Final Project: Domain Adaptation for Digits [Final Submition]

Farit Galeev

November 2019

1 Baseline Model

As a baseline model I have decided to pick a model similar to classic CNN but with slight modification. The proposed architecture (1) consists of *feature extractor* which maps image on the $\mathcal{D}$-dimensional space $f \in \mathcal{R}^{\mathcal{D}}$, label predictor (classifies for which class source image belongs) and domain classifier (classifies for which domain given image belongs). The depicted version of architecture and the learning algorithm you can see at **Figure 1**.

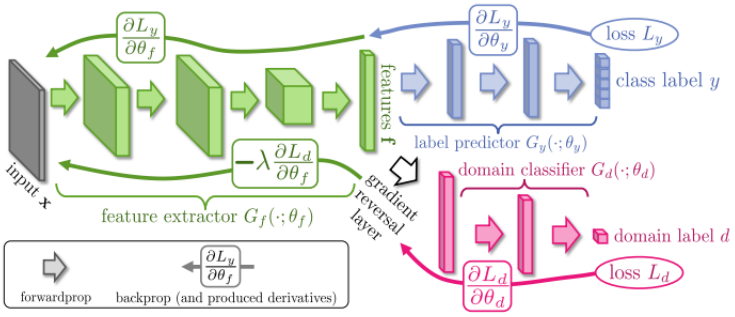

Figure 1: The proposed architecture includes a deep feature extractor (green) and a deep label predictor (blue), which together form a standard feed-forward architecture. Unsupervised domain adaptation is achieved by adding a domain classifier (red) connected to the feature extractor via a gradient reversal layer that multiplies the gradient by a certain negative constant during the back-propagation based training. Otherwise, the training proceeds in a standard way and minimizes the label prediction loss (for source examples) and the domain classification loss (for all samples). Gradient reversal ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), thus resulting in the domain-invariant features

More formally, we consider the function which we want to optimize:

$$E(\theta_f, \theta_y, \theta_d) = \sum_{\substack{i=1..N \\ d_i=0}} L_y^i(\theta_f, \theta_y) - \lambda \sum_{i=1..N} L_d^i(\theta_f, \theta_d)$$

Here $L_y(\cdot, \cdot)$ is the e loss for label prediction (e.g. multinomial), $L_d(\cdot, \cdot)$) is the loss for the domain classification (e.g. logistic) and λ is the adaptation factor which gradually changes from 0 to 1:

$$\lambda_p = \frac{2}{1+exp(-10*p)} - 1$$

Here p is the fraction between currently observed dataset with bias and the whole dataset that should be observed (i.e. $\frac{bias+curEpoch*|dataset|}{nEpochs*|dataset|}$).

2 Final model

As final model I pick a model which was described in the following paper (2). This model differs from the baseline model in that instead of using domain classifier and Gradient Reversal Layer here authors used the second classifier to obtain a feature generator that can minimize the discrepancy on target samples. Discrepancy here indicates how the two classifiers disagree on their predictions. Unsupervised Domain Adaptation is achieved by aligning source and target features by utilizing the task-specific classifiers as a discriminator in order to consider the relationship between class boundaries and target samples.

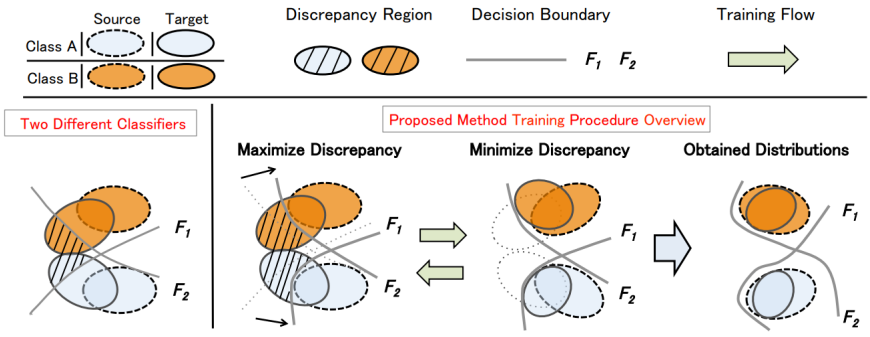

Figure 2: Example of two classifiers with an overview of the proposed method. Discrepancy refers to the disagreement between the predictions of two classifiers. First, we can see that the target samples outside the support of the source can be measured by two different classifiers (Leftmost, Two different classifiers). Second, regarding the training procedure, we solve a minimax problem in which we find two classifiers that maximize the discrepancy on the target sample, and then generate features that minimize this discrepancy.

