Evaluating the Impact of Image-to-Image Translation on Unsupervised Domain Adaptation Performance

Stathopoulos Dimitris
AUEB Msc In Data Science
Email: dim.stathopoulos@aueb.gr

***Abstract*—Unsupervised domain adaptation (UDA) aims to leverage labeled data from a source domain to improve performance in a target domain where labeled data is scarce or unavailable. In this study, we investigate the potential of UDA techniques using standard benchmarks such as MNIST, USPS, and SVHN. We evaluate the effectiveness of three models: a VGG-16 architecture augmented with Deep CORAL (Deep Correlation Alignment), a simple Convolutional Neural Network (CNN) with an integrated domain discriminator, and the Domain-Adversarial Neural Network (DANN). Despite employing these established UDA techniques, the models exhibit varying degrees of success in aligning feature representations between the source and target domains, often struggling with substantial domain shifts.**

To further explore the possibilities of enhancing the adaptation process, we introduce CycleGAN for image-to-image translation, aiming to generate translated images that resemble the target domain. Our goal is to create a new source dataset that potentially bridges the domain gap. However, our experimental results indicate that the incorporation of CycleGAN for domain adaptation does not consistently yield significant improvements in target domain performance.

This study serves as a critical examination of the efficacy of combining generative models with traditional UDA techniques. While the anticipated benefits were not realized, the findings provide valuable insights into the complexities of domain adaptation and highlight the challenges faced in achieving robust and generalized solutions. Future work will continue to refine these approaches and explore alternative methods to address these persistent issues.

I. Introduction

Unsupervised domain adaptation (UDA) is a critical area in machine learning that aims to transfer knowledge from a labeled source domain to an unlabeled target domain. This task is particularly challenging when there is a significant discrepancy between the source and target domains, a scenario commonly referred to as domain shift [1][2][3][4]. Various techniques have been proposed to address this issue, including feature alignment methods like Deep CORAL Sun et al.[5] and adversarial approaches such as the Domain-Adversarial Neural Network (DANN) Ganin et al.[6].

Deep CORAL reduces domain shift by aligning the second-order statistics of source and target feature distributions. In contrast, DANN employs a domain discriminator to encourage indistinguishability between source and target features, effectively creating a domain-invariant representation. Despite these advancements, achieving robust and generalized domain adaptation remains a significant challenge, particularly for complex and highly divergent domains.

Recently, generative models like Cycle-Consistent Generative Adversarial Networks (CycleGAN) Zhu et al.[7] have been explored for domain adaptation. CycleGANs are capable of learning mappings between two domains without paired examples, making them an attractive option for generating synthetic data that resembles the target domain. The hypothesis is that by transforming the source domain data to appear similar to the target domain, models trained on this transformed data could perform better on the target task.

In this study, we investigate the potential of CycleGAN to enhance traditional UDA methods. We apply CycleGAN to generate target-like images from the source domain and evaluate the performance of several UDA models, including VGG-16 with Deep CORAL, a simple CNN with a domain discriminator, and DANN, on standard benchmark datasets such as MNIST, USPS, and SVHN. While previous work has shown promising results with GAN-based approaches in some contexts Hoffman et al.[8], our experiments reveal the nuanced and often limited benefits of this technique for UDA.

Our contributions are twofold: first, we provide an empirical evaluation of the integration of CycleGAN with traditional UDA methods; second, we offer insights into the challenges and limitations of using generative models for domain adaptation. This study aims to inform future research directions by highlighting the complexities and potential pitfalls in combining these approaches.

II. Related Work

Unsupervised domain adaptation (UDA) has seen significant progress in recent years, with various techniques proposed to address the challenges of domain shift. Among the most notable approaches are those leveraging deep learning models like VGG-16 and domain alignment techniques such as Deep CORAL and adversarial methods like DANN.

A. VGG-16 w/ DeepCORAL

The VGG-16 architecture, introduced by Simonyan and Zisserman (2015)[9], has been widely adopted in computer vision tasks due to its deep convolutional structure and ability to learn rich feature representations. To address domain adaptation, Deep CORAL (Sun Saenko, 2016)[10] extends VGG-

16 by incorporating a correlation alignment (CORAL) loss that aligns the second-order statistics of the source and target feature distributions. This approach effectively minimizes the domain shift by ensuring that the covariance matrices of the source and target features are similar.

