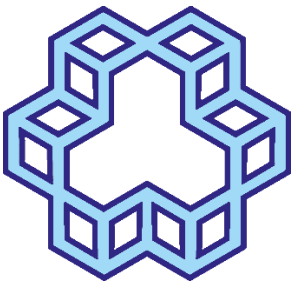


دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق

درس یادگیری ماشین

استاد دکتر علیاری

سیدمحمدرضا حسینی

شماره دانشجویی: ۴۰۲۰۴۵۸۴

گرایش: سیستم های الکترونیک دیجیتال

گزارش پروژه نهایی

[Google Colab]

[Paper]

[Github]

اطلاعات دقیق مقاله

عنوان مقاله:

TSception: A Deep Learning Framework for Emotion Detection Using EEG

نویسندگان:

Yi Ding, Neethu Robinson, Qiuhao Zeng, Duo Chen, Aung Aung Phyo Wai, Tih-Shih Lee, Cuntai Guan

سال انتشار:

2020

کنفرانس:

IEEE Transactions on Affective Computing

مقدمه

احساسات به عنوان یک جزء اساسی از زندگی روزمره، تأثیر عمیقی بر تعاملات اجتماعی، تصمیم‌گیری‌ها و وضعیت روانی افراد دارند. این احساسات می‌توانند از شادی و رضایت تا اضطراب و غم متغیر باشند و تأثیرات چشمگیری بر رفتار و سلامت روانی فرد داشته باشند. شناخت و درک دقیق احساسات می‌تواند به بهبود روابط اجتماعی، افزایش کارایی در محیط کار و ارتقای کیفیت زندگی کمک کند.

تشخیص صحیح و دقیق احساسات در حوزه‌های مختلف دارای اهمیت زیادی است:

۱. **تعاملات انسان و ماشین:**

- **رابط‌های مغز و کامپیوتر :(BCI)** تشخیص صحیح احساسات می‌تواند به بهبود تعاملات انسان و ماشین کمک کند. به عنوان مثال، در رابط‌های مغز و کامپیوتر، سیستم‌ها می‌توانند با درک حالات احساسی کاربر، تجربه کاربری را بهبود بخشند.
- **دستیارهای هوشمند :**تشخیص احساسات می‌تواند به دستیارهای هوشمند کمک کند تا واکنش‌های مناسبی به حالات احساسی کاربران نشان دهند، مثلاً تشخیص ناراحتی و ارائه پیشنهادهای آرامش‌بخش.

۲. **درمان اختلالات روانی:**

- **اضطراب و افسردگی :**تشخیص دقیق احساسات می‌تواند به درمانگران کمک کند تا روش‌های درمانی موثرتری برای اختلالات روانی مانند اضطراب و افسردگی ارائه دهند.
- **اوتیسم :**کودکان مبتلا به اوتیسم اغلب در تشخیص و ابراز احساسات مشکل دارند. تشخیص صحیح احساسات می‌تواند به بهبود برنامه‌های درمانی و آموزشی برای این کودکان کمک کند.

تکنولوژی EEG (الکتروانسفالوگرافی)

الکتروانسفالوگرافی (EEG) یکی از تکنیک‌های پرکاربرد برای ثبت فعالیت‌های الکتریکی مغز است. این تکنولوژی به دلیل مزایای زیادی که دارد، به عنوان یکی از روش‌های اصلی در تشخیص و مطالعه احساسات استفاده می‌شود.

تعریف و اصولEEG

EEG فرآیندی است که در آن فعالیت‌های الکتریکی مغز از سطح پوست سر اندازه‌گیری می‌شود. این فعالیت‌ها به صورت امواج الکتریکی ثبت می‌شوند که منعکس‌کننده فعالیت‌های نورونی در مغز هستند. سیگنال‌های EEG معمولاً با استفاده از الکترودهای قرار گرفته بر روی پوست سر جمع‌آوری می‌شوند.

امواجEEG

سیگنال‌های EEG شامل چندین نوع موج با فرکانس‌های مختلف هستند که هر کدام مرتبط با وضعیت‌های مختلف ذهنی و احساسی می‌باشند:

- **موج دلتا :(Delta)** با فرکانس ۰.۵ تا ۴ هرتز، معمولاً در خواب عمیق مشاهده می‌شود.
- **موج تتا :(Theta)** با فرکانس ۴ تا ۸ هرتز، مرتبط با خواب سبک و حالت‌های آرامش و مدیتیشن است.
- **موج آلفا :(Alpha)** با فرکانس ۸ تا ۱۲ هرتز، معمولاً در حالت‌های آرامش و بیداری آرام دیده می‌شود.
- **موج بتا :(Beta)** با فرکانس ۱۲ تا ۳۰ هرتز، مرتبط با فعالیت‌های ذهنی شدید، استرس، و تمرکز است.
- **موج گاما :(Gamma)** با فرکانس بیش از ۳۰ هرتز، مرتبط با پردازش اطلاعات و فعالیت‌های شناختی پیچیده است.

مزایای EEG نسبت به روش‌های دیگر

- **غیرتهاجمی بودن** EEG **:**یک روش غیرتهاجمی است که به راحتی و بدون درد می‌توان از آن برای ثبت فعالیت‌های مغزی استفاده کرد.
- **دقت زمانی بالا** EEG **:**دارای دقت زمانی بسیار بالاست که امکان ثبت فعالیت‌های مغزی در کسری از ثانیه را فراهم می‌کند.
- **قابل حمل بودن :**دستگاه‌های EEG مدرن کوچک و قابل حمل هستند و می‌توان از آنها در محیط‌های مختلف استفاده کرد.

کاربردهای EEG در تشخیص احساسات

- **تشخیص حالات احساسی :**با تحلیل سیگنال‌های EEG می‌توان حالات احساسی مختلف مانند خوشحالی، غم، استرس و آرامش را تشخیص داد.
- **پژوهش‌های روانشناختی** EEG **:**در پژوهش‌های مختلف برای مطالعه تأثیرات مختلف احساسی بر فعالیت‌های مغزی استفاده می‌شود.
- **درمان و توانبخشی** EEG **:**می‌تواند به عنوان یک ابزار کمکی در تشخیص و مانیتورینگ روند درمانی در اختلالات روانی مورد استفاده قرار گیرد.

در نتیجه، تشخیص صحیح و دقیق احساسات با استفاده از تکنولوژی EEG می‌تواند نقش بسیار مهمی در بهبود تعاملات انسان و ماشین، درمان اختلالات روانی، و پژوهش‌های روانشناختی ایفا کند.

در این مقاله، از تکنولوژی EEG برای تشخیص حالات احساسی استفاده شده است.

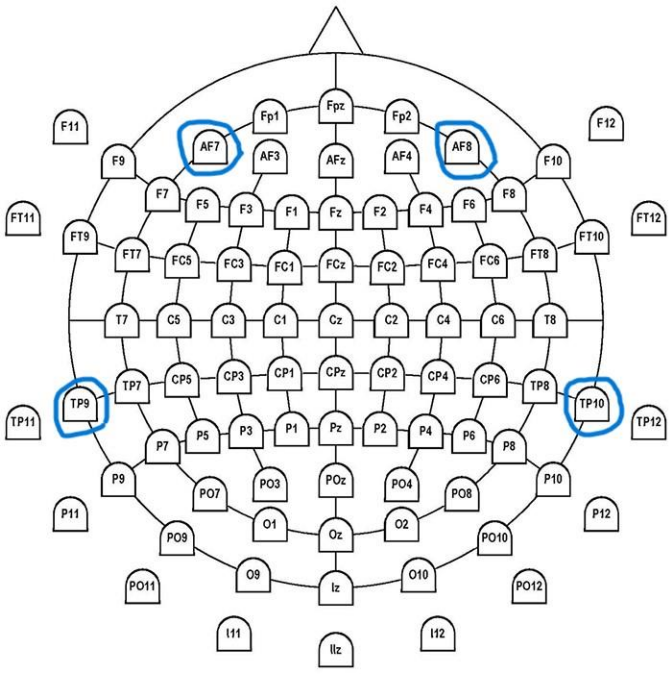
جمع‌آوری و آماده سازی دیتاست

جمع‌آوری داده‌ها

داده‌های EEG از ۱۸ نفر شامل ۹ مرد و ۹ زن با محدوده سنی بین ۲۳ تا ۴۹ سال جمع‌آوری شده است. این داده‌ها با استفاده از هدبند MUSE EEG که دارای چهار کانال (TP9, AF7, AF8, TP10) است و با نرخ نمونه‌برداری ۲۵۶ هرتز ثبت شده‌اند. هر کانال در نقاط مختلفی از سر قرار گرفته است تا فعالیت‌های مغزی مناطق مختلف را اندازه‌گیری کند:

- **TP9** ناحیه گیجگاهی چپ
- **AF7** ناحیه پیشانی چپ
- **AF8** ناحیه پیشانی راست
- **TP10** ناحیه گیجگاهی راست

این الکترودها بر اساس سیستم استاندارد بین‌المللی ۱۰-۲۰ برای قرارگیری الکترودها انتخاب شده‌اند تا دقت و جامعیت ثبت سیگنال‌ها را تضمین کنند.



تحریکات احساسی

تحریکات احساسی به دو نوع تقسیم شده‌اند: برانگیختگی کم و برانگیختگی بالا.

- **برانگیختگی کم :**در این حالت، شرکت‌کنندگان در یک صحنه آرام و دلنشین که شامل پرواز یک پرنده سفید بر فراز یک دریاچه یخزده است قرار می‌گیرند. موسیقی ملایمی نیز در پس‌زمینه پخش می‌شود تا حس آرامش و کاهش استرس ایجاد کند. هدف از این تحریک، ایجاد حالت احساسی آرام و بدون استرس است.
- **برانگیختگی بالا :**در این حالت، شرکت‌کنندگان در یک بازی واقعیت مجازی شرکت می‌کنند که در آن باید از برخورد با سنگ‌هایی که به سمت آنها پرتاب می‌شود اجتناب کنند. این بازی به گونه‌ای طراحی شده است که با افزایش سرعت و تعداد سنگ‌ها، سطح استرس و برانگیختگی فرد افزایش یابد. هدف از این تحریک، ایجاد حالت احساسی هیجان و افزایش استرس است.

پروتکل آزمایش

هر شرکت‌کننده سه جلسه آزمایشی داشته که هر جلسه شامل دو نوع تحریک احساسی (برانگیختگی کم و بالا) به مدت یک دقیقه بوده است. بین هر تحریک ۵ ثانیه زمان استراحت وجود دارد تا شرکت‌کنندگان فرصت پیدا کنند حالت احساسی خود را تنظیم کنند. مدت زمان هر جلسه ۶ دقیقه بوده و هر شرکت‌کننده در مجموع ۱۸ دقیقه در آزمایش شرکت کرده است. این زمان‌بندی به گونه‌ای طراحی شده است که داده‌های کافی برای تحلیل‌های بعدی جمع‌آوری شود و در عین حال شرکت‌کنندگان از نظر احساسی خسته نشوند.

در شکل زیر که از کد گرفته شده، شکل دیتا را برای هر جلسه مشاهده می فرمایید. (۴ کانال شامل ۱۵۳۶۰ دیتای یک دقیقه ای (۲۵۶*۶۰) برای ۶ دقیقه)

```
The shape of data is:(6, 4, 15360)
```

طبیعتا شکل لیبل هم به صورت زیر است.

```
The shape of label is:(6, 15360)
```

متاسفانه صرفا دیتای دو نفر به صورت عمومی قرار داده شده است. با اتصال این ۲ دیتا:

```
The data shape is:(2, 6, 1, 4, 15360)
```

پیش‌پردازش داده‌ها

سیگنال‌های EEG با استفاده از فیلتر باند-گذر در باند فرکانسی ۰.۳ هرتز تا ۴۵ هرتز پیش‌پردازش شده‌اند تا نویزهای فرکانس پایین و بالا حذف شوند. این فیلترگذاری به کاهش نویز و بهبود کیفیت سیگنال‌ها کمک می‌کند. همچنین، برای حذف نویزهای ناشی از حرکات چشم، از روش‌های استاندارد حذف EOG (الکتروآکولوگرافی) استفاده شده است. نرم‌افزارMNE ، یک ابزار قدرتمند برای پردازش داده‌های EEG وMEG ، برای این منظور استفاده شده است.

