<<IERG4160>>

<<Image and Video Processing>>

Report on Final Project

Members:

Liu Kam Nam (CUID: 1155109652)

Submission Date: 19/12/2020

I. OBJECTIVES

- solving Cifar-10 image classification task using neural networks.
- study the influence of hyper-parameters in LeNet.

II. DATA ANALYSIS

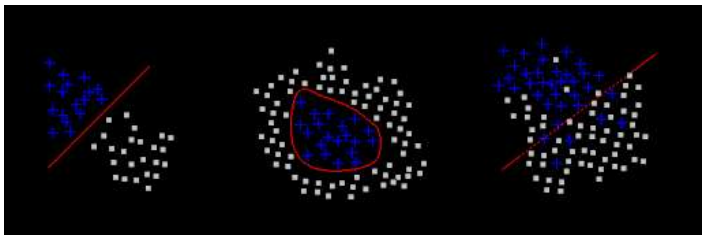
1. Train the default linear classification model and report the test result via:

python main.py

```
class Linear(nn.Module):
    def __init__(self, num_classes=10):
        super(Linear, self).__init__()
        self.fc = nn.Linear(32 * 32 * 3, num_classes)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

```
Final testing result: 29.700%
```



Linear classification only works well when the data is linear. It cannot classify non-linear data and inseparable data showed on the graph above [1]. Therefore, using the default linear classification model to solve Cifar-10 image classification task, result in a low accuracy, just 29.7%.

2. Implement an MLP model with 3 hidden layers whose channels are all 128. Each hidden layer is followed by ReLU activation. Use an additional fully connected layer to map the dimension into 10 classes. Please report the test result.

```
class MLP(nn.Module):
    def __init__(self, num_classes=10):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(32 * 32 * 3, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 128)
        self.fc4 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

```
Final testing result: 47.980%
```

MLP model is a class of feedforward artificial neural network [2]. It at least contains three layers of

nodes, including the input layer, the hidden layer and the output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes backpropagation for training. Its multiple layers and non-linear activation allow MLP to distinguish data that is not linearly separable, which stand out from the linear classification model. In result, the accuracy rise to 47.98%

3. Implement LeNet, one of the earliest CNN models. The structure is as follows. Please report the test result.

```
class LeNet(nn.Module):
    def __init__(self, num_classes=10):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, 16*5*5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
Final testing result: 60.670%
```

LeNet is a convolutional neural network structure [3]. LenNet contains 7 layers.

To prevent the input image from falling out of the boundary of convolution kernel, layer 1 is a convolution layer with six convolution kernels of 5x5 and the size of feature mapping is 28x28.

Layer 2 is a pooling layer. It outputs 6 feature graphs with size 14x14. Each cell in each feature map is connected to 2x2 neighborhoods in the corresponding feature map in layer 1.

Layer 3 is a convolution layer of 16 convolution kernels with size 5x5. The input of the first six feature maps is each continuous subset of the three feature maps in layer 2, the input of the next six feature maps comes from the input of the four continuous subsets, and the input of the next three feature maps comes from the four discontinuous subsets. Finally, the input for the last feature graph comes from all feature graphs of layer 2.

Layer 4 is similar to layer 2, but with size of 2x2 and output of 16 feature graphs of size 5x5.

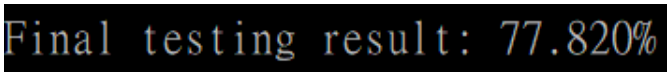
Layer 5 is a convolution layer with 120 convolution kernels of size 5x5. Each cell is connected to the 5x5 neighborhood on all 16 feature graphs of layer 4. As the feature graph size of layer 4 is also 5x5, layer 4 and layer 5 are completely connected. The output size of layer 5 is 1x1.

Layer 6 is fully connected to layer 5 to produce 84 feature graphs.

Lastly, there is an additional fully connected layer to map the dimension into 10 classes.

