DOI: 10.15439/2019F190
ISSN 2300-5963 ACSIS, Vol. 18

Improving Real-Time Performance of U-Nets for Machine Vision in Laser Process Control

Przemysław Dolata
Wrocław University of Science and Technology, ul. Wyb. Wyspianskiego 27,
50-370 Wrocław, Poland
Email: przemyslaw.dolata@pwr.edu.pl

Jacek Reiner
Wrocław University of Science and Technology
ul. Wyb. Wyspiańskiego 27,
50-370 Wrocław, Poland
Email: jacek.reiner@pwr.edu.pl

***Abstract*—Many industrial machine vision problems, particularly real-time control of manufacturing processes such as laser cladding, require robust and fast image processing. The inherent disturbances in images acquired during these processes makes classical segmentation algorithms uncertain. Among many convolutional neural networks introduced recently to solve such difficult problems, *U-Net* balances simplicity with segmentation accuracy. However, it is too computationally intensive for usage in many real-time processing pipelines.**

In this work we present a method of identifying the most informative levels of detail in the U-Net. By only processing the image at the selected levels, we reduce the total computation time by 80%, while still preserving adequate quality of segmentation.

I. Introduction

SEGMENTATION of complex, noisy images is a core problem in many industrial applications of machine vision, especially in monitoring and control of laser additive manufacturing processes, such as laser cladding [1]. Where classical image processing algorithms cannot provide necessary robustness (against, for example, plasma emissions or powder scattering), machine-learning-based solutions are applied – recently, convolutional neural networks in particular. However, they are notoriously computationally heavy. For off-line applications this issue can be trivially solved with using more compute power, but in some on-line, real-time applications it is a critical problem. If the process state changes rapidly, any delay in its measurement degrades performance of the control algorithm.

U-Net is a well-known and proven convolutional neural network architecture for image segmentation [2]. Its distinguishing property is a highly modular, symmetric, dual-path structure. In the “down” path, which comprises blocks of max-pooling and convolution layers, features are being extracted from progressively smaller inputs. Those blocks can be thought of as observing the input at progressively smaller scales. As a result, they produce feature maps with gradually more contextual information, but less spatial resolution. On the other hand, the “up” path integrates the high-context but low-resolution feature maps with intermediate levels of low-context but high-resolution information. This allows producing highly detailed segmentations for objects of different scales.

Training the U-Net on laser cladding monitoring images is a relatively straightforward task, even with a small amount of annotated data. The baseline configuration as described by Ronneberger *et al.* [2] outputs segmentations of satisfactory quality without the need to apply any tricks or problem-specific tuning. However, the time of a single image inference, on our in-house hardware, is approximately 250ms. This is unacceptable for any on-line processing purpose – especially for real-time control.

The simplest yet very effective way to decrease processing time is to reduce the size of the input images. This might have an additional benefit of reducing the cost of data acquisition, or allowing higher processing frame rates. A more advanced method would be to downsample the images in the “down” path earlier, skipping some detail scales during inference if the information they contain does not significantly contribute to the overall segmentation quality. However, it is difficult to determine a priori, at which scale should the input be observed and at which intermediate scales should it be processed. Intuitively, this depends on the specific characteristics of a particular problem. Detecting large objects might require more context – hence, deeper “down” path – than small ones. On the other hand, segmenting objects with fuzzy boundaries might not benefit from very high-resolution features as much as when objects have very clear and detailed edges.

In this study, we present a method of determining which blocks in a U-Net are really important for correctly segmenting the objects, and which can be removed or skipped to save computation time without significant degradation of prediction quality. Our contribution is primarily a way of optimizing a neural network architecture. However, identifying the levels of detail at which the objects vary can also

This work was supported from the statutory research of Mechanical Faculty of WUST. The source material (images from laser cladding process) for the dataset preparation was supplied by National Centre for Research and Development - Project AMpHOra - Additive Manufacturing Processes and Hybrid Operations Research for Innovative Aircraft Technology Development – INNOLOT/I/6/NCBR/2013.

be seen as an important insight, helpful in better understanding of the problem.

