Domain Adversarial Training

Anirudh Mandahr
CS664 Artificial Intelligence
Boston University
anirudh1@bu.edu

1 Introduction

The creation of convolution neural networks has been a breakthrough for the use of neural networks in vision problems. Despite its effectiveness, CNN in its vanilla form is not entirely ready for critical applications due to its shortcomings. One such problem with CNNs is that they learn domain features which make them less versatile on novel datasets and real-world samples. To combat this, research has been conducted on reducing dataset bias by achieving domain invariance. There have been a lot of techniques aimed at achieving domain adaptation. The interest had until recently been directed at linear hypotheses[3][4]. Only the newer researches have focused towards non-linear, neural-based methods[5][6]. Our work has mainly focussed on research by Zhongyi Pei et al[2]. In the 2015 paper by Yaroslav Ganin et al,[1] the authors attempted domain transfer such that the predictions are made on the basis of features that cannot be used to discriminate between source and target domain. In the more recent paper by Zhongyi Pei et al[2], the authors use the same principle but attempt to capture multi-mode patterns to allow finer matching of different data distributions based on multiple domain discriminators. The key element in both these approaches is a gradient reversal layer, which propagates the negative loss of the domain classifier to the input, to unlearn the domain features.

2 Related work

2.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a type of multi-layer neural network that assumes input data to be represented as tensors with the shape: $(m) \times (x_{height}) \times (x_{width}) \times (n_c)$, where m is the number of input images per batch, x is the input image, and n_c is the number of channels. CNN drastically reduces the number of parameters that need to be tuned. Therefore, CNN efficiently handles the high dimensionality of raw images. Highly complex pattern recognition can be achieved by using a network of neurons. The 3 key layers of a CNN are Convolution, Subsampling, Activation. The convolutional layer has filters/kernels that have width and height as hyper-parameters. After the convolution operation is performed on the input of the convolution block for each layer $[l]$, the output is a feature map that has shape: $(m) \times (f^l_{height}) \times (f^l_{width}) \times (n_c^{l-1}) \times (n_c^l)$, where f are the filters, n_c^{l-1} is the number of channel of the input, n_c^l is the number of filters. The dimensionality of the data is reduced by the pooling layers. This process is known as subsampling or pooling. Often, an activation called ReLU (f(x)=max(0,x)) is applied to the output of the pooling layers, generally to remove negative values from the activation map. After multiple layers of convolutional blocks, it is common to add a fully connected layers at the end of the architecture to map the feature space to the number of classes to predict.

In this paper, We use pre-trained ResNet trained on the ImageNet dataset[11], but retraining all the layers using a process called transfer learning.

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

2.2 Domain Adversarial Learning

While it is easy for a machine learning model to classify test data belonging to the same domain as the training dataset, it might not be able to generalize well in real-world scenarios. Domain adaption is a branch of deep learning that tries to tackle this question. In domain adaption, to achieve domain transfer, predictions must be based on features that cannot distinguish between the training (source) and testing (target) domains. Adversarial training enable Neural Networks to train on labeled data from the source domain and unlabeled data from the target domain. Traditionally in domain adversarial training, as training progresses, the approach favors the generation of features that are discriminating for the core learning task in the source domain and non-discriminating with respect to the shift between domains [1].

3 Materials and Methods

3.1 Deep Convolutional Neural Network

In this study, We implemented a Deep Convolutional Neural Network that used transfer learning to re-train ResNet[10] which was pretrained on ImageNet dataset [11]. This model is used to benchmark the results of transfer learning performance, when a CNN is trained on the source dataset and tested on target dataset. Due to the nature of the training, the model will not learn target representation. Using this baseline, We will be able to see how well domain adversarial models performs.

