Alternating Refinement Diffusion for Graphical Model-based Domain Adaptation

Kanad Pardeshi **Pratheek D Souza Rebello**

1 Introduction and Motivation

Diffusion models [14, 13] have surpassed generative adversarial networks (GANs) [6] in several mainstream tasks such as conditional and unconditional image generation [2]. This superiority is attributed to the ability of diffusion models to better capture distributions with multiple modes and methods which are less prone to divergence [2]. However, there are several niche tasks for which GANs are still state-of-the-art. Can diffusion models help improve performance in these tasks? We wish to explore the answer to this question by considering the case of domain adaptation.

Domain adaptation is a fairly popular problem in which we aim to learn models which can generalize well to domains other than the ones they have been trained on. This has great relevance to the use of these models in real-life settings, since several unaccounted factors can change some aspects of the underlying distribution of data, while others remain reasonable invariant. Domain adaptation aims to identify and use these domain-invariant aspects for more generalizable prediction.

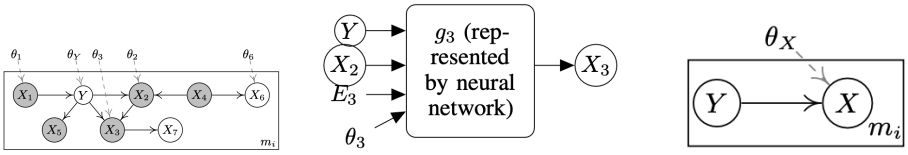
In domain adaptation, we have access to data from (possibly several) source domains during training, and our model has to adapt to a target domain which can be different from any of the source domains. In this work, we consider the task of classification. Formally, we have access to n source domains, with m_i samples for domain i given by $\mathcal{D}_{S_i} = \left\{(x_i^k, y_i^k)\right\}_{k=1}^{m_i}$. The j-th feature of the k-th sample in domain i is given by x_i^{jk}. We aim to find a classifier for the target domain for which we only have access to features i.e. $\mathcal{D}_T = \left\{x_T^k\right\}_{k=1}^{m_T}$

2 Background

Our analysis is based on [18], which adopts a graphical model perspective on domain adaptation. They introduce the notion of an augmented graphical model which consists of the given variables $X_i, i \in [d]$ and Y along with latent variables denoted by a vector $\boldsymbol{\theta}$. These latent variables vary with domain and hvae directed edges to the given variables. The subgraph defined by the given variables stays the same across domains. Figure 1a describes an example of an augmented graphical model. In this example, $\theta_1, \theta_2, \theta_3, \theta_6, \theta_Y$ are domain-dependent, and impact the conditional distributions of variables X_1, X_2, X_3, X_6, Y respectively. We also observe that conditional distributions such as $P(X_5 \mid Y)$ are independent of $\boldsymbol{\theta}$, and hence do not depend on the domain. For domain i, the domain-dependent information is encoded as $\boldsymbol{\theta}^{(i)}$, and this remains a constant for samples from the same domain.

[18] first infer the augmented graph from the data using the PC algorithm. This is followed by modeling the conditional distribution of Y given its parents using a conditional GAN (CGAN). The generator learns to generate images conditioned on the domain and parent variables, while the discriminator learns to distinguish real and generated images along with predicting the output variable and the domain for the input image. After training, the discriminator can be used on the target domain to get y_T^k. Figure 1b provides an example of the inputs and outputs for such a network.

In this work, we aim to study if replacing GANs with diffusion models leads to an improvement in classifier performance for the target domain. We wish to conduct experiments on variants of



(a) An example of an augmented graphical model

(b) Schematic network for $P(X_3 \mid Y, X_2, \theta_3)$

(c) Augmented graphical model for MNIST

Figure 1: Augmented graphical models and LV-CGANs

MNIST used in domain adaptation tasks [19]. We conduct baseline experiments using the code provided in [18], and aim to incorporate diffusion models in their method, comparing the resultant performance with the baseline. We conduct a variety of experiments on diffusion models and measure their performance on a benchmark dataset. Eventually, we develop a model which provides SOTA performance over this task.

