# Depth Dropout: Efficient Training of Residual Convolutional Neural Networks

Jian Guo
Research School of Computer Science
The Australian National University
Email: jian.guo@anu.edu.au

Stephen Gould
Research School of Computer Science
The Australian National University
Email: stephen.gould@anu.edu.au

***Abstract*—Training state-of-the-art deep neural networks is computationally expensive and time consuming. In this paper we present a method that can reduce training time while at the same time maintain nearly the same accuracy as traditional training approaches. This allows for faster experimentation and better use of computational resource. Our method extends the well-known dropout technique by randomly removing entire network layers instead of individual neurons during training and hence reducing the number of expensive convolution operations needed per training iteration. We conduct experiments on object recognition using the CIFAR10 and ImageNet datasets to demonstrate the effectiveness of our approach. Our results show that we can train residual convolutional neural networks (ResNets) 17.5% faster with only 0.4% decrease in error rate or 34.1% faster with 1.3% increase in error rate compared to a baseline model. We also perform analysis on the trade-off between testing accuracy and training speedup as a function of the drop-out ratio.**

## I. Introduction

Since the inception of AlexNet [2] in 2012, convolutional neural networks (CNNs) have dominated large-scale computer vision tasks such as ImageNet object classification [6]. Moreover, state-of-the-art accuracy on these large-scale tasks has improved steadily as the number of layers in CNNs increase, where a layer typically includes a set of convolutional and non-linear operations. Examples include the VGG [8] and GoogLeNet [3] models each having around 20 layers. This phenomena provides evidence that the expressiveness of CNN models is strongly correlated with network depth.

The recently introduced residual network model (ResNet) [7], which forms the basis of our work, distinguishes its architecture from the previous state-of-the-art models via shortcut connections between two or three consecutive layers. The shortcut connections facilitates gradients to be back-propagated from the very top layer of the network to the very bottom layer, which in turn helps when training the parameters of the network. Thanks to this groundbreaking architecture, one is now able to train ultra deep networks on the scale of hundreds of layers. The increased network depth also translates into a large boost in accuracy which has reportedly surpassed human performance in the object recognition task on the challenging ImageNet dataset [9]. We refer the reader to the ResNet paper [7] for technical details.

Despite the impressive performance of deep networks, their training time is prohibitively slow. The 152-layered residual network, for example, takes approximately two weeks to train on eight nVidia Titan X GPUs. With a two-week turnaround time on experiments, researchers can run less than 26 trials per year, which is very limiting when tuning hyper-parameters (over multiple runs) is still an important ingredient in getting optimal performance out of these models.

The contribution of this paper is to speed up the training by performing dropout along the depth of a ResNet model. Specifically, during training we randomly bypass layers altogether on an image by image basis, thereby reducing the amount of computation needed per training example. Our experiments on two standard computer vision datasets demonstrate that our method can achieve 34.1% speedup with a dropout ratio of 0.5 while still maintaining nearly same level of accuracy as the baseline model. This effectively gives researchers the ability to run additional experiments when their compute resources are constrained (i.e., limited access to GPUs).

## II. Related Work

Our proposed method is inspired by two lines of research. The first is dropout [4], originally proposed as a mechanism to regularise training of convolutional neural networks. In the original formulation of dropout, at each training iteration and for each training instance the output of individual neurons within a layer are randomly suppressed. That is, the output for a randomly selected set of neurons is set to zero. To distinguish this method from our approach we refer to it as *width dropout*. A parameter known as the dropout ratio controls the probability that any individual neuron will be suppressed.

Width dropout can be interpreted in two ways. First, dropout provides a mechanism for regularising training that prevents co-adaption of features within each layer of the network. As a result, more diverse features are learned. Second, dropout can be seen as training an exponential ensemble of sub-networks on the fly. In other words, each training iteration operates different subsets of the original network. Both interpretations are consistent with the observation that width dropout helps prevent over-fitting and leads to models with better accuracy on unseen test data.

