Domain Adaptation Using Small Group Learning

Aadil Ahamed
UC San Diego
aahamed@ucsd.edu

JiMin Mun
UC San Diego
jmmun@ucsd.edu

John O'Boyle
UC San Diego
jboboyle@ucsd.edu

Alex Tung
UC San Diego
aktung@ucsd.edu

Abstract

Domain adaptation is an unsupervised deep learning technique aimed to extract a meaningful representation of a target domain $\mathcal{D}^T$ using a model previously trained on a source domain $\mathcal{D}^S$. The method relies on the assumption that features common to both $\mathcal{D}^S$ and $\mathcal{D}^T$ have invariant properties. The existence of such features allows for disentanglement of the underlying factors of variation across the source and target domains. Different approaches to domain adaption have been implemented with varying degrees of success. In this paper, we explored applying small group learning to the problem of domain adaptation and evaluated its performance. Our final model, which composed an adversarial model with small group learning, achieved an accuracy of **72.6%** when performing domain adaptation from MNIST to the color-augmented MNIST-M dataset. The code for our project is available on Github.

1 Introduction

Transfer learning broadly describes the application of a trained neural network to another dataset. This technique can be classified by whether the adaptation is performed over the learning domain (i.e. feature space and distribution) or the learning task (i.e. label space, objective function)[1], of which we are interested in exploring the first. Domain adaptation transfers a model trained to a set task $\mathcal{T}$ on a source domain $\mathcal{D}^S$ to the same task on a different domain (target domain) $\mathcal{D}^T$ with the same feature space but a different marginal distribution of features [2].

2 Motivation

Domain adaptation is important as source and target distributions can differ in practice even when performing a conventional learning task with a constant dataset. Though network performance can be predicted by portioning training data into validation and test groups, the network may still suffer if deployed in an environment that diverges from the entire set[1]. As its name implies, optimization of a network using domain adaptation methods can improve its external validity to adapt to different domains, which in turn improves its robustness. With regard to supervised learning research in new fields, it can be expensive to label large amounts of data from the target domain if there are no existing datasets to train from – a problem that domain adaptation can also address.

Small group learning (SGL hereafter) is a newly proposed search method in which multiple networks, termed learners, are first independently trained on a labeled dataset, then cross-trained with datasets with pseudo-labels from other learners, then finally optimized on a validation set [3]. Given the nature of domain adaptation in exploring datasets that a given network has not been exposed to, any expansion in the understanding of SGL and its implementation would develop the relatively young field. In addition, the original authors implemented this topology in the context of a DARTS

[1]Under the Bayesian formulation of learning problems, this divergence is most commonly characterized as a change in prior or conditional probability distribution.

Preprint. Under review.

architecture search [4], but it would be interesting to explore different weight initializations to examine whether individual learners arriving at local loss minima can further converge towards a global minimum when cross-trained in a small group. This is of particular interest for optimization of non-convex problems.

3 Related Work

Domain adaptation was originally applied to the problem of sentiment classification across text reviews of different goods [5], but the principle has since expanded to include other sub-approaches. In their survey, Wilson et al. broadly classify these approaches into the following categories: domain invariant feature learning, domain mapping, normalization statistics and ensemble methods [1]. We will focus primarily on domain invariant feature learning (DIFL hereafter) as it relates closely to the discussions in the formulation of SGL. DIFL attempts to extract features from the input that generalize well to different domains [1]. The following sections summarize three common feature extraction methods used in DIFL.

The divergence method aims to minimize the distance between source and target distributions based on some metric. Common metrics used in this approach are maximum mean discrepancy, correlation alignment, contrastive domain discrepancy and the Wasserstein metric [1] [6].

The reconstruction method attempts to build a feature representation that can both classify labels against data from the source domain and reconstruct the original input from the source and target domains. There have been broad efforts made to adapt autoencoders to different domains altogether without necessarily keeping the feature spaces constant [7].