3 Related Work

Test data is often different from the training data in real-world applications, and domain adaptation [3] deals with addressing this issue. A variety of techniques have been proposed for domain adaptation, with some recent works introducing diffusion models for domain adaptation [1, 15]. While these works provide state-of-the-art performance on image-related tasks, other tasks can be interpreted as graphical models. Along with enabling domain adaptation, graphical model-based methods also provide us with useful information about the relationships between features and responses. Popular works such as [16] provide algorithms (such as the PC algorithm used in [18]) to learn graphical models from data. To better understand the algorithms in the primary reference [18], we also referred to [11], which proposes conditional GANs, a precursor to LV-CGANs used in [18].

4 Methods

We consider the digit prediction task described in [18], combining four popular digit datasets MNIST [9], MNIST-M [4], SVHN [12], and Synth Digits [4]. We use MNIST-M as the target domain and the rest as source domains (Figure 6). The underlying graphical model for this data of images X and labels Y is $Y \to X$, with $P(X|Y)$ dependent on the domain θ. The graphical model is presented in Figure 1c.

Formally,

$$\mathcal{D}_S^{real} = \{\mathcal{D}_{S_i}^{real}\}_{i=1}^n = \{\{x_{S_i}^k, y_{S_i}^k\}_{k=1}^m\}_{i=1}^n, \mathcal{D}_T^{real} = \{x_T^k\}_{k=1}^m$$

Our goal is to train a CNN classifier f_c to correctly predict one of $C = 10$ classes for the target domain D_T^{real}, for which labels are not available at training time. We use a 11-layer CNN followed by a dense layer as our classifier. We train for 100 epochs, with cross-entropy loss with label smoothing, Adam optimizer, and standard data augmentation tricks like `RandomRotation`, `RandomAffine`, `ColorJitter`. We keep this consistent across all our experiments

4.1 Baselines

4.1.1 poolNN

For a baseline we implement `poolNN` which simply trains classifier f_c on all available labeled data i.e. on the three source domains $\mathcal{D}_S^{real}$

To obtain a best-case accuracy for our task we also implement `target-gold` which trains f_c directly on target domain with its gold labels $\mathcal{D}_{T_{gold}}^{real}$, and `poolNN-gold`, which trains f_c on target domain with gold labels, as well as source domains $\mathcal{D}_{T_{gold}}^{real} \cup \mathcal{D}_S^{real}$. **Note:** These two techniques are not domain adaptation methods - they use gold target labels. They simply provide us with an upper-bound for our task

4.1.2 DA-Infer: Conditional GAN

Using the code provided in [18], we train a Latent Variable Condition GAN (LV-CGAN) on the three source domains $\mathcal{D}_S^{real}$ and the target domain $\mathcal{D}_T^{real}$. The generator G takes in a label, domain, and Gaussian noise to conditionally output a synthetic image. The discriminator D takes in a real or synthetic image and outputs a discriminative score, a class label and a domain label. The discriminator loss is a combination of MSE Loss on the discriminative score and cross entropy loss on the class and domain labels. After training, we simply use the Discriminator D as f_c to make class label predictions on the target domain and call this method `DA-Infer` as in [18].

4.2 Our Techniques

4.2.1 Domain-Invariant Latent Representations

Inspired by popular UDA methods like CORAL [17] and DANN [5] we make a first step by nudging our CNN feature extractor to learn domain invariant feature embeddings. Intuitively, we want the CNN to preserve features common across domains while forgetting differences between domains. To encourage this, we first train the CNN so that any two images belonging to the same class have latent representations which are close to each other, whereas images belonging to different classes have latent representations far from each other. Formally, given two images $\mathbf{x}_1$ and $\mathbf{x}_2$, we use the following loss for the CNN f_θ:

$$L_1(\mathbf{x}_1, \mathbf{x}_2) = (C \cdot \mathbb{I}(y_i = y_j) - 1) \, \| f_\theta(\mathbf{x}_1) - f_\theta(\mathbf{x}_2 \|_2^2$$

Here C is a hyper-parameter set to 10 (so that matching and non-matching pairs are given equal weights, since there are more non-matching pairs than matching ones). Since we only have labels for the source domains, the above training does not involve the target domain. Subsequently, we train the classifier end-to-end by adding this loss to standard cross entropy. This final model is tested on the target domain to get the predicted labels.

