Introduction to Deep Learning Neural Networks

Christopher Sam Roy
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14214
croy2@buffalo.edu

Abstract

The objective of this project was to analyze and compare the effectiveness and computational differences between three common Deep Learning architectures used for feature extraction from given input images. The model architecture related to VGGNet, ResNet and InceptionNet were trained and analyzed on a common dataset i.e. CIFAR-100 to infer the differences in terms of memory usage and computation cost.

1 Introduction

Deep learning being a research-intensive area, new architectures are being discovered each year such that they perform better than the models of the previous years. The models are continuously being updated with new versions that have a slightly altered architecture yet provide computational or learning improvements. This project deals with such an approach of analyzing different models and their versions on the same dataset, which makes it easier to draw direct comparisons between their architectures. For example, residual networks such as ResNet was an improvement over VGGNet (which stacked convolution layers and pooling layers to go deeper and deeper) such that the model had an extra path that could skip some convolution layers in a way preserving the input integrity for an even deeper layer's feature extraction. This problem was identified when researchers tried building deeper models using VGGNet architecture and hence such analysis of different architectures help better understand the advantages of one model over the other.

T

1.1 Dataset

The CIFAR-100 dataset contains 50,000 training and 10,000 test images of 20 object classes, along with 100 object subclasses containing 600 images each. There are 500 training images and 100 testing images per class. Each image comes with a "fine" label (the subclass to which it belongs) and a "coarse" label (the class to which it belongs). Each image is an RGB image of size 32x32.

2 Preprocessing

The dataset consists of 32x32 size image and 3 channels and hence the first step was to normalize the training and testing data by dividing it by 255 across all channels to get it in the range of [0.0,1.0] resulting in faster convergence of the model. Since the output was a class vector it was converted to a binary class matrix using 'tensorflow.keras.utils.to_categorical' function. This is necessary because all the models were trained using 'categorical_crossentropy' as the loss function.

2.1 Mean Normalization

For further normalizing the pixel values 'Mean normalization' was used across all the channels so that the pixel values get centered around the mean pixel value of the image. Both the training and test set are normalized using the mean calculated from the training data. It helps during training the model since the number of computations are large, but the magnitude is closer to a particular value (mean).

3 Architecture

3.1 VGGNet 16

VGGNet architecture is named after the Visual Geometry Group at Oxford and is known to be an important breakthrough in the field of Computer Vision using deep learning.It was known for its simplicity as it is just a combination of 3x3 convolution layers stacked on top of each other with max pooling layers in between to reduce the volume size. This architecture can be made deeper to extract more and more intricate features of the dataset. It consists of two fully-connected layers in the end which is then followed by a softmax classifier.

Implementation: A block of 2 or 3 Convolutional layers with 'relu' activation followed by a max-pooling layer was used for the implementation of VGGNet 16 in which '16' stands for the number of weight layers of the model. Deeper models following VGGNet architecture can also be implemented with similar blocks repeated 'n' number of times. The number of filters is doubled after each max-pool layer as the max-pool layer reduces the input volume and it's common to increase the filters as we go deeper in the model.

The drawback of building a deep model using VGGNet architecture is that it becomes slow to train and also the model architecture and its weights take up a lot of disk memory.

