CS284 PA3
MiniGoogLeNet

Rhodel R. Quizon (2012-31254)

Date of Submission: Dec, 9 2023

1. INTRODUCTION

Convolutional Neural Networks (CNN) are first proposed in 1998 in a study by Yann LeCun, et. al[1] . In their study, they proposed a CNN architecture called LeNet-5. It is composed of several layers of convolutions, poolings, and fully connected layers. They discussed that better pattern recognition system can be built using automatic learning and less on hand-designed features. Instead of using different modules including field extraction, segmentation, recognition, and language modeling, neural network can automate learning and give quality outputs.

Since then, CNNs have not improved much except for additonal layers, additional tricks and faster hardware. One of the recent CNNs is GoogLenet in 2014[2]. It is a 22 layers deep network which is designed for context of classification and detection. In this paper they proposed a new level of network organization called "Inception Modules". These modules somehow increase the dept of the network and increase computational efficiency.

In 2017 a paper by Chiyuan Zhang et al [3], used a network based on GoogLeNet's inception modules. The paper study uses ther inception network on CIFAR10 and ImageNet datasets and observed how different regularization techniques affect the performance of their model.

2. OBJECTIVES

The objective of this programming assignment is to attempt to recreate the inception model described in experiments in [3] and investigate the effects of dropouts. Specifically, this programming assignment aims to achieve the following:

1. Recreate the inception model described in [3] using pytorch.

2. Experiment on the dropout settings.

3. Investigating how dropout affects the performance of the network.

4. Evaluate the performance of the network on test set with different dropout settings

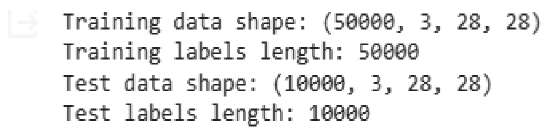
3. METHODOLOGY

Data Preparation and pre-processing

The CIFAR-10 Dataset is contained by 32x32 colour images in 10 classes with 6000 images per class. The dataset from University of Toronto already divided the dataset to training and testing. There are 50000 training data and 10000 testing data. The 10 classes are airplane, automobile, bird, cat, deer ,dog ,frog, horse, ship, truck.

The reference paper used for this programming assignment processed the image to be on dimension 28x28x3 before feeding to their neural network. Also, the mean and standard deviation of each image were computed and subtracted and divided to the image respectively. These steps were also done in this assignment.

Numpy was used for slicing and computing the mean and standard deviation. The operation was done per image as indicated in the paper. Additionally, the dimensions of the data are change to match Pytorch's image input. From height x width x channels, the dimension was changed to channels x height x width. The shape of the training and testing datasets are checked using python. Figure 1 shows the final shape of the training and testing dataset.



```
Training data shape: (50000, 3, 28, 28)
Training labels length: 50000
Test data shape: (10000, 3, 28, 28)
Test labels length: 10000
```

Figure 1: Final shape of processed dataset

The images before and after the processing was also checked. Figure 2 shows how the image was processed and how it emphasized certain features.

Take note that matploltib was used to visualize the images, this could casue the processed image in the visualization to be black or white as the processed image has value more than 1 and less than 0. It can be observed that the frog in the original image is distinguishable of a frog, the preprocessing somehow amplifies the values of features on the outline shape of the frog. This preprocessing could hopefully help the network on which features to emphasize.

Model Development

The model to be used for this assignment is also based on the design in the reference paper. The paper has three main

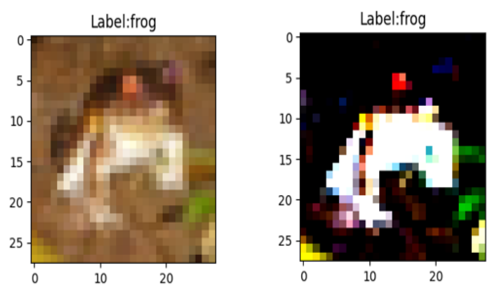


Figure 2: Comparison of original image vs preprocessed images

modules in their paper: Conv Module, Inception Mdule, and Downsample Module. Figure three shows the three main modules as described in the paper.

