Exploration of the Effect of Residual Connection on top of SqueezeNet
A Combination study of Inception Model and Bypass Layers

William Bokui Shen
Stanford University
bshen88@stanford.edu

Song Han*
Course TA for CS231N
Stanford University
songhan@stanford.edu

Abstract

Two of the most popular model as of now is the Inception module of GoogLeNet and MSRA's Deep Residual Network. Google's recent research on combining residual connection with Inception model created deep neural network models (Inception-Resnet-v1 and Inception-Resnet-v2) that have slightly better accuracies than their state-of-the-art inception model (Inception v4) networks, but have a significantly faster convergence speed. In my research, we will try to explore the effect of residual connections on SqueezeNet[1], which is a DNN designed that achieves accuracy level of AlexNet with 50x fewer parameters using variation of inception model. Although models of large neural networks like ResNet have remarkable performance on accuracy, their size consumes significant storage, memory bandwidth and computational resource. For use in embedded mobile application , it is crucial to come up with model that find a good balance between performance and memory / computational resource usage. Keeping the idea of performance and memory balance in mind, our goal is to increase the performance of the DNN without significantly increasing the amount of parameters.

1. Introduction

We are taking on the Tiny ImageNet Challenge for the final project of CS231N (Convolutional Neural Networks for Visual Recognition). The Tiny ImageNet is a simplified version of The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) that caters to a student's computing and time resource. It has a smaller number of classes (200 classes instead of 1000 of ImageNet challenge). It has in total 100,000 training images (500 training images for each class), 10,000 validation images (50 for each class), and 10,000 test images (50 for each class). Each image has a resolution of 64 × 64 pixels. Although Tiny ImageNet has fewer classes and a smaller picture size, one thing worth noticing is the effect of down-sampling. Since in Tiny ImageNet challenge, the picture is scaled down from 224x224 to 64x64, information in a picture is lost during[1] the process. The effect of down-sampling is illustrated by Figure 1, which is created by last year's student Lucas Hansen.

Two kinds of Deep Convolutional Neural Network introduced in class are GoogLetNet (ILSVRC 2014 winner) and ResNet (ILSVRC 2015 winner). My plan is to combine the creative structural ideas behind the two and make a better more efficient network. In other word, we will try to investigate the effect residual connections have on inception model. SqueezeNet is a state-of-the-art inception model based Deep Neural Network that find a good balance between classification accuracy and memory usage. Taking our computing time and resource budget into account, SqueezeNet is also the most reasonable model for us to work on. The main goal of my research will be how to use residual connections on SqueezeNet to enhance its performance while keeping the parameter count low.

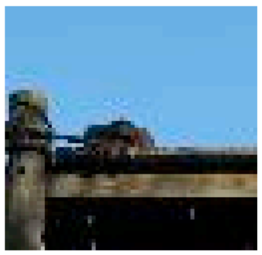
For some background of SqueezeNet performance, it has a top-1 classification accuracy of 57.5% on the ImageNet challenge. As mentioned at the start of introduction, Tiny ImageNet has its pitfalls and difficulties that arise from down-sampling. Therefore, considering our limited access to GPU and the flaws of Tiny ImageNet, our goal will be to replicate the accuracy (57.5%) of original SqueezeNet on ImageNet for my accuracy on Tiny ImageNet.

2. Technical Approach

2.1 Caffe / Data Preprocessing

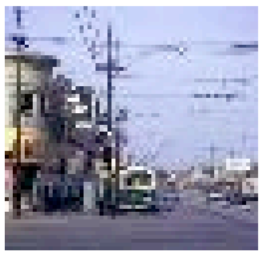
In my project, I am using the deep learning package Caffe. Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. We used the `convert_imagest` python routine in Caffe to convert the given jpg format into LMDB format. In this format, the training data size is 1.6GB.

*Song Han helped me come up with the project idea, and offered me help and support along the way. For that, I am deeply thankful.


