Transformation-based Image Interpolation

Yaroslav Ganin
MILA
Université de Montréal
yaroslav.ganin@gmail.com

1 Introduction

In this project, we are concerned with traversal of natural image manifolds. This task has several important applications including generation and editing and also reveals interesting relationships between images. It seems that in the unsupervised setting (i. e.no extra annotation is given apart from the images themselves), one of the most common approaches is to build an auto-encoder-based generative model mapping samples from a simple low-dimensional distribution (e. g., isotropic Gaussian $\mathcal{N}(0, I)$) into the image space (and vice versa). The latent space is assumed to represent a linearized version on the data manifold, and the exploration is usually performed by moving along the straight lines connecting pairs latent codes or following some precomputed direction [13]. While explicit modeling of the latent space has its merits, e. g., low-dimensional representations are very useful for semi-supervised learning, in some cases it may hinder the ability of the model to reconstruct inputs precisely. This is due to the simple fact that modern neural networks, albeit very powerful, still have limited capacity. The consequence of this is that manipulated images lose some amount of details or become altered in various unexpected ways (e. g., faces sometimes lose their identites). Such effects pose a serious obstacle for applying auto-encoder architectures for tasks requiring high level of photorealism.

We argue that for the particular use-case of the manifold traversal, explicit mapping into the latent space is not a necessary condition. Our key observation is that it is possible to explore the low-dimensional image manifold without attempting to linearize it and therefore sidestepping complications faced by conventional approaches. We propose a flexible transition model that operates in the pixel-space and therefore can be easily enchanced by domain knowledge. The model can be divided into two submodules: the first one observes a pair of images (the current point and the point we'd like to end up in eventually, not necessarily in one step) and produces a compact command that is then used by the second submodule to modify the current point thus transferring to a next point on the manifold. We give several possible types of transformations that are reasonable for natural images and can be easily implemented by fully-differentiable neural network.

We present several strategies for training our model and describe their advantags and drawbacks. In particular, we frame the task as sequential decision making and suggest that we can employ techinques developed in reinforcement learning. Finally, we test our approach on several datasets and present preliminatry results demonstrating the potential of the idea.

2 Related work

This section is still WIP. Some papers to mention:

- Sequential image generation by Bachman and Precup [1]
- DeepWarp [4]
- [12]
- [17]

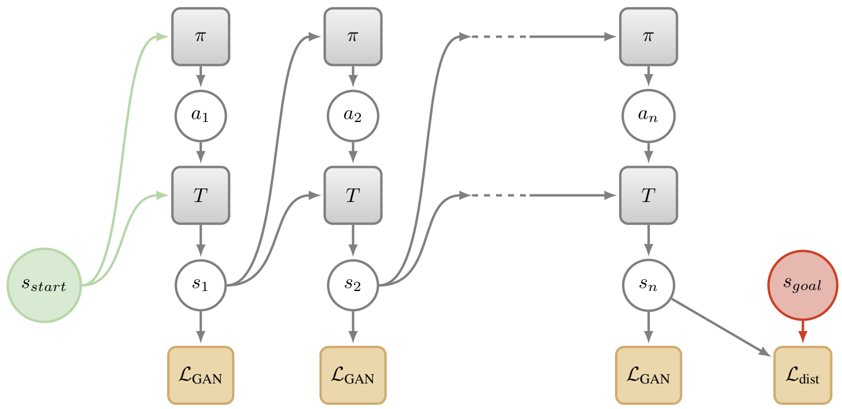


Figure 1: The **proposed model**.

3 Method

In this section, we formalize the task we are solving and give a high-level description of our model. We are going to use the terminology from *reinforcement learning* as it seems very natural here. Let us consider a distribution P_d defined over a set of images $\mathcal{S} = [0, 1]^{H\times W\times C}$ constituting the *state space* of the *agent*, where H, W, C are height, width and number of channels respectively. The agent is capable of performing actions from the *action space* $\mathcal{A}(s)$ consisting of d-dimensional vectors which can have both continuous and discrete components. The transition between states is performed by means of operator T which we call *the transformer*:

$$s' = T(s, a)\,, \quad s, s' \in \mathcal{S},\ a \in \mathcal{A}(s)\,. \tag{1}$$

We seek to find a family of *policies* $\pi(\cdot \mid s; s_{goal})$ parametrized by $s_{goal} \in \mathcal{S}$ bringing the agent from any $s_{start} \sim P_d$ to any given $s_{goal} \sim P_d$ while maximizing the expected reward along the trajectory.

