RANDOMNESS IN DECONVOLUTIONAL NETWORKS FOR VISUAL REPRESENTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

To understand the inner work of deep neural networks and provide possible theoretical explanations, we study the deep representations through the untrained, random weight CNN-DCN architecture. As a convolutional AutoEncoder, CNN indicates the portion of a convolutional neural network from the input to an intermediate convolutional layer, and DCN indicates the corresponding deconvolutional portion. As compared with DCN training for pre-trained CNN, training the DCN for random-weight CNN converges more quickly and yields higher quality image reconstruction. Then, what happens for the overall random CNN-DCN? We gain intriguing results that the image can be reconstructed with good quality. To gain more insight on the intermediate random representation, we investigate the impact of network width versus depth, number of random channels, and size of random kernels on the reconstruction quality, and provide theoretical justifications on empirical observations. We further provide a fast style transfer application using the random weight CNN-DCN architecture to show the potential of our observation.

1 INTRODUCTION

Deep neural networks have achieved impressive performance on various machine learning tasks. However, our understanding of how these deep learning models operate remains limited. Providing a theoretical explanation or empirical interpretation for their success is an important research area. Existing works Arora et al. (2015; 2014); Paul & Venkatasubramanian (2014) propose mathematical models for learning architectures, however, the theoretical analysis of which fails to capture the state-of-the-art architectures. Gilbert et al. (2017); Chang et al. (2018) leverage either compressive sensing or ordinary differential equations to facilitate the understanding of CNNs. Ma et al. (2018); Hand & Voroninski (2017) deliver rigorous proofs about the invertibility of convolutional generative models. Despite these promising progress, there is no solid theoretical foundation on why the overall random CNN-DCN architecture is capable for image reconstruction. In this paper, we bridge the gap between the empirical observation and theoretical explanation of CNNs, especially the invertibility of the overall random CNN-DCN architecture.

To understand the deep representations of intermediate layers, a variety of visualization techniques have been developed in order to unveil the feature representation and hence the inner mechanism of convolutional neural networks (CNNs) Zeiler & Fergus (2014); Mahendran & Vedaldi (2015); Yosinski et al. (2015); Xu et al. (2015). In this work we propose applying randomization on deconvolutional networks (DCNs) for a systematic investigation of deep representations, and provide insights on the intrinsic properties of deep convolutional networks. We first observe that training the DCN for reconstruction, the random CNN preserves richer information in the feature space. The training on DCN converges faster for the random CNN contrasted to pre-trained CNN and yields higher quality image reconstruction. It indicates there is rich information encoded in the random features; the pre-trained CNN discards some information irrelevant for classification and encodes relevant features in a way favorable for classification but harder for reconstruction. This leads us to be curious about what happens if we feed the images to a CNN-DCN architecture where both the CNN and the DCN have random weights.

Our motivation for studying the overall random CNN-DCN architecture is threefold. First, a series of works empirically showed that a certain feature learning architecture with random weights allowed satisfactory discriminative validity on object recognition tasks Jarrett et al. (2009), and certain convolutional pooling architectures even with random weights can be inherently frequency selective and translation invariant, leading to the potential application of fast search of network architectures Saxe et al. (2011). Second, studying a complex system with random weights rather than learned determin-

istic ones may lead to a better understanding of the system even in the learned case. For example, in the field of compressed sensing, random sampling leads to breakthroughs in the understanding of the number of required measurements for a stable reconstruction of the signal Giryes et al. (2016); Gilbert et al. (2017). For highly complicated systems with nonlinear operations along the hidden layers, there are already some investigations on random deep neural networks Saxe et al. (2011); Arora et al. (2014); Ulyanov et al. (2017a). Third, as a reversible encoder-decoder architecture, deconvolution is a valuable visualization technique for studying the feature representation of deep convolutional nets. To our knowledge there is no existing work on the random deconvolutional networks in the literature. Our work on using deconvolution to study the random intermediate features of CNN provides new insights and inspires possible applications with untrained deep neural models.

Our main results and contributions are as follows. We study the overall random CNN-DCN architecture to investigate the randomness in deconvolutional networks, i.e. there is no training at all for inverting the inputs that passes their information through a random weight convolutional network. Surprisingly, the image is inverted with satisfactory quality. The geometric and photometric features of the inputs are well preserved given a sufficient number of channels. We provide empirical evidence as well as theoretical analysis on the reconstruction quality, and bound the error in terms of the number of random nonlinearities, the network architecture, the distribution of the random weights, and local similarity of the input which is high for natual images. Extensive empirical study by varying the network width, depth, or kernel size has been performed to show the effectiveness on the inversion. The CNN-DCN architecture with random weights can be very useful on texture synthesis, style transfer, image segmentation, image inpainting, etc. As an example, we illustrate how fast style transfer can be applied using random weight CNN-DCN architecture. Note that our approach can save a big amount of time and energy as we do not need to do the pre-training on deep models, and it is very flexible as we can easily try whatever nerual network architecture as we wish.

2 Related Work

Two techniques are closely related to our work, *deconvolution* and *randomization*. Deconvolution involves a CNN-DCN architecture, where CNN indicates the portion of a convolutional neural network from the input to an intermediate convolutional layer, and DCN indicates the corresponding deconvolutional network aiming to invert the intermediate features to the original images. Randomization indicates the stochastic assignment of weights to the deep neural network.

As a generative model for encoder-decoder functions, deconvolutional networks (DCNs) are commonly used for deep feature visualization. Zeiler et al. Zeiler & Fergus (2014) propose to use a multi-layered deconvolutional network Zeiler et al. (2011) to project the feature activations back to the input pixel space, and show that the features have many intuitively desirable properties such as compositionality, increasing invariance and class discrimination for deeper layers. Dosovitskiy et al. Dosovitskiy & Brox (2016) design a deconvolution variant to invert image representations learned from a pre-trained CNN, and conclude that features in higher layers preserve colors and rough contours of the images and discard information irrelevant for the classification task that the convolutional model is trained on. As there is no back propagation, their reconstruction is much quicker than the representation inverting method on gradient descent Mahendran & Vedaldi (2015).

Randomization on neural networks can be tracked back to the 1960's where the bottom-most layer of shallow networks consisted of random binary connections Block (1962). In recent years, largely motivated by the fact that "*randomization is computationally cheaper than optimization*", randomization has been resurfacing repeatedly in the machine learning literature Scardapane & Wang (2017). For optimization problems such as regression or classification, this technique is used to stochastically assign a subset of weights in a feedforward network to derive a simpler optimization problem Igelnik & Pao (1995); Rahimi & Recht (2009). Specifically, they compute a weighted sum of the inputs after passing them through a bank of arbitrary randomized nonlinearities, such that the resulting optimization task is formulated as a linear least-squares problem. Empirical comparisons as well as theoretical guarantees are provided for the approximation Rahimi & Recht (2008; 2009); Arora et al. (2014). Other related works include random kernel approximation Rahimi & Recht (2007); Sinha & Duchi (2016) and reservoir computing on random recurrent networks Lukosevicius & Jaeger (2009); Jaeger & Haas (2004).

Specifically on convolutional neural networks (CNNs), there are a few works considering randomization. Jarrett et al. Jarrett et al. (2009) observe that, on a one-layer convolutional pooling architecture, random weights perform only slightly worse than pre-trained weights. Saxe et al. Saxe et al. (2011) prove that certain convolutional pooling architectures with random weights are inherently frequency

selective and translation invariant, and argue that these properties underlie their performance. He et al. He et al. (2016) accomplish three popular visualization tasks, image inversion, texture synthesize and style transfer, using random weight CNNs. Daniely et al. Daniely et al. (2016) extend the scope from fully-connected and convolutional networks and prove that random networks induce representations which approximate the kernel space. Gilbert et al. Gilbert et al. (2017) combine compressive sensing with random-weight CNNs to investigate the CNN architectures. Dmitry et al. Ulyanov et al. (2017a) utilize randomly-initialized neural nets to finish denoising and inpainting tasks.

Motivated by the intuition that "*random net is theoretically easier to comprehend than the complicated well-trained net*", and that it may reveal the intrinsic property of the network architecture, we use randomization to explore the convolution followed by deconvolution architecture, provide theoretical analysis on empirical observations, and show its application potentials by a style transfer case study.

3 PRELIMINARIES

3.1 DECONVOLUTIONAL NETWORK ARCHITECTURE

For the network architecture, we focus on VGG16 Simonyan & Zisserman (2015) for the deconvolution. A convolutional layer is usually followed by a pooling layer, except for the last convolutional layer. For consistency, we will explore the "feature representation" after the convolutional layer but before the pooling layer.

We build a CNN-DCN architecture on the layer of the feature representation to be studied. The convolution operator of a deconvolutional layer in DCN is the same as the convolution operator in CNN, and an upsampling operator Dosovitskiy & Brox (2016) is applied in DCN to invert the corresponding pooling operator in CNN. We will focus on the representations of the convolutional layers, and Figure 1 illustrates an example of the VGG Conv[5]-DeConv[5] architecture, where Conv[5] indicates the sequential layers from Conv1 to Conv5. At the end of the DCN, a final crop layer is added to cut the output of DeConv1 to the same shape as the original images. More details are in Appendix 1.

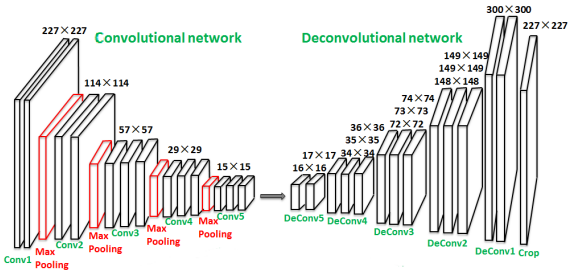


Figure 1: The CNN-DCN architecture of VGG16.

3.2 TRAINING DCN FOR RANDOM CNN

We first explore the reconstruction ability of random CNNs. We assign Gaussian random weights to the CNN part, and train the corresponding DCN to minimize the summation of the pixel-wise loss on the reconstructed images.

Training. For each intermediate layer, using the feature vectors of all training images, we train the corresponding DCN such that the summation of L_2-norm loss between the inputs and the outputs is minimized. Let $\Phi(x_i, w)$ represent the output image of the DCN, in which x_i is the i^{th} input image and w the weights of the DCN. We train the DCN to get the desired weights w^* that minimize the loss. Then for a feature vector of a certain layer, the corresponding DCN can predict an estimation of the *expected pre-image*, the average of all natural images which would have produced the current feature vector.

$$w^* = \arg\min_w L = \arg\min_w \sum_i (\Phi(x_i, w) - x_i)^2 \tag{1}$$

Specifically, we initialize the DCN by the "MSRA" method He et al. (2015) based on a modified Caffe Jia et al. (2014); Dosovitskiy & Brox (2016). We use the training set of ImageNet Deng et al. (2009) and the Adam Kingma & Ba (2015) optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ with mini-batch size 32. The initial learning rate is set to 0.0001 and the learning rate gradually decays by the "multistep"

training. The weight decay is set to 0.0004 to avoid overfitting. The maximum number of iterations is set at 200,000 empirically.

We also consider another network architecture, AlexNet Krizhevsky et al. (2012). For the random weights, we try several Gaussian distributions with zero mean and various variance. We also try several other types of random distributions, Uniform, Logistic, Laplace, to have a sound exploration. See more details and comparisons in Appendix 2.

In the following, we use CDk to represent a Conv[k]-DeConv[k] architecture. Take the VGG CD2 for elaboration, the loss curves during the training process are shown in Figure 10, which compares VGG and AlexNet on random as well as pre-trained weights. Here Conv2_Pretrained or Conv2_Random indicates whether the CNN is pre-trained or with random weights. We see that the training of DCN for reconstruction converges much quicker on random CNN and yields slightly lower loss. It indicates that by pre-training for classification, CNN encodes relevant features of the input image in a way favorable for classification but harder for reconstruction. And VGG has a much lower reconstruction loss than AlexNet.

