Computer Vision Project Report
SSNet

Saksham Sinha

May 22, 2018

Contents

Abstract

This project report is submitted as a part of coursework requirement for Computer Vision (CS7GV1) course. In this report, I have studied state of the art models like AlexNet, ResNet, VGG and GoogleNet, experimented with various convolution strategies like data augmentation, activation functions, weights initalizations, etc, created my own model and compared its performance with state of the art pretrained models and models trained from scratch. My model outperforms all state of the art (trained from scratch) compared in this paper on Tiny ImageNet dataset without overfitting by reaching the top-1 validation accuracy of 45.7% and top-5 validation accuracy of 72%. As in Deep Learning community we have LeNet, AlexNet, and GoogleNet I have named my model as SSNet based on my name initials.

1 Introduction

The study of Computer Vision (CV) seeks to automate the tasks of human vision system. CV is concerned with the theory behind artificial systems that extract information from images and videos. The CV data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems. Various tasks includes object recognition, image classification, image restoration, 3D Pose estimation, event detection etc. Convolutional networks have found great success in almost all the tasks, especially with image classification task, after the availability of very large image dataset like ImageNet and good computability power from high end GPUs.

2 Related Background

Recently, for image classification, Convolutional Neural Networks (CNN) have been very successful for solving such problems. In 2012, Alex Krizhevsky et.al [1] came up with the first successful CNN for solving image classification task. It became the first CNN to win the The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 using the ImageNet dataset[2]. Since, then there has been many CNNs to win the ILSVRC for example, ResNet[3], VGG16[4] and GoogLeNet/Inception[5]. Each of these models are discussed in detail in below subsections-

2.1 AlexNet

AlexNet[1] is the first work that popularized ConvNets in computer vision. It started the trend of deep learning and made Convolutional Networks very popular which is evidenced by its current 20,000 citations. AlexNet consists of a total of 8 layers, which are 5 convolutional layers and 3 fully-connected(FC) layers. Batch-normalization is applied after the first two convolutional layers. Dropout is applied after each FC layers. They were able to achieve winning top-5 test error rate of 15.3% compared to 26.2% achieved by the second-best entry.

2.2 VGG16

Authors of VGG16[4] contributed in thorough evaluation of networks of increasing depth using an architecture with very small (3x3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. According to author the combination of two 3x3 convolutional layers has an effective receptive field of 5x5. This in turn simulates a larger filter while keeping the benefits of smaller filter sizes. VGG secured the first and the second places in the localisation and classification tracks respectively in ImageNet Challenge.

2.3 ResNet

Authors of ResNet[3] presents a residual learning framework to ease the training of networks that are substantially deeper than those used previously. They explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. Authors showed that their extremely deep residual nets are easy to optimize and that deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results

substantially better than previous networks. ResNet is a 152 layer network architecture and won ILSVRC 2015 with an incredible error rate of 3.6%.

2.4 GoogLeNet/Inception

GoogLeNet[5] is another ImageNet winner that improved on AlexNet and VGGNet by using so called inception modules. Inception modules use multiple filter sizes at each layer and concatenate the results together. This allows the network to learn features of different sizes at each layer in the network. It used 12 times less parameters than that of AlexNet with having over 100 layers in its architecture. GoogLeNet was one of the first models that introduced the idea that CNN layers didn't always have to be stacked up sequentially. Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and computational efficiency.

2.5 Batch Normalization

The authors of paper[6] discussed the advantages of batch normalization in training. Training deep neural network is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. Authors call this as internal covariate shift. Authors method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows to use much higher learning rates and be less careful about initialization and help achieving the same accuracy with 14 times fewer training steps.

2.6 Normalized Initialization (Xavier Initialization)

: The author of paper [7] suggests that the normalization factor may be important when initializing deep networks because of the multiplicative effect through layers. The suggestion then is of an initialization procedure that maintains stable variances of activation and back-propagated gradients as one moves up or down the network. They called it as the normalized initialization.

3 Implementations

3.1 Dataset Used

I have used the standard Tiny ImageNet dataset. The Tiny ImageNet dataset consists of the same data as of the ImageNet Dataset however the images are centered cropped into size of 64x64 from 224x224. Instead of 1,000 classes of ImageNet challenge, this dataset has only 200 classes. In training data, it has 500 images in each class making the total of 100,000 training data. For validation and test data, dataset has 50 images in each class giving the total of 10,000 validation images and 10,000 test images. It uses one-hot encoding for making labels. The cropping and down-sampling of images caused degradation in quality of the images causing it to pixelate. The effect of this down-sampling includes loss of details and thus creating difficulty of locating small objects. Few examples of images are shown below-

(a) (b) (c) (d) (e) (f)

Figure 1: Dataset images of size 64x64.

3.2 Implementation of state of the art models

To implement AlexNet, ResNet18 and VGG16, I have used torchvision.model library to use inbuilt models. By setting the pretrained flag as true or false will make use of pretrained weights or train from scratch respectively. However, these models classify the images to 1000 classes. Hence these models are required to be fine tuned to classify the images to 200 classes which is discussed in the section below. Also for using the pretrained weights, all layers but the last needed to be freezed. Its implementation is also discussed in the section below.

3.3 Fine Tuning of state of the art

Since the models were designed to classify the images to 1000 classes, these models were required to be fine tuned so that they can classify to 200 classes. Each state of the art model had to be fine tuned in a different way because of how they are implemented in the torchvision.models library. Hence for each model fine tuning is explained below-

1. **ResNet18:** Fine tuning ResNet was compartively easy. Since ResNet18 implementation did not used any sequential block of pytorch model. Hence it exposed its last layer to be modified easily. Hence I took all the layers of ResNet but the last layer. I created and added the new last layer that would classify the images to 200 classes.

