# Domain Adaptation from MNIST to SVHN

Maxime Ledieu, Pierre Billaud

April 15, 2024

**Abstract**

This work uses the Mean Teacher algorithm for semi-supervised domain adaptation, focusing on transferring knowledge from the labeled MNIST dataset to the unlabeled SVHN dataset. Through the use of two convolutional neural network models, a student and a teacher, the method focuses on labeled data from MNIST and unlabeled data from SVHN to bridge the domain gap. Preprocessing steps such as resizing, color conversion, and data augmentation align the MNIST images closer to SVHN's ones and vice-versa. Then, training steps involves calculating supervised and unsupervised losses to refine the student's predictions on MNIST and align them with the teacher's on SVHN. The algorithm seems quite efficient in domain adaptation, achieving interesting performances on the SVHN dataset.

# Contents

# 1 Instructions

Train a Deep Neural Network (you can choose which one) using MNIST and then apply it on SVHN data using domain adaptation technique.

In pairs, you are tasked with developing a solution to address the challenge of domain adaptation between two distinct image datasets: MNIST and SVHN. Specifically, you will focus on adapting a deep neural network (DNN) trained on the labeled MNIST dataset to perform accurately on the SVHN dataset, for which you are provided with images but no labels during training. Your ultimate goal is to demonstrate effective domain adaptation by achieving high accuracy on the SVHN test set.

This project is carried out in the context of the Artificial Intelligence Masters of TelecomParis.

# 2 Introduction

In this paper, we will explore the application of the Mean Teacher algorithm [TV18] for semi-supervised domain adaptation, focusing on the method first, then we will detail the challenges we faced to find a pertinent algorithm to handle domain adaptation from MNIST to SVHN. After, we will show the results obtained and how we improved the accuracy on the SVHN dataset, leading to an open discussion on what we could improve to get better results.

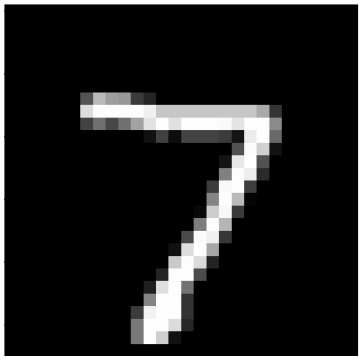

Figure 1: MNIST

Figure 2: SVHN

You can test our algorithm in here or you can go on our GitHub page (made public just after deadline) from here to download the project.

# 3 Attempts

## 3.1 CNN

In order to address the challenges of domain adaptation between the MNIST and SVHN datasets, we initially explored the potential of a Convolutional Neural Network (CNN) as our foundational model. Here is the architecture of the CNN:

$$
\begin{aligned}
\text{Conv2d} &: \text{in\_channels} = 3, \text{out\_channels} = 32, \text{kernel\_size} = 5, \text{stride} = 1, \text{padding} = 2 \\
\text{Activation} &: \text{ReLU} \\
\text{MaxPool2d} &: \text{kernel\_size} = 2, \text{stride} = 2 \\
\text{Conv2d} &: \text{in\_channels} = 32, \text{out\_channels} = 64, \text{kernel\_size} = 5, \text{stride} = 1, \text{padding} = 2 \\
\text{Activation} &: \text{ReLU} \\
\text{MaxPool2d} &: \text{kernel\_size} = 2, \text{stride} = 2 \\
\text{Flatten} &: - \\
\text{Linear} &: \text{in\_features} = 4096, \text{out\_features} = 1024 \\
\text{Activation} &: \text{ReLU} \\
\text{Linear} &: \text{in\_features} = 1024, \text{out\_features} = 10
\end{aligned}
$$

We wanted to see how a simple CNN model would react while trained on MNIST data only on SVHN test data. After several tests and getting only low accuracy results with this approach, we finally decided to go on the solution of the Generative Adversarial Networks that were supposed to increase performances.

