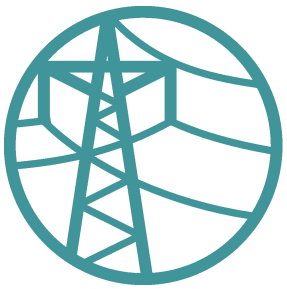


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Daniel Kleczyński 3AA/EF-DI

Wykorzystanie rekurencyjnych sieci neuronowych w dekodowaniu intencji ruchowych na podstawie sygnałów EEG

Prijekt inżynierski

Opiekun projektu:
prof. dr hab. inż. Jacek Kluska

Rzeszów, 2024

Spis treści

Projekt ma na celu stworzenie modelu rekurencyjnych sieci neuronowych (RNN), który może być wykorzystany do dekodowania dziewięciu różnych intencji ruchowych na podstawie sygnałów EEG w czasie rzeczywistym dla interfejsów mózg-komputer. Analizowane są różne architektury sieci LSTM, w tym ilość warstw, dwukierunkowość, oraz wielkość warstwy ukrytej. Badania obejmują także transformację falkową, z uwzględnieniem typu falki, długości sekwencji i rozdzielczości transformacji. Model jest trenowany na specyficznym zbiorze danych EG Motor Movement/Imagery Dataset"**??**, co pozwala na dokładne dostosowanie i optymalizację modelu. W ramach projektu rozwijane jest środowisko do testowania i szkolenia modeli, umożliwiające precyzyjną regulację hiperparametrów, takich jak wielkość wsadu, współczynnik uczenia, wielkość warstwy ukrytej oraz długość sekwencji. Wyniki mają na celu nie tylko opracowanie efektywnego modelu, ale również przyczynienie się do rozwoju technologii interfejsów mózg-komputer, zwiększając ich funkcjonalność i efektywność. Wyniki te mogą pomóc osobom niepełnosprawnym, ułatwiając komunikację i interakcję ze światem zewnętrznym, oraz przyspieszyć rozwój systemów BCI, otwierając nowe możliwości dla technologii wspomagających. Co ważne, opierając się na możliwościach technologii wspomagających bez konieczności ingerencji w ciało ludzkie, EEG stanowi zewnętrzne urządzenie, co dodatkowo zwiększa dostępność i bezpieczeństwo stosowania tych rozwiązań.

1. Wstęp

Stworzenie modeli opartdego na siechach rekurecyjnych jest wymagające pod względem odpowiednie dobrania hiperparametrów co wiąrze sie z wieloma próbami oraz w celu idh dostorjenia jeśli połaczym to z faktem iż chcemy testować różene archtektury lub podejścia do przetwarania danych ilośćiteracji uczniea wysoce zwrasta.

Podczas pracy nad projektem, zauważono, że zarządzanie środowiskiem projektowym i automatyzacja procesów są kluczowe dla zapewnienia efektywności i powtarzalności w pracy nad zaawansowanymi projektami inżynierskimi. W projekcie wykorzystano następujące narzędzia do automatyzacji i zarządzania środowiskiem: Docker, Poetry, Git, oraz procesy automatyzacji zadań.

2. Specyfikacja sprzętu

2.1. Serwer

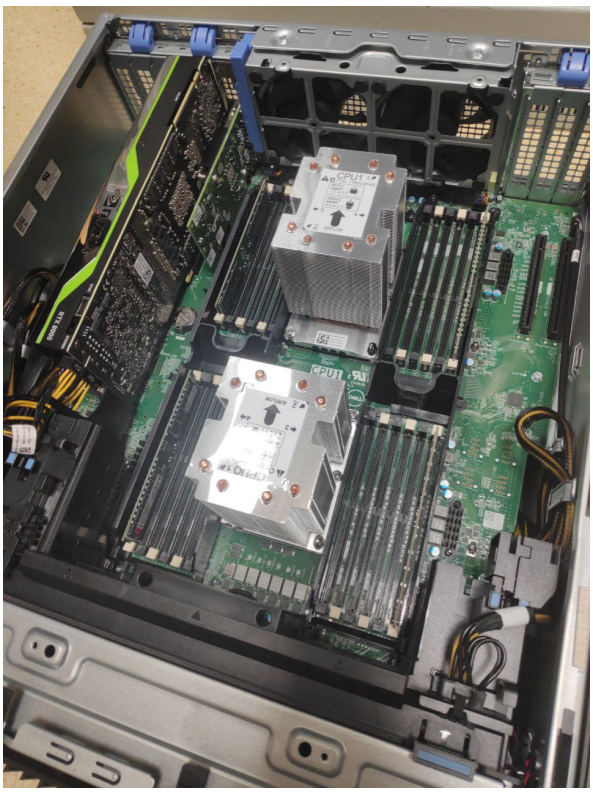
Serwer używany w projekcie 2.1 jest wysoce wydajnym systemem, przystosowanym do zadań wymagających intensywnych obliczeń. Specyfikacja techniczna serwera jest następująca:

- **Procesor**: 32 x Intel(R) Xeon(R) Gold 6234 CPU @ 3.30GHz (2 gniazda)
- **Pamięć RAM**: 256 GB
- **Dysk SSD NVMe**: 1 TB
- **Karta graficzna**: NVIDIA RTX 8000 z 48 GB pamięci VRAM
- **Dyski HDD**: 2 x 16 TB
- **System operacyjny**: Debian GNU/Linux 12

2.2. Laptop do pracy lokalnej

Laptop służy do lokalnego przetwarzania danych, szybkich poprawek i testowania kodu przed uruchomieniem pełnoskalowych eksperymentów na serwerze. Specyfikacja laptopa:

- **Model**: ROG Flow Z13 GZ301ZC_GZ301ZC



Rysunek 2.1: Serwer służący do trenowania modeli oraz poszukiwania hiperparametrów.

- **Procesor**: 12th Gen Intel i7-12700H (20 wątków) @ 4.600GHz
- **Karty graficzne**:
  - NVIDIA GeForce RTX 3050 Mobile
  - Intel Alder Lake-P
- **Pamięć RAM**: 15680 MiB
- **System operacyjny**: Pop!_OS 22.04 LTS x86_64

3. Zarządzanie środowiskiem i automatyzacja

3.1. Docker

Docker jest narzędziem do konteneryzacji aplikacji, które ułatwia ich wdrażanie i skalowanie. Zapewnia izolację aplikacji od środowiska, co zwiększa niezawodność i bezpieczeństwo.

```yaml
version: '3.9'

services:
  EEG_train_DB:
  image: postgres:14-alpine
  restart: always
  expose:
    - "5433"
  ports:
    - "5433:5433"
  volumes:
    - ./db:/var/lib/postgresql/data
  environment:
    - POSTGRES_PASSWORD=1234
    - POSTGRES_USER=user
    - POSTGRES_DB=dbtrain
  command: -p 5433

  EEG_val_DB:
  image: postgres:14-alpine
  restart: always
  expose:
    - "5434"
  ports:
    - "5434:5434"
  volumes:
    - ./db1:/var/lib/postgresql/data
  environment:
    - POSTGRES_PASSWORD=1234
    - POSTGRES_USER=user
    - POSTGRES_DB=dbval
  command: -p 5434
```

Listing 1: Docker Compose configuration

3.2. Poetry

Poetry to narzędzie do zarządzania zależnościami Pythona, które ułatwia zarządzanie pakietami i wersjami.

```toml
[tool.poetry]
name = "decoding_of_eeg"
version = "0.1.0"
description = ""
authors = ["Daniel Kleczynski <danielkleczynski@gmail.com>"]
license = "MIT"

[tool.poetry.dependencies]
python = "^3.11"
absl-py = "^2.1.0"
torch = "^2.3.1"
pytorch-lightning = "^2.2.5"
ray = "^2.24.0"
```

```
numpy = "^1.26.4"
pandas = "^2.2.2"
matplotlib = "^3.9.0"
tqdm = "^4.66.4"
psycopg2-binary = "^2.9.9"
PyWavelets = "^1.6.0"
mne = "^1.7.0"
torchmetrics = "^1.4.0"

[tool.poetry.dev-dependencies]

[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"

```

Listing 2: Poetry configuration

3.3. Git

Git jest systemem kontroli wersji używanym do zarządzania kodem źródłowym w projektach programistycznych.

Korzyści:

- Śledzenie zmian w kodzie.
- Uruchamianie testów.

3.4. Automatyzacja zadań

Automatyzacja zadań, takich jak łączenie się z serwerem oraz przekierowywanie portów, jest kluczowa dla efektywnego zarządzania i monitorowania postępów w pracy nad zaawansowanymi projektami inżynierskimi. Skrypty opisane poniżej pozwalają na automatyzację i uproszczenie procesów, które są często powtarzane, co zwiększa efektywność pracy i pozwala na skupienie się na istotnych aspektach projektu.