Reconstruction. We take 5,000 samples from the training set and validation set respectively from ImageNet, and compare their average reconstruction loss. The statistics are shown in Figure 11 ("Pre-trained net" represents pre-trained CNN while "random net" represents random CNN when we train the corresponding DCN for reconstruction). We see the pre-trained CNN and random CNN both have good generalization ability; a random VGG yields much less loss than the pre-trained VGG for the deconvolution reconstruction; for representations of deeper layers, the inverting loss increases significantly for pre-trained VGG but grows slowly for random VGG. The results indicate that the random CNN encodes much richer information of the original images; the pre-trained CNN discards information not crucial for classification, especially on deeper layers, leading to a better classifier but a harder reconstruction task.

Figure 4 shows reconstructions of various layers of the random VGG on images outside the training set. The reconstruction quality decays for intermediate representations of deeper layers. The VGG structure with random weights yields accurate reconstruction, even on CD5, which involves 26 convolution layers and 4 pooling layers. In Appendix 2, we also see that the reconstruction quality on VGG based deconvolution is better than that on AlexNet based deconvolution.

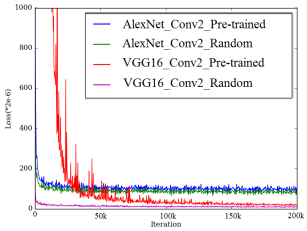


Figure 2: Training curve.

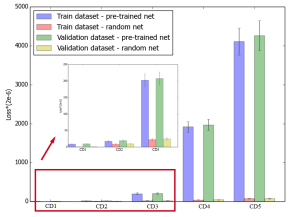


Figure 3: Reconstruction loss.

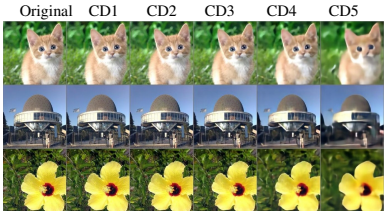


Figure 4: Reconstructions for CDk.

4 Empirical Studies on Random CNN-DCN

The above results inspire us to further explore what happens if both the CNN and DCN are of random weights. In this section we consider the reconstructions on purely random VGG CNN-DCN architecture (denoted by rrVGG for brevity), and find that the images can still be reconstructed with satisfactory quality! In other words, the CNN randomly extracts the image features and passes them to the DCN, then in an unsupervised manner the DCN reconstructs the input image by random feature extraction! Such intriguing results show that the overall random CNN-DCN architecture substantially contributes to the geometric and photometric invariance for the image inversion.

In the following, we will systematically explore the reconstruction ability of the rrVGG architecture with ReLU nonlinearity. We found that the network depth has a bigger impact than the network width, and the reconstruction quality decays with deeper layers; with plenty number of channels, an increasing number of random channels promotes the reconstruction quality; and the reconstruction quality decays with a larger kernel size.

For evaluation, we use the structural similarity (SSIM) index Wang et al. (2004), which is accurate by considering the correlation and dependency of local spatially close pixels, and consistent to the perceptron of human eyes. To remove the discrepancy on colors, we transform the inputs and outputs

in grey-scale, and in case of negative SSIM value, we invert the luminosity of the grayscale image for calculation, so the value is in $[0, 1]$. A higher value indicates a higher similarity on the images.

4.1 ON NETWORK DEPTH/WIDTH

We first explore the impact of network depth and network width for the random reconstruction, using a cat image outside the training data as an example. The weights are random in $N(0, 0.1)$ [1].

We first study the reconstruction quality for different convolutional layers, as in Figure 5. Though there is no training at all, DCN can still perceive geometric positions and contours for CD1 to CD3. The deeper the random representations are, the coarser the reconstructed image is. We can still perceive a very rough contour for the random CD4 architecture, which is already 10 layers deep. Our follow-up theoretical analysis will show that depth does affect the results, as it affects the size of receptive fields.

In Figure 6, we build a Conv1-DeConv1 (CD1) architecture with different dimensions (width) using the actual width of VGG Conv1 to Conv5 for CD1 respectively. We see that the smaller the dimension (width) is, the coarser the image is.

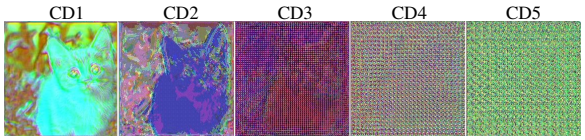


Figure 5: Reconstruction images for the rrVGG architecture(Zoom in for details).

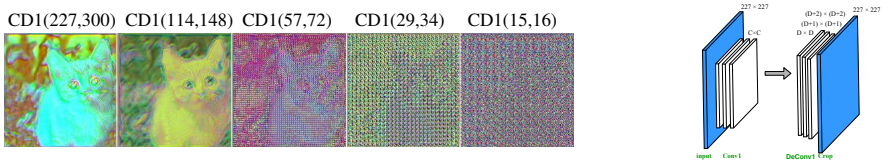


Figure 6: Left: Reconstruction images for the CD1 architecture with various network width. Right: Architecture of CD1(C,D). (Zoom in for details)

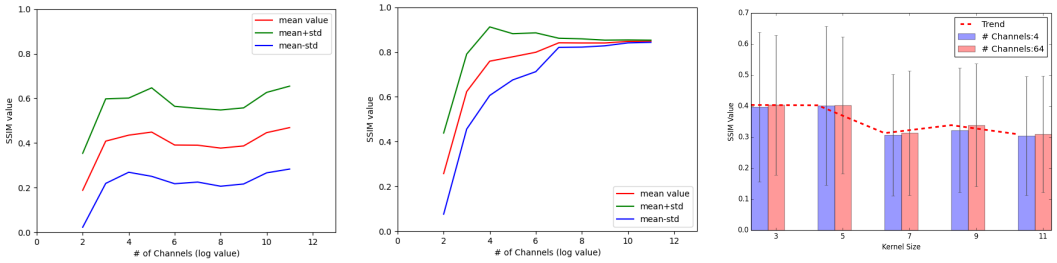
4.2 ON NUMBER OF RANDOM CHANNELS

We investigate the reconstruction quality on the number of random channels using the rrVGG CD1 (Conv1-DeConv1) architecture. For simplicity, for each network instance we use the same number of channels in all layers except the output layer. We vary the number of random channels from 4, 8 up to 2048, and for each number of channels, we generate 30 rrVGG Conv1-DeConv1 networks and all random weights are in $N(0, 0.1)$ distribution. For input images we randomly pick 50 samples from the ImageNet validation set.

To reduce occasionality on the reconstruction, we transform the inputs and outputs in grey-scale and calculate the average SSIM value on each network, then we do statistics (mean and standard deviation) on the 30 average values. Figure 7 shows the trends on SSIM when the number of channels increases, (a) is for the original rrVGG network and (b) is for a variant of rrVGG network. The variant of rrVGG is almost the same as the original network except that the last convolutional layer is replaced by an average layer, which calculates the average over all the channels of the feature maps next to the last layer. We see that the increasing number of random channels promotes the reconstruction quality. Similar in spirit to the random forest method, different channels randomly and independently extract some feature from the previous layer. With sufficient number of random channels we may encode and transform all information to the next layer. In the next section, we will prove for the variant convolutional network, when the width of the random neural network goes to infinity, the output will converge to a fixed image close to the original image.

In Figure 8, we pick some input images, and show the corresponding output images closest to the mean SSIM value for various number of channels. We transform the randomly-generated colored image to grey-scale image, for the ease of comparing the structural similarity. The SSIM value is on the top of each output image. The increasing number of channels promotes the random reconstruction

[1] In Appendix 2 Figure 1(b), we show the final reconstruction loss is similar if the variance of Gaussian distribution is small enough.



(a) (b) (c)

Figure 7: Statistics on SSIM for rrVGG Conv1-DeConv1 (a,b) and for rrVGG Conv1_1-DeConv1_1 (c).

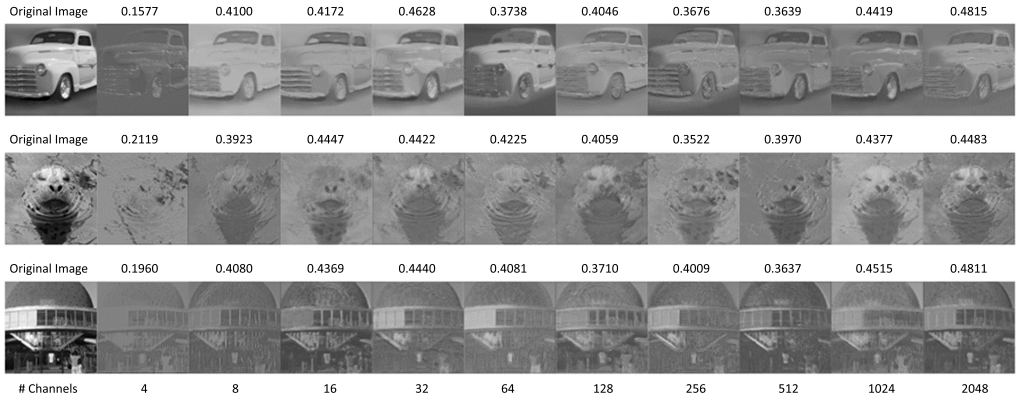


Figure 8: Reconstructions on rrVGG Conv1-DeConv1 networks (Evaluated on SSIM). Note that we transform the reconstructed colored-image to grey-scale for the ease of comparing the structural similarity.

quality. To show how the reconstruction quality decays with deeper convolutional layers, we also do experiments on the rrVGG CD2 architecture, and the quality decays by about a half as evaluated by SSIM.

4.3 On kernel size

We expect that the reconstruction quality decays with larger kernel size, as a large kernel size can not consider the local visual feature of the input. In the extreme case when the kernel size equals the image dimension, the convolution operator actually combines all pixel values of the input to an output pixel using random weights. We use the rrVGG Conv1_1 DeConv1_1 architecture, which simply contains two convolutional operators. The random weights are in $N(0, 0.1)$ distribution. For each kernel size, we randomly generate 30 networks for the reconstruction on 50 sample images as selected above. The results verifies our assumption, as in Figure 7(c).

4.4 Style transfer application

To show how our observation can be used for application, we provide a potential application using random CNN-DCN - *Style Transfer with rrVGG*. By choosing the suitable number of filters, the rrVGG CD1 architecture can achieve high-quality reconstruction. Besides, these reconstructions can also bring slight differences such as the background color and texture, which is suited to exploring more interesting style transfer results. And multiple $rrVGG$ models can be efficiently acquired without training. Recent work Gatys et al. (2016); Johnson et al. (2016); Ulyanov et al. (2017b); Huang & Belongie (2017) on style transfer employing the CNN has ignited massive research interest, while our work exhibits a novel direction of generating diverse stylized images given a single style image, as shown in Figure 17. More details are in Appendix 4.

5 Theoretical Results on Random CNN-DCN

In this section, we provide theoretical analysis to explain the empirical results. We will show that a slight variant of the random CNN architecture has the ability to reconstruct the input image. We also investigate how depth and width of the network will affect the reconstruction ability. Intuitively,

Figure 9: Style transfer from several rrVGG models. Each model has the same architecture but different random weights.

as the depth of the network increases, the receptive field of each output image pixel becomes larger, which makes the reconstruction harder whereas the width of the network, or equivalently, the number of channels, gives more basis or ways to reconstruct the original input. We theoretically show that the reconstruction ability of a random convolutional neural network rises when the number of channels in each layer increases and drops when the depth of the network increases. Note that DCN is also a kind of CNN with up-sampling layers, so our result can be directly applied to the CNN-DCN architecture.