### 3.2 GAN

We tried Generative Adversarial Networks (GANs) to investigate on their efficiency about domain adaptation from MNIST to SVHN. The GAN framework, consisting of a Generator and a Discriminator working in opposition, was seen as a foundation model for more interesting algorithms linked with Domain Adaptation like SBADAGAN [RCTC18] or CycleGAN [ZPIE17]. The GAN, quite simple, was composed by the next architecture:

Generator:

$$
\begin{aligned}
\text{Input} &\to \text{Conv2d}(3, 64, 4 \times 4, \text{stride} = 2, \text{padding} = 1) \\
&\to \text{LeakyReLU}(0.2) \\
&\to \text{Conv2d}(64, 128, 4 \times 4, \text{stride} = 2, \text{padding} = 1) \\
&\to \text{BatchNorm2d}(128) \to \text{LeakyReLU}(0.2) \\
&\to \text{ConvTranspose2d}(128, 64, 4 \times 4, \text{stride} = 2, \text{padding} = 1) \\
&\to \text{BatchNorm2d}(64) \to \text{ReLU}() \\
&\to \text{ConvTranspose2d}(64, 3, 4 \times 4, \text{stride} = 2, \text{padding} = 1) \\
&\to \text{Tanh}()
\end{aligned}
$$

Discriminator:

$$
\begin{aligned}
\text{Input} &\to \text{Conv2d}(3, 64, 4 \times 4, \text{stride} = 2, \text{padding} = 1, \text{bias} = False) \\
&\to \text{LeakyReLU}(0.2) \\
&\to \text{Conv2d}(64, 128, 4 \times 4, \text{stride} = 2, \text{padding} = 1, \text{bias} = False) \\
&\to \text{BatchNorm2d}(128) \to \text{LeakyReLU}(0.2) \\
&\to \text{Conv2d}(128, 256, 4 \times 4, \text{stride} = 2, \text{padding} = 1, \text{bias} = False) \\
&\to \text{BatchNorm2d}(256) \to \text{LeakyReLU}(0.2) \\
&\to \text{Conv2d}(256, 1, 4 \times 4, \text{stride} = 1, \text{padding} = 0, \text{bias} = False) \\
&\to \text{Sigmoid}()
\end{aligned}
$$

The idea with GAN was to create an image generator aiming to bridge the gap between the source and target domains. There was a transformation process of images served as a preparatory step

before classifying the images with a simple CNN, using the GAN-generated images into our domain adaptation approach. The architecture of the classifier is defined as follows:

$$
\begin{aligned}
&\text{Convolutional Layers:} \\
&\qquad \text{Input} \rightarrow \text{Conv2d}(3, 32, 5 \times 5, \text{stride} = 1, \text{padding} = 2) \rightarrow \text{ReLU} \\
&\qquad \rightarrow \text{MaxPool2d}(2 \times 2, \text{stride} = 2) \\
&\qquad \rightarrow \text{Conv2d}(32, 64, 5 \times 5, \text{stride} = 1, \text{padding} = 2) \rightarrow \text{ReLU} \\
&\qquad \rightarrow \text{MaxPool2d}(2 \times 2, \text{stride} = 2) \\
&\text{Fully Connected Layers:} \\
&\qquad \text{Flatten} \rightarrow \text{Linear}(4096, 1024) \rightarrow \text{ReLU} \\
&\qquad \rightarrow \text{Dropout}(0.5) \\
&\qquad \rightarrow \text{Linear}(1024, 10)
\end{aligned}
$$

The results were better than simple CNN but did not satisfied us. That's why we decided to investigate deeper the State-of-the-Art of Domain Adaptation using DNN. After a lot of researches, we found interesting papers on Self-Ensembling [FMF18] using Mean Teacher [TV18] algorithm for Domain Adaptation. These methods seems adapted for MNIST to SVHN domain adaptation with good results, outperforming previous SOTA methods using appropriate preprocessing and data augmentation techniques.