We employ four kinds of reward signals:

- A fixed negative reward for each transition in order to promote short trajectories.
- A fixed negative reward if the agent exceeds the maximum allowed number of steps (may not be necessary).
- A reward that depends on how close the current state is to the image manifold (as seen by a dedicated discriminator model [5]).
- A reward that depends on how close the agent to the goal after it has taken the last step. This reward is inversely proportional to either L_1/L_2 distance or something more elaborate like *perceptual distance* [3, 8].

The full model is shown in Figure 1.

The formulation above allows us to solve the task by applying an off-the-shelf reinforcement learning algorithm (e. g., REINFORCE [18]). Unfortunately, for most of the datasets, it's extremely hard to hand-design a suitable operator T. We therefore relax the setting and consider a multi-agent setup where both π and T are agents with learned policies. The state space of T is comprised of tuples of the form (s, a), where $s \in \mathcal{S},\ a \in \mathcal{A}(s)$. Perhaps a bit unconventionally, T produces actions that are nothing else than next states of π (i. e., transformed images). One can reduce this new setup to a single-agent case by fixing either of the agents while updating the other one.

4 Experiments

In this section, we detail the training strategies used in the experiments, give a full description of the network architectures and, finally, present results on several datasets. If not stated otherwise, both π and T sample actions from the isotropic Gaussian distributions. Addtionally, the policy network π has a single Bernoulli output unit responsible for termination of the episode.

4.1 Transformations

One interesting aspect of the proposed method is that we can easily incorporate domain knowledge into the network architecture. This is in contrast to conventional latent-to-image decoder networks, which are mostly treated as black boxes. In case of transformer networks, one could restrict the set of possible operations that the model is capable of performing. We propose the following generic list of transformations typically observed in natural images:

1. **Warping.** This is a geometric transformation based on bilinear sampling. It has been recently used for a broad range of computer vision tasks such as fine-grained classfication [7] and image resynthesis [4, 12]. Formally, the intensity of the transformed image O at location (x, y) for channel c (either R, G or B) is computed as
$$O(x, y, c) = I\{x + \mathbf{F}(x, y, 1), y + \mathbf{F}(x, y, 2), c\}\,, \tag{2}$$
where $\mathbf{F}$ is a two-channel displacement map (flow field), and the curly braces denote bilinearly interpolated intensity of the input $I(\cdot, \cdot, c)$ at a real-valued position. In our experiments, $\mathbf{F}(x, y, 1)$ is computed by a neural network.
2. **Affine recoloring.** This transformation is useful when one needs to coherently change color of a large group of pixels (e. g., in order to manipulate the time of day in the photos [15]). We define it as:
$$O(x, y, \cdot) = \alpha_{AR}(x, y)I(x, y, \cdot) + [1 - \alpha_{AR}(x, y)]\,[A \cdot I(x, y, \cdot) + b]\;, \tag{3}$$
where a 3×3 matrix A and a 3-dimensional vector b constitute the parameters of the affine transformation. The mask α_{AR} is responsible for selecting pixels to be recolored. Both α_{AR} and (A, b) can be, again, predicted by a network.
3. **Blending with direct synthesis.** The most general transformation is a direct synthesis of RGB values for each spatial location. It does not explicitly reuse input pixels which can potentially seen as a drawback but, on the other hand, unlike the transformations above, it is capable of producing novel objects. If D is a directly synthesized image, we define the full transformation as:
$$O(x, y, \cdot) = \alpha_D(x, y)I(x, y, \cdot) + [1 - \alpha_D(x, y)]\,D(x, y, \cdot)\,, \tag{4}$$
where the mask α_D controls which pixels get replaced by the corresponding values from D.

4.2 Datasets

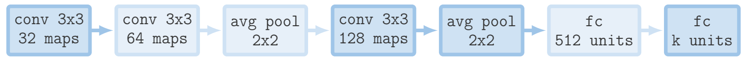
We conduct the experiments on two image datasets. The first one is MNIST [10] which we use mainly as a sanity check for the proposed approach. The dataset contains $60{,}000$ grayscale images of hand-written digits of size 28×28. This dataset is quite simple, so we expect that warping alone should be sufficient for traversing the manifold.