تقسیم‌بندی داده‌ها

داده‌های EEG به بازه‌های زمانی ۴ ثانیه‌ای با پنجره لغزان و گام ۱۰۰ میلی‌ثانیه تقسیم شده‌اند. این سگمنت‌بندی به تحلیل دقیق‌تر و ایجاد تعداد بیشتری از نمونه‌ها کمک می‌کند. هر جلسه به ۵۷۴ سگمنت تقسیم شده است که هر سگمنت شامل ۱۰۲۴ نمونه (۲۵۶ هرتز * ۴ ثانیه) است. این تقسیم‌بندی به گونه‌ای است که هر سگمنت می‌تواند به طور مستقل برای تحلیل‌های بعدی مورد استفاده قرار گیرد.

پس از سگمنت‌بندی، داده‌ها به شکل (۲, ۶, ۵۷۴, ۱, ۴, ۱۰۲۴) درآمده‌اند که ۲ فرد، ۶ دقیقه، ۵۷۴ سگمنت، ۴ کانال EEG و ۱۰۲۴ نمونه در هر سگمنت را نشان می‌دهد. برچسب‌ها نیز به شکل (۲, ۶, ۵۷۴) بوده که نشان‌دهنده ۲ فرد، ۶ دقیقه و ۵۷۴ برچسب برای هر سگمنت است.

```
The shape of data is:(6, 4, 15360)
The shape of label is:(6, 15360)
The shape of data is:(6, 4, 15360)
The shape of label is:(6, 15360)
**************Data loaded successfully!**************
The data shape is:(2, 6, 1, 4, 15360)
The data and label are splited: Data shape:(2, 6, 574, 1, 4, 1024) Label:(2, 6, 574)
Data and Label saved successfully! at: /content/data_split.hdf
```

از ابتدای کد تا پایان بخش Pre-processing on the EEG data شامل دانلود دیتاست و آماده سازی آن تا این حالت است.

پیاده سازی الگوریتم ها

در مقاله"TSception: A Deep Learning Framework for Emotion Detection Using EEG" ، چندین الگوریتم یادگیری ماشینی مختلف برای تشخیص احساسات از سیگنال‌های EEG مورد استفاده قرار گرفته و مقایسه شده‌اند. در کد پیاده سازی شده چند الگوریتم پیاده سازی شده است که به بیان آن ها می‌پردازیم.

۱. Sception

مدل Sception یک معماری شبکه عصبی عمیق است که برای پردازش و تحلیل سیگنال‌های EEG طراحی شده و بر اساس معماری Inception توسعه یافته است. هدف اصلی این مدل استخراج ویژگی‌های مختلف سیگنال‌های EEG و طبقه‌بندی دقیق آن‌ها است.

معماری Inception

Inceptionشامل بلوک‌هایی با چندین فیلتر با سایزهای مختلف است که به صورت موازی عمل می‌کنند. این ساختار به شبکه اجازه می‌دهد تا ویژگی‌های متنوعی از داده‌ها را در مقیاس‌های مختلف استخراج کند.

معماری Sception

Sceptionبا استفاده از اصولInception ، فیلترهای مختلفی مانند 1x1، 3x3، 5x5، و فیلترهای Max Pooling و Average Poolingرا به کار می‌گیرد. این فیلترها به صورت موازی بر روی سیگنال‌های EEG اعمال می‌شوند و خروجی آن‌ها ترکیب می‌شود تا ویژگی‌های مختلف سیگنال‌ها استخراج شود.

کاربردهای Sception

۱. **تشخیص بیماری‌های عصبی** :مانند صرع، آلزایمر و پارکینسون.

۲. **تحلیل حالت‌های شناختی** :مانند توجه، تمرکز و خواب.

۳. **کنترل سیستم‌های واسط مغز و کامپیوتر (BCI)**: تحلیل سیگنال‌های EEG برای کنترل این سیستم‌ها.

مزایا

- **دقت بالا** :در استخراج و طبقه‌بندی ویژگی‌های سیگنال‌های.EEG
- **انعطاف‌پذیری** :قابل استفاده در کاربردهای مختلف تحلیل سیگنال‌های.EEG
- **کاهش پیچیدگی محاسباتی** :در مقایسه با مدل‌های سنتی.

معایب

- **نیاز به داده‌های زیاد** :برای آموزش مدل.

- **پیچیدگی معماری** :نیاز به تنظیمات دقیق دارد.

مدل Sception با ترکیب فیلترهای مختلف و معماری موازی، قادر است ویژگی‌های متنوع سیگنال‌های EEG را با دقت بالا استخراج و تحلیل کند.

```
----Model saved!----
Epoch [2/200], Loss: 0.3616, Acc: 0.8648
Evaluation Loss:0.6040, Acc: 0.7279
----Model saved!----
Epoch [3/200], Loss: 0.2944, Acc: 0.8835
Evaluation Loss:0.3814, Acc: 0.8571
----Model saved!----
Epoch [4/200], Loss: 0.2442, Acc: 0.9105
Evaluation Loss:0.3725, Acc: 0.8532
Epoch [5/200], Loss: 0.2066, Acc: 0.9261
Evaluation Loss:0.3627, Acc: 0.8669
----Model saved!----
Epoch [6/200], Loss: 0.1595, Acc: 0.9460
Evaluation Loss:0.3840, Acc: 0.8655
Epoch [7/200], Loss: 0.1282, Acc: 0.9605
Evaluation Loss:0.4003, Acc: 0.8682
----Model saved!----
Epoch [8/200], Loss: 0.1034, Acc: 0.9714
Evaluation Loss:0.3919, Acc: 0.8655
Epoch [9/200], Loss: 0.0850, Acc: 0.9792
Evaluation Loss:0.4100, Acc: 0.8564
Epoch [10/200], Loss: 0.0647, Acc: 0.9881
Evaluation Loss:0.4260, Acc: 0.8597
Epoch [11/200], Loss: 0.0556, Acc: 0.9870
Evaluation Loss:0.4475, Acc: 0.8519
Epoch [12/200], Loss: 0.0455, Acc: 0.9906
Evaluation Loss:0.4440, Acc: 0.8564
----Early stopping----
Test Loss:0.6013, Acc: 0.7792
Subject:1
mACC: 0.82
std: 0.03
Mean ACC:0.844496714456392 Std:0.027199074074074125
```

همانطور که مشاهده می شود، دقت این مدل برای دو نمونه محاسبه شده و میانگین دقت آن ۸۴ درصد است.

۲. Tception

مدل Tception نیز یک معماری شبکه عصبی عمیق است که برای تحلیل و طبقه‌بندی سیگنال‌های EEG طراحی شده است. این مدل بر اساس ترکیب شبکه‌های عصبی زمانی (Temporal) و معماری Inception توسعه یافته است تا ویژگی‌های زمانی سیگنال‌های EEG را به طور کارآمد استخراج کند.

معماری Tception

Tception با هدف بهبود استخراج ویژگی‌های زمانی از سیگنال‌هایEEG ، از بلوک‌های مخصوص به خود استفاده می‌کند. این بلوک‌ها شامل فیلترهای مختلفی هستند که به صورت موازی عمل می‌کنند و قادر به استخراج ویژگی‌های زمانی در مقیاس‌های مختلف هستند.

بلوک‌های Tception

هر بلوک Tception شامل چندین فیلتر با طول‌های زمانی مختلف است که به صورت موازی اعمال می‌شوند. این فیلترها می‌توانند طول‌های زمانی کوتاه و بلند را پوشش دهند، به این ترتیب شبکه می‌تواند ویژگی‌های زمانی متنوعی را از سیگنال‌ها استخراج کند.

فیلترهای مختلف

فیلترهای Tception شامل فیلترهای زمانی با طول‌های مختلف مانند 1x3، 1x5، و 1x7 هستند. این فیلترها به صورت موازی بر روی سیگنال‌های EEG اعمال می‌شوند و خروجی آن‌ها ترکیب می‌شود تا ویژگی‌های زمانی مختلف استخراج شود.

ترکیب خروجی فیلترها

خروجی فیلترهای مختلف پس از اعمال بر روی سیگنال‌های EEG با هم ترکیب می‌شوند. این ترکیب می‌تواند شامل عملیات‌هایی مانند Concatenation باشد. هدف از این ترکیب، استخراج ویژگی‌های زمانی مختلف و بهبود دقت طبقه‌بندی است.

کاربردهای Tception

۱. **تشخیص بیماری‌های عصبی** :مانند صرع، آلزایمر و پارکینسون.

۲. **تحلیل حالت‌های شناختی** :مانند توجه، تمرکز و خواب.

۳. **کنترل سیستم‌های واسط مغز و کامپیوتر (BCI)**: تحلیل سیگنال‌های EEG برای کنترل این سیستم‌ها.

مزایا و معایب Tception

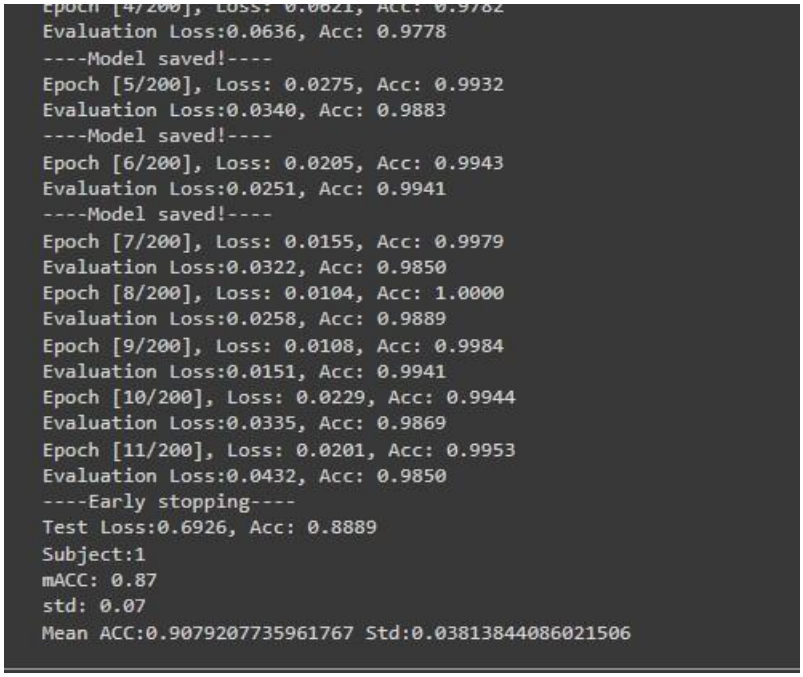
مزایا

- **دقت بالا** :در استخراج و طبقه‌بندی ویژگی‌های زمانی سیگنال‌های EEG.
- **انعطاف‌پذیری** :قابل استفاده در کاربردهای مختلف تحلیل سیگنال‌های EEG.
- **کاهش پیچیدگی محاسباتی** :به دلیل استفاده از فیلترهای زمانی موازی.

معایب

- **نیاز به داده‌های زیاد** :برای آموزش مدل.
- **پیچیدگی معماری** :نیاز به تنظیمات دقیق دارد.

مدل Tception با ترکیب فیلترهای زمانی مختلف و معماری موازی، قادر است ویژگی‌های زمانی سیگنال‌های EEG را با دقت بالا استخراج و تحلیل کند. این مدل به خصوص در کاربردهایی که نیاز به تحلیل دقیق ویژگی‌های زمانی سیگنال دارند، مانند تشخیص بیماری‌های عصبی و کنترل سیستم‌های BCI ، بسیار موثر است.



