AI1603 ASSIGNMENT REPORT — RECURRENCE AND EXPLORATION OF OOD ALGORITHMS [*]

Bangjun Wong
Shanghai Jiao Tong University
Shanghai, China
{Bangjun Wong}wangbangjun@sjtu.edu.cn

ABSTRACT

This assignment report is going to show you the backgrounds for Out-of-Distribution problems and how we apply different methods to accomplish those tasks and optimize our algorithms.The dataset we are provided with is ColoredMNIST, a dataset proposed by the paper *Invariant Risk Minimization* for validating their feasibility. For the first part, we explore the possibility of directly applying LeNet on the classification work and try to do some data augmentations to improve LeNet's performance(All of the part is done by Bangjun Wong). For the second part, we try to recur two classical OoD algorithms — IRM and DANN, which are respectively done by Xiangyuan Xue and Bangjun Wong. Based on these, we propose our optimization methods and blueprints. Finally, we have achieved comparatively promising results.

***K*eywords** OoD · Independent-Identically

1 Problem Setting

1.1 Backgrounds

Deep Neural Networks usually are trained based on the hypothesis that all data are *Independent-Identically* distributed, i.e: training data and testing data share the same or very similar distribution patterns. However, when we apply models to realistic situations, the hypothesis can be rarely true. As a result, dramatic declines on accuracy are witnessed. While those declines are acceptable for applications like recommendation systems, it's dangerous when applied to medicine areas where tolerance of risks is rather small. Promising artificial intelligence system ought to still have robust generalizing abilities under *Out-of-Distribution* circumstances. And the key to improving model's generalization abilities lies in how to drive our models to learn the causal feature of data.

A simple example: take cat-dog binary classification as an example, if all the dogs in the training set are on the grass and all the cats are on the sofa, and all the dogs in the test set are on the sofa and all the cats are on the grass, then the model, without the information of the test set, is likely to associate the grass with the dogs and the sofa with the cats based on the information of the training set, and when the model is tested on the test set will will mistake the dog on the couch for the cat.

1.2 Colored MNIST Dataset

1.2.1 Initialization

Dataset provided in this assignment is ColoredMNIST, a dataset evolved from MNIST proposed by Arjovsky et al.[1]. Three environments (two training, one test) are defined from MNIST transforming each example as follows: first, assign a preliminary binary label $\tilde{y}$ to the image based on the digit: $\tilde{y} = 0$ for digits $0 - 4$ and $\tilde{y} = 1$ for $5 - 9$. Second, obtain the final label y by flipping $\tilde{y}$ with probability 0.25. Third, sample the color id z by flipping y with probability p^e,

[*] *With help from TA Fan Wu*

where p^e is 0.2 in the first environment, 0.1 in the second, and 0.9 in the test one. Finally, color the image red if $z = 1$ or green if $z = 0$.

Via the operations above, the MNIST is evenly split into 3 enviroments (train1, train2, test). To simulate real-life occasions, 25% labels are flipped which leads to the maximal theoretical accuracy descending to 75%. Besides, image's color is randomly flipped in different proportion related to the environment it belongs to.

Environment	**Size**	**Label Flipped**	**Color Flipped**	**Theoretical Accuracy**
train1	20000	25%	20%	75%
train2	20000	25%	10%	75%
test	20000	25%	90%	75%

Table 1: Properties of ColoredMNIST

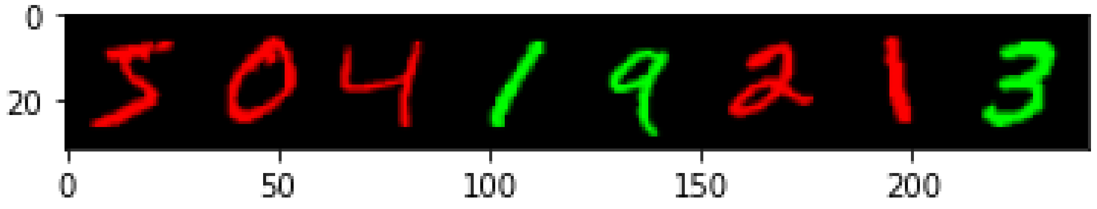

Figure 1: Colored MNIST

.

Different OoD algorithms are applied to train and test on the dataset in this paper. Our goal is to recur those mainstream OoD algorithms and try to optimize them.