Despite only training a subset of the network at each iteration, width dropout does not result in any speedup, i.e., the computational load is the same independent of the dropout ratio (including no dropout). This is because from an implementation perspective, random memory access is inefficient compared to

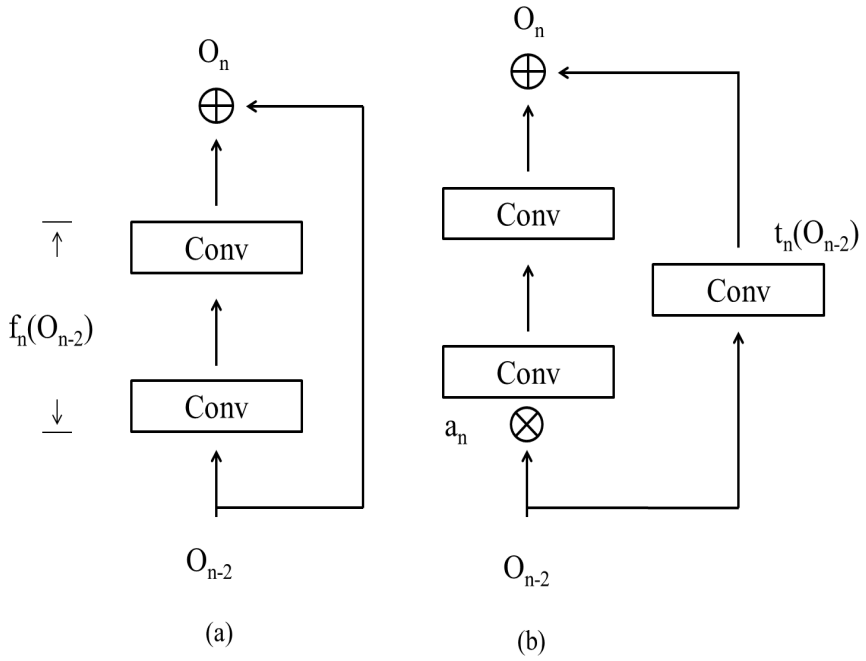



Fig. 1. (a) Original ResNet (b) ResNet with Depth Dropout. For simplicity only convolutional layers are depicted.

linear memory access especially on a GPU due to coalesced memory access strategy employed by GPUs. Or put another way, it is more efficient to perform convolution over the entire image plane than on selected regions of the image. Therefore, width dropout is always implemented as a full convolution followed by element-wise multiplication by a mask to zero out the suppressed neurons.

In contrast to this approach, our proposed method randomly skips entire layers along the depth direction. We refer to this approach as *depth dropout*. Unlike width dropout the motivation here is not to prevent co-adaption of features but to reduce training time. By skipping entire layers the convolutional operations within those layers can be avoided altogether unlike width dropout, thus giving a computational speedup. Note that depth dropout is complementary to width dropout and both can be used when training deep convolutional neural networks.

Now given that depth dropout suppresses the output of entire layers the information path through these layers is effectively blocked. As such, depth dropout only makes sense if there exists an alternative path for information to flow. This leads to the second inspiration for our work—residual networks [7]. In a residual network, each residual unit consists of a primary path of two or three convolution layers, and an alternative path which short-circuits the input to the output of the residual unit in question. More specifically, the output of the primary path through the convolutional layers is added to the unit's input to produce the output of the unit. This type of architecture allows us to randomly skip the primary path while the shortcut connection provides an alternative path for information to flow. In He et al. [7], variants of their model equip the shortcut connections with trainable weights, which they termed projection operations. We adopt a similar strategy in our work but for different reason as discussed in the next section.