(a) Orangutang

(b) Broom

(c) Popsicle


(d) Trolleybus

(e) iPod

(f) Lighthouse

(g) Bikini

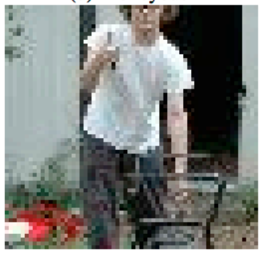

(h) Lawnmower

Figure 1: Some examples of difficult images. Note the scaling artifacts prominent in (a) and (d), the loss of texture in (c) and (g), the loss of crucial information due to cropping in (b), (f), and (h), and the difficulty of locating small objects in (c).

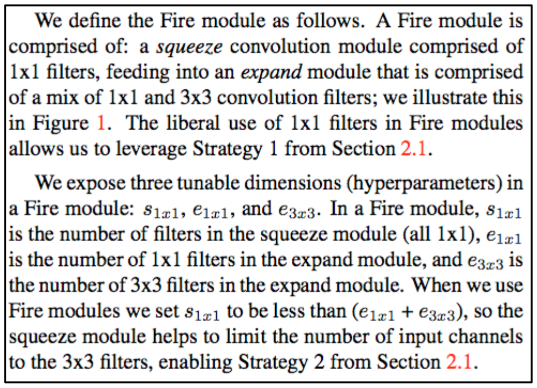
2.2 Optimization / Solver Model

Training is done by the forward pass that computes the softmax loss and the backward pass that back-propagates with regard to minimizing softmax loss. The update of the parameters is achieved through stochastic gradient descent (SGD) and momentum update. Stochastic gradient descent updates the parameters by decreasing the parameter with a constant factor of its gradient regarding to the final loss: W ← W − η∇W, where η is the learning rate. Momentum update is used to simulate a particle rolling in a direction while gaining momentum. Two kinds of such optimization approaches (Adam and Nestrov) are tested during the project. Nestrov with polynomial learning rate update policy turns out to have the best performance across all models.

2.3 Network Design

2.3.1 Original Fire Model

The original fire model of SqueezeNet is defined as follow, according to the original paper:


We define the Fire module as follows. A Fire module is comprised of: a *squeeze* convolution module comprised of 1x1 filters, feeding into an *expand* module that is comprised of a mix of 1x1 and 3x3 convolution filters; we illustrate this in Figure 1. The liberal use of 1x1 filters in Fire modules allows us to leverage Strategy 1 from Section 2.1.

We expose three tunable dimensions (hyperparameters) in a Fire module: s_{1x1}, e_{1x1}, and e_{3x3}. In a Fire module, s_{1x1} is the number of filters in the squeeze module (all 1x1), e_{1x1} is the number of 1x1 filters in the expand module, and e_{3x3} is the number of 3x3 filters in the expand module. When we use Fire modules we set s_{1x1} to be less than ($e_{1x1} + e_{3x3}$), so the squeeze module helps to limit the number of input channels to the 3x3 filters, enabling Strategy 2 from Section 2.1.

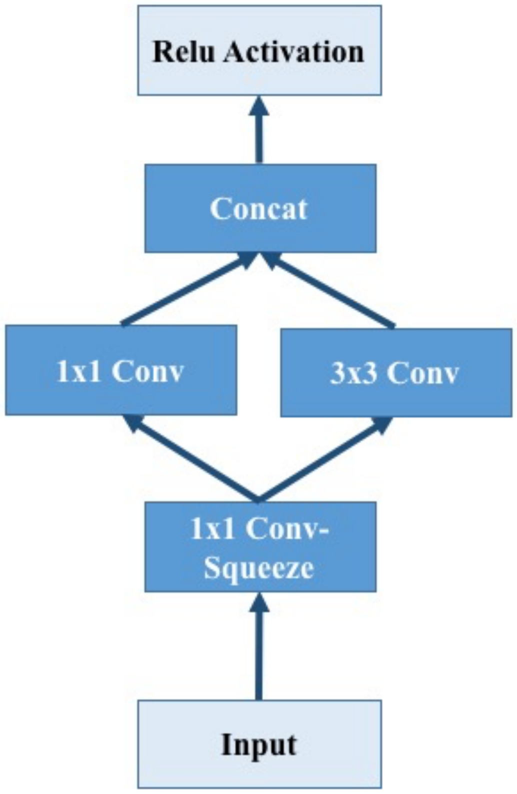


Figure 2: the design of original fire model. It's a form of inception model. The most important characteristic is the idea of squeezing of the 1x1 conv squeeze layer which significantly reduce the count for parameters.