1.2.2 Intervention Analysis

From the initialization progress, we find that labels are created according to the contours of numerals in the images and there is no relationship between labels and images' colors. As a result, by definition in OoD fields, contours(or numerals) are the **causal features** which holds the invariant information of the dataset, and colors are the **non-casual features** or **spurious features** which have no direct relationship with the labels.

2 Preliminary Stage — LeNet

2.1 Standard LeNet

Since LeNet succeeded in settling the multi-classfication work on MNIST and Colored MNIST evolves from MNIST, we naturally wonder whether LeNet can achieve similar success and prove its robustness when MLP fails. On the first stage, I directly apply the standard LeNet to the classification work. Since data in MNIST has only one channels and produces 10 outputs while ColoredMNIST has 3 channels and only 2 outputs, I slightly modify the model to fit the dataset provided as is shown in Figure 2.

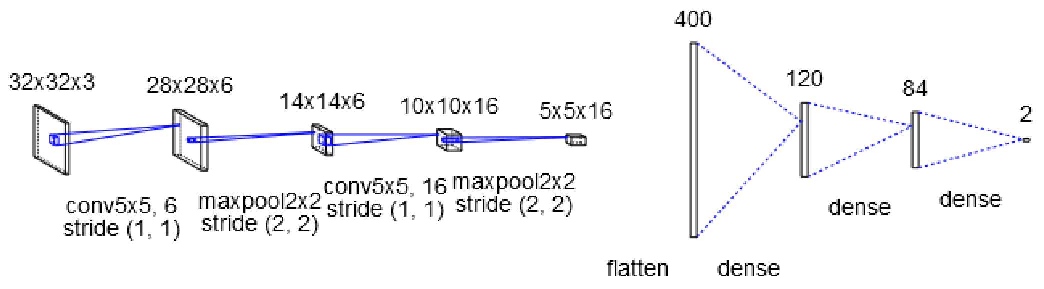

Figure 2: Modified Standard LeNet

.

Since train1 and train2 share similar data distribution, there is no need to make a comparison on training performance between them. Results below are ploted based on both train1 and train2.

With training loss descending, the test accuracy is still unstable and vibrates drastically, showing no signs of convergence. The average $TestAccuracy$ reaches about 38 %. Compared with the MLP applied in Homework 3($TestAccuracy = 10\%$), performance has risen a lot but is still far from our goals.So next, we focus on how to optimize LeNet or do something with the training set in order to reach higher accuracy.

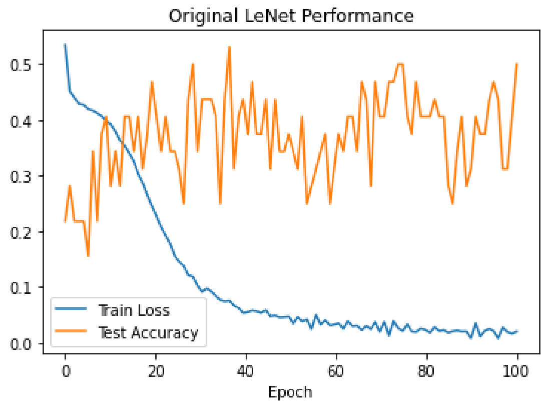

Figure 3: Original LeNet Performance

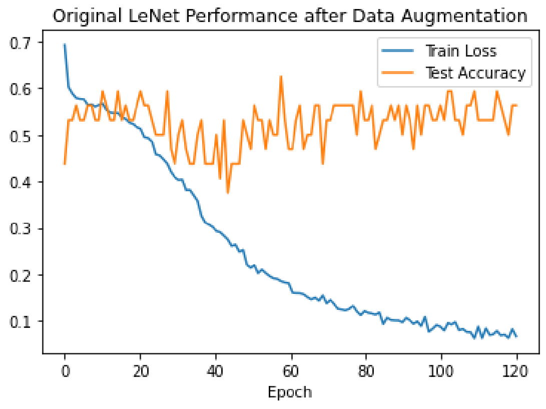

Figure 4: Data Augmented LeNet

2.2 Data Augmentation

In order to eliminate the divergence caused by color and enable our model itself to abandon its selection of learning color feature. We decide to perform data augmentation on the Colored MNIST dataset by the following steps: first, we traverse the first 20000 training data. Second, we randomly flip the colors at the possibility of 20% .Third, we append them to the dataset. Looping in this way for 10 times, then we get a dataset of 200000 samples.

