Generating Natural images with direct Patch Distributions Matching

Ariel Elnekave and Yair Weiss

Hebrew University Jerusalem
Ariel.Elnekave@mail.huji.ac.il, Yair.Weiss@mail.huji.ac.il

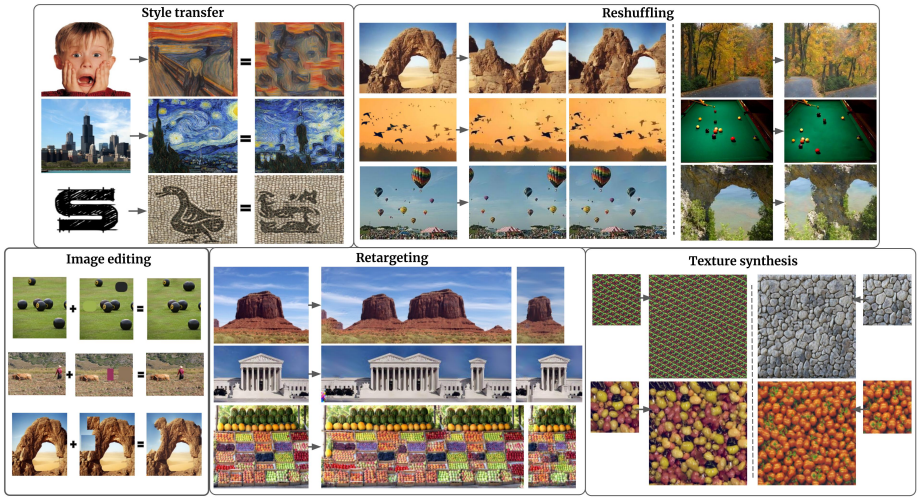

Fig. 1: By efficiently matching the distribution of patches in a target image and the generated image we can solve a broad spectrum of single-image generative tasks without training a per-image GAN and without computing patch nearest neighbors.

Abstract. Many traditional computer vision algorithms generate realistic images by requiring that each patch in the generated image be similar to a patch in a training image and vice versa. Recently, this classical approach has been replaced by adversarial training with a patch discriminator. The adversarial approach avoids the computational burden of finding nearest neighbors of patches but often requires very long training times and may fail to match the distribution of patches.
In this paper we leverage the recently developed Sliced Wasserstein Distance and develop an algorithm that explicitly and efficiently minimizes the distance between patch distributions in two images. Our method is conceptually simple, requires no training and can be implemented in a few lines of codes. On a number of image generation tasks we show that our results are often superior to single-image-GANs, require no training, and can generate high quality images in a few seconds. Our implementation is available at https://github.com/ariel415el/GPDM

1 Introduction

In a wide range of computer vision problems (e.g. image retargeting, super-resolution, novel view synthesis) an algorithm needs to generate a realistic image as an output. A classical approach to ensuring that the output image appears realistic is based on *local patches* [33],[2],[5]: if each patch in the output image is similar to a patch in a training image, we can assume that the generated image will be realistic. Similarly, if each patch in the generated image is similar to a patch in a Van-Gogh painting, we can assume that the generated image captures the "style" of Van-Gogh. This insight led to a large number of papers over the past two decades that use patch nearest neighbors to ensure high quality image outputs [5],[26],[13],[27].

One observation shared by successful methods based on patches is that the similarity between patches should be *bidirectional*: it is not enough to require that each patch in the generated image be similar to a patch in the training image. Consider a generated image that consists of many repetitions of a single patch from a Van-Gogh image: even though each patch in the generated image is similar to a patch in the target image, no one would consider such a generated image to capture the "style" of Van-Gogh. In order to rule out such solutions, the bidirectional similarity (BDS) method used in [33,2,10] also requires that each patch in the training image be similar to a patch in the generated image. As we show in section 2.1, while bidirectional similarity indeed helps push the distribution of patches in the generated image towards that of the target image, it still falls short of matching the distributions. Furthermore, optimizing the BDS loss function requires finding nearest neighbors of patches and this is both memory and computation intensive (given M patches in each image, BDS is based on an M^2 matrix of similarities between all pairs of patches).

In recent years, these classical, patch-based approaches have been overtaken by Generative Adversarial Networks (GANs) and related methods [14],[31],[32],[35]. In the adversarial approach, a discriminator is trained to classify patches at different scales as "real" or "fake" and it can be shown that under certain conditions training with an adversarial loss and a patch discriminator is equivalent to minimizing the distance between the distribution of patches in the generated image and the training image [1],[9]. The requirement that the generated image has the same *distribution* over patches as the target image addresses the limitation of early patch-based approaches that simply required that each patch in the generated image be similar to a patch in the training image. Indeed Single-Image GANs [32],[31] have yielded impressive results in generating novel images that have approximately the same distribution over patches as the target image.

Despite the considerable success of GAN-based methods, they have some notable disadvantages. While there are theoretical guarantees that globally optimizing the adversarial objective is equivalent to optimizing the distance between distributions, in practice GAN training often suffers from "mode collapse" [34] and the generated image may contain only few types of possible patches. Furthermore, GAN training is computationally intensive and a separate generator needs to be trained for different image tasks.

In this paper we leverage the recently proposed Sliced Wasserstein Distance [3],[4] to develop an algorithm that explicitly and efficiently minimizes the distance between patch distributions in two images without the need to compute patch nearest neighbors. Our method is conceptually simple, requires no training and can be implemented in a few lines of codes. On a number of image generation tasks we show that our results are often superior to single-image-GANs, require no training, and can generate high quality images in a few seconds.

2 Distances between distributions

Given M patches in two images, how do we compute the distance between the distribution of patches in the two images? The Wasserstein (or Earth Movers) distance between two distributions P, Q is defined as:

$$W(P,Q) = \inf_{\gamma \in \Pi(P,Q)} E_{x,y\sim\gamma}\|x-y\| \tag{1}$$

where $\Pi(P,Q)$ denotes the set of joint distributions whose marginal probabilities are P, Q. Intuitively, Π can be thought of as a soft correspondence between samples in P and Q and so the Wasserstein distance is the average distance between corresponding samples with the optimal correspondence. Calculating this optimal correspondence is computationally intensive ($O(M^{2.5})$[29]) making it unsuitable for use as a loss function that we wish to optimize for many iterations.

The sliced Wasserstein distance (SWD) makes use of the fact that for one dimensional data, the optimal correspondence can be solved by simply sorting the samples and so the distance between two samples of size M can be computed in $O(M \log M)$. For a projection vector w define P^w as the distribution of samples from P projected in direction w, the Sliced Wasserstein Distance is defined as:

$$SWD(P,Q) = E_w W(P^w, Q^w) \tag{2}$$

During optimization we can obtain an unbiased sample of the gradient of $SWD(P,Q)$ by randomly choosing k projection vectors w_i and computing the gradient of:

$$\tilde{SWD}(P,Q) = \frac{1}{k}\sum_i W(P^{w_i}, Q^{w_i}) \tag{3}$$

2.1 Properties and Comparisons

As mentioned in the introduction, many classical approaches to comparing patch distributions are based on bidirectional similarity. Suppose we are given a set

of samples $\{p_i\}, \{q_j\}$ from two distributions P, Q the Bidirectional Similarity (BDS) is defined as:

$$BDS(P,Q) = \frac{1}{M}\sum_i \min_j \|p_i - q_j\| + \frac{1}{M}\sum_j \min_i \|q_j - p_i\|$$

The first term ("coherence") measures the average distance between a patch in $\{p_i\}$ and its closest patch in $\{q_j\}$ and the second term ("completeness") measures the average distance between a patch in $\{q_i\}$ and its closest patch in $\{p_j\}$. Thus two images are judged to be similar if each patch in one image has a close match in the second image and vice versa.

