# A. ADDA for MNIST-USPS and RealWorld-Clipart with Wasserstein metric

**Paper followed:** Adversarial Discriminative Domain Adaptation by Tzeng et al., Wasserstein GAN by Arjovsky et al., Improved Training of Wasserstein GANs by Gulrajani et al.
**Datasets used:** MNIST (https://pytorch.org/vision/stable/generated/torchvision.datasets.MNIST.html), USPS (https://www.kaggle.com/datasets/bistaumanga/usps-dataset), Office–Home Dataset (https://www.hemanthdv.org/officeHomeDataset.html)
**Framework Used:** PyTorch Lightning (1.6.4), PyTorch (1.11.0), TorchVision (0.12.0)

## 1) Code Details

a) get_dataset: function which gives the required dataset object to be used in the dataloader object with appropriate transforms i.e., resizing, converting to RGB from Grayscale (for MNIST and USPS) and normalising. All these are done as per recommendations to work with Resnet50 architecture.
b) SourceClassifier: Used to train the source classifier. Uses Resnet50's all layers except the last one as the feature extractor. The last layer is now replaced with a layer of output of number of classes (10 for MNIST & USPS, 65 for Office-Home) acting as a classifier. Categorical cross-entropy is used to train the model on the source dataset. Evaluated on a validation dataset. The lowest validation loss epoch is chosen as final.
c) AdversarialAdapter:
    a. Used to train the target feature extractor. It is initialised from the trained source feature extractor (Reasons in the following sections). The source feature extractor is fixed.
    b. A critic which takes as input the source and target features and outputs a score which indicates whether the feature is source / target. Target feature extractor aims to learn features which are similar to source. The network is trained through adversarial Wasserstein loss.
    c. I am additionally using gradient penalty instead of weight clipping for enforcing 1-Lipshitz constraint (Reasons in the following sections).
    d. The number of times the target feature extractor and critic are trained are taken as model parameters.
    e. Best model is chosen based on the EMD estimate on the validation data
d) TargetClassifier: Just a wrapper class which takes the target feature extractor and source classifier to classify the target images. Nothing to be learnt here.
e) train_source_classifier, train_target_featurer & test_target_classifier functions train the respective source classifier, train the target feature extractor and test the performance on the target data respectively. They take parameters such as which GPUs to use etc. Even takes the model_class and the model_kwargs to instantiate the model inside the function. It uses the Trainer class of PyTorch Lightning to train, validate and test the model on the given dataset and the model. It logs everything into a tensorboard (created inside the function), which can then be used to view loss curves.
f) All randomness is controlled by using seeds which ensures reproducibility.

## 2) Architecture Choices & Details:

a) For the target feature extractor, we used pre-trained Resnet initially. All the results were around or below chance. When we used the source trained Resnet and fine-tuned it, we got a bit better results (Suggestion from a classmate). (intuition on why this was the case in later sections). So, we stuck with it.
b) From our experiments in CycleGAN, we saw that using gradient penalty instead of weight clipping gave better results in terms of translations. So we used this version in our experiments.

## 3) Experiments, Results & Inferences:

The results obtained as part of the experiment are summarised in Table 1.

| Source | Target | Source Test Accuracy | Target Test Accuracy |
|---|---|---|---|
| MNIST | USPS | 99.49% | 96.38% |
| USPS | MNIST | 96.79% | 42.62% |
| RealWorld | Clipart | 15.83% | 04.07% |
| Clipart | RealWorld | 25.10% | 15.57% |

Table 1: ADDA Results

Very clearly domain adaptation doesn't seem to help. But interesting result is on USPS dataset, with domain adap we almost reach the same accuracy level. Similarly even on RealWorld dataset, indicating that with maybe some more hyperparameter tuning, we may be able to surpass the source accuracies.

## 4) Problems, Intuitions & Solutions used:

a) On the initial runs, when we were getting around chance / below change results, we thought there is some issue in the code. Every line of code was double / triple checked. We then shifted to GAN loss instead of Wasserstein loss just to verify if everything was fine with the Wasserstein loss coding. We still got chance level accuracies.

b) We then shifted to a simple architecture (from Resnet), just to verify is everything was fine. We found some encouraging results. This reinforced the hope that the code was fine.

