Прогнозирование снимков FMRI по просмотренному видео

Создание интеллектуальных систем, МФТИ

Барабанщикова Полина, Протасов Дмитрий, Шульган Никита

13 декабря 2022

Обзор статьи

Некоторые характеристики датасета

- 30 участников с данными FMRI
- 13 чередующихся эпизодов по 30 секунд (речь / музыка)
- Всё переведено на голландский
- Аудио и видеодорожка аннотированы (время начала и конца появления каждого объекта)
- Снимки FMRI в формате BIDS
- Папка видео → 135 tsv файлов (для 129 объектов, и 6 персонажей)

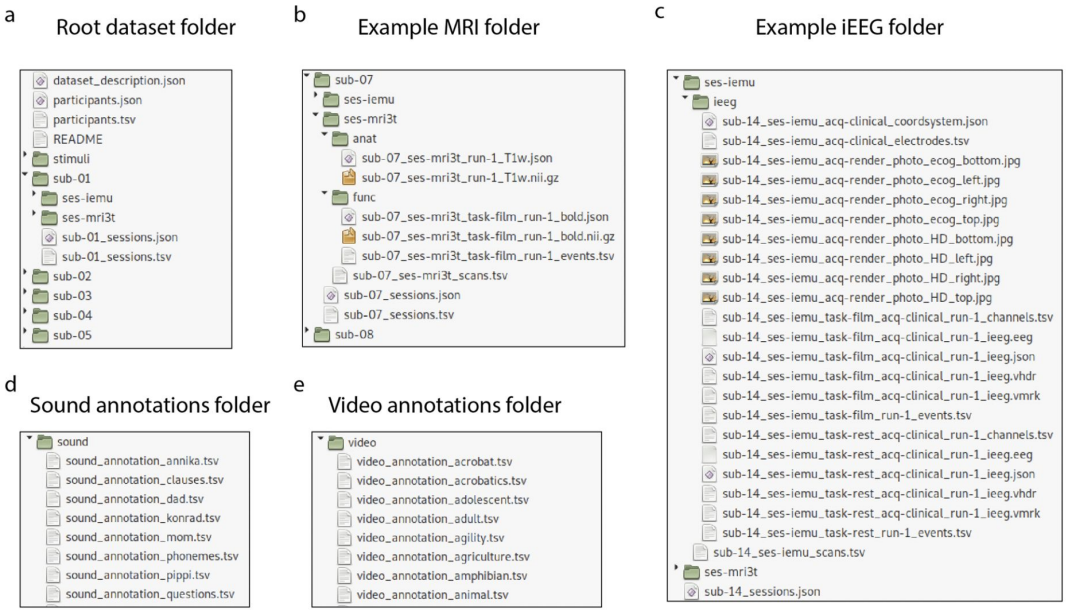


Fig. 2 Overview of data records. (**a**–**e**) Structure of folders and files in the dataset with example (f)MRI and iEEG folders from one subject.