Kolkin et al, [19] used a closely related measure which they called the "Relaxed Earth Movers Distance".

$$REMD(P,Q) = \max(\frac{1}{M}\sum_i \min_j \|p_i - q_j\|, \frac{1}{M}\sum_j \min_i \|q_j - p_i\|)$$

Again, two images are judged to be similar if each patch in one image has a close match in the second image and vice versa.

Thus SWD(P,Q), BDS(P,Q) and REMD(P,Q) are all methods to measure the similarity between the patch distributions. Why should one method be preferred over the others? The following theorem shows that neither BDS nor REMD can be considered as distance metric between distributions, while SWD can.

Theorem: SWD(P,Q)=0 if and only if P=Q. On the other hand for both BDS and REMD, there exist an infinite number of pairs of distributions P, Q that are arbitrarily different (i.e. $W(P,Q)$ is arbitrarily large) and yet BDS(P,Q)=0 and REMD(P,Q)=0.

Proof: The fact that SWD(P,Q)=0 if and only if P=Q follows from the fact that the Wasserstein distance is a metric [29]. To see that neither BDS nor REMD are metrics, note that any two discrete distributions that have the same support will satisfy $BDS(P,Q) = 0$ and $REMD(P,Q) = 0$ regardless of the densities on the support. This is because the closest match $\arg\min_j \|p_i - q_j\|$ can be the same for different i and a single sample q_{j^*} can serve as an exact match for an arbitrarily large number of samples $\{p_i\}$ For example, suppose P and Q are both distributions over the set $\{0, a\}$ for some constant a and $P(0) = \epsilon, Q(0) = 1 - \epsilon$. For any ϵ ,$BDS(P,Q) = REMD(P,Q) = 0$ (since all samples from P will have an exact matching sample in Q and vice-versa) even though as $\epsilon \to 0$ $W(P,Q) \to a$. By increasing a we can increase $W(P,Q)$ arbitrarily ■.

To illustrate the difference between SWD, BDS and REMD in the context of image patches, consider three images of sky and grass, each with 1000 patches. Images A and B, have 999 sky patches and one grass patch, while image C has 1 sky patch and 999 grass patches. Note that both when comparing A and B, and when comparing B and C *both coherence and completeness losses will be zero.* This means that BDS and REMD will consider A and B (which have the same patch distribution) to be as similar as B and C (which have very

different distributions). In contrast, the SWD will be zero if and only if the two distributions are identical, and SWD(A,B) will be much lower than SWD(B,C).

Algorithm 1 GPDM Module

Input: Target image x, initial guess $\hat{y}$,
learning rate α
Output: Optimized image y

$y \leftarrow \hat{y}$
while not converged **do**
  $L \leftarrow 0$
  for i=1,k **do**
    $w \sim N(0, \sigma I)$
    $p \leftarrow flat(conv2d(x, w))$
    $q \leftarrow flat(conv2d(y, w))$
    $L \leftarrow L + \frac{1}{k}|sort(p) - sort(q)|$
  end for
  $y \leftarrow y - \alpha \nabla_y L$
end while

Algorithm 2: A pseudo-code of the GPDM module where SWD over sets of patches in two images is computed and differentiated through. The "flat" operator reshapes a tensor into a vector.

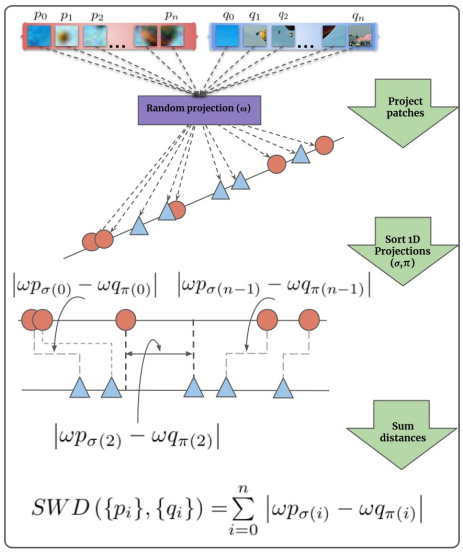

Fig. 2: Description of SWD as a patch distributions metric. SWD projects all patches to 1D then each preconditions set is sorted and the distances of matching pairs are summed. π, σ are the sorting permutations of the two projection sets and ω is a single random projection.

2.2 SWD for image patch distributions

SWD has been previously suggested for use as a training loss for different image processing methods [29],[3],[20],[4]. Here we point out that SWD allows us to efficiently compute an unbiased estimate of the distance between the patch distributions in two images using a single convolution of the two images with a random filter followed by sorting the values of the two convolved images (Figure 2). The unbiased estimate of the distance between the patch distributions in the two images is then simply the L1 distance between the sorted vectors and an unbiased estimate of the gradient of the image with respect to the SWD loss can be obtained by another convolution of the thresholded and sorted difference image with a flipped version of the filter. Thus *an SGD update of all pixels in the image with respect to the loss can be computed with two convolutions.* In order to reduce variance, we use a fixed small number of random projections. The full algorithm is described in Alg. 2.1. Note that it allows us to optimize the difference between patch distributions in two images *without finding patch nearest neighbors* and at a complexity of $M \log M$.

3 Method

Our method uses the same multi scale structure as previous works [31],[10],[5]. At each level an initial guess is transformed into an output image which is either used as an initial guess for the next level or is the final output. More formally, given a target image x we build an image pyramid out of it $(x_0, x_1, ...x_n)$ specified with a downscale ratio $r < 1$ and a minimal height for the coarsest level. We start with an initial guess $\hat{y_n}$ of the same size as x_n and at each level i we optimize the initial guess using algorithm 2.1 to minimize its patch-SWD with x_i. The optimization output y_i is a final output or up-scaled by $\frac{1}{r}$ to serve as an initial guess for the next level $i+1$ optimization.

We use the same learning rate and number of Adam steps to optimize SWD in all scales. We work on images normalized to [-1, 1] and clip the values of the optimized image to this interval at the end of each level optimization.

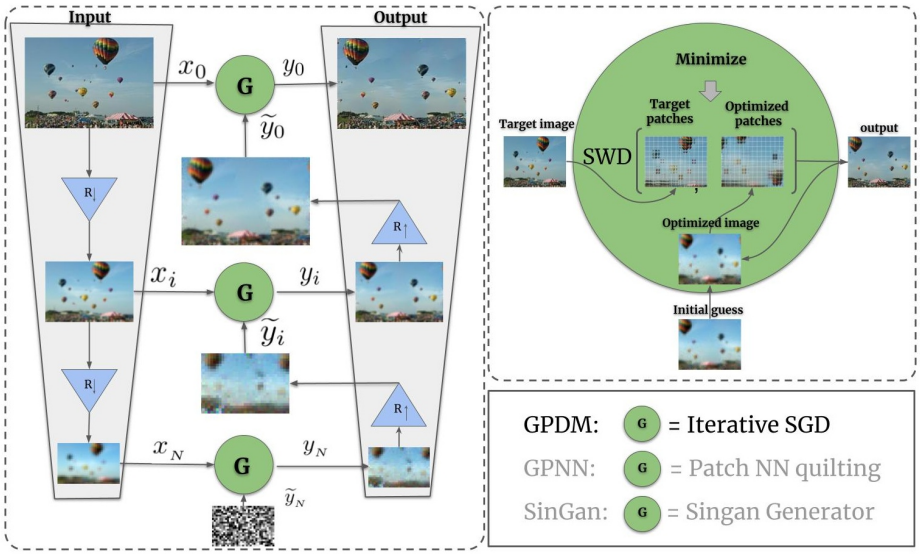

Fig. 3: GPDM's multi-scale architecture is very similar to that of GPNN [31] and SINGAN [31]: At each scale, i, an image is optimized to have similar patch distribution as the target x_i. The generation module G (a GAN in [31] or a patch NN quilting in [10]), is an optimization process of the differentiable patch-distribution metric $SWD(x, y)$. Each scale's output serves as an initial guess for the next scale. The first initial guess can be a blurred version of the target, a color map or simple pixel noise.

