TRANSFER LEARNING APPLICATION ON CROSS-SUBJECT EMOTION RECOGNITION

Jiayi Zhang
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai
{jiayizhang}@sjtu.edu.cn

ABSTRACT

In this project, several transfer learning models have been applied on a cross-subject emotion recognition task on SEED-IV dataset. First, we implement SVM classifier, MLP classifier and CNN classifiers (a naive one and ResNet18) as baseline. Then we try two domain generalization methods MixStyle and IBN and two domain adaptation methods ADDA and DANN. The DANN model achieves an accuracy of approximately 63%, which is a considerable improvement comparing with baseline models.

***K*eywords** electroencephalography (EEG) · affective brain-computer interface · domain adaptation · domain generalization

1 Introduction

Directly reflecting brain activities, electroencephalography (EGG) is a reliable and promising indicator of human mental state and has a close relationship with human emotions. Emotion recognition can help researchers to learn more about patterns of mood changes and even help psychologitsts to diagnose mental diseases. In order to estimate human emotions with their electroencephalography, an effective model for cross-subject emotion recognition based on electroencephalography data is required.

Unlike traditional classification tasks on graphs or texts, electroencephalography datasets are non-stationary and non-iid. Traditional classification models cannot achieve an ideal accuracy on cross-subject emotion recognition tasks and some researchers introduce transfer learning methods trying to improve the performance of models.

In this project, we work on the cross-subject emotion recognition task on the SEED-IV dataset. Our main contributions can be summarized as three parts below:

1. We implement SVM classifier, MLP classifier and CNN classifiers (a naive one and ResNet18) as baseline models.
2. We implement two domain generalization models MixStyle-CNN and IBN-ResNet18.
3. We implement two domain adaptation methods ADDA and DANN.

2 Materials and Methods

2.1 Dataset Description

In this project, we use SEED-IV dataset to evaluate our models. The SEED-IV is an EEG dataset from SJTU Emotion EEG Dataset (SEED) [1] provided by the BCMI laboratory, which is led by Prof.Bao-Liang Lu. SEED-IV contains four categories of emotions changes: happy, sad, fear and neutral. SEED-IV contains 3 sessions and each session contains 15 subjects.

1. In session 1, the train data dimension is $610 \times 62 \times 5$ and test data dimension is $241 \times 62 \times 5$.
2. In session 2, the train data dimension is $558 \times 62 \times 5$ and test data dimension is $274 \times 62 \times 5$.
3. In session 3, the train data dimension is $567 \times 62 \times 5$ and test data dimension is $255 \times 62 \times 5$.

62 refers to the number of electrobe channels and 5 refers to the number of frequency bands.

2.2 Baseline Methods

2.2.1 SVM Classifier

Support Vector Machines (SVM) is a simple binary classification model, it tries to learn a hyperplane which can divide data points with different labels and has the maximum distance between them.

In order to apply SVM to multi-label classification tasks, one-versus-rest (OVR) and one-versus-one (OVO) methods are introduced. OVR transfers one n-label classification task to n binary classification tasks and each binary classification task is trying to classify one given label from all the rest labels. OVO transfers one n-label classification task to $\frac{n(n-1)}{2}$ binary classification tasks and each binary classification task is trying to classify data between two given labels.

2.2.2 MLP Classifier

Multi-Layer Perception (MLP) is a simple neural network. It contains more than one fully connected layers, first is the input layer, then are hidden layers and at last is the output layer. With the input data, the MLP Classifier adjusts parameters in the network and tries to find the mapping function between data and labels. Obviously, the more complex the MLP network is, the more complex task it can deal with.

2.2.3 CNN Classifier

Convolutional Neural Network (CNN) is a slightly complex neural network comparing with MLP mentioned above. It contains three different layers: convolutional layer, max pooling layer and fully connected layer. Convolutional layers mainly serve as feature extractors, max pooling layers mainly serve for downsampling and fully connected layers mainly serve for classification.

Convolutional layers use a convolution core to extract features from input data. Convolution is a mathematical operation that combines two functions to produce a third function that represents how on function modifies the other. With a 2D matrix input, convolution layers slide a small filter 2D matrix convolution core over the matrix, multiplying the filter values with the corresponding pixel values in the matrix at each position and summing up the results to produce a new output 2D matrix.

2.2.4 ResNet18 Classifier

Except a naive CNN Classifier, we also implement ResNet18 [2] as one of our baseline model. ResNet18 is a classical convolutional network for graph classification tasks. It is a type of deep neural network which utilizes residual connections to help address the vanishing gradient problem.