2. **AlexNet:** Fine tuning Alexnet was a bit complicated. This was due to the fact that AlexNet made use of sequential block in their implementation. Hence, the whole last sequential block needed to be changed just to change the last fully connected layer. This sequential block contained all three fc layers of AlexNet with dropout. I took all the sequential blocks but the last and as implemented originally in the model, I made a new last sequential block with exact implementation and just changed the last fc layer to classify to 200 classes.

3. **VGG16:** Fine tuning VGG16 was done similar to AlexNet due to the similarity in the implementation of fully connected layers. This model was also originally implemented using the sequential blocks. Hence, I had to re-write the last sequential block completely exactly as orignially implemented and changed the last layer in the sequential block to classify to 200 classes

4. **Weights freezing for the pretrained models:** While training the pretrained models, I found a bug that even though we are downloading the weights, the model was still trained completely which removes the whole purpose of using pretrained weights. This was solved by freezing weights of all layers but the last, as we will have to retrain the last layer/block to classify to 200 classes. Hence by setting the required_gradient attribute to false of all the parameters of the model freezes the weights. This, also had to be reflected in the optimizer implementation where I have used stochastic gradient descent or else it will give error as the model layers don't require gradient. Hence I passed only the parameters that required gradient to SGD instead of passing the whole model parameters.

3.4 Inspiration for skeleton of my model SSNet

Since the PCs in the labs are restricted with the limited memory of 8GB on NVIDIA GTX 1080 graphic card, and the number of times my workstation was restarted in the middle of training of the state of the art models, I therefore decided to make my model that will train fast and still performs comparatively better. So I looked into a state of the art model that specifically was developed to reduce the number of parameters which in turn reduces the training time. The answer for this requirement was inception model by Google. GoogleNet or Inception uses network in a network model. Hence, it is able to extract information about the very fine grain details in the volume, while the 5x5 filter is able to cover a large receptive field of the input, and thus able to extract its information as well. Even with more than 100 layers deep, it had 12 times less parameters than AlexNet and hence saved huge amount of training time. It does so by removing the use of fully connected layers. Therefore, I too used the network in a network architecture as detailed below in the figure 2. The reason that Google's Inception model had so less parameters were due to the fact that they didn't had fully connected layers. Hence I decided to remove the fully connected layers as well which in turn removed the requirement of having dropout layer too.

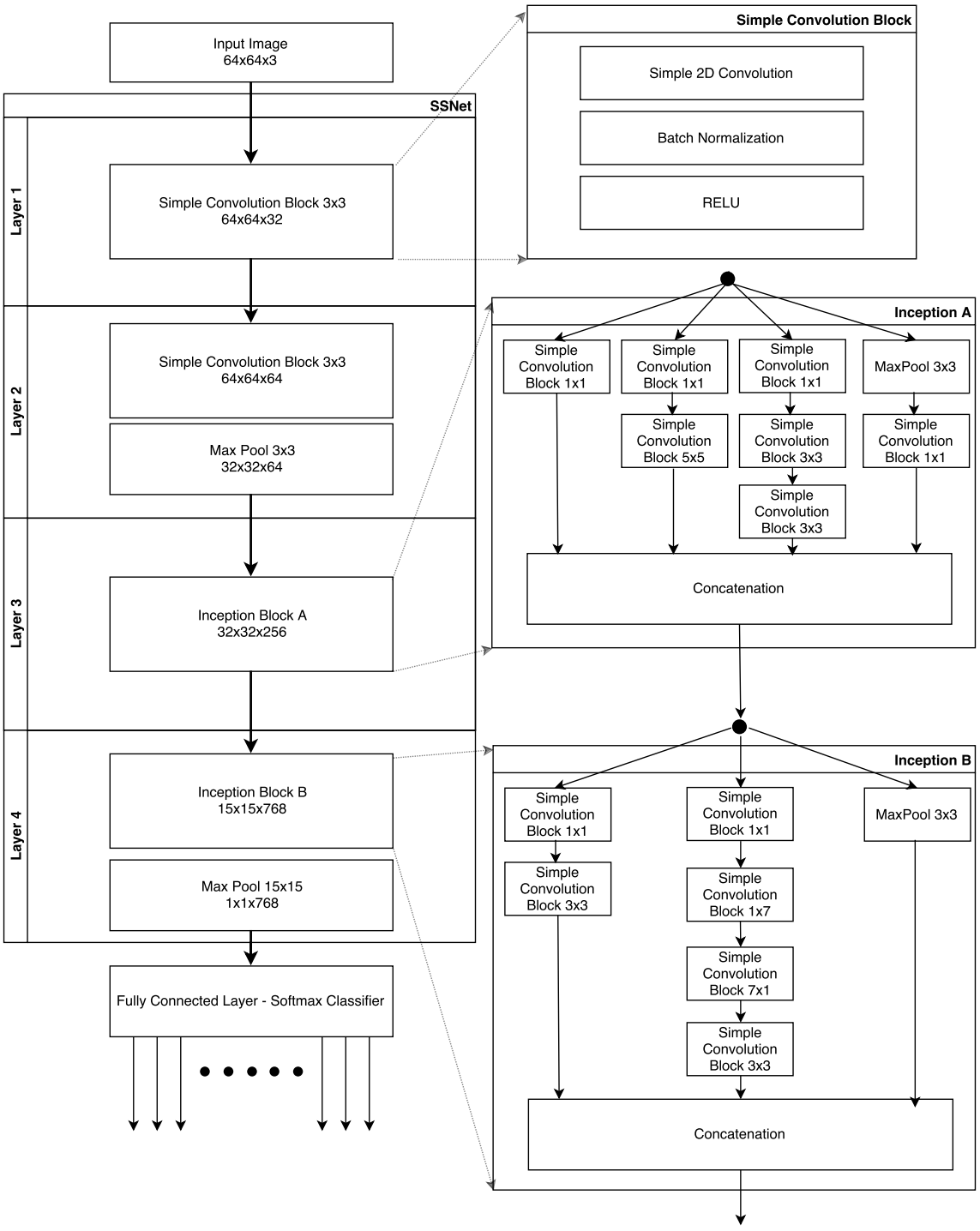

Figure 2: SSNet Model

3.5 Architecture and Implementation of SSNet

My model, as a whole has 4 major layers of which first two are Simple convolutional blocks and the other two are the Inception blocks. But if we look at the finer details, we will see that my model has actually 30 layers. Each simple convolutional block are made up of 3 layers- 2D Convolution layer, a batch normalization layer and the Relu layer. Also after two simple convolutional blocks, there is a max pooling layer. Inception A has 3 layers with 4 convolutions happening in parallel. 1st convolution process is just simple convolution block with 1x1 filter and its output is saved for concatenation with other parallel convolutions. In 2nd parallel convolutions, there are two simple

