Unsupervised Pixel–Level Domain Adaptation with Generative Adversarial Networks

Konstantinos Bousmalis
Google Brain
San Francisco, CA
konstantinos@google.com

Nathan Silberman
Google Research
New York, NY
nsilberman@google.com

David Dohan
Google Brain
Mountain View, CA
ddohan@google.com

Dumitru Erhan
Google Brain
San Francisco, CA
dumitru@google.com

Dilip Krishnan
Google Research
Cambridge, MA
dilipkay@google.com

Abstract

Collecting well-annotated image datasets to train modern machine learning algorithms is prohibitively expensive for many tasks. One appealing alternative is rendering synthetic data where ground-truth annotations are generated automatically. Unfortunately, models trained purely on rendered images often fail to generalize to real images. To address this shortcoming, prior work introduced unsupervised domain adaptation algorithms that attempt to map representations between the two domains or learn to extract features that are domain–invariant. In this work, we present a new approach that learns, in an unsupervised manner, a transformation in the pixel space from one domain to the other. Our generative adversarial network (GAN)–based method adapts source-domain images to appear as if drawn from the target domain. Our approach not only produces plausible samples, but also outperforms the state-of-the-art on a number of unsupervised domain adaptation scenarios by large margins. Finally, we demonstrate that the adaptation process generalizes to object classes unseen during training.

1. Introduction

Large and well-annotated datasets such as ImageNet [9], COCO [29] and Pascal VOC [12] are considered crucial to advancing computer vision research. However, creating such datasets is prohibitively expensive. One alternative is the use of synthetic data for model training. It has been a long-standing goal in computer vision to use game engines or renderers to produce virtually unlimited quantities of labeled data. Indeed, certain areas of research, such as deep reinforcement learning for robotics tasks, effectively require that models be trained in synthetic domains as training in real–world environments can be excessively expensive and time–consuming [38, 43]. Consequently, there has been a renewed interest in training models in the synthetic domain and applying them in real–world settings [8, 48, 43, 25, 32, 35, 37]. Unfortunately, models naively trained on synthetic data do not typically generalize to real images.

A common solution to this problem is using unsuper-
Supervised domain adaptation. In this setting, we would like to transfer knowledge learned from a source domain, for which we have labeled data, to a target domain, for which we have no ground truth labels. Previous work either attempts to find a mapping from representations of the source domain to those of the target \cite{41}, or seeks to find representations that are shared between the two domains \cite{14,44,31,45}, such that the features are invariant to the domain from which they are extracted. Ideally, such domain-invariant features would prevent a model trained on source data from concentrating on aspects not also found in the target domain. While such approaches have shown good progress, they are still not on par with purely supervised approaches trained only on the target domain.

In this work, we train a model to change the images from the source domain to appear as if they were sampled from the target domain while maintaining their original content. We propose a novel Generative Adversarial Network (GAN)–based architecture that is able to learn such a transformation in an unsupervised manner, i.e. without using corresponding pairs from the two domains. Our method offers a number of advantages over existing domain adaptation approaches:

**Decoupling from the Task-Specific Architecture:** In most domain adaptation approaches, the process of domain adaptation and the task-specific architecture inference are tightly integrated. One cannot switch a task–specific component of the model without having to re-train the entire domain adaptation process. In contrast, because our model maps one image to another at the pixel level, practitioners can alter the task-specific architecture without having to re-train the domain adaptation component.

**Generalization Across Label Spaces:** Because previous models couple domain adaptation with a specific task, the label spaces in the source and target domain are constrained to match. In contrast, our model is able to handle cases where the target label space at test time differs from the label space at training time.

**Training Stability:** Domain adaptation approaches that rely on some form of adversarial training \cite{5,14} are sensitive to random initialization. To address this, we incorporate a task–specific loss trained on both source and generated images and a pixel similarity regularization that allows us to avoid mode collapse \cite{40} and stabilize training. By using these tools, we are able to reduce variance of performance for the same hyperparameters across different random initializations of our model (see section 4). Note that incorporating the task–specific loss at training time doesn’t affect our ability to use a different task–specific classifier or a different label space during test time.

**Data Augmentation:** Conventional domain adaptation approaches are limited to learning from a finite set of source and target data. However, by conditioning on both source images and a stochastic noise vector, our model can be used to create virtually unlimited stochastic samples that appear similar to images from the target domain.

**Interpretability:** The output of the model, a domain–adapted image, is much more easily interpreted than a domain adapted feature vector.

To demonstrate the efficacy of our strategy, we focus on the tasks of object classification and pose estimation, where the object of interest is in the foreground of a given image, for both source and target domains. Our method outperforms the state-of-the-art unsupervised domain adaptation techniques on a range of datasets for object classification and pose estimation, while generating images that look very similar to the target domain (see Figure 1).