The recently introduced idea of stochastic depth [5] is similar in spirit to our work. In their work, different dropout ratios are applied to each residual unit as a linear function of depth. Our work differs from their approach in two respects. First, we apply the same dropout ratio uniformly to each residual unit and therefore we achieve uniform speedup for all residual units resulting in overall faster training. Second, we equip our shortcut connections with trainable weights as suggested by He et al. [7] As such, our shortcut connections are able to automatically adjust to the noise introduced by dropout. Without trainable weights on the shortcut connections, training can diverge if high dropout ratios exist throughout the entire network.

## III. Depth Dropout

In this section we formally introduce our depth dropout model and discuss implications on the training procedure and provide an analysis of training speedup as a function of dropout ratio.

### A. Model Formulation

Consider the $n$-th two-layered residual unit in the original ResNet architecture. The unit receives input $o_{n-2}$ from the unit below, processes it through two convolutional layers, and then adds the input to produce the output $o_n$ for the unit. We can write this formally as

$$o_n = f_n(o_{n-2}) + o_{n-2} \tag{1}$$

where $f_n$ abstracts away the operations of the convolutional layers in transforming the input. Figure 1(a) provides a visual illustration of the layer. In the figure, we denote the path with two convolutional layers as primary path. We also denote the path that directly connects the bottom and the top of the residual unit as shortcut path. Note that the figure only shows convolutional layers to highlight the main computational components. However, it should be understood that these layers also include batch normalisation [10] and element-wise non-linear transformation. We adopt this convention when referring to convolutional layers for the rest of our paper.

Now we consider a slightly generalised form of the original residual network where we allow the convolutional path to be scaled by a factor $a_n$ as follows:

$$o_n = a_n f_n(o_{n-2}) + o_{n-2} \tag{2}$$

where the original residual network can be obtained by settings $a_n$ to one.

Observe that we do not need to restrict $a_n$ to be deterministic. If we instead let $a_n$ to be generated from a random distribution, we will have a network with random architecture. In this paper, we restrict our attention to Bernoulli distribution (sampling $a_n$ to be either zero or one), which given a sampling of the $a_n$'s will result in a random subset of the original residual network. This approach delivers us a speed advantage since computation of layers with zero $a_n$ can be skipped as we discuss below.

The above model can be further generalised by allowing the shortcut path to also be transformed. In our early experiments we found this necessary to prevent the training process from diverging when high dropout ratios are used. Formally we have,

$$o_n = a_n f_n(o_{n-2}) + t_n(o_{n-2}) \tag{3}$$

where $a_n$ is drawn from Bernoulli distribution and $t_n$ is a trainable function. In our work we implement $t_n$ as yet another convolutional layer as illustrated in Figure 1(b).

In this formulation only the primary path of the residual unit undergoes random dropout while the newly introduced transformed shortcut path is always present. Our experiments show that $t_n$ with trainable weights learns to cope with the noise introduced by the dropout and prevents the training process from diverging. We next discuss details of training with depth dropout.

### *B. Training with Depth Dropout*

We follow the standard approach of using stochastic gradient descent (SGD) on mini-batches of training examples to learn the parameters of a ResNet model. At each SGD iteration, we first sample the $a_n$ from a Bernoulli distribution such that

$$a_n = \begin{cases} 0 & \text{with probability } p \\ 1 & \text{with probability } 1-p \end{cases} \tag{4}$$

where $p$ is the expected depth dropout ratio. Then for all layers with $a_n$ set to zero, we physically skip computation along the primary path for that unit, effectively removing those layers from the network (for that iteration). In this case, all information from the $n$-th layer flows through $t_n$. Computation is saved on both the forward (evaluation) and backward (gradient propagation) passes.

Note here the difference between our approach and the conventional width dropout approach. In the width dropout settings each individual feature in the mini-batch of an SGD iteration receives an independent random draw. In our approach, each residual unit receives an independent random draw but the draw for this residual unit applies to all the training samples in the minibatch. The distinction is important since it results in greater computational efficiency because we can now skip all computation along the primary path of units with $a_n = 0$. Adopting a strategy of separate draws for each individual sample would harm data parallelism across GPU cores within the mini-batch.