3 Training Process

Discrepancy loss

Discrepancy loss as it was stated earlier is the difference between two classifiers' probabilistic outputs:

$$d(p_1, p_2) = \frac{1}{K} \sum_{k=1}^{K} |p_{1k} - p_{2k}|$$

where the p_{1k} and p_{2k} denote probability output of p_1 and p_2 for class k respectively.

Training steps

We need to train two classifiers which take inputs from the generator and maximize $d(p_1(y|x_t), p_2(y|x_t))$, and the generator which tries to mimic the classifier. So it is achieved in three steps:

Step A First, we train both classifiers and generator to classify the source samples correctly. In order to make classifiers and generator obtain task-specific discriminative features, this step is crucial. We train the networks to minimize softmax cross entropy.

$$\min_{G,F_1,F_2} \mathcal{L}(X_s, Y_s)$$
$$\mathcal{L}(X_s, Y_s) = -E_{(x_s,y_s)\sim(X_s,Y_s)} \sum_{k=1}^{K} \mathbb{1}_{k=y_s} log(p(y|x_s))$$

Step B In this step, we train the classifiers (F_1, F_2) as a discriminator for a fixed generator (G). By training the classifiers to increase the discrepancy, they can detect the target samples excluded by the support of the source.

$$\min_{F_1,F_2} \mathcal{L}(X_s, Y_s) - \mathcal{L}_{adv}(X_t)$$
$$\mathcal{L}_{adv}(X_t) = E_{x_t\sim X_t}[d(p_1(y|x_t), p_2(y|x_t))]$$

Step C We train the generator to minimize the discrepancy for fixed classifiers.

$$\min_{G} \mathcal{L}_{adv}(X_t)$$

4 Implementation

For implementation of **GRL** architecture I have used materials from paper (1), its supplementary materials (3) and some code from (4). For implementation of **MCD** architecture I have used materials from paper (2) and some code from repository (5)

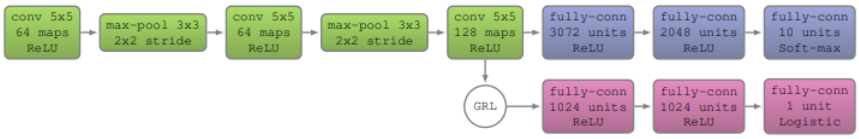

Figure 3: SVHN - MNIST Model with GRL Architecture

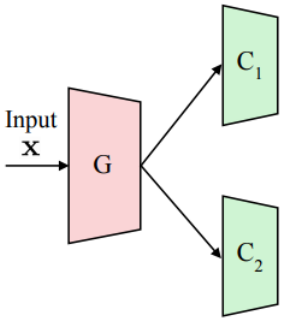

Figure 4: MCD Model Architecture consist of three parts. Generator - CNN with Dense layer and two Classifiers (C1 and C2)

5 Results

GRL models were trained using Adam optimizer with initial learning rate equals to 1e-3 with Cosine Annealing Warm Restart scheduler. MCD model was trained using Adam optimizer with constant learning rate equal to the 2e-4 and weight decay equal to 5e-4 as was stated in the paper(2)

	SVHN-train score	SVHN-test score	MNIST-test
Source only	**99.68%**	**91.55**%	69.34%
Grayscale SVHN NN-GRL	85.67%	83.39%	**70.87%**
SVHN NN-GRL	93.20%	86.62%	70.23%
MCD	89%	88%	**96%**