4 Experiments

We compare our approach with SinGAN[31] as well as a recent method, GPNN [10], which approximately optimizes the bidirectional similarity between the gener-

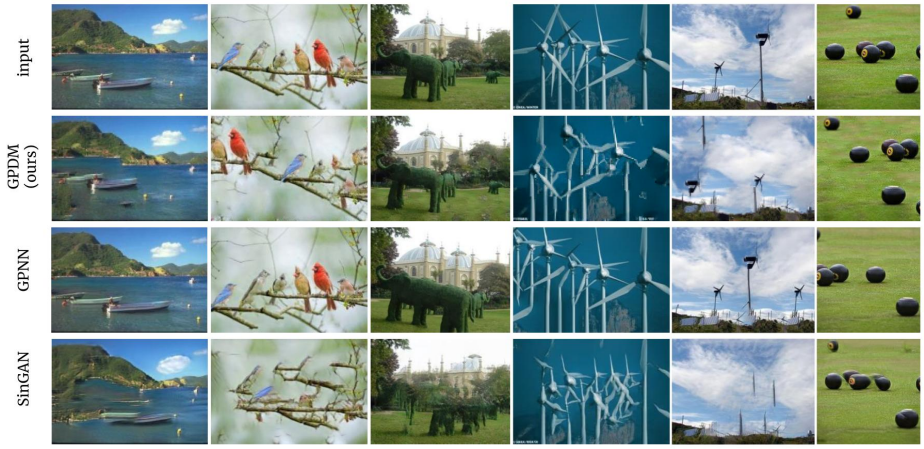

Fig. 4: Reshuffling results on natural images from the Places50 and SIGD16 selected by [10].

Dataset	METHOD	SFID	NIQE	Diversity
Places50	GT	0	4.81	
	Ours	0.068	5.53	0.56
	GPNN	0.065	4.79	0.5
	SINGAN	0.082	4.76	0.5
SIGD16	GT	0	5.18	
	Ours	0.069	5.99	0.67
	GPNN	0.122	4.99	0.52
	SINGAN	0.172	4.66	0.49

Table 1: Comparison of our method in the reshuffling task over images from the Places50 and SIGD16 datasets. Our method outperforms SINGAN and is on par with GPNN. Note that lower SFID scores are better. Diversity is computed as normalized per-pixel standard deviation over 50 generated image sets like in [31]. Generating images in a fixed diversity level ensures that the SFID comparison is meaningful and that the generated images are not all too close to the target image.

ated image and the target image. GPNN generates a new image by copying patches from the training image in a coarse to fine manner: at each iteration it searches for patches in the training image that are closest to the current estimated image and then aggregates these patches to form a new image. By construction, this method achieves high coherence (since all patches in the new image are copied from the training image) and completeness is encouraged by adding a term to the distance metric between patches so that patches that have already been used are made more distant. We use images from the Places50 and SIGD16 datasets described in [10], [31]. All three methods use exactly the same coarse to fine framework depicted in figure 3 but they differ in the way they match patch distributions: GPNN approximately minimizes the BDS, SinGAN approximately minimizes the KL divergence between patch distributions and

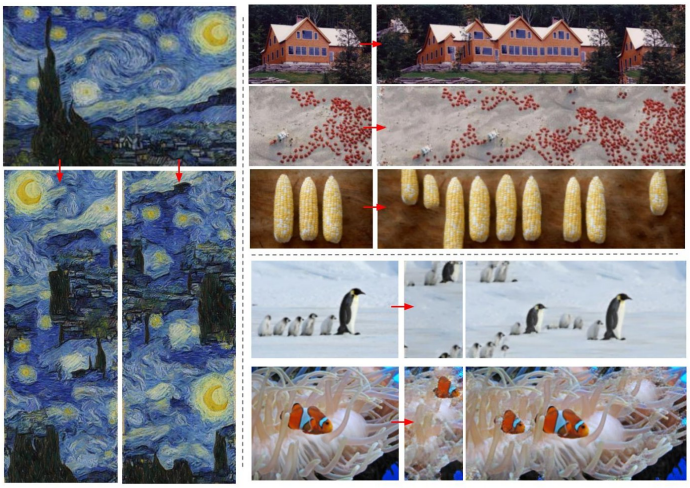

Fig. 5: Result of our method in the image retargeting task. The input images are transformed into images containing the same scene But with different aspect ratios. Inputs are at the origins of the red arrows

our method exactly minimizes the SWD. Hyperparameters and details for all methods are available in the supplementary.

We first compare the three algorithms on the single image generation (or "reshuffling") task. Given a natural image the task is to generate more images from the same scene which are different from the original one but show the same scene and are visually coherent. For this task, we use patch size 7 and start the optimization from a noise image. . Figure 4 compares our method to [10], [31] on the same images from Figure 4 in [10]. It can be seen that our method provides comparable visual quality to that of GPNN, and both methods generate more realistic and artifact-free images compared to SinGAN. Table 1 shows a quantitative comparison between our method and [10], [31]. SIFID is a full reference image quality metric described in [31]. We recompute these scores for all algorithms using the published generated images sets from these paper's supplementary files and computed their SIFID[1] scores. Our scores differ slightly from those reported in [10], presumably due to different SIFID implementations. The numerical scores are consistent with the visual inspection: our method is comparable to GPNN in terms of image quality and diversity and both methods outperform SinGAN.

Efficiency: A major disadvantage of the GAN based methods is the need to train a generator and discriminator for every image. As reported in [10] this means that generating a new image of size 180×250 using SinGAN will take about one hour while GPNN (and other methods based on bidirectional similarity) take about two seconds. Our method also does not require any new training and the run times are similar to those of GPNN when the images are small. However, as the size of the image increases, the fact that bidirectional similarity requires computing M^2 distances (where M is the number of patches in each

[1] We used this code from SinGAN's official implementation `https://github.com/tamarott/SinGAN`

