CS3507 COURSE PROJECT

Haokun Zhu
520021910058
Shanghai Jiaotong University
zhuhaokun@sjtu.edu.cn

ABSTRACT

EEG-based emotion recognition is an important branch in the field of affective computing. However, individual differences and noisy labels seriously limit the effectiveness and generalizability of EEG-based emotion recognition models. With the fast development of transfer learning, many domain adaptation and generalization methods have been introduced into this field to promote the effect of emotion recognition. In this project, I implement baselines, domain generalization and domain adpatation methods for the EEG-based emotion recognition task. I implement abundant baseline models for the EEG-based emotion recognition task, including SVM, MLP and ResNet. For domain generalization, I implement the Invariant Risk Minimization domain generalization method. For domain adaptation, I implement 4 methods: Transfer Component Analysis, Domain Adversarial Neural Network, Adversarial Discriminative Domain Adaptation and Prototypical Representation based Pairwise Learning. Moreover, I compare these methods in aspects of training stability and model performance in detail.

***K*eywords** EEG-based Emotion Recognition, Transfer Learning, Domain Adaptation, Domain Generalization

1 Introduction

Recently, electroencephalography (EEG) based emotion recognition has become an increasingly important topic for affective computing and human sentiment analysis [1, 2]. A proper design of EEG-based emotion recognition models is helpful for facilitating the data processing, benefiting discriminant feature characterization, and lightening the model performance. Currently, there exist two main critical issues in EEG-based emotion recognition. One is individual differences: how to build a generalized affective computing model which could tolerate the remarkable individual differences in the simultaneously collected EEG signals; and another is noisy label learning: how to train a reliable and stable affective computing model which is less reliant on the subjective feedback.

In recent years, more and more researchers have focused on applying transfer learning methods to alleviate the individual differences in EEG signals [3, 4, 5, 6, 7, 8] and improve feature invariant representation [9, 10, 11]. Considering the individuals with and without labels (termed as source domain and target domain), transfer learning tries to minimize the distribution difference between the source and target domains by approximately satisfying the assumption of independent and identical distribution and can consequently realize a higher recognition performance on the target domain. Through a domain-shifting strategy, the invariant feature representations across different domains are learned and the relationships among the learned features, data distribution, and labels are explored.

In this project, I mainly complete work in the following three categories:

- I implement abundant baseline models for the EEG-based emotion recognition task, including SVM, MLP and ResNet.
- I take advantage of domain generalization idea and implement the Invariant Risk Minimization[12] domain generalization method for the EEG-based emotion recognition task and compare the results of IRM to the baseline models in detail.
- I implement 4 domain adaptation method:Transfer Component Analysis[13], Domain Adversarial Neural Network[14], Adversarial Discriminative Domain Adaptation[15] and Prototypical Representation based Pairwise Learning[16] for the EEG-based recognition task and compare these methods to baselines and IRM method in detail.

2 Related Work

2.1 Non-deep transfer learning models

Transfer component analysis (TCA) was initially proposed by Pan et al. [13] to maximize the mean discrepancy between the source and target domains by learning transfer information in a reproducing kernel Hilbert space. On the other hand, Zheng and Lu [17] introduced two subject-to-subject transfer methods to handle the challenge of individual differences in EEG signal processing. One approach involved discovering a shared common feature space underlying source and target domains using TCA and kernel principal analysis (KPCA). These non-deep transfer learning strategies can bridge the gap between two domains, but their limited accuracy and stability due to low complexity and small capacity may not meet the practical requirements of EEG-based emotion recognition.

2.2 Deep transfer learning models

Deep transfer learning methods, such as domain-adversarial neural network (DANN) proposed in [14], are the basis of most existing affective models. DANN aims to find a shared feature representation for source and target domains with indistinguishable distribution differences and maintain the predictive ability of the estimated features on the source samples for a specific classification task. Li et al. first introduced DANN. These models generally improve DANN's performance in solving EEG-based emotion recognition tasks from two directions, benefiting from the powerful feature representation ability of deep networks and the high efficiency of adversarial learning in distributed adaptation.

2.3 Domain Generalization Method

Domain generalization deals with a challenging setting where one or several different but related domain(s) are given, and the goal is to learn a model that can generalize to an unseen test domain. Great progress has been made in the area of domain generalization for years. IRM[12] introduces Invariant Risk Minimization (IRM), a learning paradigm to estimate invariant correlations across multiple training distributions. To achieve this goal, IRM learns a data representation such that the optimal classifier, on top of that data representation, matches for all training distributions. Through theory and experiments, we show how the invariances learned by IRM relate to the causal structures governing the data and enable out-of-distribution generalization.

3 Method

3.1 Domain Adaptation

3.1.1 Transfer Component Analysis

Transfer Component Analysis (TCA)[13] is a domain adaptation technique that seeks to find a shared feature space between a source domain $\mathcal{S}$ and a target domain $\mathcal{T}$. Let $\mathbf{X_S}$ and $\mathbf{X_T}$ be the source and target data matrices, respectively, with n_s and n_t samples and d features. The goal of TCA is to learn a transformation matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ that maps the source and target data into a common feature space where the shared structure between the two domains is preserved.

To achieve this, TCA seeks to minimize a cost function consisting of two terms: a reconstruction error term that measures the distance between the original and transformed data, and a distribution alignment term that measures the difference between the source and target distributions in the transformed feature space. Specifically, the cost function can be written as:

$$\min_{\mathbf{A}} \quad \|\mathbf{X_S} - \mathbf{X_S}\mathbf{A}\|_F^2 + \beta\|\mathbf{X_T} - \mathbf{X_T}\mathbf{A}\|_F^2 + \gamma\|\mathbf{A}^T\mathbf{X_S} - \mathbf{A}^T\mathbf{X_T}\|_F^2 \tag{1}$$

where β and γ are hyperparameters that control the trade-off between the reconstruction error and the distribution alignment term. The first term encourages the mapped source data to be close to its original values, the second term encourages the mapped target data to be close to its original values, and the third term encourages the source and target data to have similar representations in the shared feature space.

The optimal transformation matrix $\mathbf{A}$ can be obtained by solving the following eigenvalue problem:

$$\mathbf{X_S}^T\mathbf{K_S}\mathbf{X_S}\mathbf{A} = \mathbf{X_T}^T\mathbf{K_T}\mathbf{X_T}\mathbf{A}\mathbf{\Lambda} \tag{2}$$

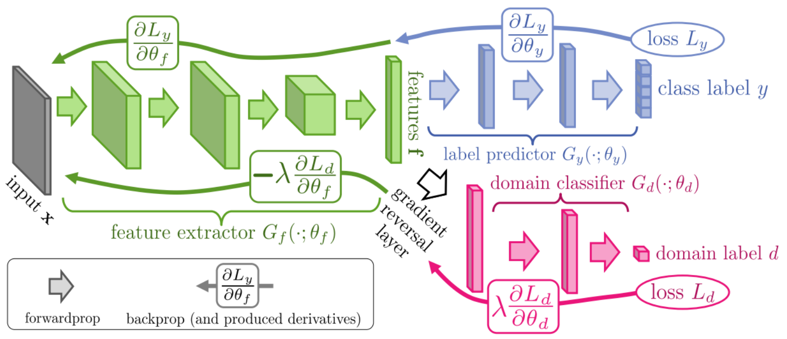
where $\mathbf{K_S}$ and $\mathbf{K_T}$ are the kernel matrices of the source and target data, respectively, and $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues. The transformation matrix $\mathbf{A}$ is then given by the first d eigenvectors corresponding to the largest eigenvalues.

