Image classification using Tiny ImageNet

Gustav Lindström, Ludwig Händel, Thomas Brunner

May 23, 2018

DD2424 Deep Learning in Data Science
Gustav Lindström, 941102-6652, gulinds@kth.se
Thomas Brunner, 920514-0792, tbrunner@kth.se
Ludwig Händel, 930531-1574, lhandel@kth.se

Abstract

In this paper, three different convolutional neural network, AlexNet, VGG-Y and VGG-F has been implemented to approach image classification of the Tiny ImageNet dataset. The models was inspired by *Alexei Bastidas work, Tiny ImageNet Image Classification* [1]. After fine-tuning and evaluating the performance of the three models, AlexNet manage to reach the highest validation accuracy and the lowest loss value after 100 epochs. AlexNet reached a validation accuracy of 32.92 % and validation loss of 3.0195, but it converges less then the VGGNet models due to overfitting. All the models tend to plateau at around or below 30 % validation accuracy. The different convolutional neural network architectures used Batch Normalization, Dropout and L-2 regularization to avoid overfitting. The models performance was analyzed by evaluating their validation accuracy and loss.

Contents

1 Introduction

For a human brain, object classification is easy, we can easily tell the difference between a cow and a horse. But this kind of problem is really difficult for a computer to solve and has always been a core task within Computer Science. The challenge for this kind of task is to extract quantifiable features from a 3 different channels, for each color, red, green and blue, pixel matrix[16]m. In the past people used manually written algorithms for this task. This approach quickly become very complex when you were dealing with different types of images from different angels and did not scale well.

Convolutional Neural Networks (CNN) were originally introduced almost 10 years ago, but improvements in network structure and computer power have enabled the training of truly deep CNN only recently. Today researchers have reach amazing results for image classification using CNN but we continue to look for more computationally efficient and more accurate models to apply for this kind of task [16].

This report focus on implementing and training three different deep convolutional neural network models to perform image classification on the ImageNet dataset. The dataset is used in the yearly ImageNet Challenge where the goal is to perform image classification at a large scale [10]. To make things computational feasible given our limited resources, a subset of the ImageNet will be used containing 200 classes subset of the 1000 million images in the original ImageNet.
The architecture of the models will follow the VGG-F and VGG-Y architecture described in Alexei Bastidas report from 2015 and the AlexNet architecture developed by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever [17]. Different techniques such as batch normalization, regularization and dropout will be used to prevent overfitting our models. The result of the different models is, as in the original ImageNet challenge measured and compared using loss and accuracy over a large range of distinct image classes.

2 Problem Formulation

Alexei Bastidas, Stanford University, performed and implemented a Tiny ImageNet Image Classification, this report will replicate Alexei's report [1] except the convolutional neural network AlexNet will be covered instead of InceptionV3. The final result of this paper will be discussed and compared with Alexei's result. In his paper, he investigated existing models, InceptionV3 and VGGNet for benchmarking on the Tiny ImageNet dataset. The performance of the network was analyzed by evaluating the training loss and accuracy with the corresponding validation set.

3 Background

In this section, an overview of relevant background to understand this report is presented.

3.1 Convolutional Neural Network

A convolutional neural network, CNN or ConvNet, is a class of deep, feed-forward artificial neural networks. A CNN consists of an input and output layer and multiple hidden layers in between. These hidden layers generally consist of convolutional layers, normalization layers, pooling layers and fully connected layers [2].

The CNN architecture is designed to be able to make use of the 2D composition of an image as well as other 2D input e.g a speech signal. This can be done by local connections and weights followed by pooling which results in different features. The features of each convolutional layer are extracted through filters of size n x k where n, k is smaller than the dimension of the image. The filters make it possible for locally connected structures and these structures creates feature maps of the image. Typically max pooling is applied to each map feature map, for small images the size is often 2, which are used in this paper, but not larger the 5 for larger inputs [3]. CNN are great

at finding patterns and using them to classify images. CNNs is also preferably easier to train, less parameters compared to fully connected networks with the same amount of hidden layers. A CNN uses back propagation algorithm to calculate the gradient with respect to the models parameters [3] [4].