LeNet possesses the basic units of convolutional neural network, such as convolutional layer, pooling layer and full connection layer, as a representative of the early convolutional neural network, it further enhances the accuracy, the accuracy of LeNet is around 60.67%.

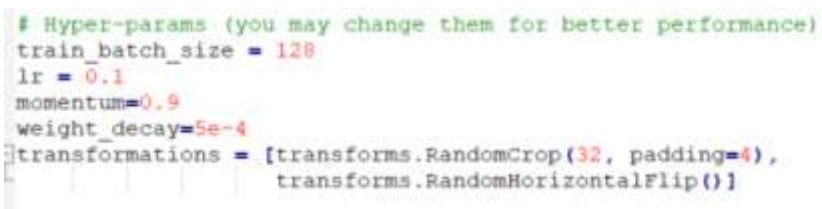
4. Implement AlexNet, the famous CNN model that marked the beginning of the deep learning era. Please report the test result.



AlexNet is the first GPU based CNN model [4]. It has 8 layers and uses the ReLu activation function. Deeper architecture with 8 layers allows it to better extract features when compared to LeNet. The ReLu activation function used in this network is unlike other activation functions. It does not limit the output, and which means there is not too much loss of features. It negates the negative output of summation of gradients and not the dataset itself. This means that it will further improve model training speed since not all perceptron are active. Therefore, the accuracy of AlexNet is the highest, it is around 77.82%.

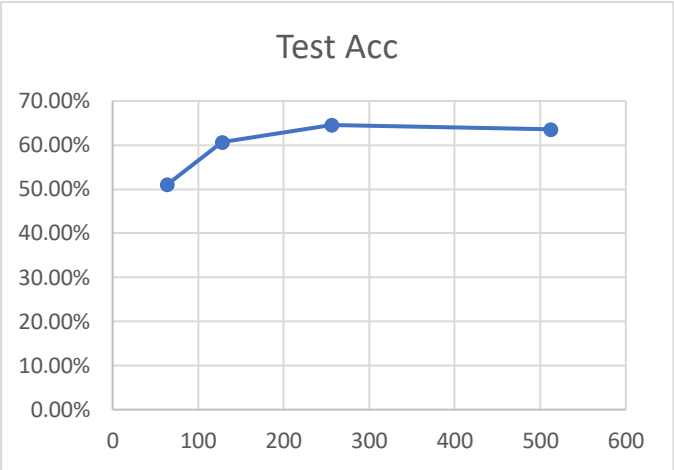
III. DISCUSSION

1. Based on LeNet, study the influence of hyper-parameters including the batch size, learning rate, momentum, weight decay, type of data augmentations.



We will change one hyper-param each time and others hyper-params will be remained unchanged, set as the default values as the figure above.

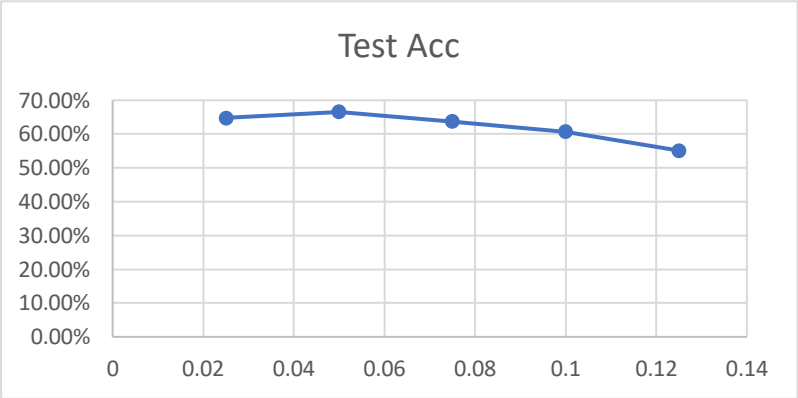
For train batch size:



Train batch size	Test Acc
64	51.07%
128	60.67%
256	64.55%
512	63.54%

We can see that under the default environment, larger train batch size seems to have better accuracy, it performs best when the batch size is 256.

