How Much Deep Learning does Neural Style Transfer Really Need? An Ablation Study

Len Du
Australian National University
Len.Du@anu.edu.au

Abstract

Neural style transfer has been a "killer app" for deep learning, drawing attention from and advertising the effectiveness to both the academic and the general public. However, we have found by ablative experiments that optimizing an image in the way neural style transfer does, while the objective functions (or more precisely, the functions to transform raw images to corresponding feature maps being compared) are constructed without pretrained weights or biases, worked almost as well. We can even factor out the deepness (multiple layers of alternating linear and nonlinear transformations) altogether and have neural style transfer working to a certain extent. This raises the question how much of the the current success of deep learning in computer vision should be attributed to training, structure or simply spatially aggregating the image.

1. Introduction

Neural Style Transfer [13, 14] has been a popular topic ever since its introduction, which involves creating a third image given a pair of images, so that the generated image resembles one of the image in content while resembling the other in style. Such techniques generated huge publicity on the internet, to the extent of becoming a "killer app" to promote deep learning towards the general public, complete with literal killer apps such as DeepArt.io[32] and Prisma[33].

Ablation study or analysis has been advocated as a crucial methodology to achieve the much-needed interpretability in the field of machine learning [29], let alone deep learning. For example, a dedicated ablation study has been successfully applied to (hyper)parameters [3]. To be precise, in this work we not only remove parts of the network but also substituting them with (usually simpler) alternative constructs, which is a very natural (and sometime necessary) move on computers but arguably deviates from its root in neuroscience where there are no alternative parts to put back on.

In addition to the widely spread pair of example images in the PyTorch tutorial [23], we also use the images provided in a very comprehensive review [24] on neural style transfer methods. That said, we do not consider the many different style transfer methods in this work except the baseline from [13]. One special technicality about this paper is that there are too many images to fit in this main text if we were to display comprehensive comparsions. To mitigate this issue, we rotate the image pairs displayed in the main text, while leaving more comprehensive comparsions to the supplementary material.

2. Related Work

While we reached our results mostly through our own experiments, we have found that our findings happen to resonate with many latest topics even beyond the field of neural style transfer. We feel the need to exposit them thoroughly.

It would be no surprise if this work were compared against [16], which makes a superficially similar claim about performing style transfer (in lieu of other computer vision tasks such as texture synthesis) using an untrained network to achieve results on par with [13]. However their titular **ranVGG** actually fails to be a verbatim untrained version of VGG-19 [41]. Most importantly, by sampling "several sets of random weights connecting the l^{th} layer", reconstructing "the target image using the rectified representation", and then choosing "weights yielding the smallest loss" [16], essentially some kind of very weak layer-wise training resembling an autoencoder has slipped into the "untrained" *ranVGG*, albeit with only one image on the spot instead of the whole ImageNet [9] in advance. A "VGG with purely random weights" [16] has even been included in one of the comparisons, indicating that *ran-*

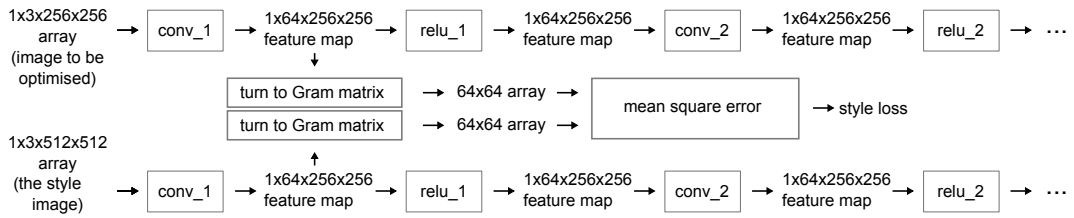


Figure 1: Pipeline to compute the style loss L_1 from the output of `conv_1` as an example

VGG is not "purely random". The algorithm to optimize for the combined image also differs from the original in [13] by adding a third term for spatial smoothness. To be fair, there is a huge difference between *ranVGG* and a typical pre-trained VGG model, but [16] has not compared two instances of style transfer using the same algorithm and the same network, the only difference being whether the weights and biases are purely random or pretrained. Moreover, it is not clear whether the pre-trained biases were kept in *ranVGG*, as there is no mentioning of biases in the paper, although both weights and biases may sometimes be referred to as weights altogether. [5] approaches texture synthesis, closely related to style transfer, from statistical physics rather than deep learning.