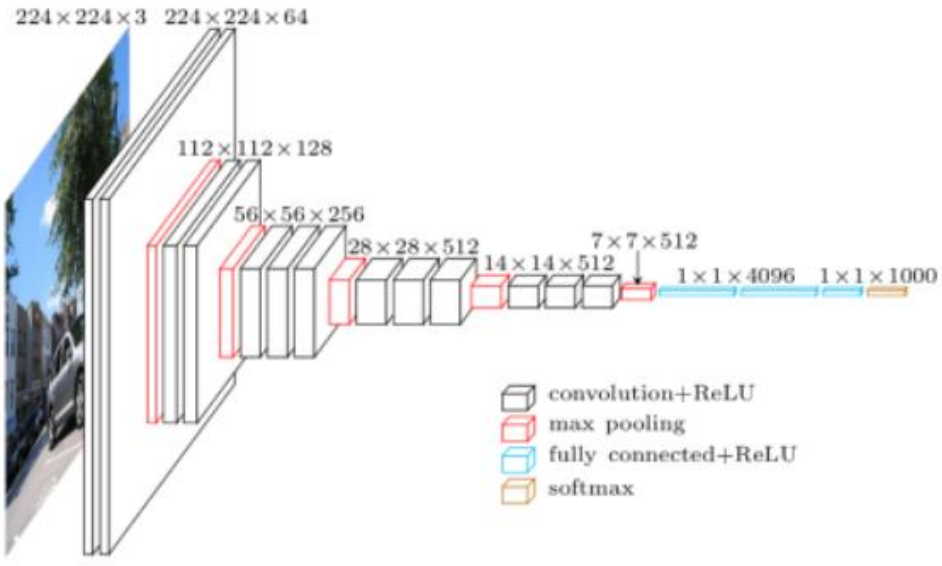


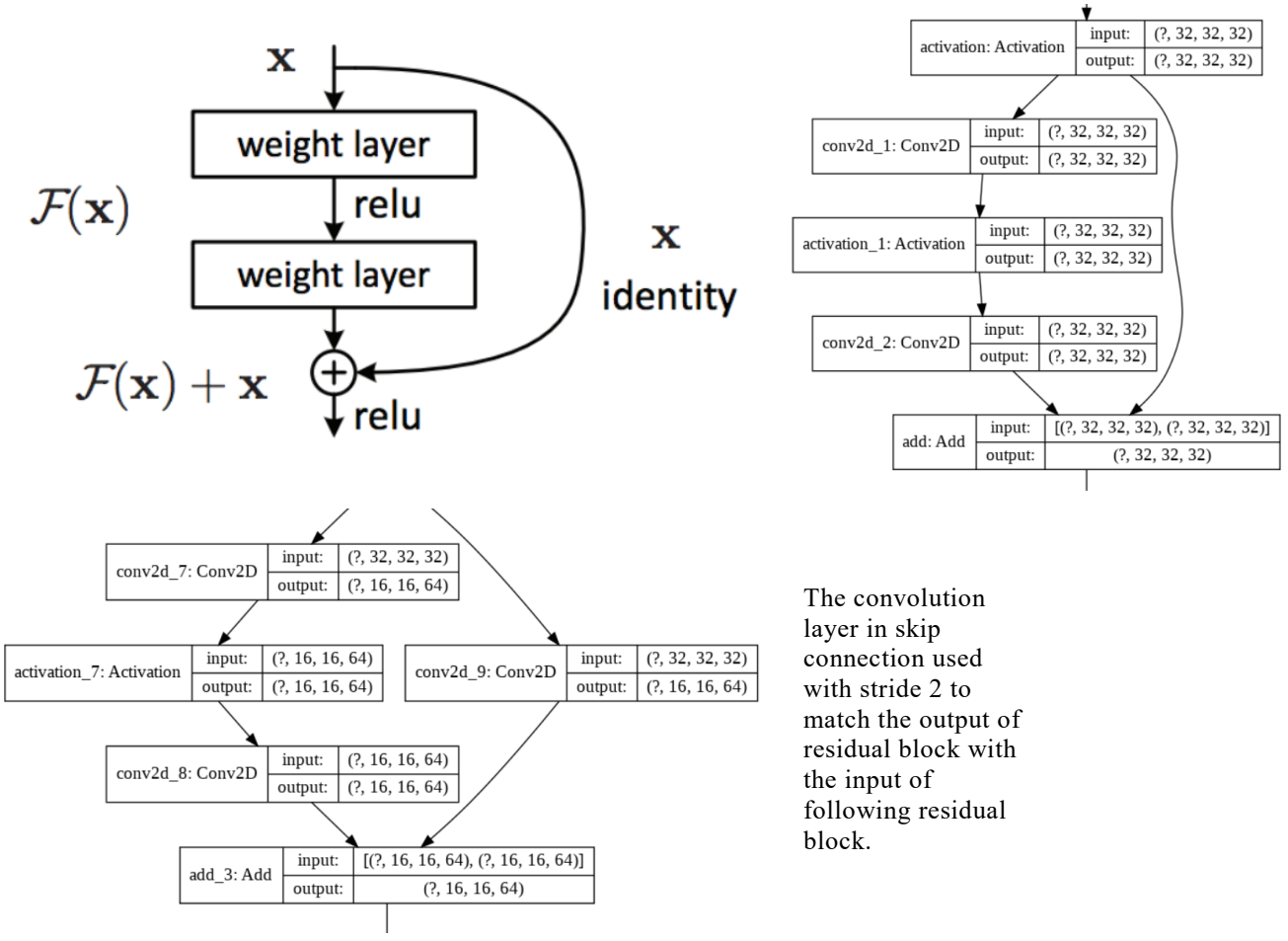
Figure 1: A visualization of the VGG architecture

3.2 ResNet 18

The Residual network or ResNet in short is a stack of residual blocks that allows the model to be substantially deep without much increase in the complexity of the model. The

innovation in ResNet architecture is defined in the residual blocks which consists of skip connections. In tradition deep learning architecture, the deeper layer only learns the input that is provided to it by the layer before it. This makes the model more and more complex with each deeper layer, thereby taking more time to converge. This drawback was addressed by the ResNet architecture in which each residual block consists on 'n' weight layers of same filter size and the output of this block consists of a combination of transformed input (passing through 'n' weight layers) and the identical input to that block skipping all the weight layers inside it. This allows the deeper layer to extract not on the remains from previous layers but also on some part of the actual input of the model. Hence it allows the model to be sufficiently deep with appropriate complexity.

Implementation: Since this architecture is different from the sequential architecture it becomes essential to keep track of the input connection and output connection separately in the residual block. The residual blocks are stacked on top of each other with an identical skip connection that skips all the weight layers of the block and gets added to the output of the weight layers which forms the output of the residual block. The ResNet architecture also uses a convolution layer with stride 2 in the skip connection to reduce the size of the input and output of the residual block in case there is a mismatch in the following blocks. This residual block with a convolution layer of stride 2 in the skip connection is used when the number of filters is needed to be increased. The activation function is always applied to the combined output of the residual block. It also uses Average Pooling at the very end of the network before the linear layers.

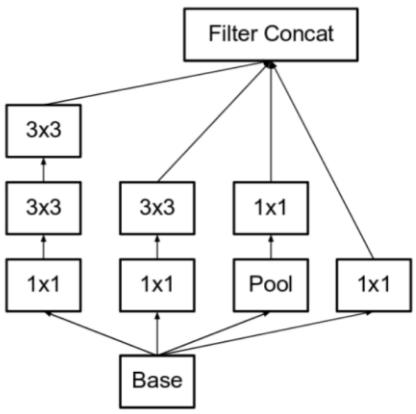


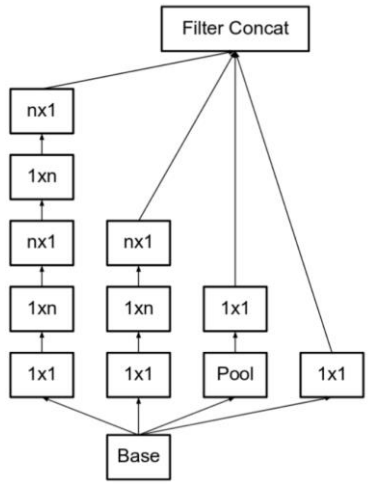
The convolution layer in skip connection used with stride 2 to match the output of residual block with the input of following residual block.

