2-c: Domain Adaptation

Domain adaptation tackles the problem of training a model on one dataset (source domain) and ensuring it performs well on a different dataset (target domain) with distinct distributions. In this practical, we will implement **Domain-Adversarial Neural Networks (DANN)**, a method that uses adversarial training to help the model learn domain-invariant feature representations, and analyse how it works in domain adaptation.

1 DANN and the GRL layer

Domain-Adversarial Neural Networks (DANN) are designed to tackle the challenge of domain adaptation, where the goal is to train a model on a labeled source domain while ensuring it performs well on an unlabeled target domain with distinct feature distributions.DANN introduces a novel **Gradient Reversal Layer (GRL)** to achieve domain invariance, enabling the model to bridge the gap between different domains effectively.

1.1 Architecture of the DANN model

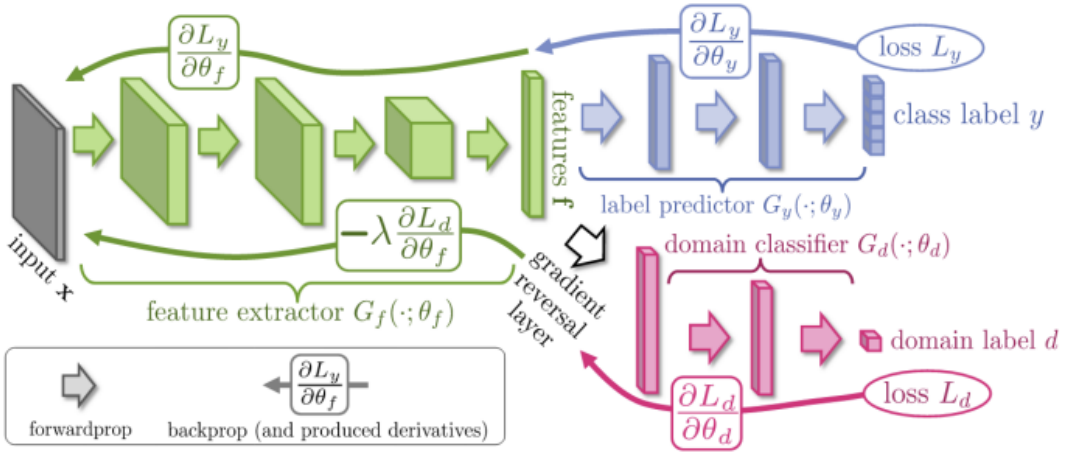


Figure 1: DANN model with its GRL layer

As it is shown in the **Figure 1**, the DANN model consists of three main components:

Component	Description
Feature Extractor (G_f)	A neural network that learns representations of input data.
Label Predictor (G_y)	A classifier trained on labeled source domain data to predict class labels.
Domain Classifier (G_d)	A binary classifier that predicts whether the input data belongs to the source or target domain.

Table 1: Components of the DANN Model

Notably, a **Gradient Reversal Layer (GRL)** is inserted between the feature extractor and the domain classifier.

1.2 General Principle

The training process of DANN simultaneously optimize the loss of Label Predictor (L_y) and the loss of Domain Classifier (L_d).

$$L_y(\theta_f, \theta_y) = \frac{1}{N} \sum_{i=1}^{N} \text{CrossEntropy}(G_y(G_f(x_i; \theta_f), \theta_y), y_i)$$

$$L_d(\theta_f, \theta_d) = \frac{1}{N} \sum_{i=1}^{N} \text{BinaryCrossEntropy}(G_d(G_f(x_i; \theta_f), \theta_d), d_i)$$

where θ_f, θ_y and θ_d are the corresponding parameters for Feature Extractor, Label Predictor and Domain Classifier.

In adversarial training, the feature extractor aims to extract more generalized features that can "confuse" the domain classifier. To achieve this, we minimize the label prediction loss (L_y) to ensure good classification performance while maximizing the domain classification loss (L_d) to reduce the domain classifier's accuracy. This adversarial process helps the feature extractor learn domain-invariant representations.