# 4 Method

We used the Mean teacher algorithm which is a semi-supervised technique dedicated here to make domain adaptation from MNIST to SVHN. The key idea behind this algorithm is to train model on labeled data from MNIST and unlabeled data from SVHN. For that, it creates a student model and a teacher model which are CNN classifiers. Then it influences the consistency loss between the predictions of a student model and a teacher model to improve the quality of the learned representations.

## 4.1 Imports and Preprocessing

Firstly, we import and apply a preprocessing on MNIST and SVHN Datasets by augmenting them directly with premade functions from torchvision.

- Resizing: We increase the size of MNIST images to 32x32 pixels to match the SVHN image dimensions. This is crucial for ensuring that the model can be trained interchangeably on both datasets without needing to adjust the input size.
- Grayscale to RGB Conversion: We convert the single-channel grayscale MNIST images to three channels to match the RGB SVHN images. This is done by duplicating the grayscale values across all three channels.
- Augmentation: The MNIST dataset undergoes several augmentations:
    1. RandomHorizontalFlip and RandomRotation: These augmentations introduce spatial variability, making the model more robust to position and orientation changes.
    2. ColorJitter: Even if MNIST images are grayscale, adding color jittering simulates slight variations in lighting and color that could make the model more adaptable to the color SVHN images.
    3. Normalization: Normalizing the images with mean and standard deviation values of 0.5 across all channels standardizes the data, aiding in model training convergence.
- SVHN Dataset Preprocessing

1. Normalization: Similar to the MNIST dataset, SVHN images are normalized with mean and standard deviation values of 0.5. This ensures that pixel values are within a similar range for both datasets, which is important for model training.

Figure 3: MNIST preparation for SVHN

## 4.2 Data Source and Batch Preparation and Data Augmentation

Once properly imported, formatted and augmented, we prepare our data for further augmentations with custom functions as detailed below.

- Supervised and Target Datasets: We create ArrayDataSource objects for both supervised (source MNIST) and target (unlabeled SVHN) datasets.
- We apply here again data augmentation on our two datasets (train MNIST with labels and train SVHN without labels) following Self-Ensemble paper for Domain Adaptation [FMF18]:
    1. Translation: We introduce random translations to the images based on a translation range. This can help the model become invariant to the position of the digits in the images, which is useful given the different contexts in which digits appear in the MNIST and SVHN datasets.
    2. Intensity Scaling: We randomly adjust the intensity scale of the images within specified bounds (intensity scale lower, intensity scale upper). This method can make the model more robust to variations in lighting and contrast. It can help to adapt from MNIST's simple background to the complex backgrounds in SVHN.
    3. Intensity Offset: We add a random offset to the intensity values (intensity offset lower, intensity offset upper) which can also further help the model to handle different lighting conditions and backgrounds.
    4. Noise Addition: We introduce a Gaussian noise with a standard deviation of noise std dev. It can help the model to become more robust to small perturbations, mimicking real-world imperfections in images.
    5. Affine Transformation: We use cv2.warpAffine to apply the translations and centered transformations to augment the spatial characteristics of the images on every color channel and we apply the transformation individually; this ensures the spatial consistency is maintained across the channels.

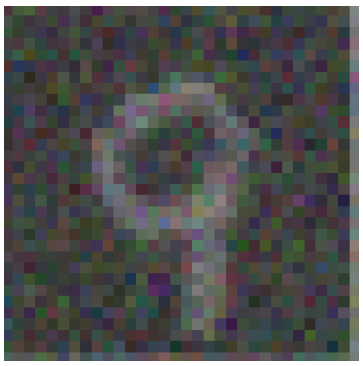

Figure 4: Number 9 after augmentation

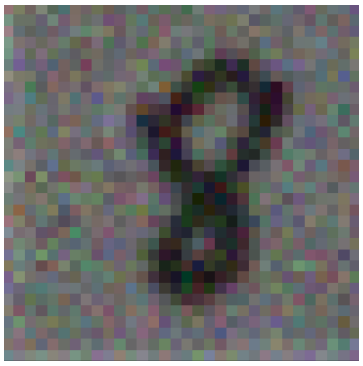

Figure 5: Number 8 after augmentation

### 4.3 Models preparation

We create two convolutional classifiers: a student model and a teacher model. Initially, these two models have the same architecture and parameters, except Student having Adam optimizer and Teacher having Weighted Average Optimizer. Here is the 3 blocks architecture of these two models:

