CSE 676 – Deep Learning
Project – 01

Bharat Sesham
Department of Computer Science
University at Buffalo, SUNY
Buffalo, NY 14214
bharatse@buffalo.edu
bharatse – 50338269

Abstract

An introductory project to explore different deep learning architectures and compare their learning and validation accuracy by varying discrete parameter settings like optimizer choice (SGD or ADAM) and regularization methods (no regularization, batch normalization or dropout).

1 Introduction

This project is mainly done to explore various deep neural network architectures used in classifying images. Also, a choice of various other parameters used during modeling the networks such as regularization parameters, weight decay methods, choice of activation functions, and optimizer selection, are explored. Some of the other additions which are implemented in this project are Tensorboard, early stopping and layer weights initialization. The architectures that are implemented in this project are VGG16, ResNet18 and Inception v2 for which the accuracy, precision and recall are calculated for various settings (like change in optimizer and regularization methods). In section 2, the details regarding the hypermeter selection for all the settings are mentioned. In Section 3, an overview of these implemented architectures is discussed. In Section 4, different plots related to the network training and validation sets are presented and explained. Finally, in Section 5 the table with the values of accuracy, recall and precision for different settings are presented.

By the end of this project, we wish to compare how different parameters effect the model training and validation accuracy. Also, this project helped me in gaining hands-on experience over modelling different architectures and implementing them in code. The comparison is useful as it lets us have practical experience while choosing different methods and see real-time changes in accuracy and training. This will allow us have to have a informed decision making in future projects when we are trying to implement a new network.

2 Implementation of the Architectures

The value of hyperparameters plays a crucial role on the model learning and validation accuracy. In this section, I will be defining the hyperparameter values that I have used across the three architectures most of which are kept constant to check their impartial effect on the model architecture and training. However, there are also some minor variations from one architecture to an another which are also stated below.

2.1 Hyperparameters

1. Optimizer: ADAM or SGD
   i. ADAM Parameters:
      - Learning Rate: 0.001
      - Weight Decay: 5e-4
   ii. SGD:
      - Learning Rate: 0.1
      - Weight Decay: 5e-4
2. Learning Rate Decay:
   - Method: MultiStepLR
   - Gamma: 0.02
   - Milestones: 50, 100, 150
3. Batch size: 256 or SGD
4. Dataset: CIFAR-100
5. Early Stopping:
   - Patience: 20
   - Minimum epoch: 150
6. Regularization Methods:
   - No Regularization.
   - Batch Normalization.
   - Dropout
7. Number of epochs: 200
8. Weight Initialization: True
9. Image size: 32x32.
10. Activation Functions:
    - ReLU.
    - SoftMax.

For every setting, the model weights with the least loss is saved along with the Tensorboard generated data consisting of validation accuracy, validation loss, train accuracy, and train loss are stored to visualize the training of the model. Finally, the hyperparameters that are changed from the initial defined values mentioned below along with the setting:

1. In Inception v2,
   - Image size is changed to 96x96.
   - Learning rate for SGD is changed to 0.01 (for all SGD settings).
2. In VGG16,
   - The ADAM optimizer learning rate is changed to 0.0005 in No Regularization settings and 0.0001 in Dropout setting.
   - For Batch Normalization setting the init_weights is set False.

Some data augmentation methods such as Normalize, RandomHorizontalFlip RandomCrop, and RandomRotate are also used on the data. Gradient Clipping is also implemented to deal with any exploding gradients.

3 Architectures

In this section, I will be discussing the three different architectures VGG16, Resnet18 and InceptionNet v2 that are implemented in this project and detail any changes made from the original implementation.

3.1 VGG16

VGG16 is a convolutional neural network architecture named after the Visual Geometry Group from Oxford by its authors K. Simonyan and A. Zisserman in 2014. The model is designed with multiple convolution layers followed by two fully connected layers with ReLU activations. There are also some max pool layers in between the convolution layers. In batch normalization setting, the batch normalization is added in between convolution layers and in case of dropout setting, the dropout layer is added in between the fully connected layers. The image is passed through a stack of convolution layers where filter with very small receptive field (3x3) is used. VGG16 is relatively simple compared to the other two models.

3.1 ResNet18

ResNet18, short for Residual Network was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun and was the winner of ImageNet challenge in 2015. The breakthrough with ResNet was it allowed us to successfully train deeper neural network with 150+ layers. Before ResNet, deep neural network used to face problem dealing with vanishing gradients. This is because the shallow layers are not able to propagate information effectively to the deeper layers. ResNet solved this problem by connecting shallow layers directly to deeper layers (called shortcut or skip connections). ResNet is designed with multiple block, these can either be Basic Block which is predominantly used in simpler ResNet models or Bottleneck Block which are generally used in deeper versions.