Another closely related work is **Deep Image Prior** [45], where untrained convolutional networks are used for denoising, super-resolution and inpainting, reaching to a similar conclusion that the structure rather than weights is more significant. Apart from that style transfer is included in its experiments, [45] does not provide a back-to-back comparison of results generated by the untrained/trained versions of precisely the same network, either. Curiously, their best-performing model resembles an autoencoder structurally. Finally, the weights are also "fitted to maximize their likelihood given a specific degraded image and a task-dependent observation model" [45], which is yet again arguably some kind of one-shot training under the hood.

Random Gaussian Weights, extensively examined in [15], are probably the most often considered type of random weights in generic discussions about random weights in neural networks, presumably because its range is $\mathbb{R}$. Even in classically-trained neural networks, **Gaussian Priors** are widely employed [48]. Random uniform weights are usually regarded as a technicality when intializing neural networks before training instead of studied as an alternative to trained weights.

Being the intersection of both random weights and discrete weights, studies on random discrete weights are scarce, except for using them for only part of the network [47] for easier training, which will be covered later in this section. On the other hand, **Discrete Weights** trained with special techniques have received significant attention [2, 51]. Some early works [31, 43] examined weights that are powers of two. In particular, **Binary Weights** in neural networks or **Binarized Neural Networks** have been extensively discussed [8, 21, 40, 35, 28, 37, 49], frequently with respect to convolutional neural networks. It is also natural to consider the possibility of zero-valued weights, resulting in **Ternary Weights** [25, 10, 55]. Another perspective on discrete weights is training a neural network as usual and then quantizing the weights, or **Quantized Neural Networks** [22, 27, 26, 18]. However, interests on discrete weights are often based on efficient execution of inference on constrained hardware rather than our standing that the discrete weights may actually be more desirable in an intrinsic way, even without considerations of computational efficiency.

Using random weights partially in a neural network to make training simpler has been examined under different names [6]. One recent work sharing our inspiration from [16, 45, 38] examines fixing the convolutional layers of a CNN under the term **Deep Weight Prior** [1]. The popular but controversial [52, 19] **Extreme Learning Machine** [20] (ELM) is a standard single layer neural network where the weights between the input layer and the hidden layer are fixed to random values, which then enables the weights between the hidden layer and the output layer to be decided by linear regression. **Random Vector Functional Link** [36] (RVFL) predated ELM, but RVFL has extra direct links from the input layer to the output layer. **Feed Forward Neural Network With Random Weights** [39] proposed in the same year does not have this extra feature and is closely resembled by the ELM except for the existence of biases and the choices of activation functions, though the former has been explicitly declared to be "not presented as an alternative learning method" [39]. The kind of **Radial Basis Function Networks** where the parameters (centers) are randomly assigned rather than trained had been proposed even earlier in [4]. In the current renaissance of deep neural networks, many multi-layer variants of

(a) Content Image (b) Style Image (c) Baseline (d) $\beta/\alpha = 10^2$ (e) Removing the content loss

Figure 2: Removing the content loss

these partially-random models have been being proposed [7, 53, 44, 46]. Their applications to computer vision, such as [34, 30, 50, 7], are no suprises either, though we are yet to see them applied to neural style transfer specifically.

Weight Agnostic Neural Networks (WANN) [12] is a way to construct neural networks by sheerly searching through different topological structures while shifting the focus from weights. Compared to a closely related topic **Neural Architecture Search** [56, 11], where combinations of smaller "component" networks searched through and weights are weights are trained with candidate combinations, WANN use a single shared weight for all connections and select architectures based on performance when given different universal weights, making the result literally "weight agnostic". WANN has been partly inspired by the fact that animals often have certain abilities immediately after birth without a chance of learning. Such innate behaviours, coupled with the fact that the genomes are unlikely to have the capacity to encode individual weights, form the grounds of criticising the current approach to training neural networks, in favour of the **Information Bottleneck** [54] between parents and children. While we may attribute any changes in the nervous system in an individual organism to learning, such changes simply do not (usually) go to the genomes in the reproductive process, even if the learned "weights" may be encoded very compactly. So the initial configuration of neurons in new-born individuals and their innate behaviours can only be attributed to natural selection rather than learning, supporting the emphasis on structure instead of weights.