همانطور که مشاهده می شود، دقت این مدل برای دو نمونه محاسبه شده و میانگین دقت آن ۹۱ درصد است.

۳. TSception

مدل TSception یک معماری شبکه عصبی عمیق است که برای تحلیل و طبقه‌بندی سیگنال‌های EEG طراحی شده است. این مدل با ترکیب دو مدل Tception وSception ، ویژگی‌های زمانی و فضایی سیگنال‌های EEG را به طور همزمان استخراج می‌کند.

معماری TSception

TSceptionبه منظور بهبود دقت در تحلیل سیگنال‌هایEEG ، از بلوک‌های مخصوص به خود استفاده می‌کند که قادر به استخراج ویژگی‌های زمانی و فضایی هستند.

بلوک‌های TSception

هر بلوک TSception شامل فیلترهای زمانی و فضایی است که به صورت موازی اعمال می‌شوند. این فیلترها می‌توانند طول‌های زمانی و اندازه‌های فضایی مختلف را پوشش دهند، به این ترتیب شبکه می‌تواند ویژگی‌های متنوعی را از سیگنال‌ها استخراج کند.

2.فیلترهای مختلف

فیلترهای TSception شامل:

- **فیلترهای زمانی** :با طول‌های مختلف مانند 1x5، 1x3، و 1x7
- **فیلترهای فضایی** :با اندازه‌های مختلف مانند 5x1، 3x1، و 7x1

این فیلترها به صورت موازی بر روی سیگنال‌های EEG اعمال می‌شوند و خروجی آن‌ها ترکیب می‌شود تا ویژگی‌های زمانی و فضایی مختلف استخراج شود.

ترکیب خروجی فیلترها

خروجی فیلترهای مختلف پس از اعمال بر روی سیگنال‌های EEG با هم ترکیب می‌شوند. این ترکیب می‌تواند شامل عملیات‌هایی مانند Concatenation باشد. هدف از این ترکیب، استخراج ویژگی‌های زمانی و فضایی مختلف و بهبود دقت طبقه‌بندی است.

کاربردهای TSception

۱. **تشخیص بیماری‌های عصبی** :مانند صرع، آلزایمر و پارکینسون.

۲. **تحلیل حالت‌های شناختی** :مانند توجه، تمرکز و خواب.

۳. **کنترل سیستم‌های واسط مغز و کامپیوتر (BCI)**: تحلیل سیگنال‌های EEG برای کنترل این سیستم‌ها.

مزایا و معایب TSception

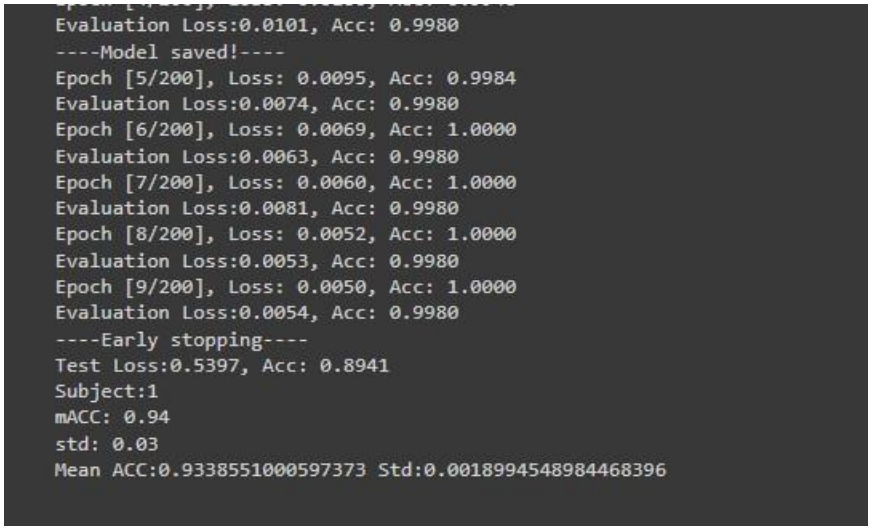
مزایا

- **دقت بالا** :در استخراج و طبقه‌بندی ویژگی‌های زمانی و فضایی سیگنال‌های EEG.
- **انعطاف‌پذیری** :قابل استفاده در کاربردهای مختلف تحلیل سیگنال‌های EEG.
- **کاهش پیچیدگی محاسباتی** :به دلیل استفاده از فیلترهای زمانی و فضایی موازی.

معایب

- **نیاز به داده‌های زیاد** :برای آموزش مدل.
- **پیچیدگی معماری** :نیاز به تنظیمات دقیق دارد.

مدل TSception با ترکیب فیلترهای زمانی و فضایی مختلف و معماری موازی، قادر است ویژگی‌های زمانی و فضایی سیگنال‌های EEG را با دقت بالا استخراج و تحلیل کند. این مدل به خصوص در کاربردهایی که نیاز به تحلیل دقیق ویژگی‌های زمانی و فضایی سیگنال دارند، مانند تشخیص بیماری‌های عصبی و کنترل سیستم‌های BCI ، بسیار موثر است.



همانطور که مشاهده می شود، دقت این مدل برای دو نمونه محاسبه شده و میانگین دقت آن ۹۳ درصد است.

در کد این ۳ مدل یادگیری عمیق در بخش Deep Learning قرار گرفته اند.

در مقاله، تکنیک "one session out" که به عنوان "Leave-One-Session-Out" (LOSO) نیز شناخته می‌شود، برای ارزیابی عملکرد مدل‌های یادگیری ماشین استفاده شده است. این تکنیک به ویژه در مشکلات دسته‌بندی زمانی و پردازش سیگنال‌هایی مانند EEG که ممکن است شامل داده‌های از جلسات مختلف باشد، کاربرد دارد. در ادامه به توضیح کامل این تکنیک پرداخته می‌شود:

تکنیک(Leave-One-Session-Out (LOSO

تعریف کلی

تکنیک Leave-One-Session-Out (LOSO) یک روش ارزیابی کراس‌ولیدیشن (cross-validation) است که به‌طور خاص برای داده‌هایی با ساختار زمانی یا توزیع‌های مختلف از جلسات آزمایشی طراحی شده است. در این تکنیک، مدل به‌طور مکرر روی زیرمجموعه‌ای از داده‌ها آموزش داده می‌شود و سپس روی داده‌های یک جلسه خاص که از فرایند آموزش کنار گذاشته شده است، ارزیابی می‌شود. این فرایند تا زمانی که تمام جلسات به‌عنوان جلسه تست انتخاب شوند، تکرار می‌شود.

نحوه عملکرد

۱. **تقسیم داده‌ها به جلسات:**

- داده‌ها به چندین جلسه (session) تقسیم می‌شوند. هر جلسه شامل داده‌های مربوط به یک دوره زمانی خاص از آزمایشات است.

۲. **آموزش و ارزیابی مدل:**

- در هر تکرار از فرایندLOSO ، یک جلسه به‌عنوان مجموعه تست انتخاب می‌شود و بقیه جلسات برای آموزش مدل استفاده می‌شوند.
- مدل بر روی داده‌های مجموعه آموزش آموزش داده می‌شود و سپس روی داده‌های جلسه تست ارزیابی می‌شود.

۳. **محاسبه معیارهای ارزیابی:**

- عملکرد مدل با استفاده از داده‌های جلسه تست مورد ارزیابی قرار می‌گیرد.
- این فرآیند برای هر جلسه تکرار می‌شود، به این ترتیب که هر جلسه به نوبت به‌عنوان داده‌های تست انتخاب می‌شود و مدل بر روی داده‌های باقی‌مانده آموزش داده می‌شود.

۴. **میانگین نتایج:**

- نتایج به‌دست‌آمده از هر تکرار جمع‌آوری شده و میانگین آنها برای به‌دست‌آوردن عملکرد کلی مدل محاسبه می‌شود.

مزایا

- **ارزیابی دقیق :**تکنیک LOSO به‌ویژه در شرایطی که داده‌های موجود به‌طور طبیعی به دسته‌های مختلف (مانند جلسات آزمایشی مختلف) تقسیم شده‌اند، دقت بیشتری را در ارزیابی مدل فراهم می‌کند. این تکنیک به مدل اجازه می‌دهد تا با داده‌هایی که هرگز در طول آموزش مشاهده نکرده است، آزمایش شود.
- **جلوگیری از :Overfitting** با استفاده از تکنیکLOSO ، می‌توان خطر overfitting (تطابق بیش از حد مدل با داده‌های آموزشی) را کاهش داد، زیرا مدل بر اساس داده‌های کاملاً جدید (جلسه تست) ارزیابی می‌شود.

معایب

- **زمان‌بر :**این تکنیک می‌تواند زمان‌بر باشد زیرا مدل باید برای هر جلسه به‌طور جداگانه آموزش و ارزیابی شود.
- **پایداری :**اگر تعداد جلسات کم باشد، نتایج ممکن است به پایداری کم‌تری دچار شوند زیرا تعداد کافی از نمونه‌های تست ممکن است برای ارزیابی دقیق مدل در دسترس نباشد.

کاربرد در مقاله

در مقاله"TSception: A Deep Learning Framework for Emotion Detection Using EEG" ، تکنیک Leave-One-Session-Out (LOSO)برای ارزیابی مدل TSception به کار رفته است. این روش به نویسندگان این امکان را می‌دهد تا مدل خود را با استفاده از داده‌های مختلف از جلسات مختلف آزمایش کنند و به طور دقیق‌تری عملکرد مدل را در شرایط واقعی ارزیابی نمایند. این روش همچنین به آنها کمک می‌کند تا اطمینان حاصل کنند که مدل به‌طور مؤثر بر روی

داده‌هایی که هرگز در طول فرایند آموزش مشاهده نکرده است، عمل می‌کند و به این ترتیب قابلیت تعمیم مدل به داده‌های جدید را بررسی کنند.

در نتیجه، تکنیک LOSO به نویسندگان این امکان را می‌دهد که ارزیابی جامع‌تری از عملکرد مدل داشته باشند و اطمینان حاصل کنند که مدل ارائه شده برای تشخیص احساسات از سیگنال‌های EEG به‌طور موثری در شرایط واقعی کار می‌کند.

۴. SVM

ماشین بردار پشتیبان (Support Vector Machine) یکی از الگوریتم‌های محبوب یادگیری ماشین است که برای طبقه‌بندی و رگرسیون مورد استفاده قرار می‌گیرد. این الگوریتم به خصوص در کاربردهایی که نیاز به دقت بالا دارند و داده‌ها دارای ویژگی‌های پیچیده و غیرخطی هستند، بسیار موثر است.

اصل کارSVM

SVM سعی می‌کند یک مرز تصمیم‌گیری (Decision Boundary) یا ابرصفحه (Hyperplane) را پیدا کند که داده‌ها را به بهترین شکل ممکن به دو دسته مجزا تقسیم کند. هدف اصلی SVM پیدا کردن ابرصفحه‌ای است که بیشترین فاصله (Margin) را از نزدیک‌ترین نمونه‌های هر کلاس داشته باشد. این نمونه‌های نزدیک به ابرصفحه، بردارهای پشتیبان (Support Vectors) نامیده می‌شوند.

انواعSVM

۱. **SVMخطی** :زمانی که داده‌ها به صورت خطی قابل تفکیک هستند، از یک ابرصفحه خطی برای جدا کردن کلاس‌ها استفاده می‌شود.

۲. **SVMغیرخطی** :زمانی که داده‌ها به صورت خطی قابل تفکیک نیستند، از روش‌های غیرخطی مانند کرنل‌ها (Kernels)استفاده می‌شود تا داده‌ها به فضای با ابعاد بالاتر منتقل شوند و در آنجا قابل تفکیک شوند.

