Project Report

Xinran Li
519030910121
Electron1@sjtu.edu.cn

Peng Tang
517020910038
tttppp@sjtu.edu.cn

Abstract

Electroencephalogram (EEG) is a recording of brain activity, which can be used to recognize emotion due to its high temporal resolution and reliability. However, the individual differences across subjects are very large, so it's difficult to build a general model to fit all subjects. Therefore, we need to train a new person-specific model for a new person, but it is time consuming and expensive to collect a large number of training data. To reduce these disadvantages, transfer learning is introduced into this field and reaches a good performance. In our project, we use three different transfer learning methods on given EEG dataset, including Style Transfer Mapping(STM), Plug-and-Play Domain Adaptation(PPDA) and Multiple Feature Spaces Adaptation Network(MFSAN). STM aims to compute a mapping matrix between source domain and target domain, then maps the target data into source domain and recognizes the mapped data through source domain classifier. MFSAN aims to align the distributions of each pair of source and target domains in multiple specific feature spaces and the outputs of classifiers by utilizing the domain-specific decision boundaries by training a deep neural network. PPDA is a method for dealing with the inter-subject variability of electroencephalography signals. In our experiments, the classification accuracy of SVM is 60%, STM is 72.2%, and MFSAN is 86.2%.

1 Introduction

Emotion plays an important role in human-human interaction(Picard [2010]). Also, in current society, emotion plays an increasing role in human-machine interaction, where emotion recognition is the key problem. There are many methods to do emotion recognition, and EEG-based emotion recognition gets great attraction due to its information sufficiency and stable neural patterns over time. However, EEG is highly subject dependent and largely varies across persons. To deal with this problem, transfer learning is applied to this filed for its adaptive features.

Transfer learning is a Machine Learning method which applies knowledge learned from previous tasks to current task, which can save a lot of time for training a new related model. Topically, we has a source domain D_S and a target domain D_T and tasks T_S in D_S and tasks T_T in D_T , where $D_S \neq D_T$ or $T_S \neq T_T$. Then, we apply the knowledge in D_S to learn knowledge in D_T. Normally, transfer learning can be divided into three categories depending on the known data label of thee source and target domains: inductive transfer learning, transductive transfer learning and unsupervised transfer learning(Wan et al. [2021]). In this project, we are going to use some transductive transfer learning methods on given dataset. What's more, because we have multiple sources and one target, we can use multiple sources transfer learning method in this project.

2 Algorithm Design

2.1 SVM

2.1.1 The principle of SVM

Support vector machines(SVM) are a set of supervised learning methods used for classification, regression and outliers detection.

SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

A hyperplane is a decision plane which separates between a set of objects having different class memberships.

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger between the classes, then it is considered to be a good margin, a smaller margin is a bad margin.

The main objective of SVM is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps:

• Generate hyperplanes which segregates the classes in the best way.
• Select the right hyperplane with the maximum segregation from the either nearest data points.

However, some problems can't be solved using linear hyperplane.

In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space.

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. Kernel trick helps to build a more accurate classifier.

• A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

• A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

• The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

2.1.2 SVM in the task

From the principle part, we can know several advantages of SVM:

• Effective in high dimensional spaces.

• Still effective in cases where number of dimensions is greater than the number of samples.

• Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

In this task, the data is in a high dimensional space, so SVM may perform well. Therefore, we can use SVM to form a baseline, which will be compared to the transfer learning methods.

From the three kernel, we choose the poly kernel and the rbf kernel, and we set different parameters to obtain three SVM models in total. The results of them will be shown in **Section** 3.

2.2 STM

Style transfer mapping(STM)(Zhang and Liu [2013]) is an effective TL method that achieved state-of-the-art performance in style transfer tasks. The object of STM is to transform source points toward target points. There are three different STM methods, Supervised STM, Unsupervised STM and Semi-Supervised STM. Supervised STM needs some labeled target data, Unsupervised STM only needs unlabeled target data and Semi-Supervised STM needs both labeled target data and unlabeled target data. We mainly focus on Supervised STM here, which means we need to use some labeled data of the target subject to compute the mapping matrix. Figure 1 shows the main idea about STM used in our project. We have multiple source domains(S_i) and one target domain(T). In order to recognize the target, we map it to each source independently at first. Then we use corresponding classifier(C_i) to recognize it. Finally we combine all results together with different weights(w_i) to give the final result(Y). Obviously, the main task is to find out the mapping parameters(A_i, b_i).

