Deep Convolutional Neural Networks for Tiny ImageNet Classification

Hujia Yu
Stanford University
hujiay@stanford.edu

Abstract

In this project, I approached image classification problem by implementing and training from scratch three state-of-art model structures, AlexNet, GoogLeNet, and ResNet on the TinyImageNet dataset. After fine-tuning and evaluating the performance on all three models, I used my best performing model, which is ResNet-20 with 55.4% validation accuracy, on the test set, and achieved test accuracy of 45% (error rate of 55%). During this process, I found out that most of the models trained from scratch tend to plateau at around or below 50% validation accuracy. It turns out that training state-of-art model architectures from scratch on reduced dimension of image datasets do not have as good performances as the original models on the Large Scale ImageNet Challenge. My intuitions of this phenomenon are that image downsampling causes loss of details and ambiguity during training, and constraint on computing power and model capacity also determined how far the model accuracies could go. For future improvement, I will improve on computing power as well as model simplicity to allow for more efficient training. I will also look into my best model, ResNet-20, to see if residual connections significantly improve fine-tuning and training efficiency in general.

1. Introduction

Our brain is great at vision. It can easily tell apart a dog and a cat, read a sign, or recognize detailed objects in a picture. But these are actually really hard problem to solve for a computer. It only seems easy because our brain is incredibly good at understanding images. Therefore, a large amount of research work has been put in to the field of computer vision. Convolutional networks(ConvNets) have enjoyed a great success in large-scale image and video recognition, and it has become popular due to large image dataset availability, such as ImageNet(Deng et al., 2009)[10], and high-power computing systems, such as GPUs. Although ConvNets were originally introduced over 20 years ago, improvements in network structure and computer hardware have enabled the training of truly deep ConvNets only recently. Since the 2012 ImageNet competition winning entry by Krizhevsky et al.[1], their network AlexNet has been applied to variety of computer vision tasks, for example to object detection. Successful models like AlexNet has great influence on research that focused on deeper and better-performing network structures. In 2014, the quality of network architecture significantly improved by using deeper and wider networks. VGGNet[7] and GoogLeNet[5] both yielded high performance in ILSVRC 2014 classification challenge. This paper uses dataset from Tiny ImageNet Challenge, which is a simplified version of Large Scale Visual Recognition Challenge 2016(ILSVRC2016)[10]. The details of the dataset is described in the section below. In this paper, I trained from scratch the reduced capacity models of GoogLeNet, AlexNet, and ResNet[6] on the dataset, and I explored and compared the results to fine-tune and improve the models. As in ImageNet classification, the objective of training models is to attain best performance on image classification problem.

2. Related Work

ConvNets architectures such as GoogLeNet, VGGNet and AlexNet are highly successful architectures in recent ILSVRC, and they have spurred a great interest in ConvNets architecture improvements. VGGNet has the compelling feature of architectural simplicity, but it is computationally expensive and inefficient. On the other hand, the Inception architecture of GoogLeNet performs well under strict constraints on memory and computation power. It employed only 5 million parameters, which represented a 12X reduction with respect to AlexNet, which uses 60 million parameters. VGGNet employed around 3X more parameters than AlexNet. Recently, SqeezeNet is a deep neural network that achieves accuracy level of AlexNet but with 50x fewer parameters by methods of sharing parameters,etc.[8], which has a good balance between performance and memory/computational resource usage.

2.1. AlexNet

AlexNet [1] is the first work that popularized ConvNets in computer vision. It features convolutional lay-

ers stacked on top of each over. AlexNet consists of a total of 8 layers, which are 5 convolutional layers and 3 fully-connected layers (final layer outputs the class labels). Batch-normalization is applied after the first two convolutional layers. Dropout is applied after each layer during the last two FC layers.

2.2. GoogLeNet/Inception

The goal of the inception module is to act as a "multi-level feature extractor" by computing 1x1, 3x3, 5x5 convolutions within he same module of the network, then stack these outputs together along the channel dimension and before being fed into the next layer the network. This model also dramatically reduced the number of parameters in the network. It uses average pooling instead of fully connected layers on top of ConvNet, enabling it to eliminate a large amount of parameters since pooling does not introduce extra parameters [3].