کرنل‌ها(Kernels)

کرنل‌ها تابع‌هایی هستند که داده‌ها را به فضای با ابعاد بالاتر می‌برند تا در آن فضا بتوانند با یک ابرصفحه خطی جدا شوند. برخی از کرنل‌های معروف عبارتند از:

- **کرنل چندجمله‌ای(Polynomial Kernel)** : داده‌ها را به توان‌های مختلف می‌برد.
- **کرنل گاوسی (Gaussian Kernel) یاRBF** : داده‌ها را به فضای بی‌نهایت ابعاد می‌برد.
- **کرنل سیگموئید(Sigmoid Kernel)** : شبیه به توابع سیگموئید در شبکه‌های عصبی عمل می‌کند.

کاربردهایSVM

۱. **تشخیص الگو و طبقه‌بندی** :مانند تشخیص چهره، تشخیص دست‌خط، و طبقه‌بندی متن.

۲. **تحلیل داده‌های زیستی** :مانند تشخیص بیماری‌ها از روی داده‌های ژنتیکی و پروتئینی.

۳. **پیش‌بینی مالی** :تحلیل و پیش‌بینی بازارهای مالی و سهام.

۴. **کنترل سیستم‌های صنعتی** :تحلیل و کنترل فرآیندهای صنعتی و تولیدی.

مزایا و معایبSVM

مزایا

- **دقت بالا** :به خصوص در مسائل طبقه‌بندی.
- **قابلیت تعمیم خوب** :به دلیل حداکثر کردن فاصله مرز تصمیم‌گیری از نمونه‌های نزدیک.
- **کارایی در داده‌های با ابعاد بالا** :با استفاده از کرنل‌ها می‌تواند با داده‌های پیچیده و غیرخطی کار کند.

معایب

- **پیچیدگی محاسباتی** :به خصوص در مسائل بزرگ و با تعداد نمونه‌های زیاد.
- **نیاز به تنظیم دقیق پارامترها** :مانند نوع کرنل و پارامترهای آن.
- **حساسیت به نویز و داده‌های پرت** :نمونه‌های نویزی و پرت می‌توانند بر عملکرد مدل تاثیر منفی بگذارند.

SVMیک ابزار قدرتمند برای طبقه‌بندی و رگرسیون است که با استفاده از کرنل‌ها می‌تواند مسائل پیچیده و غیرخطی را نیز حل کند. این الگوریتم به خصوص در کاربردهایی که نیاز به دقت بالا و تحلیل داده‌های پیچیده دارند، بسیار موثر است.

در کد برای پیاده سازی SVM ابتدا ۱۰ ویژگی زیر از ۴ کانال EEG استخراج شد:
میانگین :**(Mean)** میانگین مقدار سیگنال.

واریانس :**(Variance)** پراکندگی مقدار سیگنال.

حداکثر مقدار :**(Max Value)** بیشترین مقدار سیگنال.

حداقل مقدار :**(Min Value)** کمترین مقدار سیگنال.

دامنه :**(Range)** تفاوت بین حداکثر و حداقل مقدار سیگنال.

انحراف معیار :**(Standard Deviation)** میزان پراکندگی مقدار سیگنال نسبت به میانگین.

توان میانگین :**(Mean Power)** میانگین توان سیگنال.

انرژی :**(Energy)** مجموع توان سیگنال.

میانگین قدر مطلق :(Mean Absolute Value) میانگین مقادیر مطلق سیگنال.

بزرگی بیشینه در حوزه فرکانس :(Dominant Frequency) فرکانسی که دارای بیشترین مقدار در حوزه فرکانس است.

```python
def extract_features(signal):
    features = []
    for i in range(signal.shape[0]):
        sig = signal[i]
        mean = np.mean(sig)
        var = np.var(sig)
        max_val = np.max(sig)
        min_val = np.min(sig)
        range_val = max_val - min_val
        std_dev = np.std(sig)
        mean_power = np.mean(sig ** 2)
        energy = np.sum(sig ** 2)
        mean_abs = np.mean(np.abs(sig))
        freq_domain = np.abs(fft(sig))
        dom_freq = np.argmax(freq_domain)
        features.extend([mean, var, max_val, min_val, range_val, std_dev, mean_power, energy, mean_abs, dom_freq])
    return features
```

سپس بعد از انجام PCA، یادگیری انجام گرفت. نتیجه به شکل زیر است.

```
Subject:0, Session:0, Train ACC:1.0000, Val ACC:1.0000, Test ACC:0.5000
Subject:0, Session:1, Train ACC:1.0000, Val ACC:0.0000, Test ACC:0.5000
Subject:0, Session:2, Train ACC:0.6667, Val ACC:0.0000, Test ACC:0.5000
Subject:0
mACC: 0.50
std: 0.00
Subject:1, Session:0, Train ACC:1.0000, Val ACC:0.0000, Test ACC:0.5000
Subject:1, Session:1, Train ACC:0.6667, Val ACC:0.0000, Test ACC:0.5000
Subject:1, Session:2, Train ACC:0.6667, Val ACC:0.0000, Test ACC:0.5000
Subject:1
mACC: 0.50
std: 0.00
Mean ACC:0.5 Std:0.0
```

همانطور که مشاهده می شود، دقت این مدل برای دو نمونه محاسبه شده و میانگین دقت آن ۵۰ درصد است.

با توجه به اعداد مشخص است که feature های انتخابی ما مناسب نبوده‌اند. به همین منظور از دیتای خام به عنوان ویژگی استفاده کردیم تا شاید دقت بهتری بدست آید.

در گام اول بدون PCA کار کردیم. نتیجه به شکل زیر است:

```
Subject:0, Session:0, Train ACC:0.9940, Val ACC:0.5674, Test ACC:0.4111
Subject:0, Session:1, Train ACC:0.9466, Val ACC:0.4978, Test ACC:0.4965
Subject:0, Session:2, Train ACC:0.9542, Val ACC:0.5630, Test ACC:0.4634
Subject:0
mACC: 0.46
std: 0.04
Subject:1, Session:0, Train ACC:0.9826, Val ACC:0.8152, Test ACC:0.8197
Subject:1, Session:1, Train ACC:0.9842, Val ACC:0.8283, Test ACC:0.8362
Subject:1, Session:2, Train ACC:0.9798, Val ACC:0.8413, Test ACC:0.7857
Subject:1
mACC: 0.81
std: 0.02
Mean ACC:0.6354529616724739 Std:0.17842624854819983
0.6354529616724739
```

همانطور که مشاهده می شود، دقت این مدل برای دو نمونه محاسبه شده و میانگین دقت آن ۶۳ درصد است.

حال همین کار را با PCA انجام می دهیم.

```
Subject:0, Session:0, Train ACC:0.9924, Val ACC:0.5565, Test ACC:0.3885
Subject:0, Session:1, Train ACC:0.9444, Val ACC:0.5065, Test ACC:0.4965
Subject:0, Session:2, Train ACC:0.9526, Val ACC:0.5630, Test ACC:0.4617
Subject:0
mACC: 0.45
std: 0.05
Subject:1, Session:0, Train ACC:0.9826, Val ACC:0.8109, Test ACC:0.8319
Subject:1, Session:1, Train ACC:0.9842, Val ACC:0.7913, Test ACC:0.8066
Subject:1, Session:2, Train ACC:0.9793, Val ACC:0.8196, Test ACC:0.7439
Subject:1
mACC: 0.79
std: 0.04
Mean ACC:0.6215156794425087 Std:0.17261904761904767
0.6215156794425087
```

همانطور که مشاهده می شود، دقت این مدل برای دو نمونه محاسبه شده و میانگین دقت آن ۶۲ درصد است. با PCA پیچیدگی محاسباتی و زمان محاسبات به حد قابل توجهی کم شد اما دقت تغییر چندانی نکرد. برای همین از PCA استفاده می کنیم.

در مرحله آخر SVM را با پارامترهای مختلف gamma (scale و auto)، C (۰.۱ و ۱ و ۱۰) و kernel (rbf و linear و poly) پیاده سازی می کنیم تا بهترین پارامترها را بیابیم.

نتیجه به شکل زیر است:

```
Model number 18 with parameters: (kernel=poly, C=10, gamma=auto) is starting...
Data loaded!
Data shape:[(2, 6, 574, 1, 4, 1024)], Label shape:[(2, 6, 574)]
Train:Leave_one_session_out
1)shape of data:(2, 6, 574, 1, 4, 1024)
2)shape of label:(2, 6, 574)
3)trials:6
4)sessions:3
5)datapoint:1024
6)channel:4
Subject:0, Session:0, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:0, Session:1, Train ACC:1.0000, Val ACC:0.5283, Test ACC:0.5000
Subject:0, Session:2, Train ACC:1.0000, Val ACC:0.5348, Test ACC:0.5035
Subject:0
mACC: 0.50
std: 0.00
Subject:1, Session:0, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:1, Session:1, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:1, Session:2, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:1
mACC: 0.50
std: 0.00
Mean ACC:0.5005807200929152 Std:0.0005807200929152101


Best ACC: 0.6319686411149825
Best Parameters: (kernel: rbf, C:0.1, gamma:scale)
```

همانطور که مشاهده می شود، بهترین پارامتر ها (kernel=rbf, C=0.1, gamma=scale) هستند که مدل در این حالت دارای دقت ۶۳ درصدی است.

خلاصه کد

ابتدا کتابخانه های مورد نظر را قرار می دهیم. سپس دیتاست (که قبلا در google drive قرار داده ایم) را در کد دانلود می کنیم.

Libraries and Dataset

```
import numpy as np
import math
import h5py
from pathlib import Path
import torch
from torch.utils.data import Dataset, TensorDataset
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import os
import datetime
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from numba import njit
from sklearn.decomposition import PCA
from scipy.fftpack import fft
```

```
[2] !pip install scipy
    !pip install --upgrade --no-cache-dir gdown
    !gdown 15anDcjyEVky89mIc_4Dskg7z2hQxVifN
    !gdown 1VL-87RmRuSOTFTFBO-CzXf404rOlnsH4
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.11
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.10/dis
```

سپس دیتا را load کرده و به بازه‌های زمانی ۴ ثانیه‌ای با پنجره لغزان و گام ۱۰۰ میلی‌ثانیه تقسیم می کنیم.

Pre-processing on the EEG data

```
class Processer:
    def __init__(self):
        self.data = None
        self.label = None
        self.data_processed = None
        self.label_processed = None
    def load_data(self, path, subject):
        path = Path(path)
        data_list = []
        label_list = []
        for i in range(subject):
            file_code = 'sub_'+ str(i)+'.hdf'
            file = path / file_code
            data_dictionary = h5py.File(file, 'r')
            data = data_dictionary['data']
            label = data_dictionary['label']
            data_list.append(data)
            label_list.append(label)
            print('The shape of data is:'+ str(data_list[-1].shape))
            print('The shape of label is:' + str(label_list[-1].shape))
        self.data = np.stack(data_list, axis = 0)
        self.label = np.stack(label_list, axis = 0)
        # data: subject x trial x channels x datapoint
        # label: subject x trial x datapoint
        print('**************Data loaded successfully!**************')

    def format_data(self):
        # data: subject x trial x channels x datapoint
        # label: subject x trial x datapoint
        data = self.data
        label = self.label

        # change the label representation 1.0 -> 0.0; 2.0 -> 1.0
        label[label == 1.0] = 0.0
        label[label == 2.0] = 1.0

        #Expand the frequency dimention
        self.data_processed = np.expand_dims(data, axis = 2)

        self.label_processed = label

        print("The data shape is:" + str(self.data_processed.shape))

    def split_data(self, segment_length = 1, overlap = 0, sampling_rate = 256, save = True):
        #data: subject x trial x 1 x channels x datapoint
```

```python
def split_data(self, segment_length = 1, overlap = 0, sampling_rate = 256, save = True):
    #data: subject x trial x 1 x channels x datapoint
    #label: subject x trial x datapoint
    #Parameters
    data = self.data_processed
    label = self.label_processed
    #Split the data given
    data_shape = data.shape
    label_shape = label.shape
    data_step = int(segment_length * sampling_rate * (1 - overlap))
    data_segment = sampling_rate * segment_length
    data_split = []
    label_split = []

    number_segment = int((label_shape[2]-data_segment)//(data_step)) + 1
    for i in range(number_segment):
        data_split.append(data[:,:,:,(i * data_step):(i * data_step + data_segment)])
        label_split.append(label[:,:,(i * data_step)])
    data_split_array = np.stack(data_split, axis = 2)
    label_split_array = np.stack(label_split, axis = 2)
    print("The data and label are splited: Data shape:" + str(data_split_array.shape) +" Label:" + str(label_split_array.shape))
    self.data_processed = data_split_array
    self.label_processed = label_split_array


    #TODO: Save the processed data here
    if save == True:
        if self.data_processed.all() != None:

          save_path = Path(os.getcwd())
          filename_data = save_path / Path('data_split.hdf')
          save_data = h5py.File(filename_data, 'w')
          save_data['data'] = self.data_processed
          save_data['label'] = self.label_processed
          save_data.close()
          print("Data and Label saved successfully! at: " + str(filename_data))
        else :
          print("data_splited is None")
```

```python
[23] current_path = os.getcwd()
Pro = Processer()
Pro.load_data(path=current_path,subject=2)
Pro.format_data()
Pro.split_data(segment_length = 4, overlap = 0.975, sampling_rate = 256, save = True)
```

```
The shape of data is:(6, 4, 15360)
The shape of label is:(6, 15360)
```

حال کلاس های مربوط به ۳ الگوریتم یادگیری عمیق TSception، Tception و Sception را پیاده سازی می کنیم.