Optimization
In CNN optimization algorithms are used to maximize or minimize an error function. It is mathematical function used to adjust the parameters of the model. Weights(W) and the Bias(b) values of a neural network is internal learnable parameters used to compute the output of the model as well to learn and update the direction of the optimal solution to produce a accurate result. This can be done e.g minimizing the loss of a network's training process [5].

One of the most widely used optimization algorithms is the gradient descent, because it is easy to compute, less time consuming and converge relatively fast on datasets of larger scale [5]. There exist various of optimization algorithms to optimize gradient descent e.g momentum, adagrad and adam. Momentum adds a fraction x of the past vector to update the current, usually the value 0.9. Adagrad modifies the learning rate n at each step for every parameter based on the past gradients, no need to manually tune the learning rate but the disadvantage is that the learning rate always decreases. Adam stands for Adaptive Moment Estimation and is another method that computes adaptive learning rates for each parameter [5].

The preferable optimizer for a CNN depends on how the optimizer converge, learning and how it tunes the internal parameters to minimize the loss function. Adam is the current (2018-05-17) preferable optimizer in deep Neural Network or another Adaptive learning rate techniques [5].

Architecture
Different CNN architectures used in the ImageNet challenge for visual recognition are, AlexNet, VGGNet, GoogLeNet, ResNet, VGG.

AlexNet
The AlexNet architecture was one of the first deep neural networks to increase the ImageNet classification accuracy significant compared to earlier methods. It consist of 5 convolutional layers and 3 fully connected layers [6]. AlexNet uses ReLU activation function and dropout to deal with overfitting [7]. See figure 1.a for a visual representation of the AlexNet architecture.

VGGNet
Another CNN architecture is the VGG architecture, it's an improvement of AlexNet by replacing large sized filters with multiple of smaller sized filters. Multiple smaller stacked filters is used and preferable instead of fewer larger size filters because it increases the depth of the network and enables it to learn more complex features at a lower cost [6]. See figure 1.b - 1.c for a visual representation of the VGG-Y and VGG-F architecture inspired by Alexei Bastidas [1] paper.

3.2 Overfitting

Overfitting occurs when your model expressive capability is too high, meaning it has too many degrees of freedom [13]. A deep neural network with a large number of parameters are very powerful machine learning systems. However, overfitting is a potential and serious problem in such networks. There are a lot of techniques for addressing the overfitting problem. Hence, some of the techniques will be described bellow.

Dropout
Dropout is a technique for addressing the overfitting problem. The main idea is to randomly drop units along with their connections from the neural network during training [13]. By doing this units prevents from co-adapting too much. Dropout samples from an exponential number of different "thinned" networks during training phase. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network

that has smaller weights. This method truly gives major improvements and reduces overfitting over other regularization methods [13].

Batch normalisation
Batch normalisation is a technique for improving the performance and stability of neural networks. [9] The idea is to normalise the inputs of each layer in such a way that they have a mean output activation of zero and standard deviation of one. This is analogous to how the inputs to networks are standardised.

Regularization
Regularization is a way to avoid overfitting by penalizing high-valued regression coefficients [14]. In simple terms, it simplify the model by shrinking it and reduces parameters. Two regularization techniques used to address overfitting are: L1 and L2-regularization. A common regression model that uses L1-regularization is called Lasso Regression and a common model which uses L2-regularization Ridge Regression. The primary difference between L1 and L2 is the penalty term. L1-regularization adds "absolute value of magnitude" of coefficient as penalty term to the loss function [15]. Thus, it limits the size of the coefficients. Note that the last term is the L1-regularization element.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=0}^{p} \mid \beta_j \mid \tag{1}$$

L2-regularization adds "squared magnitude" of coefficient as penalty term to the loss function. L2 will not yield sparse models and all coefficients are shrunk by the same factor, which means that none are eliminated [15]. Note that the last term is the L2-regularization element.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=0}^{p} \beta_j^2 \tag{2}$$

3.3 Data

This experiment uses the same dataset as Alexei Bastidas, which is the Tiny ImageNet [1]. The Tiny ImageNet dataset consists of the same data as the original ImageNet but the images are cropped into size of 64x64 from 224x224. It has 200 classes and 500 training images for each of the classes, resulting in a training data of 100 000 images [8]. In addition to the training data, it has 10,000 validation images and 10,000 test images (50 for each class). It uses one-hot labels. Some ambiguities in the dataset is caused since it is down-sampled from 224x224 to 64x64. The effect of this down-sampling includes loss of details which makes it harder to locating small objects.