The adversarial learning method consists of a feature extractor and domain classifier acting as adversaries. The domain classifier tries to classify whether an input sample originated from the source or target domain and the feature extractor tries to extract domain invariant features that make it difficult for the domain classifier to classify correctly [8]. This topology can be also interpreted as the use of a regularized learning task.

4 Proposal

In this paper, we propose applying the SGL framework to the problem of domain adaptation. Specifically, we combine SGL with an existing adversarial network that has performed well for domain adaptation. We also propose a simplified model for architecture search, to help stabilize the training of the adversarial network.

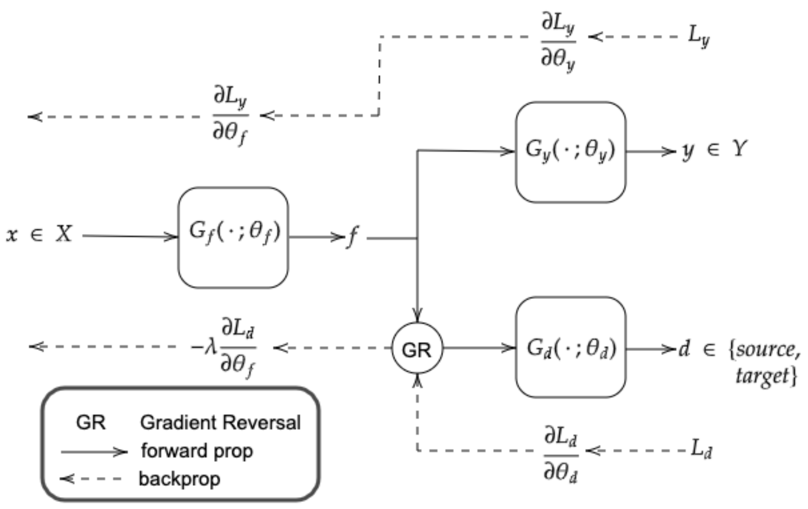
5 Methods:

5.1 DANN: Domain Adversarial Neural Network

One possible learner model is an adversarial network composed of a label classifier G_y, a binary domain classifier G_d and a feature extractor G_f [9]. For input space X, feature space f, label space Y, and domain label space $d \in \{source, target\}$ the classifiers can be expressed in function notation as follows:

$$G_f : X \mapsto f, \quad G_y : f \mapsto Y, \quad G_d : f \mapsto d$$

This network has two learning objectives. The first objective is preserved from the original SGL objective, which is to minimize the label prediction loss L_y. The second objective is to make the extracted features f domain invariant. More formally, we want to make the feature distributions extracted from the source domain $S(f) = \{G_f(x, \theta_f) | x \sim S(x)\}$ and target domain $T(f) = \{G_f(x, \theta_f) | x \sim T(x)\}$ similar before performing label classification. The performance of an optimized domain classifier G_d, is used as a measure of similarity between the two feature distributions: A domain classifier that performs poorly indicates that the extracted features are domain invariant, while one that performs well indicates that they are not. Hence, the second learning objective requires G_f and G_d to act as adversaries: The system searches for parameters θ_f that maximize L_d while simultaneously searching for parameters θ_d that minimize L_d.



$\frac{\partial L_y}{\partial \theta_y}$
L_y
$\frac{\partial L_y}{\partial \theta_f}$
$G_y(\cdot;\theta_y)$
$y \in Y$
$x \in X$
$G_f(\cdot;\theta_f)$
f
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
GR
$G_d(\cdot;\theta_d)$
$d \in \{source, target\}$
$\frac{\partial L_d}{\partial \theta_d}$
L_d
GR Gradient Reversal
forward prop
backprop

Figure 1: Adversarial network for domain adaptation

Combining these two learning objectives yields the following objective function to be optimized:

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

Specifically, we want to find the saddle point of $E(\theta_f, \theta_y, \theta_d)$:

$$\theta_f, \theta_y = \underset{\theta_f, \theta_y}{\operatorname{argmin}} E(\theta_f, \theta_y, \theta_d) \qquad \theta_d = \underset{\theta_d}{\operatorname{argmax}} E(\theta_f, \theta_y, \theta_d) \tag{2}$$

The stochastic gradient descent update for the parameters then becomes:

$$\theta_f \leftarrow \theta_f - \mu\left(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f}\right) \qquad \theta_y \leftarrow \theta_y - \mu \frac{\partial L_y^i}{\partial \theta_y} \qquad \theta_d \leftarrow \theta_d - \mu \frac{\partial L_d^i}{\partial \theta_y} \tag{3}$$

where μ is the learning rate and the loss functions L_y and L_d are multinomial and binomial cross entropy losses, respectively. The $-\lambda \frac{\partial L_d^i}{\partial \theta_f}$ term in the update for θ_f in Eq. 3 can be implemented through a gradient reversal layer. This layer can be represented mathematically as a pseudo-function $R_\lambda(x)$:

$$R_\lambda(x) = x \qquad \frac{\mathrm{d}R_\lambda(x)}{\mathrm{d}x} = -\lambda \tag{4}$$

Hence, during forward propagation, $R_\lambda(x)$ acts as an identity transform and during backpropagation it scales L_d by a factor of $-\lambda$. The diagram in Figure 1 illustrates the forward propagation of the inputs and backward propagation of gradients through the model.

5.2 SGL: Small Group Learning

SGL [3] is a multi-stage learning framework involving K learners, each with their own architectures A_k and two sets of weights $\{V_k, W_k\}$, that train cooperatively to learn some task T. Prior to training, the training data is split into three subsets: training data D^{tr}, unlabeled data D^{ul}, and validation data D^{val}. Each learner then goes through a pretraining stage, where they each update their weights V_k using the training data D^{tr} for some threshold number of epochs. Once pretraining completes, SGL, which consists of three stages, begins. In the first stage each learner keeps its architecture A_k fixed and updates its first set of weights V_k to V_k^* by minimizing the training loss on D^{tr}. In the second stage, each learner creates a pseudolabeled dataset D_k^{pl} by labeling D^{ul} using (A_k, V_k^*). Note here that the pseudolabels are represented as J dimensional vectors (J being the number of classes) where each class gets a probability between $[0, 1]$. Each learner then trains its second set of weights W_k to W_k^* by minimizing the training loss on $D^{tr} \cup \{D_{k'}^{pl}\}_{k' \neq k}^{K}$. In the final stage, each learner minimizes the validation loss on D^{val} by updating its architecture A_k with weights W_k^* using a differentiable search technique such as DARTS [4].

The learning framework can be written mathematically as:

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^{K}} \sum_{k=1}^{K} L(W_k^*, A_k, D^{val}) \\
s.t. \quad &\{W_k^*\}_{k=1}^{K} = \min_{\{W_k\}_{k=1}^{K}} \sum_{k=1}^{K} L(W_k, A_k, D^{tr}) + \lambda \sum_{j \neq k}^{K} L(W_k, A_k, D_j^{pl}(D^{ul}, V_j^*)) \qquad (5) \\
s.t. \quad &\{V_k^*\}_{k=1}^{K} = \min_{\{V_k\}_{k=1}^{K}} \sum_{k=1}^{K} L(V_k, A_k, D^{tr})
\end{aligned}$$

5.3 DARTS: Differentiable Architecture Search

DARTS is a neural architecture search method that relaxes the candidate search space to be continuous such that the architecture can be optimized via gradient descent. The advantage of DARTS is that it can achieve results comparable to state-of-the-art architecture search methods such as reinforcement learning and Bayesian learning while requiring significantly less computational resources. [4]. This is made possible because the candidate space is parameterized as a set of continuous mixing parameters rather than discrete decisions. The parameters are treated like a probability distribution via a softmax layer.