2. Related Work

Learning to perform unsupervised domain adaptation is an open theoretical and practical problem. While much prior work exists, our literature review focuses primarily on Convolutional Neural Network (CNN)–based methods due to their empirical superiority on this problem \cite{14,31,41,45}.

**Unsupervised Domain Adaptation:** Ganin et al. \cite{13,14} and Ajakan et al. \cite{3} introduced the Domain–Adversarial Neural Network (DANN): an architecture trained to extract domain-invariant features. Their model’s first few layers are shared by two classifiers: the first predicts task-specific class labels when provided with source data while the second is trained to predict the domain of its inputs. DANNs minimize the domain classification loss with respect to parameters specific to the domain classifier, while maximizing it with respect to the parameters that are common to both classifiers. This minimax optimization becomes possible in a single step via the use of a gradient reversal layer. While DANN’s approach to domain adaptation is to make the features extracted from both domains similar, our approach is to adapt the source images to look as if they were drawn from the target domain.

Tzeng et al. \cite{45} and Long et al. \cite{31} proposed versions of DANNs where the maximization of the domain classification loss is replaced by the minimization of the Maximum Mean Discrepancy (MMD) metric \cite{20}, computed between features extracted from sets of samples from each domain.

Bousmalis et al. \cite{5} introduce a model that explicitly separates the components that are private to each domain from those that are common to both domains. They make use of a reconstruction loss for each domain, a similarity loss which encourages domain invariance and a difference loss which encourages the common and private representation components to be complements of each other.

Other related techniques involve learning a mapping from one domain to the other at a feature level. In such a
setup, the feature extraction pipeline is fixed during the domain adaptation optimization. This has been applied in various non-CNN based approaches \cite{17,6,19} as well as the more recent CNN-based Correlation Alignment (CORAL) \cite{41} algorithm.

**Generative Adversarial Networks:** Our model uses Generative Adversarial Networks (GANs) \cite{18} conditioned on source images and noise vector. Other recent works have also attempted to use GANs conditioned on images. Ledig et al. \cite{28} used an image-conditioned GAN for super-resolution. Yoo et al. \cite{47} introduce a model for generating images of clothes from images of models wearing them, by training on corresponding pairs of the clothes worn by models and on a hanger. In contrast to our work, neither method conditions on both images and noise vectors, and ours is also applied to an entirely different problem space.

The work perhaps most similar to ours is that of Liu and Tuzel \cite{30}, who introduce an architecture of a pair of coupled GANs, one for the source and one for the target domain, whose generators share their high-layer weights and whose discriminators share their low-layer weights. In this manner, they are able to generate corresponding pairs of images which can be used for unsupervised domain adaptation. The success of this approach, however, is contingent on both images and noise vectors, and ours is also applied to an entirely different problem space.

**Style Transfer:** The popular work of Gatys et al. \cite{15} \cite{16} introduced a method of style transfer, in which the style of one image is transferred to another while holding the content fixed. While highly novel, the process is slow as it requires backpropagating back to the pixels. Johnson et al. \cite{24} introduce a model for feed forward style transfer. They train a network conditioned on an image to produce an output image whose activations on a pre-trained model are similar to both the input image (high-level content activations) and a single target image (low-level style activations). However, both of these approaches are optimized to replicate the style of a single image as opposed to our work which seeks to replicate the style of an entire domain of images.

**3. Model**

We begin by explaining our model in the context of image classification, though our model is not specific to this particular task. Given a labeled dataset in a source domain and an unlabeled dataset in a target domain, our goal is to train a classifier on data from the source domain that generalizes to the target domain. Previous work performs this task using a single network that performs both domain adaptation and image classification, making the domain adaptation process specific to the classifier architecture. Our model decouples the process of domain adaptation from the process of task-specific classification, as its primary function is to adapt images from the source domain to make them appear as if they were sampled from the target domain. Once adapted, any off-the-shelf classifier, e.g. Inception V3 \cite{42}, ResNets \cite{21}, can be trained to perform the task at hand as if no domain adaptation were required.

More formally, let \( \mathbf{X}^s = \{x^s_i, y^s_i\}_{i=0}^{N^s} \) represent a labeled dataset of \( N^s \) samples from the source domain and let \( \mathbf{X}^t = \{x^t_i\}_{i=0}^{N^t} \) represent an unlabeled dataset of \( N^t \) samples from the target domain. Our pixel adaptation model consists of a generator function \( G(x^s, z; \theta_G) \rightarrow x^t \), parameterized by \( \theta_G \), that maps a source domain image \( x^s \) and a noise vector \( z \sim p_z \) to an adapted, or fake, image \( x^t \). Given the generator function \( G \), it is possible to create a new dataset \( \mathbf{X}^t = \{G(x^s, z), y^s\} \) of any size. Finally, given an adapted dataset \( \mathbf{X}^t \), the task-specific classifier can be trained as if the training and test data were sampled from the same distribution.

