

# Cognitive Computational Modelling for Spatio-Temporal fMRI in Ventral Temporal Cortex

Can Kocagil, Arman Vural Budunoğlu, Emirhan İlhan  
Bilkent University  
Department of EEE

{can.kocagil,vural.budunoğlu,emirhan.ilhan}@ug.bilkent.edu.tr

## Abstract

*Visual decoding of distributed regions of the human ventral temporal cortex, saliency and attention have been active research topics in the context of computational cognitive neuroscience. We need to construct spatio-temporal decoding algorithms to perform cognitive tasks. In this study, we investigated the functional architecture of the object vision pathway in human brain by functional magnetic resonance imaging (fMRI) methods to decode the visual stimuli viewed by a human subject. We conducted state-of-the-art explanatory echo-planar, region-of-interest (RoI), statistical map, anatomical and glass brain pre-analysis to discover block-designed 4-D timeseries fMRI dataset, namely Haxby dataset, from the study on face and object representation. To understand geodesic relation in ventral temporal masked fMRI samples, we performed functional connectivity analysis based on the correlation, precision and partial correlation, and similarity analysis based on the cosine, minkowski and euclidean distances. Manifold learning and dimensionality reduction methods are performed on the per-subject ventral temporal masks to extract latent representations of spatio-temporal masks that will help further decoding of fMRIs. End-to-end machine learning algorithms from perceptrons to FREMs are developed to categorize the stimuli based on distributed and overlapping regions in ventral temporal cortex. We further constructed cognitive neural networks, precisely MLPs, 2D and 3D CNNs and spatially oriented vision transformers by taking the advantage of interactions between different streams of visual representations. Our comprehensive results demonstrate that the ensembling of regularized models achieves the best performance for robust decoding and spatially oriented attention mechanism can enlighten the understanding of attention in human brains.*

**Keywords-** Cognitive Computational Neuroscience, fMRI Decoding, Functional Connectivity, Manifold fMRI Learning, Neuroimaging, Spatio-temporal Attention.

## 1. Introduction

The ventral temporal cortex in the human brain is selective to the different representations of the visual stimuli from nature and ventral object vision pathway generates distributed and overlapping neural responses [21]. Single cell studies are conducted to demonstrate that the differential tuning of individual neurons in the ventral temporal cortex in nonhuman primates are selective the objects from different kinds and form representative features [6, 21]. However, their order of selectivity is not generalizable and scalable to higher degree of object representations [8]. To model neuro architecture of the ventral cortex, statistical algorithms are developed but the uncertainty in pathway remains. Recent developments regarding neuroimaging have demonstrated that spatio-temporal decoding of human's perception, memories and thoughts are decodable via functional magnetic resonance imaging (fMRI) methods [11]. However, the complexity and the distribution of fMRI data require sophisticated scientific tools because of its neural capacity of the spatio-temporal resolution. With the advancements of machine learning, neuroscientists discover statistical and structural patterns in large scale fMRI datasets to solve various tasks in the context of neuroscience. Further, recent advances in deep learning enable researchers to solve unsolved neuroscientific tasks [12] and concretely show the importance of deep learning. In this study, we build an end-to-end discovery machine learning pipelines to decode the category of visual stimuli viewed by a human subject based on fMRI data. Further, we construct vision transformers with spatially oriented attention mechanisms for understanding attention in human brains in depth. We utilize the state-of-the-art explanatory neuroimaging technologies such as echo-planar, region-of-interest (RoI), statistical map, anatomical and glass brain methods, to visualize and pre-analyze the visual structure of fMRI samples. Our ablation studies and experiments are based on a block-designed 4-D timeseries fMRI dataset, namely Haxby dataset [7, 15, 8], from the study on face and