Deep Learning

```
################################################ TSception ##################################################
class TSception(nn.Module):
    def conv_block(self, in_chan, out_chan, kernel, step, pool):
        return nn.Sequential(
            nn.Conv2d(in_channels=in_chan, out_channels=out_chan,
                      kernel_size=kernel, stride=step, padding=0),
            nn.LeakyReLU(),
            nn.AvgPool2d(kernel_size=(1, pool), stride=(1, pool)))

    def __init__(self, num_classes, input_size, sampling_rate, num_T, num_S, hidden, dropout_rate):
        # input_size: EEG channel x datapoint
        super(TSception, self).__init__()
        self.inception_window = [0.5, 0.25, 0.125]
        self.pool = 8
        # by setting the convolutional kernel being (1,lenght) and the strids being 1 we can use conv2d to
        # achieve the 1d convolution operation
        self.Tception1 = self.conv_block(1, num_T, (1, int(self.inception_window[0] * sampling_rate)), 1, self.pool)
        self.Tception2 = self.conv_block(1, num_T, (1, int(self.inception_window[1] * sampling_rate)), 1, self.pool)
        self.Tception3 = self.conv_block(1, num_T, (1, int(self.inception_window[2] * sampling_rate)), 1, self.pool)

        self.Sception1 = self.conv_block(num_T, num_S, (int(input_size[-2]), 1), 1, int(self.pool*0.25))
        self.Sception2 = self.conv_block(num_T, num_S, (int(input_size[-2] * 0.5), 1), (int(input_size[-2] * 0.5), 1),
                                         int(self.pool*0.25))
        self.BN_t = nn.BatchNorm2d(num_T)
        self.BN_s = nn.BatchNorm2d(num_S)

        size = self.get_size(input_size)
        self.fc = nn.Sequential(
            nn.Linear(size[1], hidden),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.Linear(hidden, num_classes)
        )

    def forward(self, x):
        y = self.Tception1(x)
        out = y
        y = self.Tception2(x)
        out = torch.cat((out, y), dim=-1)
        y = self.Tception3(x)
        out = torch.cat((out, y), dim=-1)
        out = self.BN_t(out)
        z = self.Sception1(out)
        out_ = z
        z = self.Sception2(out)
        out_ = torch.cat((out_, z), dim=2)
        out = self.BN_s(out_)
        out = out.view(out.size()[0], -1)
        out = self.fc(out)
        return out

    def get_size(self, input_size):
        # here we use an array with the shape being
        # (1(mini-batch),1(convolutional channel),EEG channel,time data point)
        # to simulate the input data and get the output size
        data = torch.ones((1, 1, input_size[-2], int(input_size[-1])))
        y = self.Tception1(data)
        out = y
        y = self.Tception2(data)
```