3.2 Domain Adaptation Neural Network

We implemented a domain adaptation model similar to [1] by building a domain classifier that helps the feature extractor G_f to generate domain invariant features. The model has a fully connected label classifier G_y like a CNN but also a fully connected domain classifier G_d, with a gradient reversal layer (GRL). The gradient reversal layer aims to back-propagate the negative gradient of the loss of the domain classifier L_d to confuse the feature extractor G_f (and therefore remove the information that helps G_d to understand the domain), see Figure 3 in the appendix. We denote the source dataset as $D_s = \{(x_i^s, y_i^s)\}_{i=1}^n$ of n labeled examples and the target domain dataset as $D_t = \{(x_j^t)\}_{j=1}^{n'}$. The label classifier $G_y(G_f(x_i;\theta_f);\theta_y)$ is trained on the source dataset. The domain classifier $G_d(G_f(x_i;\theta_f);\theta_d)$ is trained on the source and target dataset to increase the domain prediction loss. $L_y^i(\theta_f,\theta_y)$ is the label predictor loss and $L_d^i(\theta_f,\theta_d)$ is the domain predictor loss. The gradient reversal layer is denoted by R. The domain loss term L_d is the sum of source and target domain loss. The optimization problem is defined below.

$$E(\theta_f,\theta_y,\theta_d) = \tfrac{1}{n}\textstyle\sum_{i=1}^n L_y^i(\theta_f,\theta_y) - \lambda(\tfrac{1}{n}\sum_{i=1}^n L_d^i(\theta_f,\theta_d) + \tfrac{1}{n'}\sum_{i=n+1}^N L_d^i(\theta_f,\theta_y))$$

Where:

$$R(x) = x,$$

and

$$\frac{dR}{dx} = -I.$$

We define E after applying Stochastic gradient descent to E as:

$$E'(\theta_f,\theta_y,\theta_d) = \frac{1}{n}\sum_{i=1}^n L_y^i(\theta_f,\theta_y) - \lambda(\frac{1}{n}\sum_{i=1}^n L_d(G_d(R(G_f(x_i;\theta_f));\theta_d), d_i + \frac{1}{n'}\sum_{i=n+1}^N L_d(G_d(R(G_f(x_i;\theta_f));\theta_d), d_i))$$

The goal is to learn θ_f for the CNN, so that the feature extractor f is domain invariant and therefore the domain classifier $G_d(G_f(x_i;\theta_f);\theta_d)$ is not able to distinguish between the domains anymore(i.e. maximizing the cost).

This is done by finding the saddle point $\hat{\theta_f}, \hat{\theta_y}, \hat{\theta_d}$ such that:

$$(\hat{\theta_f}, \hat{\theta_y}) = arg\min_{\theta_f,\theta_y} E(\theta_f,\theta_y,\theta_d), \quad \hat{\theta_d} = arg\max_{\theta_d} E(\theta_f,\theta_y,\theta_d). \tag{1}$$

The saddle point can be found by using the following equations:

$$\theta_f \leftarrow \theta_f - \mu(\frac{\partial L_y^i}{\partial \theta_f} - \lambda\frac{\partial L_d^i}{\partial \theta_f}) \tag{2}$$

$$\theta_y \leftarrow \theta_y - \mu \frac{L_y^i}{\partial \theta_y}, \tag{3}$$

$$\theta_d \leftarrow \theta_d - \mu\lambda \frac{\partial L_d^i}{\partial \theta_d}, \tag{4}$$

The learning rate and lambda are respectively:

$$\mu_p = \frac{\mu_0}{(1+\alpha \cdot p)^\beta}, \quad \lambda_p = \frac{2}{1+\exp(-\gamma \cdot p)} - 1 \tag{5}$$

where α, β, γ are hyperparameters and p is the training progress linearly changing from 0 to 1. The first three equations(2, 3, 4) above are closely related to SGD gradients for a feed forward network with a feature extractor, a label predictor and a domain classifier, except that the gradients are not added but subtracted. This change is made so that the SGD increases domain classification loss and hence reduces the test target domain classification accuracy.