Then the overall loss function for feature extractor is:

$$E(\theta_f, \theta_y, \theta_d) = L_y(\theta_f, \theta_y) - \lambda L_d(\theta_f, \theta_d)$$

where λ is a trade-off parameter that balances the label prediction and domain alignment tasks (which can be dynamic during the training).

For two classifiers, we aim to enhance their ability to correctly classify the data. In that case, we need to minimize the two corresponding losses.

By backpropagation, each parameter is updated by the following rules:

$$\theta_f \leftarrow \theta_f - \mu \left(\frac{\partial L_y}{\partial \theta_f} - \lambda \frac{\partial L_d}{\partial \theta_f} \right)$$

$$\theta_y \leftarrow \theta_y - \mu \frac{\partial L_y}{\partial \theta_y}$$

$$\theta_d \leftarrow \theta_d - \mu \frac{\partial L_d}{\partial \theta_d}$$

where μ is the learning rate.

We can see the gradient of Domain Classification Loss L_D is only reversed when we update the parameter of Feature Extractor θ_f and that's where we introduce GRL layer.

1.3 GRL Layer

As the key component of DANN is the **Gradient Reversal Layer (GRL)**, it reverses the gradient flowing from the domain classifier to the feature extractor during training as mentioned above, in order to encourage the feature extractor to learn features that make it difficult for the domain classifier to distinguish between the source and target domains.

In general:

- **Forwardly**, it acts as an identity function, passing data from the feature extractor to the domain classifier without modification.
- **Backwardly**, it multiplies the gradient from the domain classifier by a negative scalar, effectively reversing the gradient's direction

As a result, the model learns domain-invariant features, aligning the feature distributions across domains as it learns to "confuse" the domain classifier.

2 Practice

In this section, we firstly built a naive model trained on labeled MNIST (source domain) and evaluated it on unlabeled MNIST-M (target domain) without applying any domain adaptation techniques. Next, we reproduced the DANN model described by Ganin and Lempitsky in their paper "Unsupervised Domain Adaptation by Backpropagation" to assess its effectiveness in addressing domain shifts between MNIST and MNIST-M.

The DANN model extends the naive model by adding a new branch consisting of a GRL and a domain classifier. Here's the architecture of the DANN model we used:

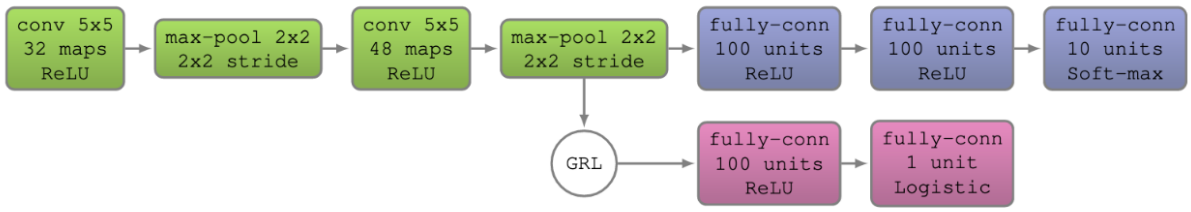


Figure 2: DANN model in paper

Notably, We used a learning rate scheduler and also dynamically adjusted the GRL factor during the training(we will discuss the reasons in Question 3). The GRL factor is upadted at each training step i by:

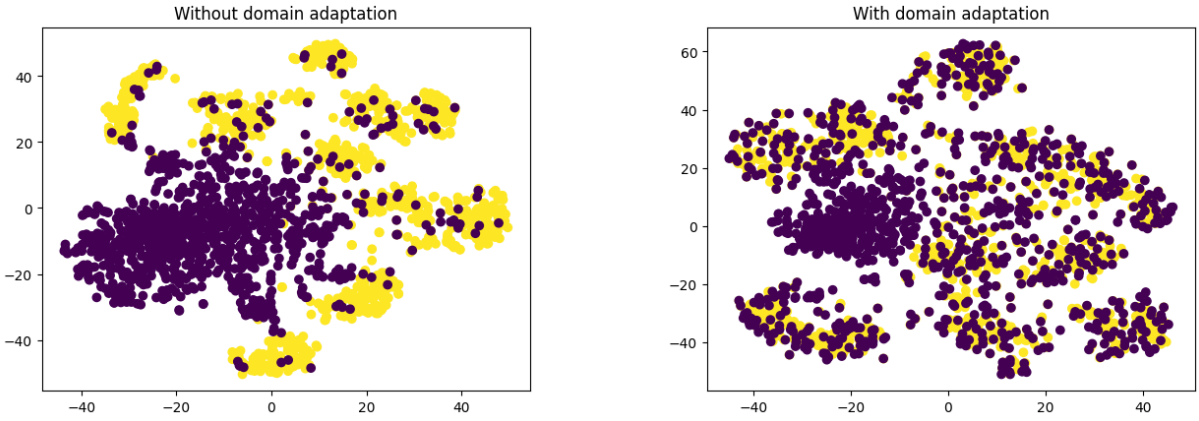
$$\lambda\left(i\right)=-1+\frac{2}{1+\exp\left(\frac{-2\cdot\mathrm{i}}{\text{total training steps}}\right)}$$

2.1 Experiments and results

Based on our experiments, the naive model achieved an impressive accuracy of 98.94% on the source dataset but only 58.9% on the target dataset, highlighting its inability to generalize across domains. In contrast, after 20 epochs of training, our DANN model demonstrated

significant improvements in domain adaptation. On the source dataset, it achieved a class accuracy of 98.62%. More importantly, on the target dataset, the DANN model achieved a class accuracy of 76.33%. These results highlight the effectiveness of DANN in learning domain-agnostic features and improving performance on the target domain.

To gain a better understanding of how well the feature extractor aligns the source and target domains, we utilized t-SNE, a dimensionality reduction technique, to observe patterns and relationships in the feature space more intuitively in a 2D plan.



(a) Naive Model (b) DANN Model

Figure 3: Visualization of the embeddings learned by the networks

The t-SNE visualizations highlight the difference in domain alignment between the Naive Model and the DANN Model.

In the Naive Model (a), the source (purple) and target (yellow) features are clearly separated, indicating that the model learned domain-specific features and failed to generalize to the target domain, as reflected by the low target accuracy of 58.9%.

In contrast, the DANN Model (b) shows significant overlap between the source and target feature distributions, demonstrating successful domain alignment through adversarial training with the Gradient Reversal Layer. This alignment led to a substantial improvement in target domain accuracy (76.33%), though with a slight trade-off in source domain performance. These results confirm the effectiveness of DANN in learning domain-agnostic features for domain adaptation.

Further Study

However, our model still performs below the reported performance in the original paper (81.49%). To improve results, we introduced a step decay learning rate scheduler and extended the number of epochs to 100.

Additionally, the GRL factor is updated at each training step i using the formula:

$$\lambda\left(i\right) = -1 + \frac{2}{1 + \exp\left(\frac{-10 \cdot \mathrm{i}}{\text{total training steps}}\right)}$$

Since the total training steps have increased fivefold (from 20 epochs to 100 epochs), we scaled the rate of λ adjustment to maintain the same growth rate as before.