For the following part, we will first show the convergence of the output image when the width of the network goes to infinity. Many researchers have worked on the infinite width fully connected networks. For instance, Williams (1996); Lee et al. (2017) focus on its relationship with Gaussian Process. They show the exact equivalence between infinitely wide deep networks and Gaussian Processes. Our work focuses on the random convolutional neural network without any training, which is different from the above-mentioned works and is in accordance with our previous empirical analysis. Then, we show the difference between the real output and the convergence value as a function of the width. Finally, we give an upper bound on the angle between the input and the convergence value. Thus, we can bound the reconstruction error.

Notations: We use $A_{:,j}$ to denote the j^{th} column vector of matrix A and $\|\mathbf{x}\|$ to denote the l_2-norm of vector $\mathbf{x}$. Let L be the number of layers in the neural network and $X^{(i)} \in \mathbb{R}^{N_i \times d_i}$ be the feature maps in the i^{th} layer, where N_i is the number of channels and d_i is the dimension of a single channel feature map (i.e. the width of the map times its height). $X = X^{(0)}$ is the input image and $\mathbf{f} = X^{(L)}$ is the output image. $w^{(i,j)}$, a row vector, is the j^{th} convolutional filter of the i^{th} layer if it is a convolutional layer. We use $ReLU(\mathbf{x}) = \max(\mathbf{x}, 0)$ as the activation function in the following analysis.

Definition 1. Random CNN architecture To make it possible for property proof, this structure is different from the classic CNN structure in the following three points:
1) Different filters in the same layer are i.i.d. random vectors and filters in different layers are independent. The probability density function of each filter is isotropic. Let $k_m^{(i)} = \frac{1}{2}\mathbb{E}|w_1^{(i,j)}|^m$ and $K_m^{(i)} = \frac{k_{2m}^{(i)} - (k_m^{(i)})^2}{(k_m^{(i)})^2}$. Suppose $k_1^{(i)}, k_2^{(i)}, k_4^{(i)}$ all exist.
2) The last layer is the arithmetic mean of the channels of the previous layer, not the weighted combination.
3) Except for $X^{(L-1)}$, each layer of convolutional feature maps are normalized by a factor of $\frac{1}{\sqrt{N_i}}$, where N_i is the number of channels of this layer.

5.1 Convergence

From the previous experiments, we see that when the number of channels increases, the quality of the output image improves. Here we prove that when the number of channels goes to infinity, the output will actually converge. Each pixel in the final convergence value is a constant times the weighted norm of its receptive field. Formally, we state our main results for the convergence value of the random CNN as follows.

Theorem 1. ***(Convergence Value)*** *Suppose all the pooling layers use l_2-norm pooling. When the number of filters in each layer of a random CNN goes to infinity, the output $\mathbf{f}$ corresponding to a fixed input will converge to a fixed image $\mathbf{f}^*$ with probability 1, where $\mathbf{f}^* = k\mathbf{z}^*$ and k is a constant only related to the CNN architecture and the distribution of random filters and $\mathbf{z}_i^* = \sqrt{\sum_{l \in \mathcal{R}_i} n_{(l,i)} \|X_{:,l}\|^2}$, where $\mathcal{R}_i$ is the index set of the receptive field of $\mathbf{z}_i^*$ and $n_{(l,i)}$ is the number of routes from the l^{th} pixel of a single channel of the input image to the i^{th} output pixel.*

The proof of Theorem 1 is in Appendix 3, Theorem 4. Here for the pooling layer, instead of average pooling, which calculates the arithmetic mean, we use l_2-norm pooling which calculates the norm

of the values in a patch. Intuitively, if most pixels of the input image are similar to their adjacent pixels, the above two pooling methods should have similar outputs. See details for average pooling in Appendix 3.

5.2 VARIANCE

Now we consider the case of a finite number of channels in each layer. We mainly focus on the difference between the real output and the convergence value as a function of the number of channels. We prove that for our random CNN architecture, as the number of channels increases, with high probability, the angle between the real output and the convergence value becomes smaller, which is in accordance with the variant rrVGG experiment results shown in previous section.

Theorem 2. ***(Multilayer Variance)*** *Suppose all the pooling layers use l_2-norm pooling. For a random CNN with L layers and N_i filters in the i^{th} layer, let Θ denote the angle between the output $\mathbf{f}$ and the convergence value $\mathbf{f}^*$, suppose that there is at most one route from an arbitrary input pixel to an arbitrary output pixel for simplicity, then with probability $1-\delta$,*

$$\sin\Theta \leq \sqrt{\frac{L-1}{\overline{N}\delta}} + \sqrt{(L-2)\sqrt{\frac{L-1}{\overline{N}\delta}}\prod_{i=0}^{L-2}\lambda_i},$$

where $\lambda_i = \frac{1}{\sqrt{1-\frac{\|\epsilon^{(i)}((\mathbf{z}^{*(i)})^2)\|^2}{\|(\mathbf{z}^{*(i)})^2\|^2}}}$ *and* $\frac{1}{\overline{N}} = \frac{1}{L-1}(\frac{K_1^{(L-2)}}{N_{L-1}} + \sum_{i=1}^{L-2}\frac{K_2^{(i-1)}}{N_i})$.

Here, $\epsilon^{(i)}(\mathbf{x})$ actually measures the local similarity of $\mathbf{x}$. The full definition of $\epsilon^{(i)}(\mathbf{x})$ and the proof of this theorem is in Appendix 3.

5.3 DIFFERENCE ON THE CONVERGENCE VALUE AND THE INPUT

Finally, we focus on how well our random CNN architecture can reconstruct the original input image. From Theorem 2, we know that with high probability, the angle between the output of a random CNN with finite channels and the convergence value will be upper-bounded. Therefore, to evaluate the performance of reconstruction, we focus on the difference between the convergence value and the input image. We will show that if the input is an image whose pixels are similar to their adjacent pixels, then the angle between the input image X and the convergence value of the output image will be small. To show the essence more clearly, we state our result for a two-layer random CNN and provide the multi-layer one in Appendix 3, which needs more complicated techniques but has the same insight as the two-layer one.

Theorem 3. *For a two-layer random CNN, suppose each layer has a zero-padding scheme to keep the output dimension equal to the dimension of the original input. The kernel size is r and stride is 1. The input image is $X \in \mathbb{R}^{d_0}$, which has only one channel, whose entries are all positive. $\epsilon_t = X_t - \overline{X_t}$ means the difference between one pixel X_t and the mean of the r-sized image patch whose center is X_t. Let Φ be the angle between the input image X and the convergence value of the output image, we have* $\cos\Phi \geq 1 - \frac{1}{M}\sum_t \epsilon_t X_t$, *where* $M = \sum_t X_t^2$.

The full proof of Theorem 3 is in Appendix 3. Note that when the kernel size r increases, ϵ_t will become larger as an image only has local similarity, so that the lower bound of the cosine value becomes worse, which explains the empirical results in previous section.

6 CONCLUSION

In this work, we introduce a novel investigation on deep random representations through the convolution-deconvolution architecture, which to our knowledge is the first study on the randomness of deconvolutional networks in the literature. We extensively explore the potential of randomness for image reconstruction on deep neural networks, and found that images can be reconstructed with satisfactory quality when there are a sufficient number of channels. Extensive investigations have been performed to show the effectiveness of the reconstruction. We also provide theoretical analysis that a slight variant of the random CNN architecture has the ability to reconstruct the input image, and the output converges to the input image when the width of the network, i.e. number of channels, goes to infinity. We also bound the reconstruction error between the input and the convergence value as a function of the network width and depth.

REFERENCES

Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *ICML*, pp. 584–592, 2014.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. Why are deep nets reversible: A simple theory, with implications for training. *arXiv preprint arXiv:1511.05653*, 2015.

H.D. Block. Perceptron: A model for brain functioning. *Review of Modern Physics*, 34:123–135, 1962.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua. Stylebank: An explicit representation for neural image style transfer. In *Proc. CVPR*, 2017.

Tian Qi Chen and Mark Schmidt. Fast patch-based style transfer of arbitrary style. *arXiv preprint arXiv:1612.04337*, 2016.

Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *NIPS*, pp. 2253–2261, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255, 2009.

Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *CVPR*, pp. 4829–4837, 2016.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pp. 2414–2423. IEEE, 2016.

Anna C Gilbert, Yi Zhang, Kibok Lee, Yuting Zhang, and Honglak Lee. Towards understanding the invertibility of convolutional neural networks. *arXiv preprint arXiv:1705.08664*, 2017.

Raja Giryes, Guillermo Sapiro, and Alexander M. Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Trans. Signal Processing*, 64(13): 3444–3457, 2016.

Paul Hand and Vladislav Voroninski. Global guarantees for enforcing deep generative priors by empirical risk. *arXiv preprint arXiv:1705.07576*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pp. 1026–1034, 2015.

Kun He, Yan Wang, and John Hopcroft. A powerful generative model using random weights for the deep image representation. In *NIPS*, pp. 631–639, 2016.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR, abs/1703.06868*, 2017.

Boris Igelnik and Yoh-Han Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Trans. Neural Networks*, 6(6):1320–1329, 1995.

H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication, 2004.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, pp. 2146–2153, 2009.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM MM*, pp. 675–678, 2014.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pp. 694–711. Springer, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pp. 1097–1105, 2012.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *CoRR*, abs/1711.00165, 2017.

Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer. *arXiv preprint arXiv:1701.01036*, 2017.

Mantas Lukosevicius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

Fangchang Ma, Ulas Ayaz, and Sertac Karaman. Invertibility of convolutional generative networks from partial measurements. In *Advances in Neural Information Processing Systems*, pp. 9628–9637, 2018.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, pp. 5188–5196, 2015.

Arnab Paul and Suresh Venkatasubramanian. Why does deep learning work?-a perspective from group theory. *arXiv preprint arXiv:1412.6621*, 2014.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, pp. 1177–1184, 2007.

Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *Allerton Conference on Communication, Control, and Computing*, pp. 555–561, 2008.

Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *NIPS*, pp. 1313–1320, 2009.

Andrew Saxe, Pang W Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Y Ng. On random weights and unsupervised feature learning. In *ICML*, pp. 1089–1096, 2011.

Simone Scardapane and Dianhui Wang. Randomness in neural networks: an overview. *WIREs: Data Mining and Knowledge Discovery*, 7(2), 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Aman Sinha and John C. Duchi. Learning kernels with random features. In *NIPS*, pp. 1298–1306, 2016.

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, pp. 1349–1357, 2016.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. *arXiv preprint arXiv:1711.10925*, 2017a.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proc. CVPR*, 2017b.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Processing*, 13(4):600–612, 2004.

Christopher K. I. Williams. Computing with infinite networks. In *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pp. 295–301, 1996.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pp. 2048–2057, 2015.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, pp. 818–833, 2014.

Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV*, pp. 2018–2025, 2011.

A APPENDIX

A.1 NETWORK ARCHITECTURE AND INITIAL WEIGHTS

A.1.1 DECONVOLUTIONAL NETWORK ARCHITECTURE

For the network architecture, we consider two typical CNNs for the deconvolution, VGG16 Simonyan & Zisserman (2015) and AlexNet Krizhevsky et al. (2012). A convolutional layer is usually followed by a pooling layer, except for the last convolutional layer, Conv5. For consistency, we will explore the output after the convolutional layer but before the pooling layer. In what follows, "feature representation" or "image representation" denotes the feature vectors after the linear convolutional operator and the nonlinear activation operator but before the pooling operator for dimension reduction.