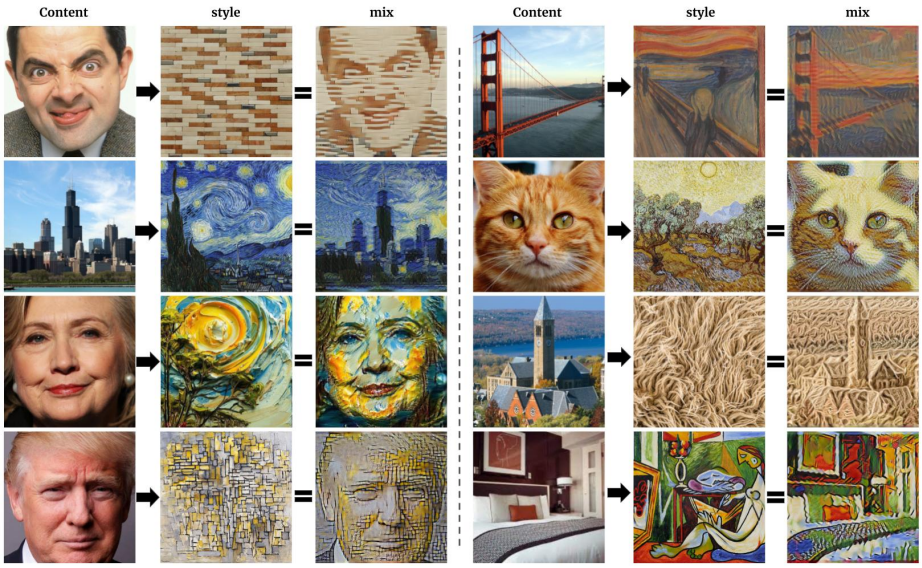

Fig. 6: Results of our method in a style/texture transfer task. Each image triplet shows a content image, a style/texture image and the results of combining them with our synthesis method.

image) means that GPNNs run times grow approximately quadratically and generating a novel image of size 1024×678 on the same computer (NVIDIA GTX-3060TI GPU) with GPNN takes more than half an hour. In contrast, our method has complexity $O(M \log M)$ and takes only 41 seconds for generating an image the same size on the same computer (see figure 7).

Both GPNN and our algorithm are iterative algorithms so there is a natural trade-off between the number of iterations and the the quality of the generated images. More random projections at each optimization step and more optimization steps help the patch-distributions to match more closely, ensuring more realistic outputs. Figure 8 shows this tradeoff by comparing SIFID ([31]) scores and average running times on the SIGD16 dataset compared for different number of SWD random projections. In what follows, we compare the runtimes for a single iteration: in our method a single iteration refers to 64 random projections while in GPNN it refers to a single pass of computing patch nearest neighbors for all patches an image.

The runtimes of methods based on bidirectional similarity can be improved by using approximate nearest neighbor search, rather than exact search. Indeed, for many of our experiments we found that the exact nearest neighbor search can be replaced with approximate nearest neighbor search with almost no loss in quality. But for many of these approximate methods, the run times will continue to grow quadratically. Figure 9 shows the run-times of a single iteration of GPNN and our method for different image sizes when using a CPU. We compare GPNN with

exact nearest neighbors to GPNN when the nearest neighbors are approximated using an inverted index (FAISS [16]). As can be seen, the use of an inverted index speeds up the search substantially, but it continues to grow quadratically and for large images it is outperformed by our method. The advantage of our method is more dramatic when using a GPU since the convolution operation takes full advantage of the GPU. Specifically, on a GTX-3060Ti GPU a single iteration of our algorithm for an image of size 1024×1024 takes 0.032 seconds. While exact nearest neighbor methods can also benefit from GPUs, we found the gain to be smaller and a single iteration of GPNN with exact nearest neighbor for an image of size 1024×1024 still takes more than 5 minutes on the same GTX-3060Ti GPU (i.e. more than 9000 times slower than a single iteration of our method). The open source implementation of approximate nearest neighbors with inverted index [16] failed to run on the same GTX-3060Ti GPU, presumably due to the fact that methods such as inverted indices require operations that are difficult to parallelize [16].

Another disadvantage of approximate nearest neighbor techniques in the context of GPNN is that it is impractical to modify the similarity metric during the image generation process in order to encourage the algorithm to avoid reusing patches. Figure 7 shows an example of high resolution style transfer: while using approximate nearest neighbors (left) speeds up the computation, it creates lower quality images than GPNN with exact nearest neighbor (middle). Our method (right) is much faster and since it explicitly optimizes the distance between patch distributions, does a better job of capturing the style (see figure 6 for the two input images).

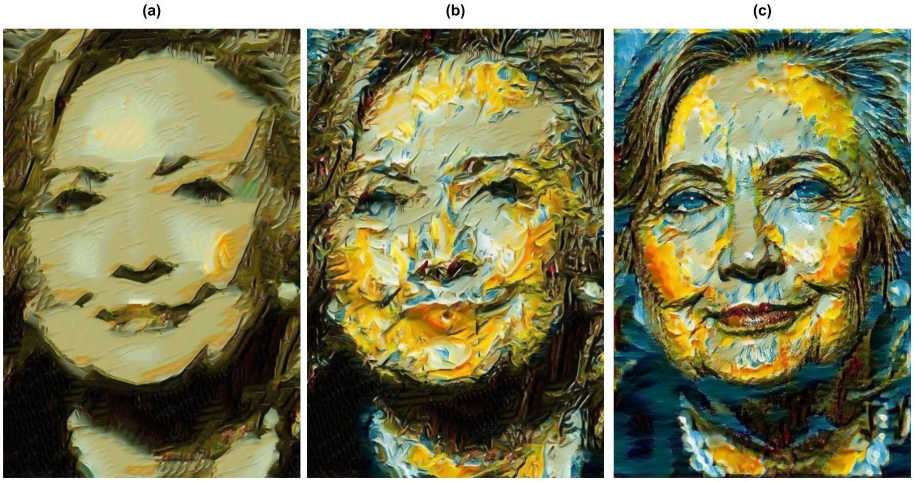

Fig. 7: High resolution style transfer images generated (a) GPNN(α=1) with approximate nearest neighbor (1687 sec, CPU), (b) GPNN(α=0.005) with exact nearest neighbor (1903 seconds, GPU). (c) Our result (41 seconds, GPU). α is the completeness enforcing parameter and α=1 means no constraint. see [10] for details.

Distribution Matching: As mentioned previously, approaches based on BDS do not explicitly optimize the similarity of patch distributions. This means that they may end up synthesizing images where each patch in the synthesized image comes from the input image (and vice versa), but the distribution is nevertheless quite different. Figure 10 shows some examples. In all these cases, GPNN outputs images that are highly coherent and contain only patches from the training image, but the relative frequency of different patches does not reproduce that of the training image and hence some parts may not be reproduced in the generated image (e.g no mountains in the bottom of figure 10). In contrast, our method attempts to reproduce the relative frequency of different patches and therefore it is extremely unlikely that our generated image will fail to reproduce a part of the training image (e.g. as can be seen in figure 1, all images generated by our method for the flock of birds input image contains mountains, sky and birds).

Additional Tasks: Our method and the classical patch-based methods have the advantage that they can be used for many additional tasks without retraining. We show here results on image retargeting, style transfer and texture synthesis.

Figure 5 show the results of our method in retargeting images into various aspect ratios. We start the optimization by resizing the target image's coarsest pyramid level to match the desired aspect ratio. In order to match images of different size using SWD we duplicated patches of the smaller image so that we compare the same number of patches.

Figure 6 shows results of style transfer: triplets of content, style and the mixed output generated by our method. These results are surprising due to the simplicity of our method. In this task we use a single-scale configuration and a patch size of 11. We start the optimization from the content image and match the patch distribution to the style image.

We also applied our method to generating texture images from texture samples. This is done similarly to retargeting but the initial condition is a noise image and the patch size may be different. Image editing can be performed by first crudely editing the image and then using our algorithm to harmonize its fine details so that it looks real. Figure 1 shows examples of texture synthesis and image editing.

