ResNet on Tiny ImageNet

Lei Sun
Stanford University
450 Serra Mall, Stanford, CA
sunlei@stanford.edu

Abstract

Deep neural networks have shown their high performance on image classification tasks but meanwhile more training difficulties. Due to its complexity and vanishing gradient, it usually takes a long time and a lot of computational resources to train deeper neural networks. Deep Residual networks (ResNets), however, can make the training process easier and faster. And at the same time, it achieves better accuracy compared to their equivalent neural networks. Deep Residual Networks have been proven to be a very successful model on image classification.

In this project, we will train our own ResNets for the Tiny ImageNet Visual Recognition Challenge - an image classification task based on a subset of the ImageNet. We first implemented a vanilla version of ResNets with 34 layers. Then its performance and accuracy is shown, followed by some detailed debugging and analysis on how to improve it. After that, an improved model with stochastic depth and data augmentation is developed as baseline. Its performance is compared with the vanilla model, as well as many other variants of Residual Networks. At last, a final model with heavy image augmentations is developed aiming the Leaderboard of the Challenge.

1. Introduction

Image classification is a fundamental problem in computer version and machine learning. It has been attracting a lot of researches on it. In recent years, there are many successful breakthroughs in the field of image classification. Around 2011, a good ILSVRC classification error rate was 25%. In 2012, AlexNet[8] was invented. It is the first convolutional neural network (CNNs) based model with 8 layers. AlexNet achieved 16% error rate on the ImageNet challenge. In the next couple of years, with more and more layers in neural networks, VGG19[10] with 19 layers and GoogleNet[11] with 22 layers reduced the error rates to a few percent. Although CNNs have made some breakthrough on the accuracy, they are hard to train for two reasons. First, the so called vanishing gradient problem - the effect of multiplying n of those small numbers from activation function to compute gradients in an n-layer network, meaning that the gradient (error signal) decreases exponentially with n, thus the front layers train very slowly. Second, CNNs usually have even more parameters in their models, introducing more complexity, so takes longer to train.

And finally in 2015 comes the ResNet[4]. The main difference in ResNets is that they have shortcut connections parallel to their normal convolutional layers. These shortcut are always alive and gradients can easily propagate through them, resulting in faster training. ResNet with 152 layers achieves the best results of 3% error rate, which is even better than human judges[9]. To study this state-of-art model in image classification, in this project, we will implement and experiment various types of ResNet, together with other widely-used techniques. And then we will discuss how to prevent overfitting with small dataset.

1.1. Tiny ImageNet

The ImageNet[1] challenge (ILSVRC) is one of the most famous benchmarks for image classification. The data set has a total of 1,200,000 labeled images from 1000 different categories in the training set and 150,000 labeled images in the validation and test set.

For this experiment, however, we will use the Tiny ImageNet - a subset of ILSVRC. It follows the same principle, though on a much smaller scale. This Tiny ImageNet only contains 200 different categories. Each category has 500 training images (100,000 in total), 50 validation images (10,000 in total), and 50 test images (10,000 in total). In additional, the images are re-sized to 64x64 pixels (256x256 pixels in standard ImageNet).

If the models described in papers perform well on the original ImageNet, it is supposed to have similar performance on Tiny ImageNet as well. The test accuracy will be evaluated on a test server when the training and tuning processes are all done.

Figure 1. Sample images from Tiny ImageNet

2. Related Work

Researchers have hypothesized and mathematically proven that deeper neural networks have more representational power[12]. Deeper nets gain this power from hierarchically composing shallower feature representations into deeper representations. For instance, in face recognition, pixels make edges and edges make corners. Corners define facial features such as eyes, noses, mouths and chins. Facial features compose to define faces[2].

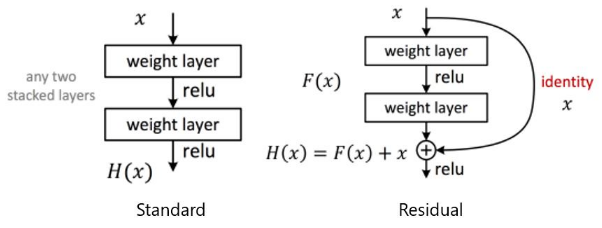


Figure 2. Residual learning: a building block

Unfortunately, deep CNNs are hard to train due to vanishing gradients in the long forward feed and backward propagate process. A residual neural network, on the other hand, has shortcut connections parallel to the normal convolutional layers. Mathematically, A ResNet layer approximately calculates $y = f(x) + id(x) = f(x) + x$. Those shortcuts act like highways and the gradients can easily flow back, resulting in faster training and much more layers. The winner model that Microsoft used in ImageNet 2015 has 152 layers, nearly 8 times deeper than best CNN.