2.3.2 New Models

In order to explore the effect of residual connection on SqueezeNet, we need to change the fire model by adding residual connection on top of the original model. Two types of new model is explored, we'll call them TypeA-Fire and TypeB-Fire.

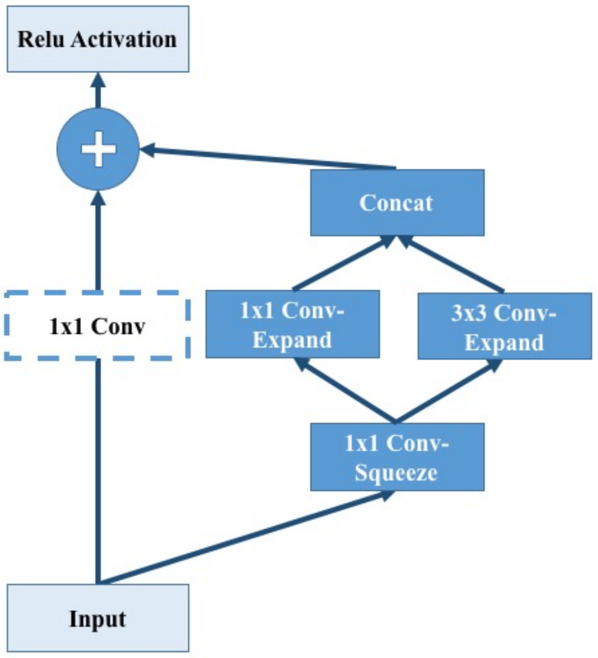


Figure 3: The schema for TypeA-Fire module of the Bypassed SqueezeNet.

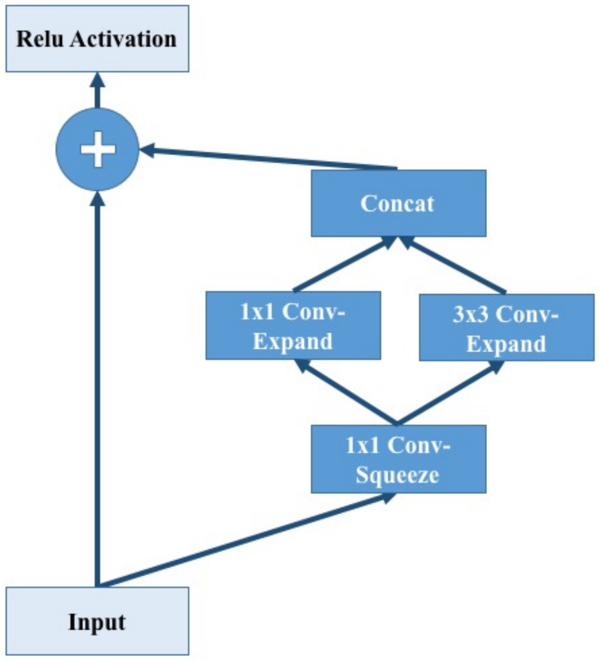


Figure 4: The schema for TypeB-Fire model of the Bypassed SqueezeNet.

Type-A Fire adds an 1x1 convolution layer on top of the bypass connection. The main advantage of Type-A Fire comparing to TypeB-Fire is its ability to ignore the disagreement between input channel number and output channel number.

2.3.3 Network structure

Three kind of new network structure has been explore in the project. They are: Partially Bypassed 8 Layer SqueezeNet, Fully Bypassed 8 Layer SqueezeNet, Fully Bypassed 12 Layer SqueezeNet. The structure of Fully Bypassed 8 Layer SqueezeNet is shown below. The full detail of this structure is shown in table 1.