**3.1. Learning**

To train our model, we employ a generative adversarial objective to encourage \( G \) to produce images that are similar to the target domain images. During training, our generator \( G(x^s, z; \theta_G) \rightarrow x^t \) maps a source image \( x^s \) and a noise vector \( z \) to an adapted image \( x^t \). Furthermore, the model is augmented by a discriminator function \( D(x; \theta_D) \) that outputs the likelihood \( d \) that a given image \( x \) has been sampled from the target domain. The discriminator tries to distinguish between ‘fake’ images \( \mathbf{X}^f \) produced by the generator, and ‘real’ images from the target domain \( \mathbf{X}^t \). Note that in contrast to the standard GAN formulation \cite{18} in which the generator is conditioned only on a noise vector, our model’s generator is conditioned on both a noise vector and an image from the source domain. In addition to the discriminator, the model is also augmented with a classifier \( T(x; \theta_T) \rightarrow \hat{y} \) which assigns task-specific labels \( \hat{y} \) to images \( x \in \{\mathbf{X}^f, \mathbf{X}^t\} \).

Our goal is to optimize the following minimax objective:

\[
\min_{\theta_G, \theta_T} \max_{\theta_D} \alpha \mathcal{L}_d(D, G) + \beta \mathcal{L}_i(G, T) \tag{1}
\]

where \( \alpha \) and \( \beta \) are weights that control the interaction of the losses. \( \mathcal{L}_d \) represents the domain loss:

\[
\mathcal{L}_d(D, G) = \mathbb{E}_{x^s} \left[ \log D(x^s; \theta_D) \right] + \mathbb{E}_{x^t} \left[ \log (1 - D(G(x^s, z; \theta_G); \theta_D)) \right] \tag{2}
\]

\( \mathcal{L}_i \) is a task-specific loss, and in the case of classification we use a typical softmax cross-entropy loss:

\[
\mathcal{L}_i(G, T) = -\mathbb{E}_{x^s, y^s} \left[ y^s \top \log T(G(x^s, z; \theta_G); \theta_T) \right] - \mathbb{E}_{x^t} \left[ y^t \top \log T(x^t; \theta_T) \right] \tag{3}
\]

where \( y^s \) is the one-hot encoding of the class label for source input \( x^s \). Notice that we train \( T \) with both adapted
and non-adapted source images. When training $T$ only on adapted images, it’s possible to achieve similar performance, but doing so may require many runs with different initializations due to the instability of the model. Indeed, without training on source as well, the model is free to shift class assignments (e.g. class 1 becomes 2, class 2 becomes 3 etc) while still being successful at optimizing the training objective. We have found that training classifier $T$ on both source and adapted images avoids this scenario and greatly stabilizes training (See Table 5). Finally, it’s important to reiterate that once trained, we are free to adapt other images from the source domain which might use a different label space (See Table 4).

In our implementation, $G$ is a convolutional neural network with residual connections that maintains the resolution of the original image as illustrated in figure 2. Our discriminator $D$ is also a convolutional neural network. The minimax optimization of Equation 1 is achieved by alternating between two steps. During the first step, we update the discriminator and task-specific parameters $\theta_D$, $\theta_T$, while keeping the generator parameters $\theta_G$ fixed. During the second step we fix $\theta_D$, $\theta_T$ and update $\theta_G$.

### 3.2. Content–similarity loss

In certain cases, we have prior knowledge regarding the low-level image adaptation process. For example, we may expect the hues of the source and adapted images to be the same. In our case, we render single objects on black backgrounds and consequently we expect images adapted from these renderings to have similar foregrounds and different backgrounds from the equivalent source images. Renderers typically provide access to z-buffer masks that allow us to differentiate between foreground and background pixels. This prior knowledge can be formalized via the use of an additional loss that penalizes large differences between source and generated images for foreground pixels only. Such a similarity loss grounds the generation process to the original image and helps stabilize the minimax optimization, as shown in Sect. 4.4 and Table 5. Our optimization objective then becomes:

$$
\min_{\theta_G, \theta_D} \max_{\theta_T} \alpha L_d(D, G) + \beta L_t(T, G) + \gamma L_c(G) \tag{4}
$$

where $\alpha$, $\beta$, and $\gamma$ are weights that control the interaction of the losses, and $L_c$ is the content–similarity loss.