2.3. VGGNet

VGGNet was introduced by Simonyan and Zisserman in 2014[7]. This network is characterized by its simplicity. It contains 16 CONV/FC layers and features an architecture that only consists of 3x3 convolutions and 2x2 pooling from the beginning to end. Even though it performs well, the downside of the VGGNet is that it is computational expensive to train and uses significant amount of memory and parameters(140M).

2.4. ResNet

Residual network is developed by Kaiming He et al. [6]. It is a special case of Highway Networks. It is the first to implement 152-layer neural networks(the deepest). It involves a heavy use of batch normalization, and it also misses fully connected layers at the end of the network. It is the state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 10, 2016).

Usually when we train a deep convolution neural nets, one common problem is vanishing gradients. However, authors of ResNet believe that it is actually an optimization problem[6]. Therefore, ResNet is proposed to have the gradients flow like Highway Networks to avoid degradation problem. The inputs of a lower layer is made available to a node in a higher layer. It is similar in structure with that of a LSTM but without gates.

2.5. SqueezeNet

SqueezeNet is a small CNN architecture that achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters, aiming to be deployed smaller CNN model on mobile devices[8]. Together with model compression techniques it is able to compress SqueezeNet to less than 0.5MB (510 smaller than AlexNet), which can fully fit on embedded device. Its major three strategies to downsize a model are:

1. Replace 3x3 filters with 1x1 filters. This reduces parameters by 9X less.
2. Decrease the number of input channels to 3x3 filters
3. Down-sample late in the network so that convolution layers have large activation maps

According to its paper[8], SqueezeNet uses a specific architecture named 'fire module' as the basic building block for the model. which is composed of 'squeeze layer' and 'expand layer'. The 'squeeze' layer is a convolution layer made up of only 1x1 filters, and the 'expand' layers are convolution filters with a mix of 1x1 and 3x3 filters. By reducing the number of filters in the 'squeeze' layer feeding into the 'expand' layer, total number of parameters are thus reduced.

In addition to reducing number of parameters in the model, the authors [8] also believe that creating a larger activation/feature map later in the network, classification accuracy actually increases, as illustrated in strategy 3 above. Having larger activation maps near the end of the network is in stark contrast to networks like VGG where activation maps get smaller as we get closer to the end of a network. However it turns out to work pretty well.

SqueezeNet also has some variants that can reduce parameters by significantly more, which is what they call 'deep compression'. With 'deep compression', the original SqueezeNet adds bypass or complex bypass on top of it, which simply bypasses some fire modules in the model to decrease parameters[2].

3. Dataset and Features

This project uses dataset from Tiny ImageNet Challenge. The original ILSVRC2016 dataset is a large set of hand-labeled photographs consisting of 10,000,000 labeled images depicting 10,000+ object categories. These pictures are collected from flickr and other search engines, labeled with the presence of absence of 1000 object categories. The Tiny ImageNet dataset consists of the same data but the images are cropped into size of 64x64 from 224x224. It has 200 classes instead of 1,000 of ImageNet challenge, and 500 training images for each of the classes. In addition to its 100,000 training data, it has 10,000 validation images and 10,000 test images (50 for each class). It uses one-hot labels. Down-sampling of images causes some ambiguities in the dataset since it is down-sampled from 224x224 to 64x64. The effect of this down-sampling includes loss of details and thus creating difficulty of locating small objects. Images are normalized by subtracted its mean before

Layer	Dimension
Input	64x64x3
CONV1-16	56x56x16
ReLU	
Pool-2	28x28x16
FC1-1024	1x1x1024
ReLU	
Output	1x1x200

Table 1. Baseline Model

training and testing in order to 'center' the data. This way the features have a similar range so that when we do back propagation, out gradients don't go out of control. I will report top-1 accuracy rate, which is the fraction of test images that are correctly classified by the model, to measure the performance. Tensorflow is used to build all models in this project.

4. Approaches

4.1. Models

4.1.1 Baseline Model

The baseline model that I implemented is just a simple one-layer input-CONV-ReLU-pooling-FC-ReLU-output model that we implemented in assignment 2. I used filter size of 16 for the convolution layer. After 10 epochs of training, this model was able to converge to training accuracy of 34.40% , and validation accuracy is 33.40% . This serves as a benchmark for the following models.