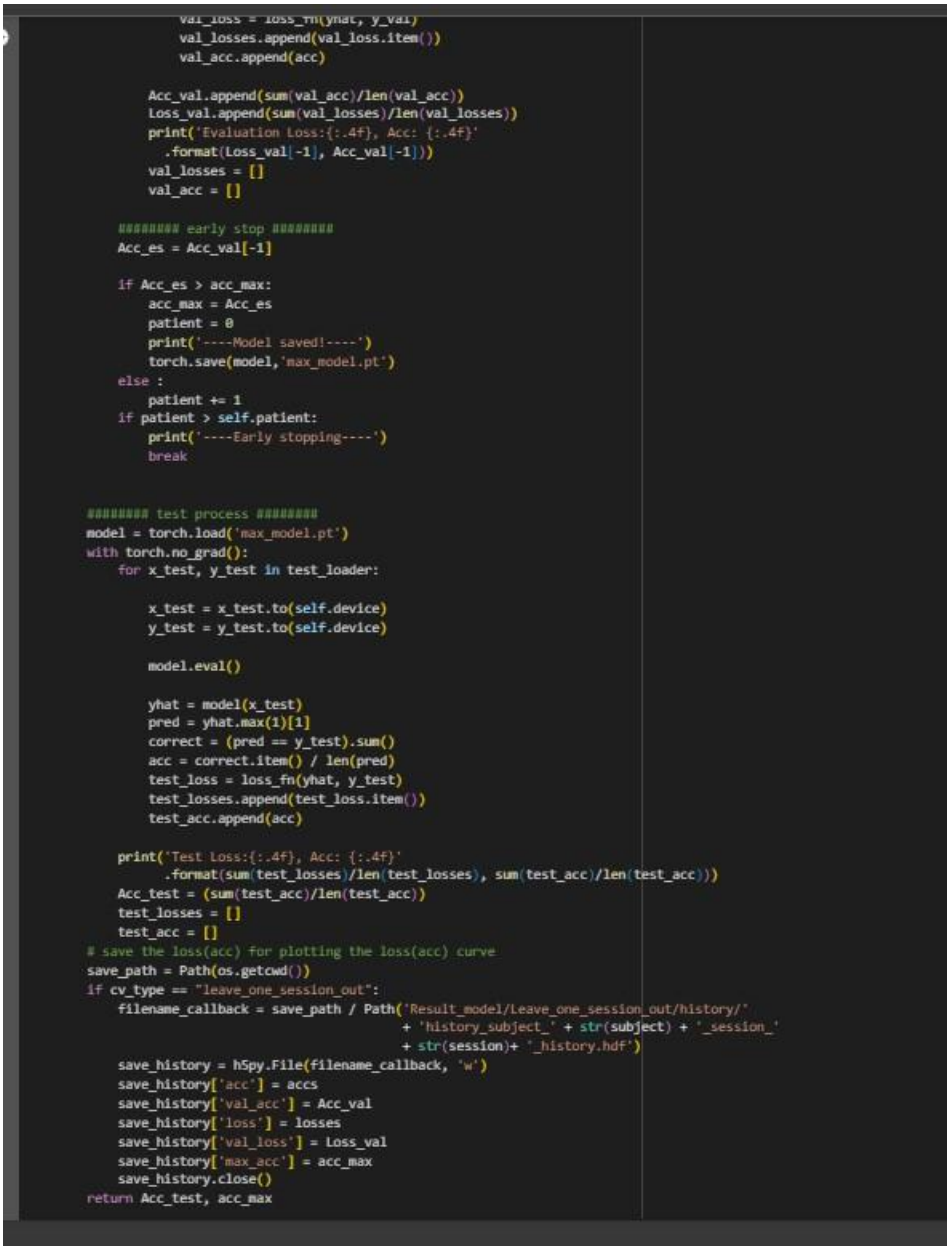
```
        data = torch.ones((1, 1, input_size[-2], int(input_size[-1])))
        y = self.Tception1(data)
        out = y
        y = self.Tception2(data)
        out = torch.cat((out, y), dim=-1)
        y = self.Tception3(data)
        out = torch.cat((out, y), dim=-1)
        out = self.BN_t(out)
        z = self.Sception1(out)
        out_final = z
        z = self.Sception2(out)
        out_final = torch.cat((out_final, z), dim=2)
        out = self.BN_s(out_final)
        out = out.view(out.size()[0], -1)
        return out.size()
##########################################  Temporal  ##########################################
class Tception(nn.Module):
    def __init__(self, num_classes, input_size, sampling_rate, num_T, hidden, dropout_rate):
        # input_size: channel x datapoint
        super(Tception, self).__init__()
        self.inception_window = [0.5, 0.25, 0.125, 0.0625, 0.03125]
        # by setting the convolutional kernel being (1,lenght) and the strids being 1 we can use conv2d to
        # achieve the 1d convolution operation
        self.Tception1 = nn.Sequential(
            nn.Conv2d(1, num_T, kernel_size=(1,int(self.inception_window[0]*sampling_rate)), stride=1, padding=0),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(1,16), stride=(1,16)))
        self.Tception2 = nn.Sequential(
            nn.Conv2d(1, num_T, kernel_size=(1,int(self.inception_window[1]*sampling_rate)), stride=1, padding=0),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(1,16), stride=(1,16)))
        self.Tception3 = nn.Sequential(
            nn.Conv2d(1, num_T, kernel_size=(1,int(self.inception_window[2]*sampling_rate)), stride=1, padding=0),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(1,16), stride=(1,16)))

        self.BN_t = nn.BatchNorm2d(num_T)

        size = self.get_size(input_size,sampling_rate,num_T)
        self.fc1 = nn.Sequential(
            nn.Linear(size[1], hidden),
            nn.ReLU(),
            nn.Dropout(dropout_rate))
        self.fc2 = nn.Sequential(
            nn.Linear(hidden, num_classes))

    def forward(self, x):
        y = self.Tception1(x)
        out = y
        y = self.Tception2(x)
        out = torch.cat((out,y),dim = -1)
        y = self.Tception3(x)
        out = torch.cat((out,y),dim = -1)
        out = self.BN_t(out)
        out = out.view(out.size()[0], -1)
        out = self.fc1(out)
        out = self.fc2(out)
        return out
    def get_size(self,input_size,sampling_rate,num_T):
        data = torch.ones((1,1,input_size[0],input_size[1]))
        y = self.Tception1(data)
        out = y
        y = self.Tception2(data)
        out = torch.cat((out,y),dim = -1)
        y = self.Tception3(data)
        out = torch.cat((out,y),dim = -1)
        out = self.BN_t(out)
```

```
        y = self.Tception2(data)
        out = torch.cat((out,y),dim = -1)
        y = self.Tception3(data)
        out = torch.cat((out,y),dim = -1)
        out = self.BN_t(out)
        out = out.view(out.size()[0], -1)
        return out.size()

################################################ Spacial ########################################################
class Sception(nn.Module):
    def __init__(self, num_classes, input_size, sampling_rate, num_S, hidden, dropout_rate):
        # input_size: channel x datapoint
        super(Sception, self).__init__()

        self.Sception1 = nn.Sequential(
            nn.Conv2d(1, num_S, kernel_size=(int(input_size[0]),1), stride=1, padding=0),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(1,16), stride=(1,16)))
        self.Sception2 = nn.Sequential(
            nn.Conv2d(1, num_S, kernel_size=(int(input_size[0]*0.5),1), stride=(int(input_size[0]*0.5),1), padding=0),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(1,16), stride=(1,16)))

        self.BN_s = nn.BatchNorm2d(num_S)

        size = self.get_size(input_size)

        self.fc1 = nn.Sequential(
            nn.Linear(size[1], hidden),
            nn.ReLU(),
            nn.Dropout(dropout_rate))
        self.fc2 = nn.Sequential(
            nn.Linear(hidden, num_classes))

    def forward(self, x):
        y = self.Sception1(x)
        out = y
        y = self.Sception2(x)
        out = torch.cat((out,y),dim = 2)
        out = self.BN_s(out)
        out = out.view(out.size()[0], -1)
        out = self.fc1(out)
        out = self.fc2(out)
        return out
    def get_size(self, input_size):
        data = torch.ones((1,1,input_size[0],input_size[1]))
        y = self.Sception1(data)
        out = y
        y = self.Sception2(data)
        out = torch.cat((out,y),dim = 2)
        out = self.BN_s(out)
        out = out.view(out.size()[0], -1)
        return out.size()
```

```python
class TrainModel():
    def __init__(self):
        self.data = None
        self.label = None
        self.result = None
        self.input_shape = None # should be (eeg_channel, time data point)
        self.model = 'TSception'
        self.cross_validation = 'Session' # Subject
        self.sampling_rate = 256

        self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
        # Parameters: Training process
        self.random_seed = 42
        self.learning_rate = 1e-3
        self.num_epochs = 200
        self.num_class = 2
        self.batch_size = 128
        self.patient = 4

        # Parameters: Model
        self.dropout = 0.3
        self.hidden_node = 128
        self.T = 9
        self.S = 6
        self.Lambda = 1e-6

    def load_data(self, path):
        '''
        This is the function to load the data
        Data format : .hdf
        Input : path
                the path of your data
                type = string
        Data dimension : (subject x trials x segments x 1 x channel x data) type = numpy.array
        Label dimension : (subject x trials x segments) type = numpy.array
        Note : For different data formats, please change the loading
               functions, (e.g. use h5py.File to load NAME.hdf)

        '''
        path = Path(path)
        dataset = h5py.File(path, 'r')
        self.data = np.array(dataset['data'])
        self.label = np.array(dataset['label'])

        # The input_shape should be (channel x data)
        self.input_shape = self.data[0,0,0,0].shape

        print('Data loaded!\nData shape:[{}], Label shape:[{}]'
              .format(self.data.shape,self.label.shape))

    def set_parameter(self, cv, model, number_class, sampling_rate,
                      random_seed, learning_rate, epoch, batch_size,
                      dropout, hidden_node, patient,
                      num_T, num_S, Lambda):
        '''
        This is the function to set the parameters of training process and model
        All the settings will be saved into a NAME.txt file
        Input : cv --
                    The cross-validation type
                    Type = string
                    Default : Leave_one_session_out
                    Note : for different cross validation type, please add the
                           corresponding cross validation function. (e.g. self.Leave_one_session_out())

                model --
                    The model you want choose
```

```
        '''
        self.model = model
        self.sampling_rate = sampling_rate
        # Parameters: Training process
        self.random_seed = random_seed
        self.learning_rate = learning_rate
        self.num_epochs = epoch
        self.num_class = number_class
        self.batch_size = batch_size
        self.patient = patient
        self.Lambda = Lambda

        # Parameters: Model
        self.dropout = dropout
        self.hidden_node = hidden_node
        self.T = num_T
        self.S = num_S


        #Save to log file for checking
        if cv == "Leave_one_subject_out":
            file = open("result_subject.txt",'a')
        elif cv == "Leave_one_session_out":
            file = open("result_session.txt",'a')
        elif cv == "K_fold":
            file = open("result_k_fold.txt",'a')
        file.write("\n"+ str(datetime.datetime.now())+
              "\nTrain:Parameter setting for " + str(self.model) +
              "\n1)number_class:" + str(self.num_class) + "\n2)random_seed:" + str(self.random_seed)+
              "\n3)learning_rate:" + str(self.learning_rate) + "\n4)num_epochs:" + str(self.num_epochs) +
              "\n5)batch_size:" + str(self.batch_size)+
              "\n6)dropout:" + str(self.dropout) + "\n7)sampling_rate:" + str(self.sampling_rate) +
              "\n8)hidden_node:" + str(self.hidden_node) + "\n9)input_shape:" + str(self.input_shape) +
              "\n10)patient:" + str(self.patient) + "\n11)T:" + str(self.T) +
              "\n12)S:" + str(self.S) + "\n13)Lambda:" + str(self.Lambda) + '\n')

        file.close()

    def Leave_one_session_out(self):
        '''
        This is the function to achieve 'Leave one session out' cross-validation
        To know more details about 'Leave one session out', please refer to our paper

        Note : all the acc and std will be logged into the result_session.txt

               The txt file is located at the same location as the python script

        '''
        save_path = Path(os.getcwd())
        if not os.path.exists(save_path / Path('Result_model/Leave_one_session_out/history')):
            os.makedirs(save_path / Path('Result_model/Leave_one_session_out/history'))
        #Data dimension: subject x trials x segments x 1 x channel x data
        #Label dimension: subject x trials x segments
        #Session: trials[0:2]-session 1; trials[2:4]-session 2; trials[4:end]-session 3
        data = self.data
        label = self.label
        shape_data = data.shape
        shape_label = label.shape
        subject = shape_data[0]
        trial = shape_data[1]
        session = int(shape_data[1]/2)
        channel = shape_data[4]
        frequency = shape_data[3]
        print("Train:Leave_one_session_out \n1)shape of data:" + str(shape_data) + " \n2)shape of label:" + str(shape_label)+
              " \n3)trials:" + str(trial) + " \n4)session:" + str(session) +
              " \n5)datapoint:" + str(frequency) + " \n6)channel:" + str(channel))
```

```
        return train, train_label, val, val_label

    def make_train_step(self, model, loss_fn, optimizer):
        def train_step(x,y):
            model.train()
            yhat = model(x)
            pred = yhat.max(1)[1]
            correct = (pred == y).sum()
            acc = correct.item() / len(pred)
            # L1 regularization
            loss_r = self.regulization(model,self.Lambda)
            # yhat is in one-hot representation;
            loss = loss_fn(yhat, y) + loss_r
            #loss = loss_fn(yhat, y)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
            return loss.item(), acc
        return train_step

    def regulization(self, model, Lambda):
        w = torch.cat([x.view(-1) for x in model.parameters()])
        err = Lambda * torch.sum(torch.abs(w))
        return err

    def train(self, train_data, train_label, test_data, test_label, val_data,
              val_label, subject, session, cv_type):
        # print('Available device:' + str(torch.cuda.get_device_name(torch.cuda.current_device())))
        torch.manual_seed(self.random_seed)
        torch.backends.cudnn.deterministic = True
        # Train and validation loss
        losses = []
        accs = []

        Acc_val = []
        Loss_val = []
        val_losses = []
        val_acc = []

        test_losses = []
        test_acc = []
        Acc_test = []

        # hyper-parameter
        learning_rate = self.learning_rate
        num_epochs = self.num_epochs

        # build the model
        if self.model == 'Sception':
            model = Sception(num_classes = self.num_class, input_size = self.input_shape,
                             sampling_rate = self.sampling_rate, num_S = self.S,
                             hidden = self.hidden_node, dropout_rate = self.dropout)
        elif self.model == 'Tception':
            model = Tception(num_classes = self.num_class, input_size = self.input_shape,
                             sampling_rate = self.sampling_rate, num_T = self.T,
                             hidden = self.hidden_node, dropout_rate = self.dropout)
        elif self.model == 'TSception':
            model = TSception(num_classes = self.num_class, input_size = self.input_shape,
                              sampling_rate = self.sampling_rate, num_T = self.T, num_S = self.S,
                              hidden = self.hidden_node, dropout_rate = self.dropout)

        optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

        loss_fn = nn.CrossEntropyLoss()

        if torch.cuda.is_available():
```



```
                val_loss = loss_fn(yhat, y_val)
                val_losses.append(val_loss.item())
                val_acc.append(acc)

            Acc_val.append(sum(val_acc)/len(val_acc))
            Loss_val.append(sum(val_losses)/len(val_losses))
            print('Evaluation Loss:{:.4f}, Acc: {:.4f}'
              .format(Loss_val[-1], Acc_val[-1]))
            val_losses = []
            val_acc = []

        ######## early stop ########
        Acc_es = Acc_val[-1]

        if Acc_es > acc_max:
            acc_max = Acc_es
            patient = 0
            print('----Model saved!----')
            torch.save(model,'max_model.pt')
        else :
            patient += 1
        if patient > self.patient:
            print('----Early stopping----')
            break


    ######## test process ########
    model = torch.load('max_model.pt')
    with torch.no_grad():
        for x_test, y_test in test_loader:

            x_test = x_test.to(self.device)
            y_test = y_test.to(self.device)

            model.eval()

            yhat = model(x_test)
            pred = yhat.max(1)[1]
            correct = (pred == y_test).sum()
            acc = correct.item() / len(pred)
            test_loss = loss_fn(yhat, y_test)
            test_losses.append(test_loss.item())
            test_acc.append(acc)

        print('Test Loss:{:.4f}, Acc: {:.4f}'
              .format(sum(test_losses)/len(test_losses), sum(test_acc)/len(test_acc)))
        Acc_test = (sum(test_acc)/len(test_acc))
        test_losses = []
        test_acc = []
    # save the loss(acc) for plotting the loss(acc) curve
    save_path = Path(os.getcwd())
    if cv_type == "leave_one_session_out":
        filename_callback = save_path / Path('Result_model/Leave_one_session_out/history/'
                                             + 'history_subject_' + str(subject) + '_session_'
                                             + str(session)+ '_history.hdf')
        save_history = hSpy.File(filename_callback, 'w')
        save_history['acc'] = accs
        save_history['val_acc'] = Acc_val
        save_history['loss'] = losses
        save_history['val_loss'] = Loss_val
        save_history['max_acc'] = acc_max
        save_history.close()
    return Acc_test, acc_max
```

در مرحله اول الگوریتم TSception را اجرا می کنیم.

TSception

```
train = TrainModel()
train.load_data('data_split.hdf')
train.set_parameter( cv = 'Leave_one_session_out',
                     model = 'TSception',
                     number_class = 2,
                     sampling_rate = 256,
                     random_seed = 42,
                     learning_rate = 0.001,
                     epoch = 200,
                     batch_size = 128,
                     dropout = 0.3,
                     hidden_node = 128,
                     patient = 4,
                     num_T = 9,
                     num_S = 6,
                     Lambda = 0.000001)
train.Leave_one_session_out()
```

```
Evaluation Loss:0.0284, Acc: 0.9883
```

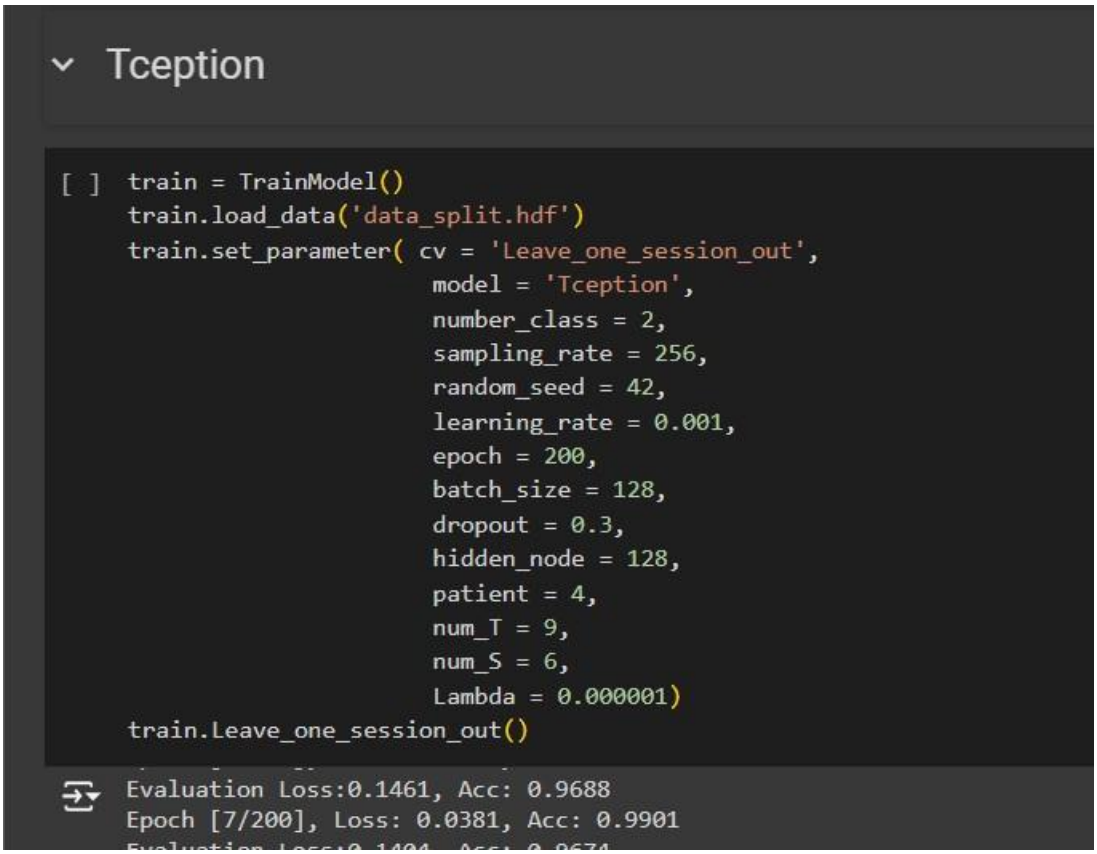
سپس الگوریتم Sception را اجرا می کنیم.