3.3 Multi-Adversarial Domain Adaptation

Just like in DANN[1], Multi-Adversarial Domain Adaptation (MADA)[2] uses an unsupervised domain adaptation approach where we have a source domain $D_s = \{(x_i^s, y_i^s)\}_{i=1}^{ns}$ of n_s labeled examples and a target domain $D_t = \{(x_j^t)\}_{j=1}^{nt}$ of n_t unlabeled examples. We again denote the feature extractor as G_f, the label predictor as G_y. To match the source and target domains on the multimode structures behind the data distributions, the authors decided to use several domain classifiers G_d^k, where k goes from 1 to the number of classes in to predict (K). The goal of these K classifiers is to capture a class-wise domain understanding of each distribution. In many cases, datasets include classes that might be extremely different in terms of background information (for instance a bike versus a pen). The purpose of these G_d^k is to model these distribution shift for each classes within a given domain. However, it is not straightforward to assign a given domain classifier to a given label, therefore, the authors used a trick for force this assignment. Before feeding the feature to each domain classifier, we multiply the feature with the predicted label ($G_d^k(\hat{y}_i^k G_f(x_i))$. The MADA paper [2] propose several architecture for these domain discriminator (MADA, MADA-full, MADA-partial), but in our study, we focused on MADA that does not share weights between domain classifiers, so each of the G_d^k has its own loss L_d^k, see Figure 4 in the appendix. We define the two important losses as followed:

- Label predictor Loss:

$L_y(\theta_f, \theta_y) = \frac{1}{ns} \sum_{x_i \in D_s} L_y(G_y(G_f(x_i; \theta_f); \theta_y))$

- Domain classifiers Loss:

$L_d(\theta_f, \theta_d^k \mid k \in [1,K]) = \frac{1}{n} \sum_{i=1}^{K} \sum_{x_i \in D_s \cup D_t} L_d^k(G_d^k(R(\hat{y}_i^k G_f(x_i; \theta_f)), d_i; \theta_d^k \mid k \in [1,K]))$
where $R(x) = x$ and $\frac{dR}{dx} = -I$ (Gradient Reversal Layer)

Therefore, the overall cost is: $E(\theta_f, \theta_y, \theta_d) = L_y(\theta_f, \theta_y) - \frac{\lambda}{n} L_d(\theta_f, \theta_d^k \mid k \in [1,K])$

And the optimization task looks as follow:

$$(\hat{\theta_f}, \hat{\theta_y}) = arg \min_{\theta_f, \theta_y} \quad E(\theta_f, \theta_y, \theta_d^k \mid k \in [1,K]) \tag{6}$$

$$(\hat{\theta_d^k} \mid k \in [1,K]) = arg \max_{\theta_d^k \mid k \in [1,K]} \quad E(\theta_f, \theta_y, \theta_d^k \mid k \in [1,K]). \tag{7}$$

4 Experiments

We evaluated our own implement of domain adversarial neural network (DANN)[1] and multi-adversarial domain adaptation (MADA)[2] models and compared them with the original papers. The codes, datasets, and configurations are available at `https://github.com/Axhk97m/MADA-PL`.

4.1 Setup

To train our models, we used PyTorch-Lightning[13] a lightweight PyTorch wrapper for high-performance AI research. Also for simplicity, we made our code executable inside docker containers to avoid package installations.

4.2 Dataset Setup

Office-31 (Saenko et al 2010)[15] is a well-known dataset that has been created to assess domain adaptation algorithms for image classification using deep learning methods. This dataset is divided into three domains: Amazon (A), DSLR (D), and Webcam (W) which denote the source of the images of 31 categories of objects found in an office such as keyboards, computers, or mugs. Our version of the Office-31 dataset is quite small with a total of 4110 images. We downloaded it using this python library `https://pypi.org/project/office31`, see Figure 1. A, W, and D contain respectively:

- 2817 packshot RGB images of size (300x300) (average of 90 images per class).
- 795 RGB images of size (423 x 423) from web camera (average of 25 images per class).
- 498 RGB images of size (1000 x1000) from SLR camera (average of 16 images per class).

We evaluated our models for the following tasks A $\rightarrow$ W, A $\rightarrow$ D, which were also evaluated in the original papers DANN[1] and MADA[2].