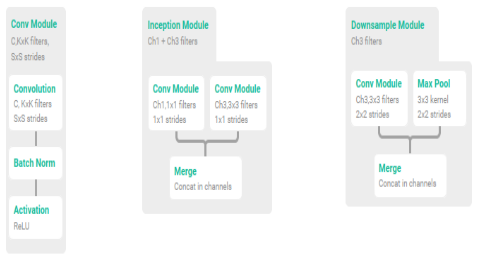


Figure 3: Main modules used in the reference paper

The specification of each module is described by the full model illustrated in Figure 4. The required channels, filter size, and strides are indicated in the full model.

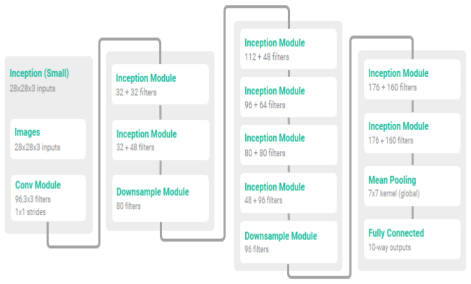


Figure 4: Full Model reference for this programming assignment

The Conv Module is constructed with using nn.Conv2d, nn.Batchnorm2d, and nn.ReLU. The input channel and output channel of the convolution layer is indicated by the dimensions of the input image and the required output channel indicated in the paper. The output of the nn.Conv2d is used as input to the nn.Batchnorm2d layer, and is then the output of batch normalization is activated using ReLU function.

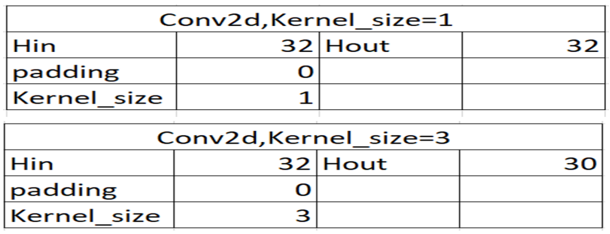
The Inception Module is composed of two Conv Modules with the same input, one with 3x3 filter while one has 1x1 filter. These two modules both have 1x1 stride. The outputs of the two modules are concatenated in the channel dimensions. During the development of this module, an error was encountered as there are dimension mismatch for concatenating output of two convolution layers with different stride. This is handled by adding a padding on the Conv Module with 3x3 strides. The checking was done with Conv2d documentation from pytorch website. See figure 5 for the Hout and Wout equation for nn.Conv2d from pytorch documentation.

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Figure 5: Computation of conv2d Hout and Wout

Figure 6 shows the comparison for an example of sizing difference when kernel size of 1 and 3 are used respectively.

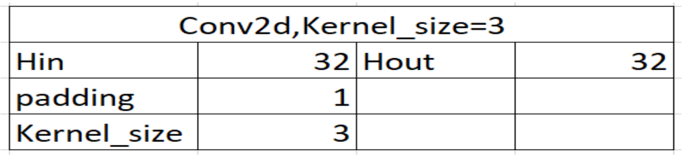


Conv2d,Kernel_size=1			
Hin	32	Hout	32
padding	0		
Kernel_size	1		

Conv2d,Kernel_size=3			
Hin	32	Hout	30
padding	0		
Kernel_size	3		

Figure 6: Kernel Sizing comparison when filter is equal to 1 and both has no padding

Adding a padding equal to 1 is able to handle the error and fix the shape mishandling. Figure 7 shows the sample calculation when a padding of 1 is added to 3x3 filter, it can be observed that output channel is now equal to the output channel of 1x1 filter.



Conv2d,Kernel_size=3			
Hin	32	Hout	32
padding	1		
Kernel_size	3		

Figure 7: Sample computation of reaching similar dimensions with 1x1 filter with 3x3 filter with padding of 1

The Downsample module uses the Conv Module and Max pooling layer. For the max pool layer, nn.Maxpool2d layer is used. The parameters for both Conv Module and the Maxpool layer are the same with kernel size of 3 and stride of 2. The outputs of the two layers are concatenated at the channels.