Following the approach used in SGL, the network architecture A_k and weights W_k are in the final stage of the learning framework via DARTS. A_k consists of three components: feature extractor A_f, label classifier A_y, and domain classifier A_d. In our model, we propose to keep the architectures of the label classifier and domain classifier fixed (same as the baseline) while updating the architecture of the feature extractor using DARTS. In effect, the present network can be seen as a direct combination of SGL and DANN.

In addition to the original DARTS model, we also try to implement a simplified model that is tuned for our application of domain adaptation. Similar to DARTS, two cell types are used, termed Normal and Reduction cells. The candidate space of a Normal Cell consists of operations that double the channel dimension but retain the spatial dimensions of the input. The candidate space of a Reduction Cell consists of operations that retain the channel dimension but halve the spatial dimensions of the input via pooling layers. Both cells consist of only a single node. This model significantly reduces the compute and memory resources required for the search, while still maintaining enough complexity to achieve reasonable performance. The candidate operations for both cell types are listed in Table 1. Each layer in the network is then composed of a Normal Cell and a Reduction Cell with a dropout layer in between. We stack two such layers together to form the architecture of our feature extractor A_f. Figure 2 illustrates the cell and layer architectures described above.

Cell Type	Candidate Operations
Normal	none, skip connection (Identity), 3x3 convolution and 5x5 convolution
Reduction	2x2 average pooling, 2x2 max pooling

Table 1: Candidate Operations for Normal and Reduction Cell used in simplified DARTS model

5.4 SGL + DANN

In this project we apply the SGL learning framework to the problem of domain adaptation. We accomplish this by using $K = 2$ learners, each using the adversarial model described previously, but with slightly different architectures and weight initialization. Since the adversarial model consists of

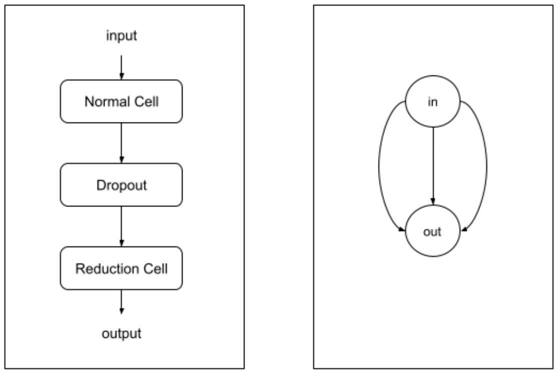


Figure 2: Layer and cell architecture of the simplified DARTS model. The figure on the left shows the architecture of each layer in the network. The figure on the right shows the cell architecture for both normal and reduction cells. The edges in the cell architecture correspond to candidate operations for the relevant cell type.

three subnetworks (feature extractor, label classifier and domain classifier), the parameters of each learner can also be broken into three sets of subparameters: $V_k = (\theta_f, \theta_y, \theta_d)_k$, $W_k = (\theta'_f, \theta'_y, \theta'_d)$ and $A_k = (A_f, A_y, A_d)$. Each learner will use the objective function $E(\theta_f, \theta_y, \theta_d)$ described in Eq. 1 as the loss function for each of the optimization steps presented in Eq. 5 to update the subparameters of V_k, W_k and A_k.

5.5 Color Space Mapping

In domain adaptation problems, there are numerous approaches in defining a source and domain-shifted target distribution. Common domain adaptation applications use distributions varying in sparsity and environment. Although the original DANN implementation is evaluated by learning across different numerical image sets (e.g. MNIST, SVHN), the choice was made to evaluate the current work using CIFAR, the dataset used by the SGL authors, then incorporate domain adaptations on it. We are interested in shifting the distributions by changing the way image information is stored. Our approach is to use the CIFAR-10 dataset as $\mathcal{D}^S$ encoded in RGB format and create a modified CIFAR-10 dataset as $\mathcal{D}^T$ encoded in HSV format. HSV stands for Hue (dominant color), Saturation (purity of the color), and Value[2] (brilliance of the color). Transcoding a picture from RGB into HSV allows for visual information to be mapped to a space that more closely corresponds to humans' perception of color.