We build a CNN-DCN architecture on the layer of feature representation to be studied. The convolution operator of a deconvolutional layer in DCN is the same as the convolution operator in CNN, and an upsampling operator is applied in DCN to invert the corresponding pooling operator in CNN, as designed in Dosovitskiy & Brox (2016). We will focus on the representations of the convolutional layers, since Dosovitskiy et al. Dosovitskiy & Brox (2016) build DCNs for each layer of the pre-trained AlexNet and find that the predicted image from the fully connected layers becomes very vague. For the activation operator, we apply the leaky ReLU nonlinearity with slope 0.2, that is, $r(x) = x$ if $x \geq 0$ and otherwise $r(x) = 0.2x$. At the end of the DCN, a final Crop layer is added to cut the output of DeConv1 to the same shape as the original images.

We build deconvolutional networks on both VGG16 and AlexNet, and most importantly, we focus on the random features of the CNN structure when training the corresponding DCN. Then we do no training for deconvolution and explore the properties of the purely random CNN-DCN architecture on VGG16.

A.1.2 RANDOM DISTRIBUTIONS

For the random weights assigned to CNN or DCN, we try several Gaussian distributions with zero mean and various variance to see if they have different impact on the DCN reconstruction. Subsequent comparison shows that a small variance around 0.015 yields minimal inverting loss. We also try several other types of random distributions, Uniform, Logistic, Laplace, to study their impact.

- The Uniform distribution is in [-0.04, 0.04), such that the interval equals $[\mu - 3\delta, \mu + 3\delta]$ where $\mu = 0$ and $\delta = 0.015$ are parameters for Gaussian distribution.
- The Logistic distribution is 0-mean and 0.015-scale of decay. It resembles the normal distribution in shape but has heavier tails.
- The Laplace distribution is with 0 mean and $2 * \lambda^2$ variance ($\lambda = 0.015$), which puts more probability density at 0 to encourage sparsity.

A.2 DETAILS ON TRAINING DCN FOR RANDOM CNN

A.2.1 TRAINING METHOD FOR DCN

For each intermediate layer, using the feature vectors of all training images, we train the corresponding DCN such that the summation of L_2-norm loss between the inputs and the outputs is minimized. Let $\Phi(x_i, w)$ represent the output image of the DCN, in which x_i is the input of the i^{th} image and w is the weights of the DCN. We train the DCN to get the desired weights w^* that minimize the loss. Then for a feature vector of a certain layer, the corresponding DCN can predict an estimation of the *expected pre-image*, the average of all natural images which would have produced the given feature vector.

$$w^* = \arg\min_w L = \arg\min_w \sum_i (\Phi(x_i, w) - x_i)^2 \tag{2}$$

Specifically, we initialize the DCN by the "MSRA" method He et al. (2015) based on a modified Caffe Jia et al. (2014); Dosovitskiy & Brox (2016). We use the training set of ImageNet Deng et al. (2009) and the Adam Kingma & Ba (2015) optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ with mini-batch size 32. The initial learning rate is set to 0.0001 and the learning rate gradually decays by the "multistep"

training. The weight decay is set to 0.0004 to avoid overfitting. The maximum number of iterations is set at 200,000 empirically.

A.2.2 RESULTS FOR INVERTING THE REPRESENTATION

Training. We observe similar results for the training loss in different layers. Take the Conv2-DeConv2 architecture for elaboration, the loss curves during the training process are shown in Figure 10. Figure 10(a) compares VGG and AlexNet on random as well as pre-trained weights. The training for reconstruction converges much quicker on random CNN and yields slightly lower loss, and this trend is more apparent on VGG. It indicates that by pre-training for classification, CNN encodes relevant features of the input image in a way favorable for classification but harder for reconstruction. Also, VGG yields much lower inverting loss as compared with AlexNet. Figure 10(b) shows that random filters of different small-variance Gaussian distributions on CNN affect the initial training loss, but the loss eventually converges to the same magnitude. (The loss curve of $N(0, 1)$ is not included as the loss is much larger even after the converge.) Figure 10(c) shows that the four different random distributions with appropriate parameters acquire similar reconstruction loss.

Generalization. We take 5000 samples from the training set and validation set respectively from ImageNet, and compare their average reconstruction loss. The statistics is as shown in Figure 11, where CDk represents a Conv[k]-DeConv[k] architecture. Figure 11(a) shows that the VGG architecture is good in generalization for the reconstruction, and random VGG yields much less loss than pre-trained VGG. For representations of deeper layers, the inverting loss increases significantly for pre-trained VGG but grows slowly for random VGG. This means that in deeper layers, the pre-trained VGG discards much more information that is not crucial for classification, leading to a better classifier but a harder reconstruction task. Figure 11(b) compares VGG and AlexNet on the CD3 architecture. It shows that the reconstruction quality on random compares favourably against that on pre-trained in VGG.

Reconstruction. Figure 12 shows reconstructions from various layers of random VGG and random AlexNet, denoted by rwVGG and rwAlexNet respectively. [2] On both rwVGG and rwAlexNet, the reconstruction quality decays for representations of deeper layers. The rwVGG structure yields more accurate reconstruction, even on Conv5, which involves 26 convolution operations and 4 max pooling operations.

Figure 13 shows reconstructions from a cat example image for various distributions of rwVGG CD2. Except for $N(0, 1)$, the reconstruction quality is indistinguishable by naked eyes. It shows that different random distributions work well when we set the random weights relatively sparse.

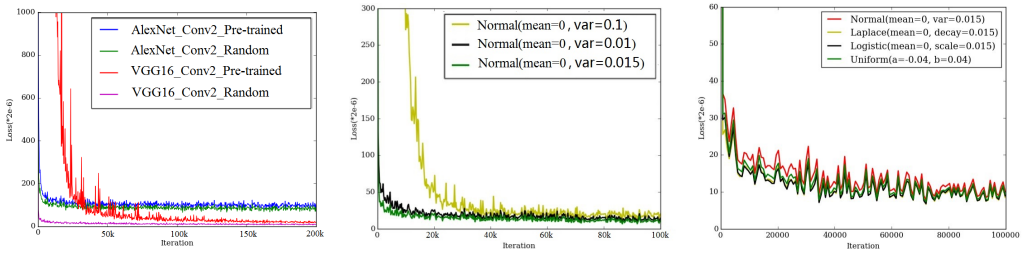
In a nutshell, it is interesting that random CNN can speed up the training process of the DCN on both VGG and AlexNet, obtain higher reconstruction quality and generalize well for other inputs. Regarding weights in the convolutional part as a feature encoding of the original image, then the deconvolutional part can decode from the feature representations encoded by various methods. The fact that the random encoding of CNN is easier to be decoded indicates that the training for classification moves the image features of different categories into different manifolds that are moving further apart. Also, it may discard information irrelevant for the classification. The pre-trained CNN benefits the classification but is adverse to the reconstruction.

A.3 PROOF OF THEORIES ON RANDOM CONVOLUTION

For completeness, we repeat the notations and the definition of random CNN architecture.

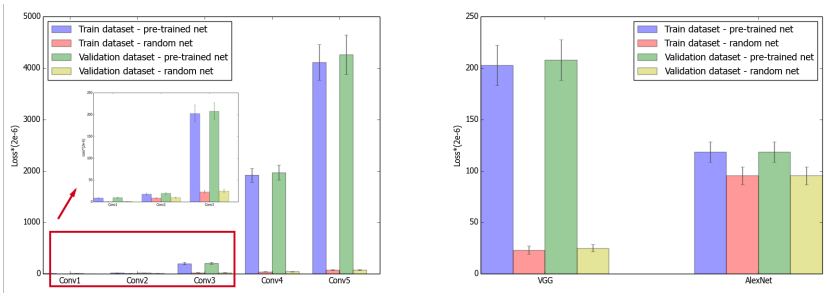
Notations: We use $A_{:,j}$ to denote the j^{th} column vector of matrix A and use A_{ij} to denote its entry. Let $\mathbf{x}_i$ be the i^{th} entry of vector $\mathbf{x}$. Let L be the number of layers in the neural network and $X^{(i)} \in \mathbb{R}^{N_i \times d_i}$ be the feature maps in the i^{th} layer, where N_i is the number of channels and d_i is the dimension of a single channel feature map (i.e. the width of the map times its height). $X = X^{(0)}$ is the input image and $\mathbf{f} = X^{(L)}$ is the output image. For convenience, we also define *convolutional feature maps* to be the feature maps after convolutional operation and define *pooled feature maps* and *up-sampled feature maps* in the same way. In the i^{th} layer, let r_i be the fixed kernel size or the pool size (e.g. 3×3). If $X^{(i+1)}$ is convolutional feature maps, let $Y^{(i)} \in \mathbb{R}^{N_i r_i \times \tilde{d}_i}$ be the patched feature

[2] For input images, the cat is an example image from caffe, and the other two are from the validation set of ImageNet.



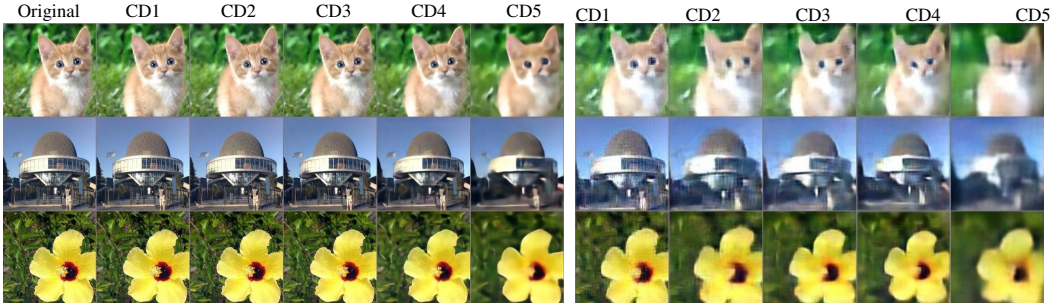
(a) VGG vs. Alexnet (b) rwVGG (c) rwVGG

Figure 10: Training loss for the Conv2-DeConv2 architecture.



(a) Random VGG vs. Pre-trained VGG (b) VGG vs. Alexnet on CD3

Figure 11: Comparison on the generalization error.



(a) rwVGG (b) rwAlexNet

Figure 12: (**Zoom in for details.**) Reconstructions for representations of different convolutional layers of rwVGG and rwAlexNet.

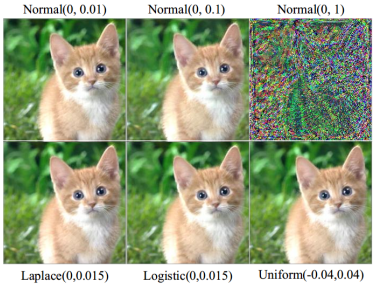


Figure 13: (**Zoom in for details.**) Reconstructions for representations of rwVGG Conv2 in various random distributions.

maps of $X^{(i)}$, where $\tilde{d}_i$ is the number of patches and in fact $\tilde{d}_i = d_{i+1}$ and $Y^{(i)}_{:,j}$ is the receptive field of the j^{th} pixel in a single channel output image after the i^{th} layer. To form $Y^{(i)}$, we first divide $X^{(i)}$ into patches and the m^{th} patch is $X^{(i)}_{:,\mathcal{D}^{(i)}_m} \triangleq \{X^{(i)}_{:,j} \mid j \in \mathcal{D}^{(i)}_m\}$, where $\mathcal{D}^{(i)}_m$ is an ordered set of indexes. $\mathcal{D}^{(i)}_{m,s}$ means the s^{th} corresponding element in $\mathcal{D}^{(i)}_m, s \in [r_i]$. We assume $\bigcup_{m=1}^{\tilde{d}_i} \mathcal{D}^{(i)}_m = [d_i]$, which is satisfied by most widely used CNNs. By transforming $X^{(i)}_{:,\mathcal{D}^{(i)}_m}$ into a column vector, we can obtain $Y^{(i)}_{:,m}$. $\mathbf{z}^{(i)} \in \mathbb{R}^{\tilde{d}_i}$ is a row vector defined by $\mathbf{z}^{(i)}_j = \|Y^{(i)}_{:,j}\|$, where $\|\cdot\|$ is the l_2-norm operation. $w^{(i,j)}$, a row vector, is the j^{th} convolutional filter of the i^{th} layer. If $X^{(i+1)}$ is pooled feature maps, we also divide $X^{(i)}$ into patches and $\mathcal{D}^{(i)}_m$ is the indexes of the m^{th} patch. If $X^{(i+1)}$ is up-sampled feature maps we have $\mathcal{D}^{(i)}_m = \{m\}$, which has only one element. So we can also define $Y^{(i)}$ and $\mathbf{z}^{(i)}$ for pooling and up-sampling layers. For the j^{th} pixel of the output image in the last layer, define its receptive filed on the input image in the first layer as $X_{:,\mathcal{R}_j} = \{X_{:,m} \mid m \in \mathcal{R}_j\}$, where $\mathcal{R}_j$ is a set of indexes. The activation function $ReLU(\mathbf{x}) = \max(\mathbf{x}, 0)$ is the element-wise maximum operation between $\mathbf{x}$ and 0 and $(\cdot)^m$ is the element-wise power operation.