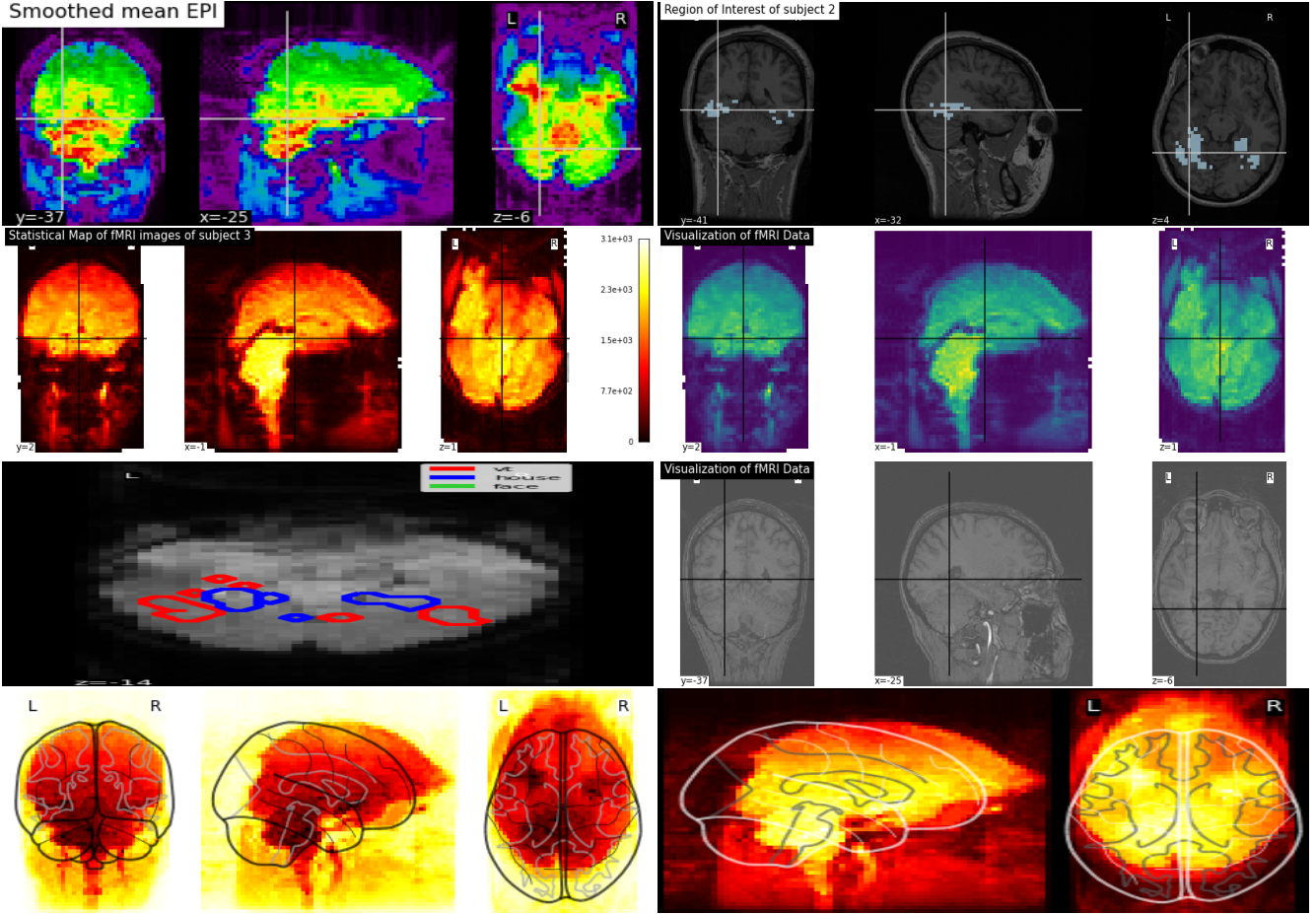


Figure 1: Visualizations of ventral temporal cortex of subjects with different kinds of neuroimaging technique. From top left to bottom right in sequential way, the visuals corresponds the smoothed temporally average EPI, RoI's, statistical map, direct fMRI, activated regions of brain by some stimulus, anatomical, white and black glass brain visuals of the subjects in the Haxby experiment.

object representation.