After the the success of ResNet, more related work has been done. The original authors of ResNet proposed an improved version of their models by adding more direct identity connections to the network[5]. Another interesting idea is to use stochastic depth in training deep networks[6], which is a modified version of the classic dropout. But it drops the whole layers out randomly in training instead of some nodes within the layers. During training, the depth of the network can be much smaller than normal. This results in fewer calculations and more efficient training throughput. Those promising techniques will also be experimented and analyzed in my project.

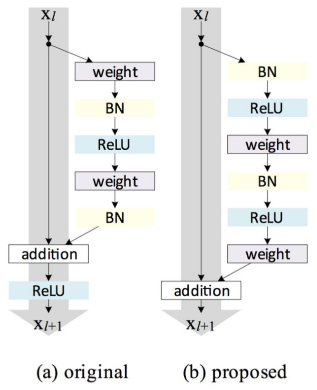


Figure 3. Identity Mappings in ResNet

3. Technical Approaches

One feasible approach is to fine-tune some pre-trained Resnet models. Some variants such as ResNet-50, ResNet-101, and ResNet-152 are released for Caffe[3]. Even if this approach is adopted, those models cannot be used directly on Tiny ImageNet - there are only 200 categories in Tiny ImageNet. Some re-train process needs to be applied on them. The standard practice would be the two phase fine-tuning method. First, add a new FC layer with output layer of size 200, train this layer exclusively for a couple of epochs. And then, re-train the full network for another couple of epochs.

One problem with fine-tuning is that we have limited model configurations to test on. But this project is more interested in the comparison among different variants, as well as how to achieve better performance by adding features and techniques step by step on base model. As such here we investigate in training models from scratch.

3.1. TensorFlow and Performance Tuning

Because of its wide application for both research and production on deep learning, TensorFlow will be used throughout this project and all the results are thus obtained from it.

All the programs run a cloud machine armed with K80 GPU on Microsoft Windows Azure. 100,000 training images are divided into 1000 mini-batches, with 100 images in each. In order to speed up the training process, a series

	Before	After
GPU (Percentage)	82%	99%
Speed (sec/iteration)	2.45	2.74

Table 1. Tensorflow perforance tuning result

of performance features are added to the model. They are able to utilize GPU resources more efficiently, thus reduces the run time from 2.74 seconds per iteration to 2.45, more than 10% less than before, as shown in Table 1. These performance improvement include but not limit to:

- Pre-process images on CPU
  Placing preprocessing operations on CPU can avoid the data bouncing back and forth between CPU and GPU.

- Support both image data format
  NCHW images performs better when using the NVIDIA cuDNN library. So my implementation supports both formats such that it can be migrated easily from CPU to GPU.

- Use fused batch normalization.
  The non-fused batch norm does computations using several individual operations. Fused batch norm combines the individual operations into a single kernel, which runs faster.

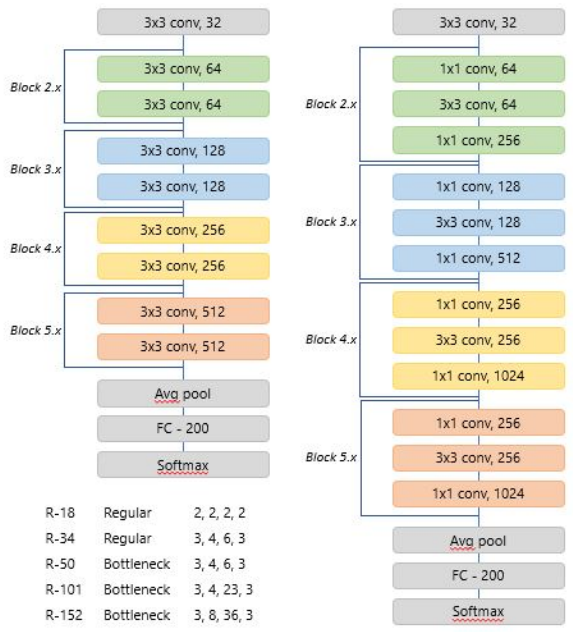
4. Vanilla Model

The original authors introduced several structures for ResNets[4]. They are different in terms of the number of layers, the number of convolutional layers in each residual block, and the filter sizes in each layer, as shown in Figure 4

A vanilla Resnet-34 is first implemented and tested, whose results are shown in Figure 5. This model shows the learning power of ResNet, without too much babysitting on the learning process, we achieved 48% error rate on validation set, and 49% on test data, with only 15000 iterations.