Once the transformation matrix $\mathbf{A}$ is learned, it can be used to adapt a model trained on the source domain to work well on the target domain by applying the transformation to the features of the target data. Specifically, given a target sample $\mathbf{x_T}$, the adapted feature vector $\mathbf{z_T}$ is obtained as:

$$\mathbf{z_T} = \mathbf{x_T}\mathbf{A} \tag{3}$$

3.1.2 Domain Adversarial Neural Network

Domain Adversarial Neural Network (DANN)[14] is a domain adaptation technique that uses a neural network to learn a shared feature representation between a source domain $\mathcal{S}$ and a target domain $\mathcal{T}$ while simultaneously minimizing the difference between the distributions of the two domains in the shared feature space. The key idea behind DANN is to use a domain classifier to distinguish between source and target samples and to train the feature extractor to produce features that are indistinguishable between the two domains.



$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
features $\mathbf{f}$
class label y
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
domain classifier $G_d(\cdot;\theta_d)$
input $\mathbf{x}$
gradient reversal layer
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 1: DANN Framework

The training of DANN involves two objectives: a supervised classification objective for the source domain and an unsupervised domain adaptation objective that encourages the feature representation to be invariant to domain shift. The unsupervised objective is achieved by introducing a domain adversarial loss that encourages the domain classifier to be unable to distinguish between the source and target domains based on the learned feature representation.

Let $\mathbf{X_S}$ and $\mathbf{X_T}$ be the source and target data matrices, respectively, with n_s and n_t samples and d features. The DANN architecture consists of three main components: a feature extractor $f(\cdot)$ that maps the input data to a shared feature representation, a domain classifier $D(\cdot)$ that predicts the domain label of the features, and a task classifier $G(\cdot)$ that performs the actual classification task (e.g., sentiment classification). Specifically, the domain classifier network is trained to predict the domain of the input data, while the feature extractor network is trained to confuse the domain classifier by making the shared feature representation indistinguishable between the source and target domains.

The overall objective of DANN can be written as:

$$\min_{f,G} \max_{D} \mathcal{L}_s(f, G) + \lambda \mathcal{L}_d(f, D) \tag{4}$$

where $\mathcal{L}_s$ is the classification loss on the source domain, $\mathcal{L}_d$ is the domain classification loss, and λ is a hyperparameter that controls the trade-off between the two losses. The domain classification loss is given by:

$$\mathcal{L}_d(f, D) = -\frac{1}{n_s + n_t} \sum_{i=1}^{n_s} \log D(f(\mathbf{x_S}^{(i)})) - \frac{1}{n_s + n_t} \sum_{j=1}^{n_t} \log(1 - D(f(\mathbf{x_T}^{(j)}))) \tag{5}$$

where $\mathbf{x_S}^{(i)}$ and $\mathbf{x_T}^{(j)}$ are the i-th sample from the source domain and the j-th sample from the target domain, respectively.

The feature extractor and task classifier are trained to minimize the classification loss on the source domain and the domain classification loss simultaneously, while the domain classifier is trained to maximize the domain classification loss. This creates a competition between the feature extractor and the domain classifier, where the feature extractor tries to produce features that are indistinguishable between the two domains, while the domain classifier tries to distinguish between the source and target features.

Once the DANN is trained, the feature extractor can be used to extract features from the target domain, which can then be used to perform the classification task on the target domain.

The Gradient Reversal Layer (GRL) is a key component in the Domain Adversarial Neural Network (DANN) architecture. It is used to reverse the gradient during backpropagation through the domain classifier, which enables the feature extractor to learn features that are domain-invariant.

The GRL is a simple layer that acts as the identity function during forward propagation, but negates the gradient during backpropagation. Specifically, given an input x and a scalar weight λ, the output of the GRL is defined as:

$$\mathrm{GRL}(x) = x + \lambda \cdot (-\nabla_x L) \tag{6}$$

where L is the loss function with respect to the domain classification task, and $\nabla_x L$ is the gradient of the loss with respect to the input x. The negative sign in front of the gradient ensures that the gradient is reversed during backpropagation.

In the DANN architecture, the GRL is inserted between the feature extractor and the domain classifier to reverse the gradient during backpropagation through the domain classifier, which forces the feature extractor to produce features that are domain-invariant.

3.1.3 Adversarial Discriminative Domain Adaptation

Adversarial Discriminative Domain Adaptation (ADDA)[15] is a domain adaptation technique that uses adversarial training to learn a feature representation that is invariant to domain shift. The key idea behind ADDA is to use a discriminative loss to stablize the training process. Let $\mathcal{S}$ and $\mathcal{T}$ be the source and target domains, respectively, and $\mathbf{X_S}$ and $\mathbf{X_T}$ be the data matrices

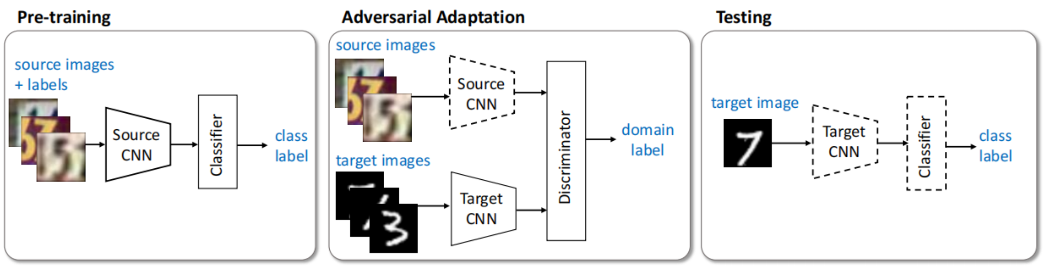


Figure 2: ADDA Framework

with n_s and n_t samples and d features. The ADDA architecture consists of three main components: 2 feature extractors M_s, M_t that maps the input data to a shared feature representation, a classifier C that performs the actual classification task (e.g., sentiment classification) and a Discriminator D to discriminate between target data and source data.

ADDA introduces two separate feature extractors for the source and target domains. It uses a two-step training procedure, where in the pre-training phase, only inputs from the source domain are used to update the parameters of the source feature extractor and the label classifier. In the adversarial training phase, both inputs from the source and target domains are used to update the parameters of the target feature extractor and the domain classifier, while keeping the parameters of the source feature extractor and label classifier fixed.