4.1.2 Reduced Capacity AlexNet

In this project, I implemented reduced capacity AlexNet architecture since the original architecture was trained on a different dimension. The reduced capacity AlexNet architecture is shown in Table 1 above. It is composed of 8 layers in total, which is 5 CONV layers and 3 fully-connected layers (the third FC layer is the output layer). There are one 7x7, two 5x5, two 3x3 kernels in total for each of the CONV layers, resulting in slow reduction of dimensions after each ConvNet. Stride is 1 across all layers, and padding is zero. Cyclic learning rate is used, which starts at 0.1 and decreases over time during each epoch. This model has a total of 15 million parameters, which is quite large and thus slow to train. One epoch took around 1.5 2 hours to train on a GPU, and around 10 hours on a CPU. Due to the long training time, only 10 epochs were trained and the result is presented in the result section below.

Layer	Dimension
Input	56x56x3
CONV1-64	56x56x64
CONV1-64	56x56x64
Pool-2	28x28x64
Batch-norm	28x28x64
CONV2-128	28x28x128
Pool-2	14x14x128
Batch-norm	14x14x128
CONV3-256	14x14x256
CONV3-256	14x14x256
Pool-2	7x7x256
Flatten	1x1x(7*7*256)
FC1-1024	1x1x1024
FC2-1024	1x1x1024
Output	1x1x200

Table 2. Reduced Capacity AlexNet Architecture

4.1.3 Reduced Capacity Inception model

I also trained from scratch GoogLeNet Inception model with reduced dimensions on the Tiny ImageNet dataset. The model architecture is similar to that of the original, but with output dimensions for each modules cut in half. It's composed of 1 stem network, 9 inception modules, 1 output classifier, and 2 auxiliary classifiers. As shown in the figure below, one inception module is a 'network in network' structure. I implemented the module with 6 convolution layers(four 1x1, one 3x3, one 5x5) and one pooling layer of size 3x3. The layers process and extract different features of the input, the output is then concatenated at the end before it is passed to the next module, which each level representing different features extracted from the module. Two auxiliary classifiers are added after inception level 3 and level 6 to help with gradient back propagation. This module structure dramatically reduces the number of parameters to train by using small filters such as 1x1 and 3x3. The entire model has 1.1 million parameters to train, which is significantly less parameters than AlexNet.

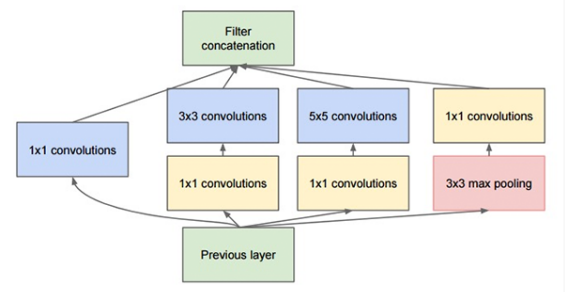

Figure 1. Inception Module Structure Used in GoogLeNet

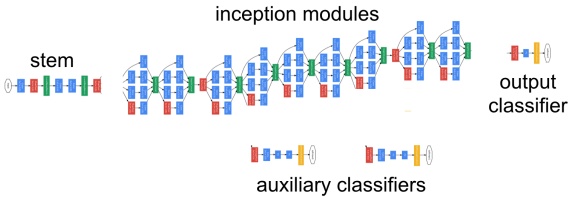

Figure 2. GoogLeNet Architecture

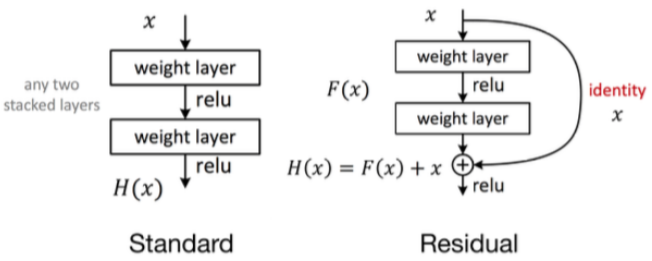

Figure 3. Residual connection vs. standard connection. We see that for residual connections, we are learning a residual mapping $F(x)$ instead of $H(x)$.