Deep CORAL has demonstrated promising results in various domain adaptation tasks. For instance, in the adaptation from the MNIST to USPS datasets, Deep CORAL with VGG-16 achieves competitive performance by reducing the domain discrepancy (Sun et al., 2016)[10]. Similarly, when applied to more complex datasets like Office-31, Deep CORAL has shown to significantly improve the classification accuracy by aligning feature distributions across domains (Sun et al., 2016)[10].

B. Domain-Adversarial Neural Network (DANN)

Another prominent approach in UDA is the Domain-Adversarial Neural Network (DANN), introduced by Ganin et al. (2016)[6]. DANN employs a gradient reversal layer that allows the model to learn domain-invariant features by adversarially training a domain discriminator alongside the primary task classifier. The domain discriminator aims to distinguish between the source and target features, while the feature extractor tries to deceive the discriminator, resulting in domain-invariant feature learning.

DANN has been extensively evaluated on several benchmark datasets, demonstrating robust performance across different domain adaptation scenarios. For example, in the adaptation from SVHN to MNIST, DANN achieves high accuracy by effectively aligning the feature distributions (Ganin et al., 2016)[6]. Additionally, DANN has been successfully applied to more challenging domain adaptation tasks, such as those involving the Office-31 dataset, where it consistently outperforms traditional non-adversarial methods (Ganin et al., 2016)[6].

Both VGG-16 with Deep CORAL and DANN represent significant advancements in UDA, offering different but complementary strategies to address the domain shift problem. While Deep CORAL focuses on aligning statistical properties of feature distributions, DANN leverages adversarial training to achieve domain invariance. These approaches have set benchmarks for acceptable performance in domain adaptation tasks, providing a foundation for further research and development.

C. CycleGAN as a Preprocessing technique

Recent advances in unsupervised image-to-image translation have introduced the CycleGAN model, which can be effectively used as a preprocessing step to transform source images to resemble the target domain. CycleGAN, or Cycle-Consistent Generative Adversarial Networks, leverages the concept of cycle consistency to learn mappings between two domains without the need for paired examples [7]. By using CycleGAN, source domain images can be translated to mimic the visual characteristics of the target domain, thus reducing the domain shift and improving the performance of downstream tasks [8]. This preprocessing step can be particularly beneficial in scenarios where the source and target domains have significant differences, as it helps in aligning the feature distributions and making the subsequent learning tasks more effective.

III. Methodology

In this section, we describe the datasets, models, and experimental setup used in our study. We detail the implementation of VGG-16 with Deep CORAL, the simple CNN with a domain discriminator, and DANN. We also outline the use of CycleGAN for image-to-image translation to generate a transformed source dataset.

A. Datasets

We utilize three standard benchmark datasets for domain adaptation: MNIST, USPS, and SVHN. MNIST as shown in fig.1 is a digit recognition dataset , while SVHN as shown in fig.2 consists of street view house numbers, posing a more complex domain adaptation challenge due to its significantly different visual characteristics.

Fig. 1: This is an example of MNIST dataset.

Fig. 2: This is an example of SVHN dataset.

B. Image Preprocessing

To ensure dimensional consistency and reduce computational complexity, we utilize PyTorch's DataLoader and transformation functions to preprocess the images. Specifically, we resize the images to 32x32 pixels and convert them to three grayscale channels. The resizing step guarantees that all images have uniform dimensions, thereby preventing any issues related to dimensional mismatches. Converting the RGB images to grayscale serves to simplify the unsupervised domain adaptation process, reducing the complexity associated with handling color information. This preprocessing step is crucial for maintaining the integrity and efficiency of the subsequent analysis. Figures 4 and 4 illustrate the preprocessing applied to the MNIST and SVHN datasets, respectively.

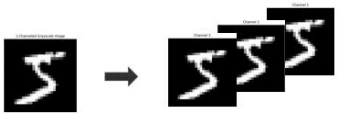

Fig. 3: Preprocessing sample of MNIST dataset.