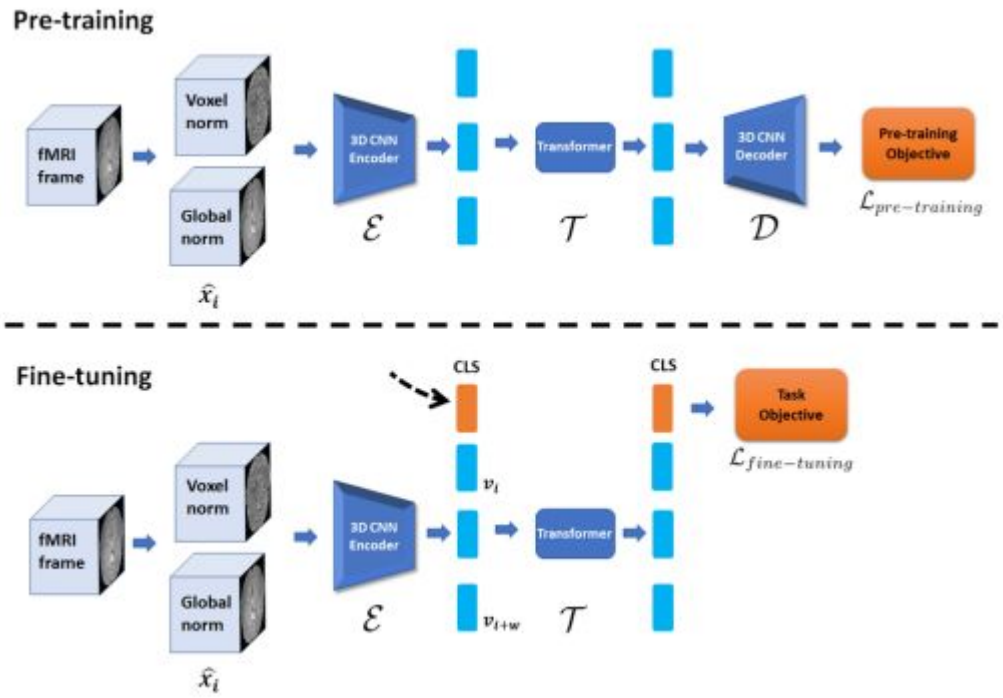
Датасет из статьи

1. Данные
   - 389.728 секунд видео = 9750 фреймов (24 кадра в секунду, размер кадра 480x640x3)
   - 641 измерение fMRI для пользователя (примерно 1.644 измерений в секунду, 40x64x64)
2. Соотношения
   - 3.04 секунды соответствуют 5 измерениям fMRI и 76 фреймам видео
   - Интервал fMRI вида 5*i:5*(i+k) соответствует интервалу видео 76*i:76*(i+k), где i >= 0, k > 0 – целые числа

Обзор литературы

TFF. Self-supervised transformers for FMRI representation

1. pretraining on reconstruction 3D-FMRI data (2 phases)
2. fine-tune on specific tasks and show SOTA performance
3. Architecture:
   - preprocessing (voxel, global)
   - 3D-CNN encoder
   - 2 layer transformer
   - 3D-CNN decoder
4. Loss = L1-rec-loss + intensity-loss + perceptual-loss



Обзор алгоритма обучения (TFF)

```
def main():
    model_phase1 = run_phase(args, None, 'autoencoder_reconstruction')
    model_phase2 = run_phase(args, model_phase1, 'tranformer_reconstruction')
    model_phase3 = run_phase(args, model_phase2, f'fine_tune_{fine_tune_task}')
    test(args, model_weights_path_phase3)

def run_phase(args, model_path, phase_name):
    S = ['train','val']
    trainer = Trainer(sets=S,**args)
    trainer.training()
    return model

class Trainer():
    def training(self):
        for epoch in range(self.nEpochs):
            self.train_epoch(epoch)

    def train_epoch(self,epoch):
        self.train()
        for input_dict in self.train_loader:
            self.optimizer.zero_grad()
            loss_dict, loss = self.forward_pass(input_dict)
            loss.backward()
            self.optimizer.step()
```

Машинное обучение для работы с видео

Общие идеи для работы с видео

- 3D свёртки по времени и пространству (минусы: много параметров, вычислительно дорого)
- Факторизация 3D свёрток на 2D для пространства и 1D для времени
- Two-Stream подход
- Inflated 3D CNN – использование предобученных на картинках 2D свёрток для начального приближения
- Использование разных типов attention

Задачи, на которые обратить внимание

- Video-to-Video
- Озвучивание видео

Статьи

- Серия обзоров методов работы с видео (https://towardsdatascience.com/deep-learning-on-video-part-one-the-early-days-8a3632ed47d4)
- Inflated 3D CNN (https://arxiv.org/pdf/1705.07750.pdf)

Предобученные модели

- https://github.com/facebookresearch/SlowFast

Video Transformer Network

(https://arxiv.org/pdf/2102.00719.pdf)

Обзор статьи

- Решается задача классификации видео
- Spatial backbone – любая сеть для обработки 2D изображений, используется для извлечения признаков (в оригинальной имплементации: Vision Transformer)
- Temporal attention-based encoder – модель трансформера с механизмом attention, учитывающая временные зависимости (в оригинальной имплементации: Longformer)
- MLP-Head – полносвязная сеть для классификации

Детали реализации

- VTN принимает на вход кадры размера 224x224
- Код и предобученные на Kinetics веса доступны в [https://github.com/bomri/SlowFast/blob/master/projects/vtn/README.md.](https://github.com/bomri/SlowFast/blob/master/projects/vtn/README.md)