The second dataset is far more challenging. The CELEB-A [11] is comprised of $202{,}599$ photos of celebrities (predominantly headshots). We use a script supplied with the `Fuel`-framework [16] to crop 64×64 face-centered images at roughly the same scale. It should be noted that the resulting training samples still contain a fair amount of variation due to different lighting conditions and view angles. For that reason, we are applying all three types of transformations listed above in the Celeb-A experiment.

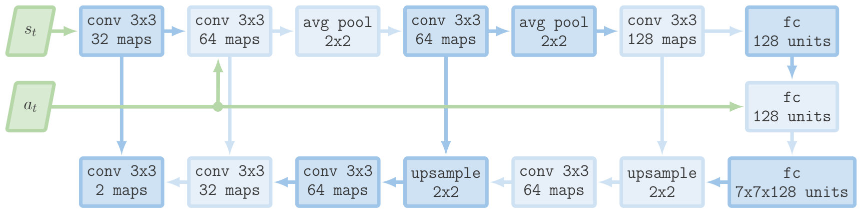
4.3 Network architecture

Figure 2 shows the architectures that was used in the MNIST experiment. Overall, our policy network π and the discriminator resemble the discriminator from [13]. The transformer is inspired by recent image processing architectures that are designed to aggregate features from large receptive fields while preserving input resolution, e. g., U-Nets [14]. These networks are relatively easy to train due to improved gradient flow promoted by skip-connections and are shown to produce high-quality images [12]. The output of T is a $H \times W \times 2$ tensor corresponding to the flow field $\mathbf{F}$. In case of a stochastic transformer, we T also produces a matrix of standard deviations, much like in variational auto-encoders [9].

We use similar architectures (but with a larger number of layers and feature maps) for the CELEB-A experiments as well.



(a) Architecture of π, the value network and the discriminator. The inputs and the number of the output units `k` depend on the purpose of the network.



(b) Architecture of the transformer network T. In the simplest case, T produces a flow field $\mathbf{F}$

Figure 2: The architectures used in the MNIST experiments.

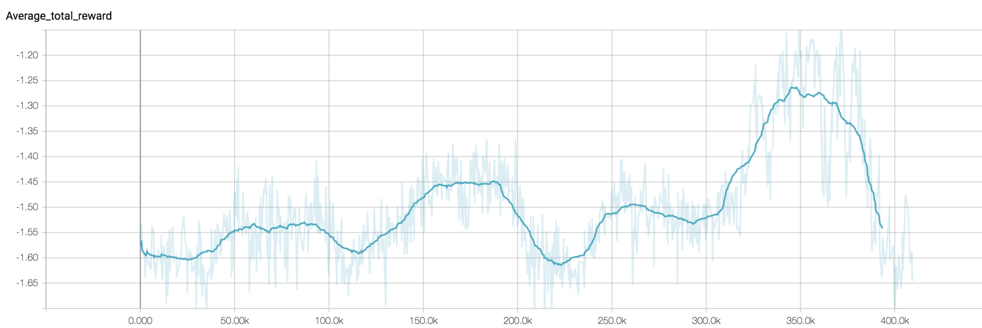


Figure 3: Instability of the hybrid model. Vertical axis corresponds to the batch-averaged return, horizontal axis corresponds to the number of training steps.

4.4 Training procedures

In our experiments, we employ several approaches to train the architectures above. We give descriptions for each of them below.

Policy gradients (PG). One of the less restrictive ways to optimize the parameters of π and T is, as we already mentioned, to use Monte-Carlo policy gradients [18]. For brevity, we are going to focus on π but the same reasoning applies to T as well. Formally, we would like to maximize the following performance measure:

$$\eta(\theta) = \mathbb{E}_{s_{start}, s_{goal} \sim P_d} \left[v_{\pi_\theta(\cdot; s_{goal})}(s_{start}) \right] \,, \tag{5}$$

where $v_{\pi_\theta(\cdot \,|\, \cdot; s_{goal})}$ is the true value function for $\pi(\cdot; s_{goal})$ depending on the learnable parameters θ. Direct application of the *policy gradient theorem* gives us a gradient estimate for θ:

$$\nabla_\theta \eta(\theta) \approx G_t \cdot \nabla_\theta \log \pi(a_t \,|\, s_t; s_{goal}) \,, \tag{6}$$

where (s_t, a_t) is the state and the action taken by the agent at timestep t, and G_t is the corresponding return obtained along the trajectory starting with s_t. We can see that (6) allows us to use discrete actions as well as non-differentiable rewards (note that we only need the value of G_t but not its gradient).