3. Preliminaries

While the reader may already know it well, it is nevertheless necessary to introduce a formulation to facilitate further discussion. Simply put, the original algorithm [13] works by minimizing a value over the pixel values of an image $\mathbf{X}$ as follows,

$$\underset{\mathbf{X}}{\operatorname{argmin}}(\alpha L_{\text{content}}(\mathbf{X}) + \beta L_{\text{style}}(\mathbf{X})) \ , \tag{1}$$

which is optimized with gradient descent almost universally.

Both $L_{\text{content}}(\mathbf{X})$ and $L_{\text{style}}(\mathbf{X})$ are computed by taking the mean square error between (a linear combination of) the feature maps generated by certain layers of the CNN when taking the optimized image and the content/style image respectively, only that the feature maps are transformed into their Gram matrices to remove spatial information before taking the MSE when computing the style loss. As the feature maps are 3-dimensional arrays, the Gram matrix of a $w \times h \times c$ feature map is computed by $A^T A$ where the matrix A, a "flattened" feature map, has wh rows and c columns. Here, w and h are the width and height of the feature map, while c is the number of channels, or the number of features at each pixel location. From the dimension $c \times c$ of the resulting matrix, we can see that taking the Gram matrices truely removes any spatial information in the sense that you can reorder the pixels spatially in the feature map and still end up with the same Gram matrix. Of course, style losses and content losses can be derived from different layers of the network. For example, $L_{\text{style}}(\mathbf{X})$ could be computed using the output from the first layer of VGG-19 [41] as in Figure 1. We use the commonly used VGG-19 model. While it would be more persuasive if we included results generated with other models such as ResNet [17], doing so would require a much longer exposition, as the results cannot be easily summarised by numbers. So we have opted for concentrating on an apple-to-apple comparison limited on results generated with VGG-19. Moreover, comparison across style transfer results with different deep learning models can be left to another dedicated work.

Because the learning rate usually needs tuning, and the magnitude of the loss is not significant, we can treat $\frac{\beta}{\alpha}$ as one hyperparameter so that we have one less hyperparameter to tune. For example, using

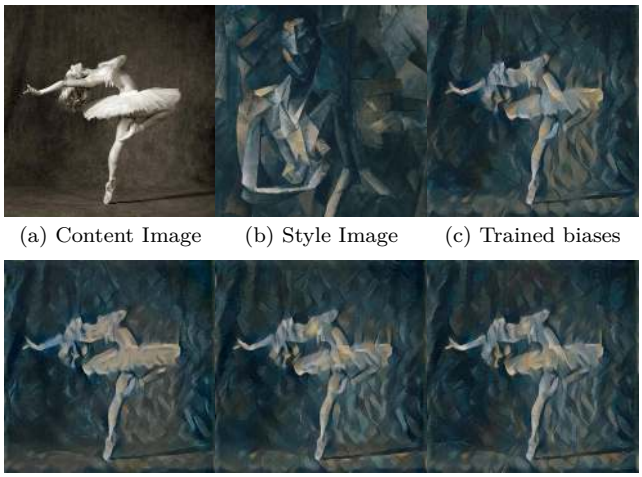
$(\alpha, \beta) = (10^1, 10^5)$ and learning rate 10^{-2} would be exactly equivalent to using $(\alpha, \beta) = (1, 10^4)$ and learning rate 10^{-1}, sans the concrete values of losses.

For more technical aspects, we use $\frac{\beta}{\alpha} = 10^6$ and a learning rate of 1, the same setup as in the well-know PyTorch tutorial [23] which we have found to be engineered well enough to provide very good results. We use the commonly used optimizer L-BFGS, but we count epochs by the larger optimizer step in which the loss may be evaluated as much as 20 times. All results are taken after 20 optimization steps, a point after which we have found the search to have converged in general. The layers chosen are `conv_1` for the content loss and `conv_1`, `conv_2`, `conv_3`, `conv_4`, `conv_5` for the style loss, a choice we have found hard to beat. To be more precise, the style loss is always computed as $L_{\text{style}} = \sum_{i=1}^{5} L_i$ where each L_i is computed from Gram matrices of feature maps after the ith convolutional layer, except for Section 6.3.