$$
\begin{aligned}
\text{Input} &\to \text{Conv2d}(3, 128, 3 \times 3, \text{padding} = 1) \to \text{BatchNorm2d}(128) \to \text{ReLU} \\
&\to \text{Conv2d}(128, 128, 3 \times 3, \text{padding} = 1) \to \text{BatchNorm2d}(128) \to \text{ReLU} \\
&\to \text{Conv2d}(128, 128, 3 \times 3, \text{padding} = 1) \to \text{BatchNorm2d}(128) \to \text{ReLU} \\
&\to \text{MaxPool2d}(2 \times 2) \to \text{Dropout} \\
&\to \dots \\
&\to \text{Conv2d}(128, 256, 3 \times 3, \text{padding} = 1) \to \text{BatchNorm2d}(256) \to \text{ReLU} \\
&\to \text{Conv2d}(256, 256, 3 \times 3, \text{padding} = 1) \to \text{BatchNorm2d}(256) \to \text{ReLU} \\
&\to \text{Conv2d}(256, 256, 3 \times 3, \text{padding} = 1) \to \text{BatchNorm2d}(256) \to \text{ReLU} \\
&\to \text{MaxPool2d}(2 \times 2) \to \text{Dropout} \\
&\to \dots \\
&\to \text{Conv2d}(256, 512, 3 \times 3) \to \text{BatchNorm2d}(512) \to \text{ReLU} \\
&\to \text{Conv2d}(512, 256, 1 \times 1, \text{padding} = 1) \to \text{BatchNorm2d}(256) \to \text{ReLU} \\
&\to \text{Conv2d}(256, 128, 1 \times 1, \text{padding} = 1) \to \text{BatchNorm2d}(128) \to \text{ReLU} \\
&\to \dots \\
&\to \text{AvgPool2d}(6 \times 6) \to \text{Flatten} \\
&\to \text{Linear}(128, 10)
\end{aligned}
$$

### 4.4 Training Loop

- Model Preparation: We set the student and teacher networks to training mode with .train(). This is important for layers like dropout and batch normalization that have distinct behaviors during training and evaluation.
- Forward Pass
  1. Source Domain: We pass the source data through the student network to obtain logits, which are used with the ground truth labels to compute the classification loss (clf loss).
  2. Target Domain: We pass independently the target data through both the student and teacher networks. The softmax probabilities from these logits are then used to compute the unsupervised loss, focusing on aligning the student's predictions with the teacher's on the target domain.
- Loss Calculation
  1. Supervised Loss: We use self.classification criterion (typically cross-entropy) to calculate the loss between the student's predictions on the source domain and the true labels.
  2. Unsupervised Loss: We calculate it by compute augmentation loss function, which measures the difference between the student's and teacher's predictions on the target domain, adjusted by a confidence threshold and class balance considerations.
- Backpropagation and Optimization
  1. We perform backpropagation (loss expression.backward()) on the combined supervised and weighted unsupervised loss.
  2. We update the student model using its Weighted Average optimizer.
  3. The teacher model's parameters are updated with the custom teacher optimizer.update parameters() method which uses the exponential moving average (EMA) update rule:

$$\theta'_t = \alpha\theta'_{t-1} + (1 - \alpha)\theta_t$$

- Loss Reporting: We use supervised and unsupervised losses for monitoring only.
- Epochs and Tracking: We run the training for X epochs, and we try to track various metrics including train loss, unsupervised (target) loss, confidence rate, and mask rate. We try to identify the best performing epoch based on the confidence rate and save the state of the teacher network accordingly.