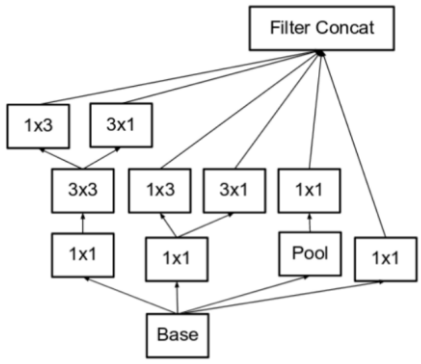
3.3 InceptionV2

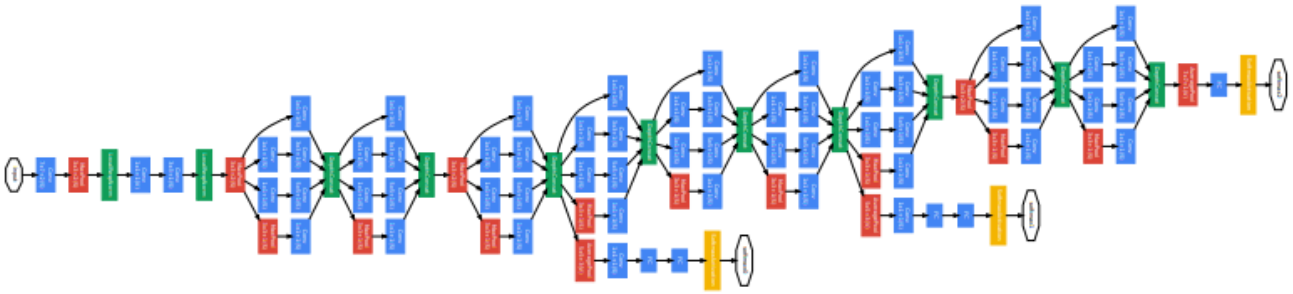
Inception architecture is based on the idea of going wide instead of going deep as it increases the model complexity and is more prone to overfitting. The inception architecture

proposed a sparsely connected networks which replaces the fully connected deep layers. The inception module consists of parallel connections (going wide) with convolution layers of different filter sizes such as 1x1, 3x3, 5x5 and their outputs are then combined in the concatenating layer before giving it as the module output. Inception architecture lifts the burden of selecting a particular filter size by parallelly computing same input with different filters and then the weight of the filters are updated in back-propagation based on the features extracted by that particular layer. These inception modules are then stacked on top of each other with some intermediated auxiliary output coming after some of the modules. The inception network consists of a global average pooling block that takes into account this auxiliary output while prediction. Inception module can also include another parallel max pooling path that allows the model to learn from more variations in the input. Inception version 2 improved on the previous version by reducing the computations and addressing the representational bottleneck. The 2nd version implemented the idea that factorizing may improve computation speed and hence the 5x5 convolution was replaced by two 3x3 convolution layers. Further factorization was implemented in a way that nxn convolution was replaced by a combination of 1xn and nx1. To avoid excessive reduction of dimensions (representational bottleneck) the inception module was made wider by concatenating the output directly from the factorized convolution layers.
Implementation: Three variations of inception module was implemented in which the first variation consists of two 3x3 convolution instead of one 5x5. Second variation consist of 1xn and nx1 factorization of nxn convolution i.e 3x3 was further replaced by 1x3 and 3x1 before concatenating the result from all convolutions. The third variation that made the module even wider was implemented such that the output from 1xn and nx1 module was concatenated as a parallel output instead of sequential outputs in the factorized layers.