MNIST (LeCun et al., 1998)[16] (Modified National Institute of Standards and Technology database) is a vast collection of handwritten digits. It has a training set of 60,000 examples and a test set of 10,000 examples. It has been extracted from NIST Special Database 1 and 3 which contain monochrome images of handwritten digits. The original NIST (20x20) grayscale images were centered in a 28x28 image by calculating the center of mass of the pixels. **MNIST-M** is a version of the MNIST dataset where patches of color randomly extracted from BSDS500 (Arbelaez et al., 2011)[17] have been added to the original dataset, see Figure 1. The number of examples of MNIST-M is exactly the same as the MNIST dataset except that images have 3 channels. We downloaded both MNIST and MNIST-M using the TorchVision library.

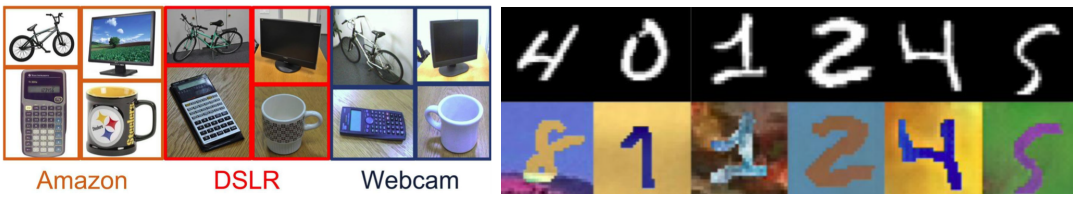

Figure 1: Dataset example of Office-31 (left) and MNIST-MNISTM (right)

We compared our implementations of DCNN, DANN, and MADA with the original papers [1][2] for the following tasks MNIST$\rightarrow$ MNIST-M, Amazon $\rightarrow$ Webcam, and Amazon $\rightarrow$ DSLR. For DANN and MADA, we follow standard evaluation protocols for unsupervised domain adaptation [1][2] by feeding test labeled source examples to both the label predictor and domain classifier(s) and test unlabeled examples only to the domain classifier(s) since we assume that we do not have labels for the target domain.

For MNIST and MNIST-M, datasets are already splited by TorchVision with an approximate ratio of 70%Train (60,000 images), 15% Validation (10,000 images), and 15% Test (10,000 images). For OFFICE-31, we splited all 3 domains with a ratio of 80% Train, 10% Validation, and 10% Test. In our study, training sets are shuffled during training but the loading remains fixed, validation sets and test sets are not shuffled. All experiments in this study used the same dataset split to correctly evaluate and compare models. Even random computations are reproducible using a seed.

5 Results

A very important part of our study relies on the Weights and Biases tool [11] which allows us to track our training metrics in real time for each experiment and compare performance between models. We ran more than 30 experiments over 2 weeks. Given our implementations, we observed several common behaviors for each model type (DCNN, DANN, MADA). These observations attest to the sanity of our training and revealed some challenges and limitations.

Table 1: Classification accuracies for test source and target domains for 3 domain adaptation tasks. Bold numbers correspond to our model performing better than the original papers [1][2], and gray numbers correspond to the accuracies in the original paper [1][2].

	MNIST → MNIST-M		Amazon → DSLR		Amazon → Webcam	
	source acc	target acc	source acc	target acc	source acc	target acc
Our DCNN (source only)	.991	.265 (.522)	.660	.600 (.689)	.762	.600 (.684)
Our DANN (from [1])	.991	**.791 (.766)**	.760	.600 (.797)	.700	.600 (.820)
Our MADA (from [2])	.817	.370	.740	.630 (.878)	.680	.650 (.900)
Our DCNN (target only)	.973 (.959)	.957	.360	.159	.987	.587