But on the other hand, the accuracy is still 20% higher than what is shown in the original paper. By looking at the curves, we noticed that, the train error quickly approached to zero after the learning rate annealing at iteration 10000, but the validation loss and error did not change at all, which shows a typical overfitting!

Although undesired, this overfitting is very expected, there are only 500 training images in each category. For every 10000 iterations, we feed exactly the same set of training images into the network and our ResNet sees them and learns from them over and over again. On the other hand, to make this challenge a fair game, we are not allowed to get more training data. Therefore by far, the biggest challenge in this project becomes how to improve our model and make better use of the training data to avoid overfitting.



R-18	Regular	2, 2, 2, 2
R-34	Regular	3, 4, 6, 3
R-50	Bottleneck	3, 4, 6, 3
R-101	Bottleneck	3, 4, 23, 3
R-152	Bottleneck	3, 8, 36, 3

Figure 4. Residual network structures

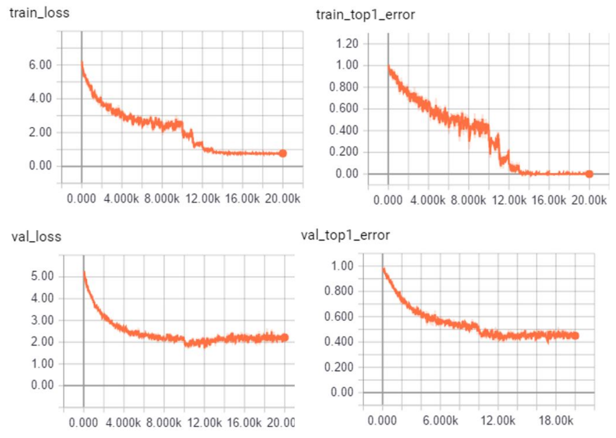


Figure 5. Result of vanilla model

5. Improved Model and Result

5.1. Stochastic Depth

Stochastic Depth is the first feature introduced in this project to fight against overfitting. It has similar idea with the classic drop-out technique, acting as a regularizer in the networks. The classic Dropout randomly drops hidden nodes or connections by multiplying activation output with an independent Bernoulli random variable. So it reduces the effect known as "co-adaptation" of hidden nodes collaborating in groups instead of independently producing useful features[6]. Similarly, stochastic depth can be interpreted as training as ensemble of networks. It randomly bypasses

	No Drop-out	With Drop-out
Avg Blocks	16	12.3
Avg Depths	34	26.6
Avg Runtime	2.45	2.13

Table 2. Performance improvement with stochastic depth

one or more ResNet blocks, thus makes the network shorter instead of thinner, as shown in Figure 6.

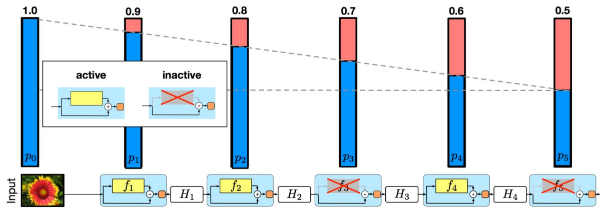


Figure 6. Structure of stochastic depth

Mathematically, the output of each ResBlock becomes:

$$H_l = ReLU(b_l f_l(H_{l-1}) + id(H_{l-1}))$$

where $b_l \in \{0, 1\}$, representing a Bernoulli random variable, which indicates if the l^{th} ResBlock is active or inactive. And the survival probability of ResBlock l is $p_l = Pr(b_l = 1)$. Therefore the ResNet fall back to the original ResNet structure when $b_l = 1$, and it becomes identity function when $b_l = 0$. As suggested in the paper, I also set the survival probabilities as

$$p_l = 1 - \frac{l}{L}(1 - p_L)$$

One more thing to note is the output is re-calibrated a little bit on testing time

$$H_l^{Test} = ReLU(p_l f_l(H_{l-1}^{Test}; W_l) + id(H_{l-1}^{Test}))$$

Stochastic depth works well as expected on our ResNet. As shown in Table 2, it reduces the networks depth by 23.12%, very close to the theoretical value 25% mentioned in the paper, resulting in around 25% less training time.