To ensure stable adversarial training between the target feature extractor and domain classifier, ADDA uses the GAN loss instead of the min-max game used in DANN. The adversarial loss is defined as:

$$\mathcal{L}_{\text{adv}_D}\left(\mathbf{X}_s, \mathbf{X}_t, M_s, M_t\right) = -\mathbb{E}_{\mathbf{x}_s \sim \mathbf{X}_s}\left[\log D\left(M_s\left(\mathbf{x}_s\right)\right)\right] - \mathbb{E}_{\mathbf{x}_t \sim \mathbf{X}_t}\left[\log\left(1 - D\left(M_t\left(\mathbf{x}_t\right)\right)\right)\right] \tag{7}$$

$$\mathcal{L}_{\text{adv}_M}\left(\mathbf{X}_s, \mathbf{X}_t, D\right) = -\mathbb{E}_{\mathbf{x}_t \sim \mathbf{X}_t}\left[\log D\left(M_t\left(\mathbf{x}_t\right)\right)\right] \tag{8}$$

To conclude, ADDA corresponds to the following unconstrained optimization:

$$\min_{M_s, C} \mathcal{L}_{\text{cls}}\left(\mathbf{X}_s, Y_s\right) = -\mathbb{E}_{\left(\mathbf{x}_s, y_s\right) \sim \left(\mathbf{X}_s, Y_s\right)} \sum_{k=1}^{K} \mathbb{1}_{[k=y_s]} \log C\left(M_s\left(\mathbf{x}_s\right)\right) \tag{9}$$

$$\min_{D} \mathcal{L}_{\text{adv}_D}\left(\mathbf{X}_s, \mathbf{X}_t, M_s, M_t\right) = -\mathbb{E}_{\mathbf{x}_s \sim \mathbf{X}_s}\left[\log D\left(M_s\left(\mathbf{x}_s\right)\right)\right] - \mathbb{E}_{\mathbf{x}_t \sim \mathbf{X}_t}\left[\log\left(1 - D\left(M_t\left(\mathbf{x}_t\right)\right)\right)\right] \tag{10}$$

$$\min_{M_s, M_t} \mathcal{L}_{\text{adv}_M}\left(\mathbf{X}_s, \mathbf{X}_t, D\right) = -\mathbb{E}_{\mathbf{x}_t \sim \mathbf{X}_t}\left[\log D\left(M_t\left(\mathbf{x}_t\right)\right)\right] \tag{11}$$

3.1.4 Prototypical Representation based Pairwise Learning

PR-PL[16] is a novel transfer learning framework with Prototypical Representation based Pairwise Learning (PR-PL) to learn discriminative and generalized prototypical representations for emotion revealing across individuals and formulate emotion recognition as pairwise learning for alleviating the reliance on precise label information.

Prototype feature extraction

PR-PL assumes that there exists a prototype for each emotion category. Through prototypical learning, the prototype features are learned to indicate the representation property of every single emotion category. Based on the sample features extracted from different subjects under different emotions, we could consider these sample features are distributed around the prototype features. In other words, for each emotion category, the prototype features could be considered the "center of mass" of all the sample features. From the perspective of a probability distribution, the prototype feature of an emotion category can be regarded as the mean value of the sample feature distribution of the emotion, and the variance of the distribution is caused by the non-stationary EEG, including but not limited to individual differences. Assume that the sample features under an emotion category c obey the Gaussian distribution N. The prototype features of the emotion category c could be calculated as the mean vector μ_c of the distribution. For the source domain data $X_s, Y_s = (x_i, y_i)_{i=0}^{N_s}$, the corresponding sample features

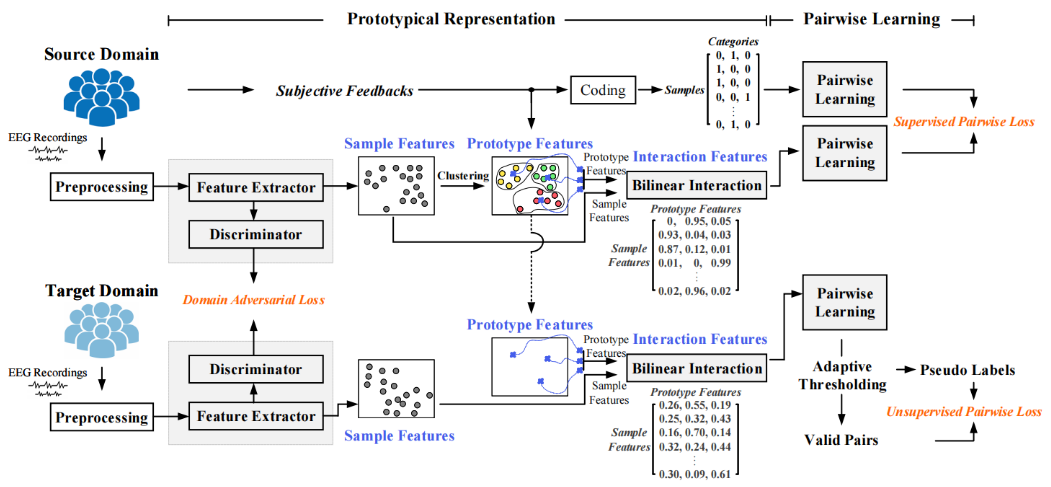


Figure 3: PR-PL Framework

are characterized by the feature extractor $f(\cdot)$, given as $f(x_i)$ (defined in Section III-A). The prototype feature vector of the emotion category c can be calculated by averaging all the sample features that belong to this category, given as

$$\mu_c = \frac{1}{|X_s^c|} \sum_{x_i^s \in X_s^c} f\left(x_i^s\right) \tag{12}$$

where $X_s^c = (x_i^s, y_i^s = c)_{i=0}^{N}$ are a collection of source domain data belonging to the emotion category c. $|Xs, c|$ is the corresponding sample size in this emotion category. In other words, μ_c could be expressed as the centroid of the sample features of $X_{s,c}$. The mean value calculation is a widely used, simple, and effective noise reduction strategy, which can make the prototype features stronger than the sample features and help to alleviate the problem of the traditional DANN network being susceptible to related noise interference. It is worth noting that since the calculation of the prototype features needs to use emotional label information, we only use the source domain data here for prototype feature extraction.

Pairwise learning on source domain

Traditional pointwise learning algorithms often regard the feature vector l as the predicted label of the sample feature x_i and use the cross-entropy loss function to match the label y_i of x_i based on supervised training. Then, if the sample features of one EEG trial match the prototype features of j-th emotion category the most, the EEG trial will be assigned to the j-th emotion category. However, this type of pointwise learning only focuses on the relationship between sample features and prototype features and ignores the relationship between different sample features. For example, the sample features belonging to different emotion categories should be separated from each other, and the sample features belonging to the same emotion category should be gathered together. To tackle this issue, we introduce pairwise learning to capture the inherent relationship of samples.