Finally, our implementation contains some measures to deal with numerical stability problems. Other implementations may have the same or similar measures as well. We describe what we use here, sheerly for reproducibility, rather than to claim any inventions in this regard. Sometimes the floating-point gradients may become `NaN`. In this case we simply replace the `NaN`s with zero. Even with `NaN` gradients suppressed, the search still explodes occasionally, depending on the random sequences used when generating filters. Since such cases are rare (a few percent), instead of lowering the learning rate, we simply reseed the program with another seed, genreate the filters again, and then restart the search, when detecting losses of floating-point values `inf` or `nan`. However, not every configuration has randomness in itself. For example, it is possible to encounter such problem when running the original configuration, which is deterministic. So we also add a 10% perturbation once we detect a last-step loss repeated exactly between two executions with different random seeds.

4. Removal of Content Loss

While in theory we need to compute the content loss, in practice we have found that as long as we make the content image the initial value, which also leads to better results than initialisation with white noise and is the preferred choice in practice, we can almost use whatever content loss function L_{content} we want, or even remove the content loss L_{content} from the objective function altogether, without a visible impact on the result (Figure 2e). This is desirable as we have one less component to experiment with. It is clear at this point that the style loss is much more interesting part of the whole neural style transfer landscape. In the rest of the paper we will use zero content loss except when showing the baselines. Even with smaller $\frac{\beta}{\alpha}$and theoretically more significant content loss, the result in Figure 2d does not show a significant difference. Note that setting $\frac{\beta}{\alpha}$ to large values is the common practice, and the ratio in Figure 2d already errs on the side of favoring contents. For example, $\frac{\beta}{\alpha} = 10^3, 10^4$ yields the best results in [13]. The role of content losses is essentially substitued by the nature of gradient descent that the solutions are gradually modified from the initial one.



(a) Content Image (b) Style Image (c) Trained biases

(d) All zero biases (e) Fixed biases (0.5) (f) Reordered biases

Figure 3: Varing the biases while keeping trained weights

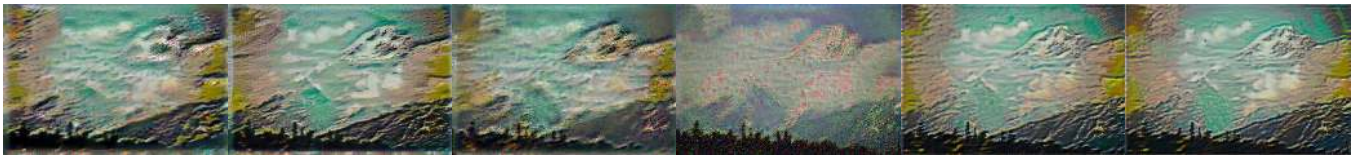
5. Removal of Learned Parameters

To compare apples to apples, we have tried resetting the weights and biases with different strategies in the VGG19 network in the first place, and then tried to figure out the simple parametric distributions resulting in best performing networks, in contrast to the alternative network structures and algorithms in [16] and [45].

5.1. (Un)importance of biases

Whether the biases were kept, zeroed, or generated from scratch were not mentioned in [16]. It might be possible that it is the bias that kept the important information. Though, our experiments quickly show that this is not the case, as in Figure 3. While simply zeroing the biases slightly weakens the result (Figure 3d), a universal positive bias as in Figure 3e could perform

(a) Content image (b) Style image (c) Baseline (d) Trained weights (e) $N(\sigma$=0.015) (trained bias) (f) $N(\sigma$=0.015) [16]



(g) $N(\sigma$=0.15) (h) $U(-0.1, 0.1)$ (i) $U(-0.5, 0.5)$ (j) $U(-1, 1)$ (k) Shuffled weights (l) Shuffled weights (trained biases)

Figure 4: Continuously or densely distributed weights

just as well as the original trained bias or random biases reproducing the original distribution (i.e. shuffled) in Figure 3f. Note that this positive value has been chosen through experiments, and larger or smaller values do not necessarily perform as well. This value perform just as good when used on other image pairs. Again, we use this universal bias 0.5 in following discussions unless stated otherwise.