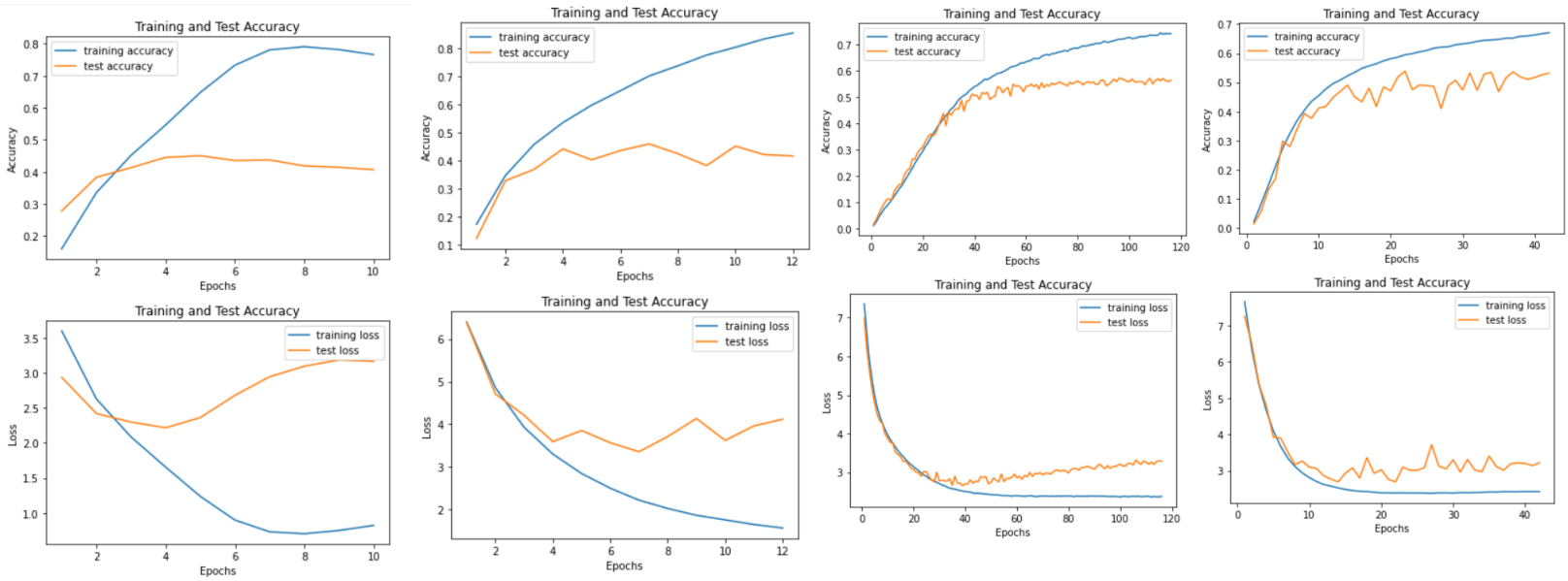


4 Result

Arch		VGG 16			ResNet 18			Inception v2		
Optimizer	**Score**	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**	**Accuracy**
	Setting	-	-	-	-	-	-	-	-	-
SGD	BatchNorm and Dropout	0.58	0.54	0.54	0.62	0.60	0.60	0.54	0.52	0.52
	With BatchNorm	0.57	0.46	0.46	0.60	0.46	0.46	0.52	0.49	0.49
	With Dropout	0.58	0.57	0.57	0.61	0.60	0.60	0.52	0.52	0.52
	No Regularization	0.48	0.45	0.45	0.51	0.47	0.47	0.47	0.46	0.46
ADAM	BatchNorm and Dropout	0.66	0.65	0.65	0.59	0.59	0.59	0.51	0.49	0.49
	With BatchNorm	0.59	0.54	0.54	0.52	0.49	0.49	0.50	0.49	0.49
	With Dropout	0.55	0.54	0.54	0.51	0.51	0.51	0.40	0.40	0.40
	No Regularization	0.51	0.49	0.49	0.45	0.45	0.45	0.40	0.40	0.40

4.1 VGGNet 16

4.1.1 SGD



The figures above show the Training and Test Accuracy/Loss for the training of VGGNet 16 with SGD optimizer. The four variations in the model are No Regularization, with Batch Normalization, with Dropout and with Batch Normalization-Dropout together in the respective order.

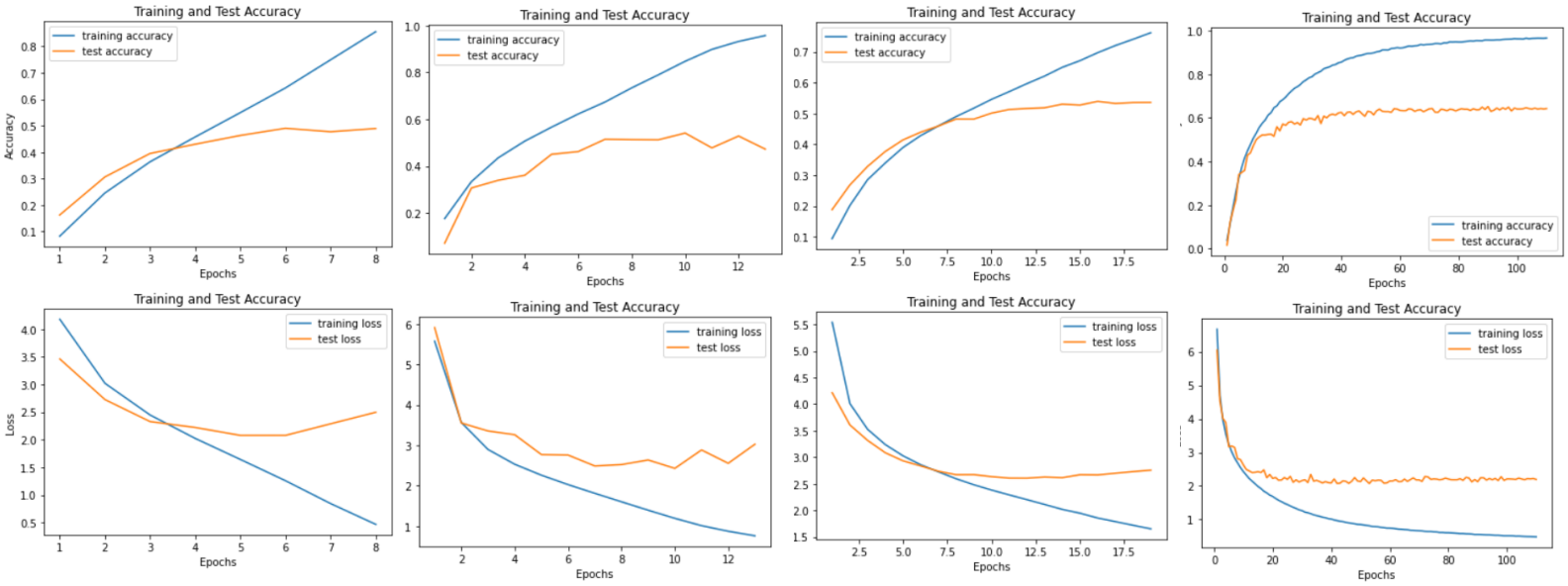
Observations-

1. While training VGGNet without regularization, using 'elu' as the activation function gave best result (acc 45%) as compared to relu (34%) and Leaky relu (acc 40%).
2. Learning rate of 0.035 with decay of 1e-5 and momentum 0.8 gave the best results. Accuracy dropped when Learning rate or decay was changed.
3. Instead of 3 fully connected layers, dropping one allowed the model to generalize

well.