For the source domain data $X_s, Y_s = (x_{s,i}, y_s)_{i=1}^{N_s}$, the corresponding loss function for pairwise learning is defined as:

$$\mathcal{L}_{\text{class}}\left(\theta\right) = \sum_{i,j} L\left(r_{ij}^s, g\left(x_i^s, x_j^s; \theta\right)\right) \tag{13}$$

where $g(x_i^s, x_j^s; \theta)$ is the similarity measurement of the samples $x_{s,i}$ and $x_{s,j}$, with the parameter of θ. According to the assumption of pairwise learning, if $y_i^s = y_j^s$, then $r_{ij}^s = 1$; otherwise $r_{ij}^s = 0$. The loss function $L(\cdot)$ is a difference calculation of r_{ij} and g.

$$L(r_{ij}^s, g) = -r_{ij}^s \log g - (1 - r_{ij}^s) \log(1 - g) \tag{14}$$

In the training process, the label information of source domain data is used to define r in a supervised manner. In other words, based on the given information of Y_s, if two samples belong to the same emotion category, then $r = 1$, otherwise $r = 0$. A supervised r can ensure the stability of the training process and the generalization ability of the model. The next key question is how to define a proper g to compute the similarity between $x_{s,i}$ and $x_{s,j}$ in terms of the characterized interaction features (termed as l_i and l_j). To make the similarity results locate in the range of $[0, 1]$ and extract better and more robust feature representations for subsequent emotion recognition, we add a norm restriction on l as

$$l^{norm} = \frac{l}{\parallel l \parallel_2} \tag{15}$$

The similarity of l_i^{norm} norm and l_j^{norm} norm is calculated as the cosine similarity, given as

$$g = l_i^{norm} l_j^{norm} = \frac{l_i^s l_j^s}{\| l_i^s \|_2 \| l_j^s \|_2} \tag{16}$$

where refers to inner product operation. As stated in Chang et al. [32], the above-mentioned norm restriction can make the vector l have a clustering function, and the elements in the vector represent the probability that the feature belongs to a certain category cluster. Overall, the objective function of pairwise learning on the source domain is defined as

$$\mathcal{L}_{pairwise}^s = \sum_{i,j} (r_{ij}^s, \frac{l_i^s l_j^s}{\| l_i^s \|_2 \| l_j^s \|_2}) + \beta \| P^T P - I \|_F \tag{17}$$

Pairwise learning on target domain

Besides of source domain, PR-PL also introduces the pairwise learning on the target domain to improve the feature separability in the target domain, as

$$\mathcal{L}_{pairwise}^t = \sum_{i,j} (r_{ij}^t, \frac{l_i^t l_j^t}{\| l_i^t \|_2 \| l_j^t \|_2}) \tag{18}$$

which is termed as unsupervised pairwise loss . Here, l_t is the interaction features of the target domain data characterized. The scalar r_{ij}^t symbolizes the pairing relationship of the samples in the target domain.

3.2 Domain Generalization

3.2.1 Invariant Risk Minimization

Invariant Risk Minimization (IRM)[12] is a domain generalization framework that aims to learn a model that is robust to distribution shift by discovering features that are invariant across different domains. The basic idea behind IRM is to learn a model that performs well across all the domains, instead of just learning a separate model for each domain. To achieve this, IRM minimizes a novel objective function called the Invariant Risk Minimization objective, which ensures that the model is invariant to the distribution shift across different domains.

Let $\mathcal{E} = 1, 2, ..., E$ denote the set of E different environments, and let $\mathcal{D}_e = (\mathbf{x}_1^{(e)}, y_1^{(e)}), ..., (\mathbf{x}_n^{(e)}, y_n^{(e)})$ denote the dataset of n labeled samples drawn from environment $e \in \mathcal{E}$. The goal of IRM is to learn a model $f(\cdot)$ that is invariant to changes in the distribution of the input data across all environments in $\mathcal{E}$.

The IRM objective can be formulated as:

$$\min_{f(\cdot)} \sum_{e=1}^{E} \mathcal{L}e(f(\cdot)) \text{ subject to } f(\cdot) \in \mathcal{F}_{\text{inv}} \tag{19}$$

where $\mathcal{L}_e(f(\cdot))$ is the empirical risk on the dataset $\mathcal{D}e$ under the model $f(\cdot)$, and $\mathcal{F}_{\text{inv}}$ is the set of functions that are invariant to changes in the distribution of the input data across all environments.

To identify the invariant features, IRM uses a penalty function that encourages the model to use only those features that remain invariant across all environments. The penalty function can be formulated as:

$$\Omega(f(\cdot)) = \sum_{e=1}^{E} ||\nabla_{\mathbf{x}} \mathcal{L}_e(f(\cdot))||^2 \tag{20}$$

where $|| \cdot ||$ denotes the L_2 norm and $\nabla_{\mathbf{x}}$ is the gradient with respect to the input features $\mathbf{x}$. The penalty function encourages the model to select features that have small gradients across all environments, since these features are less likely to change across different environments.

The final IRM objective can be written as:

$$\min_{f(\cdot)} \sum_{e=1}^{E} \mathcal{L}e(f(\cdot)) + \lambda \Omega(f(\cdot)) \text{ subject to } f(\cdot) \in \mathcal{F}_{\text{inv}} \tag{21}$$

where λ is a hyperparameter that controls the trade-off between the empirical risk and the penalty function.

By enforcing the invariance constraint, IRM learns a model that is robust to distribution shift across different domains. This is because the model learns to focus on features that are invariant across all the domains, and ignores features that are specific to individual domains. Therefore, the model is able to generalize well to new domains that were not seen during training.

4 Experiment

For the experiment setting, all the experiments are done on i9-10900X CPU @ 3.70GHz along with a GeForce RTX 3090 GPU. The SVM baseline is implemented by the scikit-learn library[18]. All the other deep learning methods are implemented based on the PyTorch[19]. In this section, I will first show the way of searching hyperparameters, then show the general results and comparison of different methods. At last, I will take Fold:12 as example to look in to the training details and quantitative experimental results of each method.

4.1 Hyperparameter Search

In order to search the best hyperparameter, I use sweep function in wandb[20]. It use the Bayes Search method to search in the given range of hyperparameters. Bayesian optimization can be used to search for a set of hyperparameters that results in the best performance of the model on a given task.

The key idea behind Bayesian optimization is to use a probabilistic model, such as a Gaussian process (GP), to approximate the unknown function that maps hyperparameters to model performance. The GP is initialized with some prior distribution over the function, and as the optimization proceeds, the prior is updated based on the results of evaluating the function at different hyperparameter settings.

At each step of the optimization, the Bayesian framework suggests a new set of hyperparameters to evaluate by balancing exploration of new regions of the hyperparameter space and exploitation of the current best hyperparameters. The exploration-exploitation trade-off is achieved by using an acquisition function, which guides the search by estimating the expected improvement in model performance that can be obtained by evaluating the function at a new set of hyperparameters.

The Bayesian optimization algorithm proceeds iteratively until some stopping criterion is met, such as a maximum number of evaluations or a threshold on the improvement in model performance.

I choose Fold:12 of DANN to conduct the search. The results is shown in Fig:4.