5.2. Continuously or densely distributed weights

When talking about random weights, the most common distributions are *Gaussian* and *uniform*. On the other hand, the random sample with a distribution closest to that of the trained weights is probably a *random permutation* of the trained weights, which is also worth consideration. Strictly speaking, the last one may not be a continuous distribution, hence the word "densely". From the results shown in Figure 4, we can see that the quality is sensitive to the parameter of distribitions. Among the best performing distributions from the three categories, we consider $U(-0.1, 0.1)$ (Figure 4h) to be the best. We can also see that the choice of variance in Gaussian parameters in [16] was suboptimal, compared to Figure 4g The result from the best uniform distribution is obviously better than the best Gaussian we have found. The uniform distribution is even arguably better than the shuffled weights in Figure 4k, which means that the distribution of (values of) weights in the trained network may be suboptimal. We theorise that it is the weights with the largest absolute values are of real use in CNN, judging from the superiority of the uniform distribution, which inspires us to experiment discrete weights to be discussed in the next section.

5.3. Discrete weights

What we have found to be the most interesting is that we can actually use discrete weights, including binary and ternary ones. We would also expect the distribution to be symmetric around 0. Actually we experimented with asymmetric weights but they performed too bad to be included. Whether there are zeroes in the weights or how much zeroes there are among the waits is also potentially significant. It is also worth investigation whether quantizing learned weight into binary values would lead to better results than random. In binarization we assign ± 1 times some magnitude according to whether the weights are greater than the mean at the layer. The mean is favored over the median because order statistics do not work very well with automatic differentiation.

From the results in Figure 5, we can see that untrained binary weights perform as good as, if not slightly better, than the best continuous/dense distribution U(-0.1,0.1) previously found. Moreover, adding zeroes making it ternary does not help. The magnitude of binary weights are significant (best 0.1). Finer quantization does not help, supporting that only the the weights with the largest absolute values are of importance. However, weights binarized from the trained weights performs visibly better, so there is still something significant demanding training.

6. Varying Structure

Now that we have a completely program-generated model void of learning, we can start trying to modify or remove some parts from the network structure, without worrying about losing or mismatching the trained parameters.

(a) Content image (b) Style image (c) Baseline (d) Trained weights (e) $U(-0.1, 0.1)$ (f) $\{\pm 0.1\}$

(g) $\{\pm 0.1\}$ trained binarized (h) $\{\pm 0.1, 0\}$ (i) $\{\pm 0.1, 0, 0\}$ (j) $\{\pm 0.2\}$ (k) $\{\pm 0.4\}$ (l) $\{\pm 0.4, \pm 0.2, \pm 0.1, 0\}$

Figure 5: Symmetric discrete weights with fixed biases (0.5)

6.1. Removal of Structure

First of all, let us try further removing some parts of the network in rather straightforward ways, from the binary weight network with weights in $\{\pm 0.1\}$ and unified bias 0.5. The results are displayed in Figure 6. The most obvious ways would be removing the ReLU layers or the max-pooling layers. When removing pooling layers, we may also like no longer doubling the number of filters or number of channels in feature maps during the convolution layer originaly after the pooling layer. The VGG model doubles the number of filters after each max-pooling layer. Of course we could remove only some of the ReLU layers or pooling layers as well, but that would be too many combinations without clear insights, nor have we found such combinations to outperform the simpler all-or-nothing ones. As can be seen from Figure 6f and Figure 6h, the difference made by removing the ReLU layer compared to Figure 6d is very visible, while removing the pooling layer does not matter too much. The irrelevance of pooling layer has been discussed by [42]. Further removing the doubling of channels (NPND) as in Figure 6g results in even more ignorable change.

Now we can simply use one universal number of channels for all the filter maps, which may not be result in the smallest amount of computations but is conceptually simpler. As a next step we try increasing or decreasing the number of filters as shown in Figure 6i to Figure 6l. It is evident that if the number of filters are too small, we indeed lose the functionality of the network, yet increasing it improves the result with quickly diminishing returns. We also shown an example with a prime number (71) of filters in Figure 6k here, suggesting that network sizes do not have to be multiples or powers of any specific number. In the next sections we will use the NPND case with the default 64 channels as the starting point.