4.2.2 Diffusion Adaptation

Conditional GANs and conditional diffusion models are SOTA generative models, known to produce high quality images conditioned on some text, class or other label. Inspired by `DA-Infer` which uses a C-GAN, we aim to use a conditional diffusion model for domain adaptation.

The high level goal is to teach the conditional diffusion model to generate synthetic images from the target domain $\mathcal{D}_T^{real}$, conditioned on a class label. If we can do this we will have a *labeled* synthetic target dataset, which we can then utilize downstream to train classifier f_c:

$$\mathcal{D}_T^{syn} = \{z_T^k, y_T^k\}_{k=1}^m, \text{ where } z_T^k \text{ is a synthetic image}$$

4.2.3 Diffusion Model Architecture, Training and Inference

We use a conditional diffusion model with UNet CNN backbone. The model takes in a (class label, domain label) = (c_i^k, d_i^k) as the conditioning input and outputs a synthetic image z_i^k from that domain, of that class. Conditioning is implemented by encoding the class and domain via independent learnable embeddings and then adding them at some stage(s) of the diffusion process to guide generation

Recall that we have domain labels for all images, however we have class labels only for source domain images. Our hope is that the diffusion model will learn knowledge about classes from the source domains and transfer this to the target domain.

We train with a DDPM (Denoising Diffusion Probabilistic Model) schedule with 400 steps for 100-150 epochs. At inference time, we generate $m = 20$k synthetic images from the target domain, with 2k images from each class. We sample the CFG scale [7] from a Uniform$(0.5, 2.5)$ while generating each image. A lower CFG scale leads to more diverse, but less class accurate images. We choose to randomize this value so as to create a robust dataset that has both high quality and diverse images to aid the classifier downstream.

(a) `DiffuseAdapt-unk` (b) 1st iteration of `DiffuseAdapt-refine`

Figure 2: Synthetic Images from Diffusion Model for Target Domain MNIST-M

4.2.4 Domain Adaptation Challenges for Diffusion Models

Both C-GANs and conditional diffusion models implement conditioning by taking in the conditioning variable as an additional input, encoding it via an embedding and then adding it at some stage(s)

A key difference between training a conditional diffusion model (H) and the generator (G) of a conditional GAN is that the G takes in *sampled* values of the conditioning variable. It then relies on the discriminator D to enforce consistency between images and their true conditioning variable. The diffusion model H however, requires the *true* value of the conditioning variable to be fed to it at training time. For UDA tasks, we do not have access to the true class label for the target domain. We propose several novel techniques to handle this problem:

4.2.5 Unk Class

To train a domain adaptation task with C classes, we create a conditional diffusion model which accepts one of $C+1$ classes as conditioning variable, together with the domain label. For an image $\mathcal{D}_S^{real}$ we simply feed in the true class label $c_k^i = y_k^i$ and the domain index $d_k^i = k$. For an image in $\mathcal{D}_S^{real}$ we feed in the unk class $c_T^i = C+1$ and the domain index $d_T^i = T$.

Algorithm 1 `DiffuseAdapt-unk`

1: Input: $\mathcal{D}_S^{real} = \{\{x_{S_i}^k, y_{S_i}^k\}_{k=1}^m\}_{i=1}^n, \mathcal{D}_T^{real} = \{x_T^k\}_{k=1}^m$
2: Train diffusion model H^{unk} on $\{\{x_{S_i}^k, c = y_{S_i}^k, d = k\}_{k=1}^m\}_{i=1}^n \cup \{x_T^k, c = C+1, d = T\}_{k=1}^m$
3: Run H^{unk} inference mode by feeding $\{c^k \in [1, ..., C], d^k = T\}_{k=1}^m$ to generate images $\{z_T^k\}_{k=1}^m$
4: Assemble the new synthetic dataset $\mathcal{D}_{unk}^{syn} = \{z_T^k, c^k\}_{k=1}^m$
5: Train f_c^{unk} on $\mathcal{D}_S^{real} \cup \mathcal{D}_{unk}^{syn}$
6: **return** f_c^{unk}