In this way, every image shows up for 10 times and as a result, the whole training dataset is escalated by 9 times, meanwhile we maintain the original color-flipping feature. Through this method, we not only can ’teach’ our model to omit the feature — color, we also succeed in maintaining the properties original training dataset holds.

Data Augmentation	**Size**	**Label Flipped**	**Color Flipped**	**Average Accuracy**
NO	20000	25%	20%	38.4%
YES	200000	25%	15%	56.2%

Table 2: Then, we load the augmented dataset into our previously-constructed model. As is shown in Figure 4, the $TestAccuracy$ has been drastically elevated and converges at about 56.2%. It firmly indicates that our tactics of data augmentation works and have led the model to focus on the contour feature and omit colors.

3 Domain-Adversarial Training of Neural Networks (DANN)

3.1 Overview

DANN is a classical algorithm in the Adversarial transfer learning field. Ganin, Yaroslav et al.[2] firstly introduced adversarial theory into transfer learning. In traditional machine learning, we often need large amounts of labeled data for training, and have to ensure training dataset and testing dataset shares similar data distribution. **Domain Adaptation** is an important branch in transfer learning, which aims at mapping source domain and target domain into the same eigenspace and finding a certain distance metering principle to make distances as small as possible (as is shown in Figure 5). Then we can directly use the label classifier trained on the source domain to classify the data on target domain. GANN shares the similar network structure with GAN[3], and is composed of mainly 3 parts:

- **Feature Extractor:** Map data into a certain eigenspace so that while label classifier is able to judge source domain data’s labels, domain classifier cannot tell which domain data in eigenspace are transformed from.
- **Label Classifier:** Classify the data from source domain and acquire accuracy as high as possible.

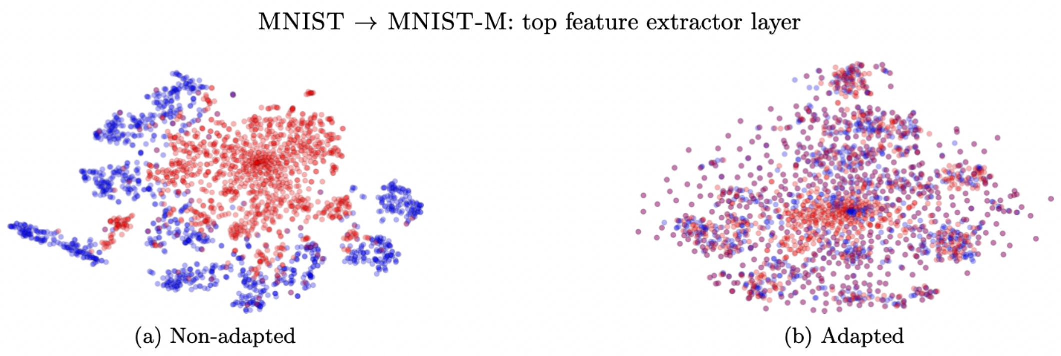

Figure 5: Example of Domain Adaptation
.

- **Domain Classifier:** Classify the data in eigenspace and try its best to classify which domain they are transformed from.

Feature extractor and label classifier constructs a forward-propagation neural network. Behind the feature extractor, we add a domain classifier connected with a ***Gradient Reverse Layer*** (GRL). During training process, for labeled data from source domain, networks keep minimizing the loss of label classifier and for all data, networks keep minimizing the loss of domain classifier. Using the theory of GAN, we can interpret fearture extractor as a **Generator**, and domain classifier as a **discriminator.**