convolution blocks. First convolution is with 1x1 filter and then 2nd with 5x5 filter. The third parallel convolutions has three simple convolution blocks with first convolution with 1x1 filter and rest two with with 3x3 filters. The last parallel convolution is the max pool layer and a simple convolution block of 1x1. Each parallel convolution's output is then concatenated and given as an input to next layer which is Inception B. The Inception B has similar architecture but with three parallel convolutions. The first parallel convolution has two simple convolution block with first block of 1x1 filter and second with 3x3 filter. The second parallel convolution has 4 simple convolution blocks with 1x1, 1x7, 7x1 and 3x3 filters. The third parallel convolution has just max pool layer. The results are concatenated and then passed to max pool layer of size 15x15 (to remove the need of fully connected layer as done in inception[5]) and then finally fully connected layer to classify in 200 classes. The weights were initialized with normalized initialization[7] and model was trained using stochastic gradient descent with weight decay (L2 penalty or L2 regularization). I also later visualized the predictions and to get deeper understanding and plotted the confusion matrix.

4 Experimentation and Results

4.1 Experimentation

To create my model I had done various experimentation which I have described below-

1. **Initial Exploration with Demo model:** I first studied the demo model, and understood the implementation details of it. I trained it to understand how the accuracy and loss is calculated. Also I understood how the gradient is back propagated and different ways to reduce the learning rate using the optimizer. I learned how dataset is being loaded and augmented before its fed to the network. Inorder to create some graphs, I first used matplotlib library to generate the graph, but after some exploration on plotting, I came to know about TensorboardX[8] which is a tensorboard library for pytorch. After this it was easy to plot the graphs by adding data to writer.scalar for tensorboard. I plotted the graph for demo model, and found out that it severely overfits. The training accuracy was more than 98% while validation accuracy was less than 25%. This gave the good start in understanding of how the neural networks work.

2. **Data Augmentation:** I used various data augmentation strategies to figure out which works the best for my model. With the given Horizontal flip and Scale Horizontal flip, I used vertical flip, random crop, random rotation, Ten crop and Color Jitter to test my model.

3. **Activation function:** With Rectified Linear Unit (Relu) which is a very standard activation function, I also tested with Leaky Relu and sigmoid as well to compare the performance of my model and to see with which activation function, my model works the best.

4. **Weight initialization and regularization:** I tested my model first with random weight initialization and no regularization. Later I added normalized initialization of weights and L2 regularization to my model.

5. **Evaluation metrics:** Top-1 and top-5 validation and testing accuracies were used to evaluate SSNet and state of the art models. Top-1 accuracy refers to the accuracy of a model to correctly classify in 1st guess. Top-5 accuracy refers to the accuracy of a model to correctly classify in top 5 guesses. If any of these five guesses are correct, then that image is considered correctly classified for top-5 accuracy.

6. **Comparison with pretrained state of the art models:** I tested my model against the pretrained state of the art models on parameters such as convergence, training and validation loss and finally training accuracy and validation accuracy.

7. **Comparison with state of the art trained from scratch:** I also tested my models against the state of the art models trained from scratch on the same parameters as stated above.

4.2 Results

The results of experimentations and final model comparison with state of the art are shown below-

1. **Weight Initialization and Regularization:** Below figure 3 shows the graph of comparison of SSNet performance with random weight initialization and no regularization with normalized initialization and L2 regularization.

