EEG 데이터 전처리 및 시각화 시스템

2024.12.18

발표자 이원준

팀원 윤종현 구나현 박철

6조 컴퓨터 공학과

CONTENTS

EEGPROJECT PRESENTATION

팀원소개

EEG PROJECT
PRESENTATION

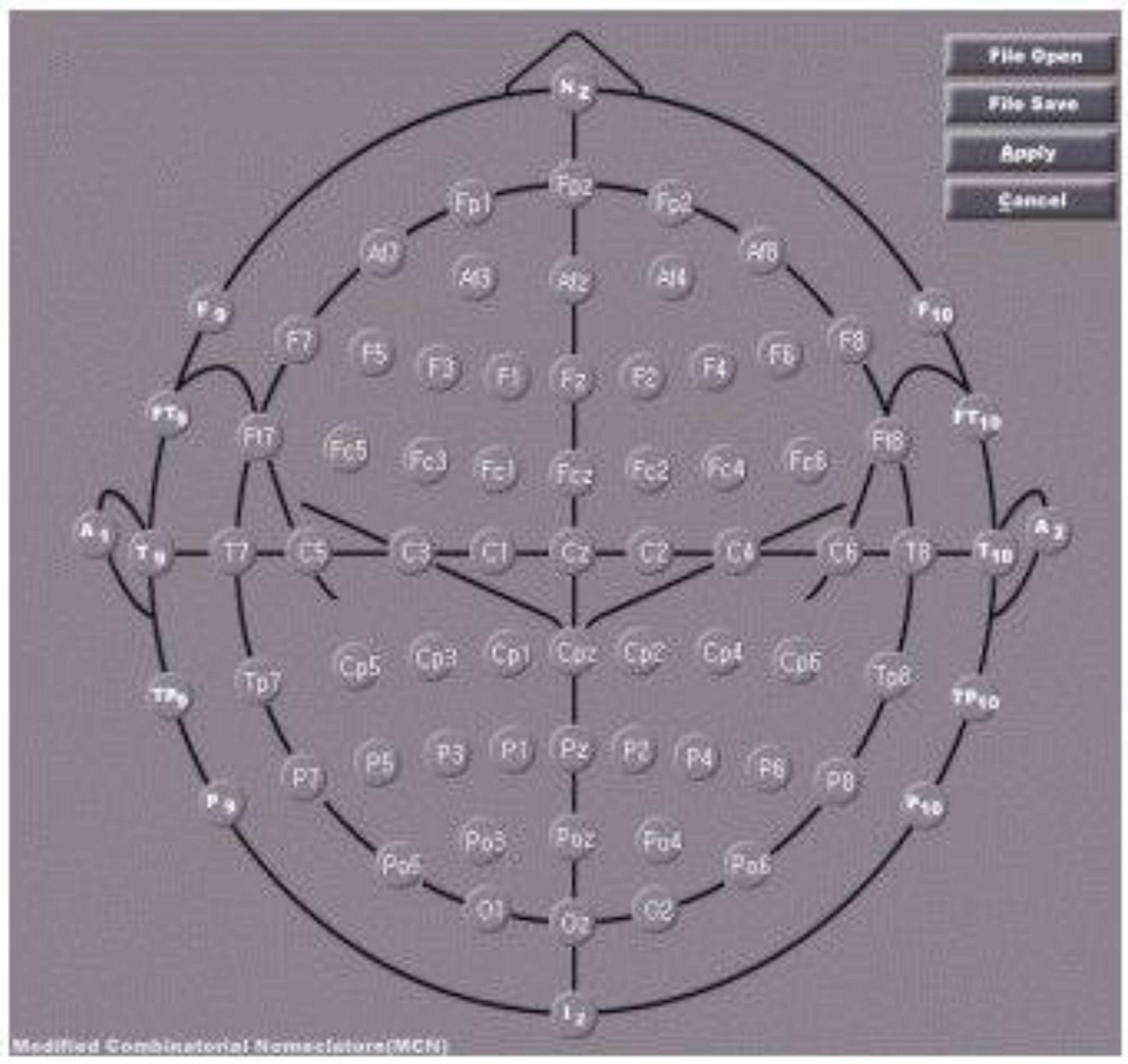
이원준
백엔드
모델학습 및 시각화
발표

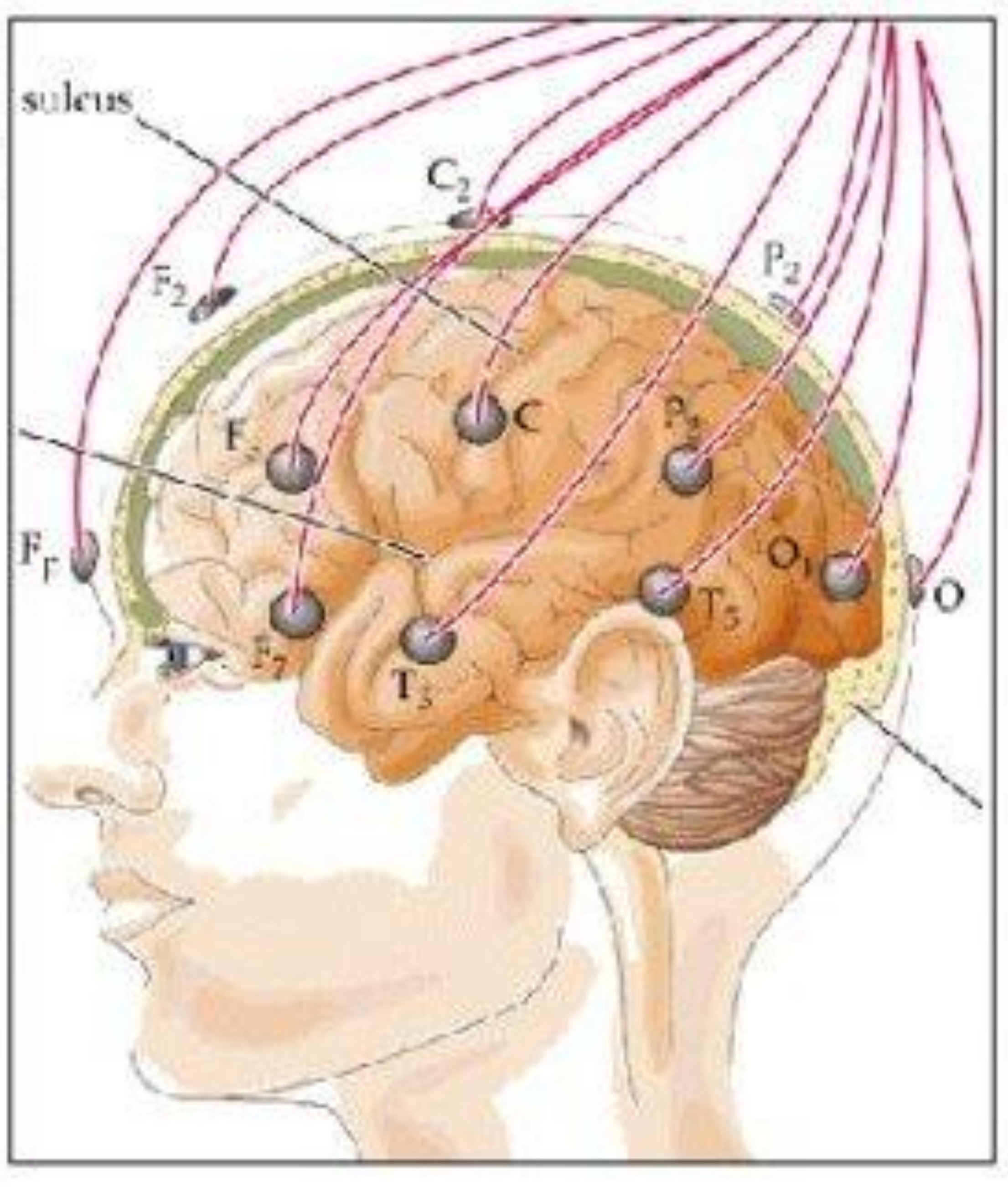
윤종현
백엔드
모델학습 및 백엔드
ppt 제작

구나현
프론트엔드
프론트 및 시각화
ppt 제작

박철
프론트엔드
프론트 및 시각화

EEG 데이터와 뇌파 채널





주제및구현전략

EEG PROJECT
PRESENTATION

GOAL

EEG데이터전처리 및 시각화 시스템

GOAL