Further, we performed functional connectivity analysis based on the correlation, precision and partial correlation, and similarity analysis based on the cosine, minkowski and euclidean distance to discover overlapping representation in ventral temporal cortex. Then, manifold learning and dimensionality reduction methods are performed on the per-subject ventral temporal masks to extract latent variables of spatio-temporal masks that will help further decoding of human brain. As dimensionality reduction methods, we applied Principal Component Analysis (PCA), Linear Discriminate Analysis (LDA), Independent Component Analysis (ICA), Non-Negative Matrix Factorization (NNMF) and Multidimensional Scaling (MDS) then compare these obtained subspaces by their 3D visualization. Additionally, we performed manifold learning algorithms to extract underlying manifold distribution in masked ventral temporal

regions. We performed t-Stochastic Neighbour Embedding (t-SNE), Uniform Manifold Approximation and Projection (UMAP), ISOMAP, Locally Linear Embedding (LLE) and Spectral Embedding (SE) then compare their lower dimensional manifolds by their 3D visualizations that further help in the decoding process.

End-to-end machine learning algorithms are developed to categorize the stimuli based on distributed and overlapping regions in the ventral temporal cortex. Precisely, we performed the following machine learning algorithms; Linear support vector classifier (LinearSVC), Stochastic Gradient Descent Classifier (SGDClassifier), Multi-Layer-Perceptron (MLP), Perceptron, Logistic Regression, Logistic Regression Cross-Validation, Support Vector Classifier (SVC), Calibrated Classifier (Probability calibration with isotonic regression), Passive Aggressive Classifier, Label Propagation Classifier, Random Forest Classifier, Gradi-

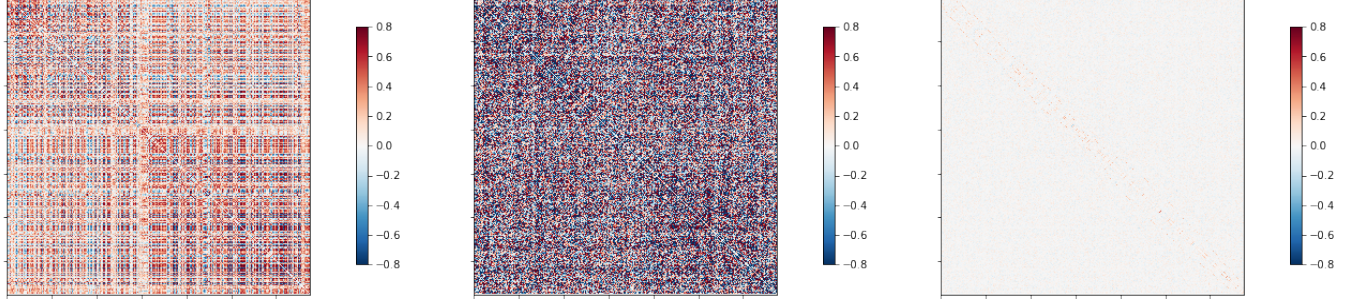


Figure 2: Functional Connectivity analysis on ventral temporal masks of the subjects are performed with correlation, precision and partial correlation measures in sequential way from left to right.

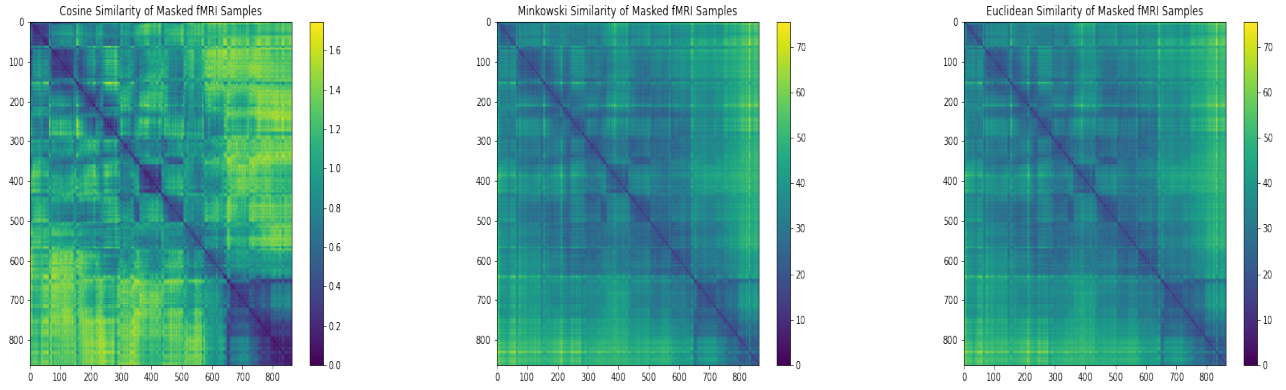


Figure 3: Similarity analysis on ventral temporal masks of the subjects are performed with cosine, minkowski and euclidean distance in sequential way from left to right.