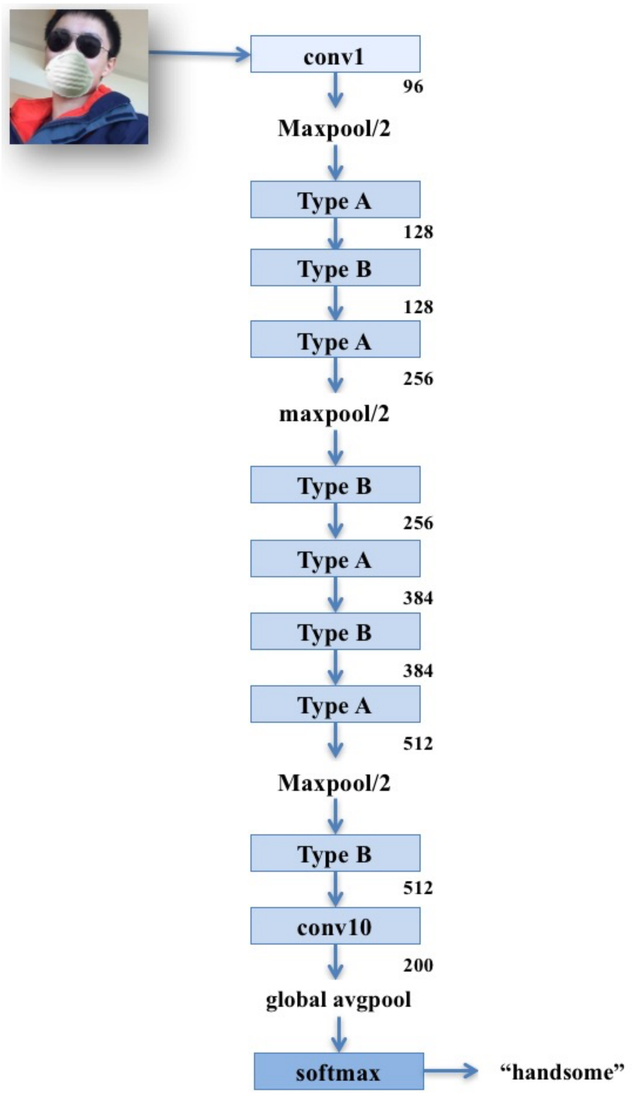


Figure 5: The structure for the Fully Bypassed 8 layer SqueezeNet. For the partially bypassed, every TypeA is being replaced by the orginal fire model, thus making all the residual connection direct identity mapping bypass. For the fully bypassed 12 layer, two

more sets of “TypeA – TypeB ” layer is added, with a maxpool layer before the last TypeA.

Table 1: Detailed structure of Fully Bypassed 8 layer SqueezeNet . In column 4 and 5, s denotes squeeze conv; e denotes expand conv.

Layer	Output Size	s	e	Param#	Param# (no-bypass)
input	64x64x3			-	-
conv1	64x64x96			2592	2592
maxpool	32x32x96			-	-
TypeA	32x32x128	16	64	24064	11776
TypeB	32x32x128	16	64	12288	12288
TypeA	32x32x256	32	128	77824	45056
maxpool	16x16x256			-	-
TypeB	16x16x256	32	128	49152	49152
TypeA	16x16x384	48	192	202752	104448
TypeB	16x16x384	48	192	110592	110592
TypeA	16x16x512	64	256	385024	188416
maxpool	8x8x512			-	-
TypeB	8x8x512	64	256	196608	196608
conv10	8x8x200			102400	102400
avgpool	1x1x200			-	-
Total				1,163,296	823,328

Here what we mean by 8 layer or 12 layer is not the total conv layer number, but rather the number of TypeA/TypeB model incorporated in the network. The depth of the 8 layer Fully Bypassed SqueezeNet is 18; the 12 layer Fully Bypassed SqueezeNet has depth 26.

Another alteration to the original SqueezeNet model is the introduction of batch normalization layers. In my implementation, every convolution layer is followed by a batch normalization layer, and this contributes a lot to the final performance.

3. Analysis

3.1 Memory / Parameter Count

As computer vision, or more specifically image classification and image recognition, being used more and more in mobile end, mobile-first companies have become very sensitive to the size of the binary files. Saving size on a feature will bring great convenience when bringing deep neural network to the mobile end.

Having this mindset, it’s important to take into account the memory needed and the total parameter count when comparing different models.