Once the diffusion model is trained, we run it in inference mode, feeding in class label $c^k \in [1, .., C]$, domain index as target $d^k = T$ to generate a synthetic labeled dataset for the target domain $\mathcal{D}_T^{syn-unk}$. We finally train a classifier f_c on this synthetic labeled data. We call this method `DiffuseAdapt-unk`. This is what we implemented in the **Mid Term Report**

When we visually inspect the synthetic target images generated by `DiffuseAdapt-unk` (Figure 2a) we find that while they are faithful to the class label, they are of poor quality and significantly different from the real images. In fact, many synthetic images look like like they are from MNIST (source) instead of MNIST-M (target). Our hope was that the diffusion model would learn that the class embedding and domain embedding provide *orthogonal* information. However it seems to us that the model has instead learned that whenever a class label $\in [1, ..., C]$ is provided, it should output images from the source domains. This because the diffusion model never sees images of the target domain with class label $\in [1, ..., C]$ during training. This motivates our next method.

4.3 Pseudo Labeled Conditional Diffusion

Hypothesizing that the chief weakness of `DiffuseAdapt-unk` is that it never sees images from the target domain with class label $\in [1, ..., C]$ during training, we propose to mitigate this by using a

classifier to give us some pseudo labels for real target images. But training a classifier for target images is the initial domain adaptation task we started out with!

The solution? Use the downstream classifier f_c^{unk} from `DiffuseAdapt-unk` to pseudo label the real target dataset, and retrain the diffusion model using these pseudo labels. But why stop there? This procedure could be repeated!

4.4 Conditional Diffusion with Alternate Refinement

This idea naturally leads to an alternating procedure which can be summarized as follows:

Algorithm 2 `DiffuseAdapt-refine`

1: Input: $\mathcal{D}_S^{real} = \{\{x_{S_i}^k, y_{S_i}^k\}_{k=1}^m\}_{i=1}^n, \mathcal{D}_T^{real} = \{x_T^k\}_{k=1}^m$
2: Run `DiffuseAdapt-unk` to obtain f_c^{unk}
3: Set $f_c^0 := f_c^{unk}$
4: **for all** t from 1 to N **do**
5: Use f_c^{t-1} to pseudo label $\mathcal{D}_T^{real}$ to get $\mathcal{D}_T^{t-pseudoreal} = \{(x_T^k, \hat{y}_T^k)\}_{k=1}^m$
6: Train diffusion model H^t on $\mathcal{D}_S^{real} \cup \mathcal{D}_T^{t-pseudoreal}$
7: Use H^t to generate a synthetic labeled target dataset $\mathcal{D}_{syn}^t$
8: Train a classifier f_c^t on $\mathcal{D}_S^{real} \cup \mathcal{D}_{syn}^t$
9: **end for**
10: **return** f_c^N

This algorithm relies on the progessively increasing classification power of f_c to provide more accurate conditioning for the diffusion model, which in turn generates better quality images to train the classifier.

We find that this method dramatically improves the quality of synthetic images even after just one iteration (Figure 2b) as well as dramatically improves the classification accuracy of the f_cs

4.5 Variants of Conditional Diffusion with Alternate Refinement

We implement several other variations of `DiffuseAdapt-refine`

1. `DiffuseAdapt-trainingWheels`: This is motivated by the fact that once a diffusion model H^t generates good enough synthetic target images, we may not require real source domain images anymore to train the classifier f_c^{t+1}. We can rely purely on the synthetic target data. Thus this method modifies line (8) of `DiffuseAdapt-refine` to say:

Algorithm 3 `DiffuseAdapt-trainingWheels`

8: If $t < N_{tw}$: Train a classifier f_c^t on $D_S^{real} \cup \mathcal{D}_{syn}^t$
9: Else: Train a classifier f_c^t on $\mathcal{D}_{syn}^t$

2. `DiffuseAdapt-poolNN`: This method eliminates the need to use `DiffuseAdapt-unk` as f_c^0. Instead we use the classifier trained on baseline `poolNN` (i.e. only source domain data) as our initial classifier. Simply modify Line 3 of `DiffuseAdapt-refine`