5 Related work

The idea of synthesizing realistic images using patches from a target image goes back to Efros and Leung [6]. Efros and Freeman [5] and Hertzmann et al [13] extended this idea to style transfer and other tasks. These classical, nonparametric approaches required finding patch nearest neighbors in high dimensions and a great deal of subsequent work attempted to make the search for nearest neighbors more efficient. The PatchMatch algorithm [2] pointed out that the coherence of patches in natural images can be used to greatly speed up the search for nearest neighbors and this enabled the use of these nonparametric

technqiques in real-time image editing. Our work is very much inspired by these classical papers but we use the recently developed SWD to directly optimize the similarity of patch distributions without computing nearest neighbors. Furthermore, in our approach patches in the synthesized image are not constrained to be direct copies of patches in the training image.

As mentioned in the introduction, a key insight behind successful patch-based methods is the use of some form of bidirectional similarity [33]: it is not enough to require that patches in the generated image be similar to patches in the training image. While [33] optimized the BDS directly, the GPNN approach [10] rewards bidirectional similarity indirectly by modifying the similarity measure to penalize patches that have already been used. In a parallel line of work [[21], [18], [15], [7]] attempts are made to make a uniform use of all patches in the source image through patch histogram matching. Our approach optimizes a well-understood similarity (the Sliced Wasserstein Distance) that is guaranteed to be zero only if the two distributions are equal and importantly it optimizes this similarity very efficiently.

SINGAN and InGAN [31],[32] are both variants of GANs that are trained on a single image with a patch based discriminator. Thus they can be seen as approximately matching the distribution of patches in the generated image and the target image. Our approach directly optimizes the similarity between patch distributions in the two images, requires no training and provides superior quality results.

The Sliced Wasserstein Distance was used in a number of image generation tasks but it is most often used to estimate the distance between distributions of full images [17]. Thus [4] train a generative model by replacing the GAN objective with an SWD objective. In contrast, here our focus is on estimating the distance between two patch distributions and we have shown that an unbiased estimate of the distance between patch distributions in the two images can be estimated using a single convolution.

Although not immediately apparent, neural style transfer [8] can be seen as a single image generative model that preserves patch distribution. [24] showed that the Gram loss in neural style transfer is equivalent to MMD [11] over features of intermediate layers of VGG hence the style transfer objective is to minimize distribution of patches (in size of the layers' receptive fields) between the optimized image and the style target. [12] too had recognized this nature of the Gram loss and suggested replacing it with SWD between the VGG features. Similarly [25], [28] and [23] use the closed form of the Optimal transport between Gaussians to push the spatial distribution in VGG feature maps of a content image into that of a style image and then decode it as a mix image.

The works of [22], [26] closely relate to the neural patch distribution for style transfer. They both suggest objectives that focus on the coherence of neural patches in the synthesized image. Similar to[10], they enforce NN similarity of neural-patches but they compute similarity in a pre-trained neural network rather than in pixel space. Similar to the classic algorithms, both of these work require explicit computation of patch nearest neighbors. In contrast, our use of

SWD allows avoiding the computation of patch nearest neighbors which make our algorithm much faster.

[30] also use SWD for texture synthesis. They minimize SWD on wavelet coefficients with SGD to synthesize new samples from a given texture image. Our paper differs from theirs in that we work in multiple scales, on pixel-level and compute SWD in a more efficient way. We are able thus to produce much better results and to apply our method to more complicated generative tasks.

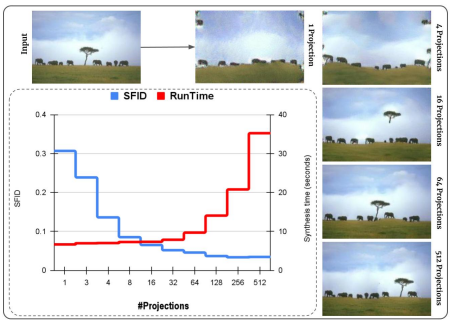

Fig. 8: The tradeoff between image quality and computation cost. We reshuffled the images of the SIGD16 dataset using SWD with k random projections for various values of k. For each such generated set we computed its SIFID compared to the original images and the average time it took to generate one image. The images show generation of one of the images for some values of k.

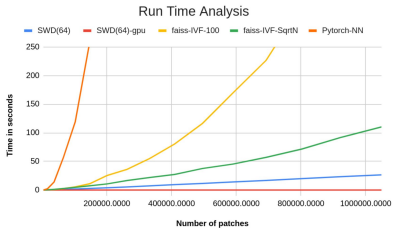

Fig. 9: Computation time (in seconds) of a single SWD (with 64 projections), Nearest neighbor and approximated nearest neighbor (100 and $\sqrt{M}$ bins) steps for different number of patches. The run-time of SWD ($MlogM$) becomes significantly shorter as the number of patches grows compared to NN (M^2).

6 Limitations and Extensions

Perhaps the biggest drawback of our method is the fact that it can only generate images with the same patch distribution as the input. For example, for an image of black balls on a grass background, all samples from our method will have the same number of balls.

The effects of the above can be seen in the image editing task. When the task is to change the amount of pixels from one texture/color and say, for example, add a block of leaves to a tree our method will shrink the size of the tree in order to have the same amount of leaves in the image.

An extension of our method that can address this drawback is to allow the user to explicitly manipulate the distribution over patches that is matched by our algorithm. Figure 11 illustrates this extension. Here the user first manipulates the target distribution of patches by indicating that certain patches should be increased in frequency and the algorithm then generates an image to match target distribution. As can be seen in figure 11, this simple modification allows us to generate images with different number of objects based on the user's preference.

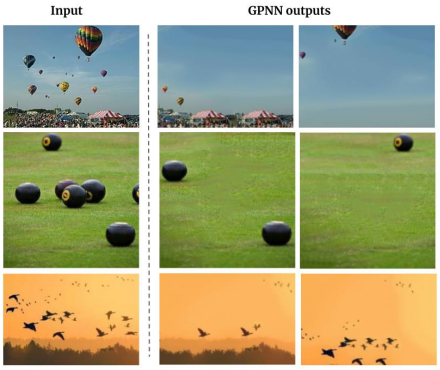

Fig. 10: Image reshuffling results generated with GPNN. While the generated images are realistic and coherent, they do not preserve the patch distribution and hence significant parts of the input image are not reproduced. Our method on the other hand, explicitly matches the patch distribution and therefore avoids such failures (see figure 1 for the results of our method on the same inputs.)

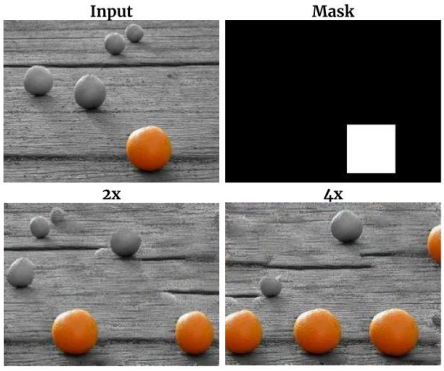

Fig. 11: An extension of our algorithm allows the user to supply a mask as an additional input to specify target patches whose frequency should be increased. The rest of the algorithm is exactly the same and this allows the user to manipulate the number of objects in the generated images.

7 Conclusion

Classical approaches for generating realistic images are based on the insight that if we can match the distribution of patches between the generated image and the target image then we will have a realistic image. In this paper we have used the same insight but with a novel twist. We use the Sliced Wasserstein Distance which was developed as a method to compare distributions of full images when training generative models and we showed that when applied to distributions of patches, an unbiased estimate of the SWD can be computed with a single convolution of the two images. Our experiments shows that this unbiased estimate is sufficient for excellent performance in a wide range of image generation tasks.