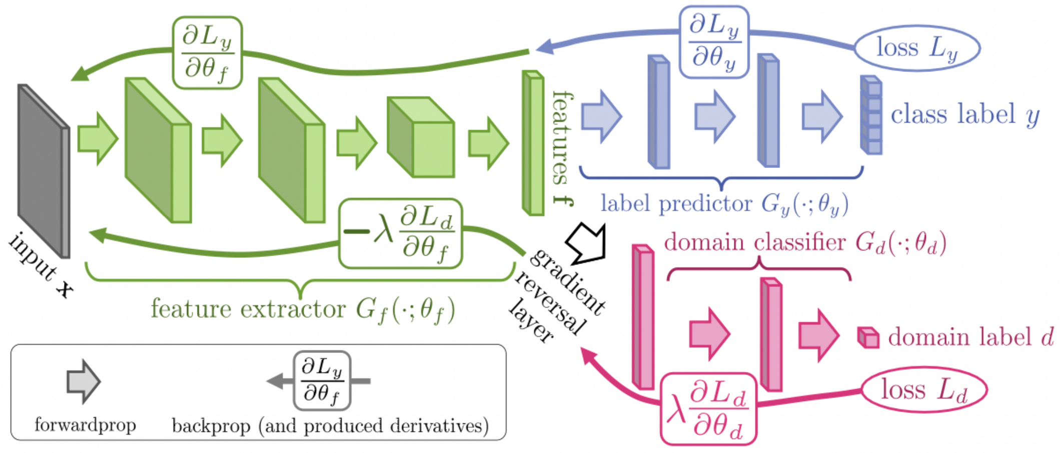
$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
features f
class label y
label predictor $G_y(\cdot;\theta_y)$
$-\lambda\frac{\partial L_d}{\partial \theta_f}$
input x
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
feature extractor $G_f(\cdot;\theta_f)$
domain label d
loss L_d
$\lambda\frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

Figure 6: DANN Network Structure
.

3.2 Theory

DANN's whole theoretical standpoint is based on the seminal theoretical works of Ben-David et al. (2006, 2010). It reckons that: Given two distribution:D_S^X D_T^X and a classification function $H$the $H_{divergence}$ can be written as :

$$d_H(D_S^x, D_T^x) = 2sup_{\eta\in H}\left|Pr_{x\sim D_S^x}[\eta(x)=1] - Pr_{x\sim D_T^x}[\eta(x)=1]\right|$$

If there exists classification function η that can completely tell data from source domain and target domain apart, then $H_{divergence} = 2$. Otherwise, the worse the classification function is , the smaller divergence is. Since in most cases, we cannot acquire all the data on source domain and target domain , Ben-David et al. gives a method to estimate divergence by sampling the data as follows:

$$\hat{d_H}(S,T) = 2\left(1 - min_{\eta\in H}\left[\frac{1}{n}\sum_{i=1}^{n} I[\eta(x_i)=0] + \frac{1}{n'}\sum_{i=n+1}^{N} I[\eta(x_i)=1]\right]\right)$$

$1-n$ come from the source domain and $n-N$ come from the target domain, i.e:

$$U = \{(x_i,0)\}_{i=1}^{n} \cup \{(x_i,1)\}_{i=n+1}^{N}$$

I is a sign function which equals 1 when predicts true , otherwise 0, i.e: The higher the prediction accuracy rate is , the smaller divergence is.

However, in realistic situations, it's hard to calculate divergence because it involves finding a best η in the function space, so in most cases learning algorithms are used. We use ϵ to represent the error rate, then the estimation of divergence can be represented by:

$$\hat{d_A} = 2(1-2\epsilon)$$

Meanwhile, Ben-David proved that $d_H(D_S^x, D_T^x)$ is the maximum of $\hat{d_H}(S,T) + O(d) + O(N)$. d represents the dimensionality of H 's VC dimension. So the risk on target domain satisfies:

$$R_{D_T}(\eta) \le R_S(\eta) + \sqrt{\frac{4}{n}(dlog\frac{2en}{d} + log\frac{4}{\delta})} + \hat{d_H}(S,T) + 4\sqrt{\frac{1}{n}(dlog\frac{2n}{d} + log\frac{4}{\delta})} + \beta$$

$\beta \ge inf_{\eta*\in H}[R_{D_S}(\eta*) + D_{D_T}(\eta*)]$ represents the minimum of the sum of classification function's losses on source domain and target domain, which indicates that given VC dimension, to reduce the loss on target domain, we have to minimize: $R_S(\eta) + \beta + \hat{d_H}(S,T)$. While under the circumstances that we target domain's labels are invisible to us, all we can do is to optimize: $R_S(\eta) + \hat{d_H}(S,T)$. DANN is constructed to optimize this function.