Having outlined our training procedure we next provide a theoretical analysis of the computational advantage of training with depth dropout.

### *C. Computational Analysis of Depth Dropout*

By skipping the computation of the primary path within a subset of residual units we gain linear speedup compared to the baseline residual network. To formalize this, let $\tau_1$ and $\tau_2$ denote the running time of the primary path and the shortcut path of a residual unit. Here we include the total time of both the forward and backward passes. Thus the time to compute the output of a residual unit is $\tau_1 + \tau_2$, where we have ignored the very small overhead associated with combining the paths and other minor housekeeping tasks. Now, with the introduction of depth dropout, the primary path is skipped with probability $p$. Thus the expected running time becomes $E_p[\tau_1] + \tau_2$, where $E_p[\cdot]$ is the expectation with respect to the Bernoulli distribution with parameter $p$.

For example, suppose we have a residual unit with two convolutional layers in the primary path $f_n$ and one convolution layer in the shortcut path as depicted in Figure 1(b). Again, by convolution layer, we refer to a stack of a convolutional layer, a batch normalisation layer and a non-linear layer. We assume that two convolution layers on the primary path have 3×3 filters while the convolution layer on the shortcut path has 1×1 filters. We further assume that all convolution layers have same input dimensions. Thus, the running time of the convolution layer with 1×1 filters is theoretically $\frac{1}{9}$ of the convolutional layer with 3×3 filters. Let $\tau$ denote the total running time of a single convolutional layer. The total running time of the baseline residual unit (Figure 1(a)) is $2\tau$. With a dropout ratio of $p = \frac{1}{2}$ the expected running time of the proposed residual unit (Figure 1(b)) is $\frac{1}{2}(\tau + \tau) + \frac{1}{9}\tau = \frac{10}{9}\tau$. Thus, the theoretical time saving is 45.5%.

Since our residual path has trainable weights, one may argue that our network has more complexity than the original baseline residual network. Nevertheless, randomly dropping out the primary path will effectively train similar or even less number of weights on expectation (with $p = \frac{1}{2}$) during each iteration compared to the baseline residual network. To see this,

consider the following example. Supposed that there are two convolutional layers on the primary path and one convolutional layer on the shortcut path and the dropout ratio is $p = \frac{1}{2}$. The expected number of trainable convolutional layers with depth dropout is $\frac{1}{2}2 + \frac{1}{9} = \frac{10}{9}$, which is on expectation less than the baseline residual network.

### D. Inference

At test time we fix $a_n$ to one. Unlike width dropout we do not divide the feature outputs by the dropout ratio as dropout in our model is performed layer-wise and we have already taken into account the effect of the dropped residual units by learning a transformation $t_n$ on the shortcut path. Note, that while the introduction of $t_n$ implies additional processing cost (at test time) over the original residual network, the test time cost is negligible compared the the cost to train the network.

## IV. Experimental Results

We conduct experiments on two standard image classification datasets and compare against results reported in the literature. Specifically, we provide results on the small CIFAR10 [1] dataset and the much larger ImageNet [6] dataset. All of our networks are trained using the Caffe deep learning software framework [11].

Our experiments are designed to study two aspects of our proposed depth dropout method. We first carry out experiments to study the variation of error rate as a function of dropout ratio. Next, we perform experiments on error rates versus training time to determine whether networks trained with depth dropout can achieve the same level of accuracy in less time than the baseline ResNet model (without dropout).

