# MonolithNet: Training monolithic deep neural networks via a partitioned training strategy

Rene Bidart University of Waterloo, ON, Canada
Alexander Wong University of Waterloo, ON, Canada

## Abstract

In this study, we explore the training of monolithic deep neural networks in an effective manner. One of the biggest challenges with training such networks to the desired level of accuracy is the difficulty in converging to a good solution using iterative optimization methods such as stochastic gradient descent due to the enormous number of parameters that need to be learned. To achieve this, we introduce a partitioned training strategy, where proxy layers are connected to different partitions of a deep neural network to enable isolated training of a much smaller number of parameters to convergence. To illustrate the efficacy of this training strategy, we introduce MonolithNet, a massive residual deep neural network consisting of 437 million parameters. The trained MonolithNet was able to achieve a top-1 accuracy of 97% on the CIFAR10 image classification dataset, which demonstrates the feasibility of the proposed training strategy for training monolithic deep neural networks to high accuracies.

## 1 Introduction

In recent years, deep learning has revolutionized many aspects of machine learning through the use of hierarchical feature extractors trained through backpropagation, rather than through the use of hand-engineered features which were commonly used in the past [1]. In particular, a type of deep neural network known as deep convolutional neural networks (CNN) has demonstrated state-of-the-art performance on a number of visual perception tasks when compared to other machine learning algorithms [2].

It was discovered that creating deep neural networks consisting of a large number of layers was crucial to improving the performance in image recognition, but with deeper networks came an increased difficulty in training the networks to convergence[3]. A number of solutions have been introduced in recent years to tackle this problem and improve the training of deeper networks. One such solution came in the form of batch normalization [4], which normalizes the layer's inputs per mini-batch, reducing internal covariate shift, and thus allowing for more reliable and faster training. Another breakthrough in training deeper networks came in the form of deep residual learning [5], where an architectural change was introduced in the form of residual connections.

Residual deep neural networks surpassed the performance of previous networks through the use of skip connections between the layers of the CNN. This enabled the network to learn identity mappings between layers, enabling networks as deep as 1000 layers to be trained, setting a new benchmark for image recognition performance.

In this work, we explore and investigate a partitioned training strategy to train even larger networks by selectively training partitions of a network in a multi-stage manner. To illustrate the efficacy of this training strategy, we create a massive deep convolutional neural network called MonolithNet, which is comprised of multiple parallel wide residual blocks [6].

Therefore, the key contributions of this paper are: i) the introduction of a novel, massively wide residual network architecture, and ii) a multi-stage partitioned training strategy to overcome the difficulty of training such a large network to convergence.

This paper is organized as follows. Section 2 provides a detailed description of the proposed MonolithNet network architecture. Section 3 presents the multi-stage partitioned training strategy used to train MonolithNet to convergence. Section 4 presents an evaluation of the performance of the trained MonolithNet using the CIFAR10 dataset, and compares it with other deep convolutional neural networks presented in previous literature.

## 2 MonolithNet Architecture

The underlying goal behind the design of the proposed MonolithNet network architecture is two-fold. First, we with to construct a deep convolutional neural network architecture that can better