3.4 Software library

The models where implemented with ersion r1.8 TensorFlow and Keras which is a higher level API that runs on top i.a. TensorFlow.

TensorFlow
TensorFlow is an open source software library for high performance numerical computation [12]. It is cross-platform, which means it runs on nearly everything: GPUs and CPUs including mobile and embedded platforms. It was originally developed by the Google Brain Team within Google's Machine Intelligence research organization for machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other scientific domains as well.

Keras
Keras is a popular programming framework for deep learning that simplifies the process of building deep learning applications [11]. Instead of providing all the functionality itself, it uses either TensorFlow or Theano behind the scenes and adds a standard, simplified programming interface on top.

4 Approach and Experiment

The models will be implemented using the open-source Google deep learning framework Tensorflow, stacked with the higher level API, Keras. Keras is designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

4.1 Data Augmentation

Data augmentation was implemented using Keras built in ImageGenerator [18] for all models while training. ImageGenerator generates batches of tensor images that are rescales, zoomed-in and flipped-version of the original images. This was done to help train a more accurate and general model.

4.2 Weight initialize

To initializing the weight for the models we used the same distribution as Alexei Bastidas, the Glorot Uniform Initializer or Xavier Uniform Initializer as it also called.

4.3 VGG-F

The VGG-F model follows the same architecture described by Alexei Bastidas [1] with 12 stacked convolutional layers along with a couple max pooling layers, to reduce the spatial dimension of the data. Every single layer is followed by batch normalization and ReLU activation. To prevent overfitting batch normalization were used. L2 regularization were used on the two last fully connected layers with a regularizer of 1e-4. The same learning rate as Alexei Bastilles were used on the model starting at 1e-3 and decreases over time during each epoch with a factor of 0.5 every 20 epoch.