In practice, (6) has a very high-variance and it's almost useless for deep neural networks. A common remedy for that problem is to incorporate a state-dependent (but not action-dependent) baseline, i. e., replace G_t with $G_t - \beta(s_t; s_{goal})$ also called *advantage*. In our experiments, we set $\beta(s_t; s_{goal})$ to be a value-function estimated by yet another NN $V(s_t; s_{goal})$ of the same architecture as π except for the output unit. That network is trained to minimize $||V(s_t; s_{goal}) - G_t||^2$ (we also tried Huber loss but it did not seem to make a lot of difference). One interesting question is whether we can reduce the variance even further by replacing the entire expression in (6) with a prediction of a separate model. That new model can be trained much like a baseline (but with vector targets) and used a source of synthetic gradients [6, 2]. We leave this prospect for the future work.

Dealing with the stochastic transformer. Training the transformer with regular policy gradient requires having stochastic output units. While this is not a big issue for the policy network, injecting isotropic noise to images (predictions of the transformer) inevitably sends them off the data manifold. We could try avoiding this by performing sampling in the space of flow fields but the naive way would have even more disastorous consequences: a flow field with an additive Gaussian noise effectively tears the image apart. In either case, the root of the problems is a lack of spatial correlations but dealing with a full covariance matrix instead of the diagonal one is very cumbersome. We notice that in (6) there is no need to compute the action log-likelihood precisely as we are mainly interested in its gradient. Moreover, policy gradients with the Gaussian likelihood can be interpreted as:

- Pushing the current mean towards a good (in terms of return) sample and away from a bad one (the gradient of the L_2 distance).
- Increasing and decreasing standard deviations so that good samples become more probable and bad samples become less probable (e. g., if we sampled a good action far away from the mean and current standard deviation is small, we should increase it).

Armed with this intuition, we propose the following trick (in the flow field space). Just like before, T predicts $\mu_{\mathbf{F}}$ and $\sigma_{\mathbf{F}}$ but now instead of computing

$$\mathbf{F} = \mu_{\mathbf{F}} + \sigma_{\mathbf{F}} \odot \epsilon\,, \quad \epsilon \sim \mathcal{N}(0, I_{H \times W \times C}) \tag{7}$$

we resize $\sigma_{\mathbf{F}}$ (as if it was an image) into smaller spatial dimensions $H' \times W'$ (e. g., if the original size was 28×28, one could shrink it into 7×7) and perform sampling in that new smaller space. Thus, the complete procedure is:

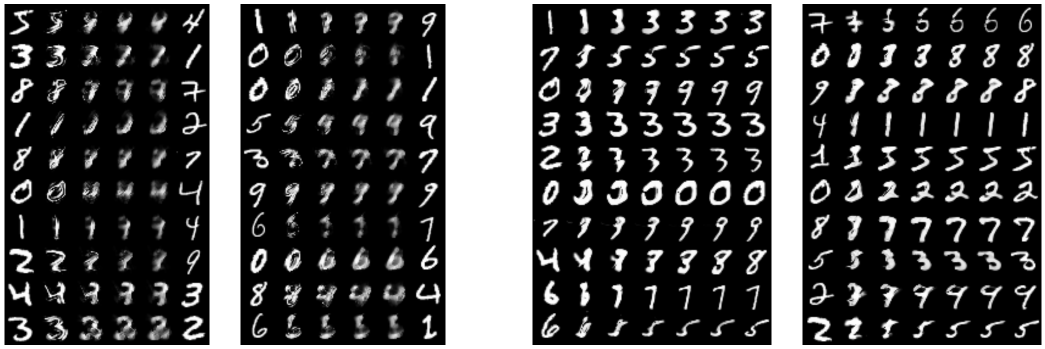
$$\mathbf{F} = \mu_{\mathbf{F}} + \texttt{upsample}(\texttt{downsample}(\sigma_{\mathbf{F}}) \odot \epsilon)\,, \quad \epsilon \sim \mathcal{N}(0, I_{H' \times W' \times C})\,. \tag{8}$$

The net effect of this trick is that now the injected "noise" has strong local spatial correlations due to the scale manipulations (in practice, we use bilinear upsampling). The resulting flow field ends up being distorted in a plausible way and the output image looks locally randomly warped rather than torn apart. Taking the interpretation of the Gaussian likelihood above into account, we compute (6) pretending that $\texttt{upsample}(\texttt{downsample}(\sigma_{\mathbf{F}}) \odot \epsilon)$ was sampled in the original $H \times W$ space using covariance $\text{diag}(\sigma_{\mathbf{F}})$.