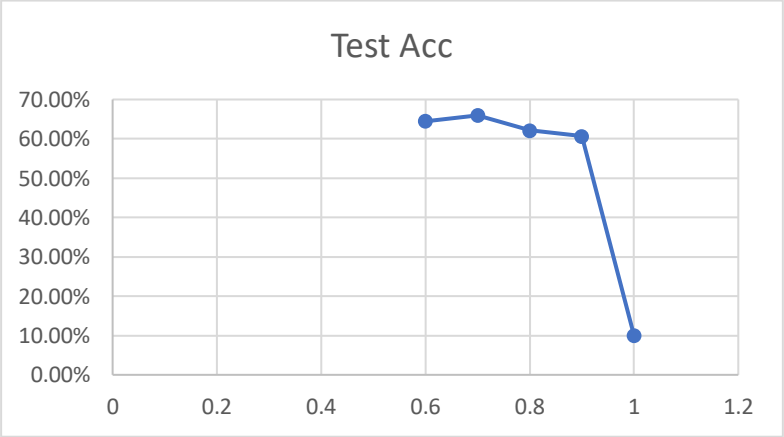
For learning rate:



lr	Test Acc
0.025	64.73%
0.05	66.52%
0.075	63.68%
0.1	60.67%
0.125	55.11%

With the default hyper-params, we found that the test accuracy is better when the learning rate is smaller, and when the learning rate is 0.05, it performs the best.

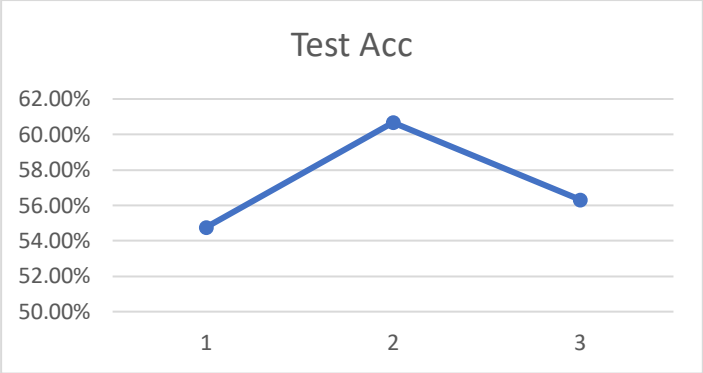
For momentum:



momentum	Test Acc
0.6	64.50%
0.7	65.96%
0.8	62.10%
0.9	60.67%
1	10.00%

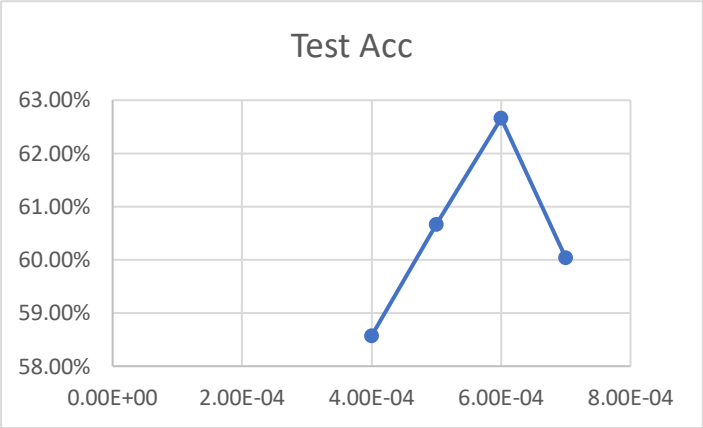
We found that keeping others hyper-params remain unchanged, the momentum performs better when the momentum is smaller, and if the value of momentum set as 1 will lead to significant drops in accuracy. It performs the best when the momentum is set to 0.7.