4.4 VGG-Y

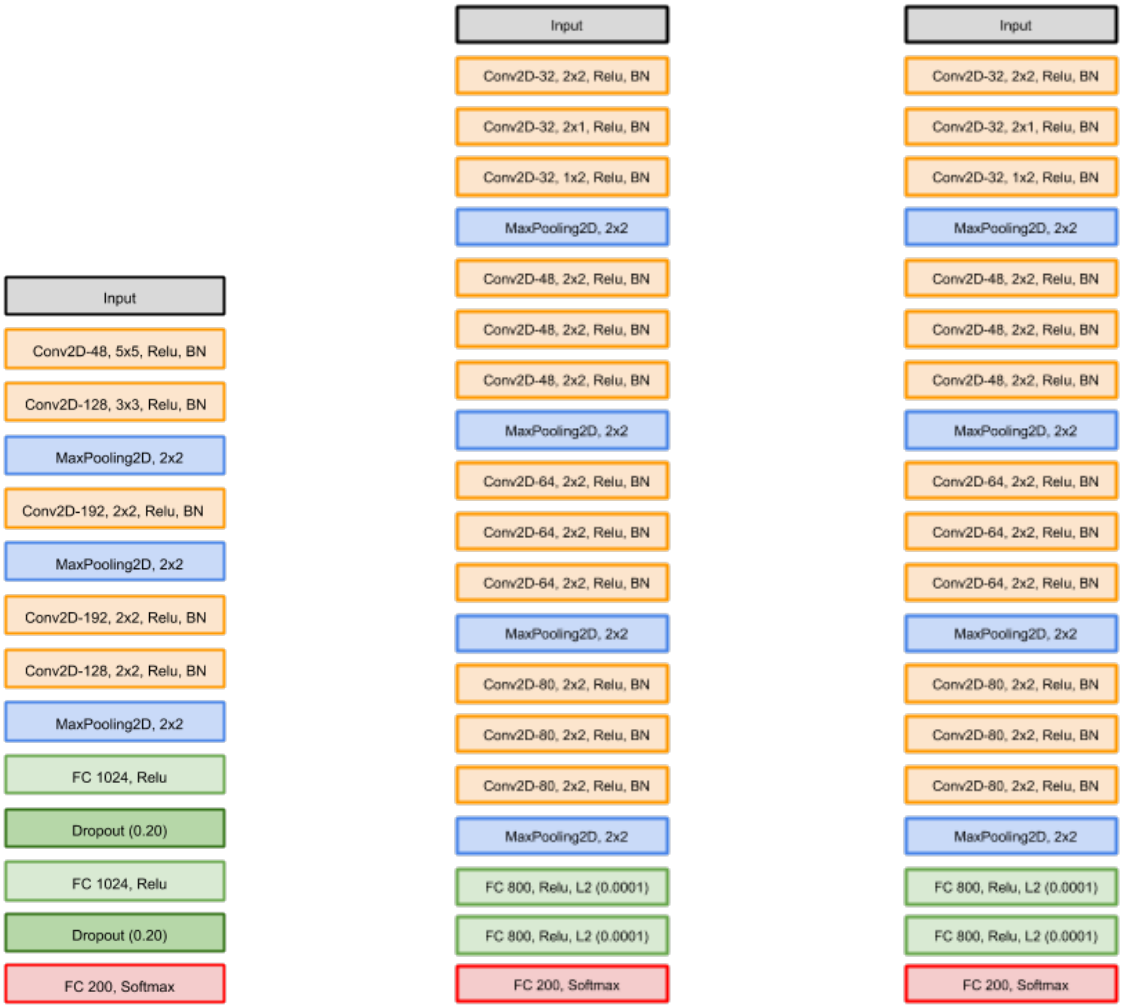
The VGG-Y model follows a similar approach as the VGG-F architecture but with less layers. VGG-Y has 9 convolutional layer stacked along with a couple max pooling layers, to reduce the spatial dimension of the data. Every single layer is followed by batch normalization and ReLU activation. VGG-Y also uses batch normalization to prevent overfitting. L2 regularization were used on the two last fully connected layers with a regularizer of 0.01. The same learning rate as Alexei Bastilles were used on the model starting at 1e-3 and decreases over time during each epoch with a factor of 0.5 every 20 epoch.

4.5 AlexNet

The AlexNet model follows the architecture described by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever with 5 convolutional layer stacked along with a couple max pooling layers [17]. To prevent overfitting dropout were used on the two last fully connected layers with a dropout rate of 0.2. Because AlexNet were originally designed to work on the original ImageNet dataset containing larger images a similar approach as the one described by Alexei Bastidas were applied a to make up for the reduced input size. This was done by reducing the size of the filters on the convolutional layers, from 11x11 to 5x5, 5x5 to 3x3 and 3x3 to 2x2. Because batch normalization were not used a lower learning rate was needed. This model were trained with a initial learning rate of 1e-4 and and decreases over time during each epoch with a factor of 0.5 every 20 epoch.

4.6 Final Architectures

Figure 1 presents the final architecture of the models.



(a) Reduced AlexNet (b) VGG-F (c) VGG-Y

Figure 1: Final Architectures

5 Results

In this section, the result of the different CNN models are presented. The graphs shows the individual performance of the models with respect to training loss and accuracy as well as validation loss and accuracy. The models were all trained with parameters presented in section 4.

Based on Table 1, we can see that AlexNet performed best during both training and validation accuracy after 100 epochs. Although, the differences between the different models scores were very low. AlexNet validation accuracy ended up at 32.92 % compared to VGG-Y and VGG-F which respectively achieved a validation accuracy of 32.25 and 31.41 %. AlexNet also achieved the best final results of training and validation loss, a training loss of 2.1373 and a validation loss of 3.0195.