We use a masked pairwise mean squared error, which is a variation of the pairwise mean squared error (PMSE) [11]. This loss penalizes differences between pairs of pixels rather than absolute differences between inputs and outputs. Our masked version calculates the PMSE between the generated foreground and the source foreground. Formally, given a binary mask $m \in \mathbb{R}^k$, our masked-PMSE loss is:

$$
L_c(G) = \mathbb{E}_{x^s, z} \left[ \frac{1}{k} \left\| (x^s - G(x^s, z; \theta_G)) \circ m \right\|^2_2 - \frac{1}{k^2} \left( (x^s - G(x^s, z; \theta_G))^\top m \right)^2 \right] \tag{5}
$$

Figure 2. An overview of the model architecture. On the left, we depict the overall model architecture following the style in [34]. On the right, we expand the details of the generator and the discriminator components. The generator $G$ generates an image conditioned on a synthetic image $x^s$ and a noise vector $z$. The discriminator $D$ discriminates between real and fake images. The task-specific classifier $T$ assigns task-specific labels $y$ to an image. A convolution with stride 1 and 64 channels is indicated as n64s1 in the image. lrelu stands for leaky ReLU nonlinearity. BN stands for a batch normalization layer and FC for a fully connected layer. Note that we are not displaying the specifics of $T$ as those are different for each task and decoupled from the domain adaptation process.
The most commonly used dataset for visual domain adaptation in the context of object classification is Office [39]. However, we do not use it in this work as there are significant high-level variations due to label pollution. For more information, see the relevant explanation in [5].
Figure 4. Visualization of our model’s ability to generate samples when trained to adapt Synth Cropped Linemod to Cropped Linemod. Top Row: Source RGB and Depth image pairs from Synth Cropped Linemod $x^s$; Middle Row: The samples adapted with our model $G(x^s, z)$ with random noise $z$; Bottom Row: The nearest neighbors between the generated samples in the middle row and images from the target training set. Differences between the generated and target images suggest that the model is not memorizing the target dataset.

3D axis) to move from the predicted to the ground truth pose [22].

4.1. Implementation Details

All the models are implemented using TensorFlow\footnote{We plan to open source our code once author feedback is released.} and are trained with the Adam optimizer [26]. We optimize the objective in Equation 1 for “MNIST to USPS” and “MNIST to MNIST-M” scenarios and the one in Equation 4 for the “Synthetic Cropped Linemod to Cropped Linemod” scenario. We use batches of 32 samples from each domain and the input images are zero-centered and rescaled to $[-1, 1]$. In our implementation, we let $G$ take the form of a convolutional residual neural network that maintains the resolution of the original image as shown in Figure 2. $z$ is a vector of $Nz$ elements, each sampled from a uniform distribution $z_i \sim U(-1, 1)$. It is fed to a fully connected layer which transforms it to a channel of the same resolution as that of the image channels, and is subsequently concatenated to the input as an extra channel. In all our experiments we use $a z$ with $Nz = 10$. The discriminator $D$ is a convolutional neural network where the number of layers depends on the image resolution: the first layer is a stride 1x1 convolution (motivated by [33]), which is followed by repeatedly stacking stride 2x2 convolutions until we reduce the resolution to less or equal to 4x4. The number of filters is 64 in all layers of $G$, and is 64 in the first layer of $D$ and repeatedly doubled in subsequent layers. The output of this pyramid is fed to a fully-connected layer with a single activation for the domain classification loss.\footnote{All details of our architecture are included in the supplementary material.} For all our experiments, the CNN topologies used for the task classifier $T$ are identical to the ones used in [14,5] to be comparable to previous work in unsupervised domain adaptation.

4.2. Quantitative Results

We have not found a universally applicable way to optimize hyperparameters for unsupervised domain adaptation. Consequently, we follow the experimental protocol of [5] and use a small set ($\sim 1,000$) of labeled target domain data as a validation set for the hyperparameters of all the methods we compare. We perform all experiments using the same protocol to ensure fair and meaningful comparison. The performance on this validation set can serve as an upper bound of a satisfactory validation metric for unsupervised domain adaptation. To our knowledge, validating parameters in an unsupervised manner is still an open research question, and is out of the scope of this work. However, as we discuss in section 4.5, we evaluate our model in a semi-supervised setting where we have 1,000 labeled examples in the target domain, to confirm that our model is still able to improve upon the naive approach of training on this small set of target labeled examples.