After designing the modules, to properly expect the full size of each layer a sample run through with a test image is done. Each layer is added one by one and the output dimensions are checked to match the input dimension for the next layer. This is done for easier troubleshooting before designing the model the nn.Module class. Table 1 shows that the sample output shape for each layer output in the tested network.

Layer Name	Output Shape
Conv Module(96,3x3 filters,1x1 stride)	1,96,26,26
Inception Module (32+32 filters)	1,64,26,26
Dropout	1,64,26,26
Inception Module (32+48 filters)	1,80,26,26
Dropout	1,80,26,26
Downsample Module (80 filters)	1,160,12,12
Inception Module (112+48 filters)	1,160,12,12
Dropout	1,160,12,12
Inception Module (80+80 filters)	1,160,12,12
Dropout	1,160,12,12
Inception Module (48+96)	1,144,12,12
Dropout	1,144,12,12
Downsample Module (96 filters)	1,240,5,5
Dropout	1,240,5,5
Inception Module (176+160 filters)	1,336,5,5
Dropout	1,336,5,5
Inception Module (176+160 filters)	1,336,5,5
Dropout	1,336,5,5
Mean Pooling (5x5 Kernel Global Pooling)	1,336,1,1
Flatten Layer	1,336
Fully Connected Output (10 ways outputs)	1,10

Table 1: Tabulated layer of my model implementation with output shapes

The full network model has an input p for probability of dropout.

Preparation of training

In preparation of training a functions for accuracy computation, training loop, and testing loop was made. This helps in better debugging and running of code and changing of parameters. Additionally, optimizer and scheduler are set based on the programming assignment requirements for hyperparameters (Epochs=80, Batch size 16), Stochastic Gradient Descent as an optimizer. The loss function used Cross Entropy Loss.

The accuracy function computes the accuracy of the prediction per batch. The accuracy per batch are added together and is then divided by the total number of batches. This will be the accuracy displayed per epoch.

The scheduler used a starting factor of 1 and end factor of 0.3 this with iteration of 80. The scheduler steps every at the end of each epoch. This means that that the the learning rate will linearly decay from 0.01 to 0.003 in course of 80 epochs. During training, training loss, training accuracy, testing loss, and testing accuracy are displayed to observe the learning curve of the model on both training and testing dataset.

The model designed was tested on different drop out rates and performance of each settings was analyzed. The confusion matrix was plotted using mlxtend and torchmetrics.

The platform used for training and testing the model is Google Colab with T4 GPU which is the usual GPU given for free account.

4. EXPERIMENTAL RESULTS

After the data preprocessing, model design, and preparation of hyperparameters and other requirements, the model was trained and tested on no dropout settings and with dropout (0.1,0.3,0.5,0.7, and 0.9). Figure 8 shows the training loss and training accuracy while Figure 9 shows the testing loss and testing accuracy.

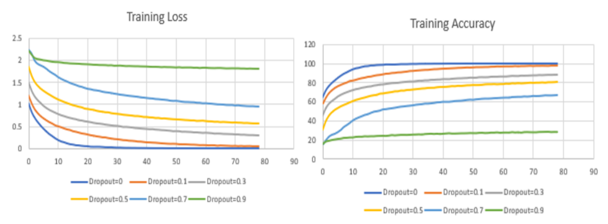


Figure 8: Training Accuracy and Training Losses

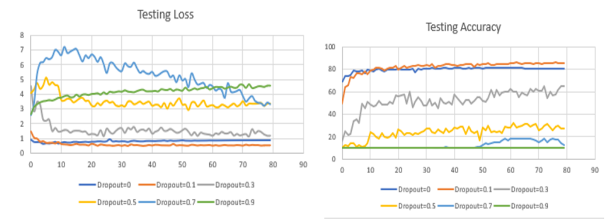


Figure 9: Testing Accuracy and Testing Losses

Figure 10 shows the confusion matrix for different dropout settings using the testing dataset.

Table 2 shows the testing time with different dropout settings. There is no observed significance in different among the training time with the resources used in this programming assignment.