4. Batch normalization layer was added after each convolutional layer and before the activation. Dropouts were added not on all convolution layers but on about half of them.
5. VGGNet model converged quickly and with better generalization but it utilized more space as compared to InceptionNet, because of its simplicity.
6. The model with dropout regularizer gave best results.

4.1.2 ADAM



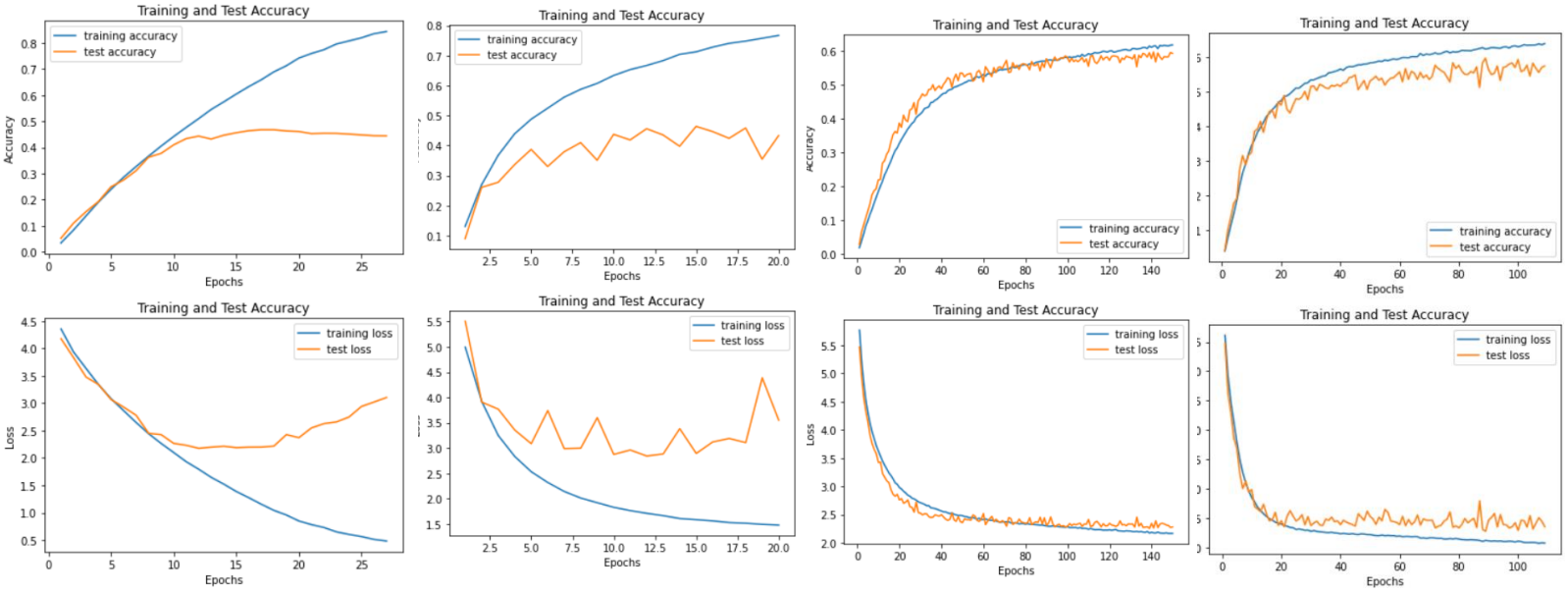
The figures above show the Training and Test Accuracy/Loss for the training of VGGNet 16 with ADAM optimizer. The four variations in the model are No Regularization, with Batch Normalization, with Dropout and with Batch Normalization-Dropout together in the respective order.

Observations-

1. While training VGGNet without regularization, using 'elu' as the activation function gave best result (acc 49%) as compared to relu and Leaky relu.
2. Learning rate of 0.001 with decay of 1e-3 gave the best results. Accuracy dropped when Learning rate or decay was changed.
3. Instead of 3 fully connected layers, dropping one allowed the model to generalize well.
4. Batch size of 128 during training gave the best results and the rate at which model converged dropped if batch size was changed.
5. Batch normalization layer was added after each convolutional layer and before the activation. Dropouts were added not on all convolution layers but on about half of them.
6. VGGNet model converged quickly and with better generalization but it utilized more space as compared to InceptionNet, because of its simplicity.
7. The model with Dropouts and batch normalization applied together gave best results.