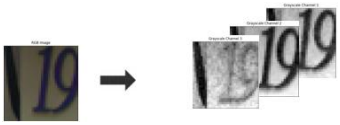

Fig. 4: Preprocessing sample of SVHN dataset.

C. Models

VGG-16 with Deep CORAL: We implement the VGG-16 model modifying to accept 32x32 size images on the first Conv2D layer(algorithm 1) and extend it with the CORAL loss, which aligns the second-order statistics of the source and target domain features.

Simple CNN with Domain Discriminator: This model consists of a basic CNN for feature extraction, coupled with a domain discriminator network that classifies features as belonging to either the source or target domain. The feature extractor is trained adversarially to produce domain-invariant features.

Domain-Adversarial Neural Network (DANN): DANN incorporates a gradient reversal layer that inverts the gradient from the domain discriminator, promoting the learning of domain-invariant features by the feature extractor.

D. CycleGAN for image Translation

To further investigate the potential of enhancing UDA, we employ CycleGAN to translate images from the source domain to the target domain style. CycleGAN uses cycle-consistent adversarial networks to generate realistic target-like images from the source images without requiring paired examples.

E. Experimental Setup

We train the models using the original source datasets and the CycleGAN-transformed source datasets. We evaluate the performance of each model on the target domain using standard classification metrics. The training and evaluation protocols are consistent across all experiments to ensure a fair comparison.

1) SimpleCNN w/ Discriminator: In this work, we adopt a training methodology inspired by recent advances in domain adaptation using deep learning. We employ a variant of the SimpleCNN architecture enhanced with a domain discriminator (Algorithm 1). This approach facilitates concurrent learning of task-specific features and domain-invariant representations through adversarial training. The SimpleCNN model is trained on labeled source domain data, while the domain discriminator distinguishes between source and target domain features, promoting better generalization to unseen target domain data. Our experiments demonstrate that integrating domain adversarial learning significantly enhances model transferability across domains, leading to improved performance on target domain tasks.

Algorithm 1 SimpleCNN with Domain Discriminator Training Algorithm

Input: SimpleCNN model `model`, domain discriminator `domain_model`, source data loader `src_loader`, target data loader `tgt_loader`, loss function `criterion`, SGD optimizer `opt`, training parameters `params`, number of epochs `epochs`
Output: Trained SimpleCNN model with domain discriminator
`model.train()`
`domain_model.train()`
for epoch in `range(epochs)` **do**
  for `(src_data, src_labels), (tgt_data, _)` in `zip(src_loader, tgt_loader)` **do**
    Train feature extractor and classifier on source data
    `opt.zero_grad()`
    `src_feat` ← `model.feature_extractor(src_data)`
    `src_out` ← `model.clf(src_feat.view(src_feat.size(0), -1))`
    `src_loss` ← `criterion(src_out, src_labels)`
    `src_loss.backward()`
    `opt.step()`
    Train domain discriminator on source and target data
    `opt.zero_grad()`
    `dom_labels` ← `torch.cat((torch.zeros(src_data.size(0)), torch.ones(tgt_data.size(0)))).long()`
    `comb_data` ← `torch.cat((src_data, tgt_data))`
    `comb_feat` ← `model.feature_extractor(comb_data)`
    `dom_out` ← `dom_model(comb_feat.view(comb_feat.size(0), -1))`
    `dom_loss` ← `criterion(dom_out, dom_labels)`
    `dom_loss.backward()`
    `opt.step()`
  end for
end for
=0

Fig. 5: Pseudocode for training SimpleCNN with a domain discriminator.

2) Pre-Training VGG-16: In our approach, we pre-trained the VGG-16 model on the MNIST and SVHN datasets following a structured algorithm comprising both training and validation steps (see Algorithm 2). During each epoch, the model was trained using the training data loader with stochastic gradient descent optimization. For each batch, the loss was computed using a specified criterion, followed by backpropagation and parameter updates. Post-training, the model's performance was evaluated using a validation data loader, calculating the validation loss and accuracy without updating the model's parameters. This ensured that the VGG-16 model was effectively pre-trained, balancing both learning and generalization across the dataset.