Definition 2. Random CNN architecture This structure is different from the classic CNN in the following three points:

- Different filters in the same layer are i.i.d. random vectors and filters in different layers are independent. The probability density function of each filter is isotropic. Let $k^{(i)}_m = \frac{1}{2}\mathbb{E}|w^{(i,j)}_1|^m$ and $K^{(i)}_m = \frac{k^{(i)}_{2m} - (k^{(i)}_m)^2}{(k^{(i)}_m)^2}$. Suppose $k^{(i)}_1, k^{(i)}_2, k^{(i)}_4$ all exist.
- The last layer is the arithmetic mean of the channels of the previous layer, not the weighted combination.
- Except for $X^{(L-1)}$, each layer of convolutional feature maps are normalized by a factor of $\frac{1}{\sqrt{N_i}}$, where N_i is the number of channels of this layer.

A.3.1 CONVERGENCE

Theorem 4. ***(Convergence Value)*** *Suppose all the pooling layers use l_2-norm pooling. When the number of filters in each layer of a random CNN goes to infinity, the output $\mathbf{f}$ corresponding to a fixed input will converge to a fixed image $\mathbf{f}^*$ with probability 1, where $\mathbf{f}^* = k\mathbf{z}^*$ and k is a constant only related to the CNN architecture and the distribution of random filters and $\mathbf{z}^*_i = \sqrt{\sum_{l \in \mathcal{R}_i} n_{(l,i)} \|X_{:,l}\|^2}$, where $n_{(l,i)}$ is the number of routes from the l^{th} input pixel to the i^{th} output pixel.*

Here for the pooling layer, instead of average pooling, which calculates the arithmatic mean, we use l_2-norm pooling which calculates the norm of the values in a patch. We also show the result for average pooling in Theorem A.7.

To prove the theorem, we first prove the following lemma.

Lemma 5. *Suppose $w \in \mathbb{R}^n, n \geq 2$ is a random row vector and its probability density function is isotropic. $Y \in \mathbb{R}^{n \times d}$ is a constant matrix whose i^{th} column vector is denoted by y_i. $\mathbf{z} \in \mathbb{R}^d$ is a row vector and $\mathbf{z}_i = \|y_i\|$. Let $\mathbf{g} = max\{wY, 0\}$. If $k_m = \frac{1}{2}\mathbb{E}|w_1|^m$ exists, then $\mathbb{E}\mathbf{g}^m = k_m\mathbf{z}^m$ and $\mathbb{E}\mathbf{g}_i\mathbf{g}_j = \frac{1}{\pi}k_2[(\pi - \theta_{ij})\cos\theta_{ij} + \sin\theta_{ij}]\mathbf{z}_i\mathbf{z}_j$, where θ_{ij} is the angle between y_i and y_j.*

Proof. Note that $max\{\cdot,\cdot\}$ and $(\cdot)^m$ are both element wise operations. The i^{th} element of $\mathbb{E}\mathbf{g}^m$ is

$$(\mathbb{E}\mathbf{g}^m)_i = \mathbb{E}max\{wy_i, 0\}^m.$$

Since the probability density function of w is isotropic, we can rotate y_i to y'_i without affecting the value of $\mathbb{E}max\{wy_i, 0\}^m$. Let $y'_i = (\|y_i\|, 0, ..., 0)^T$, we have

$$\begin{aligned}(\mathbb{E}\mathbf{g}^m)_i &= \mathbb{E}max\{\|y_i\|w_1, 0\}^m \\ &= \mathbf{z}_i^m \mathbb{E}max\{w_1, 0\}^m \\ &= \mathbf{z}_i^m \frac{1}{2}\mathbb{E}|w_1^m| \\ &= k_m\mathbf{z}_i^m.\end{aligned}$$

Where the third equality uses the fact that the marginal distribution of w_1 is also isotropic. Similarly, we have:

$$\mathbb{E}\mathbf{g}_i\mathbf{g}_j = \mathbb{E}max\{wy_i, 0\}max\{wy_j, 0\}.$$

We can also rotate y_i and y_j to y_i' and y_j'. Let $y_i' = (\|y_i\|, 0, 0, ..., 0)^T$ and $y_j' = (\|y_j\|\cos\theta_{ij}, \|y_j\|\sin\theta_{ij}, 0, ..., 0)^T$ and suppose the marginal probability density function of (w_1, w_2) is $p(\rho)$ which does not depend on ϕ since it is isotropic, where $\rho = \sqrt{w_1^2 + w_2^2}$ is the radial coordinate and ϕ is the angular coordinate. We have:

$$\mathbb{E}\mathbf{g}_i\mathbf{g}_j = \mathbf{z}_i\mathbf{z}_j \int_0^\infty p(\rho)\rho^3 d\rho \int_{\theta_{ij}-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos(\theta_{ij} - \phi)\cos\phi d\phi.$$

Note that:

$$\int_0^\infty p(\rho)\rho^3 d\rho = \frac{1}{2\pi}\mathbb{E}\rho^2 = \frac{1}{\pi}\mathbb{E}w_1^2 = \frac{2}{\pi}k_2,$$

$$\int_{\theta_{ij}-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos(\theta_{ij} - \phi)\cos\phi d\phi = \frac{1}{2}((\pi - \theta_{ij})\cos\theta_{ij} + \sin\theta_{ij}).$$

We obtain the second part of this lemma. □

Now, we come to proof of Theorem A.1.

Proof. According to Lemma A.2, if $X^{(i+1)}$ is convolutional feature maps, we can directly obtain:

$$\mathbb{E}(X_{1,:}^{(i+1)})^2 = \frac{1}{N_{i+1}}\mathbb{E}max\{w^{(i,1)}Y^{(i)}, 0\}^2 = \frac{k_2^{(i)}}{N_{i+1}}(\mathbf{z}^{(i)})^2, \quad 0 \le i \le L-2$$

where we have fixed $Y^{(i)}$ and the expectation is taken over random filters in the i^{th} layer only. Since different channels in $X^{(i+1)}$ are i.i.d. random variables, according to the strong law of large numbers, we have:

$$\sum_{j=1}^{N_{i+1}} (X_{j,:}^{(i+1)})^2 \xrightarrow{a.s.} k_2^{(i)}(\mathbf{z}^{(i)})^2 \qquad \text{when } N_{i+1} \to \infty,$$

which implies that with probability 1,

$$\lim_{N_{i+1}\to\infty} (\mathbf{z}_m^{(i+1)})^2 = \lim_{N_{i+1}\to\infty} \sum_{l\in\mathcal{D}_m^{(i+1)}} \sum_{j=1}^{N_{i+1}} (X_{j,l}^{(i+1)})^2 = k_2^{(i)} \sum_{l\in\mathcal{D}_m^{(i+1)}} (\mathbf{z}_l^{(i)})^2.$$

Suppose that all N_j for $1 \le j \le i$ have gone to infinity and $\mathbf{z}^{(i)}$ has converged to $\mathbf{z}^{*(i)}$, the above expression is the recurrence relation between $\mathbf{z}^{*(i+1)}$ and $\mathbf{z}^{*(i)}$ in a convolutional layer: $(\mathbf{z}_m^{*(i+1)})^2 = k_2^{(i)} \sum_{l\in\mathcal{D}_m^{(i+1)}} (\mathbf{z}_l^{*(i)})^2$.

If $X^{(i+1)}$ is l_2-norm pooled feature maps, we have $\|X_{:,l}^{(i+1)}\| = \mathbf{z}_l^{(i)}$ by defination. Therefore,

$$(\mathbf{z}_m^{(i+1)})^2 = \sum_{l\in\mathcal{D}_m^{(i+1)}} (\mathbf{z}_l^{(i)})^2.$$

If $X^{(i+1)}$ is up-sampled feature maps, a pixel $X_{jp}^{(i)}$ will be up-sampled to a r-sized block $\{X_{jp_q}^{(i+1)} \mid q \in [r]\}$, where $X_{jp_1}^{(i+1)} = X_{jp}^{(i)}$ and all the other elements are zeros. Define $\tilde{\mathcal{D}}_m^{(i+1)} = \{p \mid p_1 \in \mathcal{D}_m^{(i+1)}\}$, we have:

$$(\mathbf{z}_m^{(i+1)})^2 = \sum_{l\in\tilde{\mathcal{D}}_m^{(i+1)}} (\mathbf{z}_l^{(i)})^2.$$

So far, we have obtained the recurrence relation in each layer. In order to get $\mathbf{z}^{*(i+1)}$ given $\mathbf{z}^{*(i)}$, we use the same sliding window scheme on $\mathbf{z}^{*(i)}$ as that of the convolutional, pooling or upsampling operation on the feature maps. The only difference is that in a convolutional layer, instead of

calculating the inner product of a filter and the vector in a sliding window, we simply calculate the l_2-norm of the vector in the sliding window and then multiply it by $\sqrt{k_2^{(i)}}$. Note that $\mathbf{z}^{*(0)}$ can be directly obtained from the input image. Repeat this process layer by layer and we can obtain $\mathbf{z}^{*(L-2)}$.