		Accuracy		Loss	
		Training	Validation	Training	Validation
Model	AlexNet	0.4673	0.3292	2.1373	3.0195
	VGG-Y	0.3986	0.3225	2.6369	3.0675
	VGG-F	0.4042	0.3141	2.4904	3.0908

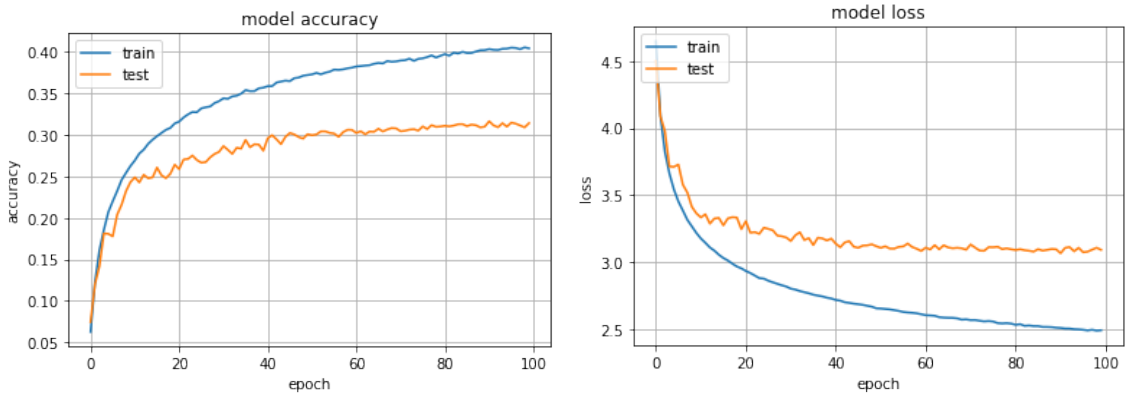
Table 1: Accuracy and Loss results for the Models.

AlexNet quickly converged during the first 15 epochs, then it was subjected to overfitting and then modestly converge against the training curve. Both VGG-Y and VGG-F accuracy and loss function converged better than AlexNet in its entirety and were less affected by overfitting. Within 10 epochs, accuracy increased rapidly and lost function decreased at the same speed, after about 25 epochs, the learning process decreased and continued modestly increasing during the rest of the epochs. All of the three models tend to plateau at around 30 % validation accuracy.
In table 2, the total time training each of the individual CNN architecture are presented. The most efficient architecture to train was VGG-Y, it manage to train and validate the Tiny ImageNet dataset within 4.44 hours, while VGG-F took 4.47 hours and AlexNet took almost twice the time, 8.33 hours.

Model	Training Time
AlexNet	8.33 h
VGG-Y	4.44 h
VGG-F	4.47 h

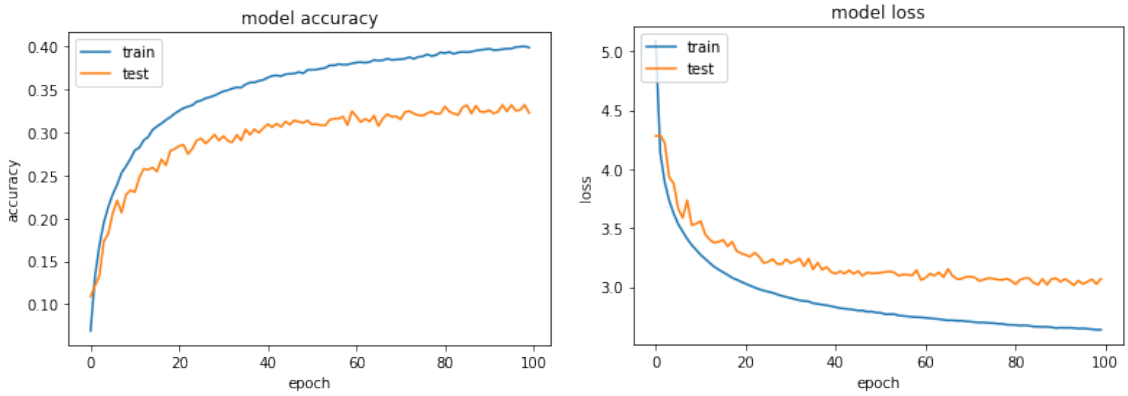
Table 2: Training time of the different models.