ent Boosting Classifier, Quadratic Discriminant Classifier, Ridge Classifier Cross-Validation, Ridge Classifier, AdaBoost Classifier, Extra Trees Classifier, K-Neighbors Classifier, Bernoulli Naive Bayes Classifier, Gaussian Naive Bayes Classifier, Nu-Support Vector Classifier, Nearest Centroid Classifier and Bagging Classifier. As a robust ensemble decoding, we applied novel ensemble of regularized models; FREM: Cross-Validated Ensemble of L2 regularized SVCs, FREM: Cross-Validated Ensemble of L2 regularized Logistic Regressions. We further constructed cognitive neural networks, precisely MLPs with GELU non-linearity [10], 2D and 3D Convolutional Neural Network and spatially oriented vision transformers by taking the advantage of interactions between different streams of visual representations.

Our main study decomposed as follows.

- We build an end-to-end discovery machine learning and deep learning pipelines to decode the category of visual stimuli viewed by a human subject based on fMRI data.
- We utilize state-of-the-art explanatory neuroimaging

technologies such as echo-planar, region-of-interest (RoI), statistical map, anatomical and glass brain methods, to visualize and pre-analyze the visual structure of fMRI samples.

- Further, we performed functional connectivity analysis based on the correlation, precision and partial correlation, and similarity analysis based on the cosine, minkowski and euclidean distances to discover overlapping representation in the ventral temporal cortex.
- Then, manifold learning and dimensionality reduction methods are performed on the per-subject ventral temporal masks to extract latent variables of spatio-temporal masks.

The rest of the paper is organized as follows. We begin with the methods section that we explained methodology from unsupervised representation learning, functional connectivity analysis to machine & deep learning pipelines. Then, we present our findings, mainly results of the decoding process in terms of widely used evaluation metrics in the results section. Then, we interpret our results in light of previous studies in the discussion section.