Algorithm 4 `DiffuseAdapt-poolNN`

3: Set $f_c^0 := f_c^{\texttt{poolNN}}$

3. `DiffuseAdapt-diffFromScratch`: It is important to mention that uptil now we have been initializing diffusion model H^t with weights from the previous iteration H^{t-1}. However we suspect that this may lead to overfitting on bad labels in early iterations. Thus we also implement a method `DiffuseAdapt-diffFromScratch` which re-initializes H^t randomly at each iteration

We find all three methods improve performance, so we define a final method `DiffuseAdapt-champion` which uses `DiffuseAdapt-trainingWheels`, `DiffuseAdapt-poolNN` and `DiffuseAdapt-diffFromScratch`

4.6 Other Techniques

We also report some other techniques we tried, but which failed to give impressive results. Despite the incomplete results for these methods, we realized that the ideas provide interesting insights into the problem, and we thus include them for completeness.

4.6.1 Zero-Shot Classifier

[10] proposes that diffusion models can be used for zero-shot classification. They claim that the loss between the added noise and the noise predicted by the diffusion model can be used as a prediction of the class a particular image belongs to. Formally, the diffusion model can be used as a classifier as

$$f(\mathbf{x}) = \arg\min_{c} \left(\mathbb{E}_{t,\epsilon} \left\| \epsilon_\theta(\mathbf{x}_t, c) - \epsilon \right\|_2^2 \right) \tag{1}$$

The expectation can be estimated by sampling over added Gaussian noise ϵ and $t \in [0, T]$. Our experiments with this method involved training the conditional diffusion model on source domain and then using it on the target domain as per (1). We observed that the accuracy on source domain images was 29.977%, whereas the accuracy on the target domain was only 13.866%, which is just very slightly better than random. The diffusion model by itself is thus not able to adapt to the target domain, which further establishes the importance of the classifier.

4.6.2 Joint Training

We also explored if the diffusion model and classifier could be trained jointly. This involved training the diffusion model on source domains first, followed by training the diffusion model and classifier in an end-to-end manner. An important consideration in this method is that the diffusion model is used to generate images on which the classifier is trained, meaning that the diffusion is being used for inference. Similar to recurrent neural networks (RNNs), we can 'unroll' the diffusion model, where the UNet block would be repeated multiple times, and the output from one iteration is reused in the subsequent iteration.

Unfortunately, owing to the size of the UNet and the number of steps, it is computationally difficult to calculate gradients all the way to the first step of the diffusion model. This problem can be overcome by truncating the gradient after a few steps and using this truncated calculation to update the weights of the diffusion model. We observed that even this method required a considerable amount of time, and we thus could not obtain final results for this method. However, we conjecture that using low-rank representations such as low-rank adaptation (LoRA) [8] for the diffusion model can allow us to consider deeper gradients, leading to better end-to-end training.

5 Results

Table 5 provides accuracy results on the target domain (`MNIST-M`) for the methods discussed above. We observe that `DA-Infer`, the method in [18], provides the highest accuracy on the task. Among our techniques, zero-shot gives the worst accuracy by far, with the domain-invariant latent representation method providing a result slightly worse than the baseline. Our baseline diffusion model, `DiffuseAdapt-unk`, does slightly better, and subsequent modifications can be seen to improve the results. We obtain near-SOTA accuracy with our final model, `DiffuseAdapt-champion`. In 4, we see that the test accuracy on the target domain is higher across iterations for our champion model, nearly approaching the gold accuracy on the target domain. We see that the iterative refinement idea results in a very significant improvement in accuracy.