The figures (2,3,4) bellow shows training time loss and accuracy as well as validation loss and accuracy. The VGG models were all trained with an initial learning rate of 1e-3 and a learning rate drop by a factor of 0.5 every 20 epochs. AlexNet where trained with a lower learning rate of 1e-4, due to the lack of batch normalization.



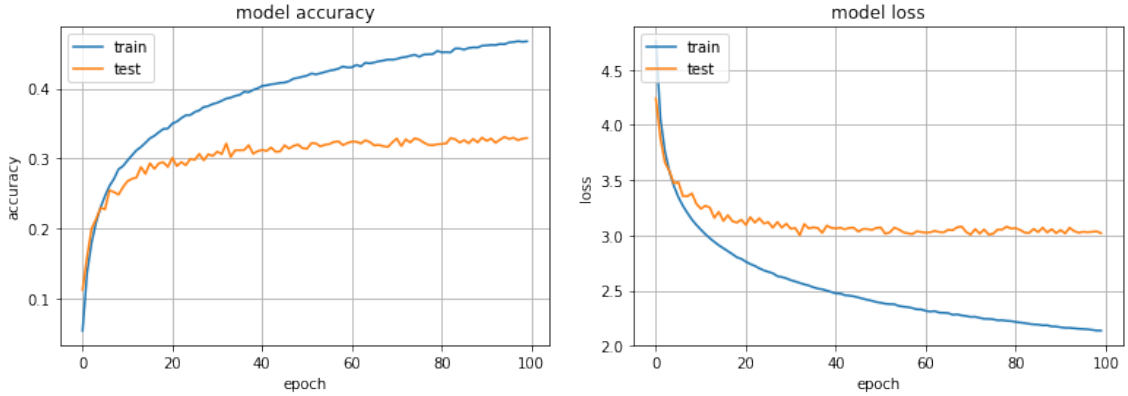
(a) Accuracy (b) Loss

Figure 2: VGG-F



(a) Accuracy (b) Loss

Figure 3: VGG-Y



(a) Accuarcy (b) Loss

Figure 4: AlexNet

6 Conclusions

In this paper the convolutional neural networks AlexNet, VGG-Y and VGG-F has been implemented and trained from scratch on the Tiny ImageNet dataset. The performance of the network was analyzed by evaluating the training loss and accuracy with the corresponding validation set.

The final result was not the expected result compared to the ImageNet challenge (hypothesis) [8] because VGGNet is an improvement of AlexNet, with deeper architecture, multiple layers and smaller kernels. One possible reason for AlexNet performing better could be the number of epochs, as we can see in figure 2-4, the convergence between the training and validation of the VGG models and AlexNet differs, the VGG models converge better than the AlexNet, which means that VGG architectures may be able to outperform AlexNet by increasing the number of epochs. We can conlude that the VGG-Y and VGG-F models minimized overfitting by looking at their training and validation curves. The gap between AlexNet validation and training curves where wider, this means that AlexNet model was subjected of overfitting. It's difficult to only follow the hypothesis from the ImageNet challenge statistics because the implemented models in this paper are variants of the standard architectures of AlexNet and VGGNet, despite the expected outcome, however, the VGGNet was believed to perform better than AlexNet. All architectures were straight forward to implement. AlexNet took the longest time to train, exactly 8.33 hours compared to VGG-Y and VGG-F who took 4.44 respectively 4.47 hours. A reason why AlexNet took longer was the number of filters on the convolutional layers.

A comparison was made between the result from this papers experiments and the results from Alexei Bastidas paper. The models in this paper are based on the same architecture as Alexei. However, some of the parameters are different. Since, Alexei's paper did not provide information about all the parameters that were needed. Thus, our results on training/validation of accuracy and loss differs from Alexei's results. Table 3. our models and Alexei's models performance on training and validation of the accuracy.

		Our - Accuracy		Alexei's - Accuracy	
		Training	Validation	Training	Validation
Model	VGG-Y	0.3986	0.3225	0.2911	0.2911
	VGG-F	0.4042	0.3141	0.4721	0.4131

Table 3: Comparison accuracy of the models.