3.4.1. Skrypt do łączenia się przez VPN

Skrypt do łączenia się przez VPN automatyzuje proces inicjalizacji połączenia VPN, co jest niezbędne do zdalnego dostępu do zasobów sieciowych w sposób bezpieczny.

```
#!/bin/bash
# Skrypt do laczenia sie z VPN

VPN_SERVER_IP="adres_ip_serwera_vpn"
```

```
VPN_USER="nazwa_uzytkownika"
VPN_PASSWORD="haslo"

echo "laczenie z VPN..."
openvpn --config $VPN_SERVER_IP --auth-user-pass <(echo -e "
   $VPN_USER\n$VPN_PASSWORD")
echo "Polaczono z VPN."
```

Listing 3: Skrypt do łączenia się przez VPN

3.4.2. Skrypt do przekierowywania portów

Przekierowanie portów jest kluczowe w celu uzyskania dostępu do usług uruchomionych na zdalnym serwerze jako lokalne. Skrypt do przekierowywania portów automatyzuje ustawienie tuneli SSH, co ułatwia bezpieczny dostęp do zdalnych aplikacji.

```
#!/bin/bash
# Skrypt do przekierowywania portow

LOCAL_PORT="8888"
REMOTE_PORT="8888"
REMOTE_IP="adres_ip_serwera"

echo "Przekierowywanie portu lokalnego $LOCAL_PORT na port
   $REMOTE_PORT na serwerze $REMOTE_IP..."
ssh -L ${LOCAL_PORT}:${REMOTE_IP}:${REMOTE_PORT} $REMOTE_IP
echo "Przekierowanie ustawione."
```

Listing 4: Skrypt do przekierowywania portów

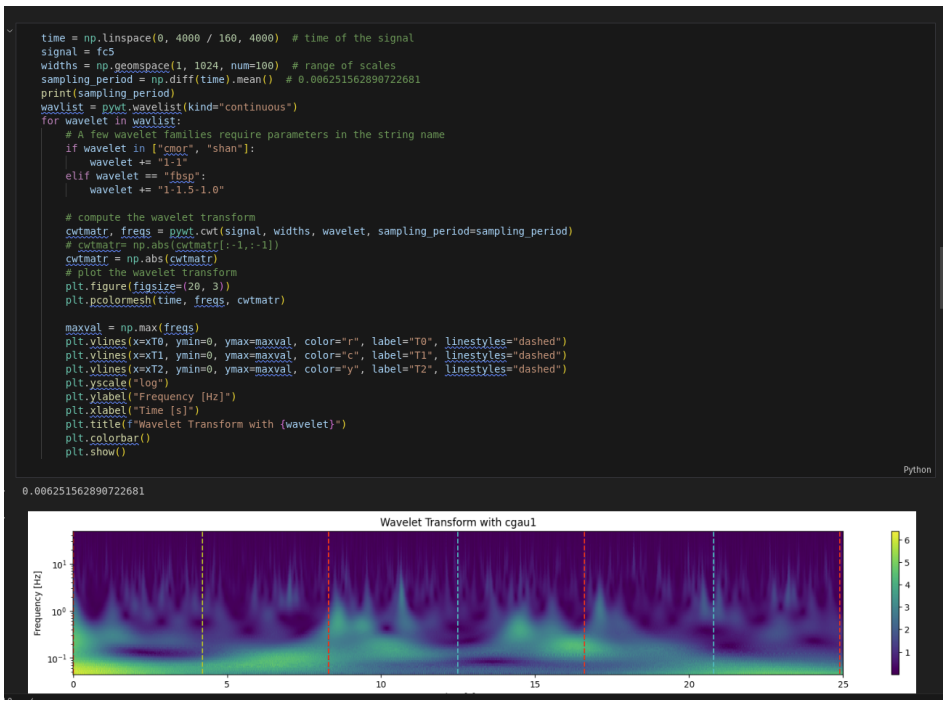
Używanie tych skryptów pozwala na szybką i efektywną obsługę połączeń sieciowych i przekierowań, co jest nieocenione w środowisku badawczym i rozwojowym, gdzie czas i niezawodność są na wagę złota.