An interesting observation is that the accuracy on the target images starts to plateau around iteration 4 for `DiffuseAdapt-refine`. This is why in `DiffuseAdapt-trainingWheels` at iteration 5 we choose to switch from training on a mixture of synthetic and real images to completely synthetic images, and thus we see the marked rise in accuracy. We do the same for `DiffuseAdapt-champion` and `DiffuseAdapt-poolNN` and observe the same trend. This clearly indicates that after a point

source domain images are no longer very useful for hard target domains for which we have good quality synthetic data

Method	Accuracy (%)
`poolNN` (baseline)†	66.31
`DA-Infer`†	**89.89** [a]
`DomainInvariantLatent`	64.63
`DiffuseAdapt-unk`	71.35
`DiffuseAdapt-refine`	80.40
`DiffuseAdapt-trainingWheels`	90.49
`DiffuseAdapt-poolNN`	89.35
`DiffuseAdapt-champion`	**93.10**
`ZeroShot`	13.87
`poolNN-gold`*	84.63
`target-gold`*	95.36

† Methods from [18]
* Not domain adaption techniques

Figure 3: Target Test Accuracy of our methods

[a] In a run of our own, we saw accuracy go up to 94.51

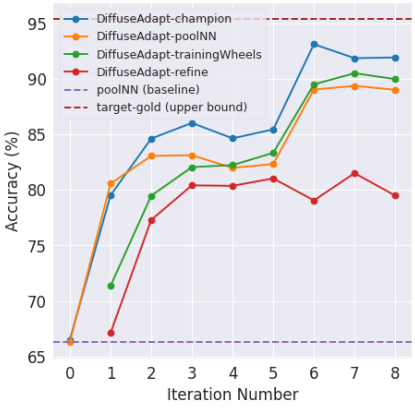


Figure 4: Target test accuracy with alternating iterations. Each iteration is one loop starting at line (4) of Algorithm 2. Note `refine` and `trainingWheels` start at epoch 1 - they use the unk diffusion dataset. But `poolNN` and `champion` start at epoch 0 with a model trained on source domains

Figure 5: Results on MNIST-M target dataset

(a) Source Domain: `MNIST, SVHN, Synth Digits`

(b) Target Domain: `MNIST-M`

Figure 6: Real Images

(a) Source Domain: `MNIST, SVHN, Synth Digits`

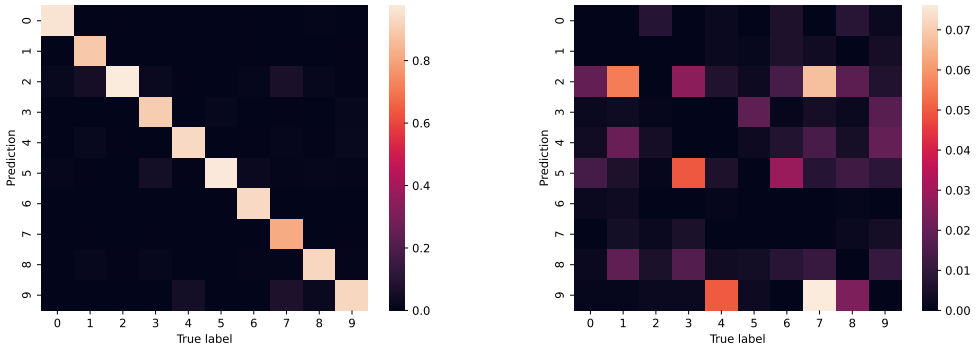
(b) Target Domain: `MNIST-M`

Figure 7: Synthetic Images from Diffusion Model trained in `DiffuseAdapt-champion`

6 Discussion and Analysis

We also analyze the class-wise accuracy of our learnt classifier from `DiffuseAdapt-champion`. The heatmap in Figure 8a shows that the predicted and true classes are almost always the same (note that we have normalized according to each column). To better understand the erroneous regions, we also construct a heatmap where the diagonals values are removed. We see that the model most frequently mistakes a) 7 for 9, b) 7 for 2, c) 1 for 2, d) 3 for 5, and e) 4 for 9. All of these digits are indeed pretty close to each other in handwritten settings.

In this work, we explored if diffusion models can be a feasible alternative to GANs in the case of domain adaptation. We consider a variety of approaches, iteratively modifying our models to achieve



(a) Predicted vs true class

(b) Incorrectly predicted vs true class

Figure 8: Heatmaps for predicted vs true class - normalized w.r.t. true class count

an accuracy which is very competitive to the LV-CGAN model proposed in [18]. Our unsuccessful methods also gave us a detailed understanding about the nuances in the problem. We conclude that diffusion models can indeed be a feasible replacement in this case.