Table 1: Results

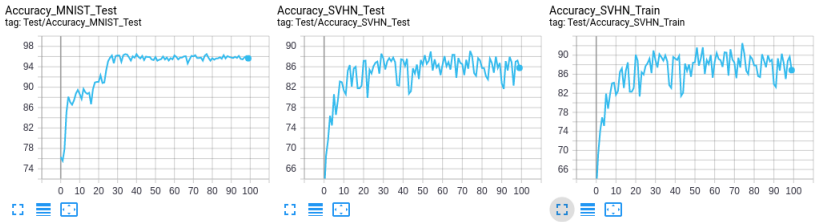

Figure 5: Accuracy changing during training process

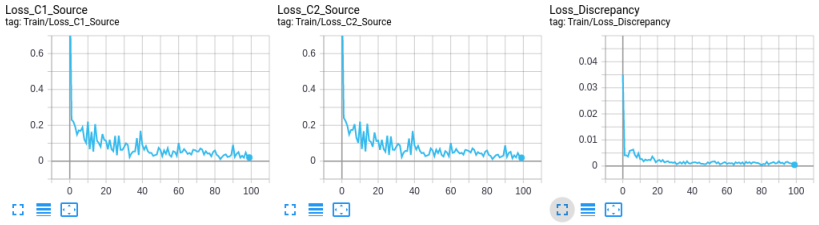

Figure 6: Loss changing during training

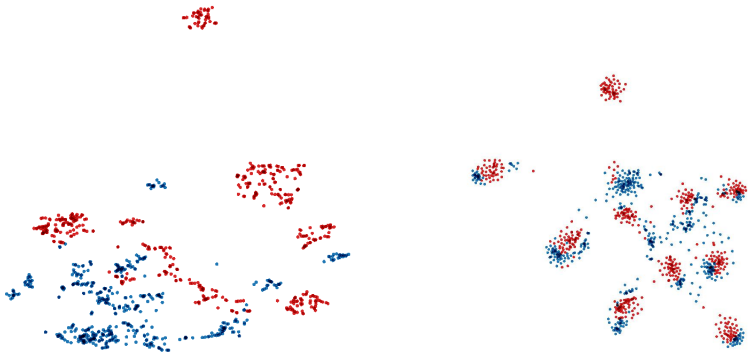

Figure 7: t-SNE visualization of features extracted by Generator of MCD model trained only on SVHN before Domain Adaptation (left),and after Domain Adaptation (right). Doing t-SNE visualization on feature extractor which is not pretrained at all will output nothing, but noise, so it doesn't make sense

6 Additional Study

During additional study I have found that MCD model is very sensitive to the position of Batch Normalization before or after activation function. So there were a lot of discussions about position of Batch Normalization and its limitation of covariate shift. Also it is very sensitive to learning rate. I've tried to use Cosine Annealing Warm Restart learning rate scheduler, so its affection was negative to the accuracy. So that they were early stopped.
In that case the important hyperparameter was a learning rate, so that for tuning I tried to use learning rate schedulers

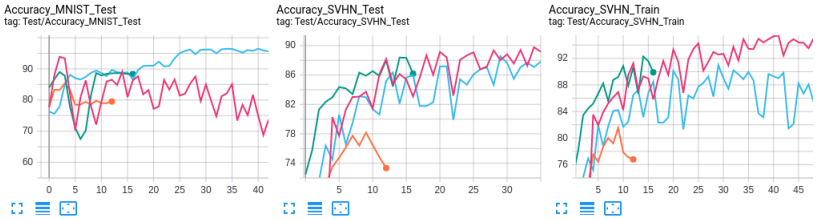

Figure 8: Accuracy of the model during training (blue) with the parameters stated in the paper (2) in comparison to losses of additional study (mix of Batch Normalization position and learning rate scheduling)

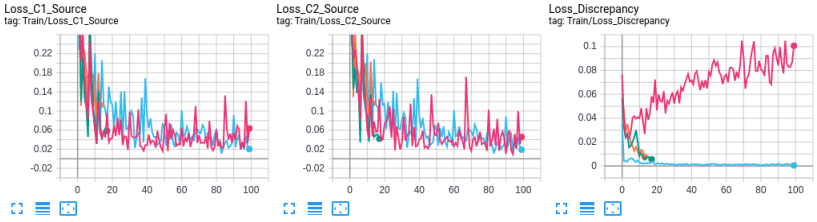

Figure 9: Loss of the model during training (blue) with the parameters stated in the paper (2) in comparison to losses of additional study (mix of Batch Normalization position and learning rate scheduling

7 Personal Thoughts

From t-SNE visualization we can notice that algorithm is able to solve domain gap, but not much perfectly and my personal thoughts that it is because of discrepancy loss, so that as improvement of model we can change discrepancy loss which is presented here as L1 loss between two probability measures of C1 and C2. One kind of this improvement was proposed by Apple Inc. on CVPR 2019 conference (6) as adding Wasserstein distance.

References

[1] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1180–1189, JMLR.org, 2015.

[2] K. Saito, K. Watanabe, Y. Ushiku, and T. Harada, "Maximum classifier discrepancy for unsupervised domain adaptation," *arXiv preprint arXiv:1712.02560*, 2017.

[3] Y. Ganin and V. Lempitsky, "Suplementary material. unsupervised domain adaptation by backpropagation.," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, 2015.

[4] "Pytorch dann." `https://github.com/wogong/pytorch-dann`.

[5] "Mcdda." `https://github.com/mil-tokyo/MCD_DA`.

[6] L. Chen-Yu, T. Batra, M. Baig, and D. Ulbricht, "Sliced wasserstein discrepancy for unsupervised domain adaptation," *arXiv:1903.04064*, 2019.