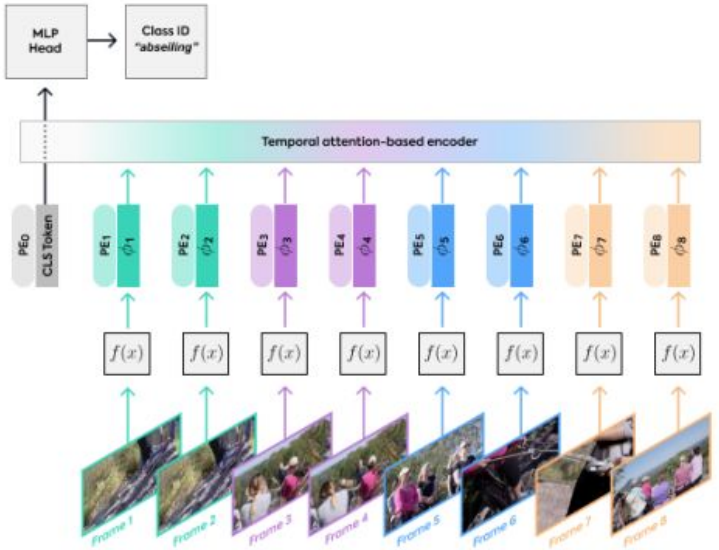


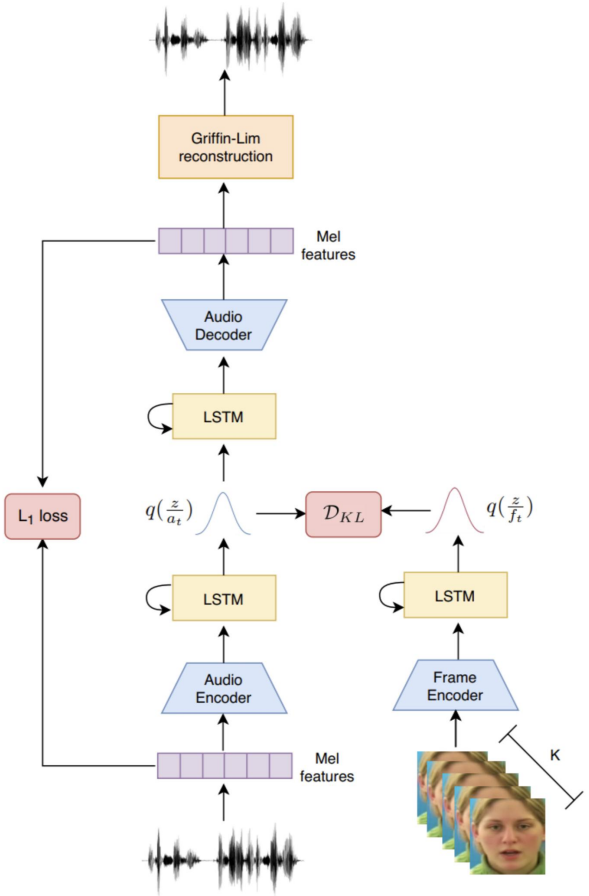
Figure 1: Video Transformer Network architecture. Connecting three modules: A 2D spatial backbone ($f(x)$), used for feature extraction. Followed by a temporal attention-based encoder (Longformer in this work), that uses the feature vectors (ϕ_i) combined with a position encoding. The $[CLS]$ token is processed by a classification MLP head to get the final class prediction.

Speech Prediction in Silent Videos using Variational Autoencoders (https://arxiv.org/abs/2011.07340)

Обзор статьи

- Решается задача озвучивания видео с применением вариационного автоэнкодера
- Во время обучения на вход подается последовательность кадров и звуковой сигнал
- Сначала с помощью энкодеров получают эмбеддинги независимо для каждого аудио/видео фрейма
- Далее применяют LSTM и полносвязный слой для получения среднего и дисперсии вариационных распределений q
- Оптимизируют ELBO (evidence lower bound)

$$\mathcal{L}(\theta, \phi; \mathrm{x}) = \sum_{t=1}^{N} \mathbb{E}_{q_{\phi_a}(z|a_t)}[\lambda \log p_{\theta_a}(a_t|z)] - \beta KL[q_{\phi_a}(z|a_t)||q_{\phi_f}(z|f_t)]$$



Предложенное решение

Описание модели

- Модель состоит из VAE для fMRI (TFF) и энкодера для видео (VTN)
- Входные данные — кортеж из предобработанной последовательности fMRI (2x40x64x64x5T), соответствующего видеоряда (3x224x224x76T) и дополнительного кадра fMRI (2x40x64x64x1)
- Последовательность fMRI подаётся на вход VAE. Энкодер генерирует векторы среднего и дисперсии размерности emb_dim для каждого измерения (5Txemb_dim). Декодер восстанавливает сигнал по семплу из гауссовского распределения с данными параметрами
- Энкодер для видео предсказывает по видеоряду векторы среднего и дисперсии той же размерности emb_dim для каждого измерения fMRI
- Они складываются со средним и дисперсией дополнительного кадра fMRI. Результат используется в качестве априорного распределения для VAE