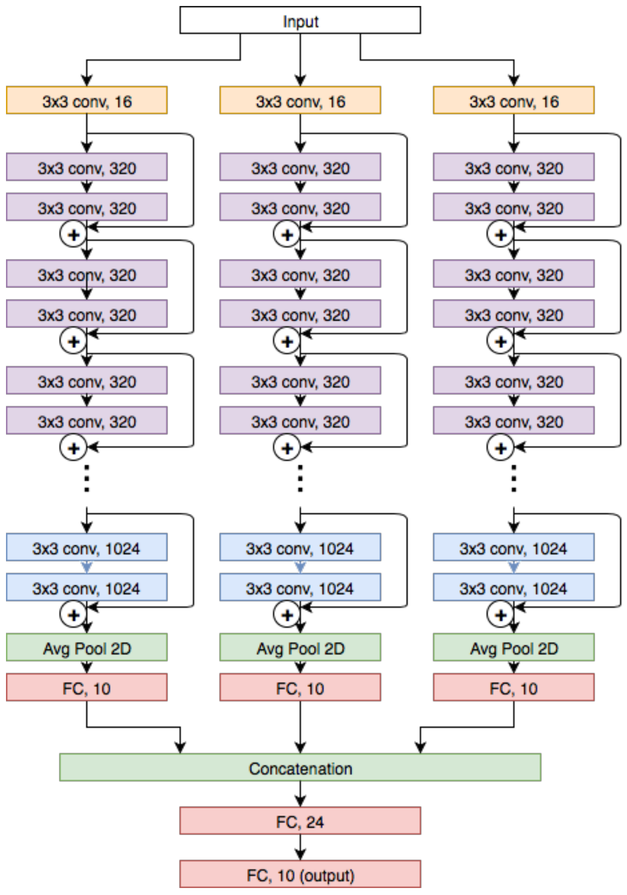

*Fig. 1:* MonolithNet network architecture. An input layer feeds the same input to three parallel wide residual blocks, where each block follows a wide residual architecture [6] with a depth of 28 and feature expansion factor of 20. Each of these blocks extract different sets of quantitative features, which are then fed into a concatenation layer, followed by two fully connected layers to output a classification prediction.

learn a larger, more diverse set of features to improve classification performance. Second, we wish to construct a massive deep convolutional neural network architecture that we can use to demonstrate the efficacy of a partitioned training strategy for training such a massive network to convergence. To achieve this goal, we leverage the notion of residual learning first proposed in [5], which has not only demonstrated state-of-the-art performance for a wide variety of applications such as general image recognition, but has been recognized in research literature for its terrific ability to perform well both for feature extraction and fine-tuning[7].

In deep residual networks, skip connections are added between convolution blocks, so the network has the ability to learn the identity function and bypass convolutional blocks. Residual mappings are of the form:

$$x_{l+1} = x_l + f(x_l, W_l) \tag{1}$$

Where $f$ is a convolutional block at layer $l$ parametrized by $W_l$. In this particular architecture, $f$ is two convolutional layers using batch normalization and the ReLU activation function. The skip connection allows the network to ignore the function $f$ by pushing its weights close to 0, in which case the identity map is learned.

In the proposed MonolithNet architecture, we leverage a particular variant of the deep residual neural network architecture known as Wide residual network architecture [6]. The main differences between the convolutional deep residual network architecture and the wide residual network architecture lies in the fact that wide residual network architectures leverage fewer layers, but in each layer there are significantly more filters. This wide residual network architecture has been shown to provide superior results to the original deep residual neural network architecture on a variety of data sets, while being significantly less complex architecturally when compared to architectures designed through existing neural architecture search methods, and as such is a good building block for the proposed MonolithNet.

The proposed MonolithNet network architecture is comprised of three parallel wide residual blocks, where each block follows a wide residual architecture [6] with a depth of 28 and feature expansion factor of 20 (which means that the number of filters is 20$\times$ greater than that of the original deep residual neural network architecture). As such, each block of MonolithNet consists of a different set of learned weights to capture diverse feature sets. The features of the last layers in each of the three parallel wide residual blocks within the proposed MonolithNet architecture are combined in a concatenation layer, which is then fed into a 16-neuron fully connected layer, followed by a ReLU layer, which then feeds into a final fully connected layer where the number of neurons is equal to the number of labels being predicted.

## 3 Partitioned Training Strategy

Training a massive network such as MonolithNet to convergence while avoiding overfitting is extremely difficult because of the large number of parameters. To tackle the aforementioned problem, we leverage a partitioned training strategy where proxy layers are connected to different partitions of a deep neural network to enable isolated training of a much smaller number of parameters to convergence.

The targeted training strategy for training MonolithNet is shown in Fig. 2, and consists of three main stages:

1. Partitioned training of individual wide residual blocks
2. Partitioned training of fully connected layers.
3. End-to-end fine-tuning of the full network.

**1. Partitioned training of individual wide residual blocks**
As the first stage of the partitioned training strategy, each of the three wide residual blocks are trained in isolation by freezing the weights of the other wide residual blocks, and a proxy fully connected output layer is connected to the particular wide residual block we wish to train. This ensures that only one particular wide residual block will be trained at a time, leaving the other blocks unaffected during this partitioned training process. Given that we wish to also encourage diverse feature learning, we train each of the individual wide residual blocks with a randomly selected subset of training data. This ensures that each wide residual blocks will converge to a different set of weights, and thus learn more diverse features when compared to the other wide residual blocks. This partitioned training process is repeated for each of the wide residual blocks.

**2. Partitioned training of fully connected layers**
After the partitioned training of individual wide residual blocks, we now focus on the partitioned training of the fully connected network layers in isolation by freezing the weights of the individual wide residual blocks. The rationale behind this is that, since the full connected layers are initialized with random weights, there is a strong probability that the network will not train to convergence if the entire MonolithNet network architecture is trained end-to-end at this point. Therefore, by freezing the weights of all the individual wide residual blocks while the fully connected layers is being trained, we allow the fully connected layers to be trained in isolation and converge to a good solution. In addition, since training is performed only on the fully connected layers, the time to convergence is greatly reduced.

