. Such procedure will make no efforts to help improve the performance. Besides I also did not pay too much efforts in parameters tuning and trial-error procedure due to limited computing resources. I believe that with more computing resources I can pay more efforts to find better way for these module to work together and show better classification performance.

Note that part of the training log of model 2 is missing because of some errors of Colab occurred when I was sleeping. Colab will disconnect with Google drive if you have no movements for a long time.

IV. Conclusions

In this experiment I learned how to use PyTorch to implement deep learning algorithms. Specifically, I implemented ResNet18 with modules like CBAM and ASPP for the classification task with given dataset CIFAR10

Though the basic model showed the best performance with the test accuracy of 95%, this does not mean that CBAM and ASPP is useless at all. Because I did not train the models too much times and make trial-error for some parameters.

Through this experiment I learned more about how to complete a deep learning project. From feature engineering to data augmentation, network structure, and other modules like CBAM and ASPP. Not just these techniques but also the general idea shall be more helpful in my future research.

Reference

[1]. He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition[C]. computer vision and pattern recognition, 2016: 770-778.

[2]. Woo S, Park J, Lee J, et al. CBAM: Convolutional Block Attention Module[C]. european conference on computer vision, 2018: 3-19.

[3]. Chen L, Papandreou G, Schroff F, et al. Rethinking Atrous Convolution for Semantic Image Segmentation[J]. arXiv: Computer Vision and Pattern Recognition, 2017.