5.1 DCNN results

The DCNNs provide a baseline for this study to compare the performance on the target test domain of a model that has not learned from the target dataset. These models are trained only with the source (or target). They perform very well at test time on data from the same distribution but provide extremely poor results on any other (target) test domain. As we can see in this dashboard (https://wandb.ai/marvtin/MADA-PL/runs/3lmapqza) showing the training of a DCNN on the Webcam dataset, the training and validation loss steadily decreased and the training and validation accuracy on the same domain reached 98%. At the time of testing, the same model achieved 98.7% accuracy on the source test dataset (Webcam), while on the target test dataset (Amazon), it only achieved 58.7%. In our experiments, it is common for DCNNs to struggle to achieve good performance on the test domain dataset when trained on the source training dataset. The only exception is training on MNIST-M (as source) and testing on MNIST (as target), probably because the parameters learned on MNIST-M carry more general feature representations (as opposed to the other way around). You can see these results in table 1.

5.2 DANN results

Our experiments revealed that DANN performed better for generalization over the test target domain dataset. Unlike DCNN, we can now visualize the losses and accuracies of the domain classifier (see `https://wandb.ai/marvtin/MADA-PL/runs/3vjrq02d`). Our observations confirmed that the reverse gradient layer (GRL) confused the feature extractor and generated domain invariant features. These invariant features tend to mislead the domain classifier and thus increase its loss and decrease its accuracy over time. For most of the DANN experiments we conducted, the accuracy of the domain classifier increases sharply at the beginning of training (meaning that the domain classifier learns to distinguish domains correctly), but as lambda increases and the negative gradient is back-propagated through the feature extractor, the input fed to the domain classifier carries less and less background information and thus stops learning. This is mainly the reason why the accuracy of the domain classifier suddenly decreased in the middle of the training.

As you can see in Table 1, our DANN achieved similar accuracies as the original paper [1] for the MNIST→ MNIST-M data. However, for the Office-31 dataset, our results are less convincing, indeed we can see from the table that the performance of DANN on the target domains is almost as bad as that of DCNN. These results on the Office-31 dataset could stem from the fact that the domains are very unbalanced in terms of training examples. Since during training we have to provide the source

and target domains simultaneously, if the source has more examples than the target, it makes batch allocation difficult. This difficulty has resulted in overfitting for many of our DANN experiments on Office-31, where the training source accuracy is very high and the validation source accuracy is low and stable. We did not address these issues in the time available to us to complete this study.

5.3 MADA results

The MADA experiments were certainly the most challenging part of our study. We performed several experiments on the MNIST and Office31 datasets to try to obtain results comparable to those obtained in the paper [2]. First, it is important to mention that the MADA experiments are much more relevant on the Office31 dataset since the multimodal structure of this dataset can be captured by the multi-domain classifiers. Unlike MNIST-M which does not have relevant background information (since it is randomly generated), the background information of Office31 is highly dependent on the image labels (a bicycle may not have the same background as a calculator). Even though we continued the experiment on MNIST-M, we knew that the results would not improve over DANN (and this is what we actually observed).

Since we did not resolve the imbalance issues with Office31 that we observed with DANN, it is difficult to say whether our implementation of MADA is correct or not. For MADA trained on Amazon as the source dataset and Webcam as the target dataset, we observe that the accuracy of the 31 domain classifiers never decreases as we observed in DANN with MNIST. This means that the feature extractor always provides domain-related features and does not mislead the domain classifiers. Similarly to Generative Adversarial Networks [22] (GAN), the architecture of two-player neural networks is difficult to train. We can see from the reference table that the raw results of our MADA implementation are disappointing (far from the original paper) and that the accuracies of our MADA test target are actually similar to those of DCNN.

6 Conclusion

Our study evaluated the reproducibility of the domain matching papers Ganin, Yaroslav, et al. 2015 (DANN) and Pei, Zhongyi, et al. 2018 (MADA). These two papers have had a very significant impact on adversarial domain training to build more robust models by matching feature distributions across domains. Our study highlights the advantage of both architectures and reveals some of the implementation challenges. Overall, we were able to reproduce DANN results for handwritten digit classification across two separated domains. However, we did not obtain comparable results for MADA due to the instability of this model in terms of training.