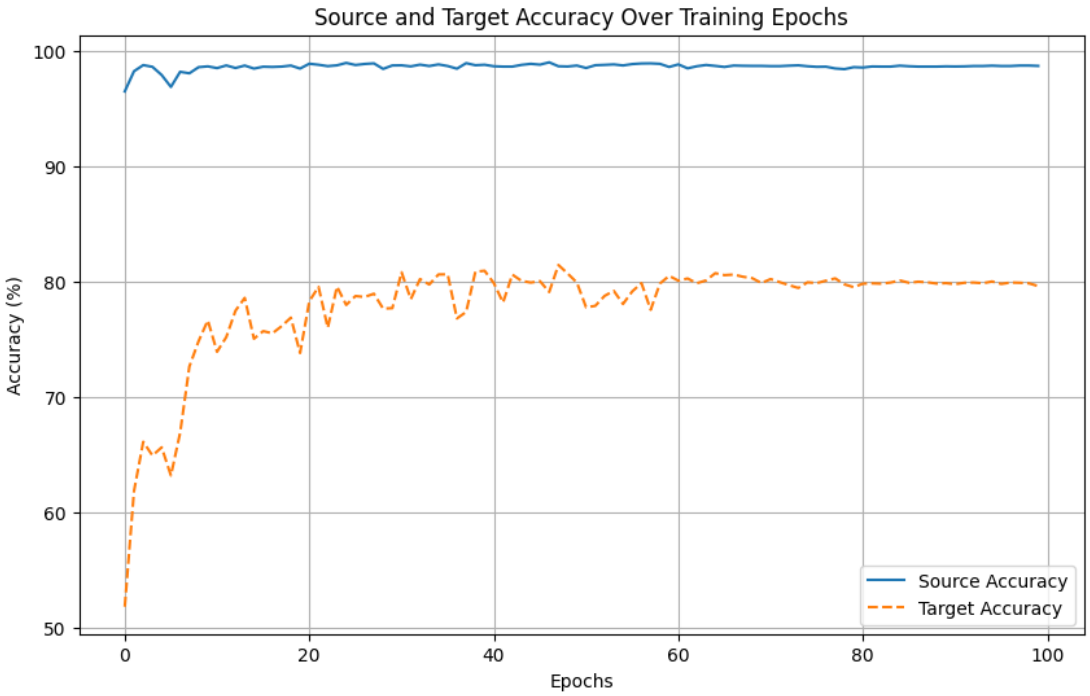


Figure 4: Source and Target Accuracy Over 100 Training Epochs

After a fortunate attempt, our target accuracy stabilized at around 80%, reaching a peak close to the reported performance in the original paper.

2.2 Questions & Answers

Q1 If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen ?

If the Gradient Reversal Layer (GRL) is removed, the feature extractor would no longer receive adversarial gradients from the domain classifier. This means that during the generator's update step, it would aim to minimize the domain classification loss rather than maximize it. As a result, the feature extractor would produce features that are easier for the domain classifier to distinguish between source and target domains, reinforcing the domain-specific nature of the features. This undermines the goal of learning domain-invariant features, leading to poor alignment between source and target domains.

Consequently, the target domain performance would degrade significantly, as the label classifier would fail to generalize. Meanwhile, the source domain performance might improve slightly since the feature extractor no longer faces the competing objective of confusing the domain classifier. Overall, the removal of GRL fundamentally breaks the adversarial dynamic that drives domain adaptation.

Q2 Why does the performance on the source dataset may degrade a bit ?

In our experiment, the source dataset accuracy started at 98.94% and dropped slightly to 98.62% after 10 epochs. This slight decrease can be explained by the fact that the feature extractor is not solely focused on optimizing for the source domain classification task. To achieve a more generalized and domain-invariant representation, it must align the source and target domains, which inevitably sacrifices a small degree of local

specificity in the features that are particularly beneficial for the source domain. This trade-off, aimed at improving the model's ability to adapt to the target domain, results in a slight reduction in the source domain's performance.

Q3 Discuss the influence of the value of the negative number used to reverse the gradient in the GRL.

The value of the negative number used in the Gradient Reversal Layer (GRL) controls the strength of the adversarial signal from the domain classifier to the feature extractor, and it significantly affects the training and model performance.

This trade-off parameter that balances the label prediction and domain alignment tasks. The larger the absolute value of this parameter, the more it pushes the feature extractor to confuse the domain classifier and focus on creating features that work well for both domains.

However, if the value is too large, the feature extractor might focus too much on confusing the domain classifier and lose important details needed for accurate predictions, which could hurt performance on both the source and target datasets.