BPTT. If we are willing to discard the first two reward signals in the list above (controlling the lenght of the trajectory) and assume that $\mathcal{A}(s)$ contains only continuous vectors, then we can merge π and T into a monolithic feed-forward fully differentiable architecture and maximize the rest of the rewards using backpropagation through time (BPTT). In this setting, we have to fix the number of steps.

One of the initial motivations for using policy gradients even for continuous actions was the fact that PG do not require allocating memory for multiple steps (unlike BPTT). This can be a serious advantage for larger computer vision models [12] — we simply won't be able to store more than 1 or 2 steps in a limited GPU RAM. In this case, truncated BPTT will certainly struggle with proper credit assignment. However, it turns out that some software packages (e. g., `TensorFlow`) are capable of performing a smart swapping of intermediate activations onto CPU RAM so that we don't need to truncate backpropagation, assuming we have enough CPU memory (which is likely to be the case). Nonetheless, BPTT cannot fully replace PG (e. g., in case of discrete actions).

Hybrid training. One might notice that T network does not really need to perform non-differentiable actions. All of the transformations discussed so far allow for the gradient propagation. We, therefore, can attempt training T with BPTT while still updating π with PG (note that we can send the PG signal through time as well). Unfortunately, initial experiments revealed the high level of instability of such a model (see Figure 3). We are hypothesizing that this instability happens when π becomes really confident in actions it's taking (standard deviation approaches zero). We have tried several tricks to sidestep this phenomenon (e. g., advantage value clipping) but did not manage to get the model into the working state. One possible solution which we didn't not have a chance to try might



(a) Policy gradients

(b) Backpropagation through time

Figure 4: Results for PG and BPTT training on MNIST

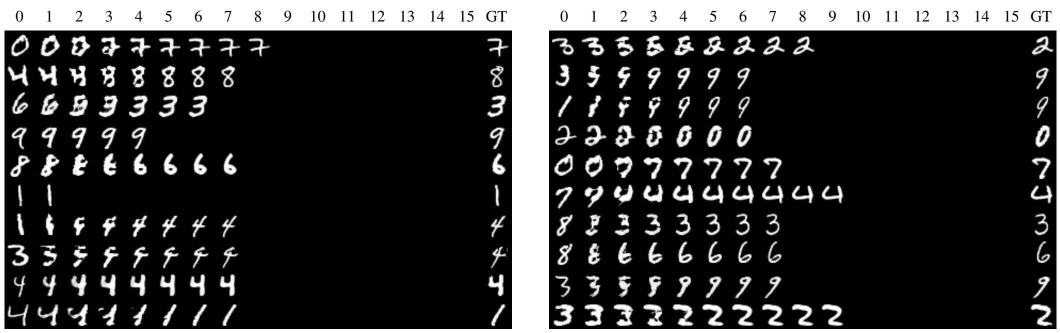


Figure 5: Sample results for the BPTT+STOP model capable of terminating the episode. The first column corresponds to s_{start}, GT (ground-truth) corresponds to s_{goal}. Black cells denote termination of an episode. The agent had a budget of 15 steps.

be to restict the norm of the gradient w.r.t. the parameters of the Gaussian distribtution, i. e.,

$$\delta_{\mu} = \frac{\nabla_{\mu} \log \pi(a_t \mid s_t; s_{goal})}{||\nabla_{\mu} \log \pi(a_t \mid s_t; s_{goal})||} \cdot \min \left[||\nabla_{\mu} \log \pi(a_t \mid s_t; s_{goal})||, \tau \right] , \tag{9}$$

where τ is some predefined threshold. One handles the gradient w.r.t. σ in a similar fashion.

BPTT with a stop action (BPTT+STOP). Finally, we consider a model that is very similar to an RNN trained with BPTT except we don't fix the number of steps but rather let an additional Bernoulli unit attached to π decide when to stop the episode. In practice, we still restrict the maximum number of transformations but the agent is encouraged to terminate as early as possible as we are now penalizing the length of the trajectory. In this setting we use PG only for the binary output.

4.5 MNIST

For the MNIST dataset, we employed all four training strategies listed above. Unfortunately, PG did not work for the full model (we observed a lot of instability), so we had to disable the Bernoulli STOP unit and fix the number of steps (to 5 or 6). This reduction allowed us to compare PG and BPTT directly as both π and T were now producing only Gaussian actions which allow for the direct backpropagation via the reparametriztion trick [9]. Figure 4 shows several sample results obtained by each of the two strategies. As it was expected, BPTT resulted in a very fast convergence (around 1 hour) whereas PG typically required 1-2 days to get to reasonble interpolations which still had worse quality[1].