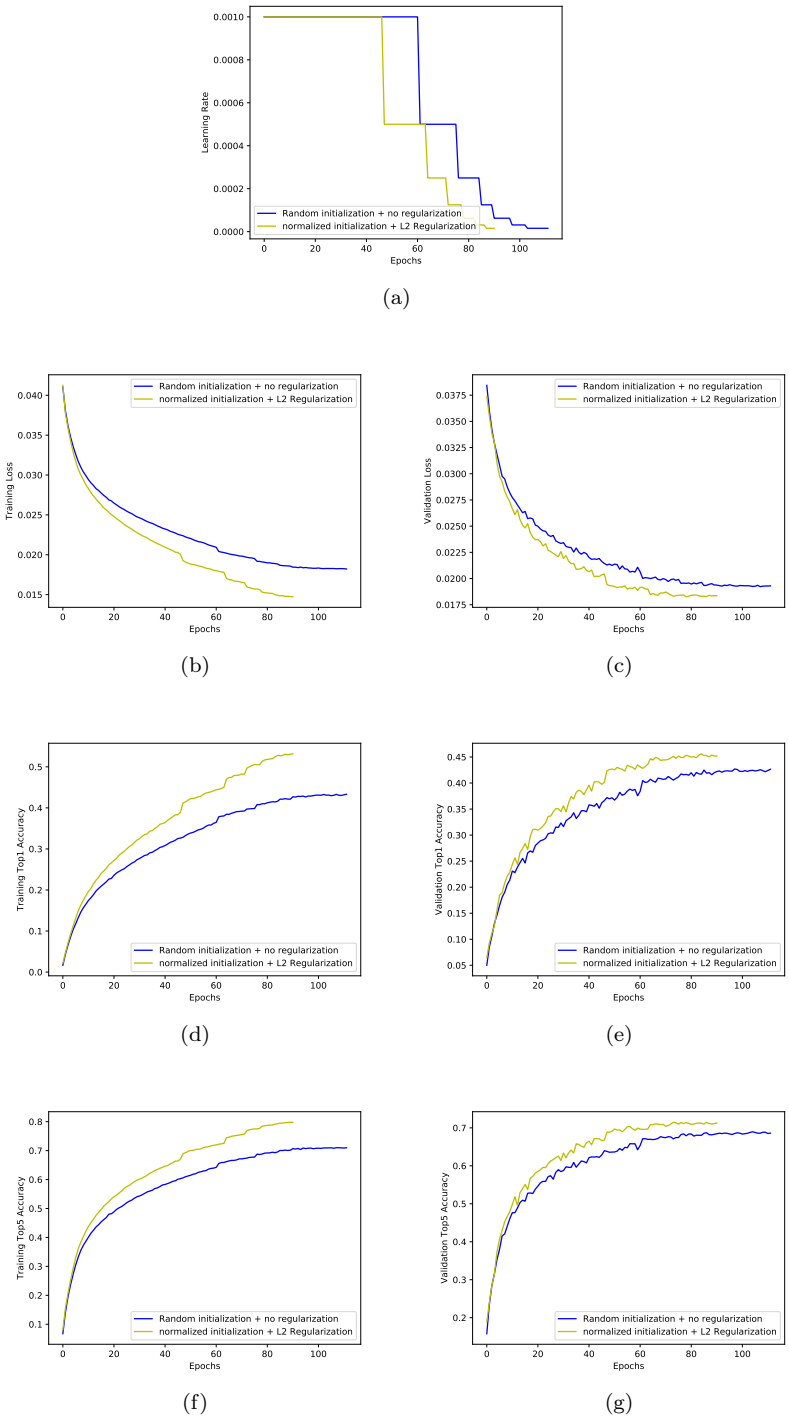

Figure 3: Comparison of SSNet with random weights initialization and no L2 regularization with normalized initialization and L2 regularization: (a) Learning rate comparison (algorithm convergence); (b) Training loss comparison; (c) Validation loss comparison; (d) Training top-1 accuracy comparison; (e) Validation top-1 accuracy comparison; (f) Training top-5 accuracy comparison; and, (g) Validation top-5 accuracy comparison;

2. **Data Augmentation:** Below figure 4 shows the graph of comparison of SSNet performance with different augmentation strategy like Random Horizontal Flip (HFlip), Random Vertical Flip (VFlip), Random Rotation (-20 degrees to +20 degrees), and Color Jitter.

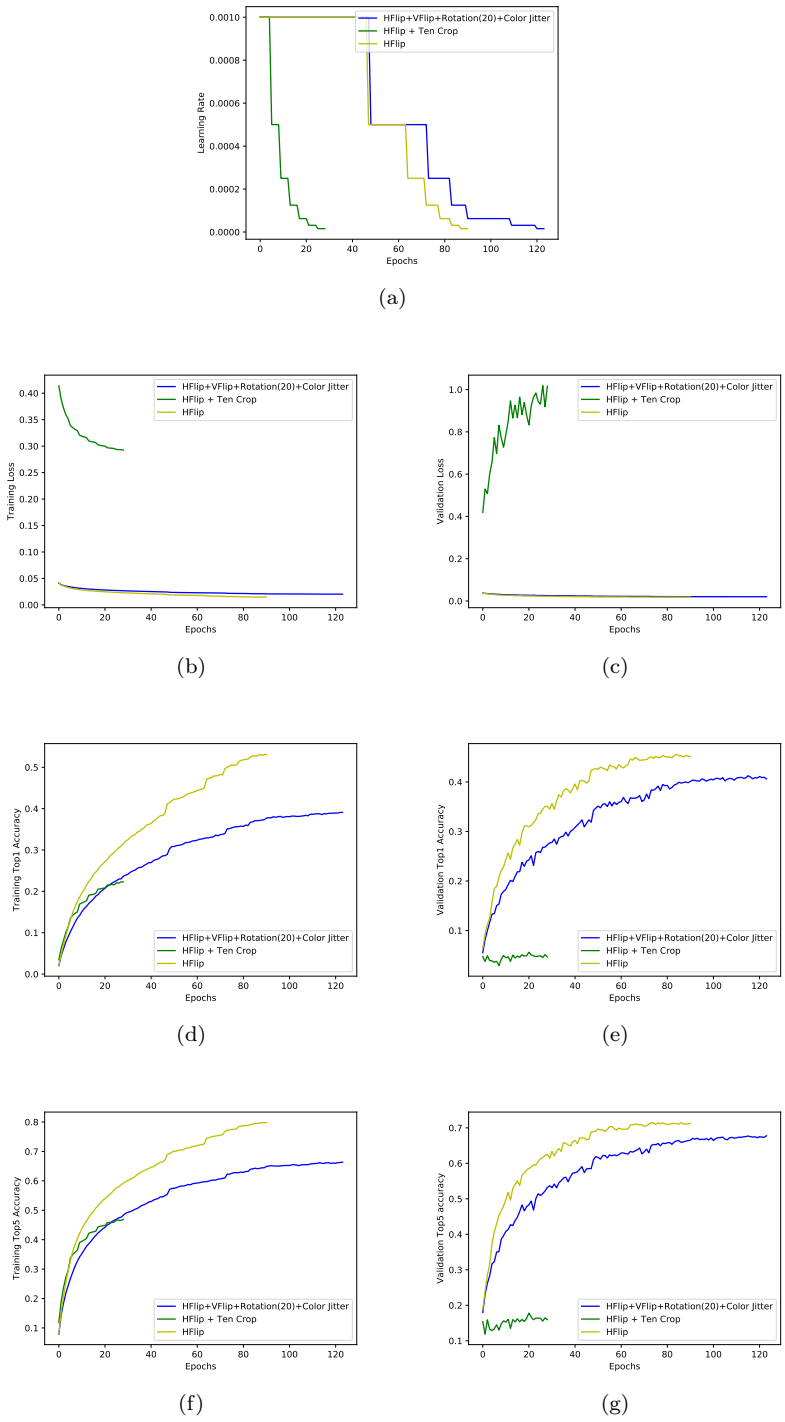

Figure 4: Comparison of SSNet with different data augmentations like HFlip, VFlip, Rotation(20), and Color Jitter: (a) Learning rate comparison (algorithm convergence); (b) Training loss comparison; (c) Validation loss comparison; (d) Training top-1 accuracy comparison; (e) Validation top-1 accuracy comparison; (f) Training top-5 accuracy comparison; and, (g) Validation top-5 accuracy comparison;

3. **Activation Function:** Below figure 5 shows the graph of comparison of SSNet performance with different activation function.