In my implementation, I have only used the Basic Block which has two convolution layers followed by skip connection. In case of batch normalization or dropout setting, there are additional batch normalization or dropout layers in between the convolution layers present in the Basic Block. The overall model has a initial convolution layer followed by a maxpool layer, four Basic Blocks and a fully connected layer in the end, all with ReLU activation.

3.1 Inception v2

InceptionNetv2 was introduced Christian Szegedy, Vincent Vanhoucke and Sergey Ioffe in 2014. My implementation has three types of inception layers and has the same structure like in the paper. The inception module act as a multi-level feature extractor by computing 1x1, 3x3, and 5x5 convolution within the module and the output of these filters are stacked before being fed into the next layer in the network. Initially there are six convolution layers with

max pool layer which are followed by the multiple inception modules (all three types as defined in the paper), adaptive pooling and fully connected layers.

I have defined the inception module defined in the figure5, figure6 and figure7 of the paper as InceptionF5, InceptionF6 and Inception F7 respectively. I haven't modified the architecture much from the original paper [1]. In the code, comments are added at each step indicating what operation are being carried out from the figure. Using multiple features from these multiple filters improve the performance of the network and by using 1x1 convolution, the inception block is performing cross-channel correlations, ignoring the spatial dimensions. The only changes made were in the regularization sections, where batch normalization or dropout is added when called in these setting condition.

4 Plots

For each model architecture, eight graphs (training loss, accuracy and validation loss, accuracy) divided into two subplots (each subplot is based on the optimizer) are presented. Also, each graph contains three different curves for each of the regularization method used to conduct the experiment.

4.1 VGG16

In the VGG16 network with both SGD and Adam optimizer, it can be observed that the training accuracy has reached maximum (overfitted) before the minimum early stopping epoch. Among the three regularization settings, batch normalization has performed better than dropout and no regularization. The adaptive learning rate also works as expected as we can see the learning jumps at momentum points. It can also be observed that the loss for the validation set started to increase after ~65 epoch before early stopping for the SGD optimizer setting. Also, in the Adam optimizer setting, we can see the effect of learning rate (0.0005) for no regularization and (0.0001) for dropout settings in the graph, and both of the validation accuracy is almost similar.

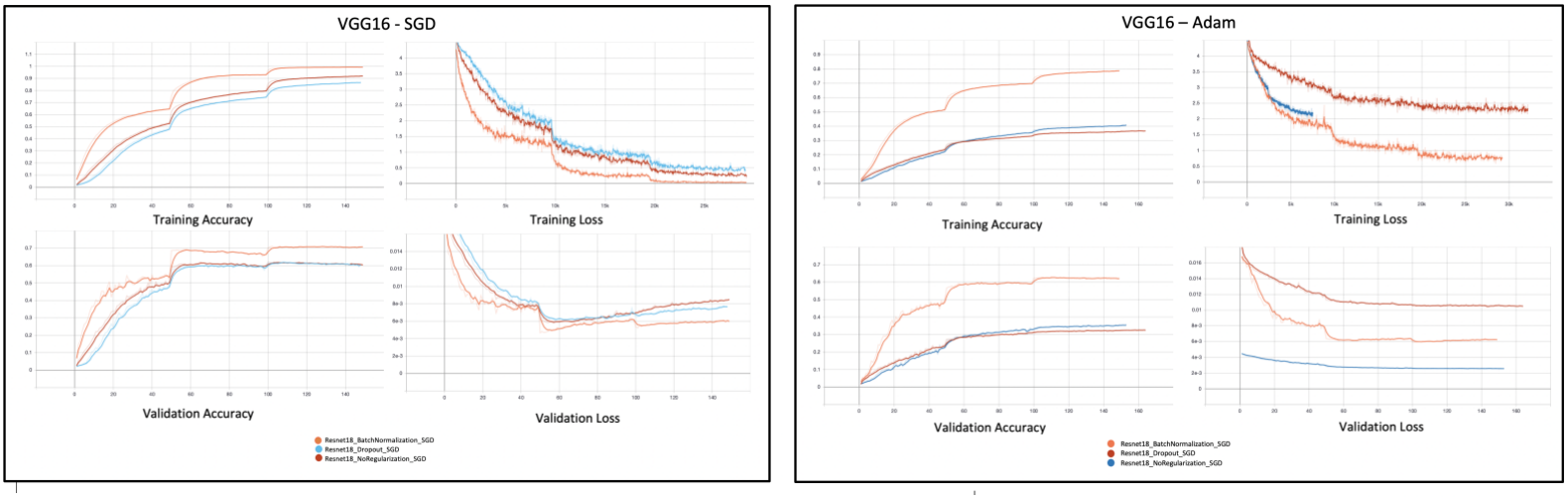

Figure 1: Training loss, accuracy and validation loss, accuracy for VGG16 architecture for SDG and ADAM optimizer respectively.