For weight decay:



Weight decay	Test Acc
5.00E-03	54.76%
5.00E-04	60.67%
5.00E-05	56.32%

We have test different values of weight decay, the result showed that the test accuracy is the best at E-04.



Weight decay	Test Acc
4.00E-04	58.57%
5.00E-04	60.67%
6.00E-04	62.66%
7.00E-04	60.04%

We then further test the accuracy of different E-04 value, we found that it performs the best when the weight decay is set as 6.00E-04.

For type of data augmentations:

transforms.RandomVerticalFlip()	51.79%
transforms.RandomHorizontalFlip()	60.67%

We have changed the default statement transforms.RandomHorizontalFlip() to transforms.RandomVerticalFlip() to see the different on accuracy, it drops from 60.67% to 51.79%. Therefore, we can say that using transforms.RandomHorizontalFlip() for data augmentation is a better choice.

[transforms.RandomCrop(32, padding=2), transforms.RandomHorizontalFlip()]	63.09%
[transforms.RandomCrop(32, padding=3), transforms.RandomHorizontalFlip()]	64.09%
[transforms.RandomCrop(32, padding=4), transforms.RandomHorizontalFlip()]	60.67%
[transforms.RandomCrop(32, padding=5), transforms.RandomHorizontalFlip()]	57.86%

Then, we have tested the performance if we keep both augmentations but with different padding. We found that smaller padding has a better accuracy. It performs the best when the

padding is set to 3.

2. Adjust the hyper-parameters mentioned above to achieve as high performance as possible. Note that number of epochs, the CNN model is not allowed to change.

```
train_batch_size = 256
lr = 0.05
momentum= 0.7
weight_decay= 5e-4
transformations = [transforms.RandomCrop(32, padding=3),
                   transforms.RandomHorizontalFlip()]
```

```
Final testing result: 63.770%
```

To achieve high performance, we change the hyper-parameters to the value that we tested to be the best performance in last part. However, the result is just 63.77%, which is similar to the original result. We can see the hyper-parameters will interact with each other, so putting these values together will not enhance the performance.

Therefore, we have consulted from research papers in the past.

We found that if the learning rate is high, the performance of lager batch size is better than smaller batch size, but for better train we should use small batch size with low learning rate [5]. Also, the number of batch sizes should be a power of 2 to take full advantage of the GPUs processing.

When setting learning rate, if the learning rate is low, then training is more reliable, but optimization will take a lot of time because steps towards the minimum of the loss function are tiny. If the learning rate is high, then training may not converge or even diverge. Weight changes can be so big that the optimizer overshoots the minimum and makes the loss worse [6].

Learning is slow if the learning rate is set too low. Gradient may be steep in some directions but shallow in others. Therefore, we can add a momentum term α [7]. Typical value for α is 0.5. If the direction of the gradient remains constant, the algorithm will take increasingly large steps.

Weight decay is to prevents the weights from growing too large and is gradient descent on a quadratic regularization term [8]. The weights are multiplied by a factor slightly less than 1 after each update.

For data augmentations, other than transforms.RandomCrop, there are transforms.CenterCrop, transforms.FiveCrop and transforms.RandomResizedCrop etc. [9]. We can enhance the accuracy by applying suitable data augmentations.

Finally, we combine all these findings and build the LeNet as follow.

```
# Hyper-params (you may change them for better performance)
train_batch_size = 2
lr = 0.001
momentum= 0.5
weight_decay= 5e-4
transformations = [transforms.CenterCrop(32),
                   transforms.RandomHorizontalFlip(0.3)]
```

```
Final testing result: 68.200%
```

IV. SUMMARY

In conclusion, when solving Cifar-10 image classification task, the first step is to choose a suitable neural network. If the data set is linear you may simply use a linear classification model. However, when you want a better accuracy on difficult cases, you may apply the MLP model, LeNet or AlexNet. MLP, its multiple layers and non-linear activation allow it to distinguish data that is not linearly separable. LeNet is a simple neural network. It works well for character recognition images. AlexNet is a deeper architecture with 8 layers which means that is better able to extract features when compared to LeNet. It also worked well for the time with color images. We can enhance the accuracy by applying a more advanced network.