If it's too small, the feature extractor won't get enough pressure to confuse the domain classifier. This means it will mostly focus on the source domain and won't learn features that generalize well to the target domain. As a result, the domain classifier will easily distinguish between the two domains, and the model's performance on the target dataset will likely remain poor.

If the domain classifier is too weak, trying to confuse it becomes meaningless because the feature extractor won't need to work hard to fool it. This is why we need to dynamically adjust the parameter to find the right balance, ensuring that the training progresses smoothly and steadily. By gradually increasing the adversarial strength, the feature extractor can effectively adapt while avoiding instability or premature convergence.

Q4 Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words.

The concept of pseudo-labeling originates from semi-supervised learning and typically involves three steps:

1. Train the model using labeled source domain data.
2. Use the trained model to predict labels for the unlabeled target domain data, thereby creating pseudo-labels.
3. Retrain the model using both the labeled source domain data and the pseudo-labeled target domain data together.

In practical, in step 3 we often select high-confidence pseudo-labeled target domain data based on the predicted probability.

To conclude, this method heavily relies on the predictions of the initial model. If the initial model's predictions are accurate, the pseudo-labels will be reliable and help improve the model's performance. However, if the initial predictions are poor, the pseudo-labels may contain errors, which can propagate during training and negatively impact the final model.

3 Conclusion

In this practical, we implemented and analyzed the Domain-Adversarial Neural Network (DANN) to tackle the challenge of domain adaptation. By introducing the Gradient Reversal Layer (GRL), we enabled the model to learn domain-invariant features through adversarial training, bridging the gap between source and target domains.

Our experiments demonstrated that the naive model, while achieving high accuracy on the source domain, struggled to generalize to the target domain due to domain shifts. In contrast, the DANN model significantly improved target domain performance, with a target accuracy of 76.33% after 20 epochs and further stabilized at around 80% after extending training to 100 epochs. These improvements were validated through t-SNE visualizations, which highlighted the effective alignment of source and target feature distributions.

In conclusion, the DANN model provides a powerful framework for domain adaptation, enabling models to generalize across domains with distinct distributions.

2-de: Generative Adversarial Networks

1 Conditional Generative Adversarial Networks

In this section, we explore the concept and implementation of Conditional Generative Adversarial Networks (cGANs), which enhance traditional GANs by incorporating additional conditional information. This section covers the general principles of cGANs and their specific application to datasets such as MNIST.

1.1 General Principle

Conditional Generative Adversarial Networks (cGANs) extend traditional GANs by introducing conditional information y into both the generator and discriminator. This conditional information can take various forms, such as class labels, text descriptions, or additional data features, which guide the generation process.

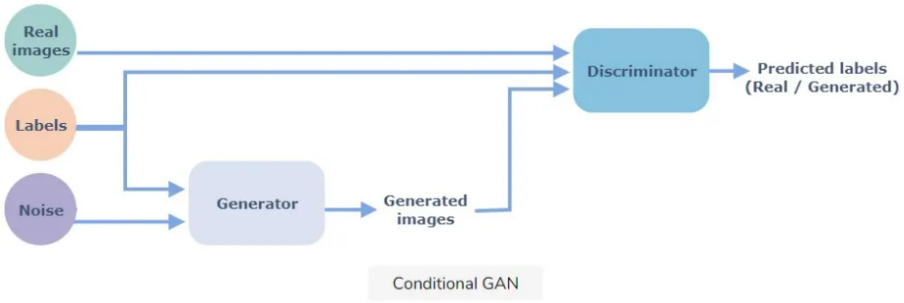


Figure 5: Visualization of the impact of z on generation

In a cGAN, the generator $G(z, y)$ produces a sample $\tilde{x}$ based on a noise vector z and the condition y (extracted from real data). Simultaneously, the discriminator $D(x, y)$ evaluates both the authenticity of the input x and whether it matches the condition y. Mathematically, we have:

$$x^* \in \text{Data}, \quad \tilde{x} = cG(z, y), \quad z \sim P(z), \quad y = attribute(x^*)$$

$$D(x, y) \in [0, 1], \quad \text{ideally}, \quad \begin{cases} D(\tilde{x}, y) = 0, & \tilde{x} = G(z, y) \\ D(x^*, y) = 1, & x^* \in \text{Data} \end{cases}$$