II. Related works

The original U-Net [2] builds on the concepts of Fully Convolutional Networks [3]. While the FCN allowed using only some of the earlier layers to improve the fidelity of segmentation, U-Net's core concept is to merge even the most early blocks to capture high-resolution features. Further development on these ideas included Pyramid Scene Parsing [4] – where the input is sequentially pooled into separately processed streams and then upsampled and merged together before final prediction – and Feature Pyramid Networks [5], similar to U-Nets except that at every scale a complete segmentation is produced.

Optimization of neural network architectures was always of great interest. Early attempts such as Optimal Brain Surgeon [6] were primarily focused on improving the generalization capability of the learner. In more recent days, most architecture optimization work is focused on improving inference performance or energy efficiency [7], but there are also attempts to use these techniques to help extract classification rules [8]. The two major directions in network structure optimization are: architecture search and network pruning. The objective of architecture search is to find the optimal network structure during training, often using genetic algorithms or growing/pruning strategies [9, 10]. Network pruning focuses on removal of inactive or inefficient units from an already trained network in order to preserve its predictive power but reduce inference time [11].

There is not much research examining the influence of particular levels of detail on the object segmentation or detection quality. Chevalier *et al.* [12] studied the influence of input image resolution on classification performance, however they did not investigate the influence of deeper, highly downsampled layers. In this work, we propose a method of optimizing not only the size of the network input, but also its intermediate levels of detail as well.

III. Experiment setup

A. Scale-specialized blocks

U-Net consists of distinct “blocks”, comprising two 3x3 convolutional layers of various kernel depths, each followed by ReLU nonlinearity. From here onwards we will refer to them as simply *blocks*. Blocks are usually separated by maxpooling (in the “down” path) or upsampling and merge layers (in the “up” path). Thus, different blocks learn to extract features on different levels of detail.

Intuitively, depending on the characteristics of the problem, some of those blocks might be less useful for segmentation. This would mean that features of the data at these levels of detail are not important for a proper recognition. Blocks detecting those features would therefore waste compute power and memory. However, the problem of identifying them is not trivial.

Naively, one could envision training and comparison of multiple networks with different selection of blocks (e.g. one with 3 blocks and downsampling by a factor of 2, or 2 blocks and downsampling by 4). Such a brute-force ap-