According to Lemma A.2, we have:

$$\mathbb{E}X_{1,:}^{(L-1)} = \mathbb{E}\max\{w^{(L-2,1)}Y^{(L-2)}, 0\} = k_1^{(L-2)}\mathbf{z}^{(L-2)}.$$

Suppose that $\mathbf{z}^{(L-2)}$ has converged to $\mathbf{z}^{*(L-2)}$, and by Definition A.1, $\mathbf{f} = \frac{1}{N_{L-1}}\sum_{i=1}^{N_{L-1}} X_{i,:}^{(L-1)}$, we have:

$$\mathbf{f} \xrightarrow{a.s.} k_1^{(L-2)}\mathbf{z}^{*(L-2)} \qquad \text{when } N_i \to \infty, \quad i \in [L-1].$$

Let $k = k_1^{(L-2)}\prod_{i=0}^{L-3} k_2^{(i)}$ and $\mathbf{z}^* = (\prod_{i=0}^{L-3} k_2^{(i)})^{-1}\mathbf{z}^{*(L-2)}$, we have $\mathbf{f}^* = k\mathbf{z}^*$. Note that $\mathbf{z}^*$ is obtained through a multi-layer sliding window scheme similar to the CNN structure. It only depends on the input image and the scheme. It is easy to verify that $\mathbf{z}_i^*$ is the square root of the weighted sum of the square of input pixel values within the receptive field of the i^{th} output pixel, where the weight of an input image pixel is the number of routes from it to the output pixel. □

A.3.2 VARIANCE

Theorem 6. ***(Variance)*** *For a two-layer random CNN with N filters in the first convolutional layer, let Θ denote the angle between the output $\mathbf{f}$ and the convergence value $\mathbf{f}^*$, then with probability $1-\delta$, $\sin\Theta \le \sqrt{K_1^{(0)}\frac{1}{N\delta}}$.*

Proof. According to Theorem A.1, we have $\mathbb{E}\mathbf{f} = \mathbf{f}^* = k_1^{(0)}\mathbf{z}^*$, where $\mathbf{z}^* = \mathbf{z}^{*(0)}$. For a two-layer CNN, we can directly obtain:

$$\mathbf{f} = \frac{1}{N}\sum_{i=1}^{N} X_{i,:}^{(1)} = \frac{1}{N}\sum_{i=1}^{N} max\{w^{(0,i)}Y^{(0)}, 0\},$$

Since different channels are i.i.d. random variables, we have $\mathbb{E}X_{i,:}^{(1)} = \mathbb{E}\mathbf{f} = k_1^{(0)}\mathbf{z}^*$. Then,

$$\begin{aligned}
\mathbb{E}\|\mathbf{f} - \mathbb{E}\mathbf{f}\|^2 &= \frac{1}{N}\mathbb{E}\|X_{i,:}^{(1)} - \mathbb{E}X_{i,:}^{(1)}\|^2 \\
&= \frac{1}{N}(\mathbb{E}\|X_{i,:}^{(1)}\|^2 - \|\mathbb{E}X_{i,:}^{(1)}\|^2) \\
&= \frac{1}{N}(\sum_{j=1}^{\tilde{d_0}} \mathbb{E}\max\{w^{(0,i)}Y_{:,j}^{(0)}, 0\}^2 - (k_1^{(0)})^2\|\mathbf{z}^*\|^2) \\
&= \frac{1}{N}(\sum_{j=1}^{\tilde{d_0}} k_2^{(0)}\|Y_{:,j}^{(0)}\|^2 - (k_1^{(0)})^2\|\mathbf{z}^*\|^2) \\
&= \frac{1}{N}(k_2^{(0)} - (k_1^{(0)})^2)\|\mathbf{z}^*\|^2
\end{aligned}$$

According to Markov inequality, we have:

$$Pr(\|\mathbf{f} - \mathbb{E}\mathbf{f}\|^2 \ge \epsilon^2) \le \frac{1}{N\epsilon^2}(k_2^{(0)} - (k_1^{(0)})^2)\|\mathbf{z}^*\|^2.$$

Let $\delta = \frac{1}{N\epsilon^2}(k_2^{(0)} - (k_1^{(0)})^2)\|\mathbf{z}^*\|^2$, then with probability $1-\delta$:

$$\sin\Theta \le \frac{\|\mathbf{f} - \mathbb{E}\mathbf{f}\|}{\|\mathbb{E}\mathbf{f}\|} \le \frac{\epsilon}{k_1^{(0)}\|\mathbf{z}^*\|} = \sqrt{\frac{k_2^{(0)} - (k_1^{(0)})^2}{(k_1^{(0)})^2}\frac{1}{N\delta}} = \sqrt{K_1^{(0)}\frac{1}{N\delta}}.$$

□

To extend the above two-layer result to a multi-layer one, we first prove the following lemma. Note that in this lemma, $\mathcal{D}^{(i)}$ should be replaced by $\tilde{\mathcal{D}}^{(i)}$ defined in the proof of Theorem A.1 if $X^{(i)}$ is up-sampled feature maps.

Lemma 7. *Let $\phi^{(i)}(\cdot) : \mathbb{R}^{\tilde{d}_i} \to \mathbb{R}^{\tilde{d}_{i+1}}$ for $0 \leq i \leq L-3$ denote the recurrence relation of $\{(\mathbf{z}^{*(i)})^2\}$ (i.e. $(\mathbf{z}^{*(i+1)})^2 = \phi^{(i)}((\mathbf{z}^{*(i)})^2)$), then $\phi^{(i)}(\cdot)$ is a linear mapping. For simplicity, suppose that for any $i \in [L-3]$, $card(\mathcal{D}_m^{(i)}) = r_i$ for any $m \in [\tilde{d}_i]$. And $\mathcal{D}_l^{(i)} \bigcap \mathcal{D}_m^{(i)} = \varnothing$ for any $l, m \in [\tilde{d}_i]$ if $l \neq m$. Then we can obtain $\|\phi^{(i)}(\mathbf{x})\|^2 = r_{i+1}(k^{(i)})^2(\|\mathbf{x}\|^2 - \|\epsilon^{(i)}(\mathbf{x})|^2) \leq r_{i+1}(k^{(i)})^2\|\mathbf{x}\|^2$, where $k^{(i)} = k_2^{(i)}$ for convolutional layers and $k^{(i)} = 1$ for pooling and up-sampling layers and $\epsilon^{(i)}(\cdot)$ is defined by $\epsilon^{(i)}(\mathbf{x})_j = \mathbf{x}_j - \frac{1}{r_{i+1}} \sum_{l \in \mathcal{D}_m^{(i+1)}} \mathbf{x}_l$, where m satisfies $j \in \mathcal{D}_m^{(i+1)}$.*

Proof. According to the definition of $\phi^{(i)}(\cdot)$ and Theorem A.1, we have:

$$\phi^{(i)}(\mathbf{x})_m = k^{(i)} \sum_{j \in \mathcal{D}_m^{(i+1)}} \mathbf{x}_j.$$

It is easy to verify that for any $c \in \mathbb{R}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{d_{i+1}}$ we have $\phi^{(i)}(c\mathbf{x}) = c\phi^{(i)}(\mathbf{x})$ and $\phi^{(i)}(\mathbf{x} + \mathbf{y}) = \phi^{(i)}(\mathbf{x}) + \phi^{(i)}(\mathbf{y})$. So $\phi^{(i)}(\cdot)$ is a linear mapping.

Define $\overline{\mathbf{x}}_m = \frac{1}{r_{i+1}} \sum_{j \in \mathcal{D}_m^{(i+1)}} \mathbf{x}_j$, which is the average value of the m^{th} patch. Let $\epsilon^{(i)}(\mathbf{x})_j = \mathbf{x}_j - \overline{\mathbf{x}}_m$, where m satisfies $j \in \mathcal{D}_m^{(i+1)}$. We have:

$$\begin{aligned}\sum_{j \in \mathcal{D}_m^{(i+1)}} \mathbf{x}_j^2 &= \sum_{j \in \mathcal{D}_m^{(i+1)}} (\overline{\mathbf{x}}_m + \epsilon^{(i)}(\mathbf{x})_j)^2 \\ &= r_{i+1}\overline{\mathbf{x}}_m^2 + \sum_{j \in \mathcal{D}_m^{(i+1)}} \epsilon^{(i)}(\mathbf{x})_j^2,\end{aligned}$$

Since $\{\mathcal{D}_m^{(i+1)} | m \in [\tilde{d}_{i+1}]\}$ is a partition of $[\tilde{d}_i]$ under our assumptions, we have $\|\mathbf{x}\|^2 = \sum_{m=1}^{\tilde{d}_{i+1}} \sum_{j \in \mathcal{D}_m^{(i+1)}} \mathbf{x}_j^2$ and $\|\phi^{(i)}(\mathbf{x})\|^2 = r_{i+1}^2(k^{(i)})^2 \sum_{m=1}^{\tilde{d}_{i+1}} \overline{\mathbf{x}}_m^2$, which implies that

$$\|\phi^{(i)}(\mathbf{x})\|^2 = r_{i+1}(k^{(i)})^2(\|\mathbf{x}\|^2 - \|\epsilon^{(i)}(\mathbf{x})|^2) \leq r_{i+1}(k^{(i)})^2\|\mathbf{x}\|^2.$$

□

Theorem 8. ***(Multilayer Variance)*** *Suppose all the pooling layers use l_2-norm pooling. For a random CNN with L layers and N_i filters in the i^{th} layer, let Θ denote the angle between the output $\mathbf{f}$ and the convergence value $\mathbf{f}^*$, suppose that there is at most one route from an arbitrary input pixel to an arbitrary output pixel for simplicity, then with probability $1 - \delta$,*

$$\sin\Theta \leq \sqrt{\frac{L-1}{\overline{N}\delta}} + \sqrt{(L-2)\sqrt{\frac{L-1}{\overline{N}\delta}} \prod_{i=0}^{L-2} \lambda_i},$$

where $\lambda_i = \frac{1}{\sqrt{1 - \frac{\|\epsilon^{(i)}((\mathbf{z}^{*(i)})^2)\|^2}{\|(\mathbf{z}^{*(i)})^2\|^2}}}$ and $\frac{1}{\overline{N}} = \frac{1}{L-1}(\frac{K_1^{(L-2)}}{N_{L-1}} + \sum_{i=1}^{L-2} \frac{K_2^{(i-1)}}{N_i})$.

Proof. We will bound Θ recursively. Suppose that the angle between $(\mathbf{z}^{(i)})^2$ and $(\mathbf{z}^{*(i)})^2$ is θ_i. We have $\theta_0 = 0$. Let $\mathbf{g}^{(i+1,j)} = (X_{j,:}^{(i+1)})^2$ and $\mathbf{g}^{(i+1)} = \frac{1}{N_{i+1}} \sum_{j=1}^{N_{i+1}} \mathbf{g}^{(i+1,j)}$. If $X^{(i+1)}$ is convolutional feature maps, we have obtained in the proof of Theorem A.1 that $\mathbb{E}\mathbf{g}^{(i+1)} = \mathbb{E}\mathbf{g}^{(i+1,j)} = k_2^{(i)}(\mathbf{z}^{(i)})^2$. Using similar method to the proof of Theorem A.3 and let α_{i+1} denote the angle between $\mathbf{g}^{(i+1)}$ and $\mathbb{E}\mathbf{g}^{(i+1)}$, we can derive that with probability $1 - \delta_{i+1}$,

$$\sin\alpha_{i+1} \leq \sqrt{K_2^{(i)} \frac{1}{N_{i+1}\delta_{i+1}}}.$$

For a l_2-norm pooling layer or an up-sampling layer, we have:

$$\sin\alpha_{i+1} = 0 \leq \sqrt{K_2^{(i)} \frac{1}{N_{i+1}\delta_{i+1}}}.$$

Let β_{i+1} denote the angle between $\mathbf{g}^{(i+1)}$ and $(\mathbf{z}^{*(i)})^2$. We have:

$$\sin\beta_{i+1} \leq \sin(\theta_i + \alpha_{i+1}) \leq \sin\theta_i + \sin\alpha_{i+1}.$$

Note that there exists a constant γ such that $\sin\beta_{i+1} = \frac{\|\gamma\mathbf{g}^{(i+1)} - (\mathbf{z}^{*(i)})^2\|}{\|(\mathbf{z}^{*(i)})^2\|}$. In fact, we can find the value of γ is $\frac{(\mathbf{z}^{*(i)})^2(\mathbf{g}^{(i+1)})^T}{\|\mathbf{g}^{(i+1)}\|^2}$, where $(\cdot)^T$ means transpose. We use $\phi^{(i)}(\cdot)$ to denote the recurrence relation of $\{(\mathbf{z}^{*(i)})^2\}$ (i.e. $(\mathbf{z}^{*(i+1)})^2 = \phi^{(i)}((\mathbf{z}^{*(i)})^2)$). Note that $(\mathbf{z}^{(i+1)})^2 = \phi^{(i)}(\mathbf{g}^{(i+1)})$ and $\phi^{(i)}(\cdot)$ is linear, using the result in Lemma A.4, we can obtain:

$$\begin{aligned}
\sin\theta_{i+1} &\leq \frac{\|\gamma(\mathbf{z}^{(i+1)})^2 - (\mathbf{z}^{*(i+1)})^2\|}{\|(\mathbf{z}^{*(i+1)})^2\|} \\
&= \frac{\|\phi^{(i)}(\gamma\mathbf{g}^{(i+1)} - (\mathbf{z}^{*(i)})^2)\|}{\|\phi^{(i)}((\mathbf{z}^{*(i)})^2)\|} \\
&\leq \frac{\|\gamma\mathbf{g}^{(i+1)} - (\mathbf{z}^{*(i)})^2\|}{\sqrt{\|(\mathbf{z}^{*(i)})^2\|^2 - \|\epsilon^{(i)}((\mathbf{z}^{*(i)})^2)\|^2}} \\
&= \lambda_i \sin\beta_{i+1} \\
&\leq \lambda_i(\sin\theta_i + \sin\alpha_{i+1}).
\end{aligned}$$

Where we defined $\lambda_i = \frac{1}{\sqrt{1 - \frac{\|\epsilon^{(i)}((\mathbf{z}^{*(i)})^2)\|^2}{\|(\mathbf{z}^{*(i)})^2\|^2}}}$ for $0 \leq i \leq L-3$, which is usually slightly bigger than 1 if the input is a natural image.