4.1.4 ResNet-20

ResNet is only composed of small sized-filters of 1x1 and 3x3, except the input convolution layer of 7x7. Every single layer is followed by batch normalization and ReLU activation. Figure 3 shows that the basic building block of the ResNet architectures is designed to learn residual functions $F(x)$ where the residual function is related to the standard function learned $H(x)$ by

$$H(x) = F(x) + x. \tag{1}$$

The authors[6] believed that the ideal $H(x)$ learned by any model is closer to the identity function x than random and as such, instead of having a network learn $H(x)$ from randomly initialized weights, we save training time by instead learning $F(x)$, the residual[13]. Additionally by introducing these identity functions x, we allow easier training by allowing the gradients to pass unaltered through these skip connections to alleviate traditional problems such as vanishing gradients. The ResNet architecture that I trained is composed on 20 layers, it is shown in Figure 4 and differs from the original ResNet architectures by having less replicas and less spatial pooling(2x, 2x, 2x), which occurs during the first conv-batch-relu for every basic block filter size change(three poolings in total). For ResNet-20, I trained the models using momentum based SGD with momentum of 0.9 starting with a cyclic learning rate of 0.1, batch size of 256, weight decay of 0.0001, and 10 epochs. Result is presented in the table below.

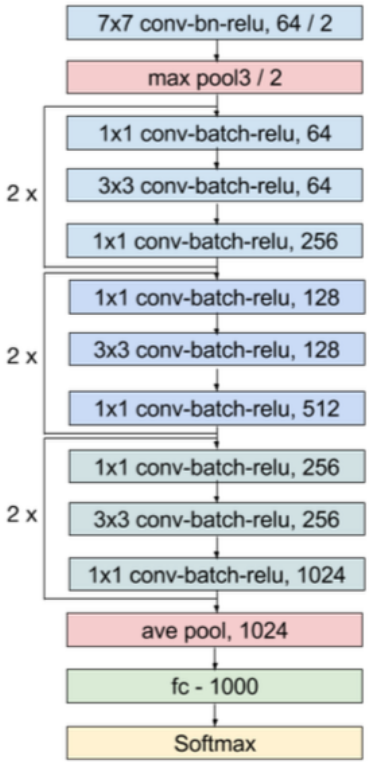

Figure 4. Structure of Reduced-Layer ResNet-20. This architecture involves less replicas of the buildings blocks and less spatial poolings. Spatial pooling is not shown in the diagram. This pooling occurs during the first conv-batch-relu for every basic block (three in total).

4.2. Regularizers

4.2.1 Data Augmentation

Data augmentation was implemented for all models while training. 10 extra crops of images are fed into the model. They consist of two upper corner crops, one center crop, two lower corner crops. These crops are then flipped horizontally, and assigned with the same training label[11]. Data augmentation serves as random noise during the training process and help train more accurate model. This is during training time only.

4.2.2 Cyclical Learning Rate

Learning rate was the hardest hyper-parameter to tune, and every model is highly sensitive to poorly fine-tuned learning rate. Instead, a cyclical learning rate solves this problem by cyclically changing the learning rate within a rea-

sonable bound [4]. I used cyclical learning rate for all model training as well, as was implemented in the original GoogLeNet, AlexNet and ResNet[11]. The calculation of cyclical learning rates is presented in the equations below, where FLAGS.learning_ rate is the input learning rate by the user as the start learning rate at the beginning of each epoch.

I calculate the cyclic learning rate using the equations below:

```
num_cycles = 5
batch_size = 256
data_size = 100,000
num_epochs = 10
num_iter = num_epochs * data_size
                / batch_size
iter_size = num_iter / num_cycles
cyclic_lr = FLAGS.learning_rate / 2
* (cos(pi * ((curr_step - 1) % (iter_size)
 / (iter_size))) + 1)
```

Therefore, at the beginning of the epoch, curr_ step is 1, the cosine term evaluates to 1, and the learning rate is just input learning rate. When current step is close to half of one iteration size, the cosine term is equivalent to $cos(pi/2)$, which is 0, so the learning rate is input learning rate divided by 2. Lastly, when current step is close to one iteration size, the cosine term evaluates to close to -1 and the learning rate is close to 0. Therefore, the cyclic learning rate fluctuates between the higher bound learning rate, which is equivalent to the input learning rate, and the lower bound, which is close to 0, and it decreases during one epoch from initial learning rate . This is also shown in Figure 5 below.