**3. End-to-end fine-tuning of the full network**. After the fully-connected layers in MonolithNet have been trained in isolation, the entire network undergoes an end-to-end fine-tuning process to allow for all of the weights in the network to be trained in unison to allow all components of MonolithNet to act cohesively as a collective. In this part of the partitioned training process, we backpropagate the gradient though the entire network, including each of the individual wide residual blocks and the fully connected layers as a whole, thus optimizing the weights of the entire network. This is done at a lower learning rate than the initial training.

## 4 Implementation Details

Pre-processing is used during the training and testing process, with the input images being normalized to 0 mean and standard deviation. Data augmentation is used during training to reduce overfitting and help generalization, with random horizontal flips and random crops. Crops are taken by padding the original image by 4 pixels on each side, and cropping a random 32x32 section of this.

Stochastic gradient descent with momentum is used for all parts of the training process. Momentum of 0.9 and weight decay of 0.0005 is used for all stages of the training policy.

Different learning rates were used for each stage in the partitioned training process. For stage 1, a learning rate of 0.1 is used, and this is decreased by a factor of 0.2 on epochs 60, 120 and 160 and is trained for a total of 200 epochs. In stage 2, partitioned training is performed for 40 epochs with a learning rate of 0.005, decreased by a factor of 0.3 every 13 epochs. For stage 3, a much lower learning rate of 0.0001 is used to train the network for 25 epochs, and this is decayed by a factor of 0.2 after 13 epochs.

In stage 1, a batch size of 128 is used to stay consistent with [6], while in stage two this is increased to 300. In stage 3, the batch size is reduced to 88 to fit in GPU memory given the size of MonolithNet. All networks are implemented and trained using [8].

## 5 Experiments

To test the efficacy of the partitioned training strategy and the resulting MonolithNet, we evaluate its accuracy on the CIFAR10 dataset [9], a widely-used image classification benchmark composed of 60000 32$\times$32 RGB images divided into 10 independent classes. We used the standard training-testing split, and used similar augmentation and normalization strategies as seen in other networks tested on this dataset. For evaluation purposes, 11 state-of-the-art deep convolutional network architectures were also compared in this study.

As shown in Table 1, it can be observed that the proposed MonolithNet has significantly more parameters than other deep convolutional network architectures, with its 457 million being more than 10$\times$ more than the next largest network architecture. Furthermore, the top-1 accuracy of the proposed MonolithNet reached 97% on the CIFAR10 image classification dataset, which is noticeably higher than 10 of the tested state-of-the-art hand-engineered deep convolutional neural network architectures compared in this study. One of the state-of-the-art hand-engineered deep neural networks (Shake-Shake 2x96d), which leveraged special training regularization in the form of stochastic regularization, perform similarly to MonolithNet (0.15% higher). As such, the accuracy of MonolithNet is impressive given that no automated network architecture search, advanced data augmentation, or special training regularization strategies were employed to achieve this level of accuracy, and so there could be promise in incorporating these methods into MonolithNet.

These results show the feasibility of training very large networks using this partitioned training strategy, and the effectiveness of the presented MonolithNet network architecture.

## 6 Conclusion

In this paper, we introduced MonolithNet, a massive deep network composed of multiple parallel wide residual blocks, as well as a partitioned training strategy where partitions of the network were trained individually. Experimental results achieved an accuracy of 97% in the widely-used CIFAR10 benchmark dataset demonstrated the efficacy of the proposed network and the partitioned training strategy in training such massive deep neural network architectures to convergence. In the future, we hope to investigate using more complex constituent networks as well as improved data augmentation strategies. Furthermore, a more comprehensive and fundamental trade-off analysis between the number of wide residual blocks within the network and the level of performance achieved would be quite useful to better understand network design.