Table 4. compares our models and Alexei's models performance on training and validation of the loss.

		Our - Loss		Alexei's - Loss	
		Training	Validation	Training	Validation
Model	VGG-Y	2.6369	3.0675	3.2248	3.2248
	VGG-F	2.4904	3.0908	2.7636	2.4724

Table 4: Comparison accuracy of the models.

We can clearly see from table 3. that Alexei's VGG-F model outperforms our model. It also outperforms our best model which is AlexNet. However, our version of VGG-Y outperforms Alexei's VGG-Y model, but the performance of the models is close; ~ 0.29 to ~ 0.32. Fine tuning on the parameters had to be done in order to achieve the same performance as Alexei's models.

If we would have had more resources and time we could have found parameters that provided more similar or even better result than Alex. Regarding the reduced AlexNet we experimented with fine-tuning the parameters for the number of filters and their sizes together with the parameter for the dropout. With more time and resources this could probably be improved by doing a more detailed search for the best parameters.

For future work, improvement of hardware equipment, GPUs, to be able to make the training process more efficient, allowing more epochs of training to evaluate different possible outcomes. Evaluate the effect of more fine-tuning of parameters to optimize the results, add or remove parameters to eliminate unnecessary parameters in the models to increase the efficiency of the model as well as performance. Alternatively, try more advanced architectures such as ResNet and InceptionV3 to see how they perform on the Tiny ImageNet dataset.

7 References

[1] *Bastidas, A. (2017)* Tiny ImageNet Image Classification. Retrieved from $http://cs231n.stanford.edu/reports/2017/pdfs/940.pdf$

[2] *Convolutional Neural Network - Wikipedia.* (2018, May 21). Retrieved from $https://en.wikipedia.org/wiki/Convolutional_neural_network$

[3] *Convolutional Neural Network.* (2018, May 20). Retrieved from $http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/$

[4] *Convolutional Neural Networks for Visual Recognition.* (2018, May 14). Retrieved from $http://cs231n.github.io/neural-networks-2/$

[5] *Walia, A. S.* (2017, June 10). Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent. Retrieved from $https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-$

[6] *ResNet, AlexNet, VGGNet, Inception:* Accessed: 2018, May 10. Understanding various architectures of Convolutional Networks. Retrieved from $http://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/$

[7] *Prabhu.* (2018, March 15). CNN Architectures - LeNet, AlexNet, VGG, GoogLeNet and ResNet. Retrieved from $https://medium.com/@RaghavPrabhu/cnn-architectures-lenet-alexnet-vgg-googlenet-and-resn$

[8] *Li, F. Justin Johnson, J. Yeung, S.* (May 2, 2017) Lecture 9 - CNN Architectures, Stanford. Retrieved from $http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf$

[9] *Jaron Collis.* (Jun 27, 2017) Glossary of Deep Learning: Batch Normalisation Retrieved from $https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2$

[10] *subhasis@stanford.edu.* (May 14, 2018) Glossary of Deep Learning: Batch Normalisation Retrieved from $https://tiny-imagenet.herokuapp.com/$

[11] *Keras.* (May 14, 2018) Keras Retrieved from $https://keras.io/$

[12] *TensorFlow.* (May 14, 2018) About TensorFlow Retrieved from $https://www.tensorflow.org/$

[13] *Amar Budhiraja.* (Dec 15, 2016) Dropout in (Deep) Machine learning Retrieved from $https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-lear$

[14] *Anuja Nagpal.* (Oct 13, 2017) L1 and L2 Regularization Methods Retrieved from $https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c$

[15] *Stephaine.* (Oct 27, 2016) Regularization: Simple Definition, L1 & L2 Penalties Retrieved from $http://www.statisticshowto.com/regularization/$

[16] *DD2323 lecture 1.* (Mars 18, 2018) Introducing the deep learning revolution.

[17] *AlexNet.* (May 18, 2018) Retrieved from $http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet$

[18] *Keras.* (May 19, 2018) Retrieved from $https://keras.io/preprocessing/image/$