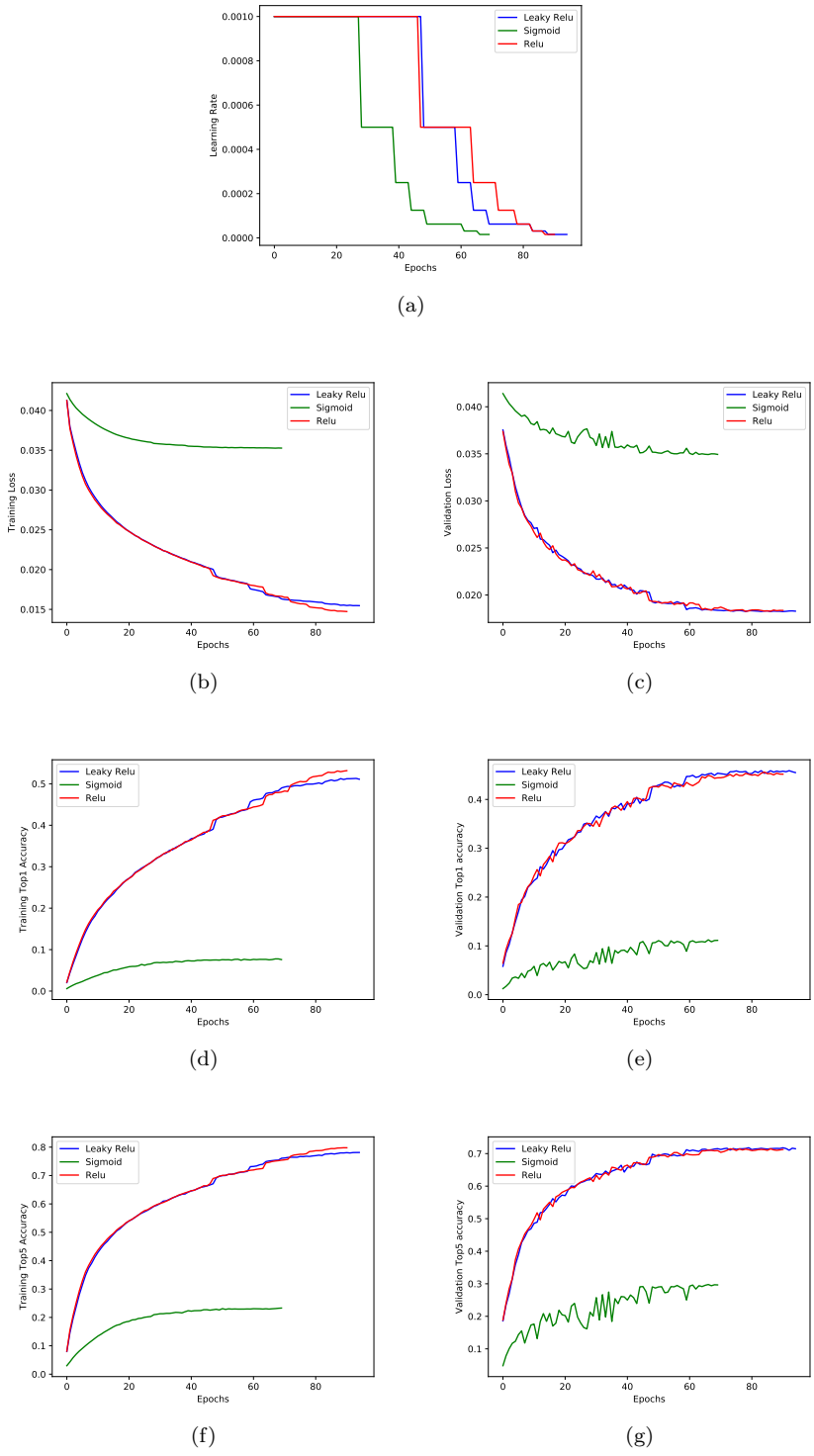

Figure 5: Comparison of SSNet with different activation functions Relu, LeakyRely and Sigmoid: (a) Learning rate comparison (algorithm convergence); (b) Training loss comparison; (c) Validation loss comparison; (d) Training top-1 accuracy comparison; (e) Validation top-1 accuracy comparison; (f) Training top-5 accuracy comparison; and, (g) Validation top-5 accuracy comparison;

4. **Final Model Comparison with fine tuned pretrained state of the art:** Below figure 6 shows the graph of comparison of SSNet with state of the art model with pretrained weights and fine tuning. These results are discussed in detail in the analysis and discussion of results section.