References

1. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: International conference on machine learning. pp. 214–223. PMLR (2017)
2. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: Patchmatch: A randomized correspondence algorithm for structural image editing. ACM Trans. Graph. **28**(3), 24 (2009)
3. Bonneel, N., Rabin, J., Peyré, G., Pfister, H.: Sliced and radon wasserstein barycenters of measures. Journal of Mathematical Imaging and Vision **51**(1), 22–45 (2015)
4. Deshpande, I., Zhang, Z., Schwing, A.G.: Generative modeling using the sliced wasserstein distance. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3483–3491 (2018)
5. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 341–346 (2001)
6. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: Proceedings of the seventh IEEE international conference on computer vision. vol. 2, pp. 1033–1038. IEEE (1999)
7. Fišer, J., Jamriška, O., Lukáč, M., Shechtman, E., Asente, P., Lu, J., Sỳkora, D.: Stylit: illumination-guided example-based stylization of 3d renderings. ACM Transactions on Graphics (TOG) **35**(4), 1–11 (2016)
8. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2414–2423 (2016)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in neural information processing systems **27** (2014)
10. Granot, N., Feinstein, B., Shocher, A., Bagon, S., Irani, M.: Drop the gan: In defense of patches nearest neighbors as single image generative models. arXiv preprint arXiv:2103.15545 (2021)
11. Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., Smola, A.: A kernel method for the two-sample-problem. Advances in neural information processing systems **19**, 513–520 (2006)
12. Heitz, E., Vanhoey, K., Chambon, T., Belcour, L.: A sliced wasserstein loss for neural texture synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9412–9420 (2021)
13. Hertzmann, A., Jacobs, C.E., Oliver, N., Curless, B., Salesin, D.H.: Image analogies. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 327–340 (2001)
14. Iizuka, S., Simo-Serra, E., Ishikawa, H.: Globally and locally consistent image completion. ACM Transactions on Graphics (ToG) **36**(4), 1–14 (2017)
15. Jamriška, O., Fišer, J., Asente, P., Lu, J., Shechtman, E., Sỳkora, D.: Lazyfluids: appearance transfer for fluid animations. ACM Transactions on Graphics (TOG) **34**(4), 1–10 (2015)
16. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with gpus. IEEE Transactions on Big Data **7**(3), 535–547 (2019)
17. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)
18. Kaspar, A., Neubert, B., Lischinski, D., Pauly, M., Kopf, J.: Self tuning texture optimization. In: Computer Graphics Forum. vol. 34, pp. 349–359. Wiley Online Library (2015)

19. Kolkin, N., Salavon, J., Shakhnarovich, G.: Style transfer by relaxed optimal transport and self-similarity. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10051–10060 (2019)
20. Kolouri, S., Pope, P.E., Martin, C.E., Rohde, G.K.: Sliced-wasserstein autoencoder: An embarrassingly simple generative model. arXiv preprint arXiv:1804.01947 (2018)
21. Kopf, J., Fu, C.W., Cohen-Or, D., Deussen, O., Lischinski, D., Wong, T.T.: Solid texture synthesis from 2d exemplars. In: ACM SIGGRAPH 2007 papers, pp. 2–es (2007)
22. Li, C., Wand, M.: Combining markov random fields and convolutional neural networks for image synthesis. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2479–2486 (2016)
23. Li, P., Zhao, L., Xu, D., Lu, D.: Optimal transport of deep feature for image style transfer. In: Proceedings of the 2019 4th International Conference on Multimedia Systems and Signal Processing. pp. 167–171 (2019)
24. Li, Y., Wang, N., Liu, J., Hou, X.: Demystifying neural style transfer. arXiv preprint arXiv:1701.01036 (2017)
25. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Universal style transfer via feature transforms. Advances in neural information processing systems **30** (2017)
26. Mechrez, R., Talmi, I., Zelnik-Manor, L.: The contextual loss for image transformation with non-aligned data. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 768–783 (2018)
27. Michaeli, T., Irani, M.: Blind deblurring using internal patch recurrence. In: European conference on computer vision. pp. 783–798. Springer (2014)
28. Mroueh, Y.: Wasserstein style transfer. arXiv preprint arXiv:1905.12828 (2019)
29. Pitie, F., Kokaram, A.C., Dahyot, R.: N-dimensional probability density function transfer and its application to color transfer. In: Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1. vol. 2, pp. 1434–1439. IEEE (2005)
30. Rabin, J., Peyré, G., Delon, J., Bernot, M.: Wasserstein barycenter and its application to texture mixing. In: International Conference on Scale Space and Variational Methods in Computer Vision. pp. 435–446. Springer (2011)
31. Shaham, T.R., Dekel, T., Michaeli, T.: Singan: Learning a generative model from a single natural image. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4570–4580 (2019)
32. Shocher, A., Bagon, S., Isola, P., Irani, M.: Ingan: Capturing and retargeting the" dna" of a natural image. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4492–4501 (2019)
33. Simakov, D., Caspi, Y., Shechtman, E., Irani, M.: Summarizing visual data using bidirectional similarity. In: 2008 IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–8. IEEE (2008)
34. Srivastava, A., Valkov, L., Russell, C., Gutmann, M.U., Sutton, C.: Veegan: Reducing mode collapse in gans using implicit variational learning. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 3310–3320 (2017)
35. Wang, T.C., Liu, M.Y., Zhu, J.Y., Tao, A., Kautz, J., Catanzaro, B.: High-resolution image synthesis and semantic manipulation with conditional gans. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)

Supplemental material

Ariel Elnekave and Yair Weiss

Hebrew University Jerusalem
Ariel.Elnekave@mail.huji.ac.il, Yair.Weiss@mail.huji.ac.il

This document contains supplementary material on our method for Generating natural images by Patch Distribution Matching (GPDM).

1 Python implementation

Our python implementation of GPDM as described in the paper is available in the 'GPDM' folder alongside this document. All GPDM results generated in this document and in the main paper are generated via the python scripts in the 'scripts' sub-folder. Please Consult the Readme.md file for additional instructions.

2 Method parameters description

In this section we describe our method's configurations in more details As detailed in the paper. GPDM is an iterative multi-scale process for generating images and has multiple configurations controlling the output given the inputs.

pyramid_factor and coarse_dimension: These parameters control the number and sizes of the different scales in which the GPDM works. The input image is repeatedly scaled down by **pyramid_factor** until scaling it will result in one of its dimensions to be less than **coarse_dimension**. All the intermediate images are used as the multi scale pyramid on which the algorithm works.

scale_factors: This parameter defines the aspect ratio of the output. It is relevant mostly for the texture synthesis and retargeting tasks as in all other tasks the output image should have the same size as the input.

init_mode and noise sigma: These parameters affect the first initial guess of the algorithm. **init_mode** can instruct the algorithm to start the optimization from a blank image, from the target image or from another image. **noise_sigma** defines the amount of gaussian pixel noise added to the initial guess in order to increase variability.

learning_rate and num_optimization_steps: These parameters affect the optimization of the synthesis image at each scale. specifically it defines the number of SGD steps and the step size used for optimizing SWD between the images.

patch_size, stride and num_projections: These parameters control the SWD computation between patches in two images. **patch_size** and **stride** control the way patches are extracted from an image. **num_projection** defines the number of projections used to estimate SWD.

3 Task specific configurations and results

In this section we describe the actual configurations we used for each of the tasks we talk about in the paper. We also add additional result images for each task generated with the noted configuration.