7 Acknowledgments

This work was supported by Boston University. I would like to thank Professor Victor Berry for sharing his expertise throughout the CS664 Artificial Intelligence course. I would also like to Marvin Martin for his immense contribution to this project.

References

[1] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand & Victor Lempitsky (2015) Domain-Adversarial Training of Neural Networks. arXiv:1505.07818v4 [stat.ML]

[2] Zhongyi Pei, Zhangjie Cao, Mingsheng Long & Jianmin Wang (2018) *Multi-Adversarial Domain Adaptation.* arXiv:1809.02176v1

[3] John Blitzer, Ryan T. McDonald & Fernando Pereira (2006) Domain adaptation with structural correspondence learning. *Conference on Empirical Methods in Natural Language Processing,* pages 120–128, 2006.

[4] Lorenzo Bruzzone & Fernando Pereira (2010) Domain adaptation problems: A DASVM classification technique and a circular validation strategy *IEEE Transaction Pattern Analysis and Machine Intelligence,* 32(5):770–787, 2010.

[5] Xavier Glorot, Antoine Bordes & Yoshua Bengio (2011) Domain adaptation for large-scale sentiment classification: A deep learning approach. *ICML* , pages 513–520, 2011.

[6] Yujia Li, Kevin Swersky & Richard Zemel (2014) Unsupervised domain adaptation by domain invariant projection. *NIPS 2014 Workshop on Transfer and Multitask Learning,* 2014.

[7]Alex Krizhevsky, Ilya Sutskever & Geoffrey E. Hinton (2012) ImageNet Classification with Deep Convolutional Neural Networks. *NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1,* Pages 1097–1105, 2012.

[8]Y. LeCun, F.J. Huang & L. Bottou. (2004) Learning methods for generic object recognition with invariance to pose and lighting. *Computer Vision and Pattern Recognition, 2004, CVPR 2004.Proceedings of the 2004 IEEE Computer Society Conference on, volume 2,* pages II–97. IEEE, 2004.

[9]N. Pinto, D.D. Cox, & J.J. DiCarlo (2008) Why is real-world visual object recognition hard? *PLoS computational biology,* 4(1):e27, 2008.

[10]Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun (2015) Deep Residual Learning for Image Recognition arXiv:1512.03385v1 [cs.CV].

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. 2009, pp. 248–255, IEEE

[12]Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, & Qing He (2020) A comprehensive survey on transfer learning *Proceedings of the IEEE, Volume 109,* Pages 43-76, 2020.

[13] Falcon, W. "PyTorchLightning/pytorch-lightning." Pytorch Lightning (2021)

[14] L. Biewald, "Experiment Tracking with Weights and Biases," Weights Biases. Available: http://wandb.com/. [Accessed: 07/2021].

[15] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In ECCV, pages 213–226, 2010.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, November 1998

[17] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. IEEE Transaction Pattern Analysis and Machine Intelligence, 33, 2011.

[18] Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

[19] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." International Conference on Machine Learning. PMLR, 2019.

[20] Gardner, Matt W., and S. R. Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences." Atmospheric environment 32.14-15 (1998): 2627-2636.

[21] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[22] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).

[23] Van Der Maaten, Laurens. "Barnes-hut-sne." arXiv preprint arXiv:1301.3342 (2013).