We evaluate our model using the aforementioned combinations of source and target datasets, and compare the performance of our model’s task architecture $T$ to that of other state-of-the-art unsupervised domain adaptation techniques based on the same task architecture $T$. As mentioned above, in order to evaluate the efficacy of our model, we first compare with the accuracy of models trained in a “Source Only” setting for each domain adaptation scenario. This setting represents a lower bound on performance. Next we compare models in a “Target Only” setting for each scenario. This setting represents a weak upper bound on performance—as it is conceivable that a good unsupervised domain adaptation model might improve on these results, as we do in this work for “MNIST to MNIST-M”.

Quantitative results of these comparisons are presented in Tables 1 and 2. Our method is able to not just achieve better results than previous work on the “MNIST to MNIST-M” scenario, it is also able to outperform the “Target Only”
Table 1. Mean classification accuracy (%) for digit datasets. The “Source-only” and “Target-only” rows are the results on the target domain when using no domain adaptation and training only on the source or the target domain respectively. We note that our Source and Target only baselines resulted in different numbers than previously published works which we also indicate in parenthesis.

<table>
<thead>
<tr>
<th>Model</th>
<th>MNIST to USPS</th>
<th>MNIST to MNIST-M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Only</td>
<td>78.9</td>
<td>63.6 (56.6)</td>
</tr>
<tr>
<td>CORAL [41]</td>
<td>81.7</td>
<td>57.7</td>
</tr>
<tr>
<td>MMD [45,31]</td>
<td>81.1</td>
<td>76.9</td>
</tr>
<tr>
<td>DANN [14]</td>
<td>85.1</td>
<td>77.4</td>
</tr>
<tr>
<td>DSN [5]</td>
<td>91.3</td>
<td>83.2</td>
</tr>
<tr>
<td>CoGAN [30]</td>
<td>91.2</td>
<td>62.0</td>
</tr>
<tr>
<td>Our model</td>
<td><strong>95.9</strong></td>
<td><strong>98.2</strong></td>
</tr>
<tr>
<td>Target-only</td>
<td>96.5</td>
<td>96.4 (95.9)</td>
</tr>
</tbody>
</table>

performance we are able to get with the same task classifier $T$. Furthermore, we are also able to learn state-of-the-art results for the “MNIST to USPS” scenario. Finally, our model is able to reduce the mean angle error for the “Synth Cropped Linemod to Cropped Linemod” scenario to more than half compared to the previous state-of-the-art.

4.3. Qualitative Results

The qualitative results of our model are illustrated in figures 1,3 and 4. In figures 3 and 4, one can see the visualization of the generation process, as well as the nearest neighbors of our generated samples in the target domain. In both scenarios, it is clear that our method is able to learn the underlying transformation process that is required to adapt the original source images to images that look like they could belong in the target domain. As a reminder, the MNIST-M digits have been generated by using MNIST digits as a binary mask to invert the colors of a background image. It is clear from figure 5 that in the “MNIST to MNIST-M” case, our model is able to not only generate backgrounds from different noise vectors $z$, but it is also able to learn this inversion process. This is clearly evident from e.g. digits 3 and 6 in the figure. In the “Synthetic Cropped Linemod to Cropped Linemod” case, our model is able to sample, in the RGB channels, realistic backgrounds and adjust the photometric properties of the foreground object. In the depth channel it is able to learn a plausible noise model.

4.4. Model Analysis

We present a number of additional experiments that demonstrate how the model works and to explore potential limitations of the model.

**Sensitivity to Used Backgrounds** In both the “MNIST to MNIST-M” and “Synthetic-Cropped LineMod to Cropped LineMod” scenarios, the source domains are images of digits or objects on black backgrounds. Our quantitative evaluation (Tables 1 and 2) illustrates the ability of our model to adapt the source images to the target domain style but raises two questions: Is it important that the backgrounds of the source images are black and how successful are data-augmentation strategies that use a randomly chosen background image instead? To that effect we ran additional experiments where we substituted various backgrounds in place of the default black background for the Synthetic Cropped Linemod dataset. The backgrounds are randomly selected crops of images from the ImageNet dataset. In these experiments we used only the RGB portion of the images—for both source and target domains—since we don’t have equivalent “backgrounds” for the depth channel. As demonstrated in Table 3, substituting various backgrounds for black makes little to no difference in the ultimate results of the model. This is consistent with the observations of prior work in which the use of random backgrounds only worked when coupled with a high degree of manual tuning to ensure that the lighting of the backgrounds and objects were consistent and the object boundary discontinuities did not appear overly synthetic.

**Generalization of the Model** Two additional aspects of the model are relevant to understanding its performance. Firstly, is the model actually learning a successful pixel-level data adaptation process, or is it simply memorizing the target images and replacing the source images with images from the target training set? Secondly, is the model able to generalize about the two domains in a fashion not limited to the classes of objects seen during training?