Table 2 is a breakdown on the memory / parameter count comparison between our modified version of SqueezeNet and the current mode with best performance in the field, ResNet. The size of caffemodels are all size before any compression. The size of ResNet’s caffemodel is the size of the 152 layer caffemodel provided in the github repository of the original ResNet paper. In the original paper about SqueezeNet, using deep compression can help to further reduce the size of the model by a factor of more than 9 without any drop in the accuracy performance of the model. In this way, the memory needed for SqueezeNet will be less than 0.8 MB even for the deeper 12 layer Fully Bypassed SqueezeNet; this level of memory needed will suit the use in embedded system very well.

As we can see, ResNet is not very memory friendly by taking up size more than 200 megabytes, but our model of SqueezeNet is significantly more memory friendly. It uses around 40 times less parameters than ResNet ($1.6 * 10^6$ vs. $6 * 10^7$). And the caffemodel size differs with a similar ratio, the caffemodel for ResNet takes 230.26 megabytes while the caffemodel for the 12 layer fully-bypassed SqueezeNet takes 6.4 megabyte. The difference is huge enough. Considering the small size Fully-Bypassed SqueezeNet, it’s reasonable for us to sacrifice certain degree of accuracy. And as the following section will show, the performance of Fully Bypassed SqueezeNet over Tiny ImageNet suggests that adding bypass layer/residual connection to SqueezeNet, the performance of the model is significantly increased.

Table 2: A detailed comparison among the memory efficiency of Original SqueezeNet, Fully Bypassed 8 layer SqueezeNet, FullyBypassed 12 layer SqueezeNet, and the 152 layer Resnet.

	CaffeModel Size	Param#
Original 8 layer	3.2 MB	0.82M
Fully Bypass 8 layer	4.8 MB	1.16 M
Fully Bypass 12 layer	6.4 MB	1.6 M
ResNet	230.26 MB	60 M

3.2 Accuracy Results

The accuracy is measured in both top1 and top5 accuracy. Top-k accuracy means the model produce the k most likely label it classifies, and if the correct label is in these k prediction, we say that the model correctly classifies the image. The state-of-the-art top1 accuracy for Deep Neural Network is around 77% accuracy, while the state-of-the-art top 5 accuracy for any Deep Neural Network is around 94%.

For our models, the accuracy is measured on the training set, the validation set and the test set on the evaluation server provided by the CS231N course stuff. The test set accuracy is the most unbiased since we don't have access to the correct label for the test set, and our result can only be measured every 2 hour by uploading our own classification result to the server.

All the training started from scratch and was run on a GPU instance on AWS (Amazon Web Service). Each model is trained for 24-36 hours on the instance due to my limited budget. This results in the following training epochs for different models: 50+ epochs for the original SqueezeNet, 50+ epochs for the partially bypassed 8-layer SqueezeNet, 35+ epochs for the fully bypassed 8-layer SqueezeNet, ~30 epochs for the fully bypassed 12-layer SqueezeNet. The difference in epoch number is due to the limitation of GPU which can take only a smaller batch size when the model complexity is increased. For the original SqueezeNet, the batch size can be increased up to 40, while the 12 layer fully bypassed SqueezeNet can only take a batch size of 10 for each training iteration.

During the training phase, if the accuracy doesn't go up by at least 1 percent for 2 epochs, the learning rate for the Nestrov solver (experimented to out perform other solver optimization like Adam) will be decrease by a factor of 5; the next drop in learning rate will be by a factor of 2; the next will be 5 again, and so on.

Table 3: The accuracy performance of different SqueezeNet models. The top 5 accuracy is only measured for the validation set for the two best model. The test set performance is only measure for the two best models as well.

	Training Set Top 1	Validation Set Top 1	Test Set Top 1	Param#
Original 8 layer	0.70	0.41	-	0.8 M
Partially Bypass 8 layer	0.71	0.44	-	0.8 M
Fully Bypass 8 layer	0.79	0.54 top 5:0.77	0.50	1.1 M
Fully Bypass 12 layer	0.90	0.58 top 5:0.83	0.55	1.6 M

The accuracy result is listed in table 3. Here are a few interesting findings.

First, partially bypassing SqueezeNet doesn't make much difference. As we can see, the top 1 accuracy on training set have less than 1% difference between original SqueezeNet and partially bypassed SqueezeNet. Although the validation set accuracy has a 3% difference, it's mainly because of the small size of validation data cause a level of randomness. We should see the two model as having almost identical performance. Thus, partially bypassing SqueezeNet is not helpful.