ResNet18 consists of 18 layers, the basic component is called "BasicBlock" and it uses "shortcut" to enable the gradient to flow more easily through the network. With this sideway, ResNet18 can prevent the problem of vanishing gradients that may occur in deep networks and improve the accuracy of the model.

2.3 Domain Generalization Methods

In this project, our task is to learn from 14 different source domains and make predictions on another one domain. The difference between 14 different source domains may have negative effects on our models. Domain generalization methods can reduce difference between different source domains, which means those methods may bring some improvement on our models.

2.3.1 MixStyle

MixStyle [3] is a domain generalization method based on convolutional neural networks aiming to learn from a set of source domains and produce a model that is generalizable to any unseen domain. In our project, our task is to learn from 14 chosen domains and make predictions on another one domain. Obviously, our project can be treated as a domain generalization task which is suitable for MixStyle.

MixStyle regularize CNN training by perturbing the style information of source domain training instances. Specifically, MixStyle can be implemented as a plug-and-play module inserted between CNN layers without the need to explicitly generate an image of new style.

Below is the working process of MixStyle module:

1. generate reference batch $\tilde{x}$ from x:
   - When domain labels are given, $x = [x^i, x^j]$ is sampled from two differet domains i and j, and $\tilde{x}$ is obtained by swapping the position of x^i and x^j.See in Fig.1.

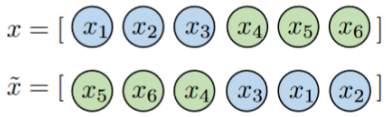

Figure 1: Shuffling batch w/ domain label

   - When domain labels are unknown, x is randomly sampled from the training data, and $\tilde{x}$ is simply obtained by $\tilde{x} = Shuffle(x)$. See in Fig.2

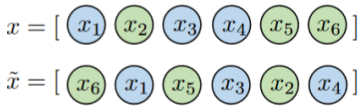

Figure 2: Shuffling batch w/ random label

2. After shuffling, MixStyle computes the mixed feature statistics by

$$\gamma_{mix} = \lambda\sigma(x) + (1-\lambda)\sigma(\tilde{x}) \tag{1}$$
$$\beta_{mix} = \lambda\mu(x) + (1-\lambda)\mu(\tilde{x}) \tag{2}$$
$$MixStyle(x) = \gamma_{mix}\frac{x-\mu(x)}{\sigma(x)} + \beta_{mix} \tag{3}$$

Following the procedure mentioned above, a MixStyle module is easy to implement [4]. It gets input tensor x and gives out style-mixed output tensor $MixStyle(x)$. With the MixStyle module inserted, convolutional neural networks get the domain generalization ability.

2.3.2 IBN-Net

Instance-Batch Normalization Networks (IBN-Net) [5] is originally designed for scene parsing tasks. Researchers try to apply a scene parsing model trained on a single domain to another domain, such as applying a model trained on the real images of Cityscape dataset to virtual images of GTA5 dataset. In our project, our task is to learn from 14 different domains and apply the trained model to another different domain. As a novel convolution architecture with domain generalization method, IBN-Net is a potential solution for our task.

IBN-Net carefully integrates Instance Normalization (IN) and Batch Normalization (BN) as building blocks, enhancing both its learning and generalization capacity. Batch Normalization uses the mean and variance of a mini-batch to normalize each feature channels during training and uses the global statistics to normalize features while in inference phase. It enables larger learning rate and faster convergence by reducing the internal convariate shift during training CNNs. Instance Normalization uses the statistics of an individual sample instead of mini-batch to normalize features both on training phase and inference phase. It allows to filter out instance-specific contrast information from the content.

IBN-Net design is based on an important observation: for BN based CNNs, appearance variance mainly lies in shallow half of the CNN. With this observation, IBN-Net introduces INs following two rules:

- Do not add INs in the last part of CNNs
- Keep part of the batch normalized features

In practical [6], IBN-Net can be implement as CNN networks with IN and BN modules. Researchers design a plug-and-play module which provides a simple way to increase both modeling and generalization capacity without adding model complexity.

2.4 Domain Adaptation Methods

Domain adaptation methods attempt to mitigate the harmful effects of domain shift, so domain adaptation models have more direct improvement on performance than models with domain generalization which has indirect improvement by reducing the harmful difference between training datasets from different source domains.

2.4.1 ADDA

Adversarial Discriminative Domain Adaptation (ADDA) [7] is a domain adaptation model based on adversarial learning method. In this adversarial adaptive method, its main goal is to regularize the learning of the source and target mappings, which means to minimize the distance between the empirical source and target mapping distributions.