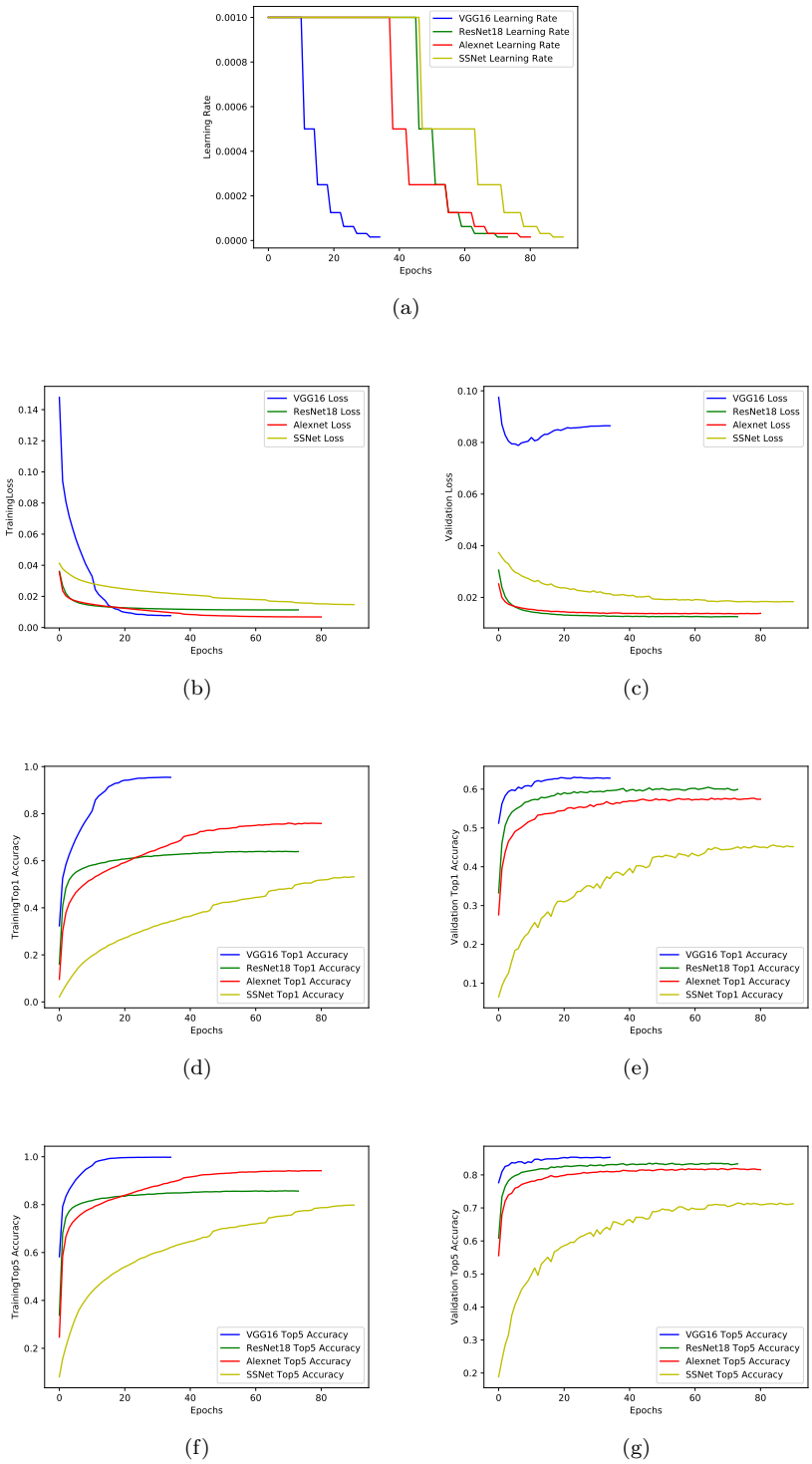

Figure 6: Comparison of SSNet with pretrained state of the art models VGG16, ResNet18, Alexnet: (a) Learning rate comparison (algorithm convergence); (b) Training loss comparison; (c) Validation loss comparison; (d) Training top-1 accuracy comparison; (e) Validation top-1 accuracy comparison; (f) Training top-5 accuracy comparison; and, (g) Validation top-5 accuracy comparison;

5. **Final Model Comparison with fine tuned state of the art trained from scratch:** Below figure 7 shows the graph of comparison of SSNet with state of the art model with training from scratch and fine tuning. These results are discussed in detail in the analysis and discussion of results section.