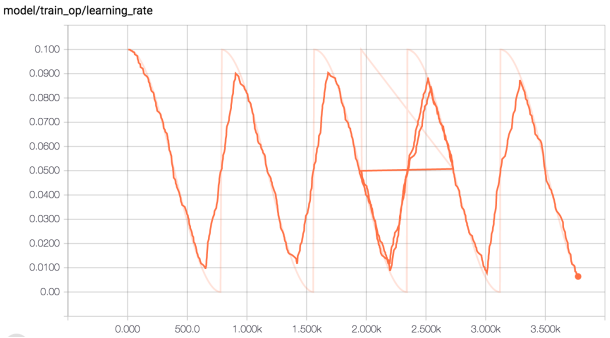

Figure 5. Cyclical learning rate from the first 5 epochs of AlexNet training. Tensorboard somehow only recorded the first five epochs in its events. However, the fluctuation is clear in its cyclical pattern from 0.1 to 0.01. The horizontal axis represent the number of iterations and the vertical axis is the learning rate. The number of iterations per epoch is around 390 (100,000 256).

Model	Training (%)	Top-1 Validation(%)
Baseline Model	34.40	33.04
Reduced AlexNet	46.20	43.65
GoogLetNet	53.43	49.40
ResNet-20	58.40	55.40

Table 3. The training and the top-1 validation accuracies of all models. ResNet-20 performed the best among all, with validation accuracy slightly better than 55% .

4.2.3 Floating Point Precision

In order to save space and increase speed, I started off used floating point precision of 32 bits instead of 64 bits, as how we implemented models in class. However, as pointed out by Gupta, et al [14] in their paper, that it is possible to train these networks using only 16-bit fixed-point numbers. Therefore I implemented the models with 16 bits. I see very little changes in terms of the speed or the accuracies change, however.

5. Results and Analysis

5.1. Results

The training accuracies and the top-1 validation accuracies are presented in the Table 3.

Looking at Table 3, we see that ResNet-20 performed the best during training. The fact that ResNet-20 is able to outperform all of the other models even with reduced architectures than the original are understandable, because it successfully eliminates the possibility of vanishing / exploding gradients and makes the optimization process easier. Therefore ResNet was significantly easier to train, and converged relatively faster than the other models. This model is later used on the test set as my best-performing model. I started off building and training AlexNet model, while its architecture is straightforward and was thus easy to build, It was the hardest model to train for me. First of all, It took the longest time to train due to its huge amount of parameters (15 million). Secondly, it could easily lead to overfitting or random noise. With parameters of this scale, even if initializations are not at the optimal value, the model was still able to train and have some accuracies due to noise or overfitting. At the start of training, I set the learning rate to be 1E-7, and 1E-5, which were later found to be too small for optimal performance. However, the training and the validation accuracies for AlexNet was still close to 10% after second epoch of training. I could only evaluate its initialization-based performance after epoch 3, when it would stay under 10% or decrease, which is 6 hours after. Whereas for GoogLeNet, due to its small size, it is highly sensitive to parameter initializations, the training accuracies would be only 2% for incorrectly initialized hyper-parameters. Therefore, though

easy to build, AlexNet is a relatively more difficult model to train due to its large amount of parameters that somehow makes it robust for poorly fine-tuned parameters at the beginning of the training, and making fine-tuning process long before locating the best performing parameters. My favorite model during training though, is GoogLeNet Inception, even though its structure was relatively complicated to implement, it was easy to debug by matching dimension of layers from one inception module to the next. Firstly, GoogLeNet is fast to train. It took nearly 4x less time to train than AlexNet and 3x less than ResNet-20. Its frequent use of small filters and 'network in network' structure also makes great intuitive sense during implementation. GoogLeNet had similar performance with ResNet up until the last three epochs, where it plateaued at slightly below 50%Ḋuring the entire process, training models that are built from scratch was long for all of the models (I only used one GPU on Google Cloud), yet the results turned out to be far from the state-of-art result that the original models generated on the Large Scale dataset, showing the limit of models built from scratch. Also, most of the models seemed to plateau at validation accuracy of around 50% for some reason, which caused great difficulty in improving classification accuracy. Fine-tuning the initializations had great effect on the model performance as well. Due to the time constraint, I stopped the training as soon as I started to notice non-convergence of the validation accuracy within 2 epochs.