6.2. Alternative Convolution Kernels

All the convolution kernels used in the VGG network can be put into two obvious categories. One includes those casting the 3-dimensional (RGB) images to the first many-dimensional (64 in VGG19) feature map. The other includes those between many-dimensional feature maps. This classification sounds trivial, but the point is that we can enumerate kernels in the first category with some rules and still have resultant feature maps with around 10^2channels, while there is no obvious way to do so for the second category due to combinatorial explosion.

We have experimented with the 3 groups of special (2×2) filter configurations. Each group is derived from a set of basic filters (without consideration of channels). We show results with $a = 0.1$ following previous experience. The results are sensitive to the actual a.

1. Line filters only $\begin{bmatrix} -a & -a \\ a & a \end{bmatrix}, \begin{bmatrix} a & a \\ -a & -a \end{bmatrix}, \begin{bmatrix} a & -a \\ a & -a \end{bmatrix}, \begin{bmatrix} -a & a \\ -a & a \end{bmatrix}$.
2. In addition to 1, add $\begin{bmatrix} a & a \\ a & a \end{bmatrix}, \begin{bmatrix} -a & -a \\ -a & -a \end{bmatrix}$.
3. In addition to 2, further add $\begin{bmatrix} 0 & -a \\ a & 0 \end{bmatrix}, \begin{bmatrix} 0 & a \\ -a & 0 \end{bmatrix}$.

Then for each basic filter set, we expand the filters to RGB channels with 4 combination strategies

1. For each single-channel (2×2) filter, create three RGB $(3 \times 2 \times 2)$ filters where one works on one channel and have all zeroes in other channels.
2. For each single-channel filter, create 7 filters where a powerset of RGB channels $(2^3 - 1)$ are enabled in the

(a) Content image (b) Style image (c) Baseline (d) $\{\pm 0.1\}$ (e) No ReLU no pooling no doubling (f) No ReLU

(g) No pooling nor doubling (NPND) (h) With doubling but no pooling (i) NPND, 4 filters (j) NPND, 16 filters (k) NPND, 71 filters (l) NPND, 512 filters

Figure 6: Removal of structure

filter except the one that degenerates to working on no channels.

3. For each single-channel filter, create 8 filters where a powerset of RGB channels (2^3) are given the original weights as in the original single-channel filter, while the rest of channels are given negated values.
4. Same as the second strategy but keep the degenerated all-zero filter, resulting in 8 filters for each single-channel filter.

We have also experimented with randomly generated convolution filters of size 2×2 instead of 3×3 in the previous experiments, as the special configurations are all (2×2).

In general, most of the designed convolutional kernels, i.e. those based on basic filter sets 2 and 3, perform at a level virtually indistinguishable from the randomly generated ones. None of the designed kernels magically outperform random kernels. However, one class of the designed kernels (1) perform particularly bad, exemplified by Figure 7a, which means we do need filters like $\begin{bmatrix} a & a \\ a & a \end{bmatrix}$ for neural style transfer to work properly, which is slightly counter-intuitive. The 2×2 random kernels also work just as well as the 3×3 ones, given the same number of channels in the feature maps. We avoid cluttering this paper with too many similar images by displaying only a few representative cases in Figure 7.

6.3. Discarding Deepness

Finally we would like to try not relying on a deep neural network structure. Then what does it mean to rely on a deep neural network structure? We consider the more than one layers of non-linearity as the key trait of "deepness". Yet it is obvious that we want some kind of hierarchical processing of the image to cover large-scale features. The traditional wisdom in image processing to deal with features at different scales falls in three categories: frequency space, wavelet, and image pyramids. We only cover using Gaussian and laplacian pyramids here, due to both limit of scope and that we have not found more sophiscated yet "shallow" approaches to work better than basic pyramids. Note that Gaussian pyramids could be regarded as average-pooling layers in deep learning jargon, but they are still linear in contrast to max-pooling.