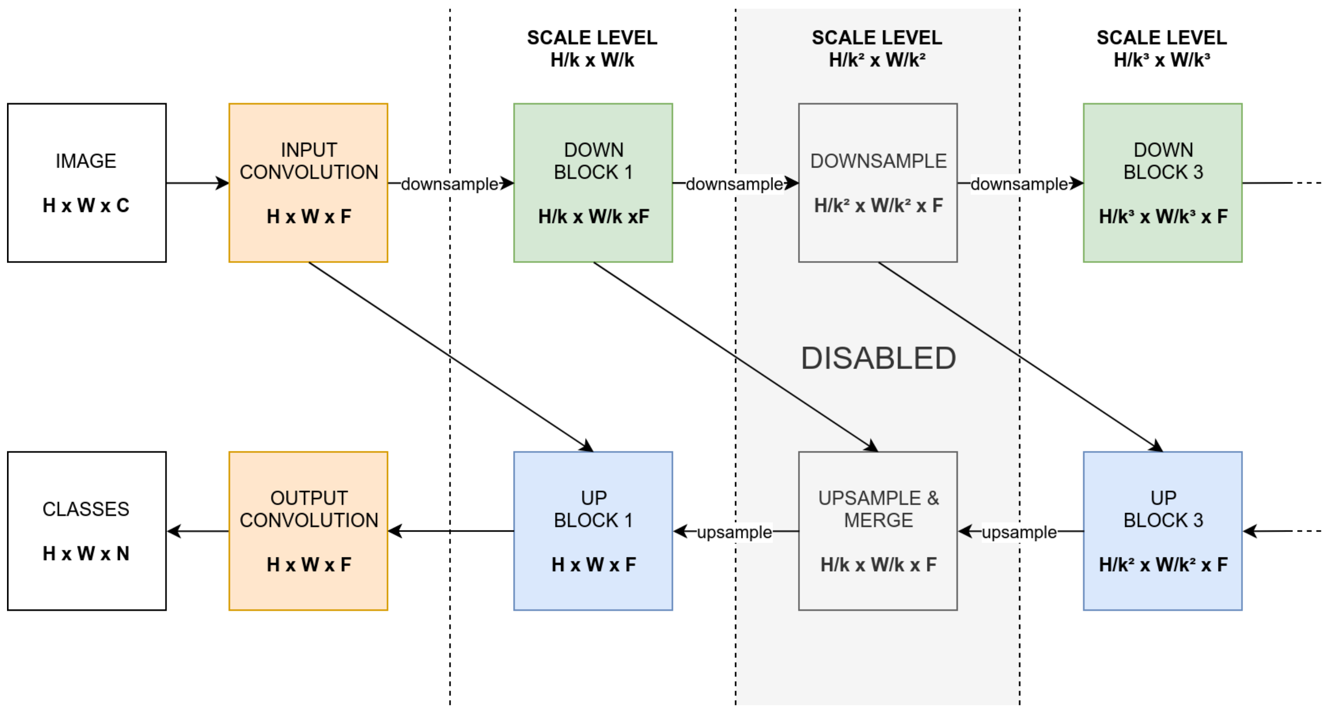
SCALE LEVEL H/k x W/k
SCALE LEVEL H/k² x W/k²
SCALE LEVEL H/k³ x W/k³
IMAGE H x W x C
INPUT CONVOLUTION H x W x F
downsample
DOWN BLOCK 1 H/k x W/k xF
downsample
DOWNSAMPLE H/k² x W/k² x F
downsample
DOWN BLOCK 3 H/k³ x W/k³ x F
DISABLED
CLASSES H x W x N
OUTPUT CONVOLUTION H x W x F
UP BLOCK 1 H x W x F
upsample
UPSAMPLE & MERGE H/k x W/k x F
upsample
UP BLOCK 3 H/k² x W/k² x F

Fig 1. Drop-path regularization algorithm adapted to the general U-Net. In this example, “down” block 1 detects features from data downsampled to some specific resolution (*scale level*), and the corresponding “up” block integrates context extracted the corresponding level. Level 2 is shown in a disabled state – the “down” block only downsamples the data, while the “up” block only upsamples and merges it, both without any other processing.

proach might be infeasible, especially if the dataset is large and the network to be optimized is very deep. Ideally, a single network would be designed and trained in such a way that individual blocks could be freely removed from it without causing a structural failure, but instead only degrading performance – in the case the block was actually useful for prediction. Identifying the useless blocks would then proceed in a manner resembling a structural kind of ablation study.

B. *Drop-path regularization*

Larsson *et al.* [13] presented a regularization algorithm, *drop-path*, that allowed them to train a very deep, multi-path network so that it behaves like an ensemble of networks. The core idea of drop-path is that if, during every training iteration, a random subset of individual paths in the network is disabled, the rest of the net will be forced to learn to still produce a correct answer. This allows the network to learn robustness against random removal of some sub-paths. Effectively, even though the network trains as a whole, every sub-path tries to become a fully capable standalone predictor itself. Larsson *et al.* report that they were able to extract even a single path of their FractalNet and it still worked almost as good as the whole.

We adapt the drop-path concept to U-Nets in order to allow them to learn robustness against removal of particular *levels of detail*. In a U-Net, the information from a particular scale is utilized twice during a single pass: once in the "down" path, where the features are extracted, and once again in the "up" path where the features are used to improve prediction resolution. Therefore, in our version of drop-path, whenever we randomly disable a "path", we actually disable both blocks processing data on a particular scale. Overview of the algorithm is shown in Fig. 1.

To allow uninterrupted flow of the data through the disabled blocks, we replace each disabled "down" block with a simple bilinear downsampling layer, and the corresponding "up" block with a similar upsampling layer. We expect the network to learn to segment the images in the absence of information from particular scales, thus allowing evaluation of their influence on segmentation performance by means of a structural ablation test.

