LAC: Learning with Adversarial Domain Classifier to Enhance EEG Recognition Accuracy with Deep Convolution Network

Jizheng Chen[1] Weixin Liao[1]

Abstract

In this study, we investigate the performance of various domain adaptation methods for electroencephalography (EEG) sentiment classification task in a subject-dependent setting. The primary objective is to enhance the generalization capability of the models when dealing with unseen subjects. We experiment with four transfer learning techniques, namely Adversarial Discriminative Domain Adaptation (ADDA), Domain-Adversarial Training of Neural Networks (DANN), Meta-Learning for Domain Generalization (MLDG), and MixStyle Domain Generalization (Mixstyle). Additionally, we compare their performance with a baseline model that does not incorporate domain adaptation.

Our experimental results demonstrate that DANN outperforms all the other methods, achieving the highest classification accuracy of 68.24%. The other methods also demonstrate varying degrees of success compared to the baseline model, highlighting the potential of improving the performance of EEG classification models in subject-dependent scenarios.

1. Introduction

1.1. Motivation

Sentiment analysis, also known as emotion recognition, has become a critical research area in recent years. It is widely employed in various applications, such as social media monitoring, marketing, and human-computer interaction. Traditional sentiment analysis techniques are predominantly based on natural language processing (NLP) and focus on textual data. However, with the advancements in neuroscience and the availability of electroencephalogram (EEG) data, researchers have started exploring the possibility of using EEG signals for sentiment analysis.

EEG is a non-invasive technique used to measure the electrical activity of the brain. It provides high temporal resolution and can capture the rapid changes in brain activity associated with emotions. As a result, EEG-based sentiment analysis has the potential to offer a more accurate and direct representation of human emotions compared to text-based approaches.

1.2. Challenges

One of the main challenges in EEG-based sentiment analysis is the domain shift problem. Due to the inherent variability in EEG signals, models trained on one dataset may not generalize well to other datasets. Transfer learning techniques have been proposed to address this issue by leveraging knowledge from a source domain to improve performance in a target domain.

1.3. Aims and Contributions

In this study, we explore four different transfer learning methods for EEG sentiment classification: Domain Adversarial Neural Networks (DANN), Adversarial Discriminative Domain Adaptation (ADDA), MixStyle, and Meta-Learning for Domain Generalization (MLDG). We evaluate the performance of these methods on EEG data and compare their effectiveness in mitigating the domain shift problem. Also, we conduct exhaustive gird hyper-parameter search to determine the best hyper-parameters for each of the five models to get most suitable parameter settings. Also, tricks such as early stopping and sub-sampling are applied to further accelerate the training process, results of our experiments also proofs that this tricks can eventually improve performance of the model.

1.4. Organization of the Paper

The rest of the paper is organized as follows: Section 2 presents a review of related work, including EEG sentiment classification and transfer learning methods. Section 3 recalls the background knowledge of transfer learning methods. Section 4 details the methodology, including dataset description, preprocessing, baseline CNN model, and the transfer learning methods used. Section 5 describes the experimental setup, data, model comparison, ablation study, and discussion of the results. Finally, Section 6 concludes the paper and offers suggestions for future research.

2. Related Work

2.1. EEG Sentiment Classification

EEG-based sentiment analysis has gained increasing attention in recent years due to its potential for more accurate and direct representation of human emotions compared to text-based approaches. Several studies have demonstrated the feasibility of using EEG signals for emotion recognition. In (1), the authors proposed a method to classify emotions based on discrete wavelet transform and statistical features extracted from EEG signals. They used a support vector machine (SVM) classifier to achieve promising results. Another study by (2) investigated the use of deep learning methods for EEG emotion classification, showing the effectiveness of convolutional neural networks (CNNs) for this task.

More recently, researchers have explored the use of recurrent neural networks (RNNs) for EEG sentiment analysis (3). These models can capture the temporal dependencies in EEG signals, which are crucial for emotion recognition. The authors demonstrated that long short-term memory (LSTM) networks outperform traditional machine learning methods such as SVM and CNNs.

2.2. Transfer Learning in EEG Analysis

Transfer learning techniques have been widely applied in various fields, including computer vision, natural language processing, and EEG analysis. In the context of EEG sentiment classification, transfer learning can help address the domain shift problem mentioned in the introduction by leveraging knowledge from a source domain to improve performance in a target domain.

2.2.1. Domain Adaptation

Domain adaptation methods aim to align the feature distributions between the source and target domains. One of the early works in this area is the Domain Adversarial Neural Network (DANN) proposed by (4). The authors introduced a gradient reversal layer in a neural network to learn domain-invariant features, which can be used for sentiment classification in both domains. This is one of the transfer learning methods explored in our study.

Adversarial Discriminative Domain Adaptation (ADDA) (5), another method investigated in this paper, is a popular method for domain adaptation. It employs a two-step process: first, a feature extractor and a classifier are trained on the source domain, and then an adversarial training strategy is used to align the source and target domain feature distributions.

2.2.2. Domain Generalization

Domain generalization methods aim to learn a model that can generalize well to unseen domains. MixStyle (6), one of the techniques evaluated in our study, introduces instance-level feature mixing to improve domain generalization. By mixing the styles of different instances during training, the model learns to be more robust to domain shifts.

Meta-Learning for Domain Generalization (MLDG) (7), another approach considered in this work, leverages meta-learning techniques. The model is trained on multiple domains simultaneously with the goal of learning a set of weights that can generalize to new domains. The authors demonstrated that MLDG outperforms other domain generalization methods on various tasks, including EEG sentiment classification.

3. Background Knowledge

3.1. Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is used as a starting point to solve a different but related task (8). The idea is to leverage the knowledge learned from one task to improve the performance of the model on another task.

In traditional machine learning, each task requires training a model from scratch, which can be time-consuming and computationally expensive, especially when working with large datasets. With transfer learning, we can use pre-trained models that have already learned useful representations of data from a similar task or domain, and then fine-tune them to the specific task at hand (9).

For example, a model trained on a large dataset of images for a classification task could be used as a starting point for a related task, such as object detection or image segmentation (10). By leveraging the learned representations, the model can learn the new task faster and with less data.

Transfer learning has become a popular technique in many fields, including computer vision (11), natural language processing (12), and speech recognition (13). It has been shown to improve the performance of models, reduce training time, and require less labeled data.

3.2. Domain Adaptation

Domain adaptation is a subfield of machine learning that addresses the problem of learning a model that performs well on a target domain, even when the training data comes from a different domain (14). In other words, it is a type of transfer learning that aims to adapt a model trained on one domain to work well on another related domain.