3.3 Network Structure

Ganin, Yaroslav et al. firstly only considers a shallow neural network with a single hidden layer. For simplicity, they suppose that the input space is formed by $m - dimensional$ vectors.

$$G_f(x;W,b) = sigm(Wx+b)$$
$$sigm(a) = [\frac{1}{1+exp9-a_i}]_{i=1}^{|a|}$$

At the last prediction layer, they perform a L classification with a softmax function, i.e: $G_y : R^D \to [0,1]^L$, and use the Logarithmic probability of correct labels as loss function, i.e:

$$L_y(G_y(G_f(x_i)), y_i) = log\frac{1}{G_y(G_f(x))_{y_i}}$$

Finally, when training on networks, regularizers can be added. Because hidden layers can be viewed as the internal representation of the neural network. Thus we denote the source sample representations and target sample representations as :

$$S(G_f) = \{G_f(x)|x\in S\}$$
$$T(G_f) = \{G_f(x)|x\in T\}$$

According to the theory above, the empirical $H_{divergence}$ of a symmetric hypothesis class H

$$\hat{d_H}(S(G_f),T(G_f)) = 2\left(1 - min_{\eta\in H}\left[\frac{1}{n}\sum_{i=1}^{n} I[\eta(G_f(x_i))=0] + \frac{1}{n'}\sum_{i=n+1}^{N} I[\eta(G_f(x_i))=1]\right]\right)$$

To optimize divergence, all we have to do is to add a layer to classify data where they come from to represent η. Logistic regression can be used here and loss function can also use Logarithmic loss.

$$G_d(G_f(x);u,z) = sigm(u^T G_f(x) + z)$$

For the original network, our goal is to make it hard for network features be classified by domain classifer, so the regularizer can be designed as:

$$R(W,b) = max_{u,z} - \left[\frac{1}{n}\sum_{i=1}^{n} L_d^i(W,b,u,z) + \frac{1}{n'}\sum_{i=n+1}^{N} L_d^i(W,b,u,z)\right]$$

Thus, the smaller the regularizer is, the worse domain classifier's best performance is. Then we have the ultimate goal function:

$$E(W,V,b,c,u,z) = \frac{1}{n}\sum_{i=1}^{n} L_y^i(W,b,V,c) - \lambda\left(\frac{1}{n}\sum_{i=1}^{n} L_d^i(W,b,u,z) + \frac{1}{n'}\sum_{i=n+1}^{N} L_d^i(W,b,u,z)\right)$$

Equations explained above mainly derives from DANN original paper and cryan's experiments[4]

3.4 Implementations:

Considering similarity between Colored MNIST and MNIST-M proposed in [2], I apply the same network as Ganin, Yaroslav et al. did when they chose MNIST as the source domain and MNIST-M as the target domain.

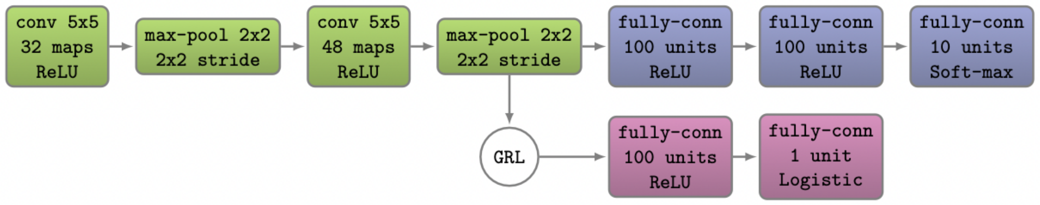

Figure 7: Example of Domain Adaptation

.

3.4.1 Stage 1:

On the first stage, I select the multi-domain which combines both train1 and train2 as the source domain, and test as the target domain. When feature extractor (CNN) is trained on source-only domain and optimized by Adam, its accuracy reaches 89.4%. When directly transfer the source-only trained model to the target domain, the test accuracy steeply drop to 13.67%, which is only a little bit higher than the MLP trained in previous task3 and can be viewed as the worst accuracy. When directly trained on the target domain, the accuracy reaches 90.03%, which represents the best accuracy for DANN to reach. However, when I directly apply DANN to the classification work, little progress has been witnessed.