ADDA tries to find a feature space which has the closest distance mapping from the source domain and the target domain. In order to extract proper features, ADDA has mainly two phases as below (also in Fig.3):

1. Pretrain Phase: Train a source feature extractor and a classifier with data from source domain. The optimizer's target at this phase is to minimize the classifier's prediction loss, which means the source feature extractor and the classifier are ideal.
2. Train Phase: Train a target feature extractor and a discriminator with data from target domain. The optimizer's target at this phase is to maximize the discriminator's predictions loss, which means the feature mapping from data in target domain to the feature space achieves the smallest distance to features in source domain, the discriminator cannot figure out whether the input feature is extracted from source domain or target domain.

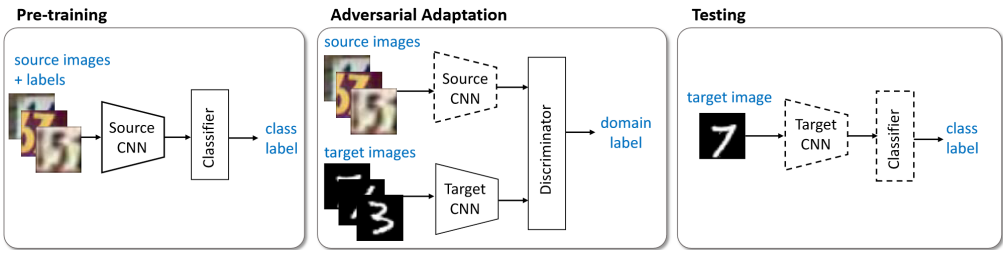

Figure 3: ADDA approach

In conclusion, ADDA has three unconstrained optimization targets as below:

$$\min_{M_s,C} \mathcal{L}_{cls}(\mathbf{X}_s, Y_s) = -\mathbb{E}_{(\mathbf{X}_s,y_s)\sim(\mathbf{X}_s,Y_s)} \sum_{k=1}^{K} \mathbb{1}_{[k=y_s]} \log C(M_s(\mathbf{x}_s)) \tag{4}$$

$$\min_{D} \mathcal{L}_{adv_D}(\mathbf{X}_s, \mathbf{X}_t, M_s, M_t) = -\mathbb{E}_{\mathbf{X}_s\sim\mathbf{X}_s}[\log D(M_s(\mathbf{X}_s))] - \mathbb{E}_{\mathbf{X}_t\sim\mathbf{X}_t}[\log(1 - D(M_t(\mathbf{X}_t)))] \tag{5}$$

$$\min_{M_t} \mathcal{L}_{adv_M}(\mathbf{X}_s, \mathbf{X}_t, D) = -\mathbb{E}_{\mathbf{X}_t\sim\mathbf{X}_t}[\log D(M_t(\mathbf{X}_t))] \tag{6}$$

2.4.2 DANN

Domain-Adversarial Neural Network (DANN) [8] [9] is a adversarial neural network taking input data from varient domains and providing predictions on another one domain. DANN uses domain-adversarial training method to minimize the distance between source feature sapce and target feature space. With this domain-adversarial design, DANN is able to learn better with data from different domains.

DANN has mainly three components (as in Fig.4): feature extractor $G_f(\cdot;\theta_f)$, label predictor $G_y(\cdot;\theta_y)$ and domain classifier $G_d(\cdot;\theta_d)$

At source domain, DANN extracts features from input data with feature extractor and then uses features to predict labels with label predictor. Assuming that the feature extractor uses "sigmoid" as activation function and the lable

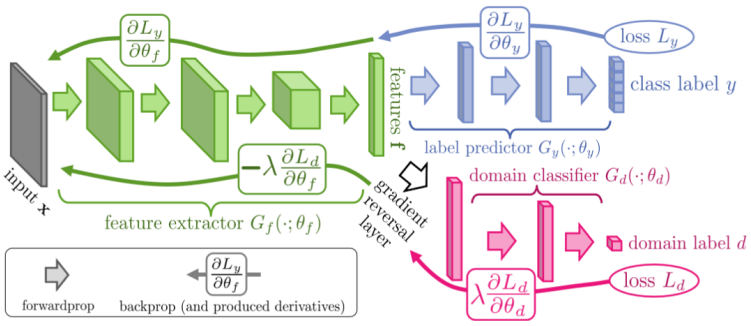

Figure 4: DANN approach

predictor uses "softmax" as activation function, then the upper path in Fig.4 learning from source domain corresponds to following equations:

$$G_f(\mathbf{x};\mathbf{W},\mathbf{b}) = sigm(\mathbf{W}\mathbf{x}+\mathbf{b}) \tag{7}$$

$$G_y(G_f(\mathbf{x});\mathbf{V},\mathbf{c}) = softmax(\mathbf{V}G_f(\mathbf{x})+\mathbf{c}) \tag{8}$$

$$\mathcal{L}_y(G_y(G_f(\mathbf{x}_i)),y_i) = \log\frac{1}{G_y(G_f(\mathbf{x}))_{y_i}} \tag{9}$$

Hence, at source domain, the optimization goal is:

$$\min_{\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}}[\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}) + \lambda\cdot R(\mathbf{W},\mathbf{b})] \tag{10}$$

Here λ is a hyper-parameter and $R(\mathbf{W},\mathbf{b})$ is an optional regularizer.