3) Pre-Trained VGG-16 w/ DeepCORAL: In our study, we adopt a training strategy leveraging the CORAL (CORrelation ALignment) loss to enhance domain adaptation in deep learning models. Algorithm 3 outlines our training procedure, where we simultaneously train a neural network on source domain (SVHN) and target domain (MNIST) data. During each iteration, the model is optimized using the cross-entropy loss for classification on the source domain, while incorporating the CORAL loss to minimize domain discrepancy between feature distributions of the source and target domains. This adversarial approach helps the model learn domain-invariant

Algorithm 2 VGG-16 Training Algorithm with Validation

```
Input: VGG-16 model vgg16, training data loader trainloader, validation data loader val_loader, loss function criterion, SGD optimizer optimizer, training parameters params
Output: Trained VGG-16 model
for epoch in range(params['num_epochs']) do
    vgg16.train()
    running_loss ← 0.0
    for images, labels in trainloader do
        optimizer.zero_grad()
        Forward pass
        outputs ← vgg16(images)
        loss ← criterion(outputs, labels)
        Backward pass and optimization
        loss.backward()
        optimizer.step()
        running_loss←running_loss+loss.item()
    end for
    Validation step
    vgg16.eval()
    val_loss ← 0.0
    correct ← 0
    total ← 0
    with torch.no_grad():
    for images, labels in val_loader do
        outputs ← vgg16(images)
        loss ← criterion(outputs, labels)
        val_loss ← val_loss + loss.item()
        predicted←torch.max(outputs.data, 1)
        total ← total + labels.size(0)
        correct←correct+(predicted==labels).sum().item()
    end for
    val_loss ← val_loss / len(val_loader)
    val_accuracy ← 100 × correct / total
end for
=0
```

Fig. 6: Pseudocode for training VGG-16 with validation step.

representations, crucial for improving performance on target domain tasks. By integrating CORAL loss into our training framework, we aim to achieve robust and generalized deep learning models capable of handling domain shift challenges effectively.

Algorithm 3 VGG-16 w/ DeepCORAL Algorithm

```
Input: VGG-16 w/ DeepCORAL model, source data loader src_trainloader, target data loader tgt_trainloader, loss function criterion, Adam optimizer optimizer,
Output: Trained VGG-16 model
model.train()
result = []
src, tgt = list(iter(src_trainloader)), list(iter(tgt_trainloader))
train_steps = min(len(src), len(tgt))
for batch_idx in range(train_steps) do
    src_data, src_label = src[batch_idx]
    tgt_data, _ = tgt[batch_idx]
    src_data, src_label = Variable(src_data)), Variable(src_label)
    tgt_data = Variable(tgt_data))
    optimizer.zero_grad()
    out1, out2 = model(src_data), model(tgt_data)
    clf_loss = F.cross_entropy(out1, src_label)
    coral_loss = CORAL(out1, out2)
    total_loss = _lambda * coral_loss + clf_loss
    total_loss.backward()
    optimizer.step()
end for=0
```

Fig. 7: Training Procedure with CORAL Loss

4) Preprocessing Step (CycleGAN): CycleGAN, or Cycle-Consistent GAN, is a type of generative adversarial network for unpaired image-to-image translation. For two domains, i.e. X is MNIST and Y is SVHN, our objective is to make MNIST look like SVHN to aid unsupervised domain adaptation. CycleGAN learns mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$. The novelty lies in trying to enforce the intuition that these mappings should be reverses of each other and that both mappings should be bijections. This is achieved through a **cycle consistency loss** that encourages $F(G(x)) \approx x$ and $G(F(y)) \approx y$. Combining this loss with the adversarial losses on X and Y yields the full objective for unpaired image-to-image translation.

For the mapping $G : X \rightarrow Y$ and its discriminator D_Y, we have the objective:

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] \\ + \mathbb{E}_{x \sim p_{\text{data}}(x)}\left[\log(1 - D_Y(G(x)))\right]\end{aligned} \tag{1}$$

where G tries to generate images $G(x)$ that look similar to images from domain Y, while D_Y tries to discriminate between translated samples $G(x)$ and real samples y. A similar loss is postulated for the mapping $F : Y \rightarrow X$ and its discriminator D_X.