Finally, similar to traditional GAN model, the adversarial optimization objective for a cGAN is as follows:

$$\min_G \max_D \mathbb{E}_{(x^*,y)\sim P_{\text{data}}(x,y)}[\log D(x^*, y)] + \mathbb{E}_{z\sim P(z), y\sim P_{\text{data}}(y)}[\log(1 - D(G(z, y), y))] \quad (1)$$

By including this additional conditional input y, cGANs gain the ability to generate samples that are not only realistic but also aligned with specific conditions, improving both diversity and control over the generated outputs.

1.2 cDCGAN Architectures for MNIST

We follow the previous DCGAN architecture since the behavior of the generator and discriminator remains unchanged. The only modification is that both now take an additional input: the label y. For MNIST, the label y is represented as a one-hot vector of size 10.

The primary difference lies in how the label y is fused with either the noise vector z (for the generator) or the image x (for the discriminator). Additionally, we adjust the number of input channels in the first layer to accommodate the inclusion of y.

For the generator, the label y is concatenated with the noise vector z directly before being processed by the network. For the discriminator, y is expanded spatially along the height and width dimensions to match the dimensions of x, and the two are stacked along the channel dimension as input to the first convolutional layer. These changes ensure that the conditional information is incorporated effectively into the generation and discrimination processes.

1.3 Questions & Answers

Q1 Comment on your experiences with the conditional DCGAN.

Based on our experiments, the conditional DCGAN generates clear and well-formed digits more quickly compared to the standard DCGAN. However, its training time is longer, which is understandable given the added complexity of incorporating conditional information. Additionally, the loss function of the cDCGAN converges more smoothly, indicating a more stable training process.

Q2 Could we remove the vector y from the input of the discriminator (so having $cD(x)$ instead of $cD(x, y)$?

Here are the digits created by the model after modifying the discriminator's input. Some samples look like a mix of different classes, while others don't resemble any recognizable digit.