뇌 부하정도에 따른 뇌파 데이터 시각화 시스템

목적

피실험자 48명에게서 부하가 높은 뇌파 데이터 (HI 데이터) 그리고 부하가 낮은 데이터 (LOW 데이터)를 사용해서 시각화를 진행

이 EEG 데이터를 가지고 부하가 높은 뇌파 그래프와 부하가 낮은 그래프에서 각 채널의 영향도를 분석하고 전처리 조합 중 어떤 조합이 좋은지 시각화를 진행

분석프로세스

EEG PROJECT
PRESENTATION

시스템FLOW

피실험자 데이터 선택
(40번 ~ 48번)
모델 옵션 선택
(CNN, CNN_FFT)
전처리 옵션 선택
(rmn, ra, avg)

Step1~2

Data 전처리 및
미리 학습시켜 놓은
모델에 적용

Step2~3

d3 를 이용한 시각화

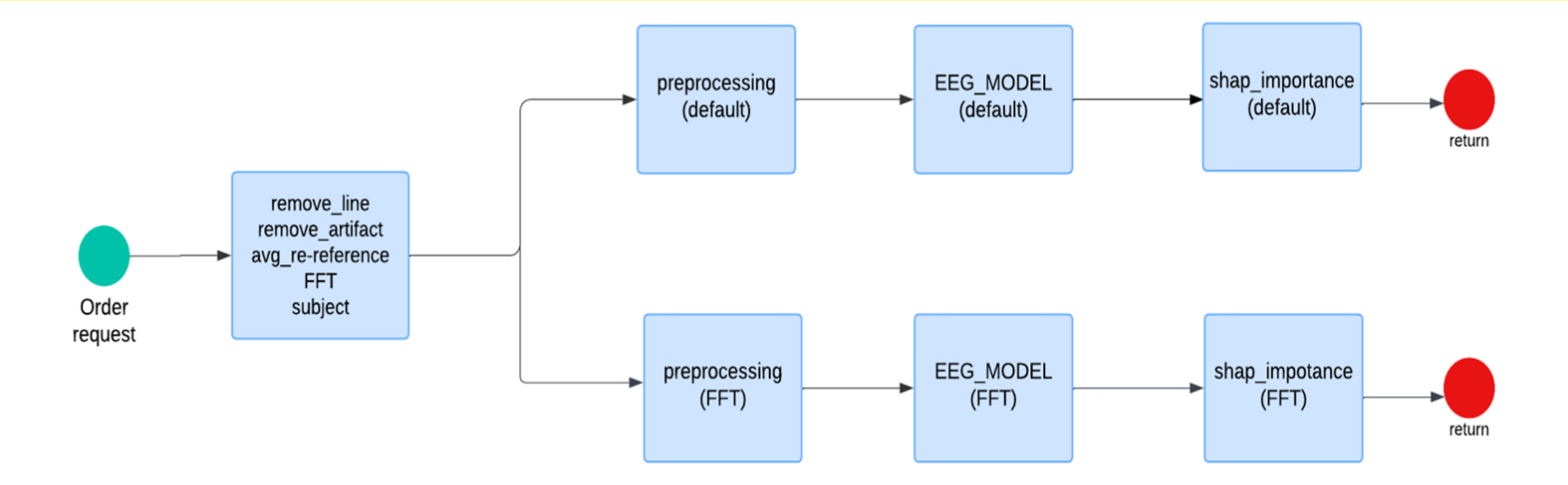
Step3~4

웹페이지에 시각화

Step 1 | Step 2 | Step 3 | Step 4

백엔드

백엔드의FLOW



백엔드

1. 쿼리스트링을통해선택된전처리옵션받아오기

```
data_option = int(request.args.get('data', '0'))
fft_option = request.args.get('fft', 'false').lower() == 'true'
rmn_option = request.args.get('rmn', 'true').lower() == 'true'
ra_option = request.args.get('ra', 'true').lower() == 'true'
avg_option = request.args.get('avg', 'true').lower() == 'true'
```