Besides, other than using more advanced model, we can enhance the accuracy simply by using suitable hyper-parameters. Firstly, the number of batch sizes should be a power of 2 to take full advantage of the GPUs processing. Secondly, we should use small batch size with low learning rate for better training. Thirdly, we should use a low learning rate, then training is more reliable. Also, adding a momentum can fasten the learning. By keeping the direction of the gradient remain constant, the algorithm will take increasingly large steps. In addition, weight decay can prevent the weights from growing too large and is gradient descent on a quadratic regularization term. Finally, we can enhance the accuracy by applying suitable data augmentations.

V. Reference

[1] Jun-ya Gotoh and Akiko Takeda, “A Linear Classification Model Based on Conditional Geometric Score”, INSTITUTE OF POLICY AND PLANNING SCIENCES [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/38628461/1096.pdf?1441070823=&response-content-disposition=inline%3B+filename%3DA_Linear_Classification_Model_Based_on_C.pdf&Expires=1608376122&Signature=CYrUpKcyNC~KgvnZQ2qyfxdS0zk3R~3VmOm1oeDKAkdArUCfAoO3ofQQ1JviHkgcHB3o2lmuntcKDAajuqY9AnJ8i4WiOvtW8Pwe86bPpz6q4p9bZXGm4I~yxfnNaCfW3XL59quPsdvNEC7SongCOISNexKeYw~F1cfKUKiCO1fbS5Sk6x5m0OdDxW7sKiQPRr3I9CQxq-zIy4C7HywITWxWhzX6vhgf-B8eM0EBd0G5SNyt8-JiEcP91U8R6CfVPREmpzPpGXyb5WrRkSPIQft9v6AOMIV5fdN72wxaFTu05ZLOo3FfohfwcsoEJTW8mFyiKHF-wI3ZijTqPC85uQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. [Accessed Dec.19, 2020].

[2] Jason Brownlee, “When to Use MLP, CNN, and RNN Neural Networks”, Machine Learning Mastery [Online]. Available: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/. [Accessed Dec.19, 2020].

[3] Richmond Alake, “Understanding and Implementing LeNet-5 CNN Architecture (Deep Learning)”, Towards Data Science [Online]. Available: https://towardsdatascience.com/understanding-and-implementing-lenet-5-cnn-architecture-deep-learning-a2d531ebc342. [Accessed Dec.19, 2020].

[4] Tejas Mohan Ayyar, “A practical experiment for comparing LeNet, AlexNet, VGG and ResNet models with their advantages and disadvantages.”, Medium [Online]. Available: https://tejasmohanayyar.medium.com/a-practical-experiment-for-comparing-lenet-alexnet-vgg-and-resnet-models-with-their-advantages-d932fb7c7d17. [Accessed Dec.19, 2020].

[5] Ibrahem Kandel, “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset.”, Science Direct [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959519303455. [Accessed Dec.19, 2020].

[6] Pavel Surmenok, “Estimating an Optimal Learning Rate For a Deep Neural Network”, Towards Data Science [Online]. Available: https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0. [Accessed Dec.19, 2020].

[7] Lecture 10(b) Training Networks

[8] Metacademy, “weight decay in neural networks”, Metacademy [Online]. Available: https://metacademy.org/graphs/concepts/weight_decay_neural_networks. [Accessed Dec.19, 2020].

[9] pytorch, “TORCHVISION.TRANSFORMS”, pytorch [Online]. Available: https://pytorch.org/docs/stable/torchvision/transforms.html. [Accessed Dec.19, 2020].