At target domain, DANN extracts features from input data with feature extractor and then uses domain classifier to classify whether the feature data comes from target domain or source domain. Assuming that the domain classifier uses "sigmoid" as activation function, then the lower path in Fig.4 learning from target domain corresponds to following equations:

$$G_d(G_f(\mathbf{x});\mathbf{u},z) = sigm(\mathbf{u}^\top G_f(\mathbf{x})+z) \tag{11}$$

$$\mathcal{L}_d(G_d(G_f(\mathbf{x}_i)),d_i) = d_i\log\frac{1}{G_d(G_f(\mathbf{x}_i))} + (1-d_i)\log\frac{1}{1-G_d(G_f(\mathbf{x}_i))} \tag{12}$$

Hence, at target domain, the optimization goal is:

$$R(\mathbf{W},b) = \max_{\mathbf{u},z}[-\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i(\mathbf{W},\mathbf{b},\mathbf{u},z) - \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i(\mathbf{W},\mathbf{b},\mathbf{u},z)] \tag{13}$$

Combining the two path above, we have the total optimization goal or the total loss of DANN:

$$E(\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c},\mathbf{u},z) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}) - \lambda(\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i(\mathbf{W},\mathbf{b},\mathbf{u},z) + \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i(\mathbf{W},\mathbf{b},\mathbf{u},z)) \tag{14}$$

3 Implementation

3.1 Make Dataset

Since we implement different models for our project and our project needs to use training data form 14 different domins as the model's training dataset, the original SEED-IV dataset cannot be used directly. Since we have to preprocess original SEED-IV dataset, we implement a `maeke_dataset` process which is decoupled from the main training process to produce datasets which can be used directly for different models.

For SVM, MLP and models based on MLP like ADDA and DANN, we have to concatenate training datasets from 14 domains and change the dimension of the tensor from $batch\_size \times 62 \times 5$ to $batch\_size \times 310$ which is a suitable dimension for SVM and MLP based models.

For CNN and models based on CNN like ResNet18, we have to reshape concatenate training datasets from 14 domains and change the dimension of the tensor from $batch\_size \times 62 \times 5$ to $batch\_size \times 5 \times 9 \times 9$ which is a suitable dimension for CNN and CNN based models. As the Fig.5, we reshape 62×5 tensor with this mapping.

0	0	0	Fp1= d[0]	Fpz =d[1]	Fp2 =d[2]	0	0	0
0	0	AF3= d[3]	0	0	0	AF4= d[4]	0	0
F7= d[5]	F5= d[6]	F3= d[7]	F1= d[8]	Fz= d[9]	F2= d[10]	F4= d[11]	F6= d[12]	F8= d[13]
FT7= d[14]	FC5= d[15]	FC3= d[16]	FC1= d[17]	FCz= d[18]	FC2= d[19]	FC4= d[20]	FC6= d[21]	FT8= d[22]
T7= d[23]	C5= d[24]	C3= d[25]	C1= d[26]	Cz= d[27]	C2= d[28]	C4= d[29]	C6= d[30]	T8= d[31]
TP7= d[32]	CP5= d[33]	CP3= d[34]	CP1= d[35]	CPz= d[36]	CP2= d[37]	CP4= d[38]	CP6= d[39]	TP8= d[40]
P7= d[41]	P5= d[42]	P3= d[43]	P1= d[44]	Pz= d[45]	P2= d[46]	P4= d[47]	P6= d[48]	P8= d[49]
PO7= d[50]	PO5= d[51]	PO3= d[52]	0	POz= d[53]	0	PO4= d[54]	PO6= d[55]	PO8= d[56]
0	CB1= d[57]	0	O1= d[58]	Oz= d[59]	O2= d[60]	0	CB2= d[60]	0

Figure 5: Reshape mapping

3.2 Baseline Methods

3.2.1 SVM Classifier

We implement SVM Classifier with `sklearn` module form `scikit-learn`. With this module, we can just call the function `clf=svm.SVC()` to create a SVM classifier object.