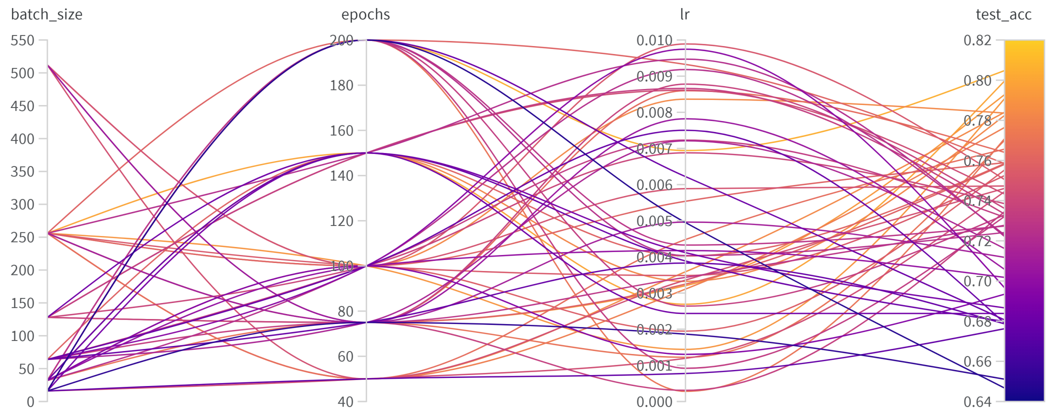


Figure 4: Hyperparameter Search Results

After conducting hyperparamter search, I finally choose the hyperparameters as follows:

- For SVM, I use linear kernel with C=0.01.
- For MLP, TCA, DANN, ADDA and IRM, I use the Adam optimizer implemented in PyTorch with learning rate setting to 1e-3. I don't use weight-decay since there are experiments that proof that weigtht decay may damage the performance of the model. The batch size is set to 256 after conducting Bayes search for a best batch size.
- For ResNet, I use the same hyperparameter as in Project 3.
- For PR-PL, I use the same hyperparameter as illustrated in the paper[16].

4.2 General Results

In this section, I first show the results of the methods used in this project. The quantitative experimental results are shown in Tab:1 and Fig:5.

Method	Model	Fold 0	Fold 3	Fold 6	Fold 9	Fold 12	Mean(↑)±Std(↓)
Baseline	SVM	0.3741	0.3399	0.3694	0.4153	0.5042	0.3566 ± 0.0732
	MLP	0.5082	0.5661	0.5549	0.6411	0.7864	0.5530 ± 0.1101
	ResNet	0.5022	0.4391	0.4503	0.5948	0.6646	0.5024 ± 0.1023
Domain Generalization	IRM	0.4990	0.5046	0.5601	0.6806	0.7693	0.5573 ± 0.1199
Domain Adaptation	TCA	0.3866	0.4941	0.5195	0.5637	0.7707	0.5114 ± 0.1078
	DANN	0.5737	0.5417	0.6603	0.7234	0.8052	0.6253 ± 0.0796
	ADDA	0.4923	0.5164	0.5746	0.6845	0.7456	0.5614 ± 0.1022
	PR-PL	0.7579	0.6489	0.5914	0.7786	0.7427	**0.6498** ± **0.0823**

Table 1: 15-fold cross validation results on SEED-IV dataset

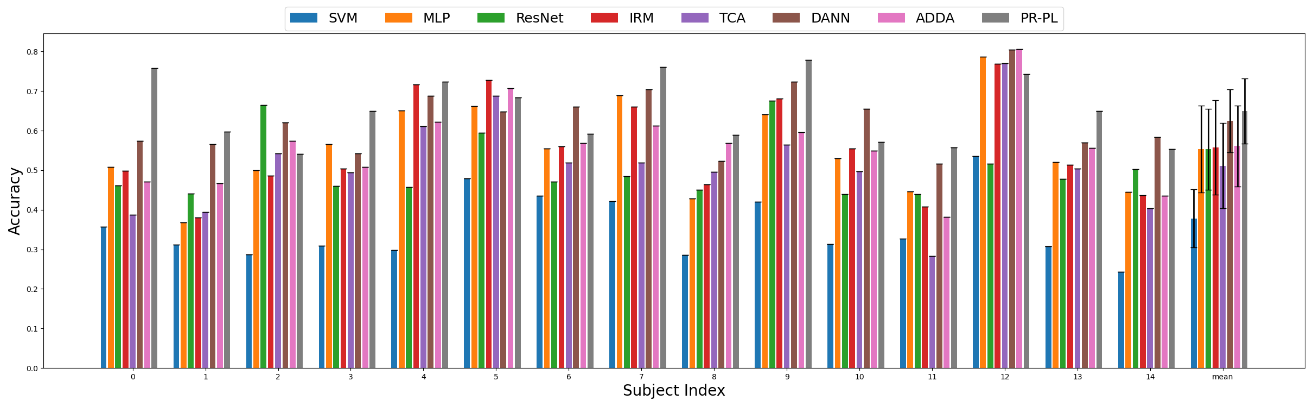


Figure 5: Comparisons between different methods across 15-fold validation

Baseline methods:

The Baseline methods include three models: SVM, MLP, and ResNet, which are trained on the source domain data and tested on the target domain data. SVM performs the worst among the Baseline methods, with a mean accuracy of 0.3566, indicating that the linear SVM model is not able to capture the complex patterns in the target domain data. MLP and ResNet perform better than SVM, with mean accuracy values of 0.5530 and 0.5024, respectively. These results suggest that the neural network models are better suited to handle the complexity of the target domain data.

Domain Generalization method:

The Domain Generalization method uses the Invariant Risk Minimization (IRM) algorithm, which aims to find a model that performs well on multiple related domains. The mean accuracy of IRM is **0.5573**, which is higher than all the Baseline methods, indicating that domain generalization can improve the model's performance. However, the improvement is relatively small compared to the best performing Domain Adaptation method.

Domain Adaptation methods:

The Domain Adaptation methods aim to adapt the model to the target domain by incorporating the target domain data during training. There are four Domain Adaptation methods evaluated in this study: TCA, DANN, ADDA, and PR-PL.

TCA: TCA is a domain adaptation method that applies a kernel transformation to align the source and target domains in a shared feature space. The mean accuracy of TCA is **0.5114**, which is higher than all the Baseline methods and the Domain Generalization method, indicating that domain adaptation can improve the model's performance. However, the improvement is relatively small compared to the other Domain Adaptation methods.

DANN: The mean accuracy of DANN is **0.6253**, which is higher than all the Baseline methods and the Domain Generalization method, indicating that domain adaptation can improve the model's performance. DANN performs better than TCA, indicating that the domain adversarial loss is a more effective way to align the feature distributions than kernel transformation.

ADDA: The mean accuracy of ADDA is **0.5614**, which is higher than all the Baseline methods and the Domain Generalization method, but lower than DANN. This suggests that the two-stage approach may not be as effective as the domain adversarial loss in aligning the feature distributions.

