21-1 Machine Learning and Artificial Intelligence (01)

Final Project

1971060

Cyber Security Major

Kim, Seohyun

Table of Contents

1. Introduction

For the final project, the given tasks were mainly about implementing inception module for CIFAR-10. After several trial and error, I have used the 'GoogLeNet' CNN algorithm and have applied this to show a higher accuracy.

The followings are short explanations about the two main factors of this project.

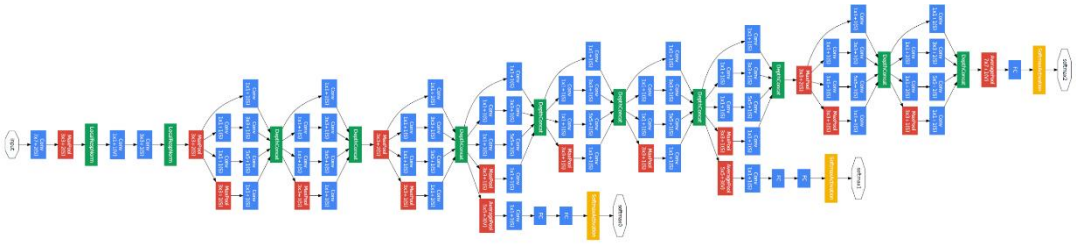
A. CIFAR-10

Image classification has several datasets. One of them is the 'CIFAR-10'. CIFAR-10 data consists of 50000 learning data and 10000 test data. It is a challenge dataset that has not been conquered until recently, even though it is a small dataset. In Auto-Augment paper 4, error rate of 1.48% was achieved. Thus, the CIFAR-10 dataset has been recently conquered.

In addition, in many cases, GPUs from personal equipment or clouds are often utilized when studying deep learning. CIFAR-10 datasets can be learned because learning is possible with using a single GPU.

B. GoogLeNet

GoogLeNet is an algorithm that won the 2014 ImageNet Image Recognition Contest (ILSVRC) over VGGNet (VGG19). The GoogLeNet consists of 22 floors (the figure bellow shows the structure) which is little deeper than the 19 floored VGG19. GoogLeNet's original paper is "Going Deeper with Convolutions" published in 2015 CVPR by Christian Szegedy et al. As the name GoogLeNet suggests, Google participated in the development of the algorithm.



GoogLeNet has four main characteristics.

First, it uses 1 x 1 size filter for convolution to reduce the number of feature maps. If the number of characteristic maps decreases, the amount of computation decreases.

For example, suppose you have a 14 x 14 feature map (14 x 14 x 480) in Chapter 48. Convolutionizing this into 48 5 x 5 x 480 filter kernels produces a characteristic map (14 x 14 x 48) of Chapter 48. (We assumed that we set zero padding to 2 and convolutional

stride to 1). What is the number of operations required? Immediately (14 x 14 x 48) x (5 x 5 x 480) = approximately 112.9 M.

This time let's reduce the number of feature maps by convolutionizing 480 chapters (14 x 14 x 480) into 16 1 x 1 x 480 filter kernels first. The result is a feature map (14 x 14 x 16) of Chapter 16. Note that 480 characteristic maps have been reduced to 16 characteristic maps. Convolutionizing this 14 x 14 x 16 feature map into 48 5 x 5 x 16 filter kernels produces a feature map (14 x 14 x 48) of Chapter 48. Let us confirm above that the resulting feature map has the same size and depth as in the absence of a 1 x 1 convolution. Then (14 x 14 x 16) * (1 x 1 x 480) + (14 x 14 x 48) * (5 x 5 x 16) = 5.3 M operations are needed. Like this, it has a much smaller computation volume compared to 112.9M. The fact that computation can be reduced is important in that it helps to make the network deeper.

Second, it uses a unique inception module. All nine-inception module used in GoogLeNet included 1 x 1 convolution. The 1x1 convolution serves to reduce the longevity of feature maps. Looking at the naive version except for the 1x1 convolution represented in yellow blocks, we stack all the feature maps obtained from the previous layer together by 1x1 convolution, 3x3 convolution, 5x5 convolution, and 3x3 max pooling. Previous CNN models such as AlexNet and VGGNet differ from convolution using the same size filter kernel on one floor. Thus, more diverse kinds of characteristics are derived. Since this includes a 1x1 convolution, of course, the computation volume must have been significantly reduced.