5.2. Image Augmentation

At this stage, we include some basic image augmentations to "cheat" our ResNet, letting it think we have more training data. First, subtract the global mean from each image. Second, randomly flip the image with probability 0.5 as illustrated in Figure 7. A lot more image augmentation will be introduced in detail in the next section.

5.3. Result

From the training process shown in Figure 8, the stochastic depth and image augmentation did provide some help

Figure 7. Left-Right flip an image

on preventing overfit. The training loss and error never go much lower than validation loss and error. It takes a few more epochs to converge, the first pleatu stops at around 60%, the final error rate stops at 43%. On the test data, this improved model achieves an error rate as low as 45.1%, nearly 5% improvement from the base model.

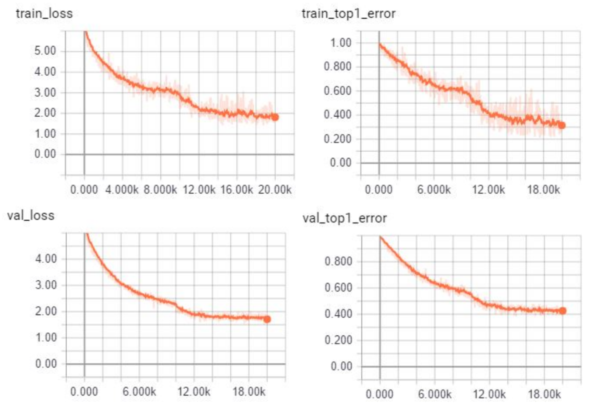


Figure 8. Result of improved model

5.4. Variants and Comparison

Due to its popularity and success on image classification, ResNet attracted a lot of attention and research on it. Many variants of ResNet are proposed. In this section, we take our improved model as baseline and compare its performance with some of its variants.

In the original paper, the author proposed 5 different structures of ResNet, ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152, as shown in Figure 4. Among these 5 structures, only ResNet-18, ResNet-34 and ResNet-50 are included in this comparison since ResNet-101 and ResNet-152 do not fit my GPU resources.

Besides the different structures, different layer setup are also available within each ResBlock, as shown in Figure 9. Due to the time limit of this project and their similarity of the performance[5], only configuration (c) and configuration (e) are tested.

Figure 10 shows the comparison of the performance of these variants. First, the accuracy are very similar. Configuration (e) have slightly lower error rate than configuration (c) on average. We estimate that the similarity between all these variants are also due to the limited number of training images we have - even simple models can easily find the

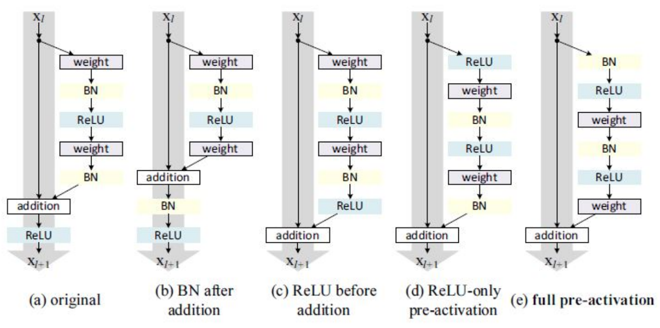
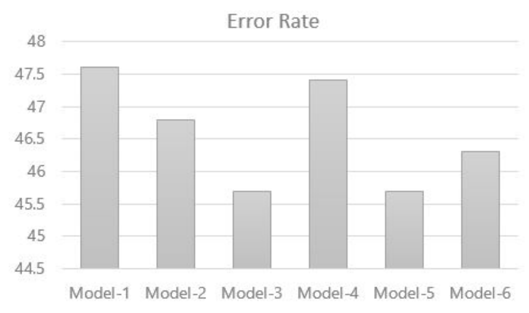
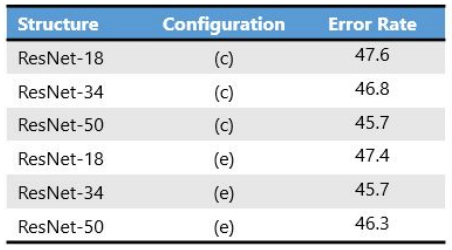


Figure 9. Different configurations for each ResBlock

patterns from the training data, and complex models have no opportunities to demonstrate their power.