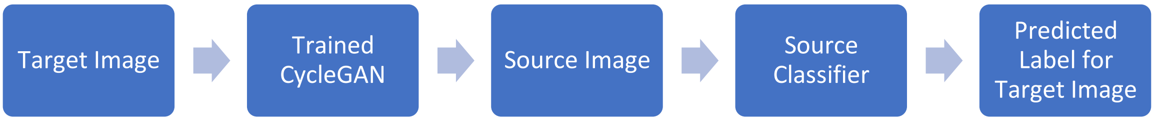
c) Upon suggestion from a classmate, instead of training the target feature extractor from pretrained Resnet, we initialised it from the source feature extractor. We suddenly saw a much better target accuracy.
**Intuition:** The critic is trying to differentiate feature of MNIST from feature of USPS (and respectively in case of Office-Home). The critic does not care which digit it represents. So, target feature extractor could be learning the feature of MNIST "3" for USPS "7" and still successfully confuse critic, leading to well behaving loss curves, but not helping the domain adaptation problem statement at all. The source feature extractor already has the idea of how a "3" should look like, i.e., the concept of a "3" is known by the feature extractor. The target feature extractor now only needs to know the domain specific nitty gritties e.g., USPS target feature extractor needs to learn that the image is bit smaller, more blurry, no padding at the top and bottom etc. sort of things. Although, there is *no guarantee* that the feature extractor may not still have the same issue, it reduces the chances of that happening. This gave a very good example for understanding the marginal and posterior issue which sir raised in the class. When we investigated this in the CycleGAN question, we saw similar thing happening (see next problem report).

## Conclusion & Future Work:

This was a very good exercise to see that not all claims made in papers can be simply replicated (This makes us wonder how the ADDA authors got their results). There could be nuances which need to be understood for working. Personally, we want to see how to improve ADDA using the "fake" target labels idea (as we don't have access to the true target labels in the unsupervised domain adaptation problem setting). A careful & systematic exploration of the hyperparameters could be useful to see how to improve the accuracies further.

## B. Cycle GAN for Domain Adaptation between MNIST-USPS

**Papers followed:** Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks by Zhu et al., Wasserstein GAN by Arjovsky et al., Improved Training of Wasserstein GANs by Gulrajani et al.
**Datasets used:** MNIST (https://pytorch.org/vision/stable/generated/torchvision.datasets.MNIST.html), USPS (https://www.kaggle.com/datasets/bistaumanga/usps-dataset)
**Framework Used:** PyTorch Lightning (1.6.4), PyTorch (1.11.0), TorchVision (0.12.0)


Target Image → Trained CycleGAN → Source Image → Source Classifier → Predicted Label for Target Image

Process for Inference using CycleGAN

### 1) Code Details

a) MUCycleGANGP – Cycle GAN for MNIST-USPS Dataset pair with Gradient Penalty
   a. It has 2 critics – CriticM and CriticU, for MNIST and USPS respectively. Both the critics have 3 down-sampling layers and a linear layer at last to output the score for the image. Each layer is followed by PReLU activation function.
   b. Also has 2 generators – U2MGenerator and M2UGenerator, for translating between USPS to MNIST and MNIST to USPS respectively. Both the generators have 3 down-sampling convolution and 3 up-sampling deconvolution layers, with variable number of channels etc to be consistent with the required data dimensions. Each layer is followed by PReLU activation function and a BatchNorm layer.
   c. The model takes ncritic, ngen, cycle_weight and penalty_weight as parameters. ncritic dictates the frequency with which the critic will be called. ngen dictates the frequency with which the generator will be called. cycle_weight indicates how much weightage to give for cycle consistency loss. penalty_weight indicates the weightage given to the gradient penalty term.
   d. During the critic update, both the critics were updated using the usual Wasserstein loss and gradient penalty. During the generator update, both the generators were updated using the usual Wasserstein loss and the cycle consistency loss. 2 different optimisers were used.
   e. The model with least sum of EMDs on both the MNIST to USPS and USPS to MNIST translations and the cycle consistency loss weighted by the cycle_weight was chosen as the best.

b) USPSClassifier and MNISTClassifier – Classifiers for respective datasets when acting as source. We did not use the ones trained in the previous question as the transformations applied on the datasets were different.

c) train_and_test – General function which allows running different configurations for the CycleGAN. train_and_test_classifier – General function which allows running different configurations for the classifiers. Both take parameters such as which GPUs to use etc. Even takes the model_class and the model_kwargs to instantiate the model inside the function. It uses the Trainer class of PyTorch Lightning to train, validate and test the model on the given dataset and the model. It logs everything into a tensorboard (created inside the function), which can then be used to view loss curves.

d) test_target_accuracies – Function to get the accuracy on the target data.

e) All randomness is controlled by using seeds which ensures reproducibility.

f) plot_translations – To view the translations between the domains in CycleGAN.