I have also plotted the validation accuracies of all models over 10 epochs in Figure 6 below. According to the graph, models seem to converge at around 50% accuracies. AlexNet has a relatively steady increase in its validation accuracies over the training, my intuition of this is that due to its large amount of parameters, the result tends to have lower standard deviation and therefore tends to fluctuate less. It is shown in the graph as a smooth curve, however it plateaued at only 43.65% at the end of 10 epochs. ResNet-20 showed relatively large fluctuations early on during epoch 2 to epoch 4, where its average batch loss fluctuated quickly (not shown in graph) and its accuracy after third epoch almost stayed flat, which is rare, since accuracies usually increase rapidly during early epochs of training. However, it pulled back soon after the third epoch and increased rapidly after. It seems from the graph that ResNet-20 has not plateaued yet, due to its large jump in the last epoch. Yet it was already clear that ResNet outperformed the other models by a slight margin. GoogLeNet converged as one of the fastest models up to epoch 7, after which its validation accuracies decreased during epoch 8 and plateaued for epoch 9 and 10. Baseline model stayed in the 30% range starting epoch 7, and serves as a benchmark to the other models.

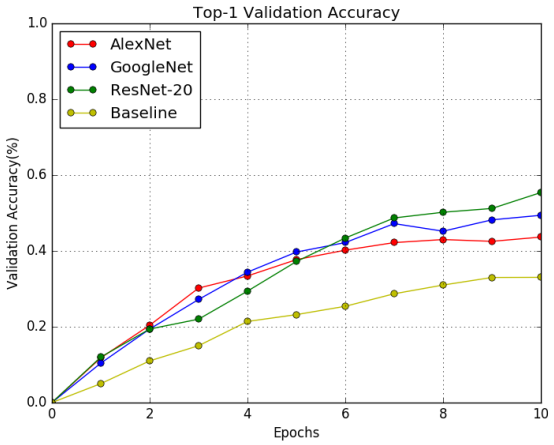

Figure 6. Top-1 Validation Accuracies of all models over 10 epochs. Due to the limit computing power, only 10 epochs of each model were run. This figure shows that all models seem to converge (plateau) at epoch 10, with ResNet-20 reaching the highest accuracy of 55.40% . All models perform better than the baseline model of 33.04 % , which is expected.

Model	Num of Parameters	Training Time
AlexNet	15 million	19 hrs
GoogLeNet	1.1 million	5 hrs
ResNet-20	11 million	14 hrs

Table 4. Number of parameters, and the time it took to train each model for 10 epochs on 1 GPU. AlexNet took the most time to train due to its large sizes, and GoogLetNet only took 5 hours to train due to its frequent use of small filters.

5.2. Number of Parameters

Number of parameters is definitely one of the most important factors to consider during training. Depth of layers, ConvNet structures, strides and paddings all contribute to different number of parameters, and the amount of parameters has great influence on gradient flow, overfitting level, computing time, and space needed. In Table 4 below, I have calculated the total number of parameters needed for each model and their corresponding total training time for one model.

5.3. Train Loss

One interesting pattern to observe is the effect of cyclical learning rate on training loss. Due to the cyclical nature of learning rate, the train loss decreases in a cyclical pattern as well, which eventually led it to converge after 8,9 epochs. According to Figure 7 attached below, the training loss, which is calculated as average classification loss over one batch, decreased from 5.50 at the start to 2.50 after 3.500k iterations. The light-colored pattern is the noise

on train loss. I believe the sudden increase of train loss at the beginning of the second epoch in the light-colored loss indicates an exploding gradient that was clipped right away. I set the gradient clipping to occur when the the norm is greater than 10.