So far, we have derived the recurrence relation between $\sin\theta_{i+1}$ and $\sin\theta_i$. So we can get the bound of θ_{L-2}. However, note that θ_{L-2} is the angle between $(\mathbf{z}^{(L-2)})^2$ and $(\mathbf{z}^{*(L-2)})^2$ instead of that between $\mathbf{z}^{(L-2)}$ and $\mathbf{z}^{*(L-2)}$. We will denote the latter by μ which is what we really need. We know that there exists a constant γ' such that $\sin\theta_{L-2} = \frac{\|\gamma'(\mathbf{z}^{(L-2)})^2 - (\mathbf{z}^{*(L-2)})^2\|}{\|(\mathbf{z}^{*(L-2)})^2\|}$. Note that for any $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{+n}$, according to Cauchy-Schwarz inequality, we have:

$$n\sum_{i=1}^{n}(\mathbf{a}_i^2 - \mathbf{b}_i^2)^2 \geq (\sum_{i=1}^{n}|\mathbf{a}_i^2 - \mathbf{b}_i^2|)^2 = (\sum_{i=1}^{n}|\mathbf{a}_i - \mathbf{b}_i|(\mathbf{a}_i + \mathbf{b}_i))^2 \geq (\sum_{i=1}^{n}(\mathbf{a}_i - \mathbf{b}_i)^2)^2,$$

which implies that $\|\mathbf{a} - \mathbf{b}\|^4 \leq n\|\mathbf{a}^2 - \mathbf{b}^2\|^2$. Then we can bound μ:

$$\begin{aligned}
\sin\mu &\leq \frac{\|\sqrt{\gamma'}(\mathbf{z}^{(L-2)}) - (\mathbf{z}^{*(L-2)})\|}{\|\mathbf{z}^{*(L-2)}\|} \\
&\leq \sqrt{\frac{\sqrt{\tilde{d}_{L-2}}\|(\mathbf{z}^{*(L-2)})^2\|}{\|\mathbf{z}^{*(L-2)}\|^2}\sin\theta_{L-2}}
\end{aligned}$$

If we define $\tilde{d}_{L-1} = 1$ and $\mathcal{D}_1^{(L-1)} = [d_{L-1}]$, then we can define $\phi^{L-2}(\cdot) : \mathbb{R}^{d_{L-1}} \to \mathbb{R}$ as $\phi^{(L-2)}(\mathbf{x}) = \sum_{j\in[d_{L-1}]}\mathbf{x}_j$. Note that $\|\mathbf{z}^{*(L-2)}\|^2 = \phi^{L-2}((\mathbf{z}^{*(L-2)})^2)$ by definition. Then according to Lemma A.2, let $\lambda_{L-2} = \frac{1}{\sqrt{1 - \frac{\|\epsilon^{(L-2)}((\mathbf{z}^{*(L-2)})^2)\|^2}{\|(\mathbf{z}^{*(L-2)})^2\|^2}}}$, we have

$$\sin\mu \leq \sqrt{\lambda_{L-2}\sin\theta_{L-2}}$$

Let v denote the angle between $\mathbf{z}^{(L-2)}$ and $\mathbf{f}$, we have obtained its bound in Theorem A.3. With probability $1 - \delta_{L-1}$, we have $\sin v \leq \sqrt{K_1^{(L-2)}\frac{1}{N_{L-1}\delta_{L-1}}}$. With all the bounds above, define $\overline{N}$ by $\frac{1}{\overline{N}} = \frac{1}{L-1}(\frac{K_1^{(L-2)}}{N_{L-1}} + \sum_{i=1}^{L-2}\frac{K_2^{(i-1)}}{N_i})$ and choose $\delta_i = \frac{\delta\overline{N}K_2^{(i)}}{(L-1)N_i}$ for $i \leq L-2$ and $\delta_{L-1} = \frac{\delta\overline{N}K_1^{(L-2)}}{(L-1)N_{L-1}}$ for simplicity, we can obtain the bound of Θ: with probability $1 - \delta$,

$$\sin\Theta \leq \sqrt{\frac{L-1}{\overline{N}\delta}} + \sqrt{(L-2)\sqrt{\frac{L-1}{\overline{N}\delta}}\prod_{i=0}^{L-2}\lambda_i}$$

□

A.3.3 DIFFERENCE BETWEEN THE CONVERGENCE VALUE AND THE INPUT

For this part, we will give a detailed proof for a two-layer CNN and argue that the result can be directly extended to a multi-layer one only with a few slight changes of definition.

Theorem 9. *For a two-layer random CNN, suppose that each layer has a zero-padding scheme to keep the output dimension equal to the dimension of the original input. The kernel size is r and stride is 1. The input image is $X \in \mathbb{R}^{d_0}$, whose entries are all positive. $\epsilon_t = X_t - \overline{X_t}$ means the difference between one pixel X_t and the mean of the r-sized image patch whose center is X_t. Let Φ be the angle between the input image X and the convergence value of the output image, we have $\cos\Phi \geq 1 - \frac{1}{M}\sum_t \epsilon_t X_t$, where $M = \sum_t X_t^2$.*

Proof. According to Theorem A.1, we know that $\mathbf{f}^* = k\mathbf{z}^*$, where $\mathbf{z}_t$ is the square root of the sum of the square of the pixels in its corresponding receptive field (i.e. the l_2-norm of the pixels), which means $\mathbf{z}_t = \sqrt{\sum_{\alpha\in\mathcal{R}_t} X_\alpha^2}$ and k is a constant related to the network structure and distribution of the random filters. With zero-padding on the input image, we can calculate the angle between $\mathbf{f}^*$ and X:

$$\cos\Phi = \frac{\sum_t \left(\sqrt{\sum_{\alpha\in\mathcal{R}_t} X_\alpha^2} X_t\right)}{\sqrt{\sum_t \sum_{\alpha\in\mathcal{R}_t} X_\alpha^2}\sqrt{\sum_t X_t^2}}.$$

As each pixel contributes at most r times to the final output, we have

$$\sqrt{\sum_t \sum_{\alpha\in\mathcal{R}_t} X_\alpha^2} \leq \sqrt{r}\sqrt{M}.$$

Also, using Cauchy-Schwarz Inequality and the fact that all X_t are positive, we can obtain

$$\sqrt{\sum_{\alpha\in\mathcal{R}_t} X_\alpha^2} \geq \frac{1}{\sqrt{r}} \sum_{\alpha\in\mathcal{R}_t} X_\alpha.$$

Now, we can bound the above $\cos\Phi$ as follows:

$$\begin{aligned}
\cos\Phi &\geq \frac{\sum_t \left(\frac{1}{\sqrt{r}} X_t \sum_{\alpha\in\mathcal{R}_t} X_\alpha\right)}{\sqrt{r}\sqrt{M}\sqrt{M}} \\
&= \frac{1}{M}\sum_t X_t \overline{X_t} \\
&= \frac{1}{M}\sum_t X_t(X_t - \epsilon_t) \\
&= 1 - \frac{1}{M}\sum_t \epsilon_t X_t.
\end{aligned}$$

□

The above theorem indicates that, if the input is an image whose pixels are similar to their adjacent pixels, then the the angle between the input image X and the convergence value of the output image will be small.

We point out that the above theorem can be directly extended to multi-layer convolutional neural networks. Suppose that the neural network has multiple layers. According to Theorem A.1, $\mathbf{f}^* = k\mathbf{z}^*$. Now, the pixels in the receptive fields contribute unequally to the corresponding output. Note that when the network is multi-layer, the receptive field is greatly enlarged. We can similarly obtain:

$$\cos\Phi = \frac{\sum_t \left(\sqrt{\sum_{\alpha\in\mathcal{R}_t} n_{(\alpha,t)} X_\alpha^2} X_t\right)}{\sqrt{\sum_t \sum_{\alpha\in\mathcal{R}_t} n_{(\alpha,t)} X_\alpha^2}\sqrt{\sum_t X_t^2}}.$$

Here, $\mathcal{R}_t$ is the index set of the receptive field of $\mathbf{f}_t$ and $n_{(\alpha,t)}$ is the number of routes from X_α to $\mathbf{f}_t$. Suppose that the receptive field of each $\mathbf{f}_t$ has the same size and shape, X_t is at a fixed relative position of the receptive field of $\mathbf{f}_t$ and $n_{(\alpha,t)}$ only depends on the relative position between X_α and X_t. Let $\overline{X_t} = \frac{\sum_{\alpha\in\mathcal{R}_t} n_{(\alpha,t)} X_\alpha}{\sum_{\alpha\in\mathcal{R}_t} n_{(\alpha,t)}}$ be the weighted average and $\epsilon_t = X_t - \overline{X_t}$. By using the same technique above, we can obtain that

$$\cos\Phi \geq 1 - \frac{1}{M}\sum_t \epsilon_t X_t.$$

Note that although the bound is the same as the two-layer convolutional neural network, as the receptive field is enlarged, ϵ_t can be much larger, so that the above bound will be worse.

We also give the convergence value for average pooling in the next theorem.

Theorem 10. ***(Convergence Value, average pooling)*** *Suppose all the pooling layers use average pooling. When the number of filters in each layer of a random CNN goes to infinity, the output* $\mathbf{f}$ *corresponding to a fixed input will converge to a fixed image* $\mathbf{f}^*$ *with probability* 1.