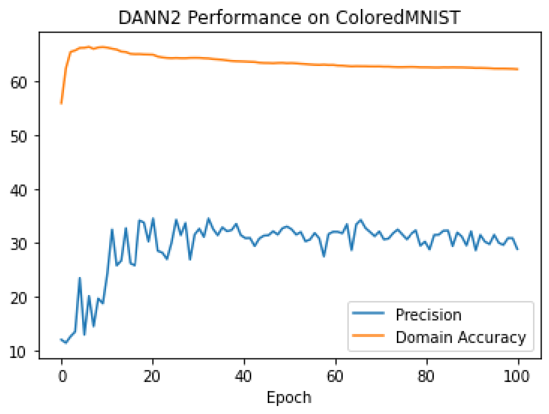

Figure 8: Precision and Domain Accuracy

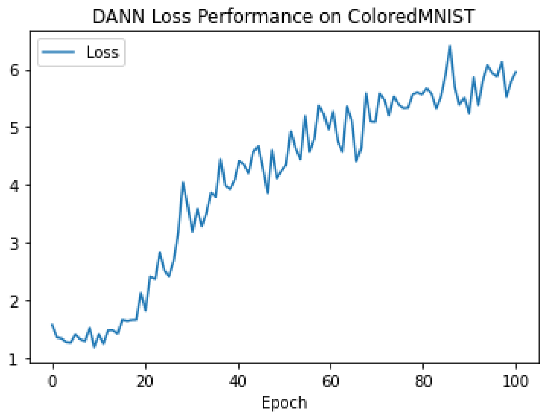

Figure 9: Loss

As is shown in Figure 8, one of the reason why model's precision rises only a little bit is the low domain accuracy. Due to the high color-flipping rate (90%) in target domain and the low color-flipping rate (15%) in source domain, the CNN feature extractor inevitably learn the color feature in the source domain. One of the methods proposed in [2] is use PCA before the feature extractor. However, when reducing the dataset's dimensionality to 2, it equals to manually abandon the color feature both in target domain and source domain. So in next stage, I change my selection of source domain and target domain, and reaches considerably higher accuracy.

3.4.2 Stage 2:

In order to leading the model to learn more features about the shapes or contours of the numerals in the images, I construct a new dataset named MNIST-source via the following steps:first, assign a preliminary binary label $\tilde{y}$ to the image based on the digit: $\tilde{y} = 0$ for digits $0 - 4$ and $\tilde{y} = 1$ for $5 - 9$. Second, obtain the final label y by flipping $\tilde{y}$ with probability 0.25. Third, sample the color id z by flipping y with probability p^e, where p^e is 0.2 in the first environment, 0.1 in the second, and 0.9 in the test one. Finally, I did not color the image by z's value.

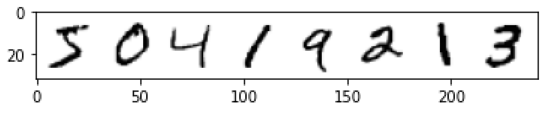

Figure 10: Source Domain

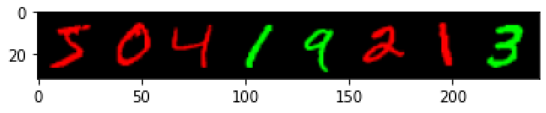

Figure 11: Target Domain

Training Method	source-only training accuracy	direct transferring accuracy	Color Flipped	Average Accuracy	Domain Accuracy
Colored MNIST	89.4%	13.67%	90.03%	31.02%	65.50
MNIST-M	73%	72.3%	30.3050%	50.41%	100%

Table 3: Through this way, source-only training accuracy is about 73%(considering the theoretical maximum is 75%, this CNN model is very promising), direct transferring accuracy is 72.3%, direct training accuracy is 30.3050%, and finally DANN accuracy is 50.41%