### A. CIFAR10 Experiments

The CIFAR10 dataset consists of 50k training images and 10k test images of size $32 \times 32$ pixels. Each image in the dataset is categorised into one of 10 object classes (such as "dog", "bird", "airplane", etc.). Our training protocol strictly follows the original residual network paper [7]. That is, we augment the CIFAR10 images by padding a border of four zeros to each side of the image and subtract the pixel-wise mean (over the training dataset) from all image pixels. In addition, we perform random horizontal flipping of the images during training to augment the dataset. Our initial learning rate is set to 0.1. We train up to 64,000 iterations and divide the learning rate by 10 after 32,000 iterations and again after 48,000 iterations. We train our models on two nVidia Titan X GPUs with a minibatch size of 64 training examples per GPU. The baseline network we used is the 56-layered network (ResNet-56) with the same architecture as described in He et al. [7]. All reported error rates are computed on the full test image set using model parameters obtained at the final training iteration.

The two convolutional layers on the primary path have are equipped with $3 \times 3$ filters. The convolutional layers on the shortcut path are equipped with $1 \times 1$ filters followed by batch normalisation and non-linearity. We also found that disabling the dropout for the residual units that performs spatial downsampling (i.e. where the convolutional stride of 2) of the input feature maps improves accuracy by a noticeable margin while still maintains reasonable speedup. For the 56-layered residual network, there are a total of 27 residual units with 3 out of them performing spatial dowmsampling as as described in He et al. [7]. Let $\tau$ denote the total running time of the primary path of a residual unit. The theoretical running time of the baseline model (i.e. 27 residual units) is $27\tau$. The theoretical running time our proposed model with those 3 residual units disabling dropout and 0.5 dropout ratio is $\frac{1}{2}24\tau + 3\tau + 24\frac{1}{9}\tau \approx 17.667\tau$. Thus, the time saving for 0.5 dropout ratio is about 34.6%. The time savings for dropout ratios of 0.25 and 0.75 are 12.3% and 56.7% respectively.

*1) Error Rates versus Dropout Ratio:* We first analyse the effect of dropout ratio on error rate training following the protocol outlined above. Results for different dropout rates are shown in Table I and summarised in Figure 2(a). As can be seen from the table, the baseline method (no dropout) achieves 0.085 error rate, which is similar to the result of 0.069 reported by He et al. [7], but which uses their in-house deep learning software instead of Caffe. With a dropout ratio of 0.25 the model achieves a relative error rate of 0.4% lower than the baseline. We hypothesize that this is because a dropout ratio of 0.25 is good balance that prevents over and under-fitting, which yields the most robust model among all the other dropout ratios. As the dropout ratio increases beyond 0.25 the error rate worsens slightly, increasing by 1.3% at 0.5 dropout and up by 2.7% at 0.75 dropout ratio.

We also plot top-1 error rate as a function of training iteration for different dropout ratios (see Figure 3(a)). As can be seen the effect of increasing dropout ratio is consistent with the observations above throughout the training process with all converging after about 32,000 iterations. However, despite all experiments converging after a fixed number of iterations, the actual running time to reach convergence differs with dropout ratio, which we analyse next.

*2) Error Rates versus Training Time:* We begin by discussing our experimental protocol to carry out this study. Since all of our experiments are run on a shared GPU server, the training speed can vary at different points of time during the training process due to variations of the GPU and server load outside of our control. We circumvent this problem by explicitly running speed benchmarks for a small number of iterations on the networks with different dropout ratios. Concretely, we run the forward and backward pass for 500 iterations for each network on the same (single) GPU exclusively, i.e., with no other jobs running on that GPU. We then record the average running time over the 500 iterations for each network. Since the computation cost of each mini-batch is independent of the data within the mini-batch nor the current network parameter settings (other than dropout ratio), this gives a reliable estimate of the running time per iteration. We use this estimate to produce the error versus running time plots shown in Figure 3(b). Notice that the higher the dropout ratio the faster the network converges.