The following steps are followed to yield an augmented CIFAR-10 set in HSV format.

$$(\hat{R}, \hat{G}, \hat{B}) = (R/255,\ G/255,\ B/255) \tag{6}$$

$$\Delta = \max(\hat{R}, \hat{G}, \hat{B}) - \min(\hat{R}, \hat{G}, \hat{B}) \tag{7}$$

$$H = \begin{cases} 60^\circ \cdot (\frac{\hat{G}-\hat{B}}{\Delta} mod(6)) & \text{if } \max(\hat{R}, \hat{G}, \hat{B}) = \hat{R} \\ 60^\circ \cdot (\frac{\hat{B}-\hat{R}}{\Delta} + 2) & \text{if } \max(\hat{R}, \hat{G}, \hat{B}) = \hat{G} \\ 60^\circ \cdot (\frac{\hat{R}-\hat{G}}{\Delta} + 4) & \text{if } \max(\hat{R}, \hat{G}, \hat{B}) = \hat{B} \end{cases} \tag{8}$$

$$S = \begin{cases} \frac{\Delta}{C_{max}} & \text{if } C_{max} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$V = C_{max} \tag{10}$$

It is worth noting that CIFAR-10 is in the sRGB format which represents a non-linear distribution of color values corresponding to colors that humans are most sensitive to, so a preliminary conversion must be made to convert sRGB representation to RGB by linearizing it by power-law of 2.2 [10].

[2]"Value" is commonly substituted with "lightness", so HSV and HSL are used interchangeably.

There are some insights that can be gained from the distributions of the two spaces. If an RGB-encoded pixel is interpreted as a linear combination of Cartesian vectors representing Red, Green, and Blue components, then an HSV-encoded pixel can be interpreted as a cyclical component (Hue) with two Cartesian bases (Saturation and Value) – effectively yielding a cylindrical coordinate system. These can be seen as equally capable of producing colors in a set in the $\mathbb{R}^3$ space[3].

Our model aims to build features invariant to each images pixel encoding such that classification can be achieved on distributions of various image representation. Should it be found that the model performs well on RGB to HSV domain shifts, future steps include exploring the domains associated with representing images in other color spaces (e.g. CMYK, YPbPr).

6 Experiments:

6.1 Baseline

We first conducted experiments on the baseline DANN model [4] to get a sense of the expected results and losses. We ran the baseline model with the MNIST [11] dataset as the source domain and the MNIST-M [9] dataset as the target domain. The MNIST-M dataset contains images of handwritten digits from MNIST blended with color patches. We split the training data $50:50$ into training and validation sets and used the following hyperparameters: $epochs = 20, batch\_size = 64, learning\_rate = 1e-3$ with Adam [12] as our optimizer. With these settings we were able to achieve an accuracy of **74.2%** on the test set for the target domain.

Next, to understand the impact of the architecture of the feature extractor on the performance of the classifier, we repeated the experiment above after making various modifications to the baseline architecture. The results are reported in Table 2. The main takeaway from this experiment is to note the impact that dropout and convolution kernel size has on the performance of the model. The no_dropout and 7x7_conv architectures in Table 2 show a huge drop in test performance when compared to the baseline model. The poor performance of 7x7_conv was not too surprising since this is quite a large kernel for the relatively small 28×28 images present in MNIST and MNIST-M datasets. However, the 32.7% drop in accuracy due to removing dropout was much more surprising. We think this might be because dropout helps prevent the feature extractor from overfitting on the training data and the current version of the domain classifier.

6.2 SGL+DANN

In this section we discuss the experiments we ran on our integrated model combining SGL with DANN. Here, we keep the architecture A_y of the label classifier and A_d of the domain classifier fixed while updating the architecture A_f of the feature extractor using DARTS.