Dropout settings	Time(seconds)
p=0.0	4199.158
p=0.1	4425.854
p=0.3	4394.714
p=0.5	4255.181
p=0.7	4356.457
p=0.9	4212.999

Table 2: Training time for each dropout settings

5. ANALYSIS AND DISCUSSION OF RESULTS

Training Loss and Training Accuracy

Figure 8 shows the figure training loss and training accuracy for all settings. It can be noticed that training loss for no

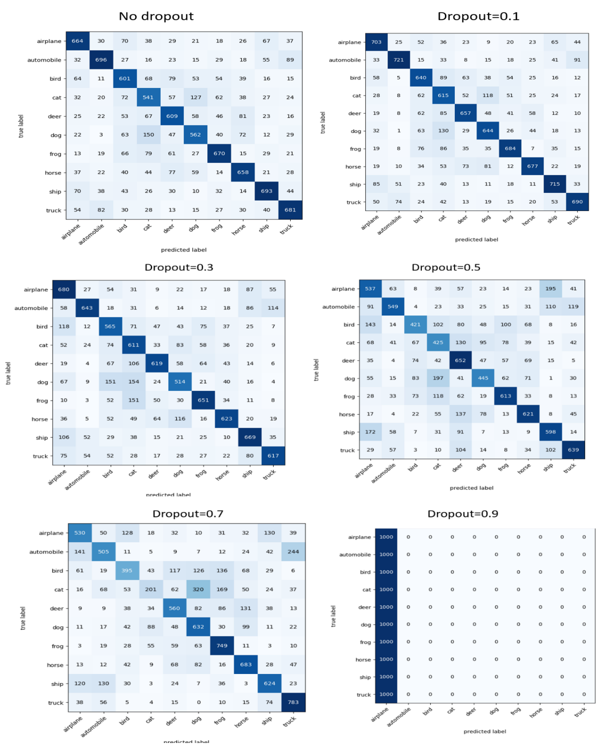


Figure 10: Confusion Matrix for different dropout settings using the testing dataset

dropout and dropout=0.1 approaches to 0 within 80 epochs. The difficulty in convergence could be due to the number of nodes being dropped out. With higher probability of nodes being dropped out, it is more difficult for the network to learn any features. It can be seen that the network is still learning but very slow for dropout probability equal to 0.3,0.5, and 0.7 while dropout probability equal to 0.9 seems to never learn at all within 80 epochs.

The same behavior can be observed from the training accuracy. The training accuracy for both no dropout and with drop out probability of 0.1 approaches training accuracy of almost 100%. The model with no dropout converges faster than with dropout setting of 0.1 as learning was slowed down by the dropping out of nodes but as the end of 80 epochs is reached they almost have the same accuracy with dropout setting down by a little bit. The rest of the dropout settings seems to need more epochs to reach higher training accuracy except for dropout of 0.9 which seems to have not improved accuracy over 80 epochs. This is mostly because of too much nodes are dropped causing inability to learn.

Testing Loss and Testing Accuracy

Figure 9 shows the testing accuracy and testing losses. Again in this scenario, the testing losses of both no dropout and dropout of 0.1 is lower than the rest. The interesting observation is that dropout probability of 0.1 is lower than no dropout. The dropout setting of 0.1 somehow shows lower calculation than with no dropout. The rest of the dropout settings are somehow slow in learning in the course of 80 epochs especially 0.5,0.07, and 0.0 dropout settings. Dropout setting of 0.3 looks like it can still improve by adding more epoch based on its trend. Compared to no dropout and dropout setting of 0.1 all other dropout settings have slow learning curves.

Testing accuracy also shows the same trend, the dropout setting of 0.1 shows improved accuracy on unseen dataset, despite it having lower accuracy in training. This is nodes forced to learn complex features in the absence of other nodes. Dropout of 0.1 shows an accuracy of 85.79% on the final epoch while no dropout only has 80.8%. This is an almost 5% improvement. The rest of the dropout settings is not nearing these two settings in the span of 80 epochs. Dropout setting of 0.9 did not seem to improve anything correctly at 80 epochs while 0.7 has just started improving at around 50th epoch. Again, dropout setting of 0.3 looks like it could still learn given some more epochs.