Table 2. Mean classification accuracy and pose error for the “Synth Cropped Linemod to Cropped Linemod” scenario.

<table>
<thead>
<tr>
<th>Model</th>
<th>Classification Accuracy</th>
<th>Mean Angle Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source-only</td>
<td>47.33%</td>
<td>89.2°</td>
</tr>
<tr>
<td>MMD [45,31]</td>
<td>72.35%</td>
<td>70.62°</td>
</tr>
<tr>
<td>DANN [14]</td>
<td>99.90%</td>
<td>56.58°</td>
</tr>
<tr>
<td>DSN [5]</td>
<td><strong>100.00%</strong></td>
<td><strong>53.27°</strong></td>
</tr>
<tr>
<td>Our model</td>
<td>99.98%</td>
<td><strong>23.5°</strong></td>
</tr>
<tr>
<td>Target-only</td>
<td>100.00%</td>
<td>6.47°</td>
</tr>
</tbody>
</table>

Table 3. Mean classification accuracy and pose error when varying the background of images from the source domain. For these experiments we used only the RGB portions of the images, as there is no trivial or typical way with which we could have added backgrounds to depth images. For comparison, we display results with black backgrounds and Imagenet backgrounds (INet), with the “Source Only” setting and with our model for the RGB-only case.

<table>
<thead>
<tr>
<th>Model–RGB-only</th>
<th>Classification Accuracy</th>
<th>Mean Angle Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source-Only–Black</td>
<td>47.33%</td>
<td>89.2°</td>
</tr>
<tr>
<td>Source-Only–INet</td>
<td>91.15%</td>
<td>50.18°</td>
</tr>
<tr>
<td>Our Model–Black</td>
<td>94.16%</td>
<td>55.74°</td>
</tr>
<tr>
<td>Our Model–INet</td>
<td><strong>96.95%</strong></td>
<td><strong>36.79°</strong></td>
</tr>
</tbody>
</table>
Table 4. Performance of our model trained on only 6 out of 11 Linemod objects. The first row, ‘Unseen Classes,’ displays the performance on all the samples of the remaining 5 Linemod objects not seen during training. The second row, ‘Full test set,’ displays the performance on the target domain test set for all 11 objects.

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Classification Accuracy</th>
<th>Mean Angle Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unseen Classes</td>
<td>98.98%</td>
<td>31.69°</td>
</tr>
<tr>
<td>Full test set</td>
<td>99.28%</td>
<td>32.37°</td>
</tr>
</tbody>
</table>

To answer the first question, we first run our generator \( G \) on images from the source images to create an adapted dataset. Next, for each transferred image, we perform a pixel-space \( L_2 \) nearest neighbor lookup in the target training images to determine whether the model is simply memorizing images from the target dataset or not. Illustrations are shown in figures 3 and 4, where the top rows are samples from \( x^s \), the middle rows are generated samples \( G(x^s, z) \), and the bottom rows are the nearest neighbors of the generated samples in the target training set. It is clear from the figures that the model is not memorizing images from the target training set.

Next, we evaluate our model’s ability to generalize to classes unseen during training. To do so, we retrain our best model using a subset of images from the source and target domains which includes only half of the object classes for the “Synthetic Cropped Linemod” to “Cropped Linemod” scenario. Specifically, the objects ‘ape’, ‘benchviseblue’, ‘can’, ‘cat’, ‘driller’, and ‘duck’ are observed during the training procedure, and the objects ‘holepuncher’, ‘iron’, ‘lamp’, ‘phone’, ‘cam’ are only used during testing. Once \( G \) is trained, we fix its weights and pass the full training set of the source domain to generate images used for training the task-classifier \( T \). We then evaluate the performance of \( T \) on the entire set of unobserved objects (6,060 samples), and the test set of the target domain for all objects for direct comparison with Table 2.

Stability Study  We also evaluate the importance of the different components of our model. We demonstrate that while the task and content losses do not improve the overall performance of the model, they dramatically stabilize training. Training instability is a common characteristic of adversarial training, necessitating various strategies to deal with model divergence and mode collapse [40]. We measure the standard deviation of the performance of our models by running each model 10 times with different random parameter initialization but with the same hyperparameters. Table 5 illustrates that the use of the task and content–similarity losses reduces the level of variability across runs and provides more reproducible performance.

Table 5. The effect of using the task and content losses \( L_t, L_c \) on the standard deviation (std) of the performance of our model on the “Synth Cropped Linemod to Linemod” scenario. \( L_t^{source} \) means we use source data to train \( T \); \( L_t^{adapted} \) means we use generated data to train \( T \); \( L_c \) means we use our content–similarity loss. A lower std on the performance metrics means that the results are more easily reproducible.