Here we have a shallow network with a convolutional layer of random binary weights from $\{\pm 0.1\}$ and bias 0.5, and then a ReLU layer, and an optional pooling layer. Then we compute $L_{\text{style}} = \sum_{i \in 1}^{5} L_i$, this time with L_i computed from pushing the ith layer of the Gaussian pyramid (1st being the original image) through the shallow network rather than taking the output of `conv_`i from the deep network. As can be seen from Figure 8, either Gaussian pyramids or Laplacian pyramids can substitute the deep structure to a certain extent. Pooling is detrimental for both types of pyramids, introducing unwanted artifacts not part of the intended "style". The simpler Gaussian pyramids work somewhat better than the Laplacian ones.

7. Discussion

It is clear that *random binarized weights* perform very well, which means current deep convolutional networks could probably be replaced by much simpler constructs. Even the necessity of *multiple layers of nonlinearity* is questionable. We also see that such networks are very sensitive to the actual *magnitude of (shared)*

(a) Basic set 1, strategy 4 (32-channel)

(b) Basic set 2, strategy 1 (18-channel)

(c) Basic set 3, strategy 3 (64-channel)

(d) 2×2 kernels, random, 64-channel

(e) 3×3 kernels, random, 64-channel

Figure 7: Alternative first-layer convolution kernels

(a) Gaussian pyramid without pooling

(b) Laplacian pyramid without pooling

(c) Gaussian pyramid with pooling

(d) Laplacian pyramid with pooling

Figure 8: Using image pyramids instead of multi-layer nonlinearities

weights and the *shared bias*. Compared to purely random configurations, training still provides some benefits, not yet replaceable by manually configuring the filters. The assumption of normal-distributed weights is no longer useful.

We suspect that an "optimal" CNN should have heavily patterned weights similar to filters in traditional computer vision, with much fewer parameters by, for example, treating the magnitue of binary weights as one parameter.

Due to the page limit we cannot include an objective and quantitative comparison of feature extraction capabilities of (part of) the original pre-trained network and our non-learning or non-deep variants with tasks other than style transfer which inherently can only be evaluated subjectively.

In the future, in addition to further ablation with style transfer, we may want to quantitatively evaluate such networks with classification problems and transfer learning without having to worry about losing or mismatching the trained parameters.

References

[1] A. Atanov, A. Ashukha, K. Struminsky, D. Vetrov, and M. Welling. The deep weight prior. In *International Conference on Learning Representations*, 2019.

[2] C. Baldassi and A. Braunstein. A max-sum algorithm for training discrete neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2015(8):P08008, 2015.

[3] A. Biedenkapp, M. Lindauer, K. Eggensperger, F. Hutter, C. Fawcett, and H. Hoos. Efficient parameter importance analysis via ablation with surrogates. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[4] D. S. Broomhead and D. Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[5] J. Bruna and S. Mallat. Multiscale sparse microcanonical models. *arXiv preprint arXiv:1801.02013*, 2018.

[6] W. Cao, X. Wang, Z. Ming, and J. Gao. A review on neural networks with random weights. *Neurocomputing*, 275:278 – 287, 2018.

[7] H. Cecotti. Deep random vector functional link network for handwritten character recognition. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3628–3633. IEEE, 2016.

[8] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc., 2015.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[10] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks*,

100:49 – 58, 2018.

[11] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[12] A. Gaier and D. Ha. Weight agnostic neural networks. 2019.

[13] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv*, Aug 2015.

[14] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2016.

[15] R. Giryes, G. Sapiro, and A. M. Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13):3444–3457, July 2016.

[16] K. He, Y. Wang, and J. Hopcroft. A powerful generative model using random weights for the deep image representation. In *Advances in Neural Information Processing Systems*, pages 631–639, 2016.

[17] 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*, pages 770–778, 2016.

[18] L. Hou, R. Zhang, and J. T. Kwok. Analysis of quantized models. In *International Conference on Learning Representations*, 2019.

[19] G. Huang. Reply to "comments on "the extreme learning machine"". *IEEE Transactions on Neural Networks*, 19(8):1495–1496, Aug 2008.

[20] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, et al. Extreme learning machine: a new learning scheme of feedforward neural networks. *Neural networks*, 2:985–990, 2004.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc., 2016.

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2018.

[23] A. Jacq. Neural transfer using pytorch. 2019.