6.2.1 SGL+DANN with Original DARTS

For our initial experiment, we used the original DARTS search cell, consisting of 4 internal nodes with the same candidate operations proposed in DARTS. We stacked 4 of these cells together to form the architecture of the feature extractor and used the same architecture for both training and testing. We used the following hyperparameters for this experiment: $pretrain\_epochs = 2, epochs = 5, batch\_size = 32, learning\_rate = 1e-3$, and Adam as the optimizer. We only use a $batch\_size = 32$ and $epochs = 5$ due to time and memory constraints. With these settings, we were able to achieve an accuracy of **18.6%** on the MNIST-M dataset. To analyze this poor result we compared the label and domain loss between the SGL+DANN and base DANN models. We noticed that SGL+DANN appeared to minimize label loss faster, but its domain loss, particularly the validation domain loss was higher. We think this might be because the DARTS feature extractor has many more parameters when compared to the domain classifier which can lead to mode collapse as it will overfit its output for a specific domain classifier. Additionally, the DARTS cell doesn't include dropout, which from our baseline experiments, proved to be very important in improving the

[3]Strictly speaking, an n-bit color space is a set of regularly sampled discrete points in $\mathbb{R}^3$ but is not a vector space. The range is limited by $R, G, B \in [0, 255]$ and the resolution of points depends on the bit depth n.

[4]https://github.com/fungtion/DANN_py3

Label	Architecture	Test Acc %
baseline	Conv2D(in_ch=3, out_ch=64, kernel_size=5), BatchNorm2D(), MaxPool2D(kernel_size=2, stride=2), Conv2D(in_ch=64, out_ch=50, kernel_size=5), Dropout2D(), MaxPool2D(kernel_size=2, stride=2)	74.2
no_dropout	Conv2D(in_ch=3, out_ch=64, kernel_size=5), BatchNorm2D(), MaxPool2D(kernel_size=2, stride=2), Conv2D(in_ch=64, out_ch=50, kernel_size=5), **NoDropout**, MaxPool2D(kernel_size=2, stride=2)	41.5
avg pool	Conv2D(in_ch=3, out_ch=64, kernel_size=5), BatchNorm2D(), **AvgPool2D**(kernel_size=2, stride=2), Conv2D(in_ch=64, out_ch=50, kernel_size=5), Dropout2D(), **AvgPool2D**(kernel_size=2, stride=2)	70.4
3x3_conv	Conv2D(in_ch=3, out_ch=64, **kernel_size=3**), BatchNorm2D(), MaxPool2D(kernel_size=2, stride=2), Conv2D(in_ch=64, out_ch=50, **kernel_size=3**), Dropout2D(), MaxPool2D(kernel_size=2, stride=2)	74.2
7x7_conv	Conv2D(in_ch=3, out_ch=64, **kernel_size=7**), BatchNorm2D(), MaxPool2D(kernel_size=2, stride=2), Conv2D(in_ch=64, out_ch=50, **kernel_size=7**), Dropout2D(), MaxPool2D(kernel_size=2, stride=2)	57.2
5x5_conv pad2	Conv2D(in_ch=3, out_ch=64, kernel_size=5, **padding=2**), BatchNorm2D(), MaxPool2D(kernel_size=2, stride=2), Conv2D(in_ch=64, out_ch=50, kernel_size=5, **padding=2**), Dropout2D(), MaxPool2D(kernel_size=2, stride=2)	77.1
ch_48_96	Conv2D(in_ch=3, **out_ch=48**, kernel_size=5), BatchNorm2D(), MaxPool2D(kernel_size=2, stride=2), Conv2D(in_ch=48, **out_ch=96**, kernel_size=5), Dropout2D(), MaxPool2D(kernel_size=2, stride=2)	76.9

Table 2: Test Accuracy on MNIST-M for different feature extractor architectures while using the baseline label and domain classifiers

performance of the model. We believe these two factors are the main reason why the SGL+DANN model performs so poorly when compared with the baseline.