3.2.2 MLP Classifier

We implement MLP Classifier with `pytorch`, and the architecture is as Fig.6. The training process is as pseudo-code 1

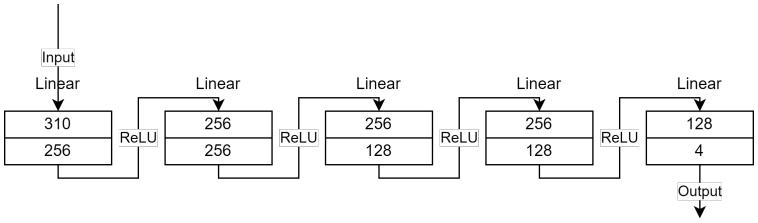

Figure 6: MLP architecture

at appendix A.

3.2.3 CNN Classifier

We implement CNN Classifier with `pytorch`, and the architecture is as Fig.7. The training process is as pseudo-code 1 at appendix A.

3.2.4 ResNet18 Classifier

We implement ResNet18 Classifier with `pytorch`, and the architecture is as Fig.8 and Fig.9. The training process is as pseudo-code 1 at appendix A.

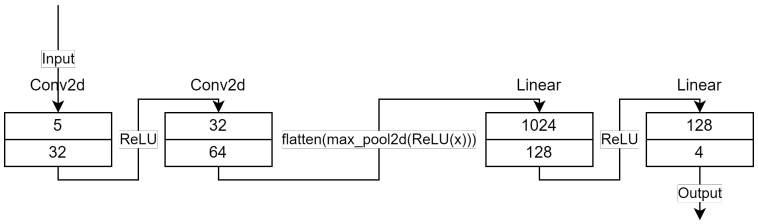

Figure 7: CNN architecture

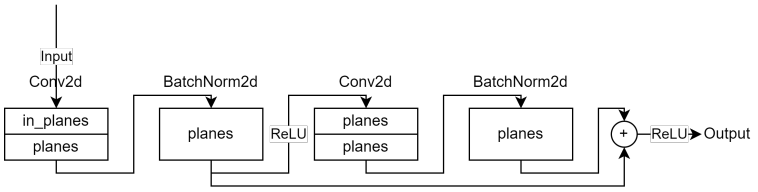

Figure 8: BasicBlock architecture

3.3 Domain Generalization Methods

3.3.1 MixStyle

MixStyle model is based on the CNN model. As we mentioned above, mixstyle module can be inserted into the CNN model. Referring to [4], we implement a MixStyle-CNN model and it's architecture is as Fig.10.

As we mentioned above, MixStyle module has two different working mode depending on whether the domain label is given or not. We implement our MixStyle-CNN model with the domain label unknown mode (or say the random mode), and the pseudo-code for the MixStyle module is pseudo-code 2.

3.3.2 IBN-Net

IBN-Net model is based on the ResNet18 model. As we mentioned above, we can insert IB module and BN module into the ResNet module. Referring to [6], we can insert IB and BN module into the BasicBlock of ResNet18 as Fig.11. Our IBN-BasicBlock's architecture is as Fig.12. The training process is as pseudo-code 1 at appendix A for baseline models.

3.4 Domain Adaptation Methods

3.4.1 ADDA

In previous section, we have figured out that the ADDA model mainly has four different components: source feature extractor, target feature extractor, classifier, discriminator. Referring to [10], we design all those components based on simple MLP network, as in Fig.13, Fig.14 and Fig.15. The training process is as pseudo-code 3 at appendix A.