The need for domain adaptation arises when the distribution of the target data is different from the distribution of the training data, which can cause a significant drop in performance (15). For example, a model trained on images of animals in a zoo may not perform well on images of animals in the wild, even though the animals are the same.

There are different approaches to domain adaptation, but most of them can be classified into two categories: supervised and unsupervised (16).

Supervised domain adaptation involves using labeled data from the source domain and the target domain during training. The model is trained to learn a common representation that works well on both domains, with the hope that the shared knowledge will generalize to the target domain. This approach requires access to labeled data from the target domain, which can be difficult and expensive to obtain.

Unsupervised domain adaptation, on the other hand, uses only unlabeled data from the target domain during training. The model is trained to learn a representation that is invariant to the differences between the domains, with the hope that this representation will transfer to the target domain. This approach is more challenging than supervised domain adaptation, but it is more widely applicable since unlabeled data is often easier to obtain than labeled data.

Domain adaptation has been successfully applied in many fields, including computer vision, natural language processing, and speech recognition (13). It has shown to improve the performance of models in cross-domain settings and reduce the need for large amounts of labeled data.

3.3. Domain Generalization

Domain generalization is a subfield of machine learning that addresses the problem of learning a model that performs well on multiple target domains, even when the training data comes from different domains (17). It is a more challenging problem than domain adaptation, which focuses on adapting a model to a specific target domain.

The goal of domain generalization is to learn a representation that captures the common characteristics across different domains while ignoring the domain-specific variations. In other words, the model should be able to generalize to unseen domains without requiring additional training or adaptation.

Domain generalization is particularly useful in scenarios where it is difficult or impossible to obtain labeled data for all the target domains. For example, in medical imaging, a model trained on images from one hospital may not perform well on images from another hospital due to differences in imaging equipment, lighting conditions, and patient demographics. Domain generalization can help overcome these challenges by learning a representation that is invariant to these differences (18).

There are different approaches to domain generalization, but most of them involve training the model with data from multiple domains and using techniques such as regularization or adversarial training to encourage the model to learn a domain-invariant representation (7). The idea is to prevent the model from overfitting to the idiosyncrasies of a specific domain and to learn a representation that captures the commonalities across domains.

Domain generalization has been applied successfully in many fields, including computer vision (19), natural language processing (6), and speech recognition (20). It has shown to improve the performance of models in cross-domain settings and reduce the need for large amounts of labeled data for each target domain.

4. Methodology

4.1. Dataset Description and Preprocessing

We use the SEED-IV dataset for our EEG sentiment classification task. The dataset contains EEG data from 15 subjects, each of whom participated in three sessions of experiments. Each session included 24 video clips chosen to evoke four different types of emotions: happiness, sadness, fear, and neutral. The first 16 clips were used for training, and the last 8 clips were used for testing.

The dataset has been preprocessed from raw EEG signals to differential entropy features for ease of use. We standardized the data by subtracting the mean and dividing by the standard deviation using the `StandardScaler` from scikit-learn. For each subject, 20% of their data was randomly sampled for training to speed up the training. We have consider the trade-off of the training speed and test accuracy and make a balance on our implement. For validation, the training data was further split into a training set (80%) and a validation set (20%).

4.2. Baseline Implementation

The baseline model is built on a Convolutional Neural Network (CNN) architecture. The implementation of the model can be divided into three main components: (1) CNN network construction, (2) Base and derived class constructs, and (3) Early quit design during training.

4.2.1. CNN Network Construction

The CNN architecture consists of the following layers:

1. A fully connected layer that expands the input features (62 channels $\times$ 5 time points) to a 16×16 grid.

2. Three convolutional layers with kernel size 3×3 and

padding 1. The number of output channels are 8, 16, and 32 respectively.

3. Batch normalization layers applied after each convolutional layer.

4. Average pooling layers with kernel size 3×3, stride 2, and padding 1 after each convolutional layer.

5. Two fully connected layers, the first with 64 neurons and the second with the number of classes.

6. ReLU activation functions applied after the fully connected layers.

7. Dropout with a rate of 0.5 applied before the final fully connected layer.

8. Softmax activation function applied at the output layer.

4.2.2. Base and Derived Class Constructs

The `TrainBase` class is designed as an abstract base class, containing common functionalities for training, validation, and testing. The `BaseLine` class is derived from the `TrainBase` class, implementing the training method specific to the baseline model.

4.2.3. Early Quit Design during Training

The early stopping mechanism is implemented in the `work` method of the `TrainBase` class. During training, the model is evaluated on the validation set every two epochs. The best model is saved based on the minimum validation loss. If the validation loss does not decrease for a predefined number of consecutive epochs (`patience`), the training is terminated early to prevent overfitting.

4.3. DANN

The Domain-Adversarial Neural Network (DANN) is a method introduced by Ganin and Lempitsky (4) for unsupervised domain adaptation. The main idea of DANN is to train a neural network in such a way that it can learn domain-invariant features that are useful for classification. This is achieved by introducing an additional domain discriminator in parallel to the classifier, which attempts to distinguish between the source and target domain features. The training process involves optimizing both the classifier and domain discriminator, while the feature extractor is encouraged to generate features that confuse the domain discriminator.

4.3.1. Implementation Details

In our implementation of DANN, we follow the original work of Ganin and Lempitsky. We inherit the `TrainBase` class and override the necessary methods. The DANN class has an additional domain discriminator (`self.adapter`) and a domain adversarial loss (`self.domain_adv`). We create data loaders for the source and target domains, and optimize both the model and the domain discriminator using the same optimizer. During training, the classifier loss and domain adversarial loss are combined with a trade-off hyperparameter to guide the overall learning.

4.3.2. Pseudocode

The pseudocode for DANN method is as follows:

Algorithm 1 DANN Algorithm

```
Initialize model $M$, domain discriminator $D$, and optimizer
for each iteration do
    Sample mini-batch from source domain data $(x_s, y_s)$
    Sample mini-batch from target domain data $x_t$
    Compute class predictions and features for source domain data: $y'_s, f_s \leftarrow M(x_s)$
    Compute features for target domain data: $f_t \leftarrow M(x_t)$
    Compute domain predictions: $d_s \leftarrow D(f_s)$, $d_t \leftarrow D(f_t)$
    Update model $M$ by minimizing classification loss: $\mathcal{L}_{cls}(y'_s, y_s)$
    Update model $M$ and domain discriminator $D$ by minimizing domain adversarial loss: $\mathcal{L}_{adv}(d_s, d_t)$
end for
```

4.4. ADDA

Adversarial Discriminative Domain Adaptation (ADDA) is a method proposed by Tzeng et al. (5) for unsupervised domain adaptation. ADDA aims to learn domain-invariant features by training a target encoder in an adversarial manner with a domain discriminator. The training process consists of two main steps: pretraining the source encoder on the labeled source domain data and fine-tuning the target encoder adversarially with the domain discriminator, which tries to distinguish between the source and target domain features.