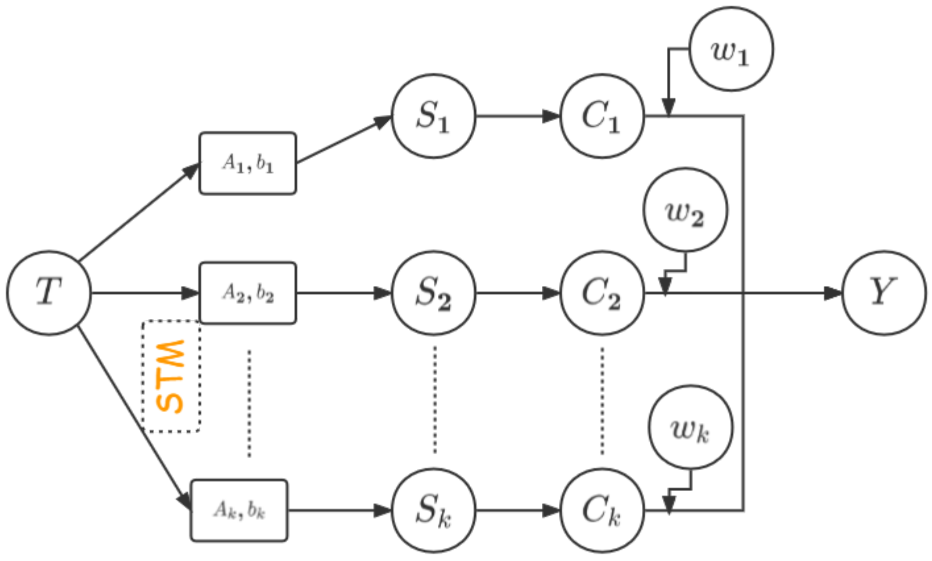

Figure 1: STM Model

Table 1: STM Notation

Notation	Meaning
S	source data, $S = \{S_k, k = 1, \cdots, N\}$
C	source classifiers, $C = \{C_k, k = 1, \cdots, N\}$
T^L	labeled target data, $\{x_j^m, y_j^m\}_{j=1}^n$
T^U	unlabeled target data, $\{x_j^m, y_j^m\}_{j=n+1}^u$
T	target data, $T^L \cup T^U$

For convenience, we use the notations in Table1 to help illustrate the computing process of STM. We are going to map T to S_k to bridge the two distributions. For more general, we don't map the points in T to S_k directly, but to some representational patterns in S_k(e.g. clusters of KMeans)(Li et al. [2020]). In our project, we call the representational patterns in S_k the *destination*, and points in T the *origin*.

Suppose we have *a destination point set*

$$D = \{d_i \in R^m | i = 1, \cdots, n\}$$

Where D is composed of the representational patterns in S_k. Then we have a one-to-one($o_i \leftrightarrow d_i$) correspondent mapping set, the *origin point set*

$$O = \{o_i \in R^m | i = 1, \cdots, n\}$$

Where O is composed of some points in T.

This change between O and D is known as concept drift. Suppose d_i is transformed to o_i with *confidence* $f_i \in [0, 1]$, then we can learn the parameters of style transfer mapping $A \in R^{m\times m}$ and $b \in R^m$ from the correspondence between D and O by minimizing the weighted squared error:

$$\min_{A\in R^{m\times m}, b\in R^m} \sum_{i=1}^{n} f_i ||Ao_i + b - d_i||_2^2 \tag{1}$$

To avoid overtransfer, we can constrain the deviation of A from I and the deviation of b from the zero vector. Then, we can get the new objective of STM:

$$\min_{A\in R^{m\times m}, b\in R^m} \sum_{i=1}^{n} f_i ||Ao_i + b - d_i||_2^2 + \beta||A - I||_F^2 + \gamma||b||_2^2 \tag{2}$$

where $||\cdot||_F$ is the matrix Frobenius norm and $||\cdot||_2$ is the vector $L2$ norm. the parameters β and γ control the tradeoff between style transfer and non transfer. We can set them as

$$\beta = \hat{\beta}\frac{1}{m}Tr(\sum_{i=1}^{n} f_i o_i o_i^T), \quad \gamma = \hat{\gamma}\sum_{i=1}^{n} f_i \tag{3}$$

where $Tr(\cdot)$ is the trace of a matrix. The parameters $\hat{\beta}$ and $\hat{\gamma}$ can be selected from $[0, 3]$.

Because Equation 2 is a convex quadratic programming problem, then we can solve it as

$$A = QP^{-1}, b = \frac{1}{\hat{f}}(\hat{d} - A\hat{o}) \tag{4}$$

where

$$Q = \sum_{i=1}^{n} f_i d_i o_i^T - \frac{1}{\hat{f}}\hat{d}\hat{o}^T + \beta I \tag{5}$$

$$P = \sum_{i=1}^{n} f_i d_i d_i^T - \frac{1}{\hat{f}}\hat{o}\hat{o}^T + \beta I \tag{6}$$

$$\hat{o} = \sum_{i=1}^{n} f_i o_i, \quad \hat{d} = \sum_{i=1}^{n} f_i d_i \tag{7}$$

$$\hat{f} = \sum_{i=1}^{n} f_i + \gamma \tag{8}$$

In order to solve the Equation 4, we need to define the origin point set O, the destination point set D and the corresponding confidence f_i. In our project, we use the T^L as O, the set of representational patterns in S_k as D. And we set f_i equals 1 because we already have the labels of x_i in O.