### 4.5 Evaluation and Model Selection

- Best State Selection: The selection of the best model state is based on the confidence rate (how confident the model makes predictions on unlabeled data). It focus on models that are more certain of their predictions on the target dataset.
- Test on SVHN dataset: Our final evaluation function calculates the global student model accuracy by accumulating the total number of correctly predicted labels and the total number of examples to calculate the overall accuracy of the model on the test dataset. We also have a Confusion Matrix which breaks down the predictions into true positives(TP), true negatives(TN), false positives(FP), and false negatives(FN). We establish a precision score which measures the model's accuracy in predicting positive labels and a recall indicating the model's ability to find all the positive samples. We use these two last metrics to calculate the F1 score to have a balanced indicator.

## 5 Results

Table 1: Comparison of accuracy between the Mean Teacher, CNN, and GAN methods

| Model | Accuracy at 10 Epochs | Accuracy at 50 Epochs |
|---|---|---|
| CNN | 20.44% | 12.21% |
| GAN | 16.12% | 33.68% |
| Mean Teacher | 78.12% | 86.21% |

Our best result was got with Mean Teacher, achieving a 89.63% of accuracy after 150 epochs for training.

## 6 Challenges and Discussion

### 6.1 Treatment Choice

In our domain adaptation process from MNIST to SVHN, we took the decision to use the RGB format of the SVHN dataset rather than converting it to greyscale. This choice was driven by the absence of explicit instructions to limit our adaptation efforts to greyscale images. We think that working with RGB images aligns with the original format of SVHN, preserving the colors of the target domain data. However, this approach also introduced additional computational complexity, because processing three-channel RGB images asks more resources and time compared to handling single-channel greyscale images.

### 6.2 Comparison of models

Our different tests with different models revealed interesting findings. A basic CNN trained for just 10 epochs better perform the same model trained for 50 epochs, likely due to the CNN's tendency to overfit on the MNIST dataset quickly which seems normal.

Among the techniques we evaluated, the Mean Teacher algorithm significantly outperforms both CNN and GAN models in adapting between these datasets. However, our decision to prioritize the Mean Teacher approach, based on its promising performances shown in the literature, pushed us to achieve substantial improvements on this model compared to CNN and GAN. We have no doubts

that if we would have focused on GAN, we would have better results (using specific techniques like StyleGAN or SBADAGAN) in adaptation accuracy than our current.

### 6.3 Important Keypoints

The domain adaptation from MNIST to SVHN was really interesting and was challenging due to the differences between the two datasets. MNIST, with grey color only and uniform background in images, is really different from the colorized and differently sized digits of SVHN. To address this challenge, specific strategies such as class balancing loss and targeted data augmentation were essential.

The introduction of data augmentations, particularly about intensity and scale changes, was important to improve accuracy. These techniques bridged the gap between the characteristics of MNIST images and those of SVHN, which showed us the importance of data augmentation in domain adaptation.

### 6.4 Comparison with State-of-the-Art

In our journey to reproduce the results of the intial paper [FMF18] using the Mean Teacher algorithm, we encountered difficulties to completely stick with SOTA performances. We believe that a couple of factors are the reasons of this difference. Firstly, the fine-tuning process for our algorithm might not have been optimized at 100%. More, we could have explored other data augmentation techniques to improve our results like different colorizing techniques, or adding more noise in background, elastic distorsion, photometric or perspective transformation etc. Lastly, we implemented the Mean Teacher algorithm using the paper and the different codebases and indications we found on the Web. Our code is not completely identical to the original paper and we may have missed details compared to the original implementation.

# References


[FMF18] Geoffrey French, Michal Mackiewicz, and Mark Fisher. Self-ensembling for visual domain adaptation, 2018.

[RCTC18] Paolo Russo, Fabio M. Carlucci, Tatiana Tommasi, and Barbara Caputo. From source to target and back: Symmetric bi-directional adaptive gan. pages 8099–8108, 2018.

[TV18] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. 2018.

[ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.