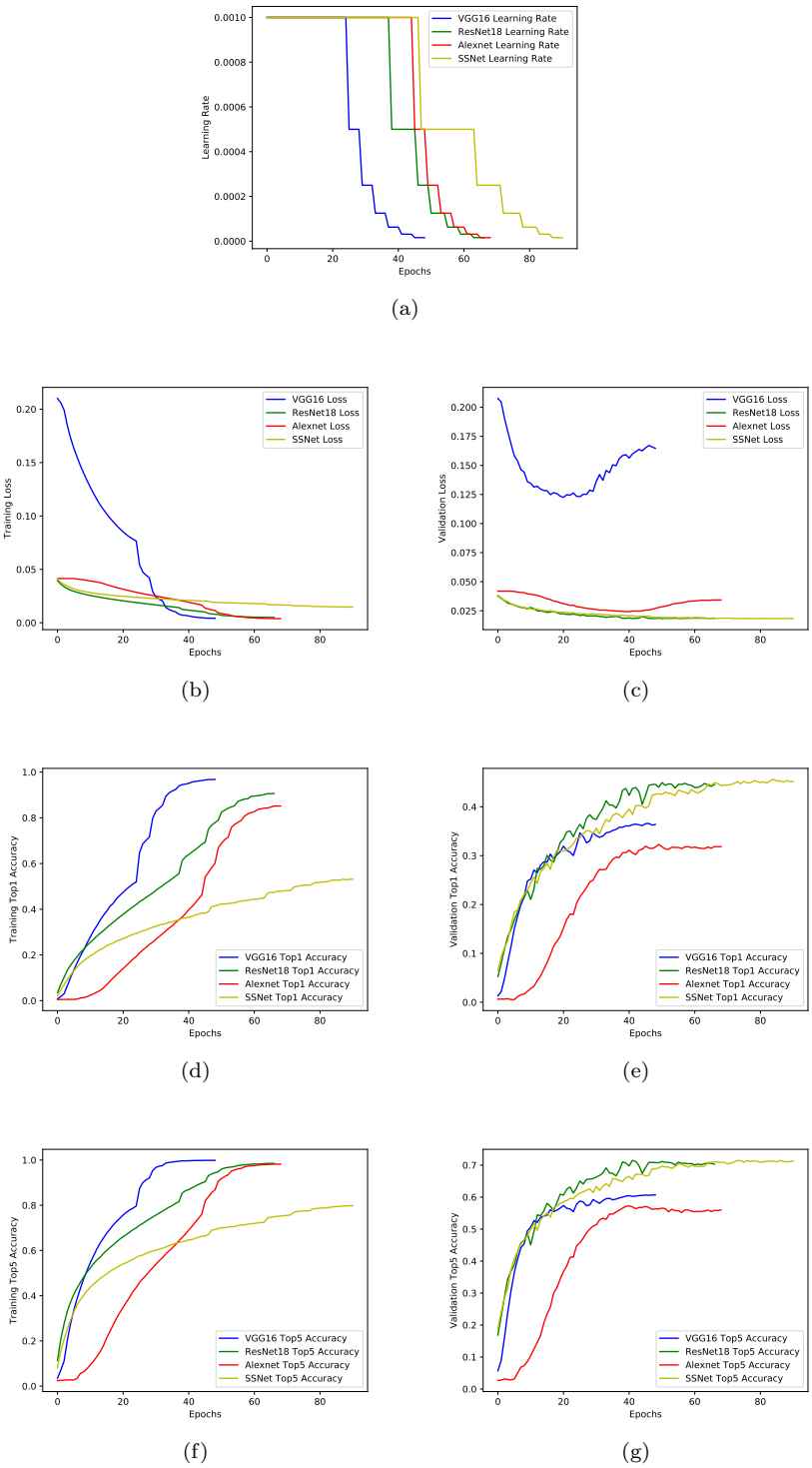

Figure 7: Comparison of SSNet with state of the art models VGG16, ResNet18, Alexnet trained from scratch: (a) Learning rate comparison (algorithm convergence); (b) Training loss comparison; (c) Validation loss comparison; (d) Training top-1 accuracy comparison; (e) Validation top-1 accuracy comparison; (f) Training top-5 accuracy comparison; and, (g) Validation top-5 accuracy comparison;

5 Analysis and Discussion of Results

1. **Weight Initialization and Regularization:** In figure3(a), I observed that my model converged faster with use of normalized initialization[7] of weights and L2 regularization. Also figure3(b) and figure3(c) clearly shows the comparison of loss calculated for training and validation and on both, loss is lower for model with normal distribution and with L2 regularization. The Top1 and Top5 accuracy of both training and validation confirms that model performs better with normalized initialization and L2 regularization.

2. **Data Augmentation:** In figures4 it is was expected that more augmentation will result in better training and hence better results. But the graphs clearly shows that its not the case. The reason could be that images already are of very poor quality with lot of noise. The effect of cropping is extremely huge on the performance of the model. This could be explained due to the fact that once a poor quality image is scaled to 255x255 from 64x64 and again cropped to 64x64 causes image to loose many features. Also to be noted that the noise too gets scaled and when I crop the 64x64 patch, most of it could be noise and less features. This in turn makes model learn noise more than learning the features. Hence even though the training accuracy increases, the model doesn't learn anything hence validation fails. This could have been better if the scaling was done to 80x80 and then ten 64x64 patch were extracted. However, I couldn't test it. The same behavior is observed when I combined many multiple data augmentation strategies as well. Hence I kept only random horizontal flip as data augmentation strategy for my model.

3. **Activation function:** In figures5 it can be observed that Leaky-Relu and Relu had comparatively similar result in performance both in training and validation. Model with Leaky-Relu performs better by 0.015% than Relu and take same amount of time to train. Hence I didn't got much motivation in using Leaky-Relu for my model. I also tested my model with sigmoid activation and as expected, it reduced the performance of my model by a large extent. This could be because of neurons saturating and gradient being killed due to the saturated neurons.

4. **Understanding the model:** To understand my final model with clarity, I plotted the confusion matrix to see how well my model is correctly able to identify the correct class and with what confidence. Figure8 shows the confusion matrix for SSNet. Confusion Matrix was plotted for whole 200 classes and then the matrix was cropped with size 20x20. As can be observed from confusion matrix, SSNet is able to classify most of the classes with strong confidence. However, it does get confused on some classes. For example, the class "n01641577" i.e frog is highly confused with the class "n01644900" i.e toad. Upon checking it manually, I too got confused to see which was toad and which was frog. An example image from the two classes is shown below in figure9. However, model is able to very strongly classify the image if its a duck (n01855672) or not with probability of 94%.

(a) (b)

Figure 9: Confused toad and frog image as evidenced in confusion matrix: (a) Frog; (b) Toad;

To be able to visualize the prediction of images, I also plotted the random images with prediction to see how well my model works. Below figure10 shows the result of prediction -

5. **Final model comparison with pretrained fine tuned state of the art models:** In the figures6, SSNet is compared with the pretrained models AlexNet, Resnet18, and VGG16. Since I had freezed the pretrained weights in fine tuning, only the last fc layer was trained to classify on 200 classes instead of 1000. It is clearly observed that all the three pretrained