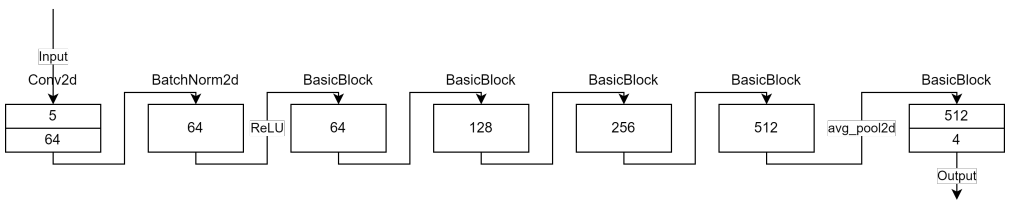

Figure 9: ResNet18 architecture

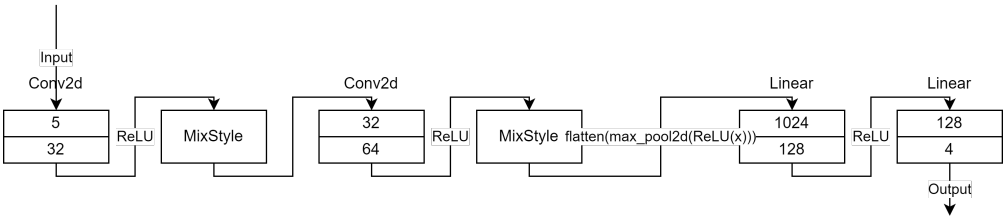

Figure 10: MixStyle architecture

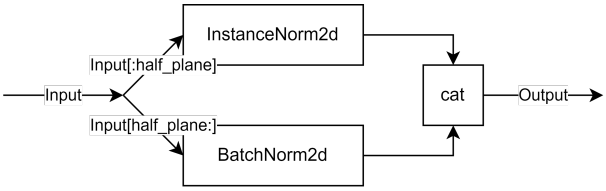

Figure 11: IBN-Module architecture

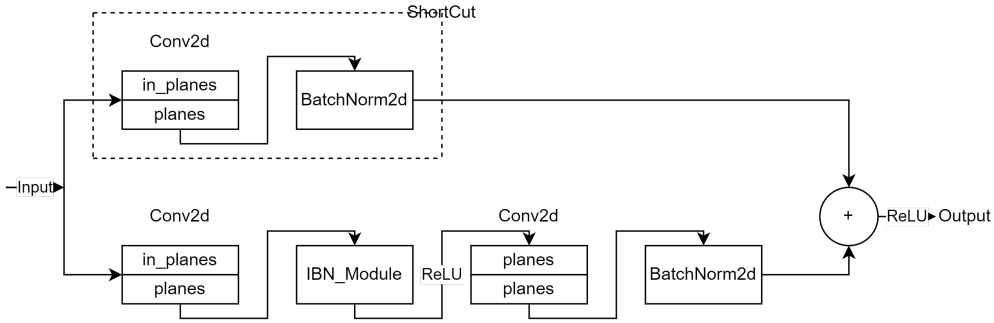

Figure 12: IBN-BasicBlock architecture

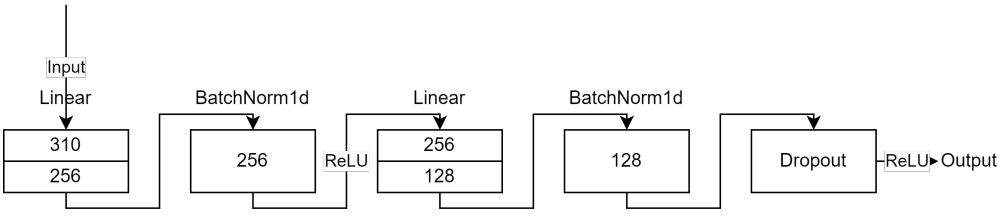

Figure 13: ADDA feature extractor (and DANN feature extractor) architecture

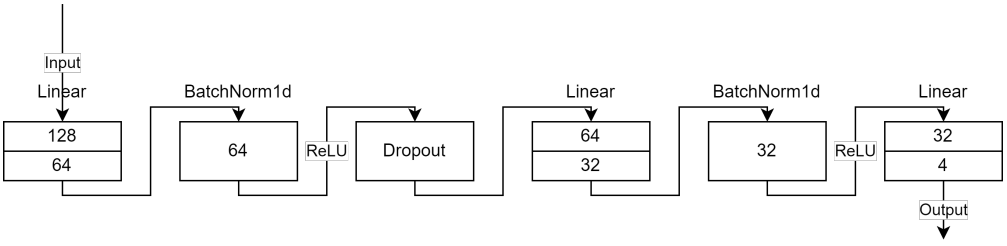

Figure 14: ADDA classifier (and DANN label classifier) architecture

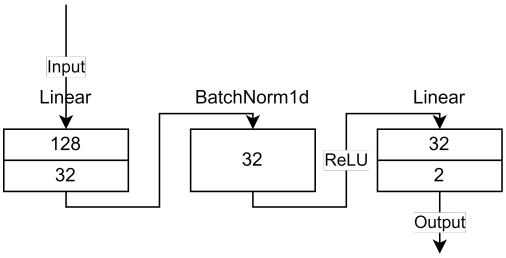

Figure 15: ADDA discriminator (and DANN domain classifier) architecture

3.4.2 DANN

In previous section, we have introduced that the DANN model mainly has three components: feature extractor, label classifier and domain classifier. We design all those components based on simple MLP network. In our implementation, DANN's feature extractor, label classifier and domain classifier share the same architecture with ADDA's feature extractor, classifier and discriminator, as in Fig.13, Fig.14 and Fig.15. Referring to [11], the training process is as pseudo-code 4 at appendix A.

4 Experiment

4.1 Experiment Setup

Our experiments are taken on Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz with GeForce GTX 1080 Ti. In order to speedup our experiments, we also implement multiprocess training with `multiprocessing` in python. Due to the limitation of VRAM (12GB), for small model like MLP, we implement 15 processes parallel, and for bigger model like CNN and ResNet18, we implement 5 processes parallel.