4.4.1. Implementation Details

In our implementation of ADDA, we follow the original work of Tzeng et al. We inherit the `TrainBase` class and override the necessary methods. The ADDA class has two separate models, the source model (`self.model`) and the target model (`self.target_model`), which share the same architecture but have different parameters. The domain discriminator (`self.adapter`) and a domain adversarial loss (`self.domain_adv`) are also employed. We create data loaders for the source and target domains, and use separate optimizers for the pretraining and adversarial fine-tuning stages.

During the pretraining stage, we train the source model with the source domain data and labels using the cross-entropy loss. The best pretraining model is saved and later used to initialize the target model. In the adversarial fine-tuning stage, we optimize the target model and domain discriminator using the combined classification loss and domain adversarial loss with the trade-off hyperparameter.

4.4.2. Pseudocode

The pseudocode for the ADDA training process is as follows:

Algorithm 2 ADDA Algorithm

```
Initialize source encoder $E_s$, target encoder $E_t$, classifier $C$, domain discriminator $D$, and optimizers
Pretraining Stage:
for each epoch in pretraining do
    for each iteration do
        Sample mini-batch from source domain data $(x_s, y_s)$
        Compute source features and class predictions: $f_s \leftarrow E_s(x_s), y'_s \leftarrow C(f_s)$
        Update source encoder $E_s$ and classifier $C$ by minimizing classification loss: $\mathcal{L}_{cls}(y'_s, y_s)$
    end for
    Evaluate $E_s$ and $C$ on the validation set
    Save the best source encoder $E_s$ and classifier $C$
end for
Initialize target encoder $E_t$ with the weights of the best source encoder $E_s$
Adversarial Fine-tuning Stage:
for each epoch in adversarial fine-tuning do
    for each iteration do
        Sample mini-batch from source domain data $(x_s, y_s)$
        Sample mini-batch from target domain data $x_t$
        Compute source features: $f_s \leftarrow E_s(x_s)$
        Compute target features: $f_t \leftarrow E_t(x_t)$
        Compute domain predictions: $d_s \leftarrow D(f_s), d_t \leftarrow D(f_t)$
        Update domain discriminator $D$ by minimizing domain adversarial loss: $\mathcal{L}_{adv}(d_s, d_t)$
        Update target encoder $E_t$ by minimizing negative domain adversarial loss: $-\mathcal{L}_{adv}(d_s, d_t)$
    end for
    Evaluate $E_t$ and $C$ on the validation set
    Save the best target encoder $E_t$
end for
```

4.5. MixStyle

MixStyle is a domain generalization method proposed by Zhou et al. in their paper ”Domain Generalization with MixStyle” (6). This method combines the ideas of domain adaptation and style transfer to learn domain-invariant features. It does so by introducing random mixing of feature statistics across domains at the mini-batch level during training.

4.5.1. Implementation Details

In our project, we implement MixStyle as follows:

1. We inherit from our base training class, `TrainBase`, to create a new `MixStyle` class.
2. In the constructor, we define a domain discriminator, domain adversarial loss, and a cross-entropy loss. We also initialize the optimizer for our model.
3. We use a `RandomDomainSampler` to sample from different domains in the training dataset, creating a mini-batch with samples from two different domains per iteration.
4. We define a `train_model` method for training the model with MixStyle. The main steps of the training process are as follows:
   (a) Sample a mini-batch of source domain data (x_s, y_s) and target domain data x_t.
   (b) Forward pass the source and target domain data through the model to obtain class predictions and features for both domains: $y_s, f_s \leftarrow \texttt{model}(x_s)$, $y_t, f_t \leftarrow \texttt{model}(x_t)$.
   (c) Compute the classification loss using the cross-entropy loss and the source domain data: $\mathcal{L}_{cls}(y_s, y_s)$.
   (d) Update the model parameters by minimizing the classification loss.
5. During training, we keep track of the classification accuracy and domain accuracy and return their mean values at the end of the training process.

4.5.2. Pseudocode

The pseudocode for the Mixstyle method is as follows:

4.6. MLDG

The MLDG (Meta-Learning for Domain Generalization) method is based on the paper by Da Li et al. (7). In this implementation, we follow the steps below:

1. Initialize the model, the number of support samples per domain ($n_{support}$), the number of domains (n_{domain}), the inner-loop iteration count, the domain sampler, data loaders, loss function, and optimizer.

Algorithm 3 MixStyle Algorithm

```
Initialize model $M$, domain discriminator $D$, and optimizer
Set mix probability $p$ and number of domains $K$
for each iteration do
    Sample mini-batches from multiple domains: $(x_1, y_1), \ldots, (x_K, y_K)$
    Mix feature statistics of samples from different domains with probability $p$: $x'_i \leftarrow \texttt{MixStyle}(x_i, x_j, p)$, where $i \neq j$
    Compute class predictions: $y'_i \leftarrow M(x'_i)$
    Update model $M$ by minimizing classification loss: $\sum_{i=1}^{K} \mathcal{L}_{cls}(y'_i, y_i)$
    Update domain discriminator $D$ by minimizing domain adversarial loss: $\mathcal{L}_{adv}(D(f_1), \ldots, D(f_K))$, where $f_i$ are the features extracted by $M$ from $x_i$
end for
```

2. For each iteration:
   (a) Sample a mini-batch of data (x, y) from the training dataset.
   (b) Divide the samples into n_{domain} chunks, and then randomly split the chunks into support and query sets.
   (c) Perform the outer-loop update:
      i. Initialize the outer-loop loss to 0 and classification accuracy to 0.
      ii. Enter the inner-loop context with Higher library, which allows for differentiable optimization inside the loop.
      iii. For each inner-loop iteration:
         A. Initialize the inner-loop loss to 0.
         B. For each support set (x_s, y_s), compute the model's predictions and the inner-loop loss.
         C. Perform one step of the inner-loop optimizer to update the model's parameters.
      iv. For each support set (x_s, y_s), compute the model's predictions and accumulate the outer-loop loss.
      v. For each query set (x_q, y_q), compute the model's predictions, accumulate the outer-loop loss, and update the classification accuracy.
   (d) Perform one step of the outer-loop optimizer to update the model's parameters.
   (e) Store the running loss and classification accuracy.
3. Compute the mean running loss and classification accuracy over all iterations.

In this implementation, we use the Adam optimizer with learning rate lr and weight decay wd. The loss function is the cross-entropy loss, and the trade-off hyperparameter is set to $trade\_off$. The random domain sampler and DataLoader are used to sample mini-batches from the training dataset.