We perform Kmeans-clustering on the data in each class of S_k to obtain patterns

$$p_{ij} \in R^m, j = 1, \cdots, n_i, i = 1, \cdots, M \tag{9}$$

where n_i is the number of patterns in class i, M is the number of class(3 in our project). Then we define the destination point of a point x in O

$$dest(x, y) = p_{ij}, \quad \text{where} \quad i = y \quad \text{and} \quad j = \arg\min_{j'=1}^{n_i} ||x - p_{ij'}||_2^2 \tag{10}$$

where y is the label of x.

With above discussion, we can obtain the Supervised STM algorithm as Algorithm 1

Algorithm 1 Supervised STM

```
Require: $T^L = \{x_i, y_i\}_{i=1}^n$, $T^U = \{x_i\}_{i=n+1}^u$, $S_k(k = 1, \cdots, N)$, hyper-parameters $\hat{\beta}, \hat{\gamma}$, weight
    of each source $w_k(k = 1, \cdots, N)$, classifier of each source $C_k(k = 1, \cdots, N)$
Ensure: The predicted labels of $T^U$: $\{y_i\}_{i=n+1}^u$
 1: for $i = 1$ to $n$ do
 2:     origin $o_i = x_i, x_i \in T^L$
 3:     confidence $f_i = 1$
 4: end for
 5: compute $\gamma, \beta$ through Equation 3
 6: for $k = 1$ to $N$ do
 7:     use K-means to find patterns $P = p_{ij}$
 8:     for $i = 1$ to $n$ do
 9:         use Equation 10 to compute destination $d_i = dest(x_i, y_i)$
10:     end for
11:     learn STM $\{A_k, b_k\}$ with $\{o_i, d_i, f_i\}$ through Equation 4-8
12:     compute mapping values through $\{A_k, b_k\}$: $x_i' \leftarrow A_k x_i + b_k, i = n + 1, \cdots, u$
13:     for $i = n + 1$ to $u$ do
14:         prediction $y_i^k = C_k(x_i')$
15:     end for
16: end for
17: for $i = n + 1$ to $u$ do
18:     prediction $y_i = \sum_{k=1}^N w_k y_i^k$
19: end for
```

2.3 PPDA

Plug-and-play domain adaptation(PPDA) (Zhao et al. [2021]) is a method for dealing with the inter-subject variability of electroencephalography signals, which make a setback in human emotion decoding in affective brain-computer interfaces.

Let's imagine a scene in the near future. A person wear a device, which serves as a brain-computer interface. After a few seconds for adapting, he can use the device to express himself. A few seconds for adapting in the scene just mean the domain adaptation in transfer learning. And for practical use, we should have a plug-and-play algorithm. Therefore, we consider a plug-and-play domain adaptation.

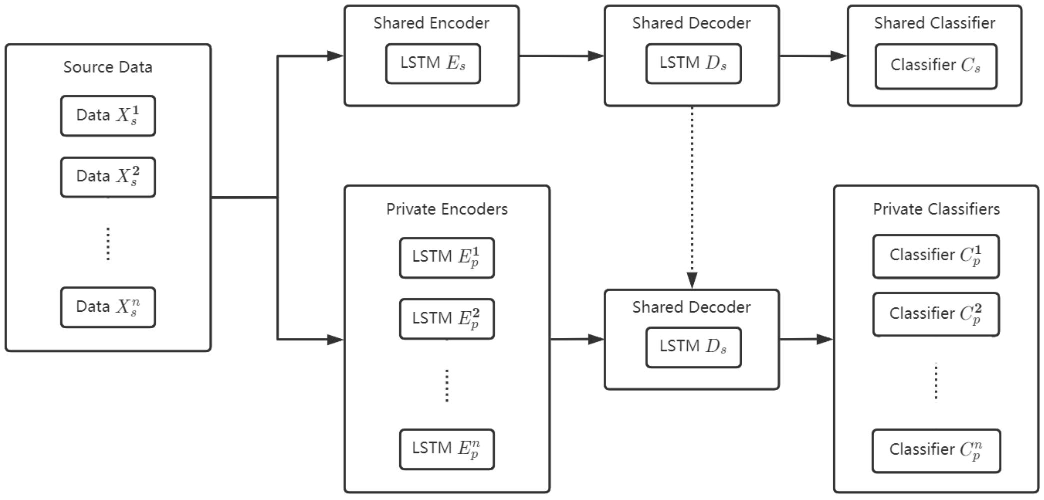

Figure 2: PPDA Training Process

In the algorithm, we use a shared classifier C_s and some private classifiers C_p^i. The shared classifier is composed of a shared encoder E_s and a shared decoder D_s. The private classifiers C_p^i are composed of private encoders E_p^i and the shared decoder D_s.

What's more, we use LSTM to make these encoders and decoders, because the EEG data is continuous in every person, and we use l=4 in every LSTM. Some equations in LSTM are shown as following, and we just use the ready-made LSTM model in Pytorch.

$$
\begin{aligned}
&i = \sigma(W_{ii}x + b_{ii}W_{hi}h + b_{hi}) \\
&f = \sigma(W_{if}x + b_{if} + W_{hf}h + b_{hf}) \\
&g = \tanh(W_{ig}x + b_{ig} + W_{hg}h + b_{hg}) \\
&o = \sigma(W_{io}x + b_{io} + W_{ho}h + b_{ho}) \\
&c' = f * c + i * g \\
&h' = o * \tanh(c')
\end{aligned}
$$

In the training process, we train the shared classifier firstly. Then we have the trained shared decoder, which can be used in the private classifiers. Secondly, we train the private classifiers as is shown in the figure. After all these, we have finished the training process.

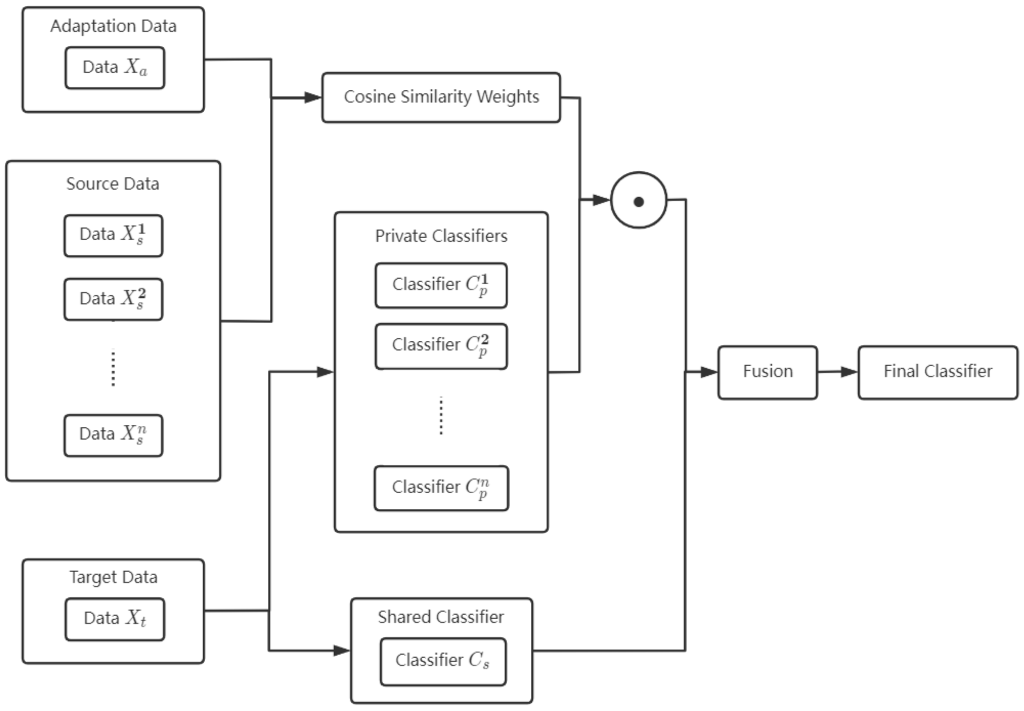

Figure 3: Test Process

In the test process, we use private classifiers and the shared classifier to classify the test data. And we use cosine similarity to compute the similarity weights between adaptation data and the source data. Then we can combine the weights, the results from private classifiers and the shared classifier.

After all these, we can get the final result, which shows the predicted emotion by the Plug-and-Play Domain Adaptation.

We can obtain the PPDA algorithm as Algorithm2.

Algorithm 2 Plug-and-play Domain Adaptation

```
Require:
    Source data $X_s^i (i = 1, \cdots, n)$.
    Adaptation data $X_a$.
    Test data $X_t$.
Ensure: Final predicted result
 1: Randomly initialize $E_p^{1\sim n}$, $E_s$, $D_s$, $C_p^{1\sim n}$, $C_s$.
 2: Optimize $E_s$, $D_s$, $C_s$.
 3: for $j = 1$ to $n$ do
 4:     Optimize $E_p^j$, $C_p^j$ by minimizing the loss
 5: end for
 6: Calculate the similarity weight $w_s$ between $X_a$ and $X_s^{1\sim n}$
 7: Prediction of weighted private classifiers:
    $y_p = w_s \cdot C_p^{1\sim n}(E_p(X_t) + E_s(X_t))$
 8: Prediction of shared classifier:
    $y_s = C_s(E_s(X_t))$
 9: Integrate predictions: $y = CF(y_p, y_s)$
10: return $y$
```

2.4 MFSAN

Multiple Feature Spaces Adaptation Network(MFSAN)(Zhu et al. [2019]) is a Multi-source Unsupervised Domain Adaptation(MUDA) algorithm, which aligns the distributions of each pair of source and target domains in multiple specific feature spaces and the outputs of classifiers by utilizing the domain-specific decision boundaries.

Suppose we have labeled source domain data $\{X_j^s, Y_j^s\}_{j=1}^N$ sampled from N different source distributions and unlabeled target domain data $X^t = \{x_i^t\}_{i=1}^n$ sampled from one target distribution. Then we are going to use MFSAN to deduce the labels of X^t by learning knowledge from $\{X_j^s, Y_j^s\}_{j=1}^N$ and X^t.

The structure of MFSAN is a two-stage alignment Framework, consisting of three components, namely a common feature extractor, domain-specific feature extractors, and domain-specific classifiers. Figure 4 shows the structure.

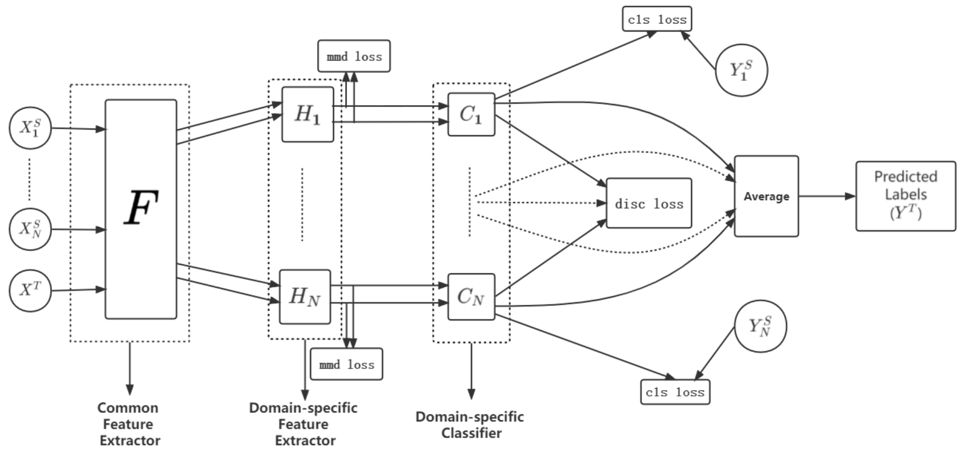

Figure 4: MFSAN Framework

Common feature extractor is a common subnetwork $f(\cdot)$, which can extract common features for all domains, mapping original feature space into a common feature space.

Domain-specific feature extractor consists of N specific subnetworks $h_j(\cdot)$, which can map each pair of source and target domain domain into a specific feature space. In order to align the distributions for each pair of source and target domains, we choose Maximum Mean Discrepancy(MMD)(Gretton et al. [2012]) as the estimate of the discrepancy between two domains. Formally, MMD is defined as

$$D_{\mathcal{H}}(p,q) \triangleq ||\mathbb{E}_p[\phi(\mathtt{x}^s)] - \mathbb{E}_q[\phi(\mathtt{x}^t)]||^2_{\mathcal{H}} \tag{11}$$

where $\mathcal{H}$ is the reproducing kernel Hillbert space (RKHS) endowed with a characteristic kernel k; $\phi(\cdot)$ is denotes some feature map to map the original samples to $RKHS$ and the kernel k means $k(\mathtt{x}^s, \mathtt{x}^t) = \langle \phi(\mathtt{x}^s), \phi(\mathtt{x}^t) \rangle$, where $\langle \cdot, \cdot \rangle$ is inner product of vectors. In practice, an estimate of the MMD is

$$\hat{D}_{\mathcal{H}}(p,q) = ||\frac{1}{n_s}\sum_{x_i \in \mathcal{D}_s} \phi(x_i) - \frac{1}{n_t}\sum_{x_j \in \mathcal{D}_t} \phi(x_j)||^2_{\mathcal{H}} \tag{12}$$

where $\hat{D}_{\mathcal{H}}(p,q)$ is the unbiased estimator of $D_{\mathcal{H}}(p,q)$. Then, we can define the MMD loss as

$$\mathcal{L}_{mmd} = \frac{1}{N}\sum_{j=1}^{N} = \hat{D}(H_j(F(X^s_j)), H_j(F(X^t))) \tag{13}$$

Domain-specific classifier consists of N domain-specific predictors $\{C_j\}_{j=1}^N$. Each predictor C_j is a softmax classifier and use $H(F(x))$ as input. For each classifier, we can compute the cross entropy loss

$$\mathcal{L}_{cls} = \sum_{j=1}^{N} \mathbb{E}_{x \sim X^s_j} J(C_j(H_j(F(x_i^{sj}))), y_i^{sj}) \tag{14}$$

where $J(\cdot)$ is the cross entropy function.

What's more, because the same target predicted by different specific-domain classifiers should have the same prediction, we align $\{C_j(H_j(F(X^t)))\}_{j=1}^N$ to minimize the discrepancy by the following loss

$$\mathcal{L}_{disc} = \frac{2}{N(N-1)} \sum_{j=1}^{N-1}\sum_{i=j+1}^{N} \mathbb{E}_{x\sim X^t}[|C_i(H_i(F(x_k))) - C_j(H_j(F(x_k)))|] \tag{15}$$

With Equation 13 14 15, we can define the total loss of this network as

$$\mathcal{L}_{total} = \mathcal{L}_{cls} + \gamma\mathcal{L}_{mmd} + \beta\mathcal{L}_{disc} \tag{16}$$

By minimizing $\mathcal{L}_{total}$, we can train the network.

Then, we can have the MFSAN algorithm as Algorithm 3

Algorithm 3 MFSAN Algorithm

```
Require: labeled source domain data $\{X^s_j, Y^s_j\}_{j=1}^N$ and unlabeled target domain data $X^t = \{x_i^t\}_{i=1}^n$
Ensure: Trained Network
1: Initialize the Common feature extractor $F$, Domain-specific feature extractor $\{H_j\}_{j=1}^N$ and
   Domain-specific classifier $\{C_j\}_{j=1}^N$
2: Initialize the training iterations $T$
3: for $t = 1$ to $T$ do
4:    Randomly sample $m$ items $\{x_i^t\}_{i=1}^m$ from $(X^t)$
5:    for $j = 1$ to $N$ do
6:       Randomly sample $m$ items $\{x_i^{sj}, y_i^{sj}\}_{i=1}^m$ from $(X^s_j, Y^s_j)$
7:       Input $\{x_i^t\}_{i=1}^m$ and $\{x_i^{sj}, y_i^{sj}\}_{i=1}^m$ to $F$ to get the common latent representations $\{F(x_i^t)\}_{i=1}^m$
         and $\{F(x_i^{sj})\}_{i=1}^m$
8:       Input $\{F(x_i^{sj})\}_{i=1}^m$ to $H_j$ to get the domain-specific representations of source samples
         $\{H_j(F(x_i^{sj}))\}_{i=1}^m$
9:       Input $\{H_j(F(x_i^{sj}))\}_{i=1}^m$ to $C_j$ to get $\{C_j(H_j(F(x_i^{sj})))\}_{i=1}^m$
```

```
10:         Input $\{F(x_i^t)\}_{i=1}^m$ to $\{H_j\}_{j=1}^N$ to get domain-specific representations of target samples
            $\{H_1(F(x_i^t))\}_{i=1}^m, \cdots, \{H_N(F(x_i^t))\}_{i=1}^m$
11:         Input $\{H_1(F(x_i^t))\}_{i=1}^m, \cdots, \{H_N(F(x_i^t))\}_{i=1}^m$ to $\{C_j\}_{j=1}^N$ to get $\{C_1(H_1(F(x_i^t)))\}_{i=1}^m$
            $, \cdots, \{C_N(H_N(F(x_i^t)))\}_{i=1}^m$
12:         Use $\{H_j(F(x_i^{sj}))\}_{i=1}^m$ and $\{H_j(F(x_i^t))\}_{i=1}^m$ to compute the mmd loss $\mathcal{L}_{mmd}$ through
            Equation13
13:         Use $\{C_j(H_j(F(x_i^{sj})))\}_{i=1}^m$ and $\{y_i^{sj}\}_{i=1}^m$ to compute the cls loss $\mathcal{L}_{cls}$ through Equation 14
14:         Use $\{C_1(H_1(F(x_i^t)))\}_{i=1}^m, \cdots, \{C_N(H_N(F(x_i^t)))\}_{i=1}^m$ to compute the disc loss $\mathcal{L}_{disc}$
            through 15
15:         Compute the total loss $\mathcal{L}_{total}$ through 16
16:         Update the the Common feature extractor $F$, Domain-specific feature extractor $\{H_j\}_{j=1}^N$
            and Domain-specific classifier $\{C_j\}_{j=1}^N$ by minimizing $\mathcal{L}_{total}$
17:     end for
18: end for
```

3 Experiment And Result

3.1 Data Description

Database "SEED" is used in our project. Fifteen Chinese film clips(positive, neutral and negative emotions) were chosen to evoke subjects' emotions in the experiments. There is a total of 15 trials for each experiment. There was a 5s hint before each clip, 45s for self-assessmet and 15s to rest after each clip in one session. And two trials with same emotion won't be arranged consecutively. For feedback, the participants reported their emotions immediately after watching each clip. The Figure 5 shows the process

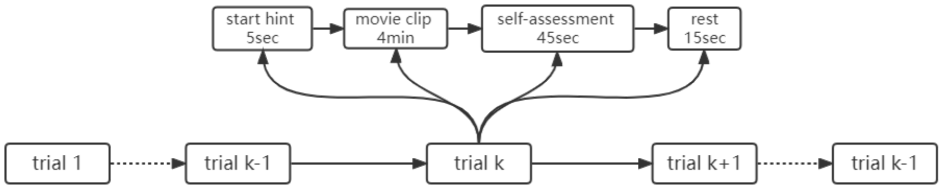

Figure 5: Data Collection Procedure of Each Participant

There are 15 subjects participated in the experiments. EEG signals were collected with the 62-channel ESI NeuroScan System. The data was downsampled to 200Hz. A bandpass frequency filter from $0 - 75$ Hz was applied. The data we use in this project is the extracted differential entropy(DE) features of the EEG signals. In detailed, the feature dimension is $310(62 \times 5)$ and the total number of features of one subject(15 sessions) is 3394. And labels are $-1, 0, 1$, representing sad, smooth, and happy.

3.2 SVM Experiment

We are going to implement SVM on the dataset to get a baseline which will be compared with other algorithms.

Table 2: SVM Experiment Results

kernel=poly, C=0.5	Target Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Mean
	Accuracy	0.355	0.395	0.462	0.799	0.449	0.594	0.620	0.435	0.345	0.437	0.921	0.687	0.666	0.665	0.794	0.575
kernel=rbf, C=0.7	Target Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Mean
	Accuracy	0.347	0.617	0.511	0.758	0.537	0.610	0.685	0.594	0.568	0.429	0.813	0.516	0.900	0.626	0.695	0.614
kernel=rbf, C=1.0	Target Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Mean
	Accuracy	0.325	0.582	0.498	0.791	0.516	0.611	0.686	0.627	0.536	0.429	0.822	0.506	0.913	0.614	0.727	0.612

From the Table 2, we can see that different kernels and parameters will cause different results, but they are similar in total. The accuracy of SVM is about 60%.

3.3 STM Experiment

We are going to implement the Algorithm 1 on the dataset and find a good model by turning the parameters.

1. **Data Split**: We choose one subject as the target, and other fourteen subjects as the sources. And we choose the first three sessions(one for each emotion) as the labeled data(T^L) and the last twelve sessions as the test data(T^U).
2. **Source Selection**: The number of sources is an important factor in multiesource TL. The correlation between some subjects is very weak, so we need to choose suitable sources. In order to do that, we use the classifiers of subjects to predict the T^L and sort the order of subjects according to the prediction accuracy. Then we can choose the first N subjects as the sources. We searched for the best N from 1 to 14, and got the result as Figure 6. According to the result, we set $N = 13$ in our project.

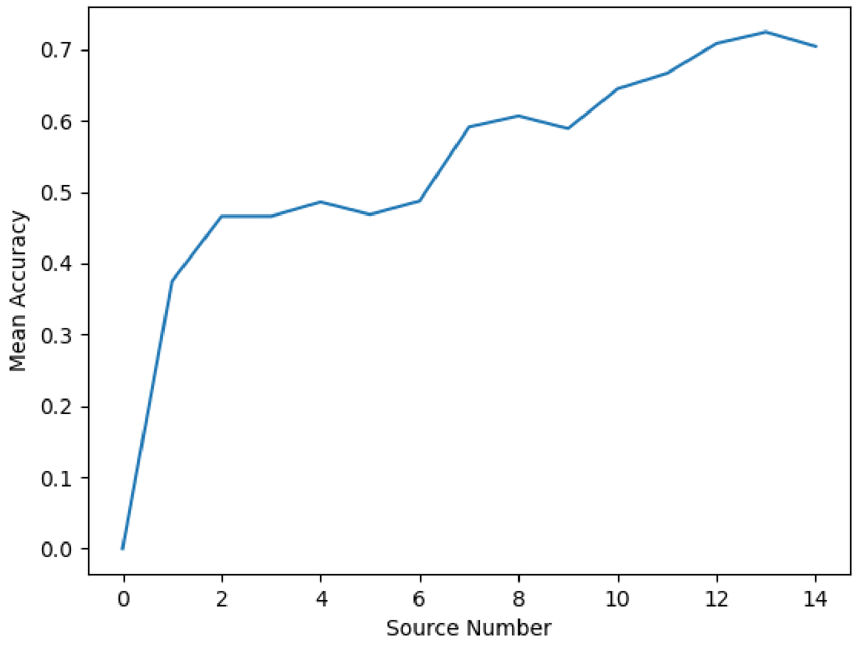

Figure 6: STM Source Selection

3. **Classifier**: To get good classifiers, we use deep neutral network(DNN) as the model. Especially, we use three linear layers and two activate function layers(*RELU()*) to construct the model. By training the DNN with data of each subject, we gain 15 classifiers.
4. **Parameters Setting**: We search for the best $\hat{\gamma}$ and $\hat{\beta}$ bewteen 0 and 3. The searching step is 0.2. We set $\hat{\gamma} = 1.2$ and $\hat{\beta} = 0.2$ after searching.

After doing above work, we get our final results as Table 3 by doing 15-fold cross-validation.