PR-PL: PR-PL is a domain adaptation method that combines adversarial domain adaptation and prototypical learning. The mean accuracy of PR-PL is **0.6498**, which is the highest among all the methods evaluated in this study. This indicates that PR-PL is the most effective method for adapting the model to the target domain data. However, it is worth noting that the standard deviation of PR-PL is relatively high (0.0823), indicating that the performance of PR-PL varies considerably across the different folds.

Among the Domain Adaptation methods, PR-PL achieves the highest mean accuracy (**0.6498**), followed by DANN (**0.6253**), ADDA (**0.5614**), and TCA (**0.5114**). The results suggest that PR-PL is the most effective method for adapting the model to the target domain in this particular dataset. However, it is worth noting that the standard deviations for all Domain Adaptation methods are relatively high, indicating that their performance varies considerably across different folds.

In summary, the Domain Adaptation methods outperform the Baseline methods and the Domain Generalization method, indicating that explicitly adapting the model to the target domain can be beneficial. Among the Domain Adaptation methods, **PR-PL** performs the best,

4.3 Baseline

4.3.1 MLP

The loss curve and training accuracy curve of MLP is shown in Fig:4.3.1.

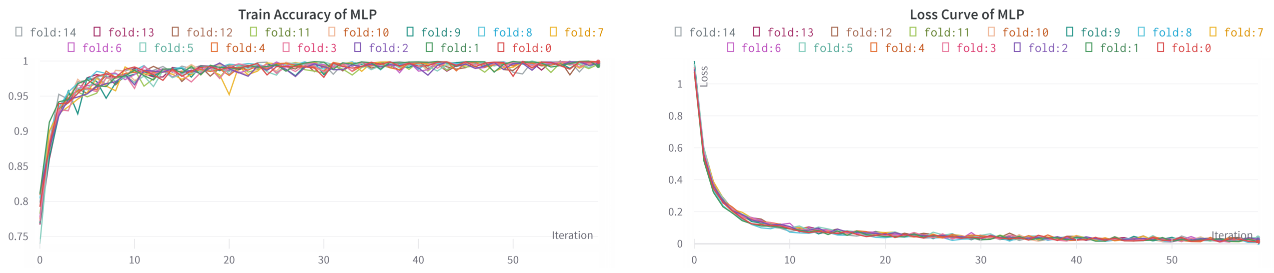


Figure 6: Experiment Results of MLP

In the 14th epoch, the loss on the training set was 0.0811, and the training accuracy was 0.9893. On the other hand, the testing accuracy was 0.7864, which suggests that the model may be overfitting to the training data since the testing accuracy is significantly lower than the training accuracy.

In the following epochs, the loss and training accuracy remain relatively stable, with some fluctuations. However, the testing accuracy fluctuates more significantly, with some epochs having a higher testing accuracy than the previous epochs, while some have a lower testing accuracy.

From the overall trend, we can see that the testing accuracy never reaches the same level as the training accuracy, indicating that the model may be overfitting.

4.3.2 ResNet

The loss curve and training accuracy curve of ResNet is shown in Fig:4.3.2.

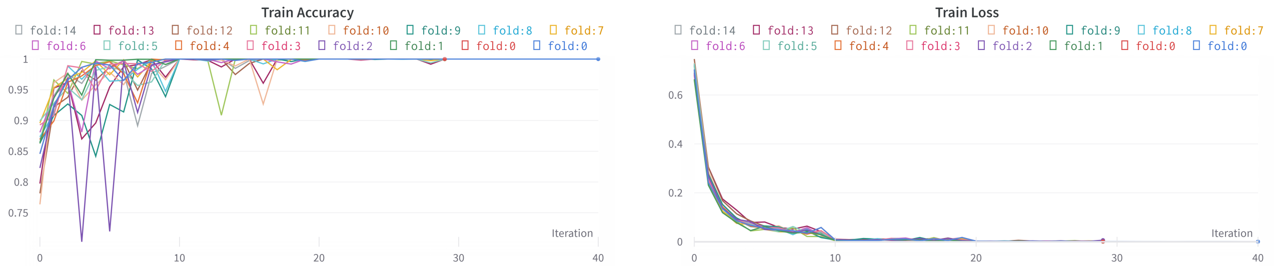


Figure 7: Experiment Results of ResNet

Looking at the result, we can see that the model's training loss and accuracy values gradually improved as the number of epochs increased. Specifically, the training loss decreased from 0.75 to 0.003, and the training accuracy increased from 0.78 to 1.0. This suggests that the model was learning to fit the training data well.

However, when looking at the validation accuracy values, we can see that the model's performance was not consistently improving. The validation accuracy fluctuated between 0.445 and 0.617, and there were even some epochs where the validation accuracy decreased. This suggests that the model may have been overfitting to the training data, meaning that it was performing well on the training data but not generalizing well to new data.

Overall, this result suggests that the ResNet model was able to achieve a high level of accuracy on the training data but had difficulty generalizing to new data. Further analysis, such as hyperparameter tuning or regularization techniques, may be necessary to improve the model's performance.

4.4 Domain Adaptation

4.4.1 Transfer Component Analysis

The trend of train loss curve and train accuracy curve of TCA is quite similar to the MLP baseline. Besides, after adding the regularization loss, the performance of the TCA improves a lot compared to the MLP baseline.

4.4.2 Domain Adversarial Neural Network

The loss curve and training accuracy curve of Domain Adversarial Neural Network is shown in Fig:4.4.2.

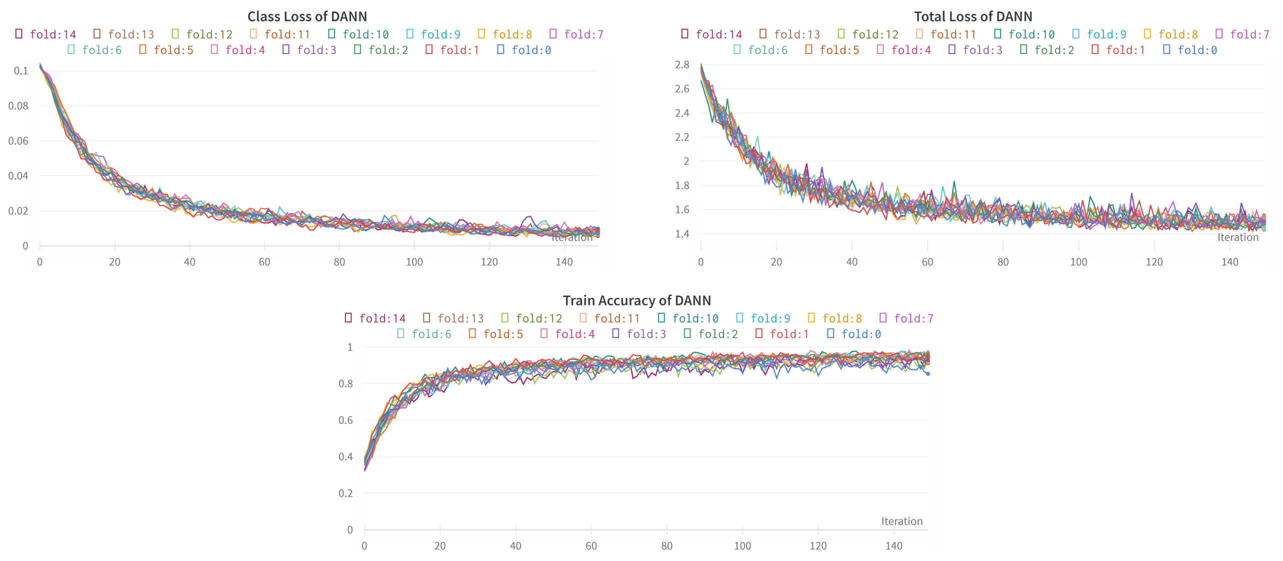


Figure 8: Experiment Results of DANN

The result shows that the network starts with low accuracy on both training and test data and gradually improves with each epoch. The class loss and domain loss both decrease during training, indicating that the network is improving at both tasks.

The test accuracy starts at 43.67% and ends at 80.52%, showing a significant improvement over training. The best test accuracy is achieved at epoch 55, where it reaches 80.52%.

The training accuracy follows a similar trend, starting at 38.20% and ending at 93.75%. The best training accuracy is achieved at epoch 74, where it reaches 93.75%.

It is worth noting that the training accuracy is consistently higher than the test accuracy, indicating some overfitting. The domain loss is generally higher than the class loss, indicating that the domain adversarial task is harder for the network than the classification task.

Overall, the result shows that the DANN is effective at learning a domain-invariant feature representation for the classification task, but there is still room for improvement in terms of reducing overfitting and improving domain adversarial performance.

4.4.3 Adversarial Discriminative Domain Adaptation

The loss curve and training accuracy curve of Adversarial Discriminative Domain Adaptation is shown in Fig:4.4.3.

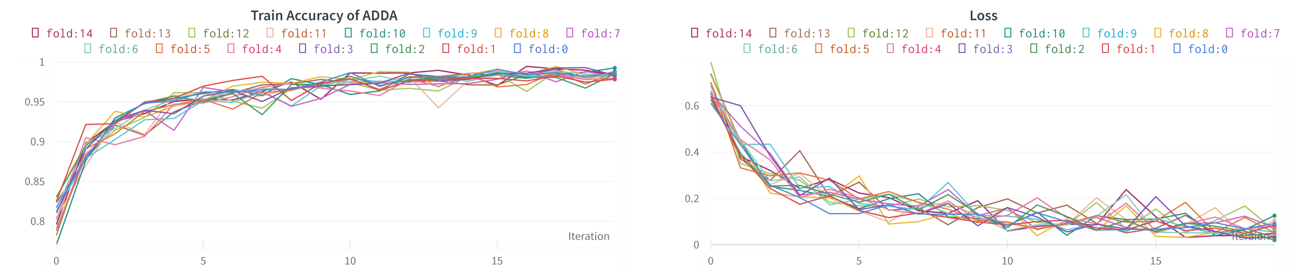


Figure 9: Experiment Results of ADDA

The training of ADDA is split into two stages: pretrain and train. In the pretrain stage, the model is trained on a source dataset, and in the train stage, it is fine-tuned on a target dataset.

The pretrain stage has 20 epochs. The training accuracy steadily increases throughout the epochs, starting at 0.7910 and ending at 0.9835. The test accuracy also shows improvement, starting at 0.7485 and ending at 0.8196, although it has some fluctuations along the way.

The train stage also has 20 epochs. However, the training accuracy is consistently high, starting at 0.9837 and ending at 0.9833. The target dataset accuracy, on the other hand, shows a different pattern. It starts at 0.7704 and drops significantly in the first few epochs, reaching a low of 0.4301 by the end.

4.4.4 Prototypical Representation based Pairwise Learning

The loss curve and training accuracy curve of Prototypical Representation based Pairwise Learning is shown in Fig:4.4.4.

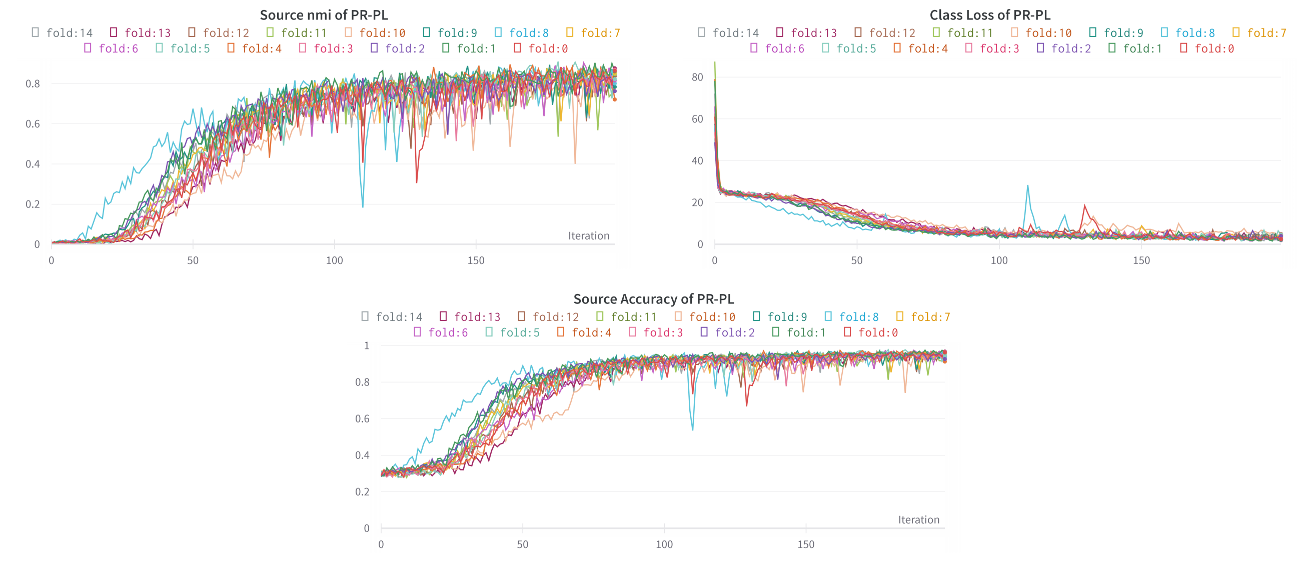


Figure 10: Experiment Results of PR-PL

The result shows that the network starts with low accuracy on both training and test data and gradually improves with each epoch. The loss decrease during training, indicating that the network is improving at both tasks.

The test accuracy starts at 0.4142 and ends at 0.7427, showing a significant improvement over training. The best test accuracy is achieved at epoch 72, where it reaches 0.7427.

The training accuracy follows a similar trend, starting at 0.2970 and ending at 0.9618. The best training accuracy is achieved at epoch 79, where it reaches 0.9618.

Overall, the result shows that the PR-PL is effective at learning a domain-invariant feature representation for the classification.

4.5 Domain Generalization

4.5.1 Invariant Risk Minimization

The loss curve and training accuracy curve of Invariant Risk Minimization is shown in Fig:4.5.1.

