Machine Learning Assignment 5

Sean Holschier z5308039
Module: 2: EEG SSVEP Recognition

Table of Contents

1. Executive Summary

To create communication devices for patients with Amyotrophic Lateral Sclerosis (ALS), these devices must allow for users to communicate through only their ocular muscles and EEG data. To accurately interpret these EEG readings, this report looks at feasibility of different Machine learning models and data processing techniques to assist with the accurate tracking of a user's intended target on a screen. This report employed the use of three different feature extraction techniques, Principal Component Analysis (PCA), Common Spatial Patterns (CSP) and Fast Fourier Transforms (FFT). These techniques were tested across three different Machine learning models, Long Short-Term Memory (LSTM), Support Vector Machine (SVM) and Random Forest Classifier to determine the most suitable configuration for real world use. Results from this experiment showed:

- The combination of a 5th order Butterworth Bandpass Filter, PCA, CSP allowed a Random Forest Classifier to predict the correct tracking of a user's target box on a screen with an accuracy of up to 100%, with a 20% test data split.
- The effectiveness of implementing Machine Learning to assist in accurate eye gaze tracking from EEG data is promising and with further fine-tuning and testing be implemented into real world use

2. Introduction

The aim for the overall project is to develop machine learning modules for a human-machine interface (HMI) solution that will allow patients with Amyotrophic Lateral Sclerosis (ALS), who are rendered immobile except for their ocular muscles, to communicate through typing. This is very important research as introducing reliable machine learning could improve the level and speed in which patients can communicate with those around them. Module 2 has the aim of utilising EEG data and attempting to determine where a user is focused on a screen to improve the accuracy of eye tracking technology in module 1.

The EEG SSVEP recognition module will require a machine learning classifier to classify between the 5 different locations shown on the subject's screen. This can include decision tree classifiers, SVM classifiers and many other classification models. There will be 6 different categories, [0, 1, 2, 3, 4, 5], where 0 is when the screen is blank. These categories align with the different frequencies of 12.00, 10.00, 8.57, 7.50 and 6.66 Hz respectively, of the boxes at which they are focused on. This dataset has been collected by the Centre for Research and Technology Hellas using an Emotiv EPOC EEG headset. (Centre for Research and Technology Hellas, 2021)

To properly assess the efficiency and accuracy of the module and utilised models, we will use a few different metrics. Accuracy/F1-scores will be used as a main metrics, but we will also be referring to the confusion matrices to analyse what kind of errors are occurring while also using other metrics such as precision and other cross validated scores as needed to determine the efficiency of a model. This report will outline the efficiency of 3 models used, Long Short-Term Memory (LSTM) neural nets, Support Vector Machines (SVM and Random Forest Classifiers to see which model performs the best. Different feature extraction techniques will be used, including principal component analysis (PCA), common special patterning (CSP) and fast Fourier transforms (FFT), alongside data pre-processing through a Butterworth bandpass filter to filter the EEG data. This report will reference a similar study done by researchers at the Universiti Malaysia Pahang which obtained an accuracy of 88% (Rashid, 2020) as a guide and a baseline from which a higher accuracy will be attempted using similar but improved techniques.

3. Dataset

3.1 Dataset information:

The dataset for this experiment was saved in .mat files, for use in matlab.

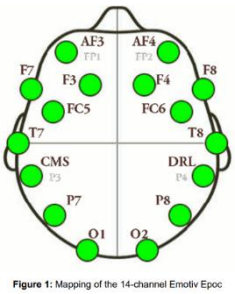
- There are:
  11 subjects (named by the number in the name; eg. U001ai.mat)
- 5 identical runs per subject (named by the letter in the name; eg. U001ai.mat)
- Each run is broken up into 2 files, i and ii (named by the 'i' in the name; eg. U001ai.mat)
- Each file has 2 tables, with the column names shown in *table 1*:

EEG (total: 2,118,528 records)

Column	Data field type
AF3	Continuous
F7	Continuous
F3	Continuous
FC5	Continuous
T7	Continuous
P7	Continuous
O1	Continuous
O2	Continuous
P8	Continuous
T8	Continuous
FC6	Continuous
F4	Continuous
F8	Continuous
AF4	Continuous
Status	Categorical - Nominal

Table 1 EEG table column names and data types

The EEG table contains the EEG data for each instance in which they were measured. Columns AF3 through to AF4 are the different locations in which EEG sensors were placed. Status shows the status of that run, which I assume to mean whether it was working well or not.



Example 1. Location of EEG sensors (Rashid, 2020)

The data in the Events table can be used to label the EEG data. Event codes signal the start of a session in which the subject looks at a certain screen object. By correlating this with the EEG measurement number, we can label the above rows.

Events (total: 4,235 records)

Column	Data field type
Index value	Categorical - Ordinal
Event code	Categorical - Nominal
EEG measurement number	Categorical - Ordinal
Index values (repeated for some reason)	Categorical - Ordinal

Table 2 Events table column names and data types

NOTE: I have used data pre-processing to label the data in EEG. (The code and more information is listed below) The current processed data is shown in *table 3*:

Column	Data field type
AF3	Continuous
F7	Continuous
F3	Continuous
FC5	Continuous
T7	Continuous
P7	Continuous

O1	Continuous
O2	Continuous
P8	Continuous
T8	Continuous
FC6	Continuous
F4	Continuous
F8	Continuous
AF4	Continuous
Status	Categorical - Nominal
Label	Categorical - Nominal

Table 3 Column names and data types for labelled dataset

Redundant data: From my understanding of what the 'Status' column does, this data is not relevant in the dataset, as only a few rows have anything other than '0'. To be sure of this, I will use a decision tree classifier to list the feature importance of the EEG data.

```
# Perform feature importance listing
model = DecisionTreeClassifier()
# fit the dataset so we can rank inportance
model = model.fit(df1.values[:,0:15],df1.values[:,15])
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
```

Code Snippet 1 script of extracting feature importance

```
Feature: 0, Score: 0.07567
Feature: 1, Score: 0.07087
Feature: 2, Score: 0.06956
Feature: 3, Score: 0.06892
Feature: 4, Score: 0.07360
Feature: 5, Score: 0.07352
Feature: 6, Score: 0.07193
Feature: 7, Score: 0.06700
Feature: 8, Score: 0.07283
Feature: 9, Score: 0.06451
Feature: 10, Score: 0.06777
Feature: 11, Score: 0.08029
Feature: 12, Score: 0.06833
Feature: 13, Score: 0.07484
Feature: 14, Score: 0.00037
```

Code Snippet 2 Feature importance scores

As we can see in *code snippet 2*, the features (order same as table above) are generally weighed about the same in terms of important, except for the 'Status' column, which shows an importance of 0.00037. Therefore, we can safely assume that the Status feature is redundant and can be removed.

Missing data: Commands are run to check that here are no missing data or infinite values in this dataset. These checks all came back clear

```
# Find NaN values in Dataset
nulls = df1.isnull().sum()
# Find Infinite values in the dataset
infs = df1.isin([np.inf, -np.inf]).sum()

# Outputs show no null and infinite values
✓ 0.4s
```

Code Snippet 3 script checking for infinite and null values

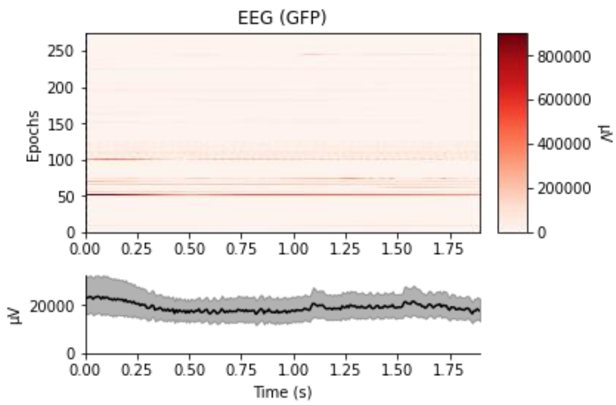
3.2 Data Distribution

In order to get an idea of how the data is separated, I graphed the epochs for the first class, 'Hz12' to look for trends between features and the Label itself. There are 1325 events in the whole dataset, and a balanced number of each class.

```python
epochs = Epochs(raw, events, event_id, tmin=0, tmax=1.9, proj=True, picks=picks,
                baseline=None, preload=True)

epochs['hz12'].plot_image()
```

Code Snippet 4 script plotting MNE epoch for class '12Hz'

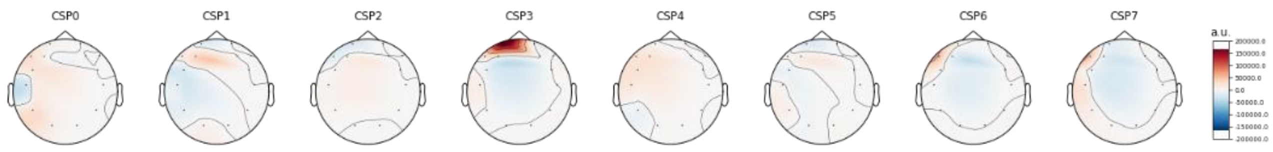


Example 2 12Hz Visualisation Example

3.2.1 Data separability

As can been seen in *example 2*, it is hard to determine any kind of trend within this dataset from just these points. However, when compared to other trials or other classes, the lines may be vastly different and easier to classify.

Another small piece of data visualisation we can do is through CSP, which I will discuss later in the project, as it is a feature selection/dimensionality reduction feature. Here is what it can look like when plotted:



Example 3 CSP Example

3.3 Data Preparation

As part of data preparation, I wrote a simple script to extract the relevant information and label the dataset. I will do this separately for each subject:

This script:

1) Extracts the relevant columns from the .mat file, and places them into a pandas dataframe.

```python
# This code block ran through all the data, labelled each row and collated them into one big dataframe
datasets = ['EEG-SSVEP-Experiment3/U001ai.mat','EEG-SSVEP-Experiment3/U001aii.mat','EEG-SSVEP-Experime
# datasets = ['EEG-SSVEP-Experiment3/U001ai.mat','EEG-SSVEP-Experiment3/U001aii.mat','EEG-SSVEP-Experi
numbers = [1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2
count = 0
df1 = LabelEEG(datasets[count], numbers[count])
count = 1
while count < len(datasets):
    df2 = LabelEEG(datasets[count], numbers[count])
    count = count+1
    df1 = pd.concat([df1, df2], axis=0)
```

Code Snippet 5 extracting of dataset from mat files and placing these in pandas dataframes

2) Using the LabelEEG() function, adds a 'Label' column and labels the dataframe with an integer value indicating where the subject is focused.

```
# Retrieves EEG data for run
# Uses events to label EEG data
def LabelEEG(dataset,type):
    data_set = loadmat(dataset)
    rows = data_set['info']
    actualrows = rows['label']
    dataVal = data_set['eeg']
    i = 0
    finalrows={}
    b = actualrows[0][0][0]
    for v in b:
        finalrows[i]=v[0]
        i = i+1
    data = dataVal.transpose()
    df = pd.DataFrame(data=data, columns=finalrows.values())
    if type == 1:
        hertzOrder = [4,2,3,5,1,2,5,4,2,3,1,5]
    else:
        hertzOrder = [4,3,2,4,1,2,5,3,4,1,3,1,3]
    indexNum=0
    hertzNum=0
    events=data_set['events']
    labelList=[0]*len(df.index)
    df['Label']= labelList
    for i in events:
        if i[1] == 32779:
            df.iloc[indexNum:i[2],15] = 0
            indexNum = i[2]
        if i[1] == 32780:
            df.iloc[indexNum:i[2],15] = hertzOrder[hertzNum]
            indexNum = i[2]
            hertzNum+=1
    return df
```

Code Snippet 6 LabelEEG function for labelling data

3) All the datasets are placed in the same file and read to a csv, so we have a saved copy to use. We can drop the Status column from the dataset, as we deemed it redundant.

```
count = 1
while count < len(datasets):
    df2 = LabelEEG(datasets[count], numbers[count])
    # df2 = scaleColumns(df2)
    count = count+1
    df1 = pd.concat([df1, df2], axis=0)
df1 = df1.drop(['Status'], axis=1)
# df1 = df1[df1['Label'] > 0]
df1.iloc[:,0:14] = df1.iloc[:,0:14].div(1000)
df1.to_csv('fullEEGdatasetU1.csv',index=False)
```

Code Snippet 7 Dropping of Status column

4) We import the dataset and place the data through a 5th order Butterworth bandpass filter. This filter can assist with improving the accuracy of the models.

```
def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a


def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y
```

Code Snippet 8 functions for the butterworth bandpass filter

5) Due to a bug in Windows intel CPUs and Scipy, CSP will not accept the filtered data, as it will throw an “LinAlgError: SVD did not converge” error. A reference to the bug is found in the following link: https://github.com/mne-tools/mne-python/issues/8784 (MNE tools, 2021)
To get around this, the dataset is put through PCA, which will also attempt to perform feature extraction on the data. The number of features is set to 14 to ensure no data is lost in the process.

```
pca = PCA(n_components=14)
filtered = pca.fit_transform(filtered)
df1.iloc[:,0:14] = pd.DataFrame(filtered).iloc[:,0:14]
```

Code Snippet 9 Principal Component Analysis code

Visualising the filtered and PCA’d data compared to the original data can be seen below. The higher lines near 4 are the original data, with the blue line signifying classes at that timeframe.

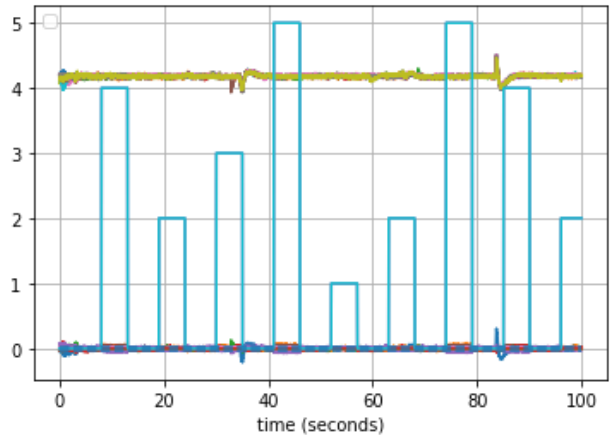


Figure 1. Comparison Between Filtered and Original Data

Once the data is prepared in this way, we have datasets that are [NumTimesteps*NumSensors]. Since this data is EEG data, we must sort this dataset into trials, to allow for time-series classification. Therefore, we will use MNE’s Epoch and Event modules to create a dataset that has the dimensions: [NumTrials* NumTimesteps*NumSensors]
To do this:
6) Using MNE’s event and epoch features, we sort the dataset into a 3D dataset with trials. This will only take the first 1.9 seconds of the trial, the timeframe in which the eye movement towards the specified object is likely to take place. Narrowing this timeframe down to 1.9 seconds will assist in preventing extra noise from effecting the trial, and training on the shorter timeframes will allow for quicker response times from the models in real world deployed use.

```
event_id = dict(hz12=1, hz10=2, hz857=3, hz750=4, hz666=5)
# Apply band-pass filter
events = find_events(raw, stim_channel='Label',output='step',initial_event=False,shortest_event=10)
picks = pick_types(raw.info, meg=False, eeg=True, stim=False, eog=False, exclude='bads')
# raw.plot(events=events, start=0, duration=50, color='gray',
#          event_color=None,scalings=dict(mag=1e-12, grad=4e-11, eeg=0.1, eog=150e-6, ecg=5e-4,
#       emg=1e-3, ref_meg=1e-12, misc=1e-3, stim=1,
#       resp=1, chpi=1e-4, whitened=1e2))

# Testing will be done with a running classifier
epochs = Epochs(raw, events, event_id, tmin=0, tmax=1.9, proj=True, picks=picks,
                baseline=None, preload=True)

epochs['hz12'].plot_image()
y = epochs.events[:, -1]
X =  epochs.get_data()
trials=1375
```

Code Snippet 10 Sorting of data into MNE epochs for event creation

7) After converting this back into an array, its dimensions are:
[NumTrials* NumTimesteps*NumSensors]

```
epochs['hz12'].plot_image()
y = epochs.events[:, -1]
X =  epochs.get_data()
```

Code Snippet 11 Plotting 12Hz event data and creation of X and y values for ML training

Visualising data as seen in *code snippet 11* will show all events where the subject was looking at the 12hz part of the screen as seen in *figure 2*:

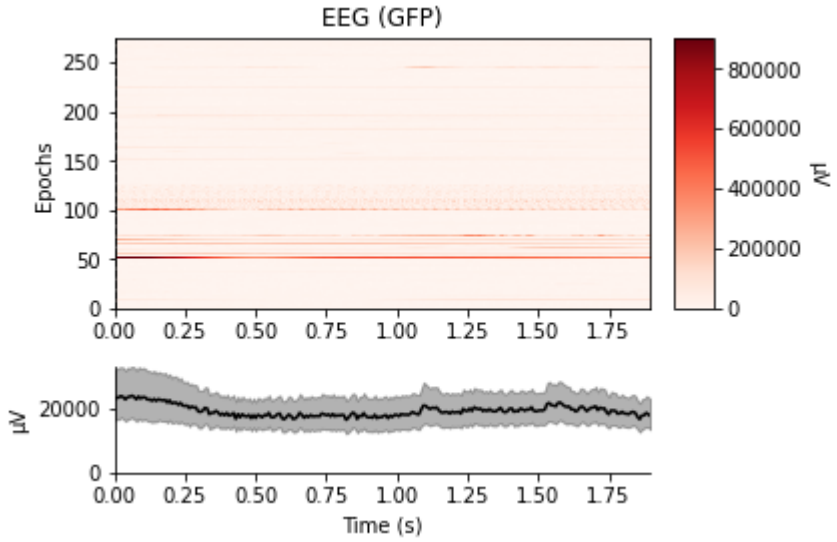


Figure 2 Visualisation of 12Hz Events

3.4 Dimensionality reduction suggestions

I suggest dimensionality reduction for this dataset as currently the data does not look very separable. By using techniques like Common Spatial Patterning (CSP), many studies have been able to achieve a higher level of accuracy in their classification algorithms. Some examples are:

- Five-Class SSVEP Response Detection using Common- Spatial Pattern (CSP)-SVM Approach (Rashid, 2020)
- One-Versus-the-Rest(OVR) Algorithm: An Extension of Common Spatial Patterns(CSP) Algorithm to Multi-class Case (Wei Wu, 2005)

I will be attempting to use CSP to reduce the number of features and allow for better separability, which will lead to a better accuracy metric.
I will also attempt to implement other dimensionality reduction features that I may come across, such as a Fourier transform, which may allow for a better representation of the data.

4. Experimental set-up

Our methodology is largely broken up into data pre-processing, feature extraction and ML training and testing. As seen in the section 3.3, the dataset goes through a 5^{th} order Butterworth bandpass filter, filtered between 4Hz and 30Hz, and then through PCA at 14 components. Once this is done, another feature extraction technique is applied. This can be common spatial patterning (CSP) or fast Fourier transforms (FFT), both techniques suited for extracting features from signal data. We are also experimenting without having the second feature extraction technique used. Once feature extraction is complete our data is split into a training set and a test set, with an 80:20 ratio.

```
Xnew = np.transpose(X,(0,2,1)).copy()
X_train, X_test, y_train, y_test = train_test_split(Xnew,y,test_size=0.2,shuffle=True,stratify=y)
```

Code Snippet 12 train_test_split for splitting the dataset

The stratify feature is used to ensure the training set is balanced, to prevent any issues that may be caused by unbalanced datasets.

Three different machine learning models will be tried during this experiment. These three models, in order of expected performance with justifications are:

1. **Long Short-Term Memory** (LSTM), an artificial neural network, which allows for time series based learning, is very promising for EEG data. I expect this model to perform the best, as it will take a lot of time to train but will better extract more important features.
2. **Support Vector Machine** (SVM) is an ML model which will attempt to classify data by drawing hyperplanes over multiple dimensions of data. Due to high performance seen in the aforementioned study (Rashid, 2020), I expect this to perform at a high standard, close to the performance of the LSTM.
3. **Random Forest Classifiers** utilise many decision trees over subsets of the dataset and collates these to perform its estimations. While Random Forest Classifiers have the capability to perform well in multi-class scenarios, the dataset is far from linear, so I believe it will struggle compared to the previous two models.

The overall methodology can be seen in *diagram 1*:

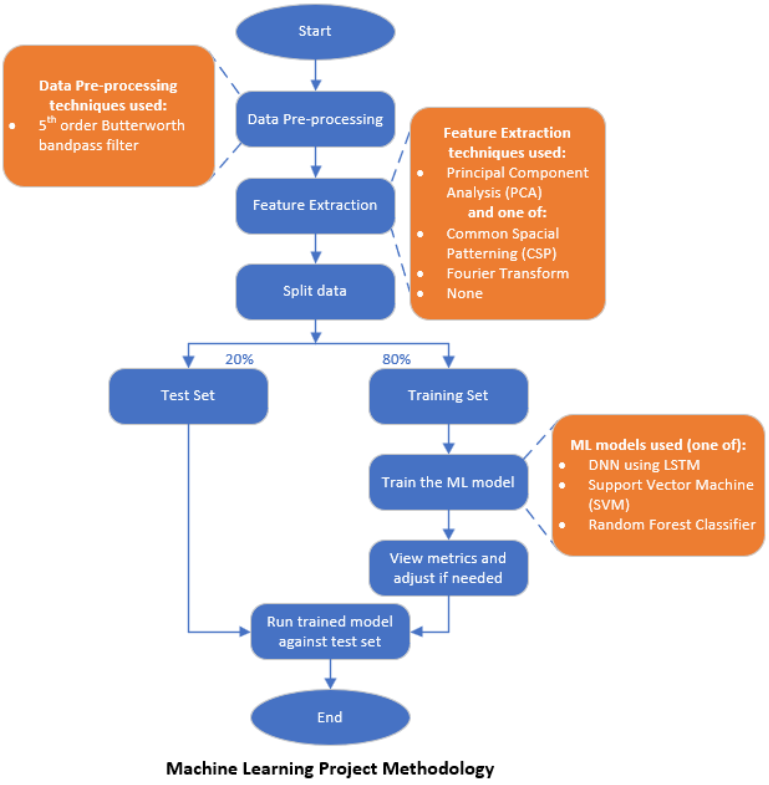


Diagram 1 Flowchart of Machine Learning Project Methodology

5. Results

5.1 LSTM Model

Accuracy scores

LSTM Model Variation	Accuracy Score (f1-score)	Inference FPS
LSTM (no feature extraction after filtering and PCA)	0.60	1060
CSP + LSTM	0.28	1061
FFT + LSTM	0.97	1047

Table 4 accuracy and FPS of LSTM trials

Best configuration is highlighted yellow

Confusion Matrices

LSTM (no feature extraction after filtering and PCA)	CSP + LSTM	FFT + LSTM
Confusion Matrix [[51 38 4 3 1] [4 12 2 2 0] [0 5 45 1 0] [0 0 14 49 36] [0 0 1 0 7]]	[[7 4 6 5 2] [12 15 8 10 8] [32 34 48 34 31] [1 1 4 5 1] [3 1 0 1 2]]	Confusion Matrix [[55 1 0 0 0] [0 51 0 0 0] [0 2 63 0 0] [0 1 3 54 0] [0 0 0 1 44]]

Table 5 confusion matrices of LSTM trials

5.1.1 LSTM (no feature extraction after filtering and PCA)

```
num_components = 14
timesteps = 244
channels_num = 14
Xnew = np.transpose(X,(0,2,1)).copy()
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(Xnew,y,test_size=0.2,shuffle=True,stratify=y)
model = Sequential()
# Set up LSTM model
model.add(LSTM(128,input_shape=(timesteps,num_components),return_sequences=True))
model.add(Dropout(0.25))
model.add(LSTM(64,input_shape=(timesteps,num_components),return_sequences=False))
model.add(Dense(5, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Train model
model.fit(X_train,y_train-1,epochs=100,verbose=1,validation_split=0.2)
# Test model
tic = time.perf_counter()
predicted = model.predict(X_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper
scores = []
# Extract prediction values
for predict in predicted:
    scores.append(np.argmax(predict))
# Exctract and save output metrics
output = "Results for LSTM (nothing after PCA)\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(scores,y_test-1)) +'\n'
output = output + str(classification_report(scores,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 13 script for LSTM

Parameters

No feature extraction after filtering and PCA was used

This LSTM model used the following parameters:

LSTM 128 dimensions -> Dropout 25% -> LSTM 64 dimensions -> Dense 5 dimensions (activation=softmax)

This model was trained with 100 Epochs with an adam optimiser.

Results

Results for LSTM (no feature extraction after filtering and PCA)

```
Results for LSTM (nothing after PCA)

Confusion Matrix
[[51 38  4  3  1]
 [ 4 12  2  2  0]
 [ 0  5 45  1  0]
 [ 0  0 14 49 36]
 [ 0  0  1  0  7]]
              precision    recall  f1-score   support

           0       0.93      0.53      0.67        97
           1       0.22      0.60      0.32        20
           2       0.68      0.88      0.77        51
           3       0.89      0.49      0.64        99
           4       0.16      0.88      0.27         8

    accuracy                           0.60       275
   macro avg       0.58      0.68      0.53       275
weighted avg       0.79      0.60      0.64       275

FPS: 1060.2357116038984FPS
```

Code Snippet 14 results for LSTM trial

5.1.2 CSP + LSTM

```
# CSP
num_components = 8
csp = CSP(n_components=num_components, reg=None, log=None, norm_trace=False,cov_est='concat',transform_into='csp_space')
# Fit and transform CSP values
csp.fit(X,y)
S = csp.transform(X)
# Show and save plot of CSP patterning
figure = plt.figure(csp.plot_patterns(epochs.info, ch_type='eeg', size=1.5,show=False))
figure.savefig('Figures/Fig3_CSPpatternplots.jpg',bbox_inches='tight')
# LSTM
timesteps = 244
channels_num = 14
S = np.reshape(S,(trials,timesteps,num_components))
# Splitting the dataset
S_train, S_test, y_train, y_test = train_test_split(S,y,test_size=0.2,shuffle=True,stratify=y)
ynew = y_train-1
model = Sequential()
# Setup LSTM model
model.add(LSTM(128,input_shape=(timesteps,num_components),return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(64,input_shape=(timesteps,num_components),return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(5, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Train dataset
model.fit(S_train,ynew,epochs=100,verbose=1,validation_split=0.2)
# Test dataset
tic = time.perf_counter()
predicted = model.predict(S_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper

scores = []
# Extract prediction values

for predict in predicted:
    scores.append(np.argmax(predict))
# Exctract and save output metrics
output = "Results for CSP + LSTM\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(scores,y_test-1)) +'\n'
output = output + str(classification_report(scores,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 15 script for CSP + LSTM

Parameters

CSP was used as a feature extraction and dimensionality reduction method to 8 features.

This LSTM model used the following parameters:

LSTM 128 dimensions -> -> Dropout 20% -> LSTM 64 dimensions -> Dropout 20% -> Dense 5 dimensions (activation=softmax)

This model was trained with 100 Epochs with an adam optimiser.

Results

```
Results for CSP + LSTM

Confusion Matrix
[[ 7  4  6  5  2]
 [12 15  8 10  8]
 [32 34 48 34 31]
 [ 1  1  4  5  1]
 [ 3  1  0  1  2]]
              precision    recall  f1-score   support

           0       0.13      0.29      0.18        24
           1       0.27      0.28      0.28        53
           2       0.73      0.27      0.39       179
           3       0.09      0.42      0.15        12
           4       0.05      0.29      0.08         7

    accuracy                           0.28       275
   macro avg       0.25      0.31      0.21       275
weighted avg       0.54      0.28      0.33       275

FPS: 1061.480893224003FPS
```

Code Snippet 16 results for CSP + LSTM

5.1.3 FFT + LSTM

```
# FFT
timesteps = 244
Xnew = np.transpose(X,(0,2,1)).copy()
# Get FFT
yf = fft(Xnew[0])
xf = fftfreq(timesteps, 1 / 128)
plt.plot(xf, np.abs(yf))
plt.savefig('Figures/Fig4_FFTtransformedsample.jpg',bbox_inches='tight')
plt.show()
Xfft = Xnew.copy()
for i in range(0, trials):
    Xfft[i] = fft(Xnew[i])
num_components = 14
channels_num = 14
# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(Xfft,y,test_size=0.2,shuffle=True,stratify=y)
model = Sequential()
# Setup LSTM model
model.add(LSTM(128,input_shape=(timesteps,num_components),return_sequences=True))
model.add(Dropout(0.25))
model.add(LSTM(64,input_shape=(timesteps,num_components),return_sequences=False))
model.add(Dense(5, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Train model
model.fit(X_train,y_train-1,epochs=100,verbose=1,validation_split=0.2)
# Test model
tic = time.perf_counter()
predicted = model.predict(X_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper
scores = []
for predict in predicted:
    scores.append(np.argmax(predict))
# Exctract and save output metrics
output = "Results for FFT + LSTM\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(scores,y_test-1)) +'\n'
output = output + str(classification_report(scores,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 17 script for FFT + LSTM

Parameters

FFT was used as a feature extraction method to better transform the signals for the ML model

This LSTM model used the following parameters:

LSTM 128 dimensions -> Dropout 25% -> LSTM 64 dimensions -> Dense 5 dimensions (activation=softmax)

This model was trained with 100 Epochs with an adam optimiser.

Results

```
Results for FFT + LSTM

Confusion Matrix
[[55  1  0  0  0]
 [ 0 51  0  0  0]
 [ 0  2 63  0  0]
 [ 0  1  3 54  0]
 [ 0  0  0  1 44]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        56
           1       0.93      1.00      0.96        51
           2       0.95      0.97      0.96        65
           3       0.98      0.93      0.96        58
           4       1.00      0.98      0.99        45

    accuracy                           0.97       275
   macro avg       0.97      0.97      0.97       275
weighted avg       0.97      0.97      0.97       275

FPS: 1047.8927312157086FPS
```

Code Snippet 18 results for FFT + LSTM

5.1.4 LSTM Summary

Overall, the LSTM after a Fourier transform showed the best accuracy, with a 0.97 F1-score accuracy. However, I do want to note that these scores were not the most consistent and can vary on where the ML model was up to when training, along with exactly which data was contained in the test split. I ran the code a few times to get trials that were fairly consistent before recording them.

My method for determining the best parameters and feature dimensions were through research on similar studies done by researchers and some experimenting to find the best parameters. When training in Epochs, I also had to consider the technical constraints, so I set all LSTM trials to 100 epochs.

Best Configuration: FFT + LSTM (configuration seen above)

5.2 SVM Model

Accuracy scores

SVM Model Variation	Accuracy Score (f1-score)	Inference FPS
SVM (no feature extraction after filtering and PCA)	0.93	1868
CSP + SVM	0.97	1787
FFT + SVM	0.92	1893

Table 6 accuracy and FPS of SVM trials

Best configuration is highlighted yellow

Confusion Matrices

SVM (no feature extraction after filtering and PCA)	CSP + SVM	FFT + SVM
Confusion Matrix [[54 3 0 1 0] [0 49 1 0 0] [0 2 60 1 0] [1 1 5 52 2] [0 0 0 1 42]]	Confusion Matrix [[54 0 0 0 0] [0 53 2 0 0] [0 0 62 0 0] [1 2 2 55 2] [0 0 0 0 42]]	Confusion Matrix [[51 2 0 0 0] [2 50 1 0 0] [0 0 60 2 0] [2 3 5 52 4] [0 0 0 1 40]]

Table 7 confusion matrices of SVM trials

5.2.1 SVM (no feature extraction after filtering and PCA)

```
timesteps = 244
channels_num = 14
# Reshaping the dataset to 2d from 3d
Xnew = np.reshape(X.copy(),(trials,timesteps*channels_num))
# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(Xnew,y,test_size=0.2,shuffle=True,stratify=y)
ynew = y_train-1
# SVM
model = svm.SVC(kernel='rbf',max_iter=-1, probability=True)
# Train model
model.fit(X_train,ynew)
# Test model
tic = time.perf_counter()
predicted = model.predict(X_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper
# Exctract and save output metrics
output = "Results for SVM (nothing after PCA)\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(predicted,y_test-1)) +'\n'
output = output + str(classification_report(predicted,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 19 script for SVM

Parameters

No feature extraction after filtering and PCA was used

This SVM model used the following parameters:

SVM type: SVC (classifier), Kernel: rbf, no limit on iterations

Results

Results for SVM (no feature extraction after filtering and PCA)

```
Results for SVM (nothing after PCA)

Confusion Matrix
[[54  3  0  1  0]
 [ 0 49  1  0  0]
 [ 0  2 60  1  0]
 [ 1  1  5 52  2]
 [ 0  0  0  1 42]]
              precision    recall  f1-score   support

           0       0.98      0.93      0.96        58
           1       0.89      0.98      0.93        50
           2       0.91      0.95      0.93        63
           3       0.95      0.85      0.90        61
           4       0.95      0.98      0.97        43

    accuracy                           0.93       275
   macro avg       0.94      0.94      0.94       275
weighted avg       0.94      0.93      0.93       275

FPS: 1868.5668416710814FPS
```

Code Snippet 20 results for SVM

5.2.2 CSP + SVM

```
# CSP
num_components = 8
csp = CSP(n_components=num_components, reg=None, log=None, norm_trace=False,cov_est='concat',transform_into='csp_space')
# Fit and transform CSP values
csp.fit(X,y)
S = csp.transform(X)
# Show and save pattern plots
figure = plt.figure(csp.plot_patterns(epochs.info, ch_type='eeg', size=1.5,show=False))
figure.savefig('Figures/Fig3_CSPpatternplots.jpg',bbox_inches='tight')
# SVM
timesteps = 244
channels_num = 14
# Reshape dataset from 3d to 2d
S = np.reshape(S,(trials,timesteps*num_components))
# Split dataset
S_train, S_test, y_train, y_test = train_test_split(S,y,test_size=0.2,shuffle=True,stratify=y)
ynew = y_train-1
model = svm.SVC(kernel='rbf',max_iter=-1, probability=True)
# Train model
model.fit(S_train,ynew)
# Test model
tic = time.perf_counter()
predicted = model.predict(S_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper
# Exctract and save output metrics
output = "Results for CSP + SVM\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(predicted,y_test-1)) +'\n'
output = output + str(classification_report(predicted,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 21 script for CSP + SVM

Parameters

CSP was used as a feature extraction and dimensionality reduction method to 8 features. This SVM model used the following parameters:

SVM type: SVC (classifier), Kernel: rbf, no limit on iterations

Results

Results for CSP + SVM

```
Results for CSP + SVM

Confusion Matrix
[[54  0  0  0  0]
 [ 0 53  2  0  0]
 [ 0  0 62  0  0]
 [ 1  2  2 55  2]
 [ 0  0  0  0 42]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        54
           1       0.96      0.96      0.96        55
           2       0.94      1.00      0.97        62
           3       1.00      0.89      0.94        62
           4       0.95      1.00      0.98        42

    accuracy                           0.97       275
   macro avg       0.97      0.97      0.97       275
weighted avg       0.97      0.97      0.97       275

FPS: 1787.7986855037868FPS
```

Code Snippet 22 results for CSP + SVM

5.2.3 FFT + SVM

```python
# FFT
timesteps = 244
Xnew = np.transpose(X,(0,2,1)).copy()
# Get FFT values and perform transformation
yf = fft(Xnew[0])
xf = fftfreq(timesteps, 1 / 128)
# Show plot and save
plt.plot(xf, np.abs(yf))
plt.savefig('Figures/Fig4_FFTtransformedsample.jpg',bbox_inches='tight')
plt.show()
Xfft = Xnew.copy()
# Perform fft for whole dataset
for i in range(0, trials):
    Xfft[i] = fft(Xnew[i])
# SVM
channels_num = 14
# Reshaping the dataset to 2d from 3d
XfftNew = np.reshape(Xfft.copy(),(trials,timesteps*channels_num))
# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(XfftNew,y,test_size=0.2,shuffle=True,stratify=y)
ynew = y_train-1
print(Xnew.shape)
model = svm.SVC(kernel='rbf',max_iter=-1, probability=True)
# Train model
model.fit(X_train,ynew)
# Test with test dataset
tic = time.perf_counter()
predicted = model.predict(X_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper
# Exctract and save output metrics
output = "Results for FFT + SVM\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(predicted,y_test-1)) +'\n'
output = output + str(classification_report(predicted,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 23script for FFT + SVM

Parameters

FFT was used as a feature extraction method to better transform the signals for the ML model

This SVM model used the following parameters:

SVM type: SVC (classifier), Kernel: rbf, no limit on iterations

Results

Results for FFT + SVM

```
Results for FFT + SVM

Confusion Matrix
[[51  2  0  0  0]
 [ 2 50  1  0  0]
 [ 0  0 60  2  0]
 [ 2  3  5 52  4]
 [ 0  0  0  1 40]]
              precision    recall  f1-score   support

           0       0.93      0.96      0.94        53
           1       0.91      0.94      0.93        53
           2       0.91      0.97      0.94        62
           3       0.95      0.79      0.86        66
           4       0.91      0.98      0.94        41

    accuracy                           0.92       275
   macro avg       0.92      0.93      0.92       275
weighted avg       0.92      0.92      0.92       275

FPS: 1893.2533737460487FPS
```

Code Snippet 24 results for FFT + SVM

5.2.4 SVM Summary

Overall, CSP + SVM showed the best accuracy scores, also with an accuracy of 0.97. Many different kernels were tried, but the rbf kernel was shown to be used in many different studies in EEG data analysis (Rashid, 2020), so they were used for these trials.

Best Configuration: CSP + SVM

5.3 Random Forest Classifier Model

Accuracy scores

Random Forest Classifier Model Variation	**Accuracy Score (f1-score)**	**FPS**
Random Forest Classifier (no feature extraction after filtering and PCA)	0.92	11434
CSP + Random Forest Classifier	1.00	13330
FFT + Random Forest Classifier	0.82	9820

Table 8 accuracy and FPS of Random Forest Classifier trials

Best configuration is highlighted yellow

Confusion Matrices

Random Forest Classifier (no feature extraction after filtering and PCA)	CSP + Random Forest Classifier	FFT + Random Forest Classifier
Confusion Matrix [[51 2 0 1 0] [2 52 3 0 1] [2 0 60 3 0] [0 1 3 51 3] [0 0 0 0 40]]	Confusion Matrix [[55 0 0 0 0] [0 55 0 0 0] [0 0 66 0 0] [0 0 0 55 0] [0 0 0 0 44]]	Confusion Matrix [[46 3 0 1 0] [6 42 6 3 0] [1 10 57 9 0] [2 0 3 42 6] [0 0 0 0 38]]

Table 9 confusion matrices of Random Forest Classifier trials

5.3.1 Random Forest Classifier (no feature extraction after filtering and PCA)

```
timesteps = 244
channels_num = 14
# Reshaping the dataset to 2d from 3d
Xnew = np.reshape(X.copy(),(trials,timesteps*channels_num))
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(Xnew,y,test_size=0.2,shuffle=True,stratify=y)
ynew = y_train-1
# RandomForestClassifier model
model = RandomForestClassifier(n_estimators=1000)
# Train model
model.fit(X_train,ynew)
# Test model
tic = time.perf_counter()
predicted = model.predict(X_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper
# Exctract and save output metrics
output = "Results for RandomForestClassifier (nothing after PCA)\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(predicted,y_test-1)) +'\n'
output = output + str(classification_report(predicted,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 25 script for Random Forest Classifier

Parameters

No feature extraction after filtering and PCA was used

This Random Forest model used the following parameters:

Number of estimators: 1000

Results

Results for Random Forest Classifier (no feature extraction after filtering and PCA)

```
Results for RandomForestClassifier (nothing after PCA)

Confusion Matrix
[[51  2  0  1  0]
 [ 2 52  3  0  1]
 [ 2  0 60  3  0]
 [ 0  1  3 51  3]
 [ 0  0  0  0 40]]
              precision    recall  f1-score   support

           0       0.93      0.94      0.94        54
           1       0.95      0.90      0.92        58
           2       0.91      0.92      0.92        65
           3       0.93      0.88      0.90        58
           4       0.91      1.00      0.95        40

    accuracy                           0.92       275
   macro avg       0.92      0.93      0.93       275
weighted avg       0.92      0.92      0.92       275

FPS: 11434.14364739026FPS
```

Code Snippet 26 results for Random Forest Classifier

5.3.2 CSP + Random Forest Classifier

```
# CSP
num_components = 8
csp = CSP(n_components=num_components, reg=None, log=None, norm_trace=False,cov_est='concat',transform_into='csp_space')
# Fit and transform CSP values
csp.fit(X,y)
S = csp.transform(X)
csp.plot_patterns(epochs.info, ch_type='eeg', size=1.5)
# LSTM
timesteps = 244
channels_num = 14
# Reshaping the dataset to 2d from 3d
S = np.reshape(S,(trials,timesteps*num_components))
# Splitting the dataset
S_train, S_test, y_train, y_test = train_test_split(S,y,test_size=0.2,shuffle=True,stratify=y)
ynew = y_train-1
# RandomForestClassifier
model = RandomForestClassifier(n_estimators=1000)
# Train model
model.fit(S_train,ynew)
# Test model
tic = time.perf_counter()
predicted = model.predict(S_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper
# Exctract and save output metrics
output = "Results for CSP + RandomForestClassifier\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(predicted,y_test-1)) +'\n'
output = output + str(classification_report(predicted,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 27 script for CSP + Random Forest Classifier

Parameters

CSP was used as a feature extraction and dimensionality reduction method to 8 features.

This Random Forest model used the following parameters:

Number of estimators: 1000

Results

Results for CSP + Random Forest Classifier

```
Results for CSP + RandomForestClassifier

Confusion Matrix
[[55  0  0  0  0]
 [ 0 55  0  0  0]
 [ 0  0 66  0  0]
 [ 0  0  0 55  0]
 [ 0  0  0  0 44]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        55
           1       1.00      1.00      1.00        55
           2       1.00      1.00      1.00        66
           3       1.00      1.00      1.00        55
           4       1.00      1.00      1.00        44

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275

FPS: 13330.435850089169FPS
```

Code Snippet 28 results for CSP + Random Forest Classifier

5.3.3 FFT + Random Forest Classifier

```
# FFT
timesteps = 244
Xnew = np.transpose(X,(0,2,1)).copy()
# Get FFT values and perform transformation
yf = fft(Xnew[0])
xf = fftfreq(timesteps, 1 / 128)
# Show plot and save
plt.plot(xf, np.abs(yf))
plt.savefig('Figures/Fig4_FFTtransformedsample.jpg',bbox_inches='tight')
plt.show()
Xfft = Xnew.copy()
# Perform fft for whole dataset
for i in range(0, trials):
    Xfft[i] = fft(Xnew[i])
channels_num = 14
# Reshaping the dataset to 2d from 3d
XfftNew = np.reshape(Xfft.copy(),(trials,timesteps*channels_num))
# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(XfftNew,y,test_size=0.2,shuffle=True,stratify=y)
ynew = y_train-1
print(Xnew.shape)
# RandomForestClassifier model
model = RandomForestClassifier(n_estimators=1000)
model.fit(X_train,ynew)
tic = time.perf_counter()
predicted = model.predict(X_test)
toc = time.perf_counter()
timeTaken = toc-tic
timeTakenper = timeTaken/1325
FPSscore = 1/timeTakenper
# Exctract and save output metrics
output = "Results for FFT + RandomForestClassifier\n\n"
output = output + 'Confusion Matrix\n' + str(confusion_matrix(predicted,y_test-1)) +'\n'
output = output + str(classification_report(predicted,y_test-1)) + '\n'
output = output + "FPS: " + str(FPSscore) + "FPS\n\n"
print(output)
f = open("results.txt", "a")
f.write(output)
f.close()
```

Code Snippet 29 script for FFT + Random Forest Classifier

Parameters

FFT was used as a feature extraction method to better transform the signals for the ML model

This Random Forest model used the following parameters:

Number of estimators: 1000

Results

Results for FFT + Random Forest Classifier

```
Results for FFT + RandomForestClassifier

Confusion Matrix
[[46  3  0  1  0]
 [ 6 42  6  3  0]
 [ 1 10 57  9  0]
 [ 2  0  3 42  6]
 [ 0  0  0  0 38]]
              precision    recall  f1-score   support

           0       0.84      0.92      0.88        50
           1       0.76      0.74      0.75        57
           2       0.86      0.74      0.80        77
           3       0.76      0.79      0.78        53
           4       0.86      1.00      0.93        38

    accuracy                           0.82       275
   macro avg       0.82      0.84      0.83       275
weighted avg       0.82      0.82      0.82       275

FPS: 9820.598055849263FPS
```

Code Snippet 30 results for FFT + Random Forest Classifier

5.3.4 Random Forest Classifier Summary

CSP + Random Forest Classifier performed the best among all the scores, with an 100% accuracy on the test set, which the model had not seen before.
Best Configuration: FFT + Random Forest Classifier

5.4 Overall Results

Best performing configuration per model	Accuracy Score (f1-score)	FPS
FFT + LSTM	0.97	1047
CSP + SVM	0.97	1787
CSP + Random Forest Classifier	1.00	13330

Table 10 accuracy and FPS of best performing configuration per model

Best configuration is highlighted yellow

From these results, we can see that CSP + Random Forest Classifier was the best Model configuration out of all the experimental setups. However, CSP + SVM and FFT + LSTM were not far behind, with only a 3% difference between accuracy scores. CSP + Random Forest Classifier displayed a much higher inference FPS, nearly 10x the FPS of LSTM models.

6. Discussion

While I had expected the LSTM models to display the best results, this was not the case. In fact, 2 of the LSTM models produced the lowest results seen in this experiment. This was rather disappointing, as the training times were much longer than the other two models. However, LSTM trials were limited by technical constraints, at 100 Epochs. In further experimentation, a higher number of epochs could be trialled to determine whether it could achieve accuracies on par or above the other two models. The big surprise was the high accuracy of the Random Forest Classifier after CSP, with an accuracy of 100% for the test set. This shows that CSP was able to extract the features in a way that would be easily determinable by random forest model, which also runs very quickly, with an extremely fast inference FPS of 13,330. This would easily keep up with live data being fed to it.

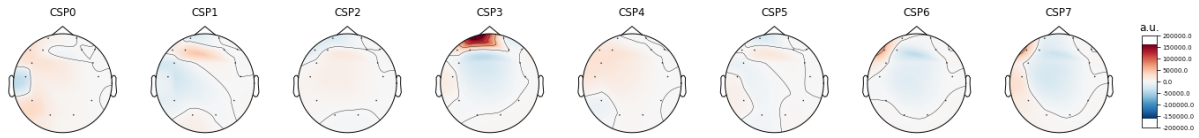


Figure 3 visualisation of CSP feature extraction

Another interesting thing to note that while the performances were not as great as the top performing configurations, CSP and FFT both displayed an increase in accuracy in at least 1 model, showing that they are both managing to extrapolate features from the dataset that these models would otherwise struggle to discover. Visualisations of this feature extraction can be seen in *figures 3 and 4.*

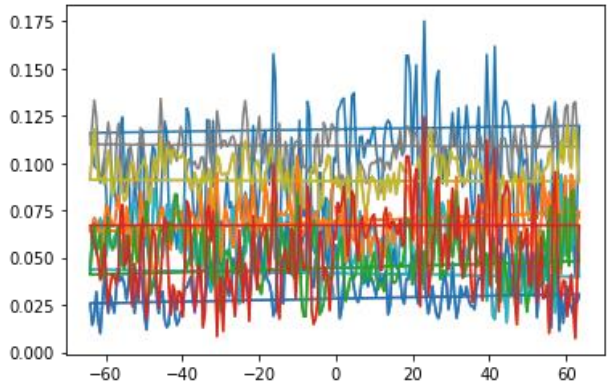


Figure 4 visualisation of fast Fourier Transformation

Another possible path of further experimentation could be through the examination of the effects of the PCA on the overall accuracy. Unfortunately, this could not be tested in our experiment as PCA was implemented as a workaround to a bug caused by the CPU being used.

7. Conclusion

In summary, through the experimentation with different configurations and models, we were achieving an accuracy of up to 100% using a 5th Butterworth Bandpass Filter, PCA, CSP through a Random Forest Classifier model. This shows great promise, as Random Forest Classifiers are quick to train and could be effectively deployed on a device with low technical specification, for both training and deployment. With further fine tuning and research, I believe that machine learning is a viable way to implement an effective and reliable method to estimate a user's intended target point on a screen based on eye gaze. While artificial neural network algorithms and models like the LSTM are gaining a lot of attention for their versatility and wide range of uses, LSTMs were out classed in this experiment by Random Forest Classifiers, showing that feature extraction techniques play a large role in the effectiveness of a model, especially in complex datasets such as EEG signal data.

8. List of Appendices/Submitted Items

- Zip file containing:
    - Jupyter python Notebook file containing code used for the experiment
    - results.txt – The text file to which results from the python notebook are written
    - Figures folder containing *figures 1-4* in SVG format
- Technical Report (this document)
- Video Presentation outlining experiment and results
- Dataset converted to csv file (if filesize permits)

9. Bibliography

Centre for Research and Technology Hellas. (2021, 05). *EEG SSVEP Dataset III*. Retrieved from MAMEM: https://www.mamem.eu/results/datasets/

MNE tools. (2021, Jan 25). *LinAlgError: SVD did not converge when plotting data (raw.plot()) #8784* . Retrieved from github.com: https://github.com/mne-tools/mne-python/issues/8784

Rashid, M. &. (2020). Five-Class SSVEP Response Detection using Common- Spatial Pattern (CSP)-SVM Approach. *International Journal of Integrated Engineering 12*, 165-173.

Wei Wu, X. G. (2005). One-Versus-the-Rest(OVR) Algorithm: An Extension of Common Spatial Patterns(CSP) Algorithm to Multi-class Case. *IEEE Engineering in Medicine and Biology 27th Annual Conference*, 2387-2390.