So in order to achieve even lower error rate, we need more techniques to overcome the difficulty caused by small dataset. And ResNet-34 with configuration (e) is selected by us for further improvement due to the balance of its high accuracy and lower training time.





Structure	Configuration	Error Rate
ResNet-18	(c)	47.6
ResNet-34	(c)	46.8
ResNet-50	(c)	45.7
ResNet-18	(e)	47.4
ResNet-34	(e)	45.7
ResNet-50	(e)	46.3

Figure 10. Error rates comparion among different structures

6. Heavy Image Augmentation

Upon the completion of all the experiments and comparison on ResNet variants, we chose our ResNet-34 with full pre-activation for further improvement, and take that model to hit the Leaderboard.

In order to make the model more capable of generalizing, we need even more image augmentations. With the help of two python packages, imgaug[7] and cv2, we did a sequence of random transformation every time we feed a mini-batch into the network. A total of 12 transformations are selected, including flipping, cropping, scaling, shifting and etc. Some of them are shown in Figure 11. For one image augmentation operation, a random selection of several transformations are applied to each image, and the intensity of each transformation is also randomly determined within a specified range. These range parameters are manually pre-defined with the criteria that the augmented images have to be obvious to humans eyes, otherwise they will be too strong for our networks.

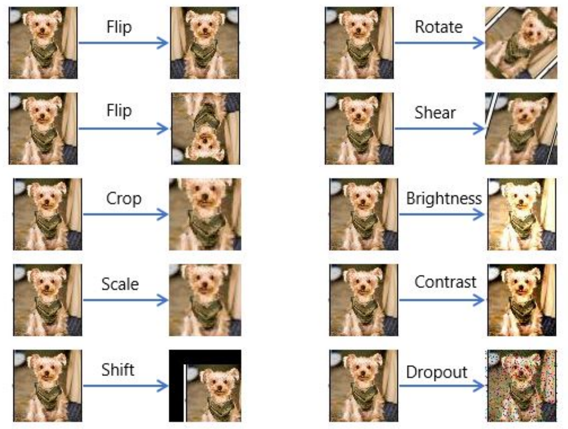


Figure 11. Example of data augmentation

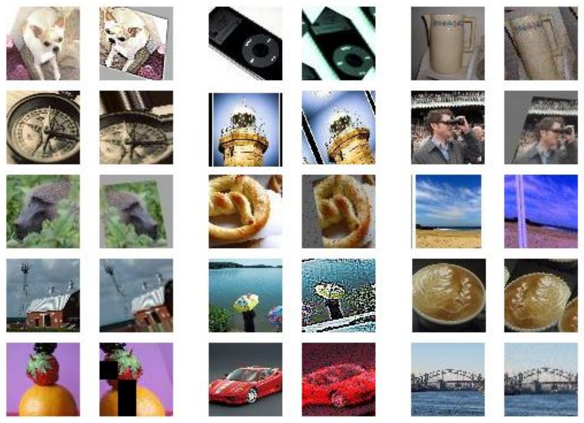


Figure 12. Sample augmented images

Figure 12 shows 15 sample images picked from the output of data augmentation. On one hand, we can easily tell that these are the same items, but on the other hand, their edges, colors or even positions are different enough for networks to treat them as different images. The final result shows that this image augmentation is the most helpful feature for boosting accuracy performance.

7. Final Result

As expected, the heavy image augmentation helps us boost the accuracy significantly. At the same time, it also takes much more time for training - 40,000 iterations to get to the first error rate plateau and totally 70,000 iterations to stabilize the trainable variables. The long training time is also expected, after all, the networks need to get familiar with all transformed images. At the end of the training process, we achieve the final error rate of 34.68%, which is very close to fine-tuned models.