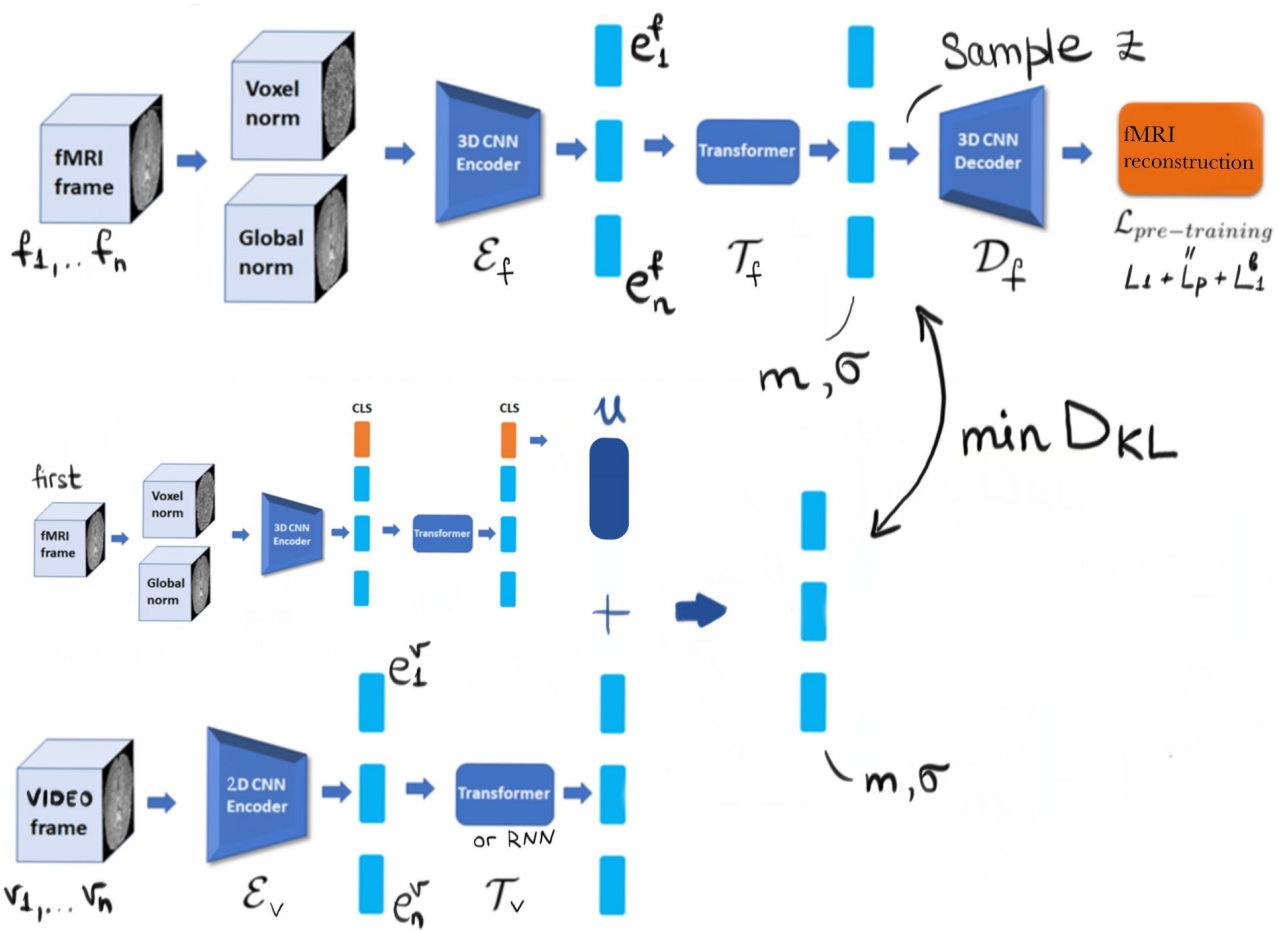
Функция потерь

$$\mathcal{L}(\theta, \psi) = E_{q_\psi} \log p_\theta(F|z, u) - D_{KL}[q_\psi(z|F, u)||p_\theta(z|u)],$$

$$\mathcal{L}(\theta, \psi) = E_{q_{\psi_f}} \log p_\theta(F|z, u) - D_{KL}[q_{\psi_f}(z|F, u)||q_{\psi_v}(z|V, u)].$$

- Аналог ELBO с априорным распределением, построенным по видео и предыдущему кадру fMRI
- Первая часть ELBO, отвечающая за реконструкцию сигнала, заменяется на трёхкомпонентный лосс из TFF