C. Simplified U-Net

As the original architecture, U-Net does not naturally accommodate images of every size, requiring cropping and matching between "down" and "up" blocks, depending on the input size. However, as a meta-architecture it is very scalable – one can easily add or remove deeper blocks at different scales in order to capture more or less context in the data. We introduce several changes to the U-Net architecture to simplify it and make it more suitable for the drop-path regularization algorithm.

We add zero padding (1px wide border) to every convolutional layer, making each block preserve its input size. This eliminates the need for complex cropping and matching of data tensors throughout the "up" path.

We set every convolution in every block to produce the exact same number of channels (64), making every layer have exactly the same number of parameters. This is crucial in implementing drop-path: if different blocks produced outputs of different depths (as in the original U-Net), skipping a connection would necessitate a non-trivial mapping between the tensors.

Additionally, we introduce BatchNorm [14] after every convolution layer in order to stabilize the gradients. This is particularly important in the "up" path where data from two separate sources is combined.

Finally, following the practice of FPN, we change the type of connections between the "up" and "down" paths from concatenation (as originally in U-Net) to addition. This forms a residual connection between the paths, similar as described in [15]. This is not a critical change, but it reduces the number of parameters in the "up" convolutions by a factor of two, additionally speeding up the computation.

D. *Evaluation by ablation*

We expect such a U-Net, trained using drop-path regularization, to behave like an ensemble of smaller networks, each processing data at a particular level of detail. This ensemble should be robust against removal of one member – at most, this should cause the overall performance to degrade, if that member (scale path) strongly contributes to the ensemble's response. Therefore, we can measure the influence of a particular scale level by a structural ablation study. To test how important a particular level of detail is, we disable its corresponding block and evaluate the network on a validation set, measuring the change in segmentation performance. Additionally, we measure the average inference time to estimate the influence of disabling a block on wall-clock performance of the network.

In the experiment to follow, we use this evaluation strategy to reason about the data – and thus the problem at hand – in two ways.

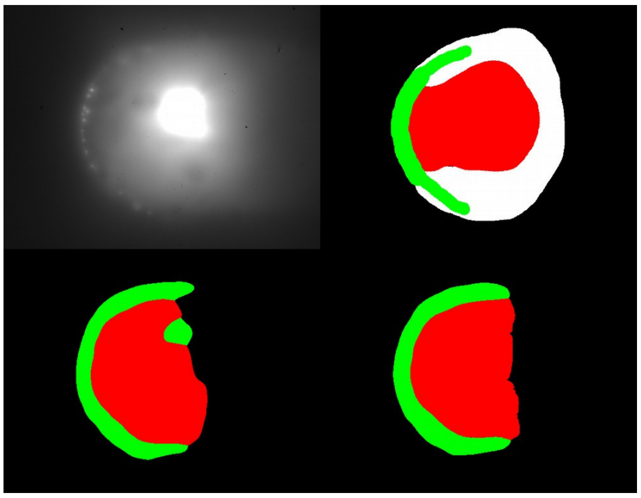
Fig 2. Example data and segmentations. Top row, left to right: source image, ground truth ("ignore" label in white); bottom row: segmentations – left: reduced model (see results, section B), right: full model.

By progressively disabling all blocks starting from the most high-resolution one, we attempt to identify the minimum scale level at which the network can observe the input images while still reliably segmenting the objects. The goal of this experiment is similar to Chevalier *et al.*, except we consider segmentation instead of classification.

By disabling subsequent blocks, starting from some given one, in different combinations, we attempt to find which of the intermediate levels of detail that extract contextual information are actually useful for a correct segmentation. This may provide an insight about how much context and on which level is really necessary, and which levels could be skipped to conserve compute time.

It is important to notice that the initial block of U-Net (on full scale) cannot be disabled – all subsequent layers require the input to be of a certain channel depth, and this first block transforms the original channel to a feature map of a common depth. This means that the initial convolutions will still be performed on the input of original resolution, constituting an approximately constant part of the computation time that cannot be trivially reduced.