The **Cycle Consistency Loss** reduces the space of possible mapping functions by enforcing forward and backward consistency:

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}\|F(G(x)) - x\|_1 \\ + \mathbb{E}_{y \sim p_{\text{data}}(y)}\|G(F(y)) - y\|_1\end{aligned} \tag{2}$$

The full objective is:

$$\begin{aligned}\mathcal{L}_{GAN}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) \\ + \mathcal{L}_{GAN}(F, D_X, Y, X) \\ + \lambda \mathcal{L}_{\text{cyc}}(G, F)\end{aligned} \tag{3}$$

where we aim to solve:

$$G^*, F^* = \arg\min_{G,F} \max D_X, D_Y \mathcal{L}_{GAN}(G, F, D_X, D_Y)$$

5) DANN Training: In this study, we employ a comprehensive training loop for both Source Only and Domain-Adversarial Neural Network (DANN) models, as illustrated in Algorithm 4. The algorithm begins by initializing the encoder (E), classifier (C), and discriminator (D) for DANN, along with the respective source and target training loaders (S_{train} and T_{train}) and necessary parameters. For each epoch, the encoder and classifier, and discriminator in the case of DANN, are set to training mode. The algorithm calculates the current progress (p) to adjust the learning rate scheduler and the domain adaptation parameter (α). For each batch of source and target data, classification loss is computed using the source data, and domain loss is computed using both source and target data if using DANN. The total loss, comprising classification and domain losses, is backpropagated to update the model parameters. The training process includes periodic progress logging and concludes with an evaluation of the trained models

on both source and target test datasets. This method ensures robust training and domain adaptation, facilitating improved generalization across domain.

Algorithm 4 Training Loop for Source Only and DANN

Input: Encoder E, Classifier C, Discriminator D (for DANN), Source Training Loader S_{train}, Target Training Loader T_{train}, Parameters $params$
Output: Trained Models E, C, (and D for DANN)
Initialize loss functions: $\text{Loss}_{\text{class}} \leftarrow$ CrossEntropyLoss()
Initialize optimizer with parameters from E, C (and D for DANN)
for epoch e from 1 to $params$['epochs'] **do**
  Set E and C (and D for DANN) to training mode
  $start\_steps \leftarrow e \times \text{len}(S_{\text{train}})$
  $total\_steps \leftarrow params$['epochs'] $\times \text{len}(T_{\text{train}})$
  for batch_idx, $(b_s, b_t) in$enumerate(zip($S_{\text{train}}, T_{\text{train}}$)) **do**
    $(s\_images, s\_labels) \leftarrow b_s$
    $(t\_images, t\_labels) \leftarrow b_t$
    $p \leftarrow \frac{(batch\_idx + start\_steps)}{total\_steps}$
    optimizer $\leftarrow$ optimizer_scheduler(optimizer, p)
    optimizer.zero_grad()
    $s\_features \leftarrow E(s\_images)$
    Classification Loss:
    $s\_class\_pred \leftarrow C(s\_features)$
    $class\_loss \leftarrow \text{Loss}_{\text{class}}(s\_class\_pred, s\_labels)$
    if Using DANN **then**
      $alpha \leftarrow \frac{2}{1+\exp(-10 \times p)} - 1$
      $combined\_images \leftarrow \text{concat}(s\_images, t\_images)$
      $combined\_features \leftarrow E(combined\_images)$
      Domain Loss:
      $domain\_pred \leftarrow D(combined\_features, alpha)$
      $domain\_labels \leftarrow$ concat(zeros_like(s_labels), ones_like(t_labels))
      $domain\_loss \leftarrow \text{Loss}_{\text{class}}(domain\_pred, domain\_labels)$
    $total\_loss \leftarrow class\_loss + domain\_loss$

    else
      $total\_loss \leftarrow class\_loss$

    end if
      $total\_loss$.backward()
      optimizer.step()

    if (batch_idx + 1) % 100 == 0 **then**
      Print progress

    end if

  end for
    Evaluate: tester($E, C, (D), S_{\text{test}}, T_{\text{test}}$, 'Source_only' or 'DANN')
end for
Save models and generate visualizations
=0

Fig. 8: The training loop for Source Only and Domain Adversarial Neural Network (DANN) methods.