### 2) Architecture Choices & Details:

a) The critics do not have a BatchNorm layer following the recommendation in the WGAN-GP paper

b) Cycle Consistency Loss: Ideally, L1 norm should be added across dimensions and mean-ed across samples. In their implementation, authors have mean-ed across dimensions too, so we kept implementation same as them.

## 3) Experiments & Results:

We explored two main hyperparameters – cycle_weight and penalty_weight. We found best results with cycle_weight = 10 and penalty_weight = 1. Figures 1 and 2 are the results on this set of hyperparameters. As we can see the identity reconstruction is very good. The translation between domains are not that great. Many of the numbers from MNIST are getting mapped to USPS “7”. Translations from USPS to MNIST are not giving very good results. We thus do not expect very good target accuracies, but certainly above chance level.

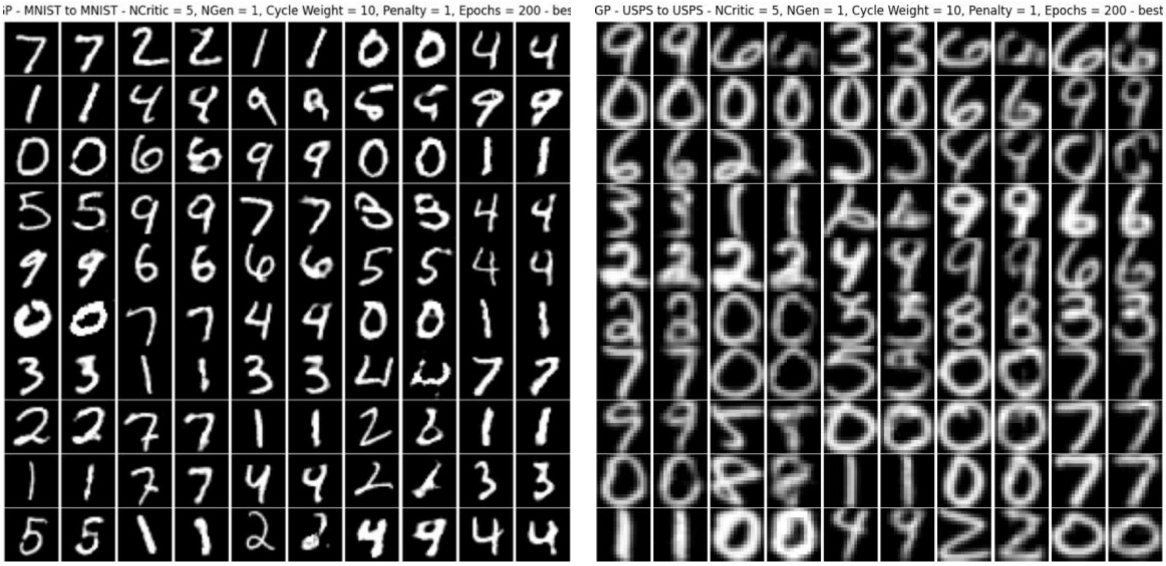

iP - MNIST to MNIST - NCritic = 5, NGen = 1, Cycle Weight = 10, Penalty = 1, Epochs = 200 - bes
GP - USPS to USPS - NCritic = 5, NGen = 1, Cycle Weight = 10, Penalty = 1, Epochs = 200 - best

Figure 1: Identity Checks i.e., MNIST to (USPS to) MNIST and USPS to (MNIST to) USPS