4.2 ResNet 18

4.2.1 SGD

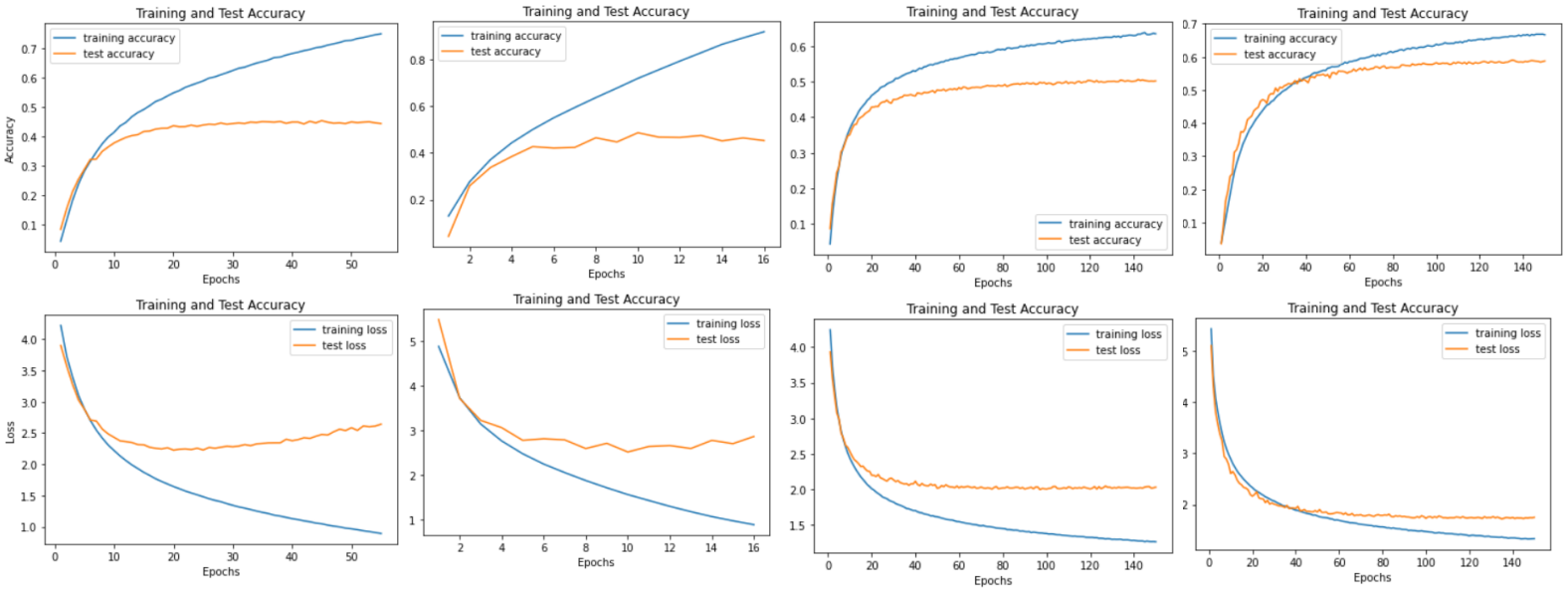


The figures above show the Training and Test Accuracy/Loss for the training of ResNet 18 with SGD optimizer. The four variations in the model are No Regularization, with Batch Normalization, with Dropout and with Batch Normalization-Dropout together in the respective order.

Observations-

1. Learning rate of 0.035 with decay of 1e-5 and momentum of 0.8 gave the best results. Accuracy dropped when Learning rate or decay was changed.
2. Instead of using max pool to reduce the input size, a convolution layer of 1x1 and stride 2 was used in the skip connection layers. It is necessary so that the input to the next residual block matches output of previous block. Identical input was also used as skip connection in residual blocks of same dimensions.
3. The size of the ResNet 18 model had to be reduced because it was overfitting the data and was not generalizing well to test set. The number of filters in this implementation of ResNet was in the range of 32 to 128 instead of the range 64 to 512 in the original architecture. Also this implementation doesn't use the 7x7 convolution in the beginning instead it used 3x3 convolution to balance the reduction in the total number of filters.
4. Batch size of 128 during training gave the best results and the rate at which model converged dropped if batch size was changed.
5. Batch normalization layer was added after each convolutional layer and before the activation. Dropouts were added not on all convolution layers but on alternate layers and dropout percentage was increased in the deeper layers to avoid overfitting.
6. ResNet model converged quickly and with better generalization as compared to both VGGNet and InceptionNet.
7. The model with Dropouts and batch normalization applied together gave best results.

4.2.2 ADAM



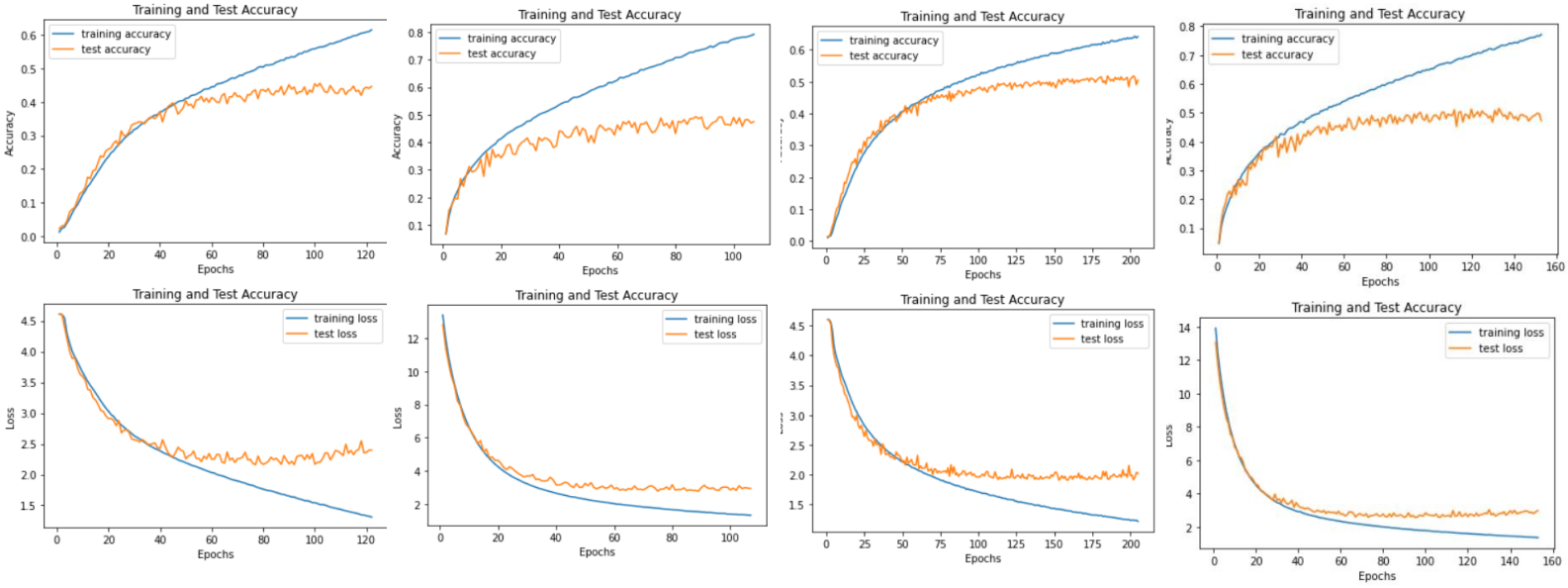
The figures above show the Training and Test Accuracy/Loss for the training of ResNet 18 with ADAM optimizer. The four variations in the model are No Regularization, with Batch Normalization, with Dropout and with Batch Normalization-Dropout together in the respective order.