We also provide a table of average running times for both

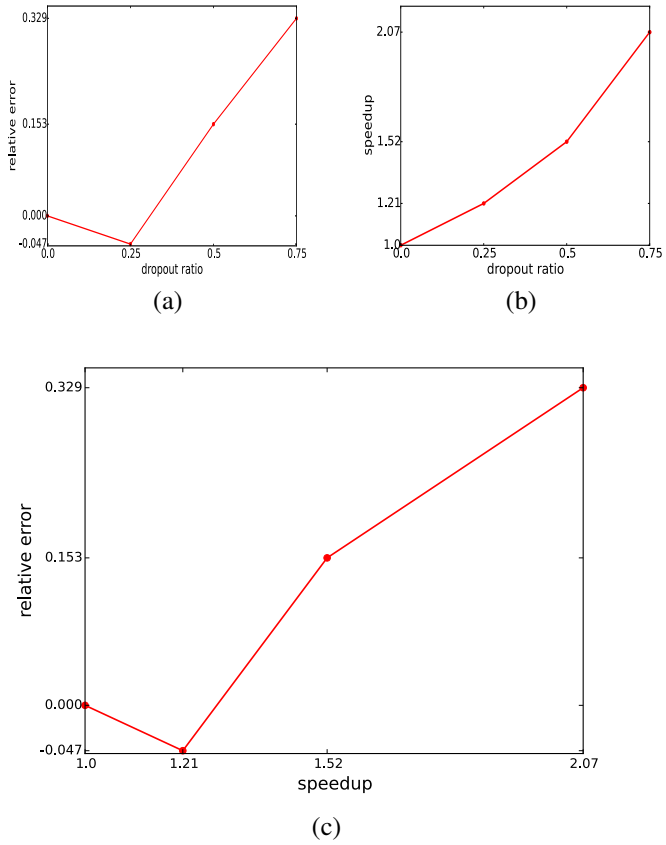



Fig. 2. Relative error rate and training time as a function of dropout ratio for the CIFAR10 dataset in (a) and (b), respectively. Also shown is error rate versus training time in (c). Relative error is defined as the error of current model divided by baseline error minus one.

the forward and backward passes in Table II. The speed up obtained with dropout ratios of 0.25, 0.5 and 0.75 over the baseline are 0.175, 0.341 and 0.516, respectively, which are close to theoretical speedups of 0.123, 0.346 and 0.567.

Figure 2 summarises the results of our CIFAR10 experiments. Displayed are the change in error rate relative to the baseline (at convergence) as a function of dropout ratio in (a); training speedup as a function of dropout ratio in (b); and relative error versus speedup in (c). As shown in (c), the increase of relative error rate appears sub-linear as the increase of speedup. We hypothesise that depth dropout has built-in regularisation effect as the original dropout. The model trained with depth dropout is robust. Therefore, the relative error grows sub-linearly as the increase of speedup.

### *B. ImageNet Experiments*

We also conduct experiments on a very large-scale dataset. Here we used the ImageNet [6] dataset, which consists of 1.2 million training and 50k validation colour images. The size of the the raw images varies but averages to about $400 \times 500$ pixels. The dataset contains 1000 different classes (with images in the validation set balanced over all classes, i.e., 50 images per class). The network we used in this experiment is

TABLE I
DROPOUT RATIO VS ERROR RATE ON CIFAR10

| Dropout ratio | Error rate |
|---|---|
| 0.0 | 0.085 |
| 0.25 | 0.081 |
| 0.5 | 0.098 |
| 0.75 | 0.113 |

TABLE II
AVERAGE RUNNING TIME (MS) PER ITERATION ON CIFAR10

| Dropout ratio | forward | backward | total |
|---|---|---|---|
| 0.0 | 64.0 | 104.5 | 168.5 |
| 0.25 | 53.6 | 85.4 | 139.0 |
| 0.5 | 43.5 | 67.6 | 111.1 |
| 0.75 | 33.0 | 48.5 | 81.5 |