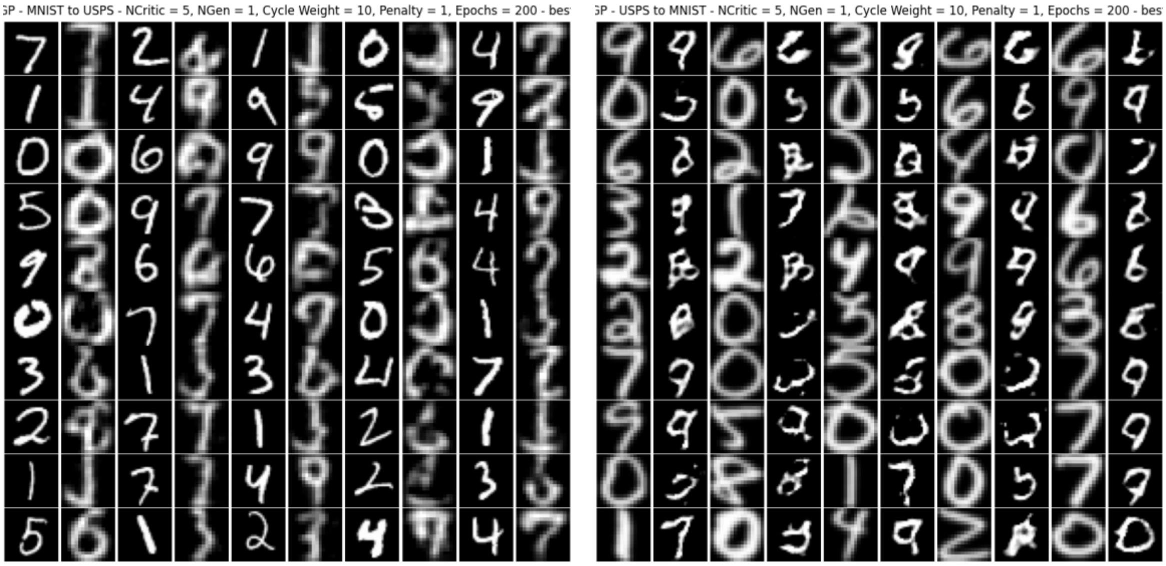

5P - MNIST to USPS - NCritic = 5, NGen = 1, Cycle Weight = 10, Penalty = 1, Epochs = 200 - bes
5P - USPS to MNIST - NCritic = 5, NGen = 1, Cycle Weight = 10, Penalty = 1, Epochs = 200 - bes

Figure 2: Translation between Domains

This part is a bit better compared to ADDA, as we can visually see what is happening between the two domains. The success of this approach depends completely on CycleGAN’s results.

The results of domain adaptation are in Table 1. Very clearly domain adaptation has not helped. **Intuition:** Let’s say that the CycleGAN learns to map MNIST “3” to USPS “7” and USPS “7” to MNIST “3”. The cycle loss would be low but the target accuracies will be worse. This is exactly what is happening as evident from the translation results in Figure 2. The model needs to know about posteriors (at least fake labels), because domain adaptation cannot happen efficiently only on marginals. Additionally, we also have very less samples in USPS compared to MNIST whose effect is visible in translation from USPS to MNIST in figure 2.

| **Source** | **Target** | **Source Test Accuracy** | **Target Test Accuracy** |
|---|---|---|---|
| MNIST | USPS | 98.65% | 25.72% |
| USPS | MNIST | 94.74% | 31.43% |

Table 1: Domain Adaptation using CycleGANs

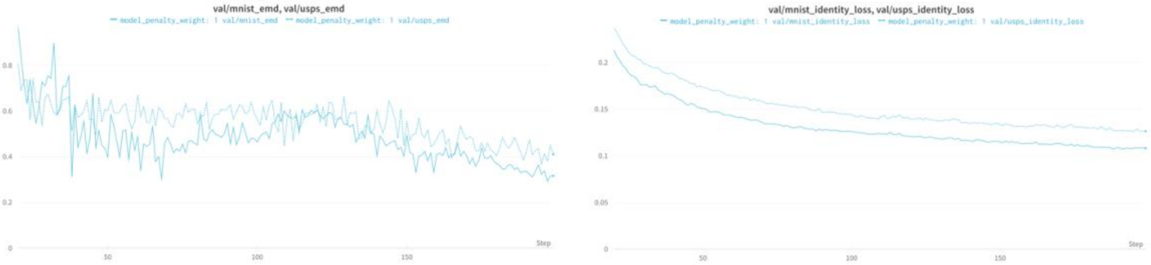


Figure 3: Relevant validation metrics

**4) Problems faced, solutions used:** We tried a bunch of hyperparameter combinations with weight clipping, none of them gave good results. So we shifted to gradient penalty approach.

## Conclusion & Future Work:

This problem showed how to make Wasserstein GANs work i.e., by using gradient penalty. Future work is to integrate the idea of "fake" target labels into this method. We propose that this can be better for doing domain adaptation compared to ADDA as we have a way to visualise what is going on in the model.