3.5. Jupyter notebook

Dzieki użyciu Jupyter notebook 3.2 oraz jupyter lab jesteśmy w stanie w łątwy sposób testować nowe metody analizy danych bądź nowe modele. Bezpośrednio uruchamiając nowy kod na serwwerze z poziomu urządznia na którym pracujemy. Przyspiesza to proces testowania nowych rozwiązań oraz pozwala na szybkie zobaczenie wyników.

4. Środowisko obliczeniowe

W ramach projektu analizy danych EEG kluczowe jest odpowiednie skonfigurowanie środowiska obliczeniowego, które musi spełniać następujące wymagania:



Rysunek 3.2: Zrzut ekrany z Jupyter Notebook podczas testowania transformacji falkowej

- **Wsparcie dla GPU:** Znaczące przyspieszenie obliczeń jest niezbędne, szczególnie przy trenowaniu głębokich sieci neuronowych i przetwarzaniu dużych zbiorów danych.

- **Możliwość pracy rozproszonej:** Środowisko powinno umożliwiać efektywne korzystanie z wielu urządzeń jednocześnie, co jest kluczowe przy skalowalnych eksperymentach i większej ilości danych.

- **Zarządzanie zasobami i optymalizacja hiperparametrów:** Automatyzacja zarządzania zasobami i procesów optymalizacji, aby maksymalizować efektywność uczenia maszynowego.

4.1. PyTorch i PyTorch Lightning

PyTorch jest zaawansowaną biblioteką do budowy i trenowania sieci neuronowych, a PyTorch Lightning to nakładka, która upraszcza i automatyzuje wiele aspektów pracy z PyTorch. Oto kluczowe zalety obu technologii:

- **Dynamiczny graf obliczeniowy (PyTorch):** Umożliwia elastyczność w pro-

jektowaniu architektury modelu, co jest korzystne przy przetwarzaniu złożonych danych jak EEG.

- **Przyspieszenie GPU (PyTorch):** Kluczowe dla efektywnego trenowania modeli, szczególnie przy dużych zbiorach danych.

- **Uproszczony proces trenowania (PyTorch Lightning):** Automatyzuje rutynowe zadania, pozwalając skupić się na architekturze modelu.

- **Zaawansowane logowanie i monitorowanie (PyTorch Lightning):** Integracja z TensorBoard czy MLFlow umożliwia szczegółowe śledzenie i optymalizację procesów.

4.1.1. Implementacja modelu i trenera

```
class CWT_EEG(LightningModule):
    def __init__(
            self,
            batch_size,
            sequence_length,
            input_size,
            hidden_size,
            num_layers,
            lr,
            label_smoothing=0,
    ):
        super().__init__()
        self.save_hyperparameters()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
    batch_first=True)
        self.fc = nn.Linear(hidden_size, self.num_of_classes)
```

Listing 5: Klasa modelu CWT_EEG

```
def forward(self, x):
    h0 = torch.zeros(self.num_layers, x.size(0), self.
   hidden_size).to(x.device)
    c0 = torch.zeros(self.num_layers, x.size(0), self.
   hidden_size).to(x.device)
    out, (hn, cn) = self.lstm(x, (h0, c0))
    out = hn[-1, :, :]
    out = self.fc(out)
```

Listing 6: Funkcja forward w modelu CWT_EEG

```
import datetime
from pytorch_lightning import Trainer, loggers

current_time = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M
   -%S")
```

```
logger = loggers.TensorBoardLogger("logs", name=f"CWT_EEG_{
    current_time}")
model = CWT_EEG(batch_size=11, sequence_length=10, input_size
    =640, num_layers=3, hidden_size=100, lr=0.001).to(device)
trainer = Trainer(max_epochs=100, logger=logger)
trainer.fit(model)
```

Listing 7: Konfiguracja i uruchomienie trenera w PyTorch Lightning

4.2. Ray i RAITune

Ray umożliwia efektywne zarządzanie rozproszonymi zasobami komputerowymi, maksymalizując ich wykorzystanie poprzez równoległe uruchamianie wielu treningów z różnymi zestawami hiperparametrów. Dzięki temu, modele mogą być trenowane znacznie szybciej, co jest kluczowe w środowiskach badawczych i produkcyjnych, gdzie czas jest krytycznym zasobem.

RAITune rozszerza możliwości Ray poprzez implementację strategii optymalizacji bayesowskiej do automatycznego dobierania i zarządzania hiperparametrami w trakcie eksperymentów. To podejście pozwala na dynamiczne dostosowywanie parametrów w odpowiedzi na wyniki treningu w czasie rzeczywistym, co skutkuje lepszą wydajnością modeli bez konieczności ręcznego eksperymentowania.

```
def train_cwt_eeg(config):
    wandb_logger = WandbLogger(project="EEG")
    engine_train = create_engine("postgresql+psycopg2://user
    :1234@0.0.0.0:5433/dbtrain", echo=True, pool_size=10)
    engine_val = create_engine("postgresql+psycopg2://user:1234
    @0.0.0.0:5434/dbval", echo=True, pool_size=10)

    model = CWT_EEG_CrossPersonValidation(
        batch_size=config['batch_size'],
        sequence_length=config['sequence_length'],
        input_size=config['input_size'],
        hidden_size=config['hidden_size'],
        num_layers=config['num_layers'],
        lr=config['lr'],
        label_smoothing=config.get('label_smoothing', 0),
        engine_train=engine_train,
        engine_val=engine_val
    )
    checkpoint_callback = ModelCheckpoint(monitor='val_loss',
    dirpath='model_checkpoints', filename='model-{epoch:02d}-{
    val_loss:.2f}', save_top_k=3, mode='min')
    early_stop_callback = EarlyStopping(monitor='val_loss',
    min_delta=0.00, patience=3, verbose=True, mode='min')

    trainer = Trainer(max_epochs=10, logger=wandb_logger,
    enable_progress_bar=False, callbacks=[checkpoint_callback,
```

```
    early_stop_callback, TuneReportCallback({"loss": "ptl/
    val_loss"}, on="validation_end")])
21   trainer.fit(model)
```

Listing 8: Integracja Ray i RAITune do efektywnego zarządzania i optymalizacji hiperparametrów

```
def train_cwt_eeg(config):
    wandb_logger = WandbLogger(project="EEG")
    model = CWT_EEG_CrossPersonValidation(
        batch_size=config['batch_size'],
        sequence_length=config['sequence_length'],
        input_size=config['input_size'],
        hidden_size=config['hidden_size'],
        num_layers=config['num_layers'],
        lr=config['lr'],
        label_smoothing=config.get('label_smoothing', 0),
        engine_train=engine_train,
        engine_val=engine_val
    )
    checkpoint_callback = ModelCheckpoint(monitor='val_loss',
    dirpath='model_checkpoints', filename='model-{epoch:02d}-{
    val_loss:.2f}', save_top_k=3, mode='min')
    early_stop_callback = EarlyStopping(monitor='val_loss',
    min_delta=0.00, patience=3, verbose=True, mode='min')

    trainer = Trainer(max_epochs=10, logger=wandb_logger,
    enable_progress_bar=False, callbacks=[checkpoint_callback,
    early_stop_callback, TuneReportCallback({"loss": "ptl/
    val_loss"}, on="validation_end")])
    trainer.fit(model)
```

Listing 9: Integracja Ray i RAITune w treningu modelu CWT_EEG

5. Przetwarzanie i wizualizacja danych

5.1. Odczyt danych

Za pomocą bibloteki mne wczytujemy dane z plików .edf, a następnie przetwarzamy je w celu uzyskania odpowiedniego formatu danych do trenowania modelu. w tym przypadku dane przetymywane są w formie tabularycznej za pomocą biblioteki pandas.

```
reader = mne.io.read_raw_edf(path, preload=True)
annotations = reader.annotations
codes = annotations.description
df = pd.DataFrame(reader.get_data().T, columns=[channel.replace(
    ".", "") for channel in reader.ch_names])
df = df[~(df == 0).all(axis=1)]
timeArray = np.array([round(x, 10) for x in np.arange(0, len(df)
    / 160, 0.00625)])
codeArray = []
counter = 0
```

```
for timeVal in timeArray:
    if (timeVal in annotations.onset):
        counter += 1
    code_of_target = int(codes[counter - 1].replace("T", ""))
    codeArray.append(code_of_target)
df["target"] = np.array(codeArray).T
return df
```

Listing 10: Odczyt z plików .edf do DataFrame za pomocą mne oraz pandas

5.2. Transformacja falkowa

Do przeprowadzenia transformacji falkowej używamy biblioteki `PyWavelets`. Typ falki użyty w naszym przypadku to `cgau4`, który jest odpowiedni do analizy sygnałów EEG ze względu na swoją zdolność do rozróżniania różnych częstotliwości z dużą precyzją czasową. Ustalona długość sekwencji wynosząca 4000 próbek została wybrana ze względu na ograniczenia pamięciowe, co umożliwia efektywne przetwarzanie danych bez konieczności ładowania wszystkich próbek jednocześnie do pamięci.

```
import numpy as np
import pywt
from tqdm import tqdm

def df_to_CWTdb(df, conn, num_of_rows=1000, wave="cgau4", frq=160, resolution=100):
    num_chunks = len(df) // num_of_rows + (1 if len(df) % num_of_rows != 0 else 0)

    # Create a tqdm progress bar for the loop
    for i in tqdm(range(0, len(df), num_of_rows), total=num_chunks, desc="Processing"):
        end_index = i + num_of_rows
        if end_index > len(df):
            end_index = len(df)
        signals = df.iloc[i:end_index].values
        list_cwt = []

        if signals.shape == (num_of_rows, 65):
            signals = signals.transpose(1, 0)

        for signal in signals[:-1]:  # Exclude the last item assuming it's the target
            signal = (signal - np.min(signal)) / (np.max(signal) - np.min(signal))
            time = np.linspace(0, len(signal) / frq, len(signal))
            widths = np.geomspace(1, 200, num=resolution)
            sampling_period = np.diff(time).mean()
            cwtmatr, freqs = pywt.cwt(
                signal, widths, wave, sampling_period=sampling_period
```

```
            )
            cwtmatr = np.abs(cwtmatr)
            list_cwt.append(cwtmatr)

        targets = signals[-1]  # Assuming the last row are the targets
        array_cwt = np.stack(list_cwt, axis=0)
        insert_cwt_data(conn, array_cwt, targets)  # Ensure this function is defined elsewhere in your code.
        del array_cwt
```

Listing 11: Przetwarzanie danych EEG za pomocą transformacji falkowej (CWT)

5.3. Wizualizacja danych i monitorowanie uczenia

W procesie uczenia modeli głębokich sieci neuronowych, kluczowe jest monitorowanie i wizualizacja różnych parametrów sieci w czasie rzeczywistym. To pozwala nie tylko na bieżące śledzenie postępów, ale również na archiwizację wyników eksperymentów, co jest niezbędne do późniejszej analizy i porównań. Wymagania dotyczące systemu monitorowania procesu uczenia obejmują:

- Możliwość podglądu parametrów sieci w czasie rzeczywistym.
- Archiwizacja danych o uczeniu się modeli, w tym informacji o hiperparametrach.
- Dostępność danych monitorowania online, umożliwiająca dostęp z dowolnego komputera.