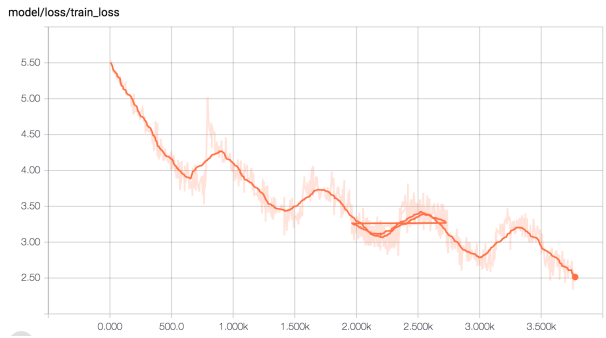

Figure 7. Train loss from the first 5 epochs of AlexNet training with noise(light color). Again Tensorboard somehow only recorded the first five epochs in its events. However, the fluctuation is clear in its cyclically decreasing pattern for each epoch. The horizontal axis represent the number of iterations and the vertical axis is the train loss. The number of iterations per epoch is around 390 (100,000 256). At the beginning of each epoch, train loss increases due to increase in cyclical learning rate, and decreases over the epoch

6. Conclusion and Future Work

In this project, I implemented and trained from scratch three state-of-art model structures, AlexNet, GoogLeNet, and ResNet-20, on the TinyImageNet dataset. After fine-tuning and evaluating the performance on all three models, I used my best performing model, which is ResNet-20 on the test set, and achieved test accuracy of 45% (error rate of 55%). During this process, I found out that the models tended to plateau at around or below 50% validation accuracy. It turns out that training state-of-art model structures with slight variations from scratch on reduced dimension of image datasets do not have as good performances as the original models on the Large Scale ImageNet Challenge. My intuitions of this phenomenon is the following.

First, downsizing of the image datasets could lead to ambiguity problems in image details and might affect model accuracy. If we take a picture at the actual downsized samples, it is very hard not to make mistakes even with human eyes. Also, according to a fellow Stanford student Jason Ting[16], the original pictures from the ImageNet data set are 482x418 pixel with an average object scale of 17.0%Since the Tiny ImageNet data set pictures are 64x64 pixels, which means 13.3% pixels are removed from the original images to make the pictures a square, and then these pictures are shrunk by a factor of 6.5. This transformation alongside with how the labels are associated with the images leads to potential problems for training the model, where some example images can be seen in Figure 8[16].

Figure 8. Examples of images in the data set that are difficult to classify even with human eyes. In (a) and (e), the objects are very blurry. You can barely see the sunglasses in (f) and (h). The labels in (c) and (g) look more of background rather than the main objects. Same problem with bow tie in (b) and banana in (d)

Secondly, limited number of computing power and time also limited how far I could go with each model training. During my training process, I was only able to implement 10 epochs for each model before it plateaued. The original models, however, each takes at least two to three weeks to train, and were trained with 90 epochs with AlexNet, baseline ResNet for 500 epochs, and 250 epochs on GoogLeNet.

For the future work, trying to eliminate unnecessary parameters in the models using techniques described in SqueezeNet[8] is definitely what I would consider doing before training. Improving on the hardware (more GPUs) to allow more epochs of training is also considerable for model improvements. If possible, I would look closely into the Tiny ImageNet Dataset to see which images have lost details and may cause ambiguity in training process, and therefore is not worth including in the samples. I will also look into my best model, ResNet-20, to see if residual connections significantly improve fine-tuning and training efficiency in general.

7. Reference

References

[1] Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E. *ImageNet Classification with Deep Convolutional Neural Networks. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf*

[2] *http://www.kdnuggets.com/2016/09/deep-learning-reading-group-squeezenet.html*

[3] *http://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/*

[4] Leslie N. Smith *Cyclical Learning Rates for Training Neural Networks*.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich *Going Deeper with Convolutions*.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun *Deep Residual Learning for Image Recognition*

[7] Karen Simonyan, Andrew Zisserman *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION*

[8] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡ 0.5MB model size*

[9] Gao Huang, Zhuang Liu, Kilian Q. Weinberger, Laurens van der Maaten *Densely Connected Convolutional Networks*

[10] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei.(* = equal contribution) *ImageNet Large Scale Visual Recognition*

[11] Tyler Romero *https://github.com/fcipollone/TinyImageNet*

[12] CS231N Convolutional Neural Network, Stanford University, spring 2017 *http://cs231n.github.io/convolutional-networks/*

[13] Andrew Zhai *Going Deeper on the Tiny Imagenet Challenge*

[14] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan *Deep Learning with Limited Numerical Precision*

[15] Jason Ting *Using Convolutional Neural Network for the Tiny ImageNet Challenge*

[16] Hansohl Kim *Residual Networks for Tiny ImageNet*

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*

[18] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, Trevor Darrell *DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition*