ITCS 6156 - Machine Learning
Dr. Minwoo Jake Lee

Project Final Report
Brain Computer Interface: Meta Learning for ERN Detection

A Project Report
Submitted by:-

Group 10 - NeuroX

Abhilash Mandlekar	(801136686)
Anushka Tibrewal	(801134231)
Jeet Jivrajani	(801134130)
Kruti Raval	(801135835)

At

University of North Carolina at Charlotte

Introduction

Brain-Computer Interface (BCI) is a way to connect the brain to an external device in order to send or receive information directly from it. Currently there are several commercial BCI's that are routinely used by thousands of people, such as deep brain stimulators that alleviate Parkinson's Disease symptoms, cochlear implants that restore hearing and and even retinal implants that restore vision[1]. Using BCI techniques it helps sending thoughts and even emotions to your friends and family wirelessly from anywhere in the world[2].

There are many applications of using Brain Computer Interface (BCI)[2]

1. Access information and data from the Internet on the fly.
2. Upload your memories.
3. Download knowledge and possibly skill sets.
4. Have a cognitive AI assistant to aid you in decision making and task management.
5. Being able to capture the ideas into words, images, or even videos in real time, without recreating them using software.

Problem Statement

Applying meta learning to BCI data. More specifically, we wish to take epoched signal recordings containing error-related negativity (ERN) and classify them as ERN or no ERN. The main goal is to see if a simple task containing ERN can translate to a more complicated task containing ERN.

Approach

Classification of Brain Signals are done using siamese network. The ERN signals are identified and classified. The two inputs are taken into consideration at the same time and their similarity scores are calculated. Based on the similarity scores, we predict the class of the data. The less data intensive approach makes it more effective as well as efficient when we want to achieve high accuracy with less data.

Steps

1. Installation and Configuring setup
2. Siamese Network on MNIST dataset
3. Real time data collection of EEG Signals
4. Preprocessing data
5. Cross task generalization
6. Validating accuracy
7. Difference in the methodology

Motivation and Challenges

The main challenge was to train on actual brain signal data. The data for actual brain signals required lot of preprocessing and cleaning. The next challenge was to train the model quickly with few samples of available data. To perform this we have studied different advance classification algorithms like Prototypical networks, EEG Net, and Siamese Network.

We chose Siamese network because it gave us good performance while training, that is the model gets trained using less number of samples. It internally uses one shot learning and gives good accuracy. To initialize the model and configure it with Siamese Network architecture was another challenge we faced during the implementation.

Concise summary of the solution

We have used Siamese Network that uses one shot learning. We studied how to pair one example of each for train and test sample and also studied the architecture of Siamese network. We implemented base model of Siamese network on MNIST dataset. After achieving the desired accuracy on the MNIST data we fine tuned the model to fit on our actual brain signal data.

Background

Deep survey of literature

Siamese neural Network for One-shot Image Recognition

In this era, Deep Convolutional Neural Network has become the prominent in classification of the Image. The major constraint regarding the classification of the images is that it requires huge amount of the labelled data. The situation doesn't guarantees the tons of labelled data for all scenarios. So, an alternative solution is to be designed such that it requires the few samples of the data and provides the accurate classification. This problem can be solved by using One-Shot Learning. In One shot classification, there is one single image per class for the training set. The model makes a prediction based on the single image per class. One shot learning can also be addressed by developing domain-specific features that have discriminating properties for the test task. The network takes two images as input and generate the similarity score. The Similarity Score denotes how similar the two input images are with each other.

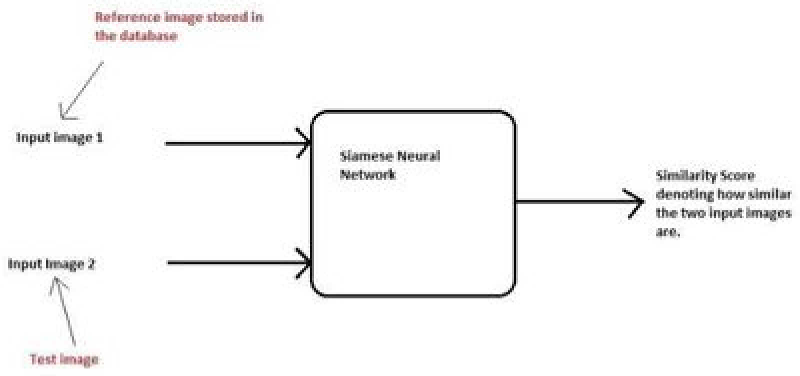

Reference image stored in the database
Input image 1
Input Image 2
Test image
Siamese Neural Network
Similarity Score denoting how similar the two input images are.

Figure 1. Block diagram of One-shot Classification

A Siamese Neural Network consists of a twin network which takes two distinct or similar input. The standard model is a Siamese convolutional neural network of 'L' layers with Nl units, where h(1,l) represents the hidden vector in layer l for the first twin and h(2,l) denotes the same for second twin.

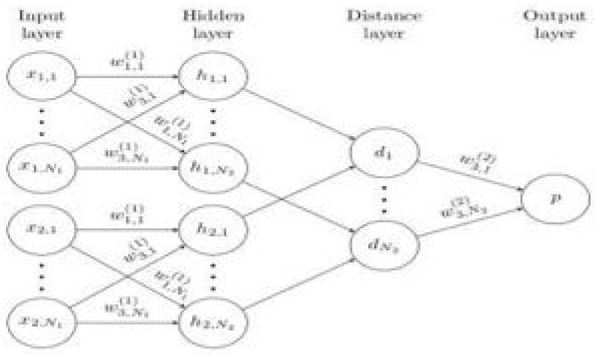


Figure 2. Siamese Network for Binary Classification

The model is a siamese convolutional neural network with L layers each with N1 units, where h(1,l) represents the hidden vector in the layer l of the first twin and h(2,l) denotes the second twin. It uses ReLU units in the first L-2 layers and sigmoidal in the remaining layers. As the ReLU is the non linear function so it can easily back propagate the error and have multiple neurons activated. The disadvantage of the It uses a sequence of convolutional layers each of which uses a single channel with filters of varying size and the fixed stride of 1. The number of convolutional filter is specified as multiples of 16 to optimize performance.(This has been observed that multiples of 16 has increased the performance of the model). Then the network used ReLU activation function to output feature map and it is followed by max pooling with stride of 2. The kth filter map in each layer takes the following form:

$$a^{(k)}{}_{1.m} = max - pool(max(0, W^{(k)}{}_{l-1,l} * h_{1,l-1} + b_l), 2)$$
$$a^{(k)}{}_{1.m} = max - pool(max(0, W^{(k)}{}_{l-1,l} * h_{2,l-1} + b_l), 2)$$

$W_{l-1,l}$ = tensor of feature maps and * signifies the valid convolution.

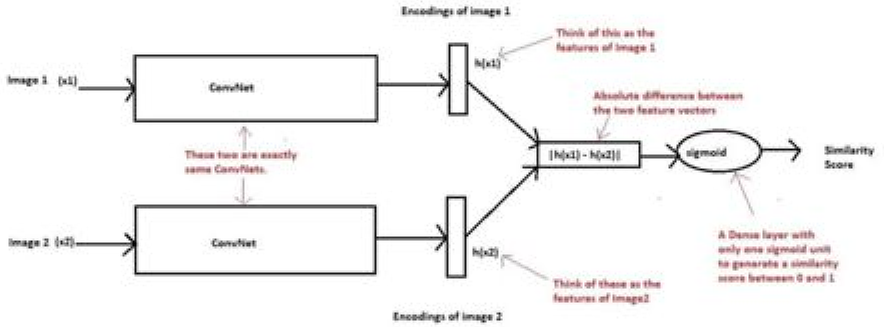

Encodings of image 1
Think of this as the features of image 1
Image 1 (x1)
ConvNet
h(x1)
Absolute difference between the two feature vectors
These two are exactly same ConvNets.
|h(x1) - h(x2)|
sigmoid
Similarity Score
Image 2 (x2)
ConvNet
h(x2)
A Dense layer with only one sigmoid unit to generate a similarity score between 0 and 1
Think of these as the features of image2
Encodings of image 2

Figure 3. A high level architecture

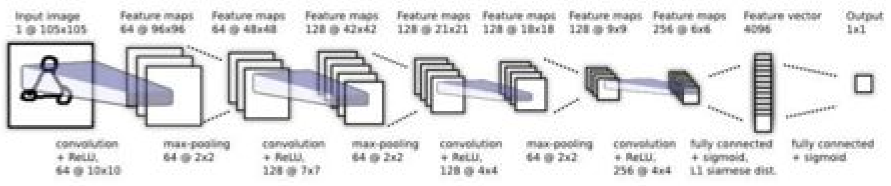


Figure 4. Deep ConvNet Architecture

Loss function- Regularized Cross Entropy loss is used in the given function.

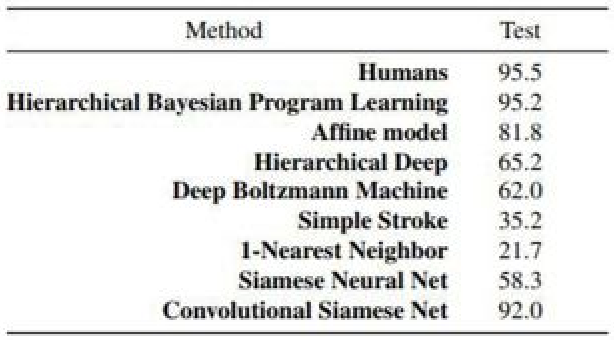
Weight initialization- The model weights are initialized in the convolutional layer from a normal distribution with mean zero and standard deviation 0.01. Biases were initialized with a normal distribution but mean of 0.5 and standard deviation 0.01. But in the fully connected layer the biases are initialized with a normal distribution having mean zero and standard deviation 0.2.

Optimization- The optimizer that is used is momentum where a mini-batch size of 128 and learning rate n_j

Learning schedule- The model used different learning rates for each layer, but learning rates are decayed uniformly by 1 percent per epoch. The momentum starts at 0.5 in each layer and linearly increases until it reaches the value uj. The model was trained for 200 epochs but monitored one-shot validation error on a set of 320 one-shot learning tasks generated randomly from validation set.

Hyperparameter optimization- The model has used beta version of Whetlab, a bayesian optimization framework, to perform hyperparameter selection. The size of the convolutional filters vary from 3*3 to 20*20 while the number of convolutional filters varied from 16 to 256 using multiples of 16. Fully connected layers ranged from 128 to 4096 units.

Results- The one-shot learning performance was evaluated by developing a 20-way within-alphabet classification in which an alphabet is first chosen from among those reserved for the evaluation set, along with twenty characters taken uniformly at random.



Method	Test
Humans	95.5
Hierarchical Bayesian Program Learning	95.2
Affine model	81.8
Hierarchical Deep	65.2
Deep Boltzmann Machine	62.0
Simple Stroke	35.2
1-Nearest Neighbor	21.7
Siamese Neural Net	58.3
Convolutional Siamese Net	92.0

Prototypical Networks for Few-shot Learning

Prototypical network is a simpler model and it is kind of a matching network. It is used for the problems of few shot classification. Main features of prototypical network:

1) It is designed for few shot learning.
2) It provides episodic training.
3) Finds a prototype for each class.
4) It uses euclidean distance
5) It uses simpler inductive bias that achieves excellent results.

Similar to matching network it is based on the philosophy that training and test condition should match. Prototypical networks learn a metric space in which classification can be performed by computing distances to prototype representations of each class.

Few Shot Learning

Few shot learning samples few classes from the training set and based on these training classes it takes only few samples for those. For example: Let us say it takes 5 examples per class and it builds a set of those example which is called the support set.
From the below figure we can say that from our training set we sampled 3 class labels and for each of the class labels we sampled 5 examples only then our sample set contains 15 sample and each of them has 3 classes. Then prototypical network learns a function f that embeds those sample sets into space.

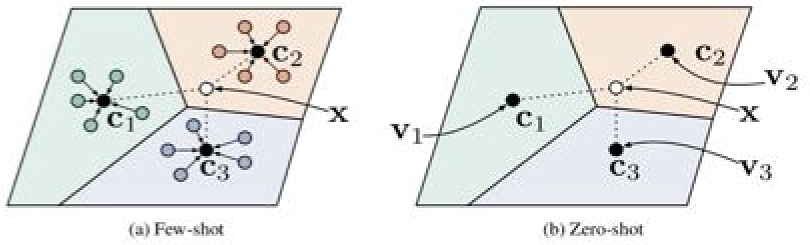


Figure 1: Prototypical networks in the few-shot and zero-shot scenarios. **Left**: Few-shot prototypes $\mathbf{c}_k$ are computed as the mean of embedded support examples for each class. **Right**: Zero-shot prototypes $\mathbf{c}_k$ are produced by embedding class meta-data $\mathbf{v}_k$. In either case, embedded query points are classified via a softmax over distances to class prototypes: $p_\phi(y = k|\mathbf{x}) \propto \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))$.

Each of these examples of the classes tends to cluster and it computes the mean of these points and tells that this mean is the prototype of this class. So this prototype is basically a class representative of the training examples. Hence, when a new example comes up and we want to find it's label we embed the new example X using the same encoding function and find the distance from each of the prototype to see which class it belongs to.

Zero Shot Learning

We use the same approach to tackle zero-shot learning; in zero shot learning each class has metadata giving high level description of the class instead of giving a small number of labeled examples. We therefore learn an embedding of the meta-data into a shared space to serve as the prototype for each class.

In general, we relate prototypical networks to clustering in order to justify the use of class means as prototypes and distances are calculated with a Bregman Divergence , such as squared euclidean distance.

Equations used in the model:

Prototype:

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\phi(\mathbf{x}_i)$$

Distribution:

$$p_\phi(y = k \mid \mathbf{x}) = \frac{\exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_{k'}))}$$

Loss:

$$J(\Phi) = -\log p_\Phi(y = K|X)$$

The first equation is used to find the mean of the embedded points of each class which we call the class prototype. And when we try to find the label of the new example X we embed it using the function f and find the euclidean distance from each of these prototypes and do softmax.

Comparison to matching networks:

Prototypical Network	**Matching Network**
Uses Euclidean distance	Uses Cosine similarity measure
It is a linear classifier	It is a weighted nearest neighbour classifier
It is a simple model	It is a complex model

EEGNet: A Compact Convolutional Network for EEG-based Brain-Computer Interfaces

EEGNet is a compact Convolutional Neural Network for classification and interpretation of EEG-based BCIs. EEGNet achieves good classification performance than the traditional CNN and converges over the less samples as required by Deep and shallow neural networks. The 10-20 system is used to capture the brain signals using 56 channels or the electrodes of EEG device. It collects the data from different parts of the brain such as pre-frontal (Fp), frontal (F), temporal (T), parietal (P), occipital (O) and central (C) .

The BCI has 5 main processing stages which can be listed as follows:

1) Data Collection: Neural data is recorded
2) Signal Processing Stage: Cleaning and processing of recorded data.
3) Feature Extraction Stage: Extracting useful information from the neural data
4) Classification Stage: Some decision is made from the data
5) Feedback Stage: Result of the decision is provided to the user.

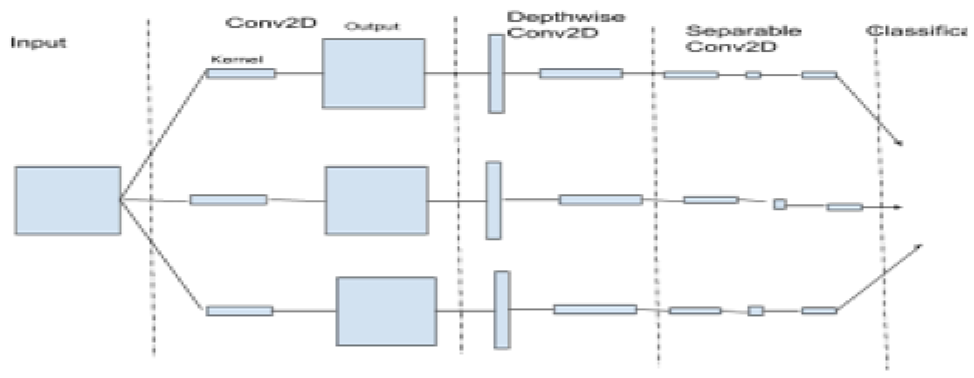


Figure 1. Architecture of EEGNet

Depthwise and Separable convolutions,is used, to construct an EEG-specific network.
The evaluation of the generalizability of EEGNet on EEG datasets is collected from four different BCI paradigms:

P300 Event-Related potential (P300):

P300 is the waveform observed over the parietal cortex, and is the large positive deflection of electrical activity. The P300 ERP is one of the strongest neural signatures observable by EEG. The targets are presented to the participants infrequently.
In a particular study, the images of natural scenery had shown to the participants with the infrequent targets of vehicle and a person are shown. When the targets are seen by the participants, they were asked to press a button by their dominant hand. The data is continuously collected by EEG during this time.

Error-related negativity (ERN):

If a person makes or perceives an error, an error-related potential can be detected in the EEG due to the person recognizing that error. Depending on the experimental task, different variants of error-related potentials can be measured.
In this, participants were shown flashing letters, arranged in 6 * 6 grid to elicit P300. The goal was to determine whether the feedback of P300 was correct or incorrect.

Movement-related cortical potential (MRCP):

It is a combination of both ERP and oscillatory components. It can be produced by the voluntary movements of hands or feet and is observable by EEG. The data here is participant's finger movements using the left index,left middle, right index or right middle fingers. In this case the data was recorded using the 256 channels of BioSemi Active II system.

Sensory motor rhythm (SMR):

SMR is similar to oscillatory component of MRCP. These signals are generally weak and varies between the subjects.
The data used here consists of 4 classes namely the data for right and left hands, feet and tongue.
It is very difficult to interpret and understand the features learned by EEGNet, and hence it is visualized using the three different approaches.

1) analyzing spatial filter outputs, averaged over trials, on the P300 dataset,
2) visualizing the convolutional kernel weights on the SMR dataset and comparing them to the weights learned by FBCSP, and
3) performing single-trial relevance analysis on the MRCP and SMR datasets

Summary of approaches

a. **Siamese Network**
   We have presented a strategy for performing one-shot classification by first learning deep convolutional siamese neural networks for verification. We outlined new results comparing the performance of our networks to an existing state-of-the-art classifier developed for the Omniglot data set.

b. **Prototypical Network**
   We have proposed a simple method called prototypical networks for few-shot learning based on the idea that we can represent each class by means of its examples in a representation space learned by a neural network. We train these networks to specifically perform well in the few-shot setting by using episodic training**.** We show how performance can be greatly improved by carefully considering the chosen distance metric, and by modifying the episodic learning procedure**.**

Pros and Cons of the used network

Pros	**Cons**
Can predict out of training set data.	More computationally intensive.
Finding similarity or a relationship between two comparable things.	More hyperparameters and fine tuning required.

Relation to the approach

Prototypical network typically uses “Few shot learning” methodology to classify the data. It uses particular number of examples for each classes. For example: Let us say it takes 5 examples per class and it builds a set of those example which is called the support set. Prototypical networks can also use “Zero shot learning” to train and classify the data. Here, prototypes and distances are calculated with a Bregman Divergence , such as squared Euclidean distance. The problem with such system is that we first require a lot of different images (training images) of each of the class, which might not be feasible every time. Especially where lot of data of a class is not available. Also, if we have the data that no longer exists , we need to remove the data and re-train the model, which is not a good approach .

In Siamese network we use a different approach. Here we are using “One shot learning” algorithm to classify the training data. In a one shot classification, we require only one training example for each class. This means this model is less data intensive. In Siamese network we take two inputs at a time and we generate the similarity scores of the input data. This similarity functions can be sigmoid function, cosine function etc. The Similarity score can range between 0 to 1. In a short while we will see that to train this network, we do not require too many instances of a class and only few are enough to build a good model.

Method

Installation and Configuring setup

Initially installed all the required libraries and set up the environment. Libraries installed are numpy, tensorflow, cv2, yaml, matplotlib, pandas, and keras.

Siamese Network on MNIST dataset

Initially we implemented and understood the working of the Siamese Network on MNIST dataset. The input of the data was the grayscale images of the MNIST dataset. The model was trained for 30000 generated pairs of the inputs with batch size of 32. The accuracy achieved was 100%.

Real time data collection of EEG Signals

The 10-20 system is used to capture the brain signals using 56 channels or the electrodes of EEG device . It collects the data from different parts of the brain such as pre-frontal (Fp), frontal (F), temporal (T), parietal (P), occipital (O) and central (C) . Here we are using 4 channels namely Fp1, Fp2, Fz and Cz. Z electrodes are often utilized as grounds or references. Cz and Fz are 'ground' or 'common' reference points for all EEG and EOG electrodes. The time stamp that we are considering for observing the ‘obs’ signals are -0.2 to 1.

Preprocessing data

A bandpass filter is applied typically with order of 3, low cutoff of .1 and high cutoff of 30. Other types of filtering include only a low cutoff of .1, a bandpass filter is applied typically with order of 2, low cutoff of .1 and high cutoff of 30, and no filtering at all. The only other noteworthy thing that takes place is a downsampling to 200 Hz sampling rate. Technically the EEG records at 201 samples per second so we have to down sample to a true 200 samples per second.

Siamese Network for Classifying Brain Signals

Initially, since we are using Brain Signals as data for the classification, we need to change the kernel size of the Siamese Network. As the network takes 2D matrix (images) as the input but our data is raw signals so we need to replace the kernel size from (2,2) to (1,1). Other Hyper parameters of the Siamese Network will remain the same. Based on the network architecture, we trained the model for different data collected for the tasks. We have mentioned the evaluation metrics in the experiments section.

Cross task generalization

Since the pre-processing is completed, further we need to modify the siamese network based on inputs of the ERN signal dataset. We will be classifying the data based on two classes of tasks, namely obstacle avoidance and observation. We are using Siamese Network to classify the data and to generalize over it. For a given data, we pair the samples with other class's data and map the target. We will show that our model is able to generalize across several matrices and it is data efficient and work in the setting of various input states.

Validating accuracy

Once the model is ready to train, we need to train and test the accuracy based on the two parameters Obstacle Avoidance(OA) and Observation(Obs). There would be four cases for validating this accuracy.

Training	Testing
OA	OA
Obs	Obs
OA	Obs
Obs	OA

Difference in method

We are attempting to perform cross task generalization by using Siamese Network, which uses one shot learning algorithm. We will take the two input into account and will calculate the similarity score, the similarity score is squished between 0 and 1, where 0 shows full participation and 1 shows full participation, and all the numbers between them is taken accordingly. This model can be trained quickly and efficiently by using this algorithm. We will maximize the accuracy by selecting the most relevant parameters.

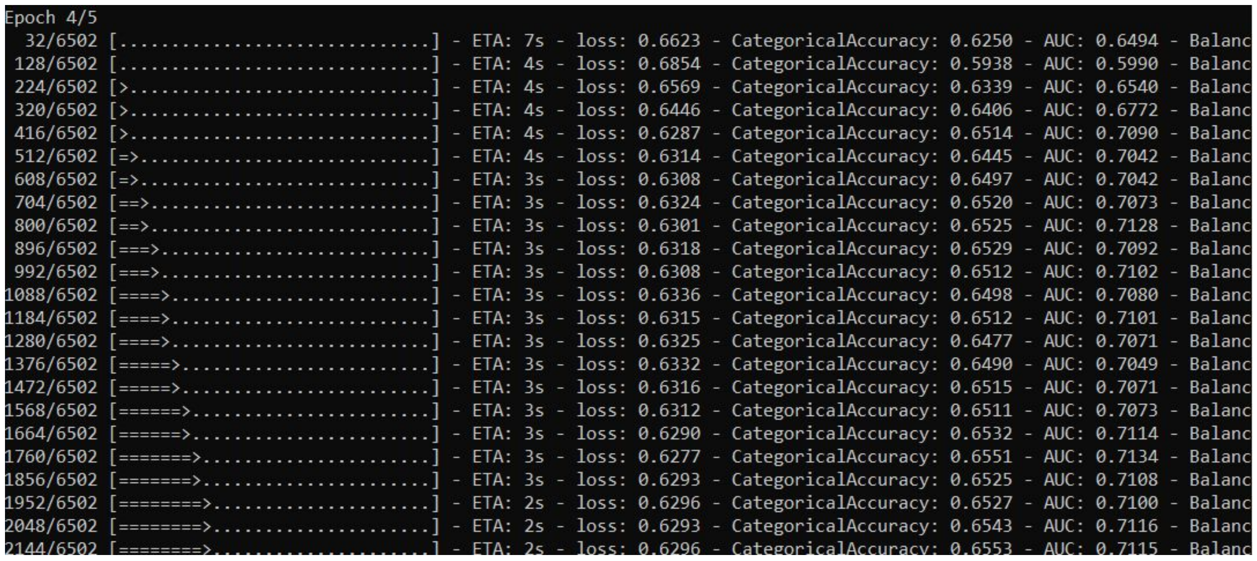
The Siamese network is used before for image classification techniques such as face recognition systems. In these type of applications input data is two dimensional. We are using this network for classifying the brain signal data where the signal data is one dimensional. This data needed cleaning and specialized preprocessing because we are using the EEG device with 4 channels. We have set the appropriate time stamp so that the 'obs' signals can be identified clearly.

Experiments

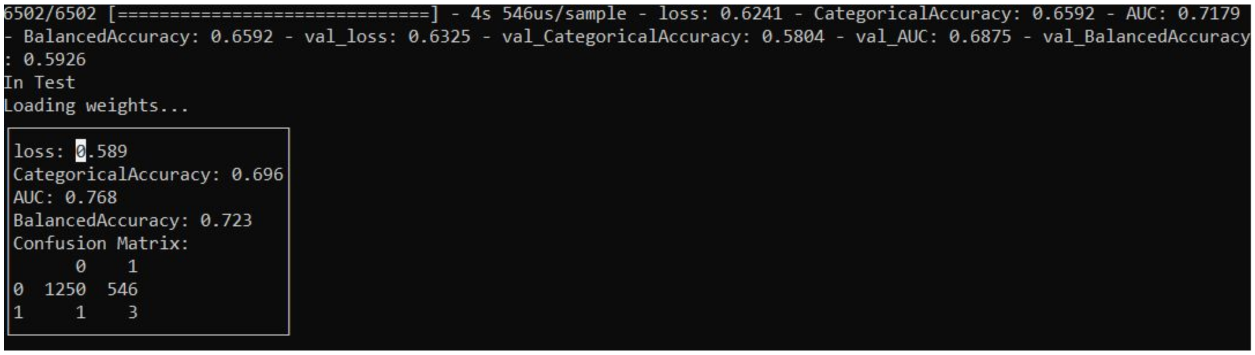
Explanation of experimental setup

The experimental setup consists of collecting the data for the two different tasks i.e. Observation and Obstacle Avoidance. The data was collected from 4 different channels Fp1, Fp2, Fz and Cz of EEG device. Data was collected for 220 different trials. For classification we require 2 classes. During training and testing the model total 6502 data instances were used for both EEGNet and the Siamese network. There are three main relevant functions in the code: the train function, the test function and the predict function.In the train function, we feed the network a positive and a negative sample (two pairs of images). We calculate the losses for each of these and add them up (with the positive sample having a target of 1 and the negative sample having a target of 0). The test function serves to measure the accuracy of the network on the test dataset. We perform the test after each training epoch to observe the training progress and to prevent overfitting. The predict function, given a pair of test images, simply predicts if they are of the same class or not. You can use predict after training is finished by setting the global variable do_learn to False.

Test results of the proposed method (Screenshots)



```
Epoch 4/5
  32/6502 [..............................] - ETA: 7s - loss: 0.6623 - CategoricalAccuracy: 0.6250 - AUC: 0.6494 - Balanc
 128/6502 [..............................] - ETA: 4s - loss: 0.6854 - CategoricalAccuracy: 0.5938 - AUC: 0.5990 - Balanc
 224/6502 [>.............................] - ETA: 4s - loss: 0.6569 - CategoricalAccuracy: 0.6339 - AUC: 0.6540 - Balanc
 320/6502 [>.............................] - ETA: 4s - loss: 0.6446 - CategoricalAccuracy: 0.6406 - AUC: 0.6772 - Balanc
 416/6502 [>.............................] - ETA: 4s - loss: 0.6287 - CategoricalAccuracy: 0.6514 - AUC: 0.7090 - Balanc
 512/6502 [=>............................] - ETA: 4s - loss: 0.6314 - CategoricalAccuracy: 0.6445 - AUC: 0.7042 - Balanc
 608/6502 [=>............................] - ETA: 3s - loss: 0.6308 - CategoricalAccuracy: 0.6497 - AUC: 0.7042 - Balanc
 704/6502 [==>...........................] - ETA: 3s - loss: 0.6324 - CategoricalAccuracy: 0.6520 - AUC: 0.7073 - Balanc
 800/6502 [==>...........................] - ETA: 3s - loss: 0.6301 - CategoricalAccuracy: 0.6525 - AUC: 0.7128 - Balanc
 896/6502 [===>..........................] - ETA: 3s - loss: 0.6318 - CategoricalAccuracy: 0.6529 - AUC: 0.7092 - Balanc
 992/6502 [===>..........................] - ETA: 3s - loss: 0.6308 - CategoricalAccuracy: 0.6512 - AUC: 0.7102 - Balanc
1088/6502 [====>.........................] - ETA: 3s - loss: 0.6336 - CategoricalAccuracy: 0.6498 - AUC: 0.7080 - Balanc
1184/6502 [====>.........................] - ETA: 3s - loss: 0.6315 - CategoricalAccuracy: 0.6512 - AUC: 0.7101 - Balanc
1280/6502 [====>.........................] - ETA: 3s - loss: 0.6325 - CategoricalAccuracy: 0.6477 - AUC: 0.7071 - Balanc
1376/6502 [=====>........................] - ETA: 3s - loss: 0.6332 - CategoricalAccuracy: 0.6490 - AUC: 0.7049 - Balanc
1472/6502 [=====>........................] - ETA: 3s - loss: 0.6316 - CategoricalAccuracy: 0.6515 - AUC: 0.7071 - Balanc
1568/6502 [======>.......................] - ETA: 3s - loss: 0.6312 - CategoricalAccuracy: 0.6511 - AUC: 0.7073 - Balanc
1664/6502 [======>.......................] - ETA: 3s - loss: 0.6290 - CategoricalAccuracy: 0.6532 - AUC: 0.7114 - Balanc
1760/6502 [=======>......................] - ETA: 3s - loss: 0.6277 - CategoricalAccuracy: 0.6551 - AUC: 0.7134 - Balanc
1856/6502 [=======>......................] - ETA: 3s - loss: 0.6293 - CategoricalAccuracy: 0.6525 - AUC: 0.7108 - Balanc
1952/6502 [========>.....................] - ETA: 2s - loss: 0.6296 - CategoricalAccuracy: 0.6527 - AUC: 0.7100 - Balanc
2048/6502 [========>.....................] - ETA: 2s - loss: 0.6293 - CategoricalAccuracy: 0.6543 - AUC: 0.7116 - Balanc
2144/6502 [========>.....................] - ETA: 2s - loss: 0.6296 - CategoricalAccuracy: 0.6553 - AUC: 0.7115 - Balanc
```



```
6502/6502 [==============================] - 4s 546us/sample - loss: 0.6241 - CategoricalAccuracy: 0.6592 - AUC: 0.7179
- BalancedAccuracy: 0.6592 - val_loss: 0.6325 - val_CategoricalAccuracy: 0.5804 - val_AUC: 0.6875 - val_BalancedAccuracy
: 0.5926
In Test
Loading weights...
 loss: 0.589
 CategoricalAccuracy: 0.696
 AUC: 0.768
 BalancedAccuracy: 0.723
 Confusion Matrix:
       0    1
 0  1250  546
 1     1    3
```

1. EEGNet

Training	Testing	Training Accuracy(%)	Testing Accuracy(%)
OA	OA	89.63	86.32
Obs	Obs	80.7	79.1
OA	Obs	70.86	66.01
Obs	OA	72.36	59.11

2. Siamese

Training	Testing	Training Accuracy(%)	Testing Accuracy(%)
OA	OA	91.03	88.26
Obs	Obs	83.12	76.8
OA	Obs	72.3	62.8
Obs	OA	69.6	68

Deep analysis/discussion about the results

One shot learning method is used to train this model for ERN detection. The model gets trained quickly because of one-shot learning algorithm. We obtained very good training and testing accuracy as shown in the above table. The model is trained on 6508 examples and with 100 epochs. We have considerably reduced the losses of our model to 0.589.

We have generalized the model on Observation(Obs) and Obstacle Avoidance(OA) tasks. This is cross-task generalization. The model is trained on OA and tested on Obs and vice versa. We have plotted confusion matrix which is often used to describe the performance of a classification model on a set of test data for which the true values are known.

Amount of effort made by the team

Since the classification algorithms needed to classify the data for brain signals, which was new to the team, hence we studied and understood different research papers. The first research paper that we read was "Prototypical Network". It gives in-depth architecture for prototypical networks and also provides the information for few shot and zero shot learning algorithms. The research paper uses these learning algorithms to train the model. Additionally, we read about EEGNet which is a convolutional neural network model and it is used to classify the brain signals. It also gives understanding of BCI, EEG device and channels used for capturing the neural activity. Later we come up with Siamese Network which is found to be effective than EEGNet because it required very less data to train the model as compared to EEGNet architecture.

We used the EEG device and considered actual brain signals to train the model. We have cleaned and preprocessed raw data from EEG device. We have implemented the base model on our captured EEG data and observed how it is performed on our data. We have calculated the accuracy of EEGNet model.

After implementation of EEGNet model, we studied Siamese Network that uses one shot learning. We studied how to pair one example of each for train and test sample and also studied the architecture of Siamese network. We implemented base model of Siamese network on MNIST dataset. After achieving the desired accuracy on the MNIST data we fine tuned the model to fit on our actual brain signal data.

Reflections

List of summary of comments from the instructor

1) Given a short description of the approach or intention.

2) Citation
   We have not listed readings for the survey in literature survey. We have used regular citation for the literature survey in reference section.

3) Cross task generalization
   Yes we have only generalize over different tasks. No we did not consider cross person generalization. Since we have changed the model from prototypical network to siamese neural network model, we decided to work with only cross task generalization.

References

1. “Triumph: Parkinson's, Tremor and DBS.” Deep Brain Stimulator for Parkinson's and Essential Tremor | Abbott U.S., https://www.abbott.com/life-changing-tech/parkinsons-tremor-and-dbs.html.
2. Svilen. “The Complete Guide to Brain-Computer Interfaces.” Medium, Svilen's Portfolio, 16 Mar. 2019, https://medium.com/svilenk/bciguide-246a9ca76fcd.
3. Snell, et al. “Prototypical Networks for Few-Shot Learning.” ArXiv.org, 19 June 2017, https://arxiv.org/abs/1703.05175.
4. “10–20 System (EEG).” *Wikipedia*, Wikimedia Foundation, 20 June 2019, https://en.wikipedia.org/wiki/10–20_system_(EEG).
5. Spüler, Martin, and Christian Niethammer. “Error-Related Potentials during Continuous Feedback: Using EEG to Detect Errors of Different Type and Severity.” Frontiers in Human Neuroscience, Frontiers Media S.A., 26 Mar. 2015, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4374466/.

Conclusion

Brain Computer Interface: Meta Learning for ERN Detection had much learnings and insights from the project implementation. Right from understanding the working of EEG device and collecting brain signals from it to process the data and generate ERN signals, the entire process was quite interesting but challenging at the same time. Choosing the right classification algorithm played an important role in bringing out the efficient outcomes. Switching from prototypical to Siamese network, showed noteworthy improvements in the accuracy and performance of the model. With one shot learning, our model has become less data intensive with which we can process the data faster and accurately. With the help of Siamese network, prediction can be done with the help of training data set and it can also find similarity between two comparable things from which we can test and predict results. While learning the advantages of using siamese network on our model, we also found some challenges and drawbacks of it such as, it is more computationally intensive, more hyperparameters and fine tuning is required.

As we trained our model to collect the ERN signals from the brain signals, we can also enhance it by using reinforcement learning and train the model to detect errors. For eg. When an action is performed and a mistake is being made, a brain signal i.e ERN signal will be generated that notifies the person about the mistake being made so that they can recover it.