TABLE III
DROPOUT RATIO VS ERROR RATE ON IMAGENET

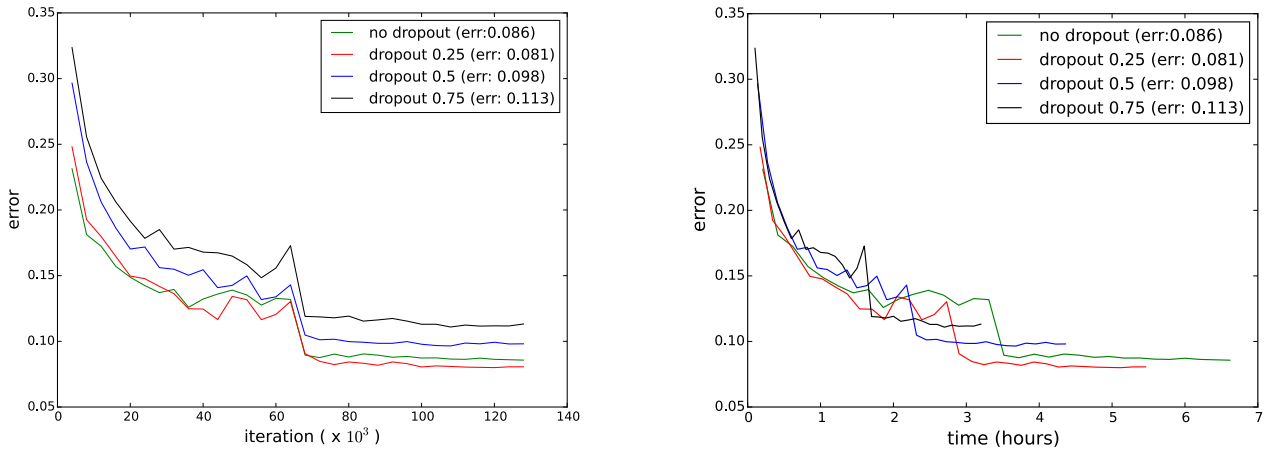
| Dropout ratio | Error rate |
|---|---|
| 0.0 | 0.340 |
| 0.5 | 0.341 |

the 18-layered ResNet. We choose 18-layered network to fit into our computational budget. Again, our training protocol strictly follows the original residual network paper [7]. Our initial learning rate is 0.1 and we use a minibatch size of 256 training examples. We train up to a maximum of 600k iterations. We also use the same training data augmentation techniques employed by the original residual network paper. Namely, we perform random scales and random flipping of the images during training. During test, we report our accuracy figures using the center crop and using network parameters obtained at 600k iterations.

Due to the size of the dataset we only run experiments with a dropout ratio of 0.5. As seen in Table III, the performance gap between the baseline and depth dropout is 0.1%, which is small. As with the CIFAR10 experiments we also produce plots of top-1 error rate as a function of training iteration and running time in Figures 4(a) and (b), respectively. Note here that the running time is significantly longer than on the smaller CIFAR10 dataset, but we again see a speedup over the baseline when using dropout. Here the learning curve for both models tends to flatten out after around 300k iterations with a small gap between the two models.

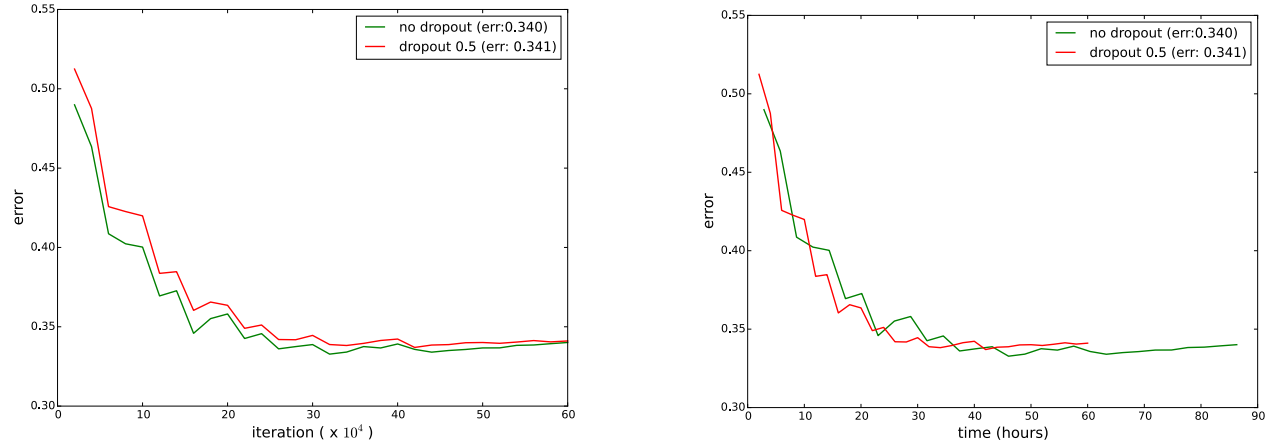
## V. CONCLUSION AND FUTURE WORK

In this paper, we proposed depth dropout, a novel method for training very deep residual networks. Depth dropout improves training time with negligible difference in the accuracy of the final model on hold-out data as compared to a baseline model without dropout. While the current speedup of around 30% may




(a) (b)

Fig. 3. Top-1 test set error on CIFAR10 versus (a) training iteration number and (b) training time in hours.




(a) (b)

Fig. 4. Top-1 validation error on ImageNet versus (a) training iteration number and (b) training time in hours.

appear small, the gain needs to be considered in light of the very long training times and the large number of experiments that researchers typically need to run. At the end of the day even a small saving in computation allows researchers to run many more experiments when working with large-scale data and in environments with limited compute resources.

The baseline network used in our CIFAR10 experiments is relatively deep, containing 56 convolutional layers, which we denote by ResNet-56. In the ImageNet experiments, however, we used ResNet-18 as our baseline network. We picked this baseline network mainly for faster experimental turnaround to meet the time budget of our project. Due to the built-in regularisation effect of depth dropout, we hypothesise that less deep networks such as ResNet-18 benefit less from deep networks, which we observe in our experimental results but would need to be confirmed in future work. Nevertheless, the effectiveness of our proposed method is promising.

In our work we used a fixed dropout ratio for all layers in the network. Recent work by Huang et al. [5] showed that the dropout ratio can be varied as a function of depth, but here it is not clear whether the approach can obtain the same level of speedup. In the future we would like to explore data dependent dropout strategies. That is, have the dropout rate adapt to the training data, which has the potential to improve both model accuracy and significantly speed up training time. We would also like to explore alternative transformations for the shortcut path to have even greater asymmetry in the computational cost between dropout and no dropout, thus leading to further speedups.

## VI. Acknowledgement

This work was funded in part by the Australian Centre for Robotic Vision (ACRV). We are also grateful for the support

from National Computational Infrastructure (NCI) for providing GPUs to run our experiments.

## REFERENCES

[1] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf. 2009

[2] A. Krizhevsky, I. Sutskever, and G. Hinton. *Imagenet classification with deep convolutional neural networks*. In NIPS, 2012.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In CVPR, 2015.

[4] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. *Improving neural networks by preventing coadaptation of feature detectors*. arXiv:1207.0580, 2012.

[5] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. *Deep networks with stochastic depth*. arXiv preprint arXiv:1603.09382, 2016.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. *Imagenet: A large-scale hierarchical image database*. in Proc. CVPR, 2009.

[7] K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning*. In CVPR 2016.

[8] K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In ICLR, 2015.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and Li Fei-Fei *ImageNet Large Scale Visual Recognition Challenge*. In IJCV, 2015.

[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell *Caffe: Convolutional Architecture for Fast Feature Embedding*. arXiv preprint arXiv:1408.5093, 2014.