[24] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song. Neural style transfer: A review. *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[25] F. Li, B. Zhang, and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[26] H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein. Training quantized nets: A deeper understanding. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5813–5823, USA, 2017. Curran Associates Inc.

[27] D. Lin, S. Talathi, and S. Annapureddy. Fixed point quantization of deep convolutional networks. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2849–2858, New York, New York, USA, 20–22 Jun 2016. PMLR.

[28] X. Lin, C. Zhao, and W. Pan. Towards accurate binary convolutional neural network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 345–353. Curran Associates, Inc., 2017.

[29] Z. C. Lipton and J. Steinhardt. Troubling trends in machine learning scholarship. *arXiv preprint arXiv:1807.03341*, 2018.

[30] J. Lu, J. Zhao, and F. Cao. Extended feed forward neural networks with random weights for face recognition. *Neurocomputing*, 136:96–102, 2014.

[31] M. Marchesi, G. Orlandi, F. Piazza, L. Pollonara, and A. Uncini. Multi-layer perceptrons with discrete weights. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 623–630 vol.2, June 1990.

[32] M. McFarland. This algorithm can create a new van gogh or picasso in just an hour. *WashingtonPost.com*, 2015.

[33] C. McGoogan. Prisma: The world's coolest new app taking over your instagram. *The Telegraph*, 2016.

[34] A. A. Mohammed, R. Minhas, Q. J. Wu, and M. A. Sid-Ahmed. Human face recognition based on multidimensional pca and extreme learning machine. *Pattern Recognition*, 44(10-11):2588–2597, 2011.

[35] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. Verifying properties of binarized deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[36] Y.-H. Pao, S. M. Phillips, and D. J. Sobajic. Neural-net computing and the intelligent control of systems. *International Journal of Control*, 56(2):263–289, 1992.

[37] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[38] A. M. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 1089–1096. Omnipress, 2011.

[39] W. F. Schmidt, M. A. Kraaijveld, and R. P. W. Duin. Feedforward neural networks with random weights. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4, Aug 1992.

[40] T. Simons and D.-J. Lee. A review of binarized neural networks. *Electronics*, 8(6):661, 2019.

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

[42] J. T. Springenberg, A. Dosovitskiy, T. Brox, and

M. A. Riedmiller. Striving for simplicity: The all convolutional net. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[43] C. Z. Tang and H. K. Kwan. Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Transactions on Signal Processing*, 41(8):2724–2727, Aug 1993.

[44] M. D. Tissera and M. D. McDonnell. Deep extreme learning machines: supervised autoencoding architecture for classification. *Neurocomputing*, 174:42–49, 2016.

[45] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

[46] M. Uzair, F. Shafait, B. Ghanem, and A. Mian. Representation learning with deep extreme learning machines for efficient image set classification. *Neural Computing and Applications*, 30(4):1211–1223, 2018.

[47] M. van Heeswijk and Y. Miche. Binary/ternary extreme learning machines. *Neurocomputing*, 149:187–197, 2015.

[48] M. Vladimirova, J. Verbeek, P. Mesejo, and J. Arbel. Understanding priors in bayesian neural networks at the unit level. In *International Conference on Machine Learning*, pages 6458–6467, 2019.

[49] D. Wan, F. Shen, L. Liu, F. Zhu, J. Qin, L. Shao, and H. Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[50] W. Wan, Z. Zhou, J. Zhao, and F. Cao. A novel face recognition method: Using random weight networks and quasi-singular value decomposition. *Neurocomputing*, 151:1180–1186, 2015.

[51] L. Wang, Q. Zhou, T. Jin, and H. Zhao. Feed-back neural networks with discrete weights. *Neural Computing and Applications*, 22(6):1063–1069, May 2013.

[52] L. P. Wang and C. R. Wan. Comments on" the extreme learning machine. *IEEE Transactions on Neural Networks*, 19(8):1494–1495, 2008.

[53] Y. Yang and Q. J. Wu. Multilayer extreme learning machine with subnetwork nodes for representation learning. *IEEE transactions on cybernetics*, 46(11):2570–2583, 2015.

[54] A. M. Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):1–7, 2019.

[55] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[56] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578, 2016.