- subject
- FFT(Fast Fourier Transform)
- remove_line_noise
- remove_artifact
- average re-reference

백엔드

EEG PROJECT
PRESENTATION

```python
def preprocessing(df,channels,rmn,ra,avg):
    if len(df) != 0:
        # 원본 데이터의 주파수
        sfreq = 128

        # 1. NaN removed
        df = df.dropna(axis=0)

        # 2. high pass filter
        with mne.use_log_level(50):
            info = mne.create_info(ch_names=channels, sfreq=sfreq, ch_types='eeg')
            raw = mne.io.RawArray(df[channels].T, info)
            raw.filter(1, 40, fir_design='firwin')


            if (rmn) :
                    raw = remove_line_noise(raw)
            if (ra) :
                    raw = Remove_artifact(raw)

            # 5. 평균 재참조
            if (avg) :
                raw.set_eeg_reference('average')

            # 6. 데이터를 Pandas DataFrame으로 변환
            df = pd.DataFrame(raw.get_data().T , columns=channels)
            df = df.dropna(axis=0)

            # 7. Min-Max 정규화
            scaler = MinMaxScaler(feature_range=(0, 1))  # 0~1 범위로 정규화
            df[channels] = scaler.fit_transform(df[channels])
            return df
```

2.STEW_DATA_SET에해당옵션에맞는전처리수행

사용자가 모델을 CNN (fft = false) 를 선택한 경우

백엔드

EEG PROJECT
PRESENTATION

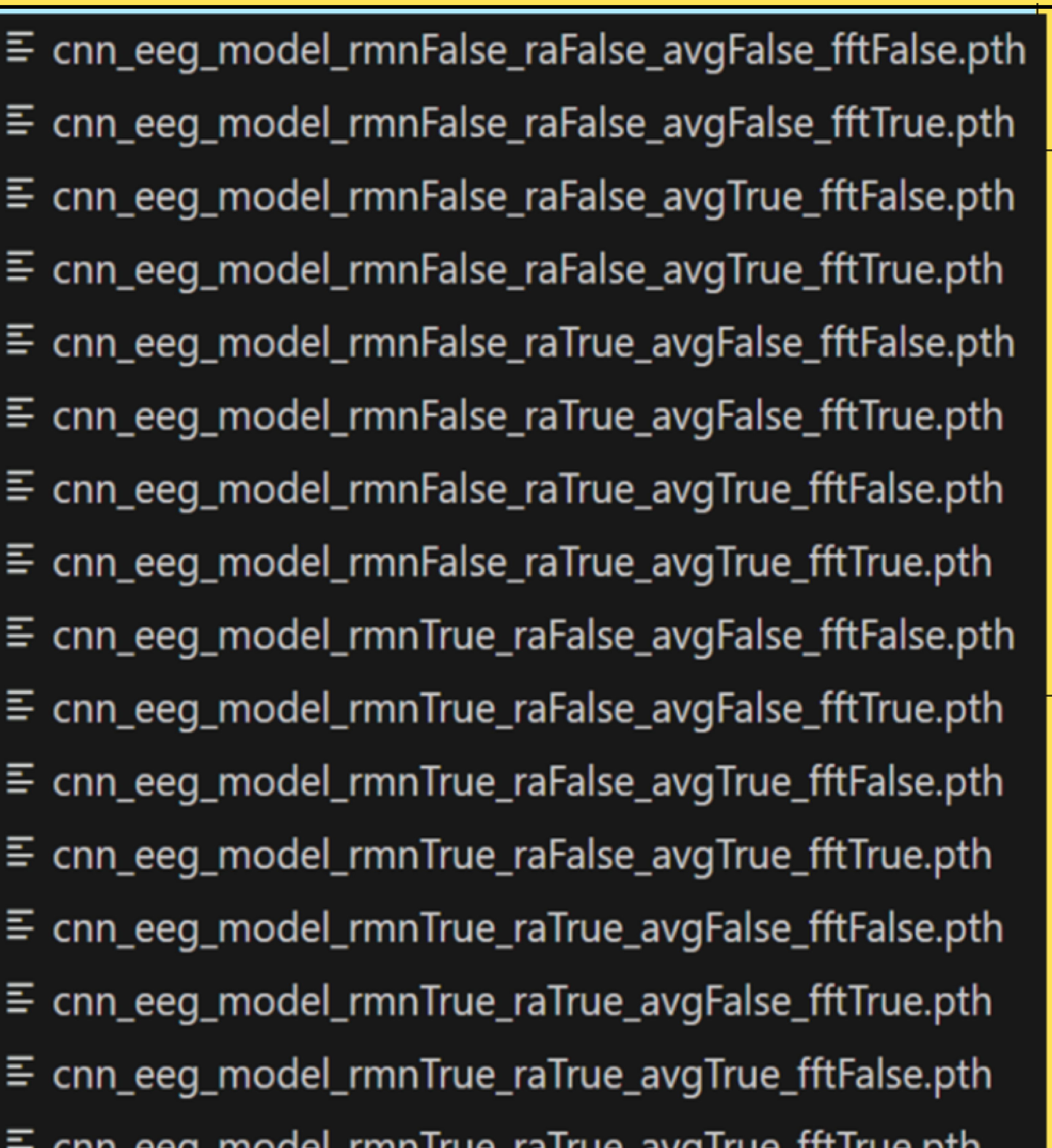
```
def preprocessing_fft(df,channels,freq_bands,rmn,ra,avg):
    if len(df) != 0:
        # 원본 데이터의 샘플링 주파수
        sfreq = 128

        # 1. NaN removed
        df = df.dropna(axis=0)

        # 2. high pass filter
        with mne.use_log_level(50):
            info = mne.create_info(ch_names=channels, sfreq=sfreq, ch_types='eeg') # MNE Info 객체 생성
            raw = mne.io.RawArray(df[channels].T, info)
            raw.filter(1, 40, fir_design='firwin')


            if (rmn) :
                    raw = remove_line_noise(raw)
            if (ra) :
                    raw = Remove_artifact(raw)
            # 5. 평균 재참조
            if (avg) :
                raw.set_eeg_reference('average')

            # 6. 주파수 대역 분리
            band_data = {}
            for band, (l_freq, h_freq) in freq_bands.items():
                band_raw = raw.copy().filter(l_freq, h_freq, fir_design='firwin')
                band_data[band] = pd.DataFrame(band_raw.get_data().T, columns=channels)

            # 7. Min-Max 정규화
            scaler = MinMaxScaler(feature_range=(0, 1))  # 0~1 범위로 정규화
            for band, (l_freq, h_freq) in freq_bands.items():
                band_data[band][channels] = scaler.fit_transform(band_data[band][channels])

            return band_data
```

2.STEW_DATA_SET에해당옵션에맞는전처리수행

사용자가 모델을 CNN_fft (fft=true) 를 선택한 경우

백엔드

EEG PROJECT
PRESENTATION



3.옵션에따라미리학습시켜놓은모델의가중치들을
CNN_EEG모델에적용
(학습된 CNN_EEG가중치)