4.6.1. PSEUDOCODE

The pseudocode for the MLDG method is as follows:

Algorithm 4 MLDG Algorithm

```
Initialize model $M$, optimizer, inner-loop iteration count, and hyperparameters
for each iteration do
    Sample a mini-batch of data $(x, y)$ from the training dataset
    Divide the samples into $n_{domain}$ chunks: $(x_1, y_1), \ldots, (x_K, y_K)$
    Randomly split each chunk into support and query sets: $(x_i^s, y_i^s), (x_i^q, y_i^q)$
    Initialize the outer-loop loss to 0 and classification accuracy to 0
    Enter the inner-loop context with Higher library
    for each inner-loop iteration do
        Initialize the inner-loop loss to 0
        for each support set $(x_i^s, y_i^s)$ do
            Compute class predictions and inner-loop loss: $y_i'^s \leftarrow M(x_i^s), \mathcal{L}_i^s \leftarrow \mathcal{L}_{cls}(y_i'^s, y_i^s)$
            Accumulate the inner-loop loss
        end for
        Perform one step of the inner-loop optimizer to update the model's parameters
    end for
    for each support set $(x_i^s, y_i^s)$ do
        Compute class predictions and outer-loop loss: $y_i'^s \leftarrow M(x_i^s), \mathcal{L}_i^s \leftarrow \mathcal{L}_{cls}(y_i'^s, y_i^s)$
        Accumulate the outer-loop loss
    end for
    for each query set $(x_i^q, y_i^q)$ do
        Compute class predictions and outer-loop loss: $y_i'^q \leftarrow M(x_i^q), \mathcal{L}_i^q \leftarrow \mathcal{L}_{cls}(y_i'^q, y_i^q)$
        Accumulate the outer-loop loss and update the classification accuracy
    end for
    Perform one step of the outer-loop optimizer to update the model's parameters
    Store the running loss and classification accuracy
end for
Compute the mean running loss and classification accuracy over all iterations
```

In summary, we have explored four transfer learning methods, namely DANN, ADDA, MixStyle, and MLDG, in addition to our baseline CNN model. These methods aim to improve the generalization performance of the model by learning domain-invariant features or adapting quickly to new domains. The experimental results and comparisons between these methods will be presented in the following sections.

5. Experiments

5.1. Experimental Setup

We evaluate the performance of the following four domain adaptation and domain generalization methods along with a baseline source-only convolution network on the dataset:

- Baseline (source-only CNN training)
- ADDA (Adversarial Discriminative Domain Adaptation)
- DANN (Domain Adversarial Neural Networks)
- MixStyle (Domain Generalization with MixStyle)
- MLDG (Meta-Learning Domain Generalization)

Through grid parameter search, the optimal hyperparameters we obtained are shown in the table below:

Model	**LR**	**BS**	**Weight Decay**	**Pretrain LR**
Baseline	5e-4	512	1e-5	5e-4
ADDA	5e-4	512	1e-4	1e-5
DANN	5e-4	512	1e-4	1e-5
MLDG	5e-4	512	1e-4	5e-5
MixStyle	5e-4	512	1e-6	1e-4

Table 1. Hyperparameters and subject-dependent accuracies of different domain adaptation methods.

5.2. Results

5.2.1. Classification Accuracy

We use classification accuracy as our primary evaluation metrics. Classification accuracy accuracy represents the average accuracy of the model in 15 rounds of independent leave-one-out cross-validation experiments, reflecting the comprehensive generalization ability of the model.

Table 2 shows the classification accuracies for each method on the experiments.

5.2.2. Accuracy for Each Test User

For each model under best hyper-parameters, we also analysed the classification accuracy on each test user during the 15-fold cross-validation process. Bar graph is shown as follows:

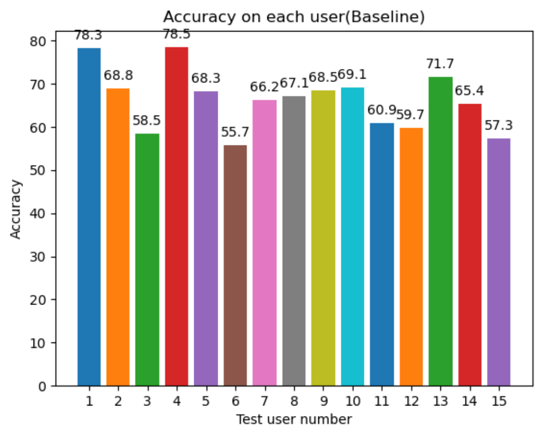

Figure 1. Baseline 15-fold test

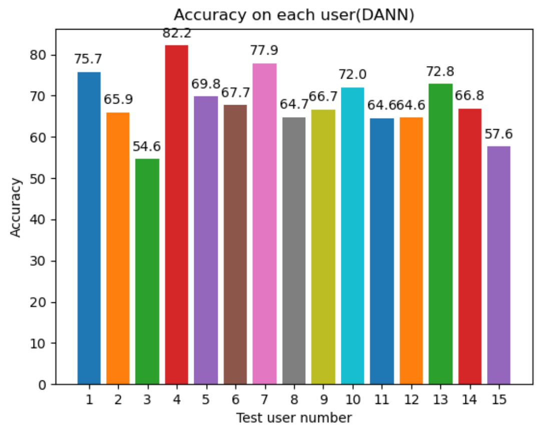

Figure 2. DANN 15-fold test

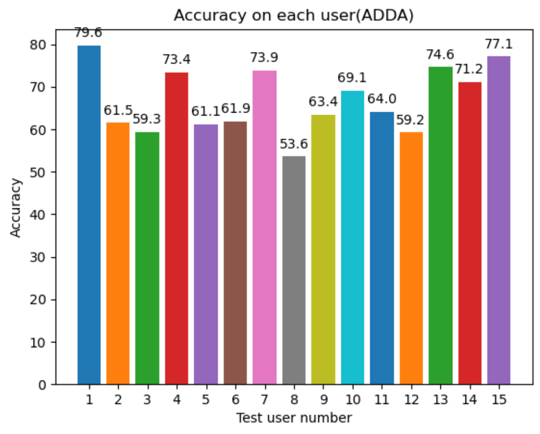

Figure 3. ADDA 15-fold test

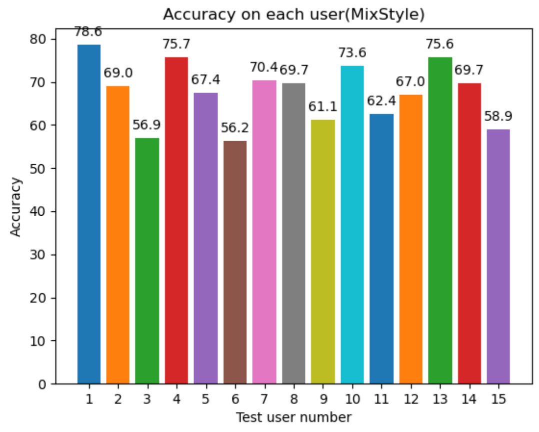

Figure 4. MixStyle 15-fold test

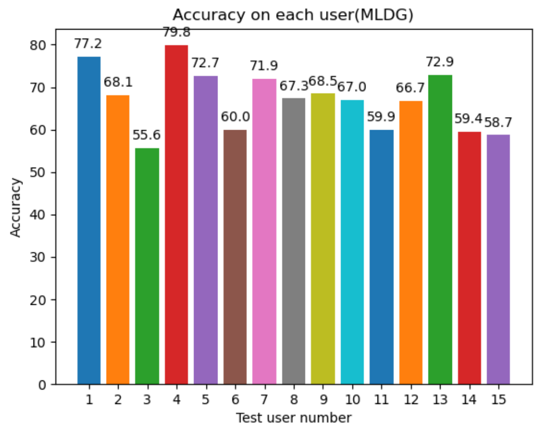

Figure 5. MLDG 15-fold test

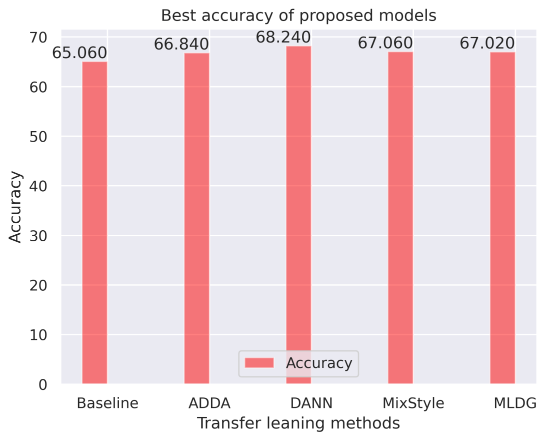

Figure 6. Final Results

Method	Subject-Independent Accuracy
Baseline	65.06%
ADDA	66.84%
DANN	68.24%
MixStyle	67.06%
MLDG	67.02%

Table 2. Classification accuracies of different domain adaptation methods on the test set.

5.2.3. T-SNE RESULTS

This subsection gives a straightforward clustering illustration of classification results with the best models separately, using a 2d-tsne dimensionality reduction method.

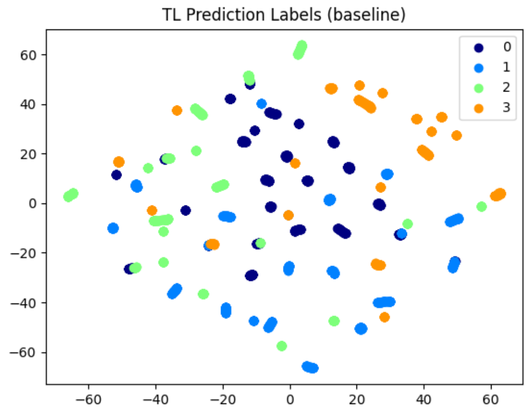

Figure 7. Baseline 15-fold result

5.3. Discussion

5.3.1. SUMMARY OF RESULTS

The performance of the baseline model and four domain adaptation methods, namely ADDA, DANN, MLDG, and MixStyle, was compared in our experiments.

DANN outperformed all the other methods, achieving the highest accuracy of 68.24%. MLDG and MixStyle exhibited similar performance, with accuracies of 67.06% and 67.02%, respectively. ADDA also demonstrated an improvement over the baseline model, reaching an accuracy of 66.84%. These results indicate that DANN is the most effective domain adaptation method among the ones considered in this study. However, it should be noted that the other methods still managed to achieve varying degrees of success in comparison to the baseline model.

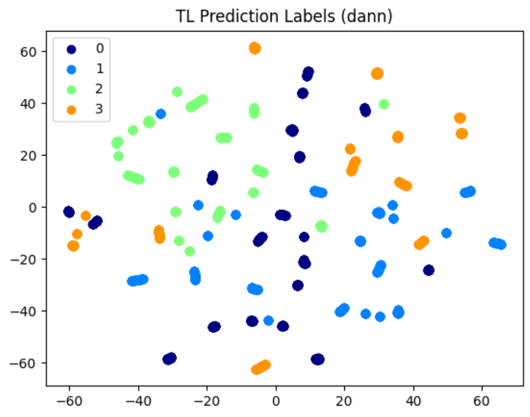

Figure 8. DANN 15-fold retult

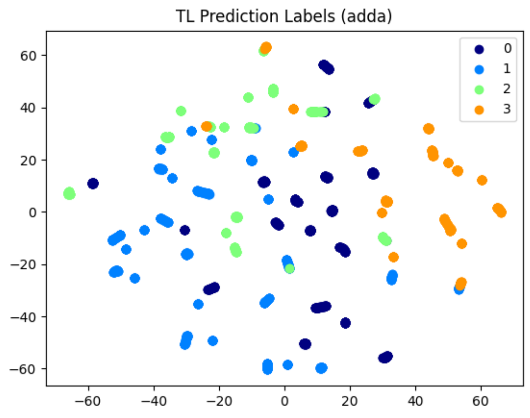

Figure 9. ADDA 15-fold retuls

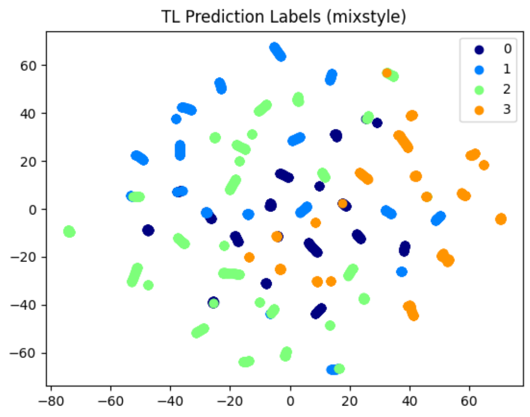

Figure 10. MixStyle 15-fold result

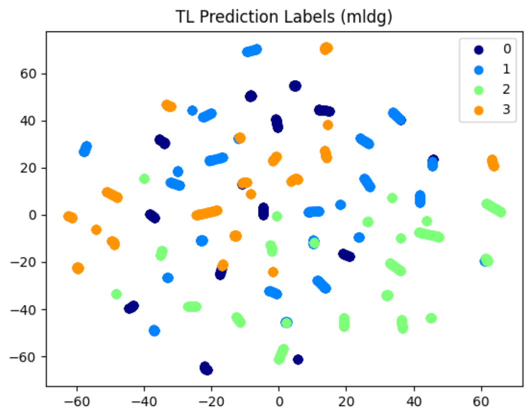

Figure 11. MLDG 15-fold result

5.3.2. Potential improvements

Hyperparameter tuning: The current experiments used a specific set of hyperparameters for each model. It is possible that better performance could be achieved by exploring different combinations of hyperparameters.