IV. Experimental Results

We conduct extensive experiments to evaluate the effectiveness of VGG-16 with Deep CORAL, the simple CNN with domain discriminator, and DANN on the task of domain adaptation. We present quantitative results in terms of classification accuracy and provide qualitative analyses of the feature alignments achieved by each method.

A. VGG-16 Pre-Training

Pre-training of the VGG-16 model took place for MNIST and SVHN datasets seperatly, thus, resulting in two in total pre-trained VGG-16 classifiers. The train,validation and test sets where comprised of 84% of the original training set (from torchvision) with the remaining percentage acting as a validation set and the test set was 10000 samples, for both datasets.

VGG-16	Validation	Set Test Set
Loss	0.0105	0.0101
Accuracy	99.73%	99.78%

TABLE I: MNIST VGG-16 pre-training results.

The pre-training results for the VGG-16 model on the MNIST dataset are summarized in Table I. The model achieved an impressive validation loss of 0.0105 and a test set loss of 0.0101. In terms of accuracy, the model performed exceptionally well, with a validation accuracy of 99.73% and a test set accuracy of 99.78%. These results indicate that the VGG-16 model is highly effective at learning from the MNIST training data, generalizing well to both the validation and test sets.

VGG-16	Validation	Set Test Set
Loss	0.0624	0.0688
Accuracy	98.27%	98.16%

TABLE II: SVHN VGG-16 pre-training results.

In contrast, the pre-training results for the VGG-16 model on the SVHN dataset, as shown in Table II, were slightly less impressive. The validation loss was recorded at 0.0624, while the test set loss was slightly higher at 0.0688. The model achieved a validation accuracy of 98.27% and a test set accuracy of 98.16%. Although the performance on the SVHN dataset is still quite strong, it is evident that the model had a harder time generalizing compared to the MNIST dataset, as indicated by the higher loss values and lower accuracy percentages.

B. Results on UDA

Following we present the results from our UDA operations on three different models. Specifically we use USPS, MNIST and SVHN as previously explained.

The performance of different models when the source domain is MNIST and the target domain is SVHN is presented in Table III. The baseline model achieves a high source accuracy of 94.26% but suffers a significant drop in target accuracy, reaching only 19.99%. The pre-trained VGG-16-DC model shows a marked improvement with a source accuracy of 99.07% and a target accuracy of 29.11%. The DANN model also improves upon the baseline with a source accuracy of 96.48% and a target accuracy of 21.33%, indicating better generalization to the target domain compared to the baseline but not as effective as the pre-trained VGG-16-DC.

Models	Souce Accuracy	Target Accuracy
Baseline	94.26%	19.99%
Pre-Tr VGG-16-DC	99.07%	29.11%
DANN	96.48%	21.33%

TABLE III: Source: MNIST, Target: SVHN and VGG-16 is pre-trained on MNIST.

When the source domain is transformed-MNIST and the target domain is SVHN, the results change significantly as shown in Table IV. The baseline model struggles in this scenario, with a source accuracy of 14.43% and a target accuracy of 11.57%. The pre-trained VGG-16-DC model performs much better, achieving a source accuracy of 84.73% and a target accuracy of 27.55%. The DANN model shows moderate performance improvements over the baseline, with a source accuracy of 15.82% and a target accuracy of 18.34%, highlighting the challenge of domain adaptation between these datasets.

Models	Souce Accuracy	Target Accuracy
Baseline	14.43%	11.57%
Pre-Tr VGG-16-DC	84.73%	27.55%
DANN	15.82%	18.34%

TABLE IV: Source: transformed-MNIST, Target: SVHN and VGG-16 is pre-trained on MNIST.

Table V presents the results when the source domain is SVHN and the target domain is MNIST, with the VGG-16 model pre-trained on SVHN. The baseline model performs reasonably well with a source accuracy of 85.10% and a target accuracy of 55.01%. The pre-trained VGG-16-DC model exhibits the best performance, with a source accuracy of 91.21% and a target accuracy of 70.77%. The DANN model, however, shows an unusual pattern with a much lower source accuracy of 30.80% but a target accuracy almost identical to the baseline at 55.00%. This suggests that while the DANN model may be more effective at adapting to the target domain, it does so at the expense of performance on the source domain.