4.2 Experiment Results

For different transfer learning models in our project, they have different corresponding baseline models. IBN-ResNet18 is based on ResNet18, so its baseline model is ResNet18. MixStyle-CNN is based on CNN, so its baseline model is CNN. ADDA and DANN are based on MLP, so their baseline model is MLP.

For each model (except SVM), we train them with 3 different `batch size` and 3 different `learning rate` (which means 9 experiments for 1 model). Finally, we choose the best performance results as our experiment results.

Our experiment results are as below. As Fig.16 and Fig.17, we can see the best accuracy for different models with

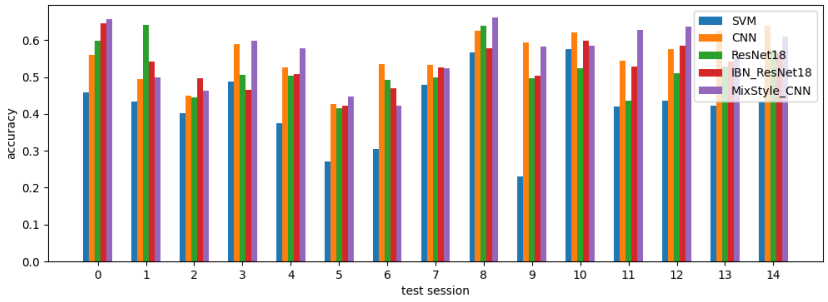

Figure 16: Domain Generalization Models Results

different test sessions. As table 1, we can see that the average accuracy and the standard deviation between different test sessions.

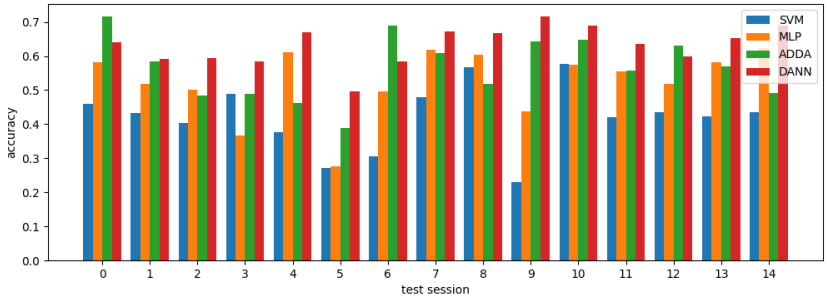

Figure 17: Domain Adaptation Models Results

Table 1: Experiment Results

Baseline					**Domain Generalization**		**Domain Adaptation**	
Model	**SVM**	**MLP**	**CNN**	**ResNet18**	**IBN-ResNet18**	**MixStyle-CNN**	**ADDA**	**DANN**
Accuracy	0.4201	0.5243	0.5565	0.5208	0.5324	0.5634	0.5657	0.6326
Standard deviation	0.0932	0.0956	0.0620	0.0655	0.0555	0.0735	0.0892	0.0549

4.3 Experiment Results Analysis

Baseline Models: According to our experiment results, it's obvious that SVM performs the worst, and MLP, CNN, ResNet18 have approximately the same performance. Unexpectedly, out naive CNN outperforms ResNet18. This phenomenon maybe because the deeper ResNet18 learns too many details from the training dataset which are harmful for making predictions on test dataset.

Domain Generalization Models: According to our experiment results, IBN-ResNet18 and MixStyle-CNN both perform better than their corresponding baseline models.

Domain Adaptation Models: According to our experiment results, ADDA and DANN both perform better than their corresponding baseline models.

Between Methods: Obviously, all our transfer learning models outperform traditional models in the baseline. However, we have also noticed that although IBN-ResNet18, MixStyle-CNN and ADDA beat their corresponding baseline models, the performance improvement is not significant (we regard the difference between DANN and MLP as a quite significant improvement). Here are some reasons might explain this problem:

1. Due to the limitation of time and computing resource, the hyper parameters maybe not the best for our models.
2. The main bottleneck of our task is not the difference between 14 domains of the training datasets, so the improvement of domain generalization models are not significant.
3. ADDA has different feature extractors for source domain data and target domain data so the training process is not as directly as DANN.

5 Conclusion

In our project, we implement 4 different transfer learning models to deal with a cross-subject emotion recognition task with SEED-IV dataset. We have implemented 2 domain generalization models (IBN-ResNet18 and MixStyle-CNN) and 2 domain adaptation models (ADDA and DANN). In the end, the DANN achieves approximately 63% accuracy, which is a significant improvement comparing with our baseline models using traditional methods.

Implementation code for our models can be found at LaGrange151235/EEG_Transfer-Learning and model files can be found at jbox.