백엔드

EEG PROJECT
PRESENTATION

```python
def load_basic_models(models_dir=".", model_prefix="cnn_eeg_model"):
    """
    Load all 16 models based on preprocessing options.
    Returns a dictionary with keys as option strings and values as loaded models.
    """
    model_dict = {}
    options = {
        'rmn': [False, True],
        'ra': [False, True],
        'avg': [False, True],
        'fft': [False, True]
    }

    # Generate all combinations of options
    for rmn, ra, avg, fft in product(options['rmn'], options['ra'], options['avg'], options['fft']):
        # Construct model filename
        model_name = f"{model_prefix}_rmn{rmn}_ra{ra}_avg{avg}_fftFalse.pth"
        model_path = os.path.join(models_dir, model_name)

        if not os.path.exists(model_path):
            print(f"Warning: Model file {model_path} not found. Skipping this configuration.")
            continue

        # Initialize model
        model = CNNEEG(input_channel=len(channels))
        try:
            model.load_state_dict(torch.load(model_path, map_location=device))
            model.to(device)
            model.eval()
            key = f"rmn{rmn}_ra{ra}_avg{avg}_fft{fft}"
            model_dict[key] = model
            #print(f"Loaded model: {model_name} as key: {key}")
        except Exception as e:
            print(f"Error loading model {model_name}: {e}")

    return model_dict
```

3.옵션에따라미리학습시켜놓은모델의가중치들을
CNN_EEG모델에적용
(모델불러오기)

백엔드

EEG PROJECT
PRESENTATION

```
def predict_eeg_state(model, preprocessed_data, channels, device):
    model.eval()

    # Preprocess data
    processed_data = preprocessed_data

    # Create sequences
    seq_data = []
    for i in range(0, len(processed_data) - 240, 120):
        seq_data.append(processed_data.iloc[i+120:i+240].values)

    if not seq_data:
        print("Insufficient data for prediction.")
        return None, None

    seq_data = np.array(seq_data).transpose(0, 2, 1)
    seq_tensor = torch.tensor(seq_data, dtype=torch.float32).to(device)

    # Predict
    with torch.no_grad():
        predictions = model(seq_tensor)
        probabilities = predictions.cpu().numpy()
        classes = np.argmax(probabilities, axis=1)

    return classes, probabilities
```

3.옵션에따라미리학습시켜놓은모델의가중치들을

CNN_EEG모델에적용

(전처리된 데이터에 모델 적용)

백엔드

EEG PROJECT PRESENTATION

```
def calculate_shap_values(model, preprocessed_data, channels, device):

    # Ensure the model is in evaluation mode
    model.eval()

    # Preprocess the data for SHAP
    seq_data = []
    for i in range(0, len(preprocessed_data) - 240, 120):
        seq_data.append(preprocessed_data.iloc[i + 120:i + 240].values)

    if not seq_data:
        print("Insufficient data for SHAP analysis.")
        return None, None

    seq_data = np.array(seq_data).transpose(0, 2, 1)  # (batch, channels, time)
    seq_tensor = torch.tensor(seq_data, dtype=torch.float32).to(device)

    # Define SHAP explainer
    explainer = shap.DeepExplainer(model, seq_tensor)  # Pass the model and some baseline data
    shap_values = explainer.shap_values(seq_tensor)  # Compute SHAP values

    # Average SHAP values across time steps and sequences to get channel-level importance
    shap_importance = np.mean(np.abs(shap_values[0]), axis=(0, 2))  # (channels,)

    # Pair channels with their importance scores
    channel_importance = dict(zip(channels, shap_importance))

    return shap_values, channel_importance
```

4. Shap을 이용하여 모델의 예측에 영향을 미치는 채널별 중요도 계산 및 정규화

백엔드

EEG PROJECT
PRESENTATION

```
return jsonify({
    'before_high' : before_high,
    'before_low' : before_low,
    'after_high': high_dict,
    'after_low': low_dict,
    'predicted_high': predicted_result_high,
    'predicted_low': predicted_result_low,
    'stats_high': stats_high,
    'stats_low': stats_low,
    'importance_high': importance_high,
    'importance_low': importance_low,
    'shap_result' : shap_result,
})
```

5. 반환값

'before_high'	:	전처리 전 high
'before_low'	:	전처리 전 low
'after_high'	:	전처리 후 high
'after_low'	:	전처리 후 low
'predicted_high'	:	예측된 high
'predicted_low'	:	예측된 low
'stats_high'	:	stats_high
'stats_low'	:	stats_low
'importance_high'	:	importance_high
'importance_low'	:	importance_low
'shap_result'	:	SHAP을 이용한 채널별 중요도

시스템시연