4.2 Resnet18

In the Resnet18 network, it can again be observed that the batch normalization has performed better than dropout and no regularization settings with both SGD and Adam optimizers. Also, with SGD optimizer between dropout and no regularization, dropout performs a bit better than no regularization setting. However, in the Adam setting, both of them almost have similar accuracy and not much difference is observed even after adding many dropout layers at every place of batch normalization (unlike with other architectures where the dropout is only added before the fully connected layers in the end). Also, we can see that the loss increased drastically for the no regularization setting with SGD optimizer, from which we can infer that even though the accuracy is same for both no regularization and dropout setting, the dropout regularization method helped in keeping the validation loss under control. So, we can say that the presence of a regularization method is important for model to generalize well on unseen data.

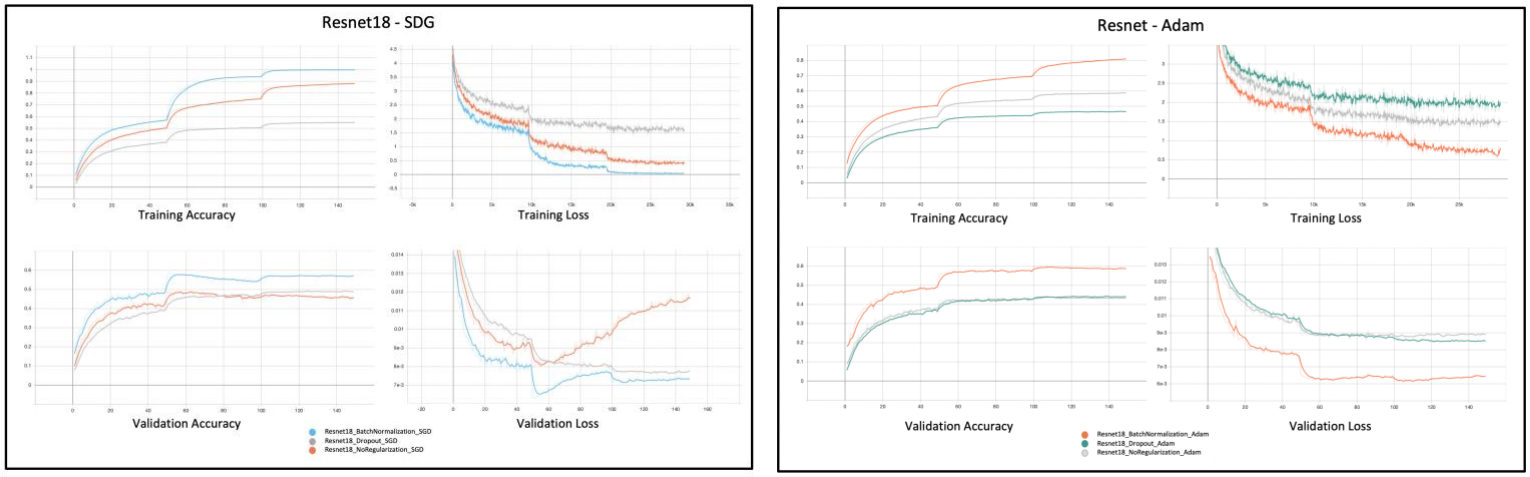

Figure 2: Training loss, accuracy and validation loss, accuracy for Resnet18 architecture for SDG and ADAM optimizer respectively.

4.3 InceptionNet v2

In the InceptionNetv2 network, for both SGD and Adam settings, again batch normalization performed far better than no regularization and dropout methods. Similar to Resnet18, we can see dropout regularization alone does not improve the accuracy much as compared with no regularization setting. Also, we can see that the training data overfitted and training loss almost reached zero. We can also see the effect of learning rate decay method MultiStepLR at the momentum point 50, 100. Finally, in the ADAM optimizer, no regularization and dropout settings trained a bit more than the minimum early stopping epoch with a patience of 20.

Also, among the three architectures, Inception v2 got the highest accuracy on the validation set. This higher accuracy can be the result of the deeper model architecture of InceptionNet v2.