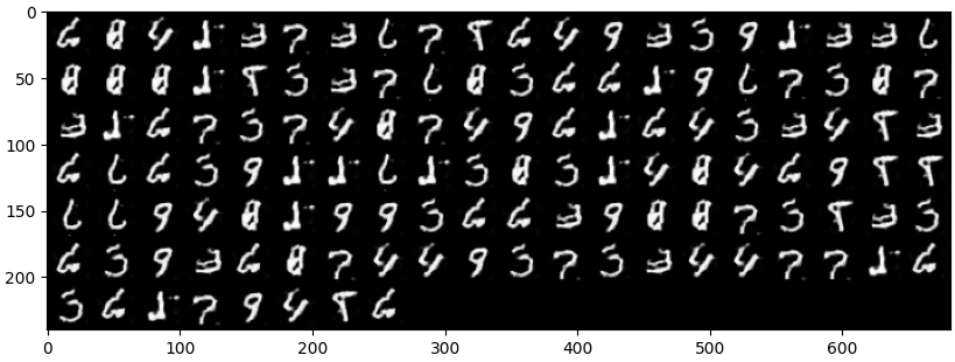
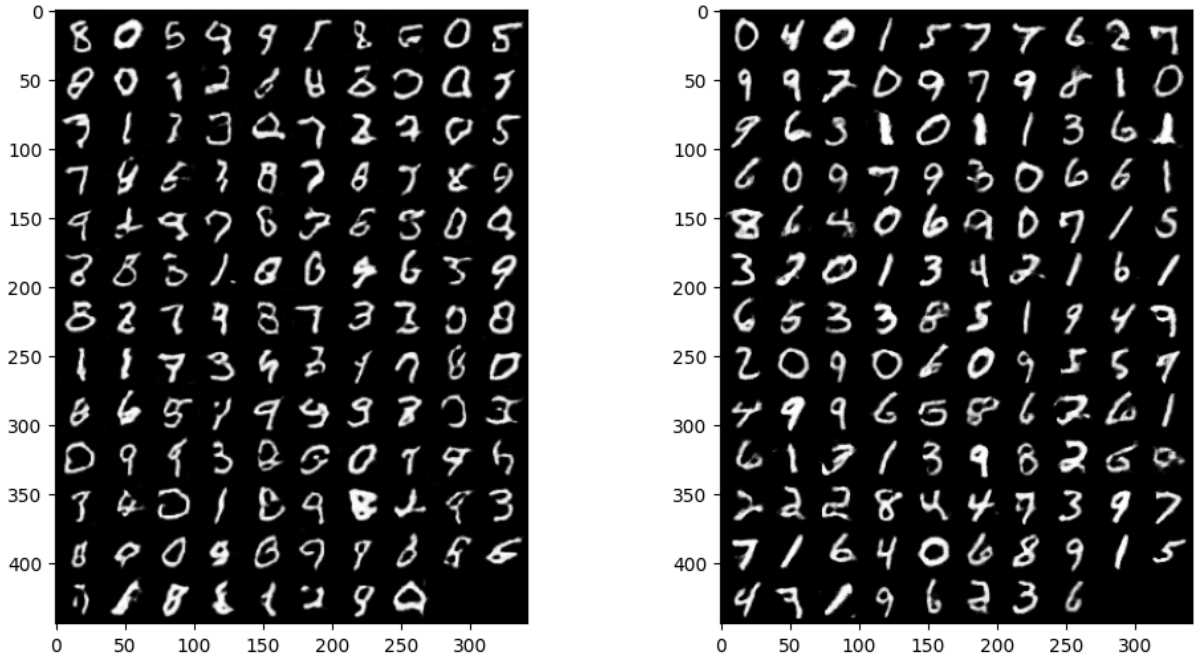


Figure 6: Digits Generated by the Model with Modified Discriminator Input

Obviously, if the vector y is removed from the discriminator's input, it would significantly impact the training and performance of the cDCGAN. This is because the discriminator's

role is not only to differentiate between real and fake images but also to ensure that the generated images align with the given condition y. Without y, the discriminator can no longer check if the generated images match the specified class. As a result, the generator struggles to create digits that match the target classes. Also, removing y makes training harder. The generator has to learn how to produce all kinds of images at the same time without any guidance for specific classes, leading to blurry and unclear images with mixed or inconsistent categories.

Q3 Was your training more or less successful than the unconditional case ? Why ?