Third, GoogLeNet uses global average pooling method instead of fully connected layers. The global average pooling is the average of the characteristic maps produced on the entire layer, followed by the creation of a one-dimensional vector. This is because it is necessary to create a one-dimensional vector to finally connect the softmax layer for image classification. If a feature map of 7 x 7 in Chapter 1024 was created on the whole floor, then the values of 1024 obtained by averaging each of the 7 x 7 feature maps in Chapter 1024 are connected by a vector. The advantage of doing so is that it eliminates a significant number of weights. If the FC method is used, the number of weights that require training is 7 x 7 x 1024 x 1024 = 51.3M, but with global average pooling, no single weight is required.

Finally, it uses two auxiliary classifiers. As the depth of the network deepens, the vanishing gradient problem becomes more difficult to avoid. Thus, backpropagation is the main use of backpropagation in the process of training weights, in which the gradient used to update weights becomes smaller and smaller, leading to zero. Therefore, the weights within the network are not properly trained. To overcome this problem, GoogLeNet puts two auxiliary classifiers in the middle of the network. The configuration of the auxiliary classifier is like

the following. Convolution -> 1024 FC layer -> 1000 FC layer -> softmax with 5 x 5 mean pooling (stride 3) -> 128 1x1 filter kernels.

2. Tasks

A. Implementing Inception module for CIFAR-10

i. Trial 1: using VGGNet algorithm

I have tried using the VGGNet algorithm to implement the module given.

The reason why I choose this algorithm as my first trial was because VGGNet was the model that won the 2014 ImageNet Image Recognition Contest. I had expected finding a suitable code for the project would be easier than others as it is one the oldest algorithm that is still used and have won the contest.

However, I could not succeed in using the algorithm and had repeated runtime errors. As a result, I decided to use a different algorithm.

In short, this trial had an output of 0% accuracy.

ii. Trial 2: using GoogLeNet algorithm

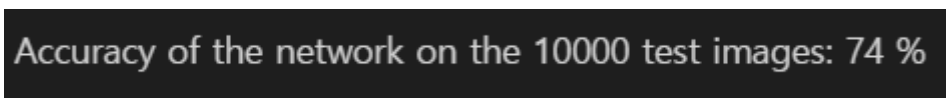
The next popular algorithm that is used nowadays and has a history of winning the contest appeared to be the GoogLeNet. So, I tried using this algorithm and have succeeded.

The main code defining the class is like the following.

```
class GoogLeNet(nn.Module):
    def __init__(self):
        super(GoogLeNet, self).__init__()

        self.pre_layers = nn.Sequential(
            nn.Conv2d(3, 192, kernel_size=3, padding=1),
            nn.BatchNorm2d(192),
            nn.ReLU(True),
        )

        self.a3 = Inception(192,  64,  96, 128, 16, 32, 32)
        self.b3 = Inception(256, 128, 128, 192, 32, 96, 64)
```

```
        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a4 = Inception(480, 192,  96, 208, 16,  48,  64)
        self.b4 = Inception(512, 160, 112, 224, 24,  64,  64)
        self.c4 = Inception(512, 128, 128, 256, 24,  64,  64)
        self.d4 = Inception(512, 112, 144, 288, 32,  64,  64)
        self.e4 = Inception(528, 256, 160, 320, 32, 128, 128)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a5 = Inception(832, 256, 160, 320, 32, 128, 128)
        self.b5 = Inception(832, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AvgPool2d(8, stride=1)
        self.linear = nn.Linear(1024, 10)

    def forward(self, x):
        out = self.pre_layers(x)
        out = self.a3(out)
        out = self.b3(out)
        out = self.maxpool(out)
        out = self.a4(out)
        out = self.b4(out)
        out = self.c4(out)
        out = self.d4(out)
        out = self.e4(out)
        out = self.maxpool(out)
        out = self.a5(out)
        out = self.b5(out)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.linear(out)

    return out
```

In short, this trial had an output of 74% accuracy(elapsed time: 639.7237727642059s).