Models	Souce Accuracy	Target Accuracy
Baseline	85.10%	55.01%
Pre-Tr VGG-16-DC	91.21%	70.77%
DANN	30.80%	55.00%

TABLE V: Source: SVHN, Target: MNIST and VGG-16 is pre-trained on SVHN.

C. CycleGAN results

The figure below demonstrates an attempt to transform the MNIST dataset into the appearance of the SVHN dataset using image-to-image translation with cycleGAN. The top row displays the original MNIST images, while the bottom row showcases the transformed images meant to resemble the SVHN style. Although the results are not particularly promising in terms of achieving a high-quality transformation, there is a noticeable visual change. The generator appears to have inherited features from both datasets, resulting in the transformed MNIST images exhibiting some characteristics of the SVHN dataset. This includes a more complex and textured background, as well as distortions in the digits. However, the core structure of the MNIST digits remains relatively intact, indicating that while there is some degree of feature transfer, the transformation is not yet fully successful.

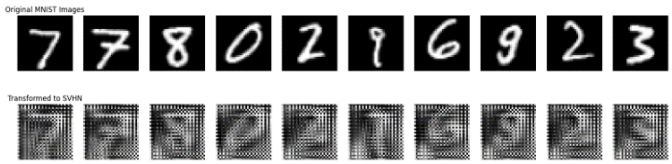

Fig. 9: An attempt to transform the MNIST dataset with image-to-image translation to resemble the SVHN dataset using cycleGAN. The top row shows original MNIST images, while the bottom row shows the transformed images.

V. Conclusions

In summary, the experiments underscore the inherent challenges of domain adaptation, particularly when transitioning between datasets with distinct characteristics. The relatively limited feature space of MNIST presents a significant hurdle, as it lacks the complexity found in more diverse datasets like SVHN. This limitation was further compounded by the absence of fine-tuning, which adversely impacted the performance of the models. Notably, some baseline models, despite their simplicity, managed to outperform more specialized architectures, highlighting the potential pitfalls of over-engineering solutions without adequate dataset-specific adjustments. These findings suggest that there is substantial room for improvement in domain adaptation techniques. Future work should focus on enhancing feature extraction processes, incorporating robust fine-tuning strategies, and exploring more sophisticated models that can better generalize across domains.

References

[1] Gabriela Csurka. *Domain Adaptation for Visual Applications: A Comprehensive Survey*. 2017. arXiv: 1702.05374 [cs.CV]. URL: https://arxiv.org/abs/1702.05374.

[2] Haowei Wang et al. *Enhanced Global Optimization with Parallel Global and Local Structures*. 2022. arXiv: 2201.10255 [math.OC]. URL: https://arxiv.org/abs/2201.10255.

[3] Khac-Hoang Ngo et al. *Joint Constellation Design for Noncoherent MIMO Multiple-Access Channels*. 2022. arXiv: 2009.11548 [cs.IT]. URL: https://arxiv.org/abs/2009.11548.

[4] Yaroslav Ganin and Victor Lempitsky. *Unsupervised Domain Adaptation by Backpropagation*. 2015. arXiv: 1409.7495 [stat.ML]. URL: https://arxiv.org/abs/1409.7495.

[5] Baochen Sun, Jiashi Feng, and Kate Saenko. *Return of Frustratingly Easy Domain Adaptation*. 2015. arXiv: 1511.05547 [cs.CV]. URL: https://arxiv.org/abs/1511.05547.

[6] Yaroslav Ganin et al. *Domain-Adversarial Training of Neural Networks*. 2016. arXiv: 1505.07818 [stat.ML]. URL: https://arxiv.org/abs/1505.07818.

[7] Jun-Yan Zhu et al. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. arXiv: 1703.10593 [cs.CV]. URL: https://arxiv.org/abs/1703.10593.

[8] Judy Hoffman et al. *CyCADA: Cycle-Consistent Adversarial Domain Adaptation*. 2017. arXiv: 1711.03213 [cs.CV]. URL: https://arxiv.org/abs/1711.03213.

[9] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[10] Baochen Sun and Kate Saenko. *Deep CORAL: Correlation Alignment for Deep Domain Adaptation*. 2016. arXiv: 1607.01719 [cs.CV]. URL: https://arxiv.org/abs/1607.01719.