3.1 Image reshuffling

For the task of reshufling images from the SIGD16, Places50 datasets we used the following configuration:

pyramid_factor=0.85, coarse_dim=28, scale_factors=(1,1), init_mode='zeros', noise_sigma=1.5, patch_size=7, stride=1, num_projections=64, learning_rate=0.05, num_optimization_steps=300

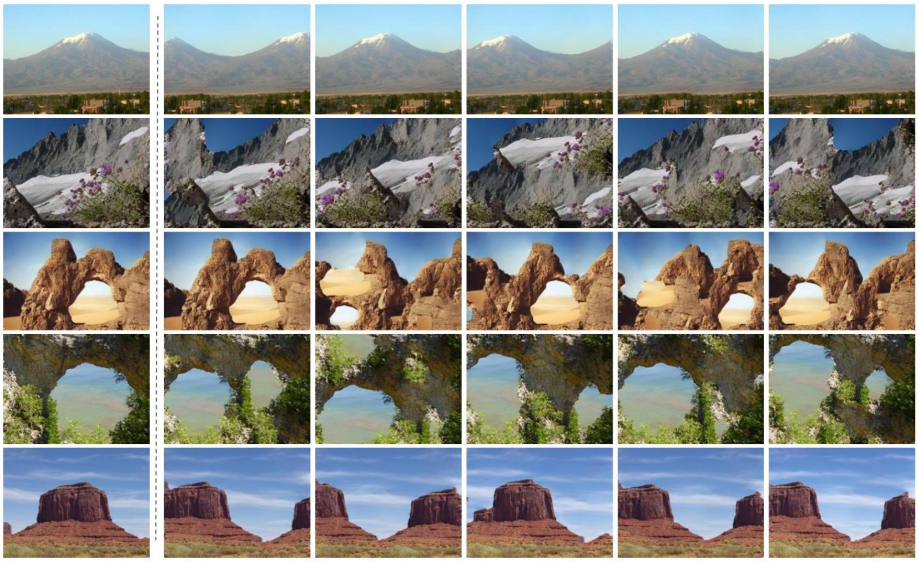

Fig. 1: Reshuffling of images from the SIGD15 dataset. Inputs on the left.

3.2 Image retargeting

As mentioned before, unlike in reshuffling, retargeting generates an output in different size thatn the output. For that purpose we set the scale-factor parameter as desired, i.e (1,2) for an output which is twice as wide from the input. Another difference in the retargeting task is that we aim for less variability and more coherence in the output. For that reason we start the optimization from an

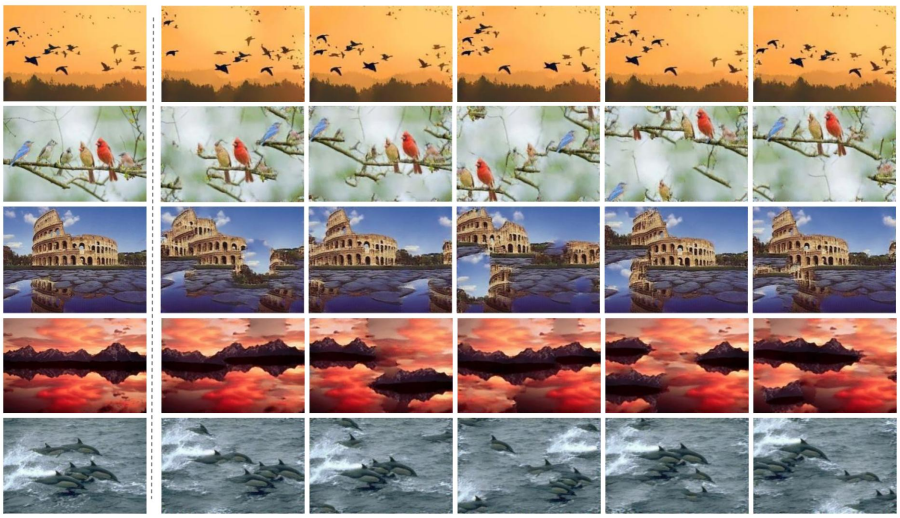

Fig. 2: Reshuffling of images from the Places50 dataset. Inputs on the left.

image resized to a different aspect ratio defined by the parameter scale-factor, blur it, (init_mode='blurred_target') and add no pixel noise (noise_sigma=0). Figures [3, 5, 4] show some additional retargeting results. Each batch of images shows the input on the upper left and results where the scale factor parameter is set to (2,2) (1,2), (2,1) in a clockwise order from it. The other parameters used for creating these images are:

pyramid_factor=0.85, coarse_dim=35
init_mode='blurred_target', noise_sigma=0, patch_size=7, stride=1,
num_projections=128, learning_rate=0.05,
num_optimization_steps=300

3.3 Image style transfer

As described in the paper the way we perform style transfer is by starting the optimization process from a content image and use the style image as the target. We found that the best outputs are generated using a single pyramid level and using a patch size of 11. Of course the best parameters may change from image to image. Figure 6 shows generation of a high resolution image of size 1024x642 with two different texture "styles". We also show the output when using multiple pyramid level for reference. These are the parameters we used for style transfer:

pyramid_factor=1, coarse_dim=max_dim, scale_factors=(1,1),
init_mode=content_image, noise_sigma=0, patch_size=11, stride=1,
num_projections=64, learning_rate=0.05,
num_optimization_steps=300

Fig. 3: Retargeting of famous paintings.

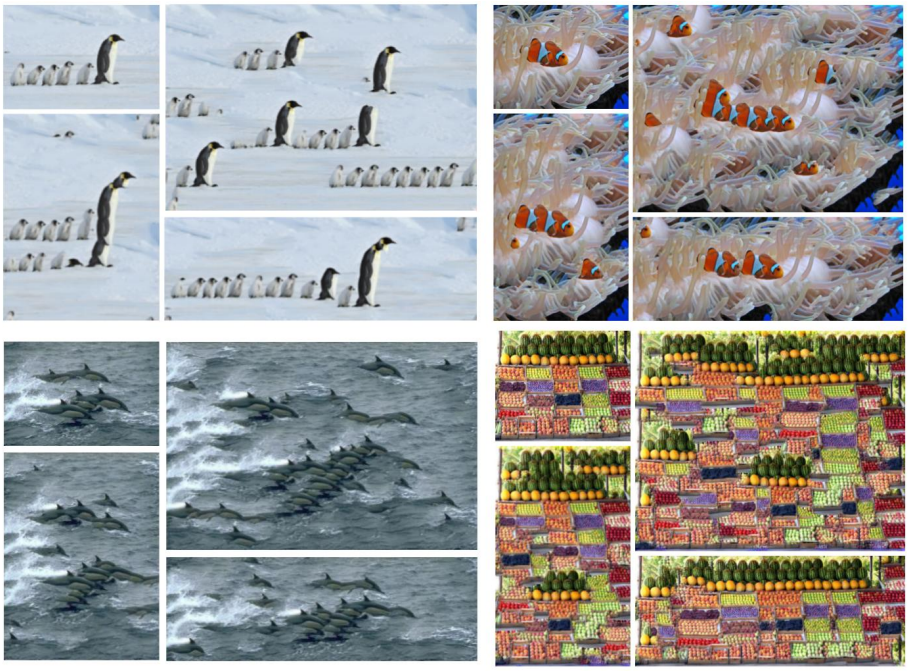

Fig. 4: Retargeting of some selected images from previous papers.