[1] In this experiment, we are not using the trick for stochatic transformers

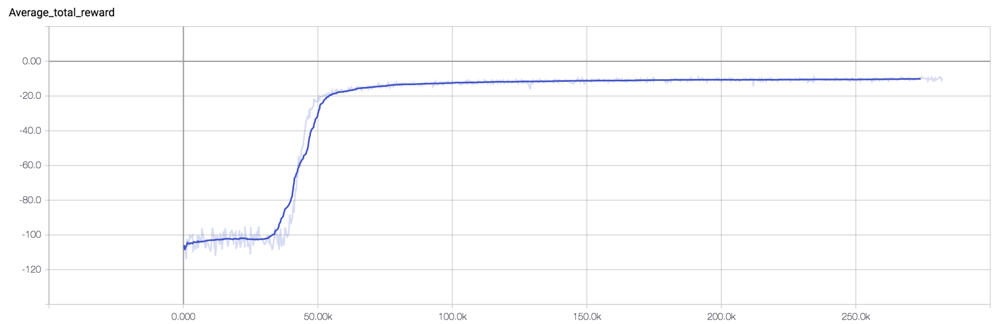


Figure 6: Training curve for the BPTT+STOP model. The agent tends to terminate the episode at the very first step in the early stages of training (up to $\approx 40{,}000$ iterations) which results in low returns.

Figure 7: Examples of interpolations produced by an BPTT-trained version of the proposed model. The leftmost column corresponds to s_{start}, while the rightmost is s_{goal}.

As we mentioned before, we couldn't overcome the instability of the hybrid training, so we are not presenting any results for that model. Interestingly, we could still train an RNN with a STOP action (the last strategy in the list). We observed that the model tends to terminate the episode at the very first step in the early stages of training even if the Bernoulli unit is set up to prefer episode continuation. At some point this behaviour ceases and the agent starts to produce the STOP action at reasonable moments (see Figure 5 and Figure 6).

4.6 CELEB-A

We also conducted an additional experiment on the CELEB-A dataset. Here we only tested the model trained with BPTT. Preliminary qualitative results are demonstrated in Figure 7. Although, interpolations do not always look plausible, we can see how the proposed approach takes advantage of having an access to raw pixels of the previous step in order to generate the next one (e. g., preserves facial features).

5 Conclusion

We have presented a transformation-based method for traversing natural image manifolds and demonstrated its potential in a series of experiments on image datasets. While the results are encouraging, it it clear that there is much more to be done in terms of develoment of the idea as well as its practical implementation. Some of the important venues for the future work include:

- Stabilization of training with policy gradients.
- Careful balancing of different reward types in order to obtain realistic interpolations (currently, it seems that L_2-distance reward overpowers the GAN objective).
- Using the transformation model as a pure generative model. This can be achieved by marginalizing the action distribution of π over the goal images. While direct marginalization is obviously intractable, we could side step this by modeling the desired distribution with a GAN or VAE.

- Incorporation of discrete actions for the policy network (along with the STOP action). Some transformations are difficult to describe with continuous commands (e. g., change hair color or digit class).

References

[1] Philip Bachman and Doina Precup. Data generation as sequential decision making. In *NIPS*, 2015.

[2] Wojciech Marian Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. *arXiv preprint arXiv:1703.00522*, 2017.

[3] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *NIPS*, 2016.

[4] Yaroslav Ganin, Daniil Kononenko, Diana Sungatullina, and Victor Lempitsky. Deepwarp: Photorealistic image resynthesis for gaze manipulation. In *ECCV*, 2016.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[6] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*, 2016.

[7] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015.

[8] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

[9] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[11] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, 2015.

[12] Eunbyung Park, Jimei Yang, Ersin Yumer, Duygu Ceylan, and Alexander C Berg. Transformation-grounded image generation network for novel 3d view synthesis. *arXiv preprint arXiv:1703.02921*, 2017.

[13] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015.

[15] Yichang Shih, Sylvain Paris, Frédo Durand, and William T Freeman. Data-driven hallucination of different times of day from a single outdoor photo. *ACM Transactions on Graphics (TOG)*, 2013.

[16] Bart Van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*, 2015.

[17] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[18] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.