Table 3: STM Experiment Result

Target Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Mean
Accuracy	0.797	0.688	0.681	0.823	0.496	0.692	0.711	0.618	0.749	0.739	0.783	0.588	0.964	0.678	0.824	0.722

3.4 MFSAN Experiment

We implement algorithm 3 with following special settings:

1. **Data Reshaping**: Because the feature of data is collected from 62 channels, we reshape the vector x(length is 310) to an image with size 62×5, which is a better feature of EEG.
2. **Network Construction**: In our work, we choose the pretrained **ResNet18**(He et al. [2015]) as the Common feature extractor because it has a great advantage on image recognition. What's more, we use 14 small DNNs(3 linear layers and 2 activate function layers($Relu()$)) as Domain-specific feature extractor and 14 small DNNs(1 linear layer and 1 activate function layer($Softmax()$)) as Domain-specific classifier.

After training the network for enough episodes, we test it by doing 15-fold cross-validation and get the results as Table 4.

Table 4: MFSAN Experiment Result

Target Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Mean
Accuracy	0.704	0.859	0.964	0.951	0.789	0.741	0.764	0.903	0.841	0.864	0.901	0.764	0.947	0.938	1.0	0.862

4 Discussion

From above experiments, we can see that both MFSAN and Supervised STM perform better than the baseline model, SVM, which confirms that transfer learning can improve the accuracy of EEG recognition. In Supervised STM, we need to use some labeled data in target domain, while we need a lot of unlabeled data in MFSAN. Although MFSAN performs much better than STM in our experiment, STM spends less time to find the mapping matrix while MFSAN needs more time to train the network. Therefore, two methods have their own advantages and disadvantages.

In Supervised STM, we try different number of labeled data to compute the mapping matrix and we find less labeled data(e.g. 30 for each emotion) can also give the same performance as the total data of the first three sessions. We can also try other methods to compute the destination of the point in target domain, such as Gaussian Model(Li et al. [2020]).

In MFSAN, from Table 4, we can see that most experiments give good results, while several give bad results, such as 1, 6, and 7, which means the distribution of these domains has weaker relationship with others. Therefore, we can also select some source domains to train the network like STM, not all, which may gain better performance. To select suitable source domains for target domain, maybe we can use the mmd loss $\mathcal{L}_{mmd}$ as a criterion(less means better). Besides mmd loss, we can also use other methods to compute the loss between source domain and target domain, such as DIP(Baktashmotlagh et al. [2013]).

What's more, we also try the Semi-Supervised STM and plug-and-play domain adaptation(PPDA) , but we failed to implement them. For Semi-Supervised STM, we implement it as the description of Li et al. [2020], but we don't gain good performance. This may be because we don't compute the deduced labels of target data rightly, but we can't find out the mistake in our implementation. For PPDA, the reason may be that the network has some mistakes, which gives rise to some logical mistakes in the model, and we should pay more attention to this in the future.

5 Conclusion

In this project, we use different transfer learning methods, Supervised STM, Semi-Supervised STM, PPDA and MFSAN, to recognize the EEG data and gain better results than SVM. And the results also show that deep neural network is a powerful tool in EEG recognition, especially in Multi-source domain adaptation. Totally, EEG recognition is a complex problem, which needs more attention.

6 Roles of The Team

1. Peng Tang: Implementation of MFSAN and STM; Report Writing(50%).

2. Xinran Li: Implementation of SVM and PPDA; Report Writing(50%).

References

Mahsa Baktashmotlagh, Mehrtash T Harandi, Brian C Lovell, and Mathieu Salzmann. Unsupervised domain adaptation by domain invariant projection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 769–776, 2013.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13(1):723–773, March 2012. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2503308.2188410`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Jinpeng Li, Shuang Qiu, Yuan-Yuan Shen, Cheng-Lin Liu, and Huiguang He. Multisource transfer learning for cross-subject eeg emotion recognition. *IEEE Trans. Cybern.*, 50(7):3281–3293, 2020. URL `http://dblp.uni-trier.de/db/journals/tcyb/tcyb50.html#LiQSLH20`.

Rosalind W Picard. Affective computing: from laughter to ieee. *IEEE Transactions on Affective Computing*, 1(1):11–17, 2010.

Zitong Wan, Rui Yang, Mengjie Huang, Nianyin Zeng, and Xiaohui Liu. A review on transfer learning in eeg signal analysis. *Neurocomputing*, 421:1–14, 2021.

Xu-Yao Zhang and Cheng-Lin Liu. Writer adaptation with style transfer mapping. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(7):1773–1787, 2013. URL `http://dblp.uni-trier.de/db/journals/pami/pami35.html#ZhangL13`.

Li-Ming Zhao, Xu Yan, and Bao-Liang Lu. Plug-and-play domain adaptation for cross-subject eeg-based emotion recognition. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*. sn, 2021.

Yongchun Zhu, Fuzhen Zhuang, and Deqing Wang. Aligning domain-specific distribution and classifier for cross-domain classification from multiple sources. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:5989–5996, jul 2019. doi: 10.1609/aaai.v33i01.33015989. URL `https://doi.org/10.1609%2Faaai.v33i01.33015989`.