B. Increasing Test Set Accuracy

i. Trial 3: decreasing inception value

The GoogLeNet structure is like the following.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

I thought the number of operations to learn parameters is too expensive since GoogLeNet is specialized with 336 * 336 px sized image classification. Therefor I have rearranged the order of inceptions and resized convolutional layer sizes. To minimize computation time duration, I used the following inception layers.

```
Conv2d
BatchNorm2d
ReLU
Inception: n1x1(48), n3x3(8), n5x5(16), pool(16)
Inception: n1x1(96), n3x3(16), n5x5(32), pool(32)
MaxPool2d
Inception: n1x1(160), n3x3(256), n5x5(64), pool(64)
Inception: n1x1(256), n3x3(256), n5x5(128), pool(128)
Inception: n1x1(256), n3x3(256), n5x5(128), pool(128)
MaxPool2d
Inception: n1x1(256), n3x3(512), n5x5(128), pool(128)
Inception: n1x1(384), n3x3(384), n5x5(128), pool(128)
AvgPool2d
Linear
```

In short, this trial had an output of 78% accuracy(elapsed time: 476.0275800228119s).

ii. Trial 4: increasing epochs value

Candidates of deformable values were the following. Batch size, filter size, number of filters, pooling vs strided convolution, network architectures, optimizers, activation functions, regularizations, model ensembles, data augmentation. Among these, changing the epoch seemed most well-marking for me.

During the two trials above, the epoch value was set to 3. At the first attempt, I tried running the dataset using 20 epoch. However, Colab GPU would not support this learning environment as for the free version. So, I decided to set the value of epoch as 10.

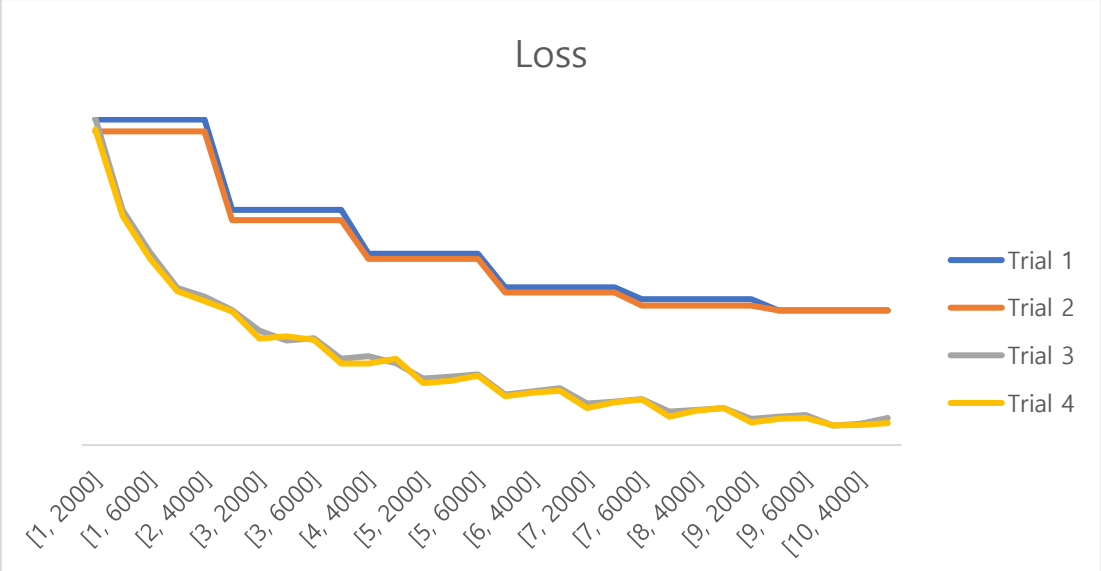
The result of this trial showed big decreasing rate in loss and notable increasing rate in accuracy. Specific rate will be explained in the next section.

In short, this trial had an output of 86% accuracy(elapsed time: 10399.19404554367s).

```
Accuracy of the network on the 10000 test images: 86 %
```

C. Finding Hyper-parameters

To check whether the machine is being learned, I tracked the loss value three times for each epoch. The result is like the following. Specific value is marked in the code submitted.



3. Conclusion

For the final project, I had four main trials in increasing the accuracy. As a result, the accuracy has

increased from 0% to 86% in result.

The below chart shows the alteration of the accuracy rate for which trial I have tried.

Trial	Content	Accuracy(%)
1	using VGGNet algorithm	0
2	using GoogLeNet algorithm	74
3	decreasing inception value	78
4	increasing epoch value	86

4. References

'[CNN 알고리즘들] GoogLeNet(inception v1)의 구조(https://bskyvision.com/539)'

'CIFAR-10 정복 시리즈 0: 시작하기(https://dnddnjs.github.io/cifar10/2018/10/07/start_cifar10/)'

Pytorch official documentation.

Stanford CS231n lectures.

Szegedy et al., "Going deeper with convolutions", CVPR 2015