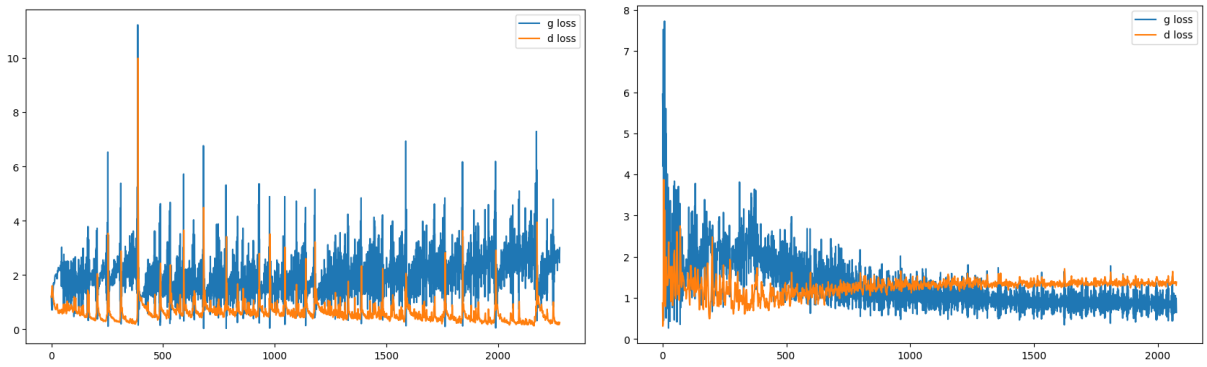


(a) DCGAN (b) cDCGAN

Figure 7: Digits Generated by GAN and cGAN Models

The training of the cDCGAN was significantly more successful compared to the unconditional DCGAN. As shown in Figure 7, the samples generated by the cGAN are much clearer and easier to distinguish.

The cDCGAN leverages the conditional label y, which provides explicit guidance for generating images corresponding to specific classes. This additional information helps the generator focus on creating digits consistent with the class labels, resulting in samples that are not only higher quality but also more diverse within each class. In contrast, the unconditional DCGAN struggles with consistency due to the lack of structured guidance. As a result, many of its samples are distorted or noisy, with some digits being difficult to recognize.



(a) DCGAN (b) cDCGAN

Figure 8: Generator and Discriminator Loss Dynamics in DCGAN and cDCGAN Models over 5 Epochs

Additionally, as shown in Figure 8, the generator and discriminator losses in the DCGAN fluctuate significantly throughout the training process. This reflects the instability commonly associated with adversarial training. In contrast, the cDCGAN exhibits much more stable and convergent loss curves. The conditional label y reduces the complexity of the adversarial game by narrowing the generator's focus to a specific class for each input. This stability not only improves training dynamics but also accelerates convergence.

In conclusion, the cDCGAN achieves more successful training by utilizing the conditional label y, which effectively guides both the generator and discriminator. This guidance enables the cDCGAN to produce higher-quality samples that are consistent with class labels and stabilizes the training process, making it more robust and efficient compared to the unconditional DCGAN.

Q4 Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here ?

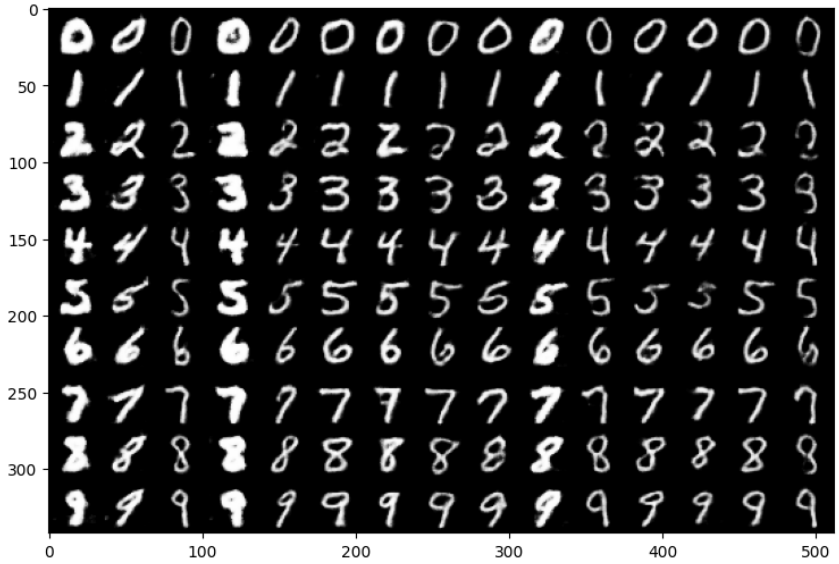


Figure 9: Visualization of the impact of z on generation

From this generated image, we can see that although the class of digits changes across rows, the overall style remains consistent within a column.

In that case, z can be interpreted as a latent vector that controls the style or global characteristics of the generated images. This latent vector captures abstract features such as the thickness of strokes, the slant, or the curvature of the handwritten digits.

In simple terms, z decides the "style," and y decides the "class" of the digit. This separation allows the cDCGAN to create a wide range of images where the digits are the same type but have different styles.

2 Conclusion