IV. Results

A. Reference network

We conduct the experiments on an in-house dataset of images acquired by coaxial on-line monitoring of a laser cladding process. Images obtained during this process are inherently noisy and blurry due to plasma emissions and powder scattering. However, they carry important information about process status, encoded in the shape of the pool of metal molten by the laser beam. The dataset consists of 250 grayscale images 600x600 pixels, manually annotated in 4 classes: background, two object classes of different shape characteristics (“edge” and “pool”) and an ignore label. Data was split in training and validation sets (150 and 100 images, respectively). Example data and segmentations shown in Fig. 2.

Table I.
Learning hyperparameters

Parameter	Value
Learning rate schedule	constant 0.01
Adamax momenta	0.99, 0.999
Weight decay	0.0001
Batch size	64
Total iterations	750 000
Drop-path probability	0.25

The reference network consists of 5 levels of feature extraction blocks, at following scale levels: 600px, 300px, 150px, 75px, 25px and 5px. Each block comprises two 3x3 convolutional layers with 64 kernels, each followed by a BatchNorm layer and a ReLU nonlinearity. The network was trained using the Adamax [16] optimizer under the cross-entropy loss function. The complete training parameters are given in Table I.

Due to a small number of data samples and the need to train from scratch, heavy data augmentation routine was used in the form of elastic deformations [17] and horizontal and vertical flips. All augmentations were performed on-line in a random manner, directly before feeding data into the network. For testing, the intersection-over-union (IoU) metric was used. Results are given separately for either object class, due to different characteristics of their shapes.

Experiments were conducted in the PyTorch framework [18] using a single Nvidia RTX 2080 Ti GPU for training and an Nvidia TITAN Z for performance testing.

The reference network trained in approximately 11 hours achieving an IoU metric of 0.654 for the “edge” class and 0.809 for “pool” class. The average inference time (with gradient computation disabled) was 154.5ms, which is already approximately 38% faster than the original U-Net.

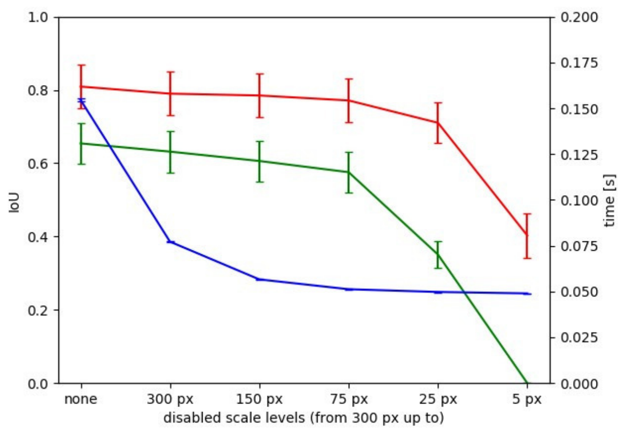

Fig 3. Results of the input size study. Segmentation performance (IoU for both classes, red and green plots) on the left axis, inference time (blue plot) on the right axis.

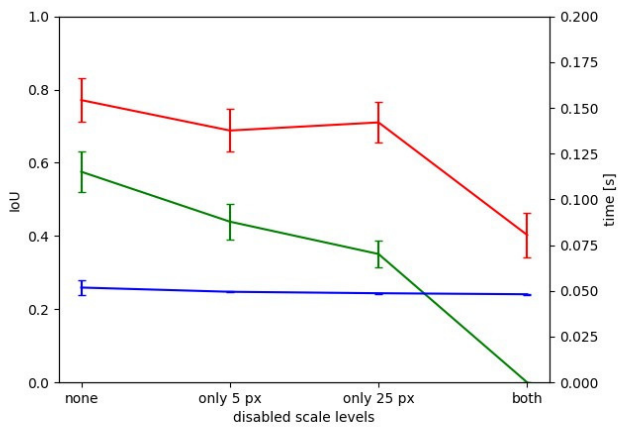

Fig 4. Results of the context levels study. Annotations the same as in Fig. 4.

B. Input size study

Results of the progressive structural ablation study show relatively small degradation of segmentation quality when disabling the early, high-resolution blocks. As illustrated in Fig. 3, removing only the first block cuts the inference time in half, down to 77.2ms while only reducing the IoU score by 0.02. Performance improvements continue to be significant up until the scale levels of 150-75px, saturating at approximately 50ms (80% reduction with respect to the original U-Net). After that point, any potential speed-up is smaller than the cost of repeatedly downsampling the data to the desired resolution, while segmentation accuracy falls rapidly.