Observations-

1. Learning rate of 0.0006 with decay of 1e-3 and clipvalue of 0.5 gave the best results. Accuracy dropped when Learning rate or decay was changed. Without using clipvalue the model was underfitting during training. Also it was observed that decay used in ADAM optimizer was greater (in addition to the smaller learning rate) as compared to the decay in SGD.
2. Instead of using max pool to reduce the input size, a convolution layer of 1x1 and stride 2 was used in the skip connection layers. It is necessary so that the input to the next residual block matches output of previous block. Identical input was also used as skip connection in residual blocks of same dimensions.
3. The size of the ResNet 18 model had to be reduced because it was overfitting the data and was not generalizing well to test set. The number of filters in this implementation of ResNet was in the range of 32 to 128 instead of the range 64 to 512 in the original architecture. Also this implementation doesn't use the 7x7 convolution in the beginning instead it used 3x3 convolution to balance the reduction in the total number of filters.
4. Batch size of 128 during training gave the best results and the rate at which model converged dropped if batch size was changed.
5. Batch normalization layer was added after each convolutional layer and before the activation. Dropouts were added not on all convolution layers but on alternate layers and dropout percentage was increased in the deeper layers to avoid overfitting.
6. ResNet model converged quickly and with better generalization as compared to both VGGNet and InceptionNet.
7. The model with Dropouts and batch normalization applied together gave best results.

4.3 Inception v2

4.3.1 SGD

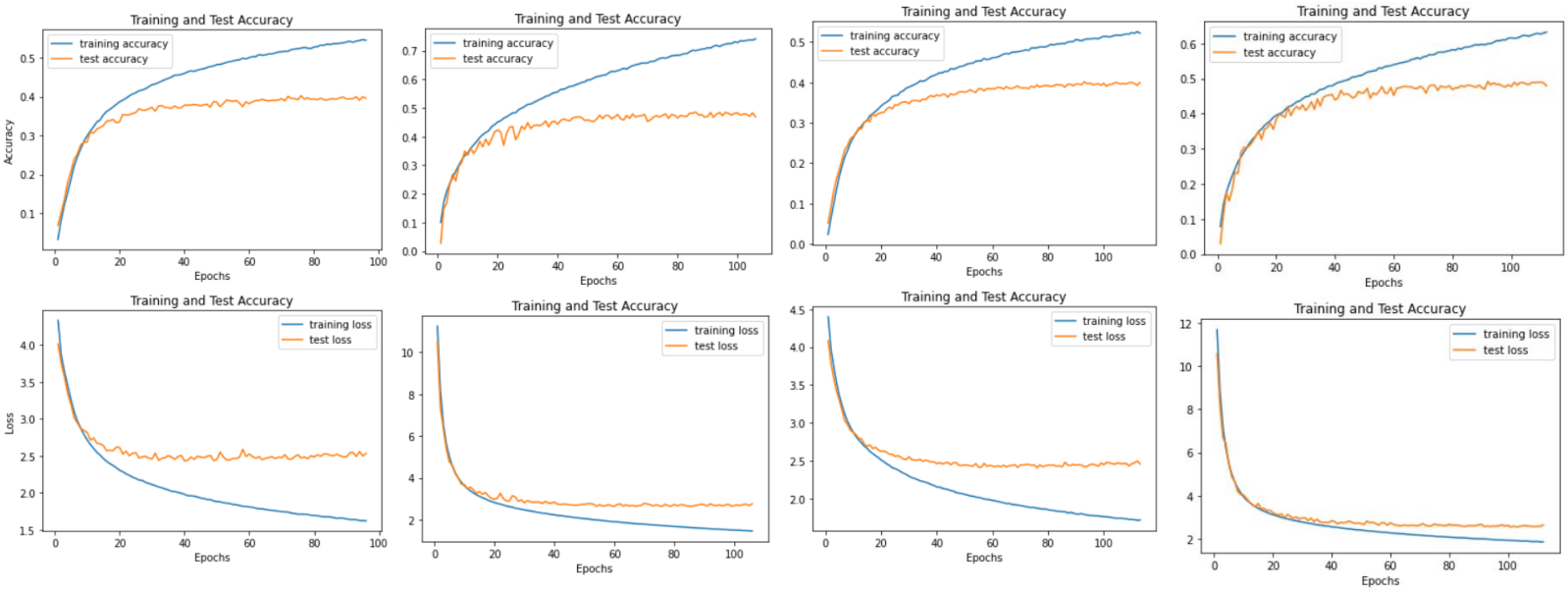


The figures above show the Training and Test Accuracy/Loss for the training of InceptionNet v2 with SGD optimizer. The four variations in the model are No Regularization, with Batch Normalization, with Dropout and with Batch Normalization-Dropout together in the respective order.