Архитектура: TFF + VTN



Эксперименты

Датасеты для обучения

1. Предобработка
   - по каждому снимку строятся voxel norm и global norm, которые сохраняются в отдельный файл
   - кадры видео обрезаются до размера 224x224
2. Датасет с fMRI
   - элемент датасета – 2 нормализованные версии для T измерений fMRI
   - каждый элемент является тензором размером (2, 40, 64, 64, T)
   - элементы подгружаются из памяти динамически
3. Датасет с fMRI и видео
   - элемент датасета – 2 нормализованные версии для T(+1) измерений fMRI, N соответствующих фреймов видео, индексы позиций для видео
   - пользователи поделены на трейн и тест
   - берется каждый i-ый кадр видео

Обучение

1. Обучается отдельно Энкодер-Декодер TFF (длина последовательности=1)
2. Обучается полная модель TFF (длина последовательности>1)
3. Загружаются предобученный веса VTN
4. В VTN число нейронов на последнем слое заменяется на 2*emb_dim
5. Обучается автоэнкодер с VTN (длина последовательности fMRI=5*k)

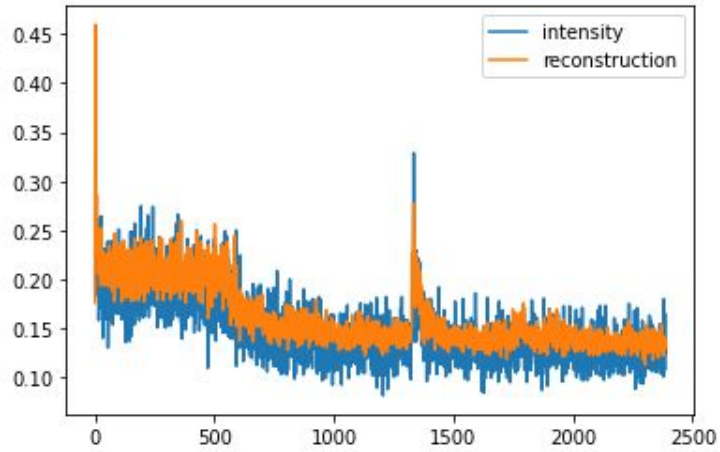
```
def main(base_path):
    args = get_arguments(base_path)
    # pretrain step1
    print('starting phase 1...')
    run_phase(args,None,'1','autoencoder_reconstruction')
    print('finishing phase 1...')
    #pretrain step2
    print('starting phase 2...')
    model_weights_path_phase1 = os.path.join(base_path, 'TFF_weights/AutoEncoder_last_epoch.pth')
    run_phase(args,model_weights_path_phase1, '2', 'tranformer_reconstruction')
    print('finishing phase 2...')
    #train step3
    print('starting phase 3...')
    model_weights_path_phase2 = os.path.join(base_path, 'TFF_weights/Transformer_last_epoch.pth')
    run_phase(args, model_weights_path_phase2,'3','video_vae')
    print('finishing phase 3...')
```

Псевдокод

```
class VAE_with_VTN(BaseModel):
    def __init__(self, dim,**kwargs):
        super(VAE_with_VTN, self).__init__()
        # ENCODING
        self.encoder = Encoder(**kwargs)

        # BottleNeck into bert
        self.into_bert = BottleNeck_in(**kwargs)

        # transformer
        self.transformer = Transformer_Block(self.BertConfig, **kwargs)

        # BottleNeck out of bert
        self.from_bert = BottleNeck_out_VAE(**kwargs)

        # DECODER
        self.decoder = Decoder(**kwargs)

        #VTN
        self.vtn = VTN((emb_dim, vtn_path)
```

```
def forward(self, input_dict):
    x = input_dict['fmri_seq']
    x_0 = input_dict['fmri_img']
    video = input_dict['video_seq']
    pos_idx = input_dict['pos_idx']
    #vae
    encoded = self.encoder(x)

    encoded = self.into_bert(encoded)
    encoded = encoded.reshape(batch_size, T, -1)
    out = self.transformer(encoded)
    out, mean_1, log_std_1 = self.from_bert(out)

    reconstructed_image = self.decoder(out)

    #encode fmri image
    encoded = self.encoder(x_0)
    encoded = self.into_bert(encoded)
    out = self.transformer(encoded)
    _, mean_0, log_std_0 = self.from_bert(out)

    #vtn
    mean_2, log_std_2 = self.vtn((video, pos_idx)).chunk(2, dim=-1)
    mean_2 = mean_2 + mean_0
    log_std_2 = torch.logaddexp(log_std_0, log_std_2)

    return {'reconstructed_fmri_sequence': reconstructed_image,
            'mean_1':mean_1, 'log_std_1':log_std_1,
            'mean_2':mean_2, 'log_std_2':log_std_2}
```

Autoencoder with Transformer



VAE with VTN