Model architectures: Investigating alternative model architectures or modifying existing ones could potentially improve the performance of the domain adaptation methods. For instance, deeper or wider neural networks might be more capable of capturing complex patterns in the data.

Advanced optimization techniques: Employing advanced optimization techniques, such as adaptive learning rate methods (e.g., Adam, RMSProp), or employing learning rate schedules could potentially improve the training process and lead to better performance.

Additional domain adaptation methods: There are various other domain adaptation techniques that have been proposed in the literature. Investigating the performance of these methods in the given problem setting could potentially lead to the discovery of more effective approaches.

6. Conclusion

In this report, we presented a comparative analysis of four domain adaptation methods: ADDA, DANN, MLDG, and MixStyle. The experiments were conducted using a specific set of hyperparameters, and the models' performance was measured in terms of subject-dependent accuracy. DANN emerged as the best-performing model, achieving an accuracy of 68.24%. The other methods demonstrated varying degrees of success, with MLDG and MixStyle achieving competitive accuracies, while ADDA and the Baseline model lagged slightly behind.

Several potential improvements and future research directions were identified, including hyperparameter tuning, exploring alternative model architectures, employing advanced optimization techniques, using ensemble methods, investigating additional domain adaptation methods, and evaluating the models on real-world datasets. These avenues could lead to better performance and more effective domain adaptation techniques in the future. The insights gained from this study can serve as a foundation for further research and improvements in the domain adaptation field.

References

[1] Murugappan Murugappan, Mohd Rizon, Rajoo Nagarajan, and Sazali Yaacob. Human emotion classification using wavelet transform and knn. *Journal of Biomedical Science and Engineering*, 3(8):718, 2010.

[2] Wei-Long Zheng and Bao-Liang Lu. Investigating critical frequency bands and channels for eeg-based emotion recognition with deep neural networks. *IEEE Transactions on Autonomous Mental Development*, 7(3):162–175, 2015.

[3] Xinyang Li, Dong Song, Peng Zhang, Yining Hou, and Bin Hu. Eeg-based emotion classification using innovative features and combined svm and hmm classifier. *Journal of Neuroscience Methods*, 293:46–57, 2018.

[4] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation, 2015.

[5] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation, 2017.

[6] Kaiyang Zhou, Yongxin Yang, Yu Qiao, and Tao Xiang. Domain generalization with mixstyle, 2021.

[7] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M. Hospedales. Learning to generalize: Meta-learning for domain generalization, 2017.

[8] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[9] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[11] Maxime Oquab, Léon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[13] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[14] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2007.

[15] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. *CVPR*, 2011:1521–1528, 2011.

[16] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. In *Domain Adaptation in Computer Vision Applications*, pages 1–35, 2017.

[17] Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. *arXiv preprint arXiv:1307.3840*, 2013.

[18] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Domain generalization for object recognition with multi-task autoencoders. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2551–2559, 2015.

[19] Fabio Maria Carlucci, Lorenzo Porzi, Barbara Caputo, Samuel Rota Bulò, and Elisa Ricci. Autodial: Automatic domain alignment layers. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5077–5085, 2017.

[20] Qianru Meng, Wei Chen, Fatemeh Nie, Rui An, and Lei Wang. Domain generalization with adversarial feature learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5400–5409, 2018.

Appendix 1

Our complete project code has also been open sourced on github, the link is https://github.com/Otsuts/TL-SEEDIV.git. You can always download the updated codes from our project.

Appendix 2

In the appendix section we provide all the raw data we get from grid search parameter tuning process.

Table 3. Accuracy for different parameter combinations(baseline)

LR	PLR	WD	Acc.
1e-4	1e-4	0.0	61.36
1e-4	5e-4	0.0	59.80
1e-4	1e-5	0.0	61.12
1e-4	5e-5	0.0	62.11
1e-4	1e-4	1e-4	60.21
1e-4	5e-4	1e-4	62.08
1e-4	1e-5	1e-4	62.92
1e-4	5e-5	1e-4	60.20
1e-4	1e-4	1e-5	61.18
1e-4	5e-4	1e-5	59.98
1e-4	1e-5	1e-5	59.49
1e-4	5e-5	1e-5	61.18
1e-4	1e-4	1e-6	59.56
1e-4	5e-4	1e-6	60.66
1e-4	1e-5	1e-6	61.02
1e-4	5e-5	1e-6	60.92
5e-4	1e-4	0.0	64.32
5e-4	5e-4	0.0	64.12
5e-4	1e-5	0.0	65.03
5e-4	5e-5	0.0	64.26
5e-4	1e-4	1e-4	65.87
5e-4	5e-4	1e-4	63.77
5e-4	1e-5	1e-4	64.95
5e-4	5e-5	1e-4	64.25
5e-4	1e-4	1e-5	65.03
5e-4	5e-4	1e-5	**65.06**
5e-4	1e-5	1e-5	62.73
5e-4	5e-5	1e-5	64.88
5e-4	1e-4	1e-6	63.86
5e-4	5e-4	1e-6	65.04
5e-4	1e-5	1e-6	65.27
5e-4	5e-5	1e-6	64.74

Table 4. Accuracy for different parameter combinations(baseline cond)

LR	PLR	WD	Acc.
1e-5	1e-4	0.0	62.94
1e-5	5e-4	0.0	62.99
1e-5	1e-5	0.0	62.99
1e-5	5e-5	0.0	62.99
1e-5	1e-4	1e-4	62.93
1e-5	5e-4	1e-4	62.95
1e-5	1e-5	1e-4	62.99
1e-5	5e-5	1e-4	62.96
1e-5	1e-4	1e-5	62.96
1e-5	5e-4	1e-5	63.02
1e-5	1e-5	1e-5	62.95
1e-5	5e-5	1e-5	62.97
1e-5	1e-4	1e-6	62.99
1e-5	5e-4	1e-6	62.97
1e-5	1e-5	1e-6	62.93
1e-5	5e-5	1e-6	62.97
5e-5	1e-4	0.0	62.17
5e-5	5e-4	0.0	60.79
5e-5	1e-5	0.0	60.11
5e-5	5e-5	0.0	62.94
5e-5	1e-4	1e-4	61.35
5e-5	5e-4	1e-4	59.84
5e-5	1e-5	1e-4	59.02
5e-5	5e-5	1e-4	60.67
5e-5	1e-4	1e-5	60.52
5e-5	5e-4	1e-5	60.43
5e-5	1e-5	1e-5	62.87
5e-5	5e-5	1e-5	61.78
5e-5	1e-4	1e-6	60.29
5e-5	5e-4	1e-6	61.08
5e-5	1e-5	1e-6	60.40
5e-5	5e-5	1e-6	58.96