Proof. Define $C^{(i)} \in \mathbb{R}^{d_i\times d_i}$ by $C^{(i)}_{jk} = (X^{(i)}_{:,j})^T X^{(i)}_{:,k}$. If $X^{(i+1)}$ is convolutional feature maps, according to Lemma A.2, we have:

$$\mathbb{E}X^{(i+1)}_{1,j}X^{(i+1)}_{1,k} = \frac{1}{N_{i+1}}\mathbb{E}max\{w^{(i,1)}Y^{(i)}_{:,j},0\}max\{w^{(i,1)}Y^{(i)}_{:,k},0\} = \frac{1}{N_{i+1}}k^{(i)}_2 h(\varphi^{(i)}_{jk})\mathbf{z}^{(i)}_j\mathbf{z}^{(i)}_k,$$

where $\varphi^{(i)}_{jk}$ is the angle between $Y^{(i)}_{:,j}$ and $Y^{(i)}_{:,k}$ and we defined $h(x) = \frac{1}{\pi}[(\pi - x)\cos x + \sin x]$ for abbreviation. We have fixed $Y^{(i)}$ and the expectation is taken over random filters in the i^{th} layer only. Since different channels in $X^{(i+1)}$ are i.i.d. random variables, according to the strong law of large numbers, we have:

$$C^{(i+1)}_{jk} = \sum_{l=1}^{N_{i+1}} X^{(i+1)}_{l,j}X^{(i+1)}_{l,k} \xrightarrow{a.s.} k^{(i)}_2 h(\varphi^{(i)}_{jk})\mathbf{z}^{(i)}_j\mathbf{z}^{(i)}_k \qquad \text{when } N_{i+1}\to\infty.$$

Note that:

$$\mathbf{z}^{(i)}_j = \sqrt{\sum_{l\in\mathcal{D}^{(i)}_j}\|X^{(i)}_{:,l}\|^2} = \sqrt{\sum_{l\in\mathcal{D}^{(i)}_j} C^{(i)}_{ll}},$$

$$\cos\varphi^{(i)}_{jk} = \frac{(Y^{(i)}_{:,j})^T Y^{(i)}_{:,k}}{\mathbf{z}^{(i)}_j\mathbf{z}^{(i)}_k} = \frac{\sum_{s=1}^{r_i} C^{(i)}_{\mathcal{D}^{(i)}_{j,s}\mathcal{D}^{(i)}_{k,s}}}{\mathbf{z}^{(i)}_j\mathbf{z}^{(i)}_k}.$$

Suppose that all N_j for $1\leq j\leq i$ have gone to infinity and $C^{(i)}$ has converged to $C^{*(i)}$, the above expressions are the recurrence relation between $C^{*(i+1)}$ and $C^{*(i)}$ for a convolutional layer. If $X^{(i+1)}$ is average-pooled feature maps, we have:

$$X^{(i+1)}_{:,j} = \frac{1}{r_i}\sum_{l\in\mathcal{D}^{(i)}_j} X^{(i)}_{:,l}.$$

We have:

$$C^{(i+1)}_{jk} = (X^{(i+1)}_{:,j})^T X^{(i+1)}_{:,k} = \frac{1}{r_i^2}\sum_{l\in\mathcal{D}^{(i)}_j, m\in\mathcal{D}^{(i)}_k}(X^{(i)}_{:,l})^T X^{(i)}_{:,m} = \frac{1}{r_i^2}\sum_{l\in\mathcal{D}^{(i)}_j, m\in\mathcal{D}^{(i)}_k} C^{(i)}_{lm},$$

which is the recurrence relation for an average pooling layer.

For an up-sampling layer, a pixel $X^{(i)}_{jk}$ will be up-sampled to a block $\{X^{(i+1)}_{jk_m} \mid m\in[r]\}$, where $X^{(i+1)}_{jk_1} = X^{(i)}_{jk}$ and all the other elements are zeros. We have:

$$C^{(i+1)}_{j_l k_m} = \begin{cases} C^{(i)}_{jk} & \text{for } l = m = 1, \\ 0 & \text{otherwise.}\end{cases}$$

Note that we can directly calculate $C^{*(0)}$ according to the input image. So we can recursively obtain $C^{*(L-2)}$ and thus $\mathbf{z}^{*(L-2)}$.

According to Lemma A.2, we have:

$$\mathbb{E}X_{1,:}^{(L-1)} = \mathbb{E}\max\{w^{(L-2,1)}Y^{(L-2)}, 0\} = k_1\mathbf{z}^{(L-2)}.$$

Suppose that $\mathbf{z}^{(L-2)}$ has converged to $\mathbf{z}^{*(L-2)}$, and by Definition A.1, $\mathbf{f} = \frac{1}{N_{L-1}}\sum_{i=1}^{N_{L-1}} X_{i,:}^{(L-1)}$, we have:

$$\mathbf{f} \xrightarrow{a.s.} k_1\mathbf{z}^{*(L-2)} \qquad \text{when } N_i \to \infty, \quad i \in [L-1].$$

We can obtain the convergence value $\mathbf{f}^*$ through the above process. □

A.4 Style Transfer Applications on rrVGG

We observe that by choosing a suitable number of random filters, the rrVGG Conv1-DeConv1 architecture can achieve high-quality reconstruction. The reconstructions also bring slight differences in the background color and texture, which is suited for exploring more interesting style transfer results. Hence we utilize the framework, as shown in Fig. 14 to explore style transfer on rrVGG models.

Recent work on style transfer employing the Convolutional Neural Network(CNN) raised significant research interest. They either utilize the pretrained CNN for iterative optimization Gatys et al. (2016) or train a feed-forward network Johnson et al. (2016); Ulyanov et al. (2017b) to achieve the stylization efficiently. These approaches all adopted CNN to extract the feature map for constructing both the style loss and content loss, achieving impressive stylization results. However, most of the work either relies on optimization process Gatys et al. (2016) which is particularly slow or be limited by a single style for each network Johnson et al. (2016); Ulyanov et al. (2017b). Hence, rather than taking images as the input and output for the optimization process, we feed the feature vector gained from the random CNN into the Feature Vector Transformation (FVT) component and output the stylized feature vector V_{new}. Feeding V_{new} into random DCN will generate the stylized image which is comparable to existing methods with less computational time.

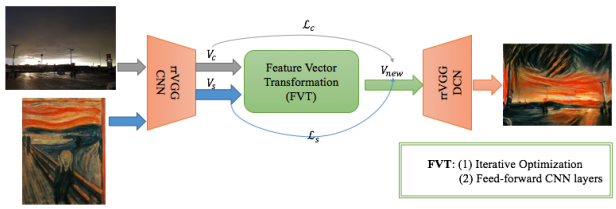


Figure 14: Overview of style transfer on rrVGG (random convolution and random decomvolution).

As Fig. 14 shows, in the rrVGG Conv1-DeConv1 network, we extract the activation vectors of both content and style images, V_s, V_c after the Conv1_2 layer. Afterwards, feeding the vectors V_c, V_s into the FVT component. This framework is flexible as FVT can be replaced either by feed-forward network or optimization process. Here we use iterative optimization for FVT to generate the stylized feature vector.

Based on Ulyanov et al. (2016); Gatys et al. (2016), we also adopted the linear combination of both content loss and style loss, $\mathcal{L}(V_s, V_c, V_{new}) = \alpha\mathcal{L}_c(V_c, V_{new}) + \beta\mathcal{L}_s(V_s, V_{new})$. $\mathcal{L}_c$ indicates the content loss which is the euclidean distance between the content vector V_c and stylized feature vector V_{new}, where A_i is the activation vector in *i-th* layer of the FVT.

$$\mathcal{L}_c(V_c, V_{new}) = \sum_{i \in L} \|A_i(V_c) - A_i(V_{new})\|^2 \tag{3}$$

The style loss $\mathcal{L}_s$ is obtained from the Gram matrix of the feature vectors. In Eq. equation 4, G represents the gram matrix. Inspired by Li et al. (2017)Huang & Belongie (2017), we also utilize the the mean value and standard deviation of feature vectors to calculate the style loss, the result of which is similar to the gram loss $\mathcal{L}_s$.

$$\mathcal{L}_s(V_s, V_{new}) = \sum_{i \in L} \|G(A_i(V_s)) - G(A_i(V_{new}))\|^2 \tag{4}$$

In Fig. 14, FVT (iterative optimization) only contains the convolutional layer from Conv2 to Conv5 and rrVGG contains Conv1 and DeConv1. In addition, we can also utilize more layers on rrVGG and FVT will contain less layers on the optimization network correspondingly, which will further speed up the style transfer process. In experiments, our framework is faster than the original optimization based approach and can transfer the arbitrary styles. The following illustrated figures are all generated by implementing FVT with the optimization process. Compared with AdaIN Huang & Belongie (2017), our process only consume time on FVT to generate the stylized feature vector V_{new} and employ random DCN to convert V_{new} to the stylized image while Huang *et al.* Huang & Belongie (2017) propose an AdaIN layer for transferring feature statistics and consume most of their time at the expense of training DCN. For the FVT, as Fig. 14 shows, it can also be implemented by a feed-forward network to accomplish the generation of stylized feature vector, which has achieved a comparable speed with respect to the existing work Johnson et al. (2016); Ulyanov et al. (2017b); Chen & Schmidt (2016); Chen et al. (2017). Since feature vectors V_s, V_c are acquired from Conv1_2 layer, thus having lower spatial resolution, which makes the optimization (FVT) on $\mathcal{L}$ converge faster than existing works utilizing images as the input. Meanwhile, rrVGG CNN-DCN parts are free of training, which compares favourably against Huang & Belongie (2017).

Figure 15: Style Transfer results Comparison

As for the stylization effectiveness, we compared our results with Gatys *et al.* Gatys et al. (2016) and Ulyanov *et al.* Ulyanov et al. (2017b). In Fig. 15, $rrVGG_1$ and $rrVGG_2$ columns denote the stylization results acquired from our framework, applying two different rrVGG models. As shown in Fig. 15, our stylization results are competitive to other well-trained approaches. Focused on $rrVGG_1$ column, our stylized result is inclined to extract more features from the style image and slightly weaken the representation of content image. Since we utilize rrVGG CNN and DCN to complete the transformation between feature space and image space, some content information is possible to be lost during the reconstruction process. Despite that, our approach is still capable of generating high quality stylized images.

In addition, we also investigate the stylized effectiveness when modifying the balance between style and content in FVT. As shown in Figure 16, the number below each column indicates the relative weightings between style and content reconstruction. In our framework, the transition from content to style is smooth with increasing ratio.

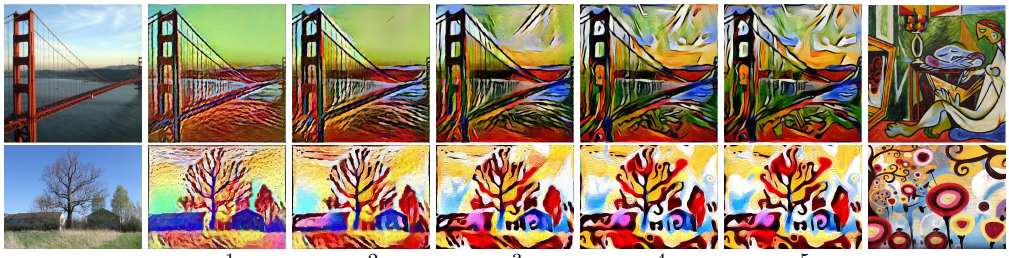


Figure 16: Detailed results for the rrVGG style transfer. Each column represents the ratio between style and content during the optimization phase.

As shown in Figure 17, our stylized result is inclined to extract more features from the style image and slightly weaken the representation of the content image.

Figure 17: Style transfer from several rrVGG models. Each model has the same architecture but different random weights.

As proposed in our paper, in terms of different distributions and number of filters, rrVGG can reconstruct images with diverse textures and background colors. In Fig. 14, replacing CNN and DCN parts with different rrVGG models, our framework can generate abundant stylized images depending on a single style image. Since rrVGG models are generated without training, it won't incur additional computational cost. As shown in Fig. 17, the rightmost three columns comprise the stylized images with different rrVGG model weights while the leftmost two columns represent input content and style images respectively. For each row, given content and style images, we choose three stylized images generated by our framework using different rrVGG models. For instance, in the *3-rd* row of Fig. 17, the parameters of chosen rrVGG models are as following: rrVGG$_1$:($N(0, 0.01)$, filter size:3, filter num:128), rrVGG$_2$:($N(0, 0.01)$, filter size:5, filter num:256) and rrVGG$_3$:($N(0, 0.1)$, filter size:3, filter num:32). As shown in Fig. 17, those stylized images not only well preserve the style structure such as the shape of the curved lines, waves and abstract objects, but also exhibit novel combinations of the structure, shade and hue. Coupled with various rrVGG models, the proposed style transfer framework is able to unleash the diversity and variation inside a single style image, which works well in practice. Meanwhile, it's flexible as well as fast, since the FVT part can be implemented either by an optimization process or some feed-forward convolutional layers.