Fig. 2 visually compares segmentations produced by a full model (bottom right) and a model reduced by disabling scale levels 300 px and 150 px (bottom left).

C. Context levels study

By disabling blocks at lower scale levels, we can determine the influence of particular context sizes. In this example we disable the first 2 levels (300px and 150px scales), fix the 75px scale as enabled and proceed to disable the remaining scale blocks (25px and 5px) in different combinations. We observe that in our case, disabling any level of context beyond the initial scale of 75px causes a rapid deterioration of segmentation quality, while providing zero practical improvement in inference time. Notice in Fig. 4 how disabling scale level 25px has a much more significant effect on the “edge” class (green) than on “pool” (red). Those most contextual blocks operate at very high relative scales (downsampling by the factors of 3 and 5, respectively) – we surmise that due to this, they learn very independent features that are critical to correct segmentation.

V. Conclusion

In this work we presented a simplified and parameterized version of U-Net and adapted the drop-path algorithm to help the network learn as an ensemble of blocks specialized to detect features at specific levels of detail. This allowed us to analyze the importance of individual blocks on the collective network using a structural ablation study. That in turn let us identify blocks that did not contribute significantly to the segmentation, enabling us to make an informed decision to remove them in order to save compute time.

We argue that if the block was deemed unimportant, this might mean that at this particular scale there are no valuable features to be extracted – the data itself contains little valuable information. Therefore, processing inputs at this size is not worth the compute time. Aside from being a useful finding for optimizing a solution, this might also be a valuable insight into the nature of the problem itself.

Acknowledgment

We wish to thank Mariusz Mrzygłód for valuable comments and discussions.

References

[1] W. Rafajłowicz, P. Jurewicz, J. Reiner, and E. Rafajłowicz, “Iterative Learning of Optimal Control for Nonlinear Processes With Applications to Laser Additive Manufacturing,” *IEEE Transactions on Control Systems Technology*, pp. 1–8, 2018. https://doi.org/10.1109/TCST.2018.2865444

[2] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Medical Image Computing and Computer-Assisted Intervention* – MICCAI 2015, 2015, pp. 234–241. http://dx.doi.org/10.1007/978-3-319-24574-4_28

[3] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440. https://doi.org/10.1109/TPAMI.2016.2572683

[4] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid Scene Parsing Network,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6230–6239.

[5] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944. https://doi.org/10.1109/CVPR.2017.106

[6] B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” in A*dvances in neural information processing systems*, 1993, pp. 164–171.

[7] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” presented at the *International Conference on Learning Representations (ICLR 2016)*, 20176

[8] H. Lu, R. Setiono, and Huan Liu, “Effective data mining using neural networks,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 957–961, 1996. https://doi.org/10.1109/69.553163

[9] C. Liu et al., “Progressive Neural Architecture Search,” arXiv:1712.00559 [cs, stat], Dec. 2017. [preprint]

[10] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, “Neural Architecture Optimization,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 7816–7827.

[11] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning Convolutional Neural Networks for Resource Efficient Inference,” presented at the *International Conference on Learning Representations (ICLR 2017)*, 2017.

[12] M. Chevalier, N. Thome, M. Cord, J. Fournier, G. Henaff, and E. Dusch, “Low resolution convolutional neural network for automatic target recognition,” in *7th International Symposium on Optronics in Defence and Security*, Paris, France, 2016.

[13] G. Larsson, M. Maire, and G. Shakhnarovich, “FractalNet: Ultra-Deep Neural Networks without Residuals,” arXiv:1605.07648 [cs], May 2016. [preprint]

[14] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” in *International Conference on Machine Learning*, 2015, pp. 448–456.

[15] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. https://doi.org/10.1109/CVPR.2016.90

[16] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[17] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” in *Seventh International Conference on Document Analysis and Recognition*, 2003. Proceedings., 2003, pp. 958–963.

[18] A. Paszke et al., “Automatic differentiation in PyTorch,” in *NIPS-W*, 2017.