Confusion Matrix

Figure 10 shows the Confusion on the test dataset for different dropout settings. At first glance we can see that the same trend as discussed in the training and testing loss. Dropout setting with probability of 0.1 shows great improvement on all classes compared to other settings. There are some instances in which the dropout probability of 0.7 has surpassed the number of true positives for truck and frog. It has 783 and 749 true positives respectively while dropout probability of 0.1 only has 663 and 674 in these classes. Table 3 shows the computation for accuracy using the confusion matrix. This accuracy metrics is computed by adding all the true positives (diagonal values) in the matrix and dividing it by the total number of test dataset.

Dropout settings	Accuracy
p=0.0	63.75%
p=0.1	67.46%
p=0.3	61.92%
p=0.5	55.00%
p=0.7	56.62%
p=0.9	10.00%

Table 3: Accuracy computed for the Confusion Matrix

The trend of accuracy values from the confusion matrix matches what is observed during training. The highest accuracy is still with dropout probability of 0.1, then the accuracy deteriorates as higher probability of dropout is increased. This is discussed in the previous

Figure 11 shows the precision and recall values calculated from the confusion matrix. Precision is computed as the true positives divided by the sum of the true positives and the false positives while recall is computed as the true positives divided by the sum of true positives and false negatives.

It can be observed that in precision parameter dropout probability of 0.1 surpasses almost all classes except the deer class in which the dropout probability of 0.3 is more precise. This could mean that dropout probability of 0.3 has only learned the features of a deer on the given hyperpa-

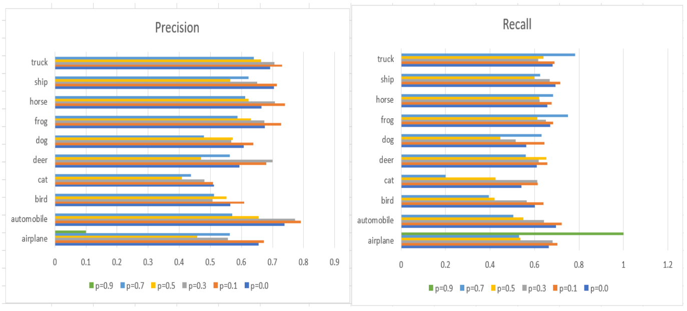


Figure 11: Precision and recall comparison across classes and dropout settings

rameters. Given additional epochs and some adjustment on learning rate maybe performance of this dropout setting could be improved. Precision parameters basically measures how many of the predicted into a certain class actually belongs to a class. Higher precision means there are less false positives and more true positives.

For the recall parameter, most classes again is surpassed by dropout setting of 0.1 except for some classes. The frog and truck classes are highest with dropout setting of 0.7. It can be recalled from the previous section that during this dropout setting highest of true positives for frog and truck are observed. This means that the dropout setting of 0.7 is very good in identifying which is a frog and a truck and which are not. Higher recall means there are fewer false negatives which means the model is able to classify if a dataset does not belong to that particular class. A recall value of 1 is seen for dropout probability of 0.1, this is because it classified all dataset to airplane thus not having any false negatives.

6. CONCLUSION

From the results, it can observed that with proper dropout settings, an improvement on the performance of the neural network is achieved. With the appropriate dropout settings, a model can perform very well even on testing despite the no dropout model converging faster. The slowing down of convergence paves way to more opportunities for adjustment of parameters thus having a more robust model even on unseen data. Also, it can be observed from this programming assignment that too high of dropout probability causes very slow convergence and at worst case the model does not learn at all since per iteration there are very few nodes left to learn complex features. When setting dropout parameters, it is necessary to judge and observe appropriately if it is worth it to train a model given time and available resources.

7. REFERENCES

[1]Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998, doi: https://doi.org/10.1109/5.726791.

[2]C. Szegedy et al., “Going Deeper with Convolutions,” arXiv (Cornell University), Sep. 2014, doi: https://doi.org/10.48550/arxiv.1409.4842.

[3]C. Zhang, Samy Bengio, M. Hardt, B. Recht, and Oriol Vinyals, “Understanding deep learning requires rethinking generalization,” arXiv (Cornell University), Jan. 2017, doi: https://doi.org/10.48550/arxiv.1611.03530.