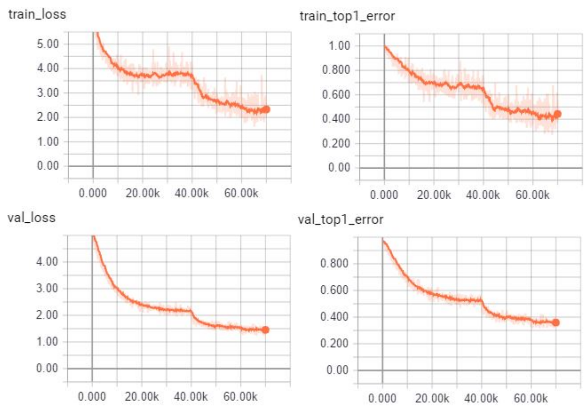


Figure 13. Results of final model

8. Error Analysis

We visualize some interesting examples of top-1 errors made by our final model, shown in Figure 14. In overall, these errors are generally due to the low resolution of the images, the misunderstanding of the main entity in the image, or just the confusion by similar items.

First of all, since all the images are only 64x64 pixels, it is hard to tell which class the image belongs to without the details in the picture. Some of them are even too hard to humans. The [pill bottle] is mis-classified to [Christmas stocking] because of their similar shape, and the colorful strips on its body. The [stop watch] and [compass] have very similar appearance, even humans cannot tell which one it is without more details such as the needles or the numbers on it.

Another type of error is caused by the misunderstanding of the intent of the image. The plate of fruits is classified to [banana] simply because there is indeed a banana in it, however, the official label is [orange]. A plate of food is supposed to be categorized to [meatloaf], but our model says that is a [plate], which is perfectly reasonable. These kind of errors are impossible to fix. There are often more than entities in a single image, our network interprets the images correctly, but does not interprets what we want correctly, and even sometime, I believe there is some judging noise in the data set.

At last, we see some errors are due to the reason that the categories are too close to each other, sometimes there are even some overlap between each other. [Convertible] cars are misclassified to [sports car], but in reality, a large part of [convertible] cars are also [sports cars]. We won't get too much objection if we claim the red car in our example is a sport car. Similarly, our ResNet successfully finds a bottle in the image, but it is not that obvious to tell [pop bottle] from [beer bottle].

In summary, although more than 30% of the images are not classified correctly, a large number of the errors are reasonable, some of them are even un-blamable. We will obtain more accuracy if the training data have higher resolution.

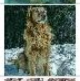
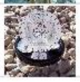
	['sports car', 'sport car']	['convertible']
	['frying pan', 'frypan', 'skillet']	['meat loaf', 'meatloaf']
	['pizza', 'pizza pie']	['plate']
	['banana']	['orange']
	['black widow']	['chain']
	['pop bottle', 'soda bottle']	['beer bottle']
	['Christmas stocking']	['pill bottle']
	['sea cucumber', 'holothurian']	['coral reef']
	['fur coat']	['golden retriever']
	['stopwatch', 'stop watch']	['magnetic compass']

Figure 14. Examples of mis-classified images

9. Conclusion

In this project, we approached the Tiny ImageNet Challenge by training our ResNet from scratch. We first implemented a base model of 34 layers. The curves in the training process and the performance on validation set shows there is a strong sign of overfitting. Then a couple of features, such as stochastic depth and image augmentation, are added to the model to fight against overfitting. The error rate is 43.7% for the improved model. Taking this performance as baseline, we also compare it with more than 5 variants of

ResNet, including different number of layers, different order of layers in each ResBlock and etc. Finally, to hit the Leaderboard of this Tiny ImageNet Challenge, we trained a final model with heavy image augmentation, which helped us achieve error rate of 34.68%, reducing the error rate by more than 10%. At last, some basic error analysis are conducted to see what kinds of errors are made by the model.

References

[1] ImageNet. `http://www.image-net.org/`.

[2] R. Dionne. Residual neural networks are an exciting area of deep learning research. `https://blog.init.ai/`, 2016.

[3] S. Gross and M. Wilber. Training and investigating Residual Nets. `http://torch.ch/blog/2016/02/04/resnets.html`, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[6] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

[7] A. Jung. Image augmentation for machine learning experiments. `https://github.com/aleju/imgaug`, 2016.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[11] 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*, abs/1409.4842, 2014.

[12] M. Telgarsky. Benefits of depth in neural networks. *CoRR*, abs/1602.04485, 2016.