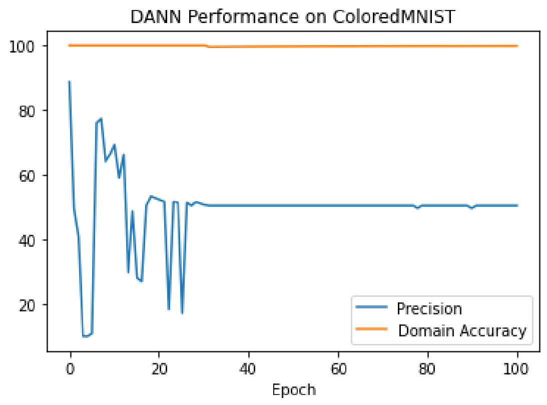

Figure 12: Precision and Domain Accuracy

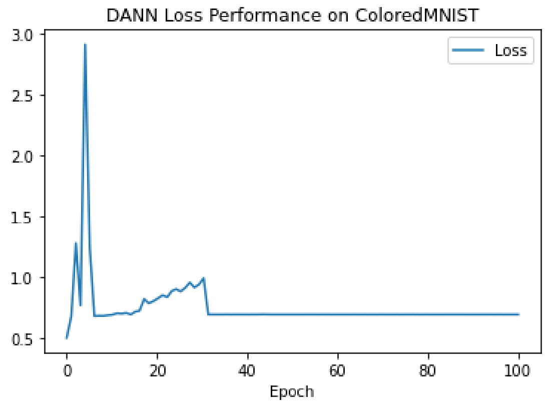

Figure 13: Loss

3.4.3 Analysis

Due to different color-flipping possibilities, simply using train1 and train2 as source domain will inevitably teach the model to learn from the color feature. However, those features are flipped in test dataset and as a result the domain classifier still find it easy to tell data apart. That is the reason why former DANN performs far away from my expectations. After adopting the new training method, feature extractors are led to learn more from the shapes or contours rather than colors, which leads to the latter one performing much better both on accuracy dimension and stability dimension. From this analysis, the main reason that making the model find it hard to find the true causal features still lies in the feature extractor. For further research, we can try to substitute other networks for current 5-layer CNN.

4 Further Exploration

In previous OoD algorithms(VREx [5], ILC[6], IRM[1], Self-Challenging algorithm[7]), their selections of feature extractor are often supervised one. Inspired by PCA, Xiangyuan Xue proposed that we may replace those supervised algorithms with an auto-encoder to realize the same purpose of dimensionality reduction. Related network details can be found in his report. Based on the naive network, I implement the research on which classifier can bring us better

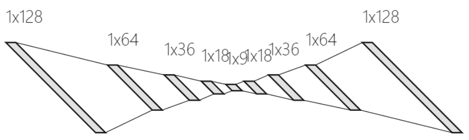

Figure 14: Auto Encoder Model

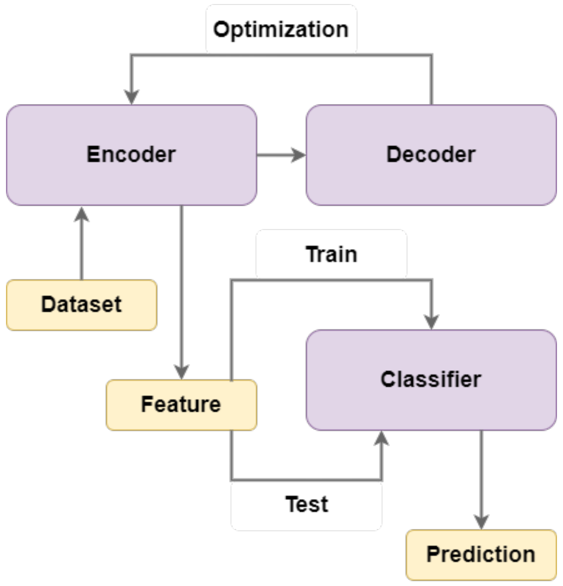

Figure 15: Network

performance and better stability. For comparison, we choose Random Forest (adopted by Xiangyuan Xue), Adaboost and Gradient Boost as the classifiers to be tested on. The main purpose of this selection is trying to find out which **Ensemble Learning** method (Bagging or Boosting) performs better on high error-rate datasets like ColorMNIST. All the classifiers take the same auto encoders as feature extractor (MLP) and train for 50 epochs. Generally, Boosting

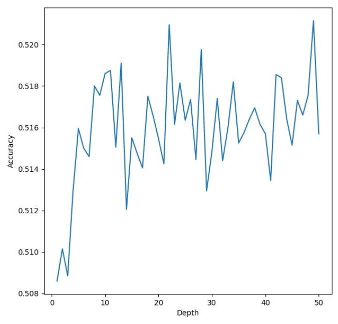

Figure 16: AE + RF

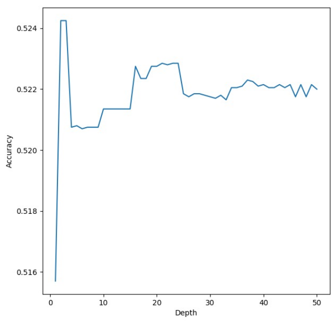

Figure 17: AE + AdaBoost

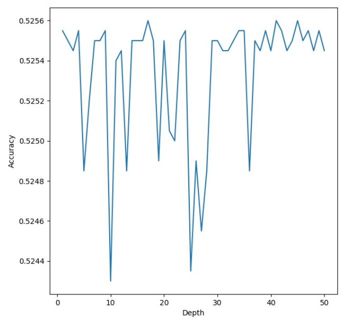

Figure 18: AE + GradientBoost

algorithms perform better than RF whether it's on accuracy dimension or on stability dimension. The difference between Bagging and Boosting can be categorized into 3 parts:

- **Structure** : Different Processing Method: Bagging is parallel computing while Boosting is serial processing.
- **Training Methods**: Every base learner in Bagging algorithm is individually trained while Boosting's base learners are trained on the basis of its previous base learner.
- **Goal**: Bagging aims at reducing the invariance while Boosting aims at reducing bias

Classifier	**Random Forest**	**AdaBoost**	**Gradient Boost**	**Boost Average**
Average Accuracy	51.6%	52.2%	52.50%	52.35%
Peak	52.1%	52.4%	52.56%	50.48%
Valley	51.2%	52.1%	52.44%	52.27%
Stability	$Violent$	$Gentle$	$Steady$	$Steady$
Amplitude	0.9%	0.3%	0.12%	0.21%

Table 4:

For these reasons, Boosting algorithms perform better on high-error-rate dataset like Colored MNIST due to its low bias and its base learners' learning methods. This experience can be applied to future research and applications.

5 Contributions:

On the first stage, I analyzed the features of the provided dataset: Colored MNIST and make a comparison with MNIST to make clear what is causal feature and what is spurious feature. Based on the analysis on dataset, I apply the method used on MNIST — LeNet on the provided dataset and modify the LeNet to suit for the input and output requirements. After acquiring the terrible result of LeNet's baseline, I implement data augmentations on the dataset and successfully elevated the accuracy to another level. For the next stage, I read the original paper of *Domain-adversarial training of neural networks* , explained all the mathematical equations in the paper and succeeded in recurring and modifying the code on provided dataset without using the provided codes by *meta research*. Besides, I think out another training way and constructed MNIST-source to help training the model and have elevated model's performance for another time. For the advanced stage, I research on different ensemble learning algorithms' performance on network which uses unsupervised auto encoder as feature extractor proposed by Xiangyuan Xue , analyzed the reasons, and propose small tactics on high-error-rate dataset for future research or application. What's more, all the assignment is implemented under our both discussions.

6 Innovations:

For LeNet, I implement data augmentation on Colored MNIST without affecting the distribution and the features of the dataset and successfully lead the model to learn more from the numeral's contours and omit the color feature on its own initiative. For DANN, after finding the mainstream or traditional training measure doesn't work, I created a new dataset from ColoredMNIST and lead the model to learn from contours and succeeded in raising the accuracy to a promising level. Based on our respective experience accumulated on previous stage, Xiangyuan introduced usinfg unsupervised auto-encoder as the feature extractor. Based on this network, I conduct empirical learning on different ensemble learning algorithms' performance , analyzed the reasons behind and propose tactics for future high-error-rate dataset research.

Acknowledgments

This assigment is finished by cooperating with Xiangyuan Xue. Also thanks our TA Fan Wu for his kind help.

7 Code Repository

All the source code, plots and report have been uploaded to my Github repository, and I ensure all the data can be recurred by directly running my files under proper environments. https://github.com/BangjunWong/Assignment-for-AI1603-2-2021-2022-3-

References

[1] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

[2] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[4] Cryan. Cryan's experiments on domain adaptation. [EB/OL]. `https://zhuanlan.zhihu.com/p/126185010` Accessed July 10, 2022.

[5] David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex). In *International Conference on Machine Learning*, pages 5815–5826. PMLR, 2021.

[6] Giambattista Parascandolo, Alexander Neitz, Antonio Orvieto, Luigi Gresele, and Bernhard Schölkopf. Learning explanations that are hard to vary. *arXiv preprint arXiv:2009.00329*, 2020.

[7] Zeyi Huang, Haohan Wang, Eric P Xing, and Dong Huang. Self-challenging improves cross-domain generalization. In *European Conference on Computer Vision*, pages 124–140. Springer, 2020.