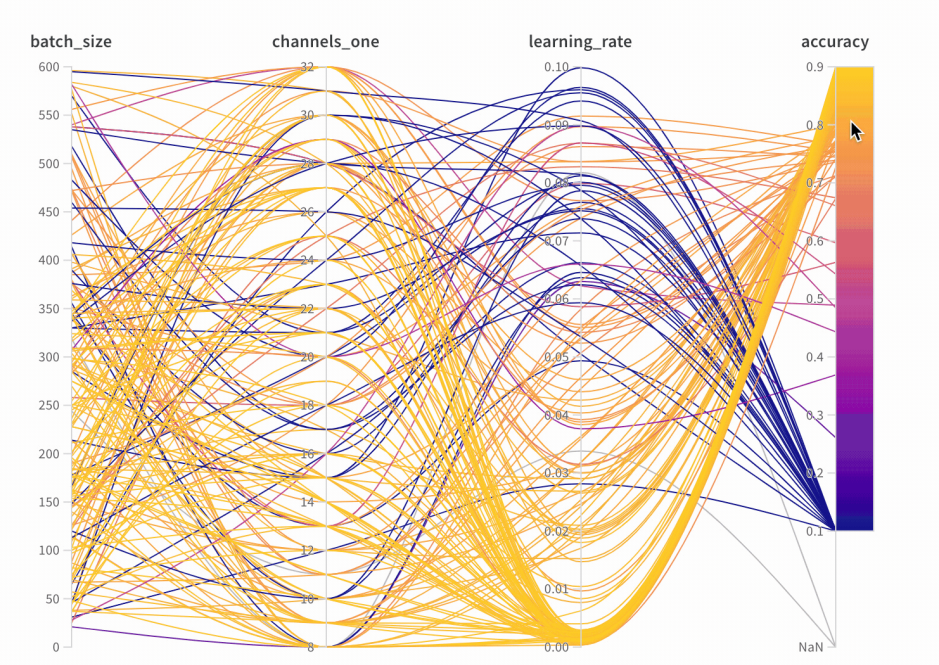
5.3.1. Logery w PyTorch Lightning

PyTorch Lightning oferuje mechanizm logowania, który ułatwia zapisywanie i monitorowanie danych w trakcie uczenia modelu. Zintegrowany z platformami takimi jak TensorBoard, MLFlow czy Weights & Biases, logery w PyTorch Lightning umożliwiają automatyczne śledzenie i zapisywanie nie tylko metryk, ale także hiperparametrów i wyników walidacji.

5.3.2. Platforma Weights & Biases

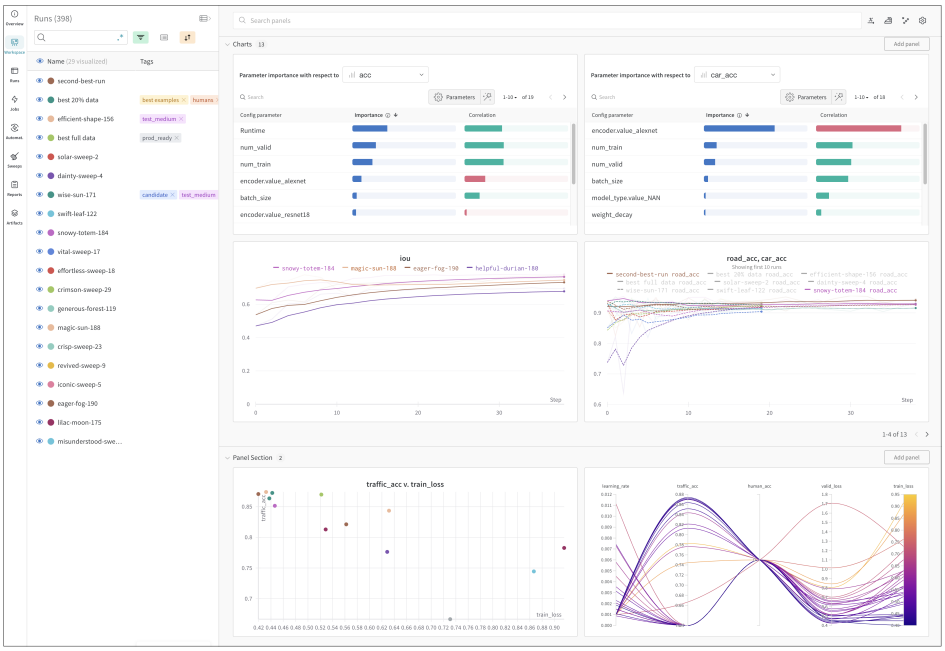
Weights & Biases (WandB) to platforma, która spełnia powyższe wymagania, oferując zaawansowane narzędzia do wizualizacji danych uczenia. Jedną z kluczowych funkcji, którą oferuje WandB, jest graf *Parallel Coordinates*, który pozwala na wizualizację wielowymiarowych danych. Użytkownik może łatwo porównywać wyniki

różnych uruchomień eksperymentów, analizując, jak zmiana hiperparametrów wpływa na wyniki modelu.



Rysunek 5.3: Przykład grafu Parallel Coordinates na platformie Weights & Biases, pozwalający na analizę wpływu hiperparametrów na wyniki modelu.

WandB umożliwia nie tylko wizualizację, ale także kompleksowe zarządzanie eksperymentami uczenia maszynowego, co czyni tę platformę nieocenionym narzędziem w procesie budowy i optymalizacji modeli.



Rysunek 5.4: Panel podglądu platformy Weights & Biases, prezentujący real-time monitoring parametrów sieci.

Załączniki

Repozytorium kodu na GitHubie: `https://github.com/Kleczyk/Decoding_of_EEG` (dostęp: 14.06.2024).

Literatura

[1] Schalk, G. et al.: EEG Motor Movement/Imagery Dataset. PhysioNet. Dostępne na: `https://physionet.org/content/eegmmidb/1.0.0/` (dostęp: 14.06.2024).