<table>
<thead>
<tr>
<th>( L_t^{source} )</th>
<th>( L_t^{adapted} )</th>
<th>( L_c )</th>
<th>Classification Accuracy</th>
<th>Mean Angle Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>23.26</td>
<td>16.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>22.32</td>
<td>17.48</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>2.04</td>
<td>3.24</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.60</td>
<td>6.97</td>
</tr>
</tbody>
</table>

4.5. Semi-supervised Experiments

Finally, we evaluate the usefulness of our model in a semi-supervised setting, in which we assume we have a small number of labeled target training examples. The semi-supervised version of our model simply uses these additional training samples as extra input to classifier \( T \) during training. We sample 1,000 examples from the Cropped Linemod not used in any previous experiment and use them as additional training data. We evaluate the semi-supervised version of our model on the test set of the Cropped Linemod target domain against the 2 following baselines: (a) training a classifier only on these 1,000 target samples without any domain adaptation, a setting we refer to as ‘1,000-only’; and (b) training a classifier on these 1,000 target samples and the entire Synthetic Cropped Linemod training set with no domain adaptation, a setting we refer to as ‘Synth+1000’. As one can see from Table 6, our model is able to greatly improve upon the naive setting of incorporating a few target domain samples during training. We also note that our model is able to use these results to achieve an even better performance than in the completely unsupervised setting for the “Synthetic Cropped Linemod to Cropped Linemod” scenario (Table 2).

Table 6. Semi-supervised experiments for the “Synthetic Cropped Linemod to Cropped Linemod” scenario. When a small set of 1,000 target data is available to our model, it is able to improve upon baselines trained on either just these 1,000 samples or the synthetic training set augmented with these labeled target samples.

<table>
<thead>
<tr>
<th>Method</th>
<th>Classification Accuracy</th>
<th>Mean Angle Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000-only</td>
<td>99.51%</td>
<td>25.26°</td>
</tr>
<tr>
<td>Synth+1000</td>
<td>99.89%</td>
<td>23.50°</td>
</tr>
<tr>
<td>Our model</td>
<td>99.93%</td>
<td>13.31°</td>
</tr>
</tbody>
</table>

5. Conclusion

We present a state-of-the-art method for performing unsupervised domain adaptation. Our models outperform previous work on a set of unsupervised domain adaptation scenarios, and in the case of the challenging “Synthetic Cropped Linemod to Cropped Linemod” scenario,
our model more than halves the error for pose estimation compared to the previous best result. They are able to do so by using a GAN–based technique, stabilized by both a task-specific loss and a novel content–similarity loss. Furthermore, our model decouples the process of domain adaptation from the task-specific architecture, and provides the added benefit of being easy to understand via the visualization of the adapted image outputs of the model.

References


Supplementary Material

A. Additional Generated Images

Figure 1. Linear interpolation between two random noise vectors demonstrates that the model is able to separate out style from content in the MNIST-M dataset. Each row is generated from the same MNIST digit, and each column is generated with the same noise vector.
B. Model Architectures and Parameters

We present the exact model architectures used for each experiment along with hyperparameters needed to reproduce results. The general form for the generator, G, and discriminator, D, are depicted in Figure 2 of the paper. For G, we vary the number of filters and the number of residual blocks. For D, we vary the amount of regularization and the number of layers.

Optimization consists of alternating optimization of the discriminator and task classifier parameters, referred to as the D step, with optimization of the generator parameters, referred to as the G step.

Unless otherwise specified, the following hyperparameters apply to all experiments:

- Batch size 32
- Learning rate decayed by 0.95 every 20,000 steps
- All convolutions have a 3x3 filter kernel
- Inject noise drawn from a zero centered Gaussian with stddev 0.2 after every layer of discriminator
- Dropout every layer in discriminator with keep probability of 90%
- Input noise vector is 10 dimensional sampled from a uniform distribution $U(-1, 1)$
- We follow conventions from the DCGAN paper [36] for several aspects
  - An L2 weight decay of $1e^{-5}$ is applied to all parameters
  - Leaky ReLUs have a leakiness parameter of 0.2
  - Parameters initialized from zero centered Gaussian with stddev 0.02
  - We use the ADAM optimizer with $\beta_1 = 0.5$

B.1 USPS Experiments

The Generator and Discriminator are identical to the MNIST-M experiments.