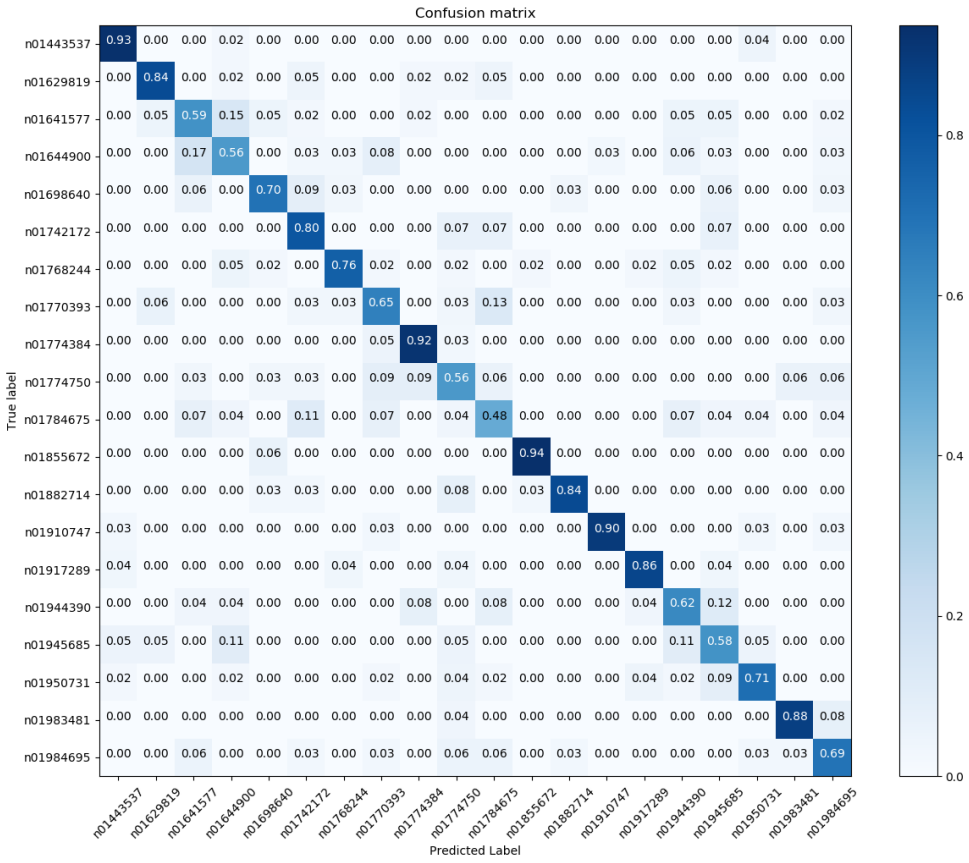

Figure 8: SSNet Model confusion matrix

models outperforms SSNet trained from scratch. From the figure6(a), it is observed that VGG16 took least epochs to converge. However, the training time was maximum for VGG16. On GTX1080 8GB memory with training only the last layer, VGG16 took 5.5 hours while, ResNet18 took 2 hours, Alexnet took 1.2 hours to train and my model took 5.2 hours to train from scratch. VGG16 and AlexNet seemed to overfit as training top1 and top5 accuracy is 20% more than validation top1 and top5 accuracy. Infact, VGG16 acheives near perfect top5 accuracy while validation top5 accuracy is about 83%. Resnet18, on the other hand, seems to perform well. It acheives training top1 accuracy as 60% while validation top1 accuracy as 58%. SSNet's performance was not as good when compared with pretrained models however, it didn't overfit and performs consistently due to the L2 regularization, data augmentation, batch normalization and normalized initialization of weights. The final ranking was 1st VGG16, 2nd ResNet18, 3rd AlexNet and at the last SSNet.

6. **Final model comparison with state of the art model trained from scratch:** The last comparison of state of the art models with SSNet was on models trained from scratch. Figures7 shows the performances of each model. Figure7(a) shows the similar result as observed in the previous comparison with pretrained. VGG16 again converges in least number of epochs, however, this time VGG16 took approximately 20 hours on the same configuration, ResNet18 took approximately 5.5 hours to train, AlexNet took 2 hours and SSNet took 5.2 hours. All state of the art models suffers from severe overfitting this time. VGG16 achieved training top-1 accuracy as good as 92% while validation top-1 accuracy only at 35%. This could also be observed from the figure7(c) that shows the VGG16 validation loss starts

Figure 10: SSNet visualizing the predictions. Out of 6 random images, 4 are classified correctly.

increaing hence confirming the overfitting. Similar were the results of ResNet18 and AlexNet as well. ResNet18 achieved 85% top1 training accuracy but achieved only 44.15% validation top-1 accuracy. Alexnet achieved 80% training top1 accuracy and achieved only 32% in validation top-1 accuracy. Same can be observed from the results of top-5 training and validation accuracies as well for all the state of the art models. SSNet, as observed above with pretrained, performs well. It doesn't overfit and outperforms all state of the art models used in this report. SSNet achieved 53% in top-1 training accuracy and 45.7% in top-1 validation accuracy. Similarly for top-5, SSNet achieved 78% top-5 training accuracy and 72% top-5 validation accuracy. The final standing of models were 1st SSNet, 2nd ResNet18, 3rd VGG16 and last AlexNet.

6 Conclusion

In this project, I have studied state of the art models like AlexNet, ResNet, VGG and GoogleNet. I created my own model named as SSNet and compared its performance with state of the art pretrained models and models trained from scratch on Tiny ImageNet dataset. Where VGG16 performs best in pretrained model however all models suffered from overfitting. SSNets outperforms all state of the art (trained from scratch) without overfitting by reaching the top-1 validation accuracy of 45.7% and top-5 validation accuracy of 72%.

Finally, I conclude that models performs well when they are trained on ImageNet dataset, and then transfered to Tiny-ImageNet dataset. I believe that if SSNet is trained on ImageNet dataset, and then tested on tiny-imagenet dataset, it can achieve comparable performance with the pretrained models as well.

7 Future work

There are several ideas that I haven't tested to improve the accuracy of SSNet. Also my hypothesis that I stated in conclusion can be tested by training the SSNet on ImageNet dataset and then

testing it on tiny-imagenet dataset. Also I can reduce unnecessary parameters in the models using techniques described by the authors of SqueezeNet[9].

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[7] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[8] T.-W. Huang, "Tensorboard-Pytorch," https://github.com/lanpa/tensorboard-pytorch, [Online; accessed 20-February-2018].

[9] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360