Sception

```
[ ] train = TrainModel()
    train.load_data('data_split.hdf')
    train.set_parameter( cv = 'Leave_one_session_out',
                         model = 'Sception',
                         number_class = 2,
                         sampling_rate = 256,
                         random_seed = 42,
                         learning_rate = 0.001,
                         epoch = 200,
                         batch_size = 128,
                         dropout = 0.3,
                         hidden_node = 128,
                         patient = 4,
                         num_T = 9,
                         num_S = 6,
                         Lambda = 0.000001)
    train.Leave_one_session_out()
```

```
Epoch [13/200], Loss: 0.0321, Acc: 0.9958
Evaluation Loss:0.2973, Acc: 0.9079
```

سپس الگوریتم Tception را اجرا می کنیم.



حال سراغ SVM می رویم. ابتدا SVM با استخراج ویژگی و PCA

Feature extraction and PCA

```
@njit
def fast_accuracy_score(y_true, y_pred):
    correct = 0
    for i in range(len(y_true)):
        if y_true[i] == y_pred[i]:
            correct += 1
    return correct / len(y_true)

def extract_features(signal):
    features = []
    for i in range(signal.shape[0]):
        sig = signal[i]
        mean = np.mean(sig)
        var = np.var(sig)
        max_val = np.max(sig)
        min_val = np.min(sig)
        range_val = max_val - min_val
        std_dev = np.std(sig)
        mean_power = np.mean(sig ** 2)
        energy = np.sum(sig ** 2)
        mean_abs = np.mean(np.abs(sig))
        freq_domain = np.abs(fft(sig))
        dom_freq = np.argmax(freq_domain)
        features.extend([mean, var, max_val, min_val, range_val, std_dev, mean_power, energy, mean_abs, dom_freq])
    return features

class TrainModel():
    def __init__(self):
        self.data = None
        self.label = None
        self.result = None
        self.input_shape = None
        self.cross_validation = 'Session' # Subject
        self.sampling_rate = 256

        # Parameters: Training process
        self.random_seed = 42
        self.num_class = 2
        self.test_size = 0.2

        # Parameters: Model
        self.kernel = 'rbf'
        self.C = 1.0
        self.gamma = 'scale'

    def load_data(self, path):
        path = Path(path)
        dataset = h5py.File(path, 'r')
        self.data = np.array(dataset['data'])
        self.label = np.array(dataset['label'])

        print('Data loaded!\nData shape:[{}], Label shape:[{}]'
              .format(self.data.shape, self.label.shape))

    def set_parameter(self, cv, kernel, C, gamma, test_size):
        self.kernel = kernel
        self.C = C
        self.gamma = gamma
        self.test_size = test_size

        # Save to log file for checking
        if cv == "Leave_one_subject_out":
```

```
                pca = PCA(n_components=0.99)  # Keep 99% of variance
                data_train = pca.fit_transform(data_train)
                data_val = pca.transform(data_val)
                data_test = pca.transform(data_test)

                # # Print the number of features after applying PCA
                # print(f'Number of features after PCA: {data_train.shape[1]}')

                model = SVC(kernel=self.kernel, C=self.C, gamma=self.gamma, random_state=self.random_seed)
                model.fit(data_train, label_train)

                acc_train = fast_accuracy_score(label_train, model.predict(data_train))
                acc_val = fast_accuracy_score(label_val, model.predict(data_val))
                acc_test = fast_accuracy_score(label_test, model.predict(data_test))

                ACC_subject.append(acc_test)
                ACC_subject_val.append(acc_val)

                print(f'Subject:{i}, Session:{j}, Train ACC:{acc_train:.4f}, Val ACC:{acc_val:.4f}, Test ACC:{acc_test:.4f}')

            ACC_subject = np.array(ACC_subject)
            mAcc = np.mean(ACC_subject)
            std = np.std(ACC_subject)

            ACC_val = np.array(ACC_subject_val)
            mAcc_val = np.mean(ACC_val)

            print("Subject:" + str(i) + "\nmACC: %.2f" % mAcc)
            print("std: %.2f" % std)

            file = open("result_session.txt", 'a')
            file.write('Subject:' + str(i) + ' MeanACC:' + str(mAcc) + ' Std:' + str(std) + '\n')
            file.close()

            ACC.append(ACC_subject)
            ACC_mean.append(mAcc)
            ACC_mean_val.append(mAcc_val)

        self.result = ACC
        file = open("result_session.txt", 'a')
        file.write("\n" + str(datetime.datetime.now()) +'\nMeanACC:'+ str(np.mean(ACC_mean)) +
                   ' Std:' + str(np.std(ACC_mean)) + ' Mean Val ACC:'+ str(np.mean(ACC_mean_val)) + '\n')
        file.close()
        print("Mean ACC:" + str(np.mean(ACC_mean)) + ' Std:' + str(np.std(ACC_mean)))

        save_path = Path(os.getcwd())
        filename_data = save_path / Path('Result_model/Result.hdf')
        save_data = h5py.File(filename_data, 'w')
        save_data['result'] = self.result
        save_data.close()
        return np.mean(ACC_mean)

if __name__ == "__main__":
    print(f'Model with parameters: (kernel=\'rbf\', C=1.0, gamma=\'scale\') is starting...')
    train = TrainModel()
    train.load_data('data_split.hdf')
    train.set_parameter(cv='Leave_one_session_out', kernel='rbf', C=1.0, gamma='scale', test_size=0.2)
    train.Leave_one_session_out()
```

```
Model with parameters: (kernel='rbf', C=1.0, gamma='scale') is starting...
Data loaded!
Data shape:[(2, 6, 574, 1, 4, 1024)], Label shape:[(2, 6, 574)]
Train:Leave_one_session_out
1)shape of data:(2, 6, 574, 1, 4, 1024)
2)shape of label:(2, 6, 574)
3)trials:6
4)sessions:3
```

سپس SVM بدون استخراج ویژگی و بدون PCA

Without feature extraction

Whithout PCA

```
@njit
def fast_accuracy_score(y_true, y_pred):
    correct = 0
    for i in range(len(y_true)):
        if y_true[i] == y_pred[i]:
            correct += 1
    return correct / len(y_true)

class TrainModel():
    def __init__(self):
        self.data = None
        self.label = None
        self.result = None
        self.input_shape = None
        self.cross_validation = 'Session' # Subject
        self.sampling_rate = 256

        # Parameters: Training process
        self.random_seed = 42
        self.num_class = 2
        self.test_size = 0.2

        # Parameters: Model
        self.kernel = 'rbf'
        self.C = 1.0
        self.gamma = 'scale'

    def load_data(self, path):
        path = Path(path)
        dataset = h5py.File(path, 'r')
        self.data = np.array(dataset['data'])
        self.label = np.array(dataset['label'])

        # The input_shape should be (channel x data)
        self.input_shape = self.data[0,0,0,0].shape

        print('Data loaded!\nData shape:[{}], Label shape:[{}]'
              .format(self.data.shape,self.label.shape))

    def set_parameter(self, cv, kernel, C, gamma, test_size):
        self.kernel = kernel
        self.C = C
        self.gamma = gamma
        self.test_size = test_size

        # Save to log file for checking
        if cv == "Leave_one_subject_out":
            file = open("result_subject.txt",'a')
        elif cv == "Leave_one_session_out":
            file = open("result_session.txt",'a')
        elif cv == "K_fold":
            file = open("result_k_fold.txt",'a')
        file.write("\n"+ str(datetime.datetime.now())+
              "\nTrain:Parameter setting for SVM" +
              "\n1)kernel:" + str(self.kernel) + "\n2)C:" + str(self.C) +
              "\n3)gamma:" + str(self.gamma) + "\n4)test_size:" + str(self.test_size) + '\n')
```

```
                scaler = StandardScaler()
                data_train = scaler.fit_transform(data_train)
                data_val = scaler.transform(data_val)
                data_test = scaler.transform(data_test)

                # # Applying PCA
                # pca = PCA(n_components=0.99)  # Keep 99% of variance
                # data_train = pca.fit_transform(data_train)
                # data_val = pca.transform(data_val)
                # data_test = pca.transform(data_test)

                model = SVC(kernel=self.kernel, C=self.C, gamma=self.gamma, random_state=self.random_seed, max_iter = 10000)
                model.fit(data_train, label_train)

                acc_train = fast_accuracy_score(label_train, model.predict(data_train))
                acc_val = fast_accuracy_score(label_val, model.predict(data_val))
                acc_test = fast_accuracy_score(label_test, model.predict(data_test))

                ACC_subject.append(acc_test)
                ACC_subject_val.append(acc_val)

                print(f'Subject:{i}, Session:{j}, Train ACC:{acc_train:.4f}, Val ACC:{acc_val:.4f}, Test ACC:{acc_test:.4f}')

            ACC_subject = np.array(ACC_subject)
            mAcc = np.mean(ACC_subject)
            std = np.std(ACC_subject)

            ACC_val = np.array(ACC_subject_val)
            mAcc_val = np.mean(ACC_val)

            print("Subject:" + str(i) + "\nmACC: %.2f" % mAcc)
            print("std: %.2f" % std)

            file = open("result_session.txt", 'a')
            file.write('Subject:' + str(i) + ' MeanACC:' + str(mAcc) + ' Std:' + str(std) + '\n')
            file.close()

            ACC.append(ACC_subject)
            ACC_mean.append(mAcc)
            ACC_mean_val.append(mAcc_val)

        self.result = ACC
        file = open("result_session.txt", 'a')
        file.write("\n" + str(datetime.datetime.now()) + '\nMeanACC:' + str(np.mean(ACC_mean)) +
                   ' Std:' + str(np.std(ACC_mean)) + ' Mean Val ACC:' + str(np.mean(ACC_mean_val)) + '\n')
        file.close()
        print("Mean ACC:" + str(np.mean(ACC_mean)) + ' Std:' + str(np.std(ACC_mean)))

        save_path = Path(os.getcwd())
        filename_data = save_path / Path('Result_model/Result.hdf')
        save_data = h5py.File(filename_data, 'w')
        save_data['result'] = self.result
        save_data.close()
        return(np.mean(ACC_mean))

print(f'Model with parameters: (kernel=\'rbf\', C=1.0, gamma=\'scale\') is starting...')
train = TrainModel()
train.load_data('data_split.hdf')
train.set_parameter(cv='Leave_one_session_out', kernel='rbf', C=1.0, gamma='scale', test_size=0.2)
train.Leave_one_session_out()
```

```
Model with parameters: (kernel='rbf', C=1.0, gamma='scale') is starting...
Data loaded!
Data shape:[(2, 6, 574, 1, 4, 1024)], Label shape:[(2, 6, 574)]
Train:Leave_one_session_out
1)shape of data:(2, 6, 574, 1, 4, 1024)
```