Fig. 5: Less sucsessfull examples of retargeting

Fig. 6: Image style transfer of an HQ image into two different styles/textures and with two different values for the parameter coarse_dim. The right-most column shows results when using multiple pyramid level (specifically coarse_dim=512) and the second column from the right is showing the results with a single pyramid level (coarse_dim=1024).

3.4 Image texture synthesis

The examples in the papers together with the results on figures 7 are all generated with a scale_factors=(2,2) and with same other parameters as in the reshuffling task.

Fig. 7: Results of GPDM on texture synthesis creating a twice as big a texture from a given sample that shows in the upper left cornet of each image.

4 GPNN with Approximate nearest neighbor

In the paper we discussed the possibility to use approximate nearest neighbor (ANN) methods in order to increase GPNN's speed. The main problem with this approach is that it is much harder to control the completeness of the output with such methods. As mentioned in the text, GPNN approximately optimizes a bidirectional loss by adding a term to the patch distance that penalizes patches that have already been used. Specifically, GPNN chooses a patch that minimizes:

$$S_{ij} = \frac{D_{ij}}{\alpha + \min_l D_{lj}}$$

As written in the GPNN paper " The parameter α is used as a knob to control the degree of completeness, where small α encourages completness, and $\alpha >> 1$ is essentially the same as using MSE.".

α is set to 0.005 in the standard GPNN (the penalty is harder for smaller values of α). When using approximate nearest neighbor we just use MSE (this is equivalent to $\alpha >> 1$).

Here we add more results from the experiment [1] described in figure 7 of the paper showing the affect of using GPNN with approximate nearest neighbor calculations. Figure 8 shows the affect of using ANN in the reshuffling task. The optimization process in this figure starts from a blurred version of the target (init-mode='blurred_target') which makes exact-NN more prone to selecting smooth patches. The figure shows the crucial effect of the α parameter on the completeness of the outputs but also that using ANN leads to very similar result to exact-NN when α is not used.

In Figure 9 we show the same affect in the style transfer task with GPNN. Notice how using α makes the result contain more details from the style image. Moreover using ANN does not seem to have a negative affect compared to exact-NN without the α parameter.

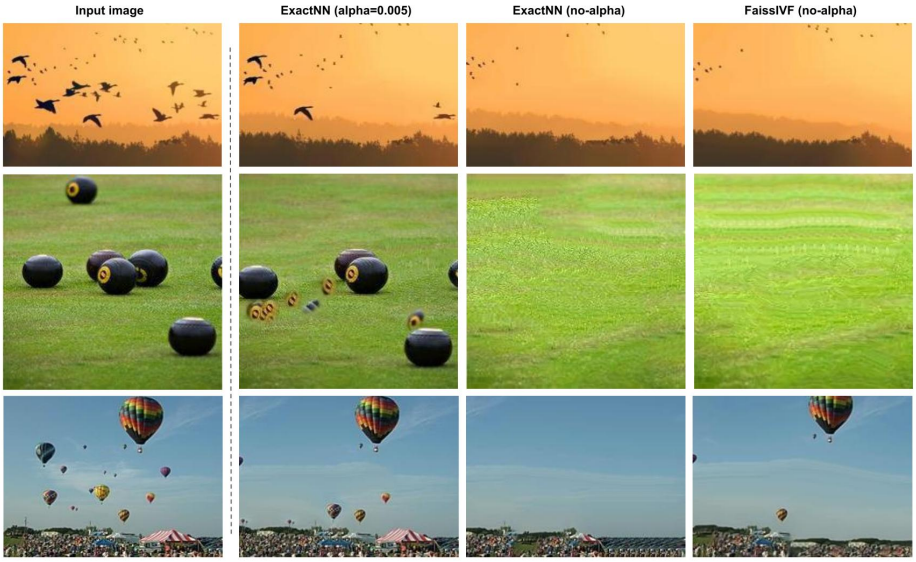

Fig. 8: GPNN reushuffling results on images from the SIGD dataset with different NN computation methods.

[1] We used this implementation for all our experiments with GPNN https://github.com/ariel415el/Efficient-GPNN

5 Run-time analysis

We add here, in figure 10, the full run-time analysis graph from the paper including the running time of exact NN with pytorch on GPU so that readers can appreciate the difference in the affect of using GPU between SWD and exact-NN computations with pytorch.

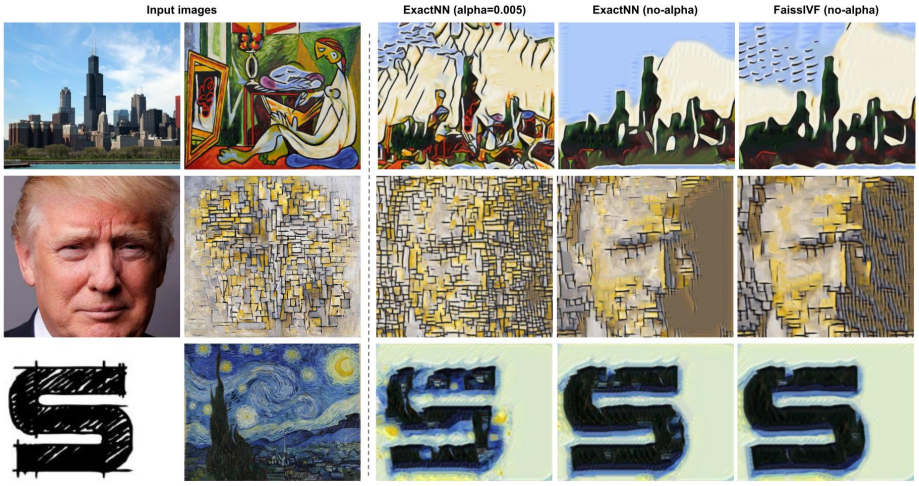

Fig. 9: GPNN style transfer results on images from the SIGD dataset with different NN computation methods. The input content and style images are on the left of each row.

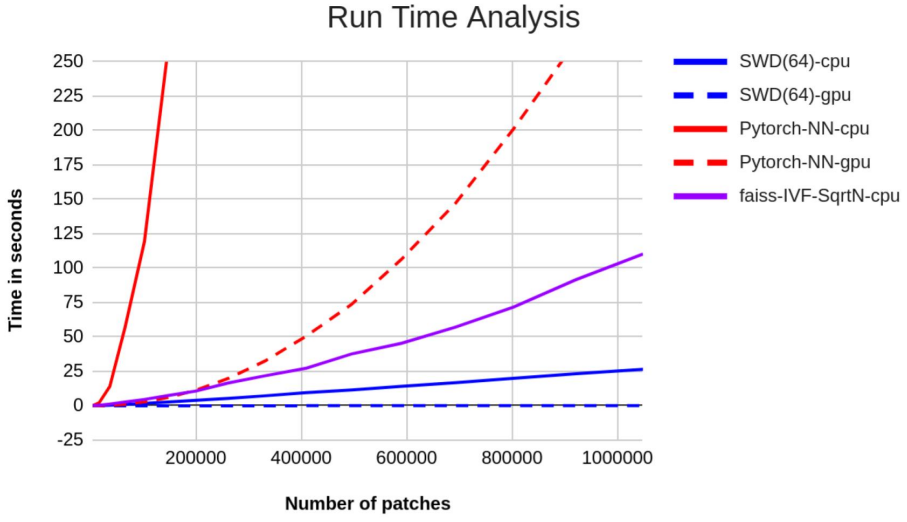

Fig. 10: Run time analysis table.