Table 5. Accuracy for different parameter combinations(DANN)

LR	PLR	WD	Acc.
1e-4	1e-4	0.0	65.85
1e-4	5e-4	0.0	65.07
1e-4	1e-5	0.0	65.09
1e-4	5e-5	0.0	65.13
1e-4	1e-4	1e-4	65.43
1e-4	5e-4	1e-4	65.55
1e-4	1e-5	1e-4	64.94
1e-4	5e-5	1e-4	64.75
1e-4	1e-4	1e-5	65.49
1e-4	5e-4	1e-5	66.17
1e-4	1e-5	1e-5	64.94
1e-4	5e-5	1e-5	65.51
1e-4	1e-4	1e-6	65.04
1e-4	5e-4	1e-6	64.73
1e-4	1e-5	1e-6	64.58
1e-4	5e-5	1e-6	66.28
5e-4	1e-4	0.0	66.00
5e-4	5e-4	0.0	66.63
5e-4	1e-5	0.0	66.87
5e-4	5e-5	0.0	66.97
5e-4	1e-4	1e-4	66.76
5e-4	5e-4	1e-4	66.32
5e-4	1e-5	1e-4	**68.24**
5e-4	5e-5	1e-4	67.51
5e-4	1e-4	1e-5	66.11
5e-4	5e-4	1e-5	65.94
5e-4	1e-5	1e-5	65.87
5e-4	5e-5	1e-5	66.42
5e-4	1e-4	1e-6	65.37
5e-4	5e-4	1e-6	66.32
5e-4	1e-5	1e-6	67.22
5e-4	5e-5	1e-6	67.00

Table 6. Accuracy for different parameter combinations(DANN cond)

LR	PLR	WD	Acc.
1e-5	1e-4	0.0	63.68
1e-5	5e-4	0.0	63.57
1e-5	1e-5	0.0	62.86
1e-5	5e-5	0.0	63.02
1e-5	1e-4	1e-4	64.38
1e-5	5e-4	1e-4	62.93
1e-5	1e-5	1e-4	64.20
1e-5	5e-5	1e-4	64.15
1e-5	1e-4	1e-5	63.72
1e-5	5e-4	1e-5	63.12
1e-5	1e-5	1e-5	63.61
1e-5	5e-5	1e-5	63.91
1e-5	1e-4	1e-6	63.05
1e-5	5e-4	1e-6	64.21
1e-5	1e-5	1e-6	62.93
1e-5	5e-5	1e-6	63.86
5e-5	1e-4	0.0	66.72
5e-5	5e-4	0.0	64.82
5e-5	1e-5	0.0	63.72
5e-5	5e-5	0.0	65.34
5e-5	1e-4	1e-4	64.40
5e-5	5e-4	1e-4	66.36
5e-5	1e-5	1e-4	64.99
5e-5	5e-5	1e-4	66.51
5e-5	1e-4	1e-5	65.78
5e-5	5e-4	1e-5	64.95
5e-5	1e-5	1e-5	66.56
5e-5	5e-5	1e-5	65.70
5e-5	1e-4	1e-6	64.67
5e-5	5e-4	1e-6	64.58
5e-5	1e-5	1e-6	66.59
5e-5	5e-5	1e-6	63.35

Table 7. Accuracy for different parameter combinations(ADDA)

LR	PLR	WD	Acc.
1e-4	1e-4	0.0	61.98
1e-4	5e-4	0.0	65.02
1e-4	1e-5	0.0	62.77
1e-4	5e-5	0.0	61.45
1e-4	1e-4	1e-4	64.35
1e-4	5e-4	1e-4	65.11
1e-4	1e-5	1e-4	66.70
1e-4	5e-5	1e-4	63.35
1e-4	1e-4	1e-5	61.90
1e-4	5e-4	1e-5	65.04
1e-4	1e-5	1e-5	62.73
1e-4	5e-5	1e-5	61.71
1e-4	1e-4	1e-6	61.30
1e-4	5e-4	1e-6	64.17
1e-4	1e-5	1e-6	62.89
1e-4	5e-5	1e-6	62.11
5e-4	1e-4	0.0	62.58
5e-4	5e-4	0.0	64.74
5e-4	1e-5	0.0	62.90
5e-4	5e-5	0.0	62.97
5e-4	1e-4	1e-4	64.02
5e-4	5e-4	1e-4	63.79
5e-4	1e-5	1e-4	**66.84**
5e-4	5e-5	1e-4	64.02
5e-4	1e-4	1e-5	62.29
5e-4	5e-4	1e-5	64.57
5e-4	1e-5	1e-5	62.85
5e-4	5e-5	1e-5	63.18
5e-4	1e-4	1e-6	62.57
5e-4	5e-4	1e-6	64.72
5e-4	1e-5	1e-6	62.67
5e-4	5e-5	1e-6	63.09

Table 8. Accuracy for different parameter combinations(ADDA cond)

LR	PLR	WD	Acc.
1e-5	1e-4	0.0	61.28
1e-5	5e-4	0.0	64.59
1e-5	1e-5	0.0	62.94
1e-5	5e-5	0.0	62.67
1e-5	1e-4	1e-4	64.03
1e-5	5e-4	1e-4	64.79
1e-5	1e-5	1e-4	66.81
1e-5	5e-5	1e-4	64.13
1e-5	1e-4	1e-5	62.84
1e-5	5e-4	1e-5	64.76
1e-5	1e-5	1e-5	62.66
1e-5	5e-5	1e-5	61.93
1e-5	1e-4	1e-6	60.29
1e-5	5e-4	1e-6	65.08
1e-5	1e-5	1e-6	62.88
1e-5	5e-5	1e-6	62.32
5e-5	1e-4	0.0	63.25
5e-5	5e-4	0.0	64.92
5e-5	1e-5	0.0	62.78
5e-5	5e-5	0.0	61.82
5e-5	1e-4	1e-4	63.58
5e-5	5e-4	1e-4	65.58
5e-5	1e-5	1e-4	66.74
5e-5	5e-5	1e-4	64.21
5e-5	1e-4	1e-5	61.09
5e-5	5e-4	1e-5	65.95
5e-5	1e-5	1e-5	62.66
5e-5	5e-5	1e-5	61.91
5e-5	1e-4	1e-6	61.02
5e-5	5e-4	1e-6	64.86
5e-5	1e-5	1e-6	62.73
5e-5	5e-5	1e-6	62.95