References

[1] SEED Dataset. `https://bcmi.sjtu.edu.cn/~seed/index.html`.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Kaiyang Zhou, Yongxin Yang, Yu Qiao, and Tao Xiang. Domain generalization with mixstyle. *ArXiv*, abs/2104.02008, 2021.

[4] KaiyangZhou/mixstyle-release. `https://github.com/KaiyangZhou/mixstyle-release`.

[5] Xingang Pan, Ping Luo, Jianping Shi, and Xiaoou Tang. Two at once: Enhancing learning and generalization capacities via ibn-net. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[6] XingangPan/IBN-Net. `https://github.com/XingangPan/IBN-Net`.

[7] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[8] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

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

[10] corenel/pytorch-adda. `https://github.com/corenel/pytorch-adda`.

[11] fungtion/DANN_py3. `https://github.com/fungtion/DANN_py3`.

Appendix

A Pseudo-code

Algorithm 1 Baseline Model Training

```
TrainDataloader = DataLoader(TrainDataset)
for epoch in range(EpochNum) do
    for idx, TrainData in enumerate(TrainDataloader) do
        TrainValue, TrainLabel=TrainData
        optimizer.zero_grad()
        pred=model(TrainValue)
        loss=criterion(pred, TrainLabel)
        loss.backward()
        optimizer.step()
    end for
end for
```

Algorithm 2 MixStyle Module

```
x=input
B=x.size(0)
mu=x.mean(dim=[2,3,4], keepdim=True)
var=x.var(dim=[2,3,4], keepdim=True)
sig=(var+eps).sqrt()
mu,sig=mu.detach(),sig.detach()
x_normed=(x-mu)/sig
lmda=beta.sample((B,1,1,1,1))
perm=randperm(B)
mu2,sig2=mu[perm],sig[perm]
mu_mix=mu*lmda+mu2*(1-lmda)
sig_mix=sig*lmda+sig2*(1-lmda)
return x_normed*sig_mix+mu_mix
```

Algorithm 3 ADDA Training

```
TrainDataloader=DataLoader(TrainDataset)
for epoch in range(EpochNum) do
    for idx, TrainData in enumerate(TrainDataloader) do
        SourceValue, SourceLabel=TrainData
        optimizer.zero_grad()
        pred=Classifier(SourceFeatureExtractor(SourceValue))
        loss=criterion(pre, SourceLabel)
        loss.backward()
        optimizer.step())
    end for
end for

TestDataloader=DataLoader(TestDataset)
TargetFeatureExtractor=copy(SourceFeatureExtractor)
for epoch in range(EpochNum) do
    for idx, TrainData, TestData in enumerate(TrainDataloader, TestDataloader) do
        SourceValue, SourceLabel=TrainData
        TargetValue, TargetLabel=TestData
        DiscriminatorSourceLabel=zeros
        DiscriminatorTargetLabel=ones
        SourceFeature=SourceFeatureExtractor(SourceValue)
        TargetFeature=TargetFeatureExtractor(TargetValue)
        DiscriminatorFeature=[SourceFeature, TargetFeature]
        DiscriminatorLabel=[DiscriminatorSourceLabel, DiscriminatorTargetLabel]
        DiscriminatorPred=Discriminator(DiscriminatorFeature)
        DiscriminatorLoss=criterion(DiscriminatorPred, DiscriminatorLabel)
        DiscriminatorLoss.backward()
        DiscriminatorOptimizer.step()
        TargetFeature=TargetFeatureExtractor(TargetValue)
        TargetLabel=zeros
        TargetPred=Discriminator(TargetValue)
        TargetLoss=criterion(TargetFeature, TargetPred)
        TargetLoss.backward()
        TargetFeatureExtractorOptimizer.step()
    end for
end for
```

Algorithm 4 DANN Training

```
TrainDataloader=DataLoader(TrainDataset)
TestDataloader=DataLoader(TestDataset)
for epoch in range(EpochNum) do
    for idx, TrainData in enumerate(TrainDataloader) do
        SourceValue, SourceLabel=TrainData
        TargetValue, TargetLabel=TestData
        DomainSouceLabel=zeros
        DomainTargetLabel=ones
        SourceLabelPred, SourceDomainPred=model(SourceValue)
        _, TargetDomainPred=model(TargetValue)
        SourceLabelLoss=LabelLoss(SourceLabelPred, SourceLabel)
        SourceDomainLoss=DomainLoss(SourceDomainPred, DomainSouceLabel)
        TargetDomainLoss=DomainLoss(TargetDomainPred, DomainTargetLabel)
        loss=SourceLabelLoss+SourceDomainLoss+TargetDomainLoss
        loss.backward()
        optimizer.step()
    end for
end for
```