Some possible modifications which can lead to even better results can be:

1. When using the synthetic images labeled using the classifier as data for the diffusion model, we can 'prune' the dataset by removing the images for which the classifier is least confidence. This can be achieved by ranking the predictions using some metric such as entropy, and discarding a certain proportion of synthetic samples having the highest entropy.
2. In our `DiffuseAdapt-champion` model, each iteration involves training a diffusion model initialized with randomly assigned weights, since reusing the previous model can lead to poor results on the target domain. We believe that an intermediate method involving perturbing the weights instead of randomizing them completely can also be beneficial. Since the diffusion model doesn't learn from scratch in this case, this can also lead to a potential speedup in training time. The perturbation schedule would be an additional hyperparameter to tune.

A likely pitfall of our approach can be that since the synthetic images are being used for training the diffusion model in subsequent steps, the errors in one iteration might be exacerbated in subsequent iterations. This can lead to a potential runaway effect in which the performance might be good on the synthetic images but poor on the target domain. However, we also develop some useful methods in our work to alleviate this problem.

Although there are other useful methods for domain adaptation, a particularly attractive feature of this method is that the augmented graphical model learnt using the PC algorithm also provides us with some understanding about the interactions between the variables.

7 Teammates and work division

Our team consists of Kanad Pardeshi (`kpardesh`) and Pratheek D Souza Rebello (`pdsouzar`). We have divided the work equally between ourselves.

8 Code

The code for the project can be found at `https://github.com/pdrebello/pgm-diffusion-adapt.git`. We use some code from `https://github.com/mgong2/DA_Infer.git` which implements DA-Infer and `https://github.com/TeaPearce/Conditional_Diffusion_MNIST.git` which implements a vanilla conditional diffusion model for pure MNIST

References

[1] Yasser Benigmim, Subhankar Roy, Slim Essid, Vicky Kalogeiton, and Stéphane Lathuilière. One-shot unsupervised domain adaptation with personalized diffusion models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 698–708, 2023.

[2] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. Advances in neural information processing systems, 34:8780–8794, 2021.

[3] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R Arabnia. A brief review of domain adaptation. Advances in data science and information engineering: proceedings from ICDATA 2020 and IKE 2020, pages 877–894, 2021.

[4] Yaroslav Ganin, E. Ustinova, Hana Ajakan, Pascal Germain, H. Larochelle, François Laviolette, Mario Marchand, and Victor S. Lempitsky. Domain-adversarial training of neural networks. In Journal of machine learning research, 2015.

[5] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks, 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. Advances in neural information processing systems, 27, 2014.

[7] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

[8] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[10] Alexander C Li, Mihir Prabhudesai, Shivam Duggal, Ellis Brown, and Deepak Pathak. Your diffusion model is secretly a zero-shot classifier. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 2206–2217, 2023.

[11] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784, 2014.

[12] Yuval Netzer, Tao Wang, Adam Coates, A. Bissacco, Bo Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[13] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In International conference on machine learning, pages 8162–8171. PMLR, 2021.

[14] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In International conference on machine learning, pages 2256–2265. PMLR, 2015.

[15] Kunpeng Song, Ligong Han, Bingchen Liu, Dimitris Metaxas, and Ahmed Elgammal. Diffusion guided domain adaptation of image generators. arXiv preprint arXiv:2212.04473, 2022.

[16] Peter Spirtes, Clark N Glymour, and Richard Scheines. Causation, prediction, and search. MIT press, 2000.

[17] Baochen Sun, Jiashi Feng, and Kate Saenko. Correlation alignment for unsupervised domain adaptation. CoRR, abs/1612.01939, 2016.

[18] Kun Zhang, Mingming Gong, Petar Stojanov, Biwei Huang, Qingsong Liu, and Clark Glymour. Domain adaptation as a problem of inference on graphical models. Advances in neural information processing systems, 33:4965–4976, 2020.

[19] Han Zhao, Shanghang Zhang, Guanhang Wu, José M. F. Moura, Joao P Costeira, and Geoffrey J Gordon. Adversarial multiple source domain adaptation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018.