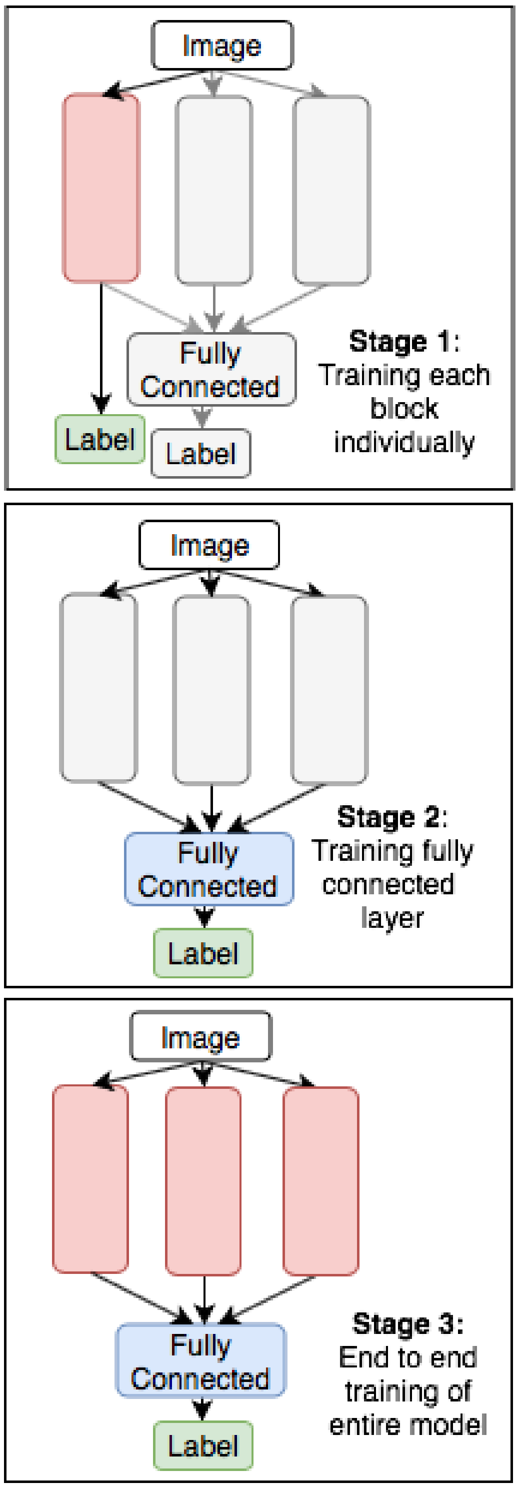
Image
Fully Connected
Label
Label
Stage 1: Training each block individually
Image
Fully Connected
Label
Stage 2: Training fully connected layer
Image
Fully Connected
Label
Stage 3: End to end training of entire model

*Fig. 2:* Partitioned training strategy for MonolithNet. In stage 1, each of the wide residual blocks are trained individually in isolation, while in stage 2 the weights of the three wide residual blocks are frozen, and only the fully connected layers is trained in isolation. In stage 3, the entire network is trained in an end-to-end to promote cohesion of all components of the network.

## Acknowledgments

The authors thank the Natural Sciences and Engineering Research Council of Canada, the Canada Research Chairs Program, and the Queen Elizabeth II Graduate Scholarship in Science & Technology for partially supporting this research. Furthermore, the authors thank Nvidia for the GPUs used in this study that were provided as part of a hardware grant.

*Table 1:* Comparison of different hand-engineered deep convolutional network architectures in terms of the number of parameters and top-1 accuracy on CIFAR10. Networks marked with * leverage special training regularization techniques.

| Network | Parameters (millions) | Accuracy |
|---|---|---|
| DenseNet (L=100, k=24) [10] | 27.2 | 3.70% |
| DenseNet-BC (L=100, k=24) [10] | 25.6 | 3.46% |
| PyramidNet (alpha=270) [11] | 28.3 | 3.73% |
| PyramidNet (bottleneck,$\alpha$=270) [11] | 27.0 | 3.38% |
| PyramidNet (bottleneck,$\alpha$=200) [11] | 26.0 | 3.38% |
| FractalNet [12] | 38.6 | 4.60% |
| ResNet-101 [5] | 1.7 | 6.43% |
| ResNet1202 [13] | 19.4 | 7.93% |
| Pre-activation ResNet [13] | 10.2 | 4.62% |
| WideResNet-28-10 [6] | 36.5 | 3.89% |
| Shake-Shake 2x96d* [14] | 26.2 | 2.86% |
| **MonolithNet** | **437** | **3.01**% |

## References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

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

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

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[6] S. Zagoruyko and N. Komodakis, "Wide residual networks," *CoRR*, vol. abs/1605.07146, 2016.

[7] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?," *arXiv preprint arXiv:1805.08974*, 2018.

[8] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[9] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[10] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[11] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," *CoRR*, vol. abs/1610.02915, 2016.

[12] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *CoRR*, vol. abs/1605.07648, 2016.

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

[14] X. Gastaldi, "Shake-shake regularization," *CoRR*, vol. abs/1705.07485, 2017.