سپس SVM بدون استخراج ویژگی و با PCA

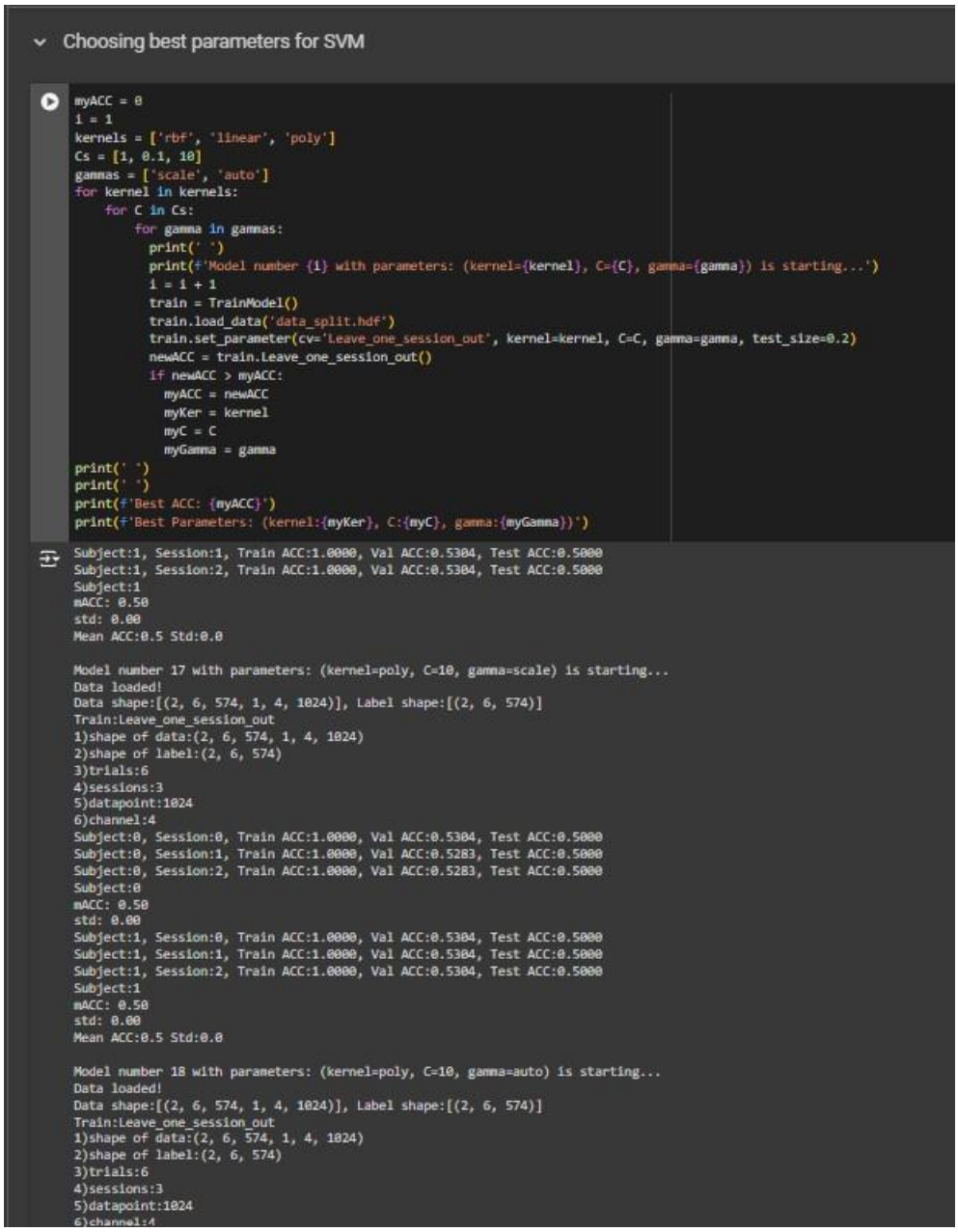
With PCA

```
@njit
def fast_accuracy_score(y_true, y_pred):
    correct = 0
    for i in range(len(y_true)):
        if y_true[i] == y_pred[i]:
            correct += 1
    return correct / len(y_true)

class TrainModel():
    def __init__(self):
        self.data = None
        self.label = None
        self.result = None
        self.input_shape = None
        self.cross_validation = 'Session' # Subject
        self.sampling_rate = 256

        # Parameters: Training process
        self.random_seed = 42
        self.num_class = 2
        self.test_size = 0.2

        # Parameters: Model
        self.kernel = 'rbf'
        self.C = 1.0
        self.gamma = 'scale'

    def load_data(self, path):
        path = Path(path)
        dataset = h5py.File(path, 'r')
        self.data = np.array(dataset['data'])
        self.label = np.array(dataset['label'])

        # The input_shape should be (channel x data)
        self.input_shape = self.data[0,0,0,0].shape

        print('Data loaded!\nData shape:[{}], Label shape:[{}]'
              .format(self.data.shape,self.label.shape))

    def set_parameter(self, cv, kernel, C, gamma, test_size):
        self.kernel = kernel
        self.C = C
        self.gamma = gamma
        self.test_size = test_size

        # Save to log file for checking
        if cv == "Leave_one_subject_out":
            file = open("result_subject.txt",'a')
        elif cv == "Leave_one_session_out":
            file = open("result_session.txt",'a')
        elif cv == "K_fold":
            file = open("result_k_fold.txt",'a')
        file.write("\n"+ str(datetime.datetime.now())+
              "\nTrain:Parameter setting for SVM" +
              "\n1)kernel:" + str(self.kernel) + "\n2)C:" + str(self.C) +
              "\n3)gamma:" + str(self.gamma) + "\n4)test_size:" + str(self.test_size) + '\n')

        file.close()

    def Leave_one_session_out(self):
        save_path = Path(os.getcwd())
        if not os.path.exists(save_path / Path('Result_model/Leave_one_session_out/history')):
            os.makedirs(save_path / Path('Result_model/Leave_one_session_out/history'))
```

```
                data_train = scaler.fit_transform(data_train)
                data_val = scaler.transform(data_val)
                data_test = scaler.transform(data_test)

                # Applying PCA
                pca = PCA(n_components=0.99)  # Keep 99% of variance
                data_train = pca.fit_transform(data_train)
                data_val = pca.transform(data_val)
                data_test = pca.transform(data_test)

                model = SVC(kernel=self.kernel, C=self.C, gamma=self.gamma, random_state=self.random_seed, max_iter = 100000)
                model.fit(data_train, label_train)

                acc_train = fast_accuracy_score(label_train, model.predict(data_train))
                acc_val = fast_accuracy_score(label_val, model.predict(data_val))
                acc_test = fast_accuracy_score(label_test, model.predict(data_test))

                ACC_subject.append(acc_test)
                ACC_subject_val.append(acc_val)

                print(f'Subject:{i}, Session:{j}, Train ACC:{acc_train:.4f}, Val ACC:{acc_val:.4f}, Test ACC:{acc_test:.4f}')

            ACC_subject = np.array(ACC_subject)
            mAcc = np.mean(ACC_subject)
            std = np.std(ACC_subject)

            ACC_val = np.array(ACC_subject_val)
            mAcc_val = np.mean(ACC_val)

            print("Subject:" + str(i) + "\nmACC: %.2f" % mAcc)
            print("std: %.2f" % std)

            file = open("result_session.txt", 'a')
            file.write('Subject:' + str(i) + ' MeanACC:' + str(mAcc) + ' Std:' + str(std) + '\n')
            file.close()

            ACC.append(ACC_subject)
            ACC_mean.append(mAcc)
            ACC_mean_val.append(mAcc_val)

        self.result = ACC
        file = open("result_session.txt", 'a')
        file.write("\n" + str(datetime.datetime.now()) + '\nMeanACC:' + str(np.mean(ACC_mean)) +
                   ' Std:' + str(np.std(ACC_mean)) + ' Mean Val ACC:' + str(np.mean(ACC_mean_val)) + '\n')
        file.close()
        print("Mean ACC:" + str(np.mean(ACC_mean)) + ' Std:' + str(np.std(ACC_mean)))

        save_path = Path(os.getcwd())
        filename_data = save_path / Path('Result_model/Result.hdf')
        save_data = h5py.File(filename_data, 'w')
        save_data['result'] = self.result
        save_data.close()
        return(np.mean(ACC_mean))

print(f'Model with parameters: (kernel=\'rbf\', C=1.0, gamma=\'scale\') is starting...')
train = TrainModel()
train.load_data('data_split.hdf')
train.set_parameter(cv='Leave_one_session_out', kernel='rbf', C=1.0, gamma='scale', test_size=0.2)
train.Leave_one_session_out()
```

```
Model with parameters: (kernel='rbf', C=1.0, gamma='scale') is starting...
Data loaded!
Data shape:[(2, 6, 574, 1, 4, 1024)], Label shape:[(2, 6, 574)]
Train:Leave_one_session_out
1)shape of data:(2, 6, 574, 1, 4, 1024)
2)shape of label:(2, 6, 574)
3)trials:6
4)sessions:3
```

در مرحله نهایی به دنبال پیدا کردن SVM با بهترین پارامتر ممکن هستیم.



Choosing best parameters for SVM

```
myACC = 0
i = 1
kernels = ['rbf', 'linear', 'poly']
Cs = [1, 0.1, 10]
gammas = ['scale', 'auto']
for kernel in kernels:
    for C in Cs:
        for gamma in gammas:
          print(' ')
          print(f'Model number {i} with parameters: (kernel={kernel}, C={C}, gamma={gamma}) is starting...')
          i = i + 1
          train = TrainModel()
          train.load_data('data_split.hdf')
          train.set_parameter(cv='Leave_one_session_out', kernel=kernel, C=C, gamma=gamma, test_size=0.2)
          newACC = train.Leave_one_session_out()
          if newACC > myACC:
            myACC = newACC
            myKer = kernel
            myC = C
            myGamma = gamma
print(' ')
print(' ')
print(f'Best ACC: {myACC}')
print(f'Best Parameters: (kernel:{myKer}, C:{myC}, gamma:{myGamma})')
```

```
Subject:1, Session:1, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:1, Session:2, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:1
mACC: 0.50
std: 0.00
Mean ACC:0.5 Std:0.0

Model number 17 with parameters: (kernel=poly, C=10, gamma=scale) is starting...
Data loaded!
Data shape:[(2, 6, 574, 1, 4, 1024)], Label shape:[(2, 6, 574)]
Train:Leave_one_session_out
1)shape of data:(2, 6, 574, 1, 4, 1024)
2)shape of label:(2, 6, 574)
3)trials:6
4)sessions:3
5)datapoint:1024
6)channel:4
Subject:0, Session:0, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:0, Session:1, Train ACC:1.0000, Val ACC:0.5283, Test ACC:0.5000
Subject:0, Session:2, Train ACC:1.0000, Val ACC:0.5283, Test ACC:0.5000
Subject:0
mACC: 0.50
std: 0.00
Subject:1, Session:0, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:1, Session:1, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:1, Session:2, Train ACC:1.0000, Val ACC:0.5304, Test ACC:0.5000
Subject:1
mACC: 0.50
std: 0.00
Mean ACC:0.5 Std:0.0

Model number 18 with parameters: (kernel=poly, C=10, gamma=auto) is starting...
Data loaded!
Data shape:[(2, 6, 574, 1, 4, 1024)], Label shape:[(2, 6, 574)]
Train:Leave_one_session_out
1)shape of data:(2, 6, 574, 1, 4, 1024)
2)shape of label:(2, 6, 574)
3)trials:6
4)sessions:3
5)datapoint:1024
6)channel:4
```

برای مشاهده دقیقتر و کاملتر به کد مراجعه کنید.