6.3 SGL+DANN with simplified DARTS

Next, we conducted an experiment with the simplified DARTS model described in Section 5.3. The simplified DARTS model uses a two layer network consisting of one normal cell and one reduction cell with a dropout layer in between. Using this simplified DARTS model, we reran SGL+DANN with the following hyperparameters: $pretrain\_epochs = 2, epochs = 20, batch\_size = 64, learning\_rate = 1e-3$. The simplified DARTS model requires significantly less compute and memory resources, when compared with the original DARTS model, hence we could increase the batch size and number of epochs and still train the model very quickly. With these settings we were able to achieve an accuracy of **72.6%** on the MNIST-M dataset. This is much closer to the baseline accuracy achieved by DANN, which is not surprising since the converged architecture is very similar to the baseline architecture. To save time, we reuse the weights learned during training during test time rather than retraining the model from scratch. This might explain why the performance of the simplified DARTS model is still a bit lower than that of the baseline.

Model	MNIST-M accuracy %
baseline	74.2
SGL+DANN with DARTS	18.6
SGL+DANN with simplified DARTS	72.6

Table 3: Summary of results on MNIST-M test set for the experiments conducted in this paper.

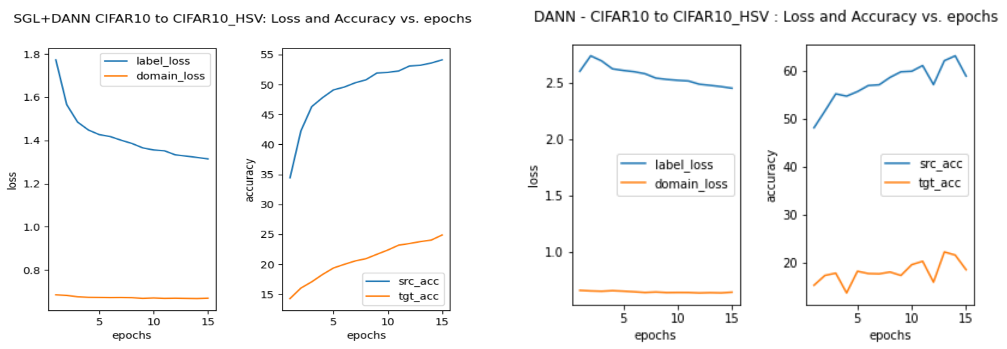


Figure 3: SGL+DANN (left), DANN (right): Label loss, domain loss, source accuracy and target accuracy from CIFAR10 (source) to CIFAR10-HSV (target)

.

6.4 Color Space Mapping

Color Space Mapping experiment was conducted so that we first change the target dataset to HSV following Section 5.5. We used this transformed target CIFAR10 dataset as our target dataset for domain adaptation and the results [5] are shown in Figure 3 .

The results show that SGL+DANN appears to perform better than DANN. While this observation appears promising, it should be noted that the DANN architecture used was designed for 1-channel 28x28 sized images. It is possible that our modifications of the architecture to use 3 channel 32x32 is sub-optimal and could be improved. Further experimentation is needed to conclude that our model is a better domain adaptation solution than DANN for CIFAR-10 to CIFAR-10 domain shifts.

While results showed SGL+DANN with simplified DARTS outperformed DANN in color space transformations, the overall performance was worse than our model's performance on MNIST to MNIST-M domain shifts. This can be explained by the fact that the mapping from RGB to HSV is nonlinear due to the modulo and max/min operations. In contrast, the MNIST-M augmentation simply splits a single dimension of grayscale lightness values into three color dimensions based on the BSD500 color patches. Given that most of the shift in MNIST-M is in hue, it may be worth using HSV-encoded MNIST-M as a target distribution, though it is beyond the time constraints of this project and can be reserved as a future exploration.

7 Conclusion

In this project, we tackled the problem of domain adaptation using the small group learning framework. In our formulation, each learner in SGL consisted of the DANN model with a differentiable feature extractor architecture.