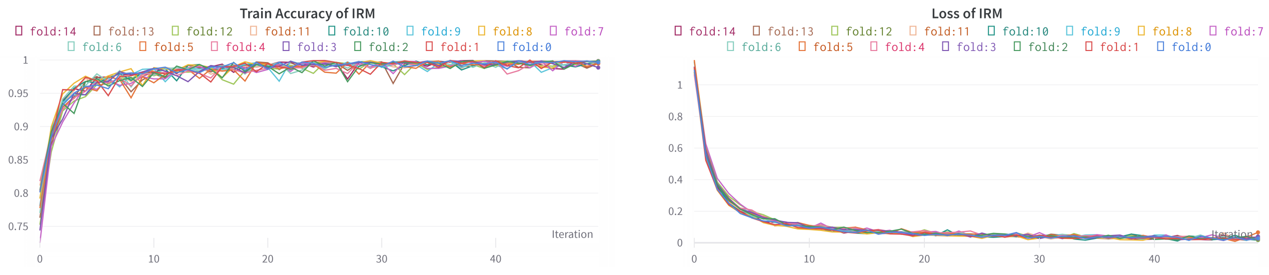


Figure 11: Experiment Results of IRM

Epoch 0 starts with a high loss of 1.1484, but the training accuracy is 0.7524, and the test accuracy is 0.7341, indicating that the model is learning something from the training data, but not generalizing well to the test data.

In the subsequent epochs, the loss decreases, and both the training and test accuracies increase, which suggests that the model is improving. From epoch 5 onwards, the model achieves a higher test accuracy than the previous epoch, indicating that the model is starting to generalize better to unseen data.

However, the result also shows that the performance of the model on the test set fluctuates across epochs. For example, the test accuracy drops from 0.7693 in epoch 5 to 0.7190 in epoch 6, and then increases to 0.7234 in epoch 7. Similarly, the test accuracy drops from 0.7293 in epoch 10 to 0.7034 in epoch 12.

Overall, the result shows that the model trained using IRM is able to learn from the data and improve its performance over time, but it struggles to generalize well to unseen data, and its performance on the test set is somewhat unstable across epochs. Further analysis is needed to determine the cause of these issues and improve the model's performance.

5 Conclusion

In this project, I implement baselines, domain generalization and domain adpatation methods for the EEG-based emotion recognition task. I implement abundant baseline models for the EEG-based emotion recognition task, including SVM, MLP and ResNet. For domain generalization, I implement the Invariant Risk Minimization domain generalization method. For domain adaptation, I implement 4 methods: Transfer Component Analysis, Domain Adversarial Neural Network, Adversarial Discriminative Domain Adaptation and Prototypical Representation based Pairwise Learning. Moreover, I compare these methods in aspects of training stability and model performance in detail.

After carefully looking into the quantitative experimental results and comparing the training stability and model performance of methods mentioned above, I come up with the conclusion that PR-PL model performs best, followed by DANN, in this project for the EEG-based emotion recognition task. This conclusion is reasonable as PR-PL model take advantage of prototype feature extraction and pairwise learning and it is a DANN-based model, in another word, it has all the advantages which DANN possesses.

6 Acknowledgement

I sincerely thank Prof. LU, Prof. Zhang and TAs for their guidance in transfer learning and introduction for the EEG-based emotion recognition task. This project is very inspiring and educational. I have learned a lot about transfer learning from this project. At last, thanks for reading to this end and hope this paper will inspire you to some extent.

References

[1] Xin Hu, Jingjing Chen, Fei Wang, and Dan Zhang. Ten challenges for eeg-based affective computing. *Brain Science Advances*, 5(1):1–20, 2019.

[2] Wanrou Hu, Gan Huang, Linling Li, Li Zhang, Zhiguo Zhang, and Zhen Liang. Video-triggered eeg-emotion public databases and current methods: a survey. *Brain Science Advances*, 6(3):255–287, 2020.

[3] Vinay Jayaram, Morteza Alamgir, Yasemin Altun, Bernhard Scholkopf, and Moritz Grosse-Wentrup. Transfer learning in brain-computer interfaces. *IEEE Computational Intelligence Magazine*, 11(1):20–31, 2016.

[4] Jinpeng Li, Shuang Qiu, Yuan-Yuan Shen, Cheng-Lin Liu, and Huiguang He. Multisource transfer learning for cross-subject eeg emotion recognition. *IEEE transactions on cybernetics*, 50(7):3281–3293, 2019.

[5] Yang Li, Wenming Zheng, Lei Wang, Yuan Zong, and Zhen Cui. From regional to global brain: A novel hierarchical spatial-temporal neural network model for eeg emotion recognition. *IEEE Transactions on Affective Computing*, 13(2):568–578, 2019.

[6] Heng Cui, Aiping Liu, Xu Zhang, Xiang Chen, Kongqiao Wang, and Xun Chen. Eeg-based emotion recognition using an end-to-end regional-asymmetric convolutional neural network. *Knowledge-Based Systems*, 205:106243, 2020.

[7] Peixiang Zhong, Di Wang, and Chunyan Miao. Eeg-based emotion recognition using regularized graph neural networks. *IEEE Transactions on Affective Computing*, 13(3):1290–1301, 2020.

[8] Xiaotong Gu, Zehong Cao, Alireza Jolfaei, Peng Xu, Dongrui Wu, Tzyy-Ping Jung, and Chin-Teng Lin. Eeg-based brain-computer interfaces (bcis): A survey of recent studies on signal sensing technologies and computational intelligence approaches and their applications. *IEEE/ACM transactions on computational biology and bioinformatics*, 18(5):1645–1666, 2021.

[9] Ozan Özdenizci, Ye Wang, Toshiaki Koike-Akino, and Deniz Erdoğmuş. Adversarial deep learning in eeg biometrics. *IEEE signal processing letters*, 26(5):710–714, 2019.

[10] Ozan Özdenizci, Ye Wang, Toshiaki Koike-Akino, and Deniz Erdoğmuş. Learning invariant representations from eeg via adversarial inference. *IEEE access*, 8:27074–27085, 2020.

[11] David Bethge, Philipp Hallgarten, Tobias Grosse-Puppendahl, Mohamed Kari, Ralf Mikut, Albrecht Schmidt, and Ozan Özdenizci. Domain-invariant representation learning from eeg with private encoders. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1236–1240. IEEE, 2022.

[12] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization, 2020.

[13] Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2011.

[14] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks, 2016.

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

[16] Rushuang Zhou, Zhiguo Zhang, Hong Fu, Li Zhang, Linling Li, Gan Huang, Yining Dong, Fali Li, Xin Yang, and Zhen Liang. Pr-pl: A novel transfer learning framework with prototypical representation based pairwise learning for eeg-based emotion recognition, 2022.

[17] Wei-Long Zheng and Bao-Liang Lu. Personalizing eeg-based affective models with transfer learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 2732–2738. AAAI Press, 2016.

[18] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[20] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.