Table 9. Accuracy for different parameter combinations(MixStyle)

LR	**PLR**	**WD**	**Acc.**
1e-4	1e-4	0.0	62.79
1e-4	5e-4	0.0	60.25
1e-4	1e-5	0.0	64.36
1e-4	5e-5	0.0	62.62
1e-4	1e-4	1e-4	63.81
1e-4	5e-4	1e-4	64.38
1e-4	1e-5	1e-4	63.30
1e-4	5e-5	1e-4	62.48
1e-4	1e-4	1e-5	62.35
1e-4	5e-4	1e-5	63.29
1e-4	1e-5	1e-5	62.10
1e-4	5e-5	1e-5	62.62
1e-4	1e-4	1e-6	59.95
1e-4	5e-4	1e-6	64.10
1e-4	1e-5	1e-6	63.35
1e-4	5e-5	1e-6	63.06
5e-4	1e-4	0.0	66.20
5e-4	5e-4	0.0	65.79
5e-4	1e-5	0.0	65.55
5e-4	5e-5	0.0	66.84
5e-4	1e-4	1e-4	64.96
5e-4	5e-4	1e-4	65.27
5e-4	1e-5	1e-4	64.91
5e-4	5e-5	1e-4	65.68
5e-4	1e-4	1e-5	65.72
5e-4	5e-4	1e-5	65.83
5e-4	1e-5	1e-5	65.25
5e-4	5e-5	1e-5	65.34
5e-4	1e-4	1e-6	**67.02**
5e-4	5e-4	1e-6	65.55
5e-4	1e-5	1e-6	64.51
5e-4	5e-5	1e-6	66.16

Table 10. Accuracy for different parameter combinations(MixStyle cond)

LR	**PLR**	**WD**	**Acc.**
1e-5	1e-4	0.0	63.97
1e-5	5e-4	0.0	63.98
1e-5	1e-5	0.0	63.97
1e-5	5e-5	0.0	63.99
1e-5	1e-4	1e-4	63.96
1e-5	5e-4	1e-4	63.96
1e-5	1e-5	1e-4	63.95
1e-5	5e-5	1e-4	63.95
1e-5	1e-4	1e-5	63.97
1e-5	5e-4	1e-5	63.95
1e-5	1e-5	1e-5	63.96
1e-5	5e-5	1e-5	63.95
1e-5	1e-4	1e-6	63.98
1e-5	5e-4	1e-6	63.95
1e-5	1e-5	1e-6	63.97
1e-5	5e-5	1e-6	63.97
5e-5	1e-4	0.0	60.49
5e-5	5e-4	0.0	60.47
5e-5	1e-5	0.0	60.60
5e-5	5e-5	0.0	63.22
5e-5	1e-4	1e-4	62.62
5e-5	5e-4	1e-4	61.59
5e-5	1e-5	1e-4	59.61
5e-5	5e-5	1e-4	62.94
5e-5	1e-4	1e-5	62.90
5e-5	5e-4	1e-5	60.43
5e-5	1e-5	1e-5	62.16
5e-5	5e-5	1e-5	60.23
5e-5	1e-4	1e-6	62.19
5e-5	5e-4	1e-6	61.78
5e-5	1e-5	1e-6	60.83
5e-5	5e-5	1e-6	59.95

Table 11. Accuracy for different parameter combinations(MLDG)

LR	PLR	WD	Acc.
1e-4	1e-4	0.0	63.75
1e-4	5e-4	0.0	61.39
1e-4	1e-5	0.0	62.79
1e-4	5e-5	0.0	63.19
1e-4	1e-4	1e-4	62.62
1e-4	5e-4	1e-4	60.62
1e-4	1e-5	1e-4	62.12
1e-4	5e-5	1e-4	63.79
1e-4	1e-4	1e-5	61.92
1e-4	5e-4	1e-5	62.50
1e-4	1e-5	1e-5	63.90
1e-4	5e-5	1e-5	61.98
1e-4	1e-4	1e-6	64.91
1e-4	5e-4	1e-6	60.62
1e-4	1e-5	1e-6	63.19
1e-4	5e-5	1e-6	63.32
5e-4	1e-4	0.0	64.42
5e-4	5e-4	0.0	63.80
5e-4	1e-5	0.0	64.01
5e-4	5e-5	0.0	65.17
5e-4	1e-4	1e-4	66.42
5e-4	5e-4	1e-4	65.55
5e-4	1e-5	1e-4	64.83
5e-4	5e-5	1e-4	**67.06**
5e-4	1e-4	1e-5	65.51
5e-4	5e-4	1e-5	64.17
5e-4	1e-5	1e-5	65.32
5e-4	5e-5	1e-5	65.06
5e-4	1e-4	1e-6	64.96
5e-4	5e-4	1e-6	66.36
5e-4	1e-5	1e-6	64.37
5e-4	5e-5	1e-6	64.01

Table 12. Accuracy for different parameter combinations(MLDG cond)

LR	PLR	WD	Acc.
1e-5	1e-4	0.0	64.24
1e-5	5e-4	0.0	64.26
1e-5	1e-5	0.0	64.26
1e-5	5e-5	0.0	64.26
1e-5	1e-4	1e-4	64.24
1e-5	5e-4	1e-4	64.28
1e-5	1e-5	1e-4	64.23
1e-5	5e-5	1e-4	64.26
1e-5	1e-4	1e-5	64.26
1e-5	5e-4	1e-5	64.27
1e-5	1e-5	1e-5	64.24
1e-5	5e-5	1e-5	64.23
1e-5	1e-4	1e-6	64.27
1e-5	5e-4	1e-6	64.28
1e-5	1e-5	1e-6	64.27
1e-5	5e-5	1e-6	64.25
5e-5	1e-4	0.0	62.11
5e-5	5e-4	0.0	61.77
5e-5	1e-5	0.0	60.85
5e-5	5e-5	0.0	62.39
5e-5	1e-4	1e-4	62.05
5e-5	5e-4	1e-4	63.16
5e-5	1e-5	1e-4	63.09
5e-5	5e-5	1e-4	62.13
5e-5	1e-4	1e-5	61.89
5e-5	5e-4	1e-5	62.83
5e-5	1e-5	1e-5	61.42
5e-5	5e-5	1e-5	62.38
5e-5	1e-4	1e-6	60.26
5e-5	5e-4	1e-6	61.29
5e-5	1e-5	1e-6	64.25
5e-5	5e-5	1e-6	60.92