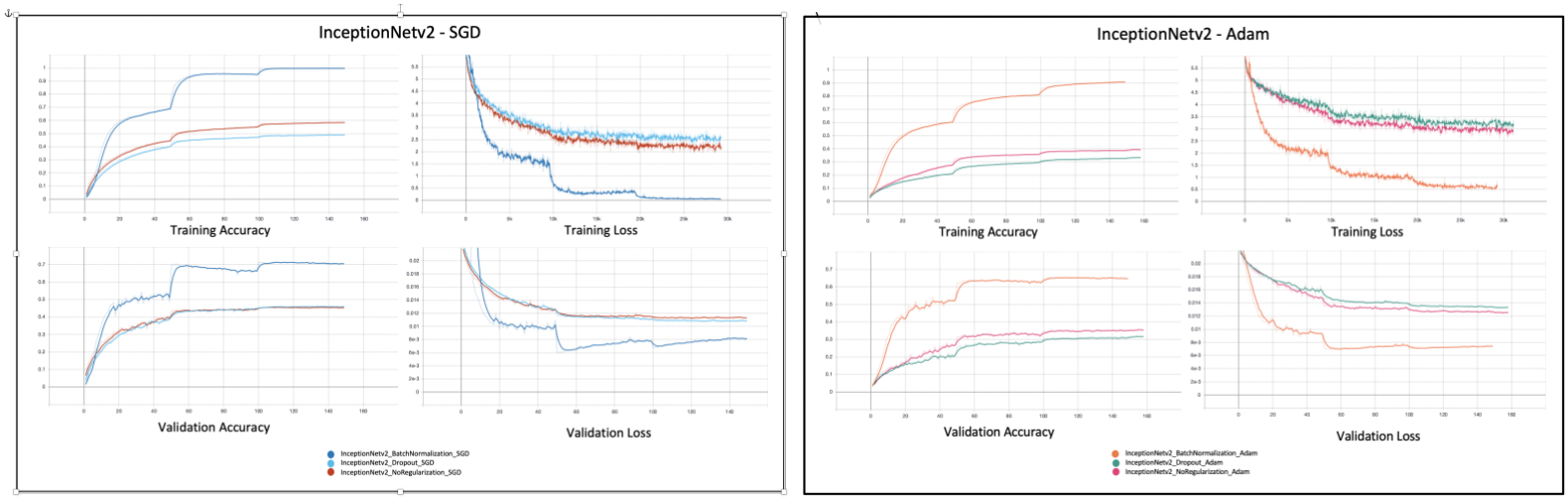

Figure 3: Training loss, accuracy and validation loss, accuracy for InceptionNet v2 architecture for SDG and ADAM optimizer respectively.

5 Results

	Arch	VGG16			Resnet18			Inception v2		
Optimizer	Score / Setting	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
SGD	With Batch Norm	**0.685**	0.695	0.685	0.582	0.585	0.582	**0.695**	0.705	0.695
	With Dropout	0.593	0.605	0.593	0.491	0.497	0.491	0.460	0.455	0.460
	No Regularization	0.609	0.619	0.609	0.486	0.505	0.486	0.455	0.451	0.455
ADAM	With Batch Norm	0.623	0.624	0.623	**0.593**	0.593	0.593	0.636	0.638	0.636
	With Dropout	0.323	0.458	0.323	0.438	0.458	0.438	0.315	0.289	0.315
	No Regularization	0.349	0.374	0.349	0.435	0.453	0.435	0.357	0.347	0.357

Table 1: Experimental results of VGG16, Resnet18, and Inception Net v2 architectures.

Out of three networks, InceptionNetv2 with Batch Normalization regularization and SGD optimizer performed best and got an accuracy of 69.5 on the validation set. Also, VGG16 with SGD and Batch Normalization came close to InceptionNet v2 with an validation accuracy of 68.5. These are followed by Resnet18 with Adam optimizer and Batch Normalization with an accuracy of 59.3.

6 Conclusion

This project was very helpful as it helped me in understanding the importance of various concepts like regularization function, optimizer selection and effect of learning rate on the training, and importance of weight initialization for various layers. Along with this, I was able to explore new concepts like Tensorboard to visualize the training and validation learning. It can also be seen that InceptionNetv2 performed best out of the three followed by VGG16 and Resnet18 respectively. Also, Batch Normalization was giving the best performance out of three regularization settings. So, a deeper model with suitable regularization can generalize well on the classification task.

References

[1] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, ‘‘Rethinking the inception architecture for computer vision,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2016, pp. 2818–2826.

[2] K. He, X. Zhang, S. Ren, and J. Sun, ‘‘Deep residual learning for image recognition,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2016, pp. 770–778.

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015

[4] Vision: Inception

[5] Pytorch-CIFAR100

[6] CIFAR-100

[7] Vision: VGG

[8] Pytorch Documentation

[9] Github: Research-Paper-Summary

[10] Vision: ResNet