From our baseline experiments we discovered that the dropout layer is very important for achieving strong performance. We then conducted an experiment to evaluate the performance of DANN composed with SGL on the MNIST to MNIST-M datasets but observed poor performance due to mode collapse resulting from the large number of parameters in the DARTS based feature extractor and the lack of dropout layers. Next, we conducted an experiment using a simplified DARTS model which reduced the number of parameters in the feature extractor and also added dropout layers. We achieved a final accuracy of **72.6%** on the MNIST-M test set using this model. We also proposed an original domain adaption dataset by transforming image color spaces, and tested our model on this transformation of the target domain. We found our model outperformed DANN on these color space transform domain shifts, but suspect incorrect modifications to the DANN network layers led to this result. Additionally, the search space for our simplified DARTS model is very small and hence is unable to generalize to more complex datasets. We believe that spending more time formulating a bigger architecture search space for adversarial networks that maintains the balance of feature extractor and domain classifier during training would be the best way to overcome this issue.

[5] A late recognition of a bug in the color transformation code leads us to the believe the results gathered in section 6.4 may be invalid.

8 Individual Contributions

8.1 Aadil:

I implemented the code to integrate SGL with DANN and the code for the simplified DARTS model. For the experiments, I collected data for the results presented in sections 6.1 to 6.3 (baseline, original DARTS and simplified DARTS). For the report, I wrote most of the DANN and SGL content in sections 5 and 6 (Methods and Experiments) and also contributed to the introduction and conclusion sections.

8.2 Alex:

I conducted the initial literature review regarding domain adaptation problems with specific regard to transformations in the covariate space. I formalized the theoretical foundation between the color mapping experiment and the baseline MNIST-M transformation and provided direction and pair-programming guidance to the team on integrating the mappings into the SGL-DANN framework. I contributed to the introduction, color space exploration, and discussion sections regarding differences in network performance.

8.3 Jack:

I implemented code for baseline testing DANN on MNIST$\rightarrow$MNIST-M and CIFAR10$\rightarrow$CIFAR10-HSV and worked with JiMin in writing code color spaces transformations for CIFAR10. For the experiments, I collected the data for SGL+DANN on CIFAR10$\rightarrow$CIFAR10-HSV, as well as DANN MNIST$\rightarrow$MNIST-M and CIFAR10$\rightarrow$CIFAR10-HSV. For the report, I contributed to the abstract, introduction, color space mapping, plots, and conclusion.

8.4 JiMin:

I implemented the code for color space mapping transformation of CIFAR10 dataset as well as the experiment using the transformed data. For the report I wrote parts of the color space mapping method discussion and the experiment specification.

References

[1] Wilson, G., D. J. Cook. A survey of unsupervised deep domain adaptation, 2020.

[2] Wang, M., W. Deng. Deep visual domain adaptation: A survey, 2018.

[3] Du, X., P. Xie. Small-group learning, with application to neural architecture search, 2020.

[4] Liu, H., K. Simonyan, Y. Yang. Darts: Differentiable architecture search, 2019.

[5] Glorot, X., A. Bordes, Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 513–520. Omnipress, Madison, WI, USA, 2011.

[6] Olkin, I., F. Pukelsheim. The distance between two random vectors with given dispersion matrices. *Linear Algebra and its Applications*, 48:257 – 263, 1982.

[7] Ghifary, M., W. B. Kleijn, M. Zhang, et al. Domain generalization for object recognition with multi-task autoencoders. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2551–2559. 2015.

[8] Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, et al. Generative adversarial networks, 2014.

[9] Ganin, Y., V. Lempitsky. Unsupervised domain adaptation by backpropagation, 2015.

[10] Gowda, S. N., C. Yuan. Colornet: Investigating the importance of color spaces for image classification. *CoRR*, abs/1902.00267, 2019.

[11] LeCun, Y., C. Cortes. MNIST handwritten digit database. 2010.

[12] Kingma, D. P., J. Ba. Adam: A method for stochastic optimization, 2017.