Second, fully bypassing SqueezeNet result in a much better performance and a faster convergence rate with respect to epochs. The difference is significant: in the same training period of time, the Fully-Bypassed 8 layer SqueezeNet reaches a top 1 validation accuracy of 54.3 percent. It out performs the original 8 layer SqueezeNet by almost 10 percent in validation set top 1 accuracy and 8 percent in training set top 1 accuracy. Considering the Fully-Bypassed SqueezeNet receive less epochs due to its higher complexity. It's safe for us to say that it converges faster than the original with regard to epoch. The test set performance of the fully bypassed 8 layer is a **49.3** percent error rate. This (validation accuracy differs with test set accuracy by 3-4 percent) is an interesting phenomenon that is observed by other participants in the Tiny ImageNet challenge. The likely reasons behind this is the characteristics of different classes not reflected in the 200 training images for each class, resulting in the model's unfamiliarity with the test set images.

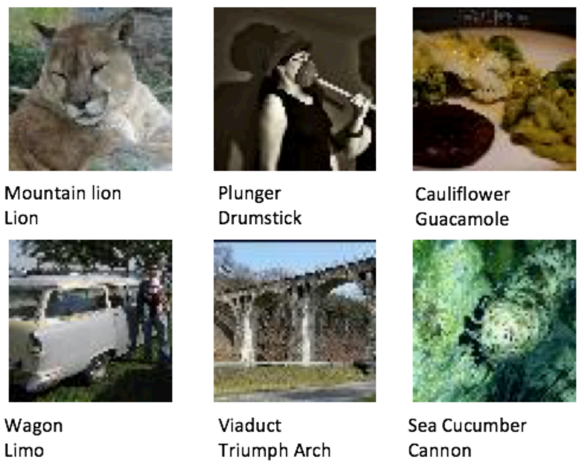
Third, going deeper does increase the performance, but inevitably facing the problem of over fitting the Tiny ImageNet dataset. As shown in the table, the Fully Bypassed 12 layer SqueezeNet has the best overall performance by reaching validation accuracy of 58% and test set error rate of **45.2%** (54.8% top 1 accuracy) which ranks us 5th place on the course leader board (**1st** in groups not using ResNet). It has a four percent increase in validation set top 1 accuracy and a five percent increase in test set top 1 accuracy than the Fuly Bypassed 8 layer SqueezeNet. The model does out perform, but in the process, it faced the problem of significant over fitting. Our first try of the deeper structure result in a brutal over fit, reaching a 97 percent accuracy on training set top 1 accuracy (with validation accuracy of 56 percent and test accuracy of 51 percent). Hieu Pham, the course TA, gave me the idea of adding dropout layer. The retain ratio of 0.5, 0.6 and 0.7 has been experimented, and it turns out that 0.5 works the best. With the help of dropout layer, the problem of over fitting is mitigated but still present. The training set accuracy dropped by 7 percent to 90%, but interestingly, the performance of top 1 accuracy of validation set and test set both increased, to 58% and 54.8% respectively.

To conclude, we do see a significant increase in model performance if we introduce residual connection. Our hypothesis that residual connection will help an inception model like SqueezeNet is validated.

3.3 Error Analysis

Among best performance categories for our model are school bus, banana, orangutan, triumph arch and go-kart. Here, an example set of incorrect classification is presented.

Figure 6: An example set of erroneously classified images. The first line is the correct label, the second line is the label the model produced.



For the first image, it's understandable why the model made the mistake, even myself can't explicitly say the difference between a mountain lion and a female lion. The second image is shape-wise confusion: the plunger handle has a similar shape to a drummer holding a drumstick. The third image is very likely to be color confusion: the image is mainly green which might remind the model of guacamole. The fourth is taken from an interesting angle which made the car seems longer, which might cause the model to think that it's a limo. In the fifth image, the model made the mistake of looking at part of the image rather than the whole picture, viaduct is like a sequence of triumph arch aligned in sequence. The last image symbolized the totally random and uninterpretable mistakes that model makes.