A Appendix

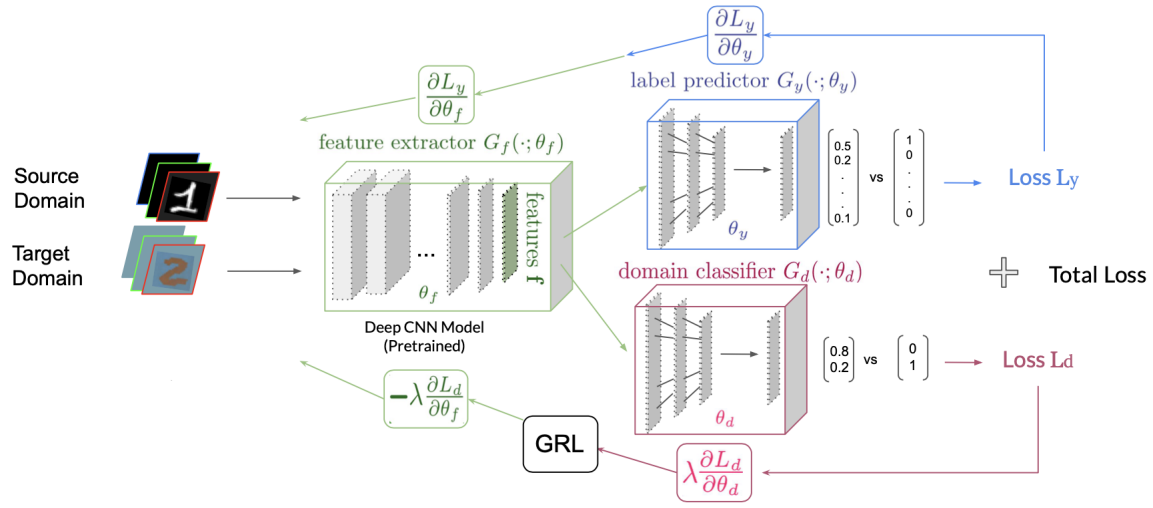

Figure 2: The proposed architecture of DANN by Ganin, Yaroslav, et al. "Domain-adversarial training of neural networks. 2015

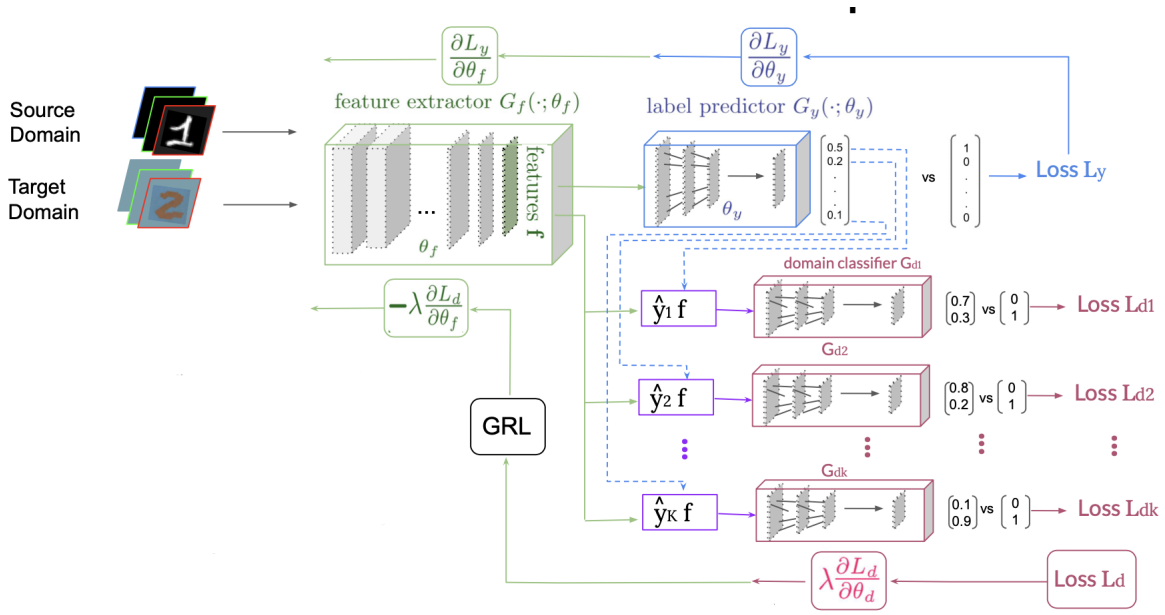

Figure 3: The proposed architecture of MADA by Pei, Zhongyi, et al. "Multi-adversarial domain adaptation." 2018