Observations-

1. Learning rate of 0.015 with decay of 1e-4 and momentum of 0.8 gave the best results. Accuracy dropped when Learning rate or decay was changed. Larger learning rate was also tried but the model couldn't learn new features after a certain threshold leading to divergent behavior.
2. When the exact architecture of InceptionNet was used to train, the model was overfitting quickly i.e. it was learning too much from the training set and hence the 5-inception block of type B (1xn/nx1 factorized deep module) was replaced with 3 inception blocks of same type.
3. Since the number of training data per class is not large, "ImageDataGenerator" was used to train the model with random variations so that the model generalizes well without overfitting.
4. Original inception block consists of max pool layer, replacing it with average pool layer improved the accuracy of the model when trained on CIFAR-100.
5. Removing the auxiliary outputs and Global Average pool layer and replacing it with one average pool layer at the end helped in reducing the complexity of the model thereby letting the model generalize well on the test set.
6. Batch size of 128 during training gave the best results and the rate at which model converged dropped if batch size was changed.
7. Batch normalization layer was added after most convolutional layers in the inception block (including factorized layers) and before the activation. Dropout was added only before the last linear layer of the model as described in the InceptionNet paper.
8. InceptionNet model took a longer time to train as compared to VGGNet and ResNet but it utilized less memory for weights.
9. The model with Dropouts and batch normalization applied together gave best results.

4.3.2 ADAM



The figures above show the Training and Test Accuracy/Loss for the training of InceptionNet v2 with ADAM optimizer. The four variations in the model are No Regularization, with Batch Normalization, with Dropout and with Batch Normalization-Dropout together in the respective order.

Observations-

1. Learning rate of 0.0006 with decay of 1e-3 and clipvalue of 0.5 (in models with no regularization and with only dropouts the clipvalue of 1 gave better result) gave the best results. Accuracy dropped when Learning rate or decay was changed. Larger learning rate was also tried but the model couldn't learn new features after a certain threshold leading to divergent behavior. Without using clipvalue the model was underfitting during training.
2. When the exact architecture of InceptionNet was used to train, the model was overfitting quickly i.e. it was learning too much from the training set and hence the 5-inception block of type B (1xn/nx1 factorized deep module) was replaced with 3 inception blocks of same type.
3. Since the number of training data per class is not large, "ImageDataGenerator" was used to train the model with random variations so that the model generalizes well without overfitting. It was found that smaller variations such as (rotation_range=5, zoom_range=0.1) improved the learning but further increase in the variations resulted in model underfitting.
4. Original inception block consists of max pool layer, replacing it with average pool layer improved the accuracy of the model when trained on CIFAR-100.
5. Removing the auxiliary outputs and Global Average pool layer and replacing it with one average pool layer at the end helped in reducing the complexity of the model thereby letting the model generalize well on the test set.
6. Batch size of 128 during training gave the best results and the rate at which model converged dropped if batch size was changed.
7. Batch normalization layer was added after most convolutional layers in the inception block (including factorized layers) and before the activation. Dropout was added only before the last linear layer of the model as described in the InceptionNet paper.
8. In ADAM setting introducing batch normalization to the model creates a significant improvement in the performance which is evident through the performance table.
9. InceptionNet model took a longer time to train as compared to VGGNet and ResNet but it utilized less memory for weights.

10. The model with Dropouts and batch normalization applied together gave best results.

5 Conclusion

VGGNet architecture was the simplest architecture to build as it only consists of convolution layers and max-pooling layers stacked on top of each other. When VGGNet architecture is used to build deeper models, it results in increasing the complexity of the overall model thereby leading to long time to train and utilizes larger memory for storing weights. This drawback of VGGNet was addressed by ResNet architecture which allowed building deeper models still maintaining the integrity of the input image to the very first layers. ResNet made use of skip connections in which the input had a shorter path as well as the traditional path consisting layers of convolution. This allowed ResNet based models to work on and extract features not only from the operated output of several convolutions but also from the identical input and hence it maintained the model's complexity. Using ResNet architecture very deep models were able to be built but the Inception architecture argues of building wider networks with use of parallel connecting layers. This architecture allowed the models to weigh the filters that works best for the given input by forming a parallel convolution layers of different filter size. It elevated the need of determining the number of filters as the model itself updates the weights of important filters while training. It also involved a parallel max pooling layer in the inception block to extract features from varying input size. When the model becomes very complex it tries to learn the training set completely and hence reducing the complexity either by removing deeper layers or by introducing dropout allows the model to generalize better.

Using regularizations such as dropouts, kernel regularizers and batch normalization greatly helps the model to converge in a much faster rate with better generalizability. It can be observed in all three architectures, using Batch normalization and dropout together helped in improving the model's accuracy as compare to the model with no regularization. It can also be observed that ADAM optimizer learns better with smaller learning rate and larger decay as compared to SGD optimizer. Momentum in SGD optimizer also helps the model to converge faster and similarly clipping the gradient in ADAM setting improves the model's performance. If the training data is not vast and number of samples per class in less then ImageDataGenerator can be helpful as it randomly does some variations in the input so that the model doesn't try to learn the training dataset exactly.

References

[1] https://arxiv.org/pdf/1409.1556.pdf
[2] https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202
[3] https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/
[4] https://machinelearningmastery.com/how-to-implement-major-architecture-innovations-for-convolutional-neural-networks/
[5] https://arxiv.org/pdf/1512.03385.pdf
[6] https://arxiv.org/pdf/1512.00567v3.pdf
[7] https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035
[8] https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
[9] https://www.cs.toronto.edu/~kriz/cifar.html