Loss weights:

- Base learning rate is $2e^{-4}$
- The discriminator loss weight is 1.0
- The generator loss weight is 1.0
- The task classifier loss weight in G step is 1.0
- There is no similarity loss between the synthetic and generated images

B.2 MNIST-M Experiments (Paper Table 1)

Generator: The generator has 6 residual blocks with 64 filters each

Discriminator: The discriminator has 4 convolutions with 64, 128, 256, and 512 filters respectively. It has the same overall structure as paper Figure 2

Loss weights:

- Base learning rate is $1e^{-3}$
- The discriminator loss weight is 0.13
- The generator loss weight is 0.011
- The task classifier loss weight in G step is 0.01
- There is no similarity loss between the synthetic and generated images

B.3 LineMod Experiments

All experiments are run on a cluster of 10 TensorFlow workers.

Generator: The generator has 4 residual blocks with 64 filters each

Discriminator: The discriminator matches the depiction in paper Figure 2. The dropout keep probability is set to 35%.
Parameters without masked loss (Paper Table 2):

- Base learning rate is $2.2e^{-4}$, decayed by 0.75 every 95,000 steps
- The discriminator loss weight is 0.004
- The generator loss weight is 0.011
- The task classification loss weight is 1.0
- The task pose loss weight is 0.2
- The task classifier loss weight in G step is 0
- The task classifier is not trained on synthetic images
- There is no similarity loss between the synthetic and generated images

Parameters with masked loss (Paper Table 5):

- Base learning rate is $2.6e^{-4}$, decayed by 0.75 every 95,000 steps.
- The discriminator loss weight is 0.0088
- The generator loss weight is 0.011
- The task classification loss weight is 1.0
- The task pose loss weight is 0.29
- The task classifier loss weight in G step is 0
- The task classifier is not trained on synthetic images
- The MPSE loss weight is 22.9

C. InfoGAN Connection

In the case that $T$ is a classifier, we can show that optimizing the task loss in the way described in the main text amounts to a variational approach to maximizing mutual information $\text{(7)}$, akin to the InfoGAN model $\text{(7)}$, between the predicted class and both the generated and the equivalent source images. The classification loss could be re-written, using conventions from $\text{(7)}$ as:

$$ L_t = -\mathbb{E}_{x^* \sim D^s}[\mathbb{E}_{y' \sim p(y|x^*)}\log q(y'|x^*)] 
- \mathbb{E}_{x^f \sim G(x^*, z)}[\mathbb{E}_{y' \sim p(y|x^f)}\log q(y'|x^f)] $$

$$ \geq -I(y', x^*) - I(y', x^f) + 2H(y), $$

where $I$ represents mutual information, $H$ represents entropy, $H(y)$ is assumed to be constant as in $\text{(7)}$, $y'$ is the random variable representing the class, and $q(y'|.)$ is an approximation of the posterior distribution $p(y'|.)$ and is expressed in our model with the classifier $T$. Again, notice that we maximize the mutual information of $y'$ and the equivalent source and generated samples. By doing so, we are effectively regularizing the adaptation process to produce images that look similar for each class to the classifier $T$. This helps maintain the original content of the source image and avoids, for example, transforming all objects belonging to one class to look like objects belonging to another.
Figure 2. Additional generation examples for the LineMod dataset. The left 4 columns are generated images and depth channels, while the corresponding right 4 columns are L2 nearest neighbors.
### MNIST-M Task Classifier

- **Private**
  - Convolutional layer: 5x5x32, ReLU
  - Max-pool: 2x2, 2x2 stride

- **Shared**
  - Convolutional layer: 5x5x48, ReLU
  - Max-pool: 2x2, 2x2 stride
  - Fully connected (FC): 100 units, ReLU
  - Fully connected (FC): 100 units, ReLU
  - Fully connected (FC): 10 units, sigmoid

### LineMod Task Classifier

- **Private**
  - Convolutional layer: 5x5x32, ReLU
  - Max-pool: 2x2, 2x2 stride

- **Shared**
  - Convolutional layer: 5x5x64, ReLU
  - Max-pool: 2x2, 2x2 stride
  - Fully connected (FC): 128 units, ReLU
  - Dropout: 50%
  - Fully connected (FC): 11 units, sigmoid (class)
  - Fully connected (FC): 1 units, tanh (angle)

---

**Figure 3.** Task classifier (T) architectures for each dataset. We use the same task classifiers as [5][14] to enable fair comparisons. The MNIST-M classifier is used for USPS, MNIST, and MNIST-M classification. During training, the task classifier is applied to both the synthetic and generated images when $L_{\text{source}}^t$ is enabled (see Table 5). The 'Private' parameters are only trained on one of these sets of images, while the 'Shared' parameters are shared between the two. At test time on the target domain images, the classifier is composed of the Shared parameters and the Private parameters which are part of the generated images classifier. These first private layers allow the classifier to share high level layers even when the synthetic and generated images have different channels (such as 1 channel MNIST and RGB MNIST-M).