By seeing the above example mistakes. We can see that the model does "understand" a lot of the image even when it's making mistakes. It can sniff out features in the picture and make a reasonable deduction based on that.

4. Conclusion

First, residual connection does benefit an inception model like SqueezeNet. On the aspect of convergence rate, it agrees with the Google's inception v4 paper that adding residual connection or bypass layer significantly increases the convergence speed of the model. Result in a shorter training time with regard to epochs. On the aspect of actual model performance, our result disagrees with Google's paper. We can see an significant increase (10 percent) in performance when we fully bypass SqueezeNet. My interpretation why adding residual connection to SqueezeNet yields such an improvement is as follow: Residual connection deals with the information downsampling during the squeezing process. SqueezeNet will lose a significant amount of information during the squeeze convolution layer which reduce the channel number by around 80% (the idea to save computing resource will obviously take some sacrifice), the model is then trying to recover this loss of information by doing the expand operation but it's impossible to do so since a huge amount of information is not recoverable. Adding residual connection can address this information leakage very well. Whether it is TypeA or TypeB module, the input information is either expanded with the help of 1x1 convolution (as in TypeA) or directly preserved (as in Type B) to the output; therefore, the information will not be wasted when flowing through the neural network.

Second, going deeper with residual connection on top of SqueezeNet can create a better performance but should proceed with caution. The extra performance is substantiated by the result shown in table 3. But there are two aspects that we should keep in mind. First, the problem of over fitting. Going deeper can cause the problem of severe over fitting, especially in the case of Tiny ImageNet, which has a smaller input size and class number. The problem might be mitigated in the case of ImageNet, but if we go even deeper, the problem might surface. What we can do to address the problem is by using dropout layer. The second thing worth noticing is that by going deeper, we quickly pack up more memory and parameter count, which is going backward from our goal of saving space.

Third, I would like to argue that the performance improvement is worth the memory and parameter count increase. As demonstrated above, changing the model to 12 layer fully bypassed increase the model size by 33% (4.8 mB to 6.4 mB). With the deep compression technology in the original SqueezeNet paper, the final compressed model size will be around 680 kB without losing performance accuracy. The size is still way below 1 mB and the extra 170 kB is worth it when we can get an additional 10% in accuracy.

5. Future work

First, getting more computing resource. The training time is limited on every model, as a student, financial budget is a concern. With sufficient training time, I expect the performance of the model to go even higher. Considering that the typical training epoch for a deep neural network competing for ImageNet challenge goes more than 90 epochs, my guess is that if we reaches somewhere around 90 epochs, the performance will go up by around 5 percent.

Second, I would like to experiment my model on the ImageNet challenge. It would be interesting to see how the model performs when trying to process a much larger input and a larger set of categories.

Third, the effect of dropout layer will be examined more thoroughly. As demonstrated above, one way to improve performance is by going deeper, but the effect of dropout layer becomes more important along the way. Where to add dropout layer, how to set dropout ratio will be two problems worth investigating.

Fourth, the effect of squeeze ratio is worth exploring. Squeeze ratio is negatively correlated to the final performance since it's a process of sacrificing information for the sake of memory efficiency. With the help of residual connection, a larger portion of information can flow through the neural network. My hypothesis is that we can increase the squeeze ratio and keep the original performance if residual connection is implemented.

6. Special Thanks

First, I thank Song Han for introducing me to SqueezeNet, and for helping me with the project along the way.

Second, I thank Andreg Karpathy, Justin Johnson and Professor Fei Fei Li for bringing such a wonderful class and introducing me to the wonderland of Deep Convolution Neural Network.

Third, I thank other couse TA (Namrata Anand, Hieu Pham, Serena Yeung) for helping me with the technicality like caffe, AWS etc.

Finally, I thank Red Bull, Starbucks and tea. I would have fallen asleep without you guys.

References

[1] *Going Deeper with Convolutions*, Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. arXiv:1409.4842

[2] *Deep Residual Learning for Image Recognition,* Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. arXiv:1512.03385

[3] *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size*, Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, Kurt Keutzer. arXiv:1602.07360

[4] *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*, Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke arXiv:1602.07261

[5] *Tiny ImageNet Challenge Submission*, Lucas Hansen, http://cs231n.stanford.edu/reports/lucash_final.pdf