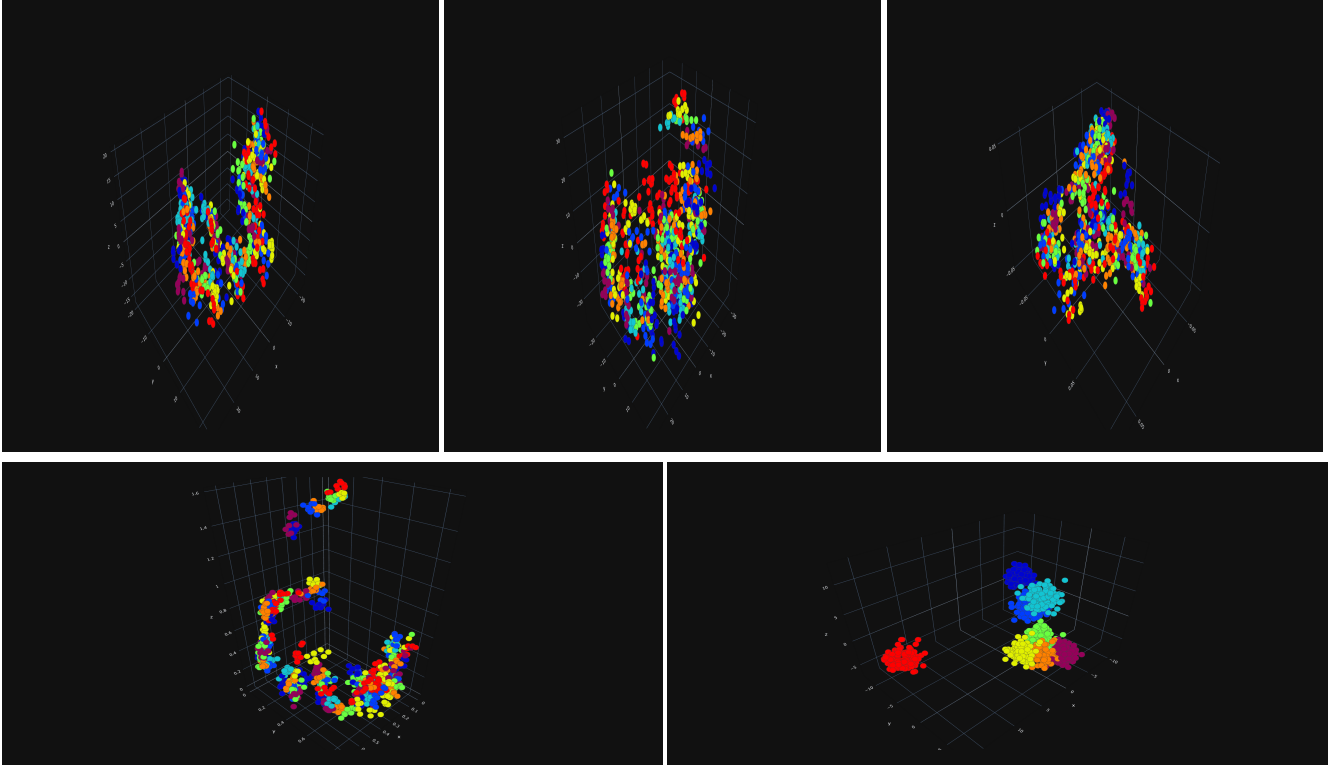


Figure 4: Dimension reduction algorithms: PCA, MDS, ICA, NNMF and LDA are performed with 3 component for visualization purposes that are presented in sequential way from top left to bottom right. Unique colors represent unique classes.

## 2. Methods

Our experiments are based on a block-designed 4-D time-series fMRI dataset, namely Haxby dataset [7, 15, 8], from the study of face and object representation. It consists of 6 subjects with 12 runs per subject [8]. In each run, the subjects passively viewed grey-scale images of eight object categories, grouped in 24s blocks separated by rest periods [8, 7]. Each image was shown for 500ms and was followed by a 1500ms inter-stimulus interval [7]. Full-brain fMRI data were recorded with a volume repetition time of 2.5s, thus, a stimulus block was covered by roughly 9 volumes [8]. It consist of per-subject high resolution anatomical images except for the sixth, 4D fMRI timeseries image data in the shape of 1452 volumes with 40x64x64 voxels (corresponding to a voxel size of 3.5 x 3.75 x 3.75 mm and a volume repetition time of 2.5 seconds) [8]. We have 8 different stimuli categories that are scissors, face, cat, scrambledpix, bottle, chair, shoe and house. The visuals of stimuli is presented in appendix A. The chunks of resting-state is eliminated as it provides no additional information on decoding visual stimuli [8]. We performed multiple explanatory and decoding experiments. In the following sections, we described our methodology in detail. We started with discovery and advanced visualizations of ventral temporal

cortex by different neuroimaging techniques, their underlying manifolds, functional connectome analysis of overlapping regions, and we provide descriptions of more than 30 ML & DL brain decoders.

### 2.1. Discovery fMRI Analysis by Neuroimaging

We performed pre-analysis based on the neuroimaging technologies to visualize the dataset. To accomplish that, we utilized the Echo-planar Averaging for 4-D visualization, region of interest (RoI), statistical map, anatomic, glass brain visualization tools embedded in statistical learning and neuroimaging framework, namely Nilearn<sup>1</sup>.

#### 2.1.1. Echo-Planar Imaging for 4-D Visualization of the fMRI

Echo-planar imaging is a very fast magnetic resonance (MR) imaging technique capable of acquiring an entire MR image in only a fraction of a second [18]. In single-shot echo-planar imaging, all the spatial-encoding data of an image can be obtained after a single radio-frequency excitation [18]. Here, we visualize the EPI for subjects of the Haxby experiment from cuts of frontal, axial and lateral regions in figure 8 in appendix B. EPI visualization provides realistic insights from the activated regions in the brain that plays a crucial role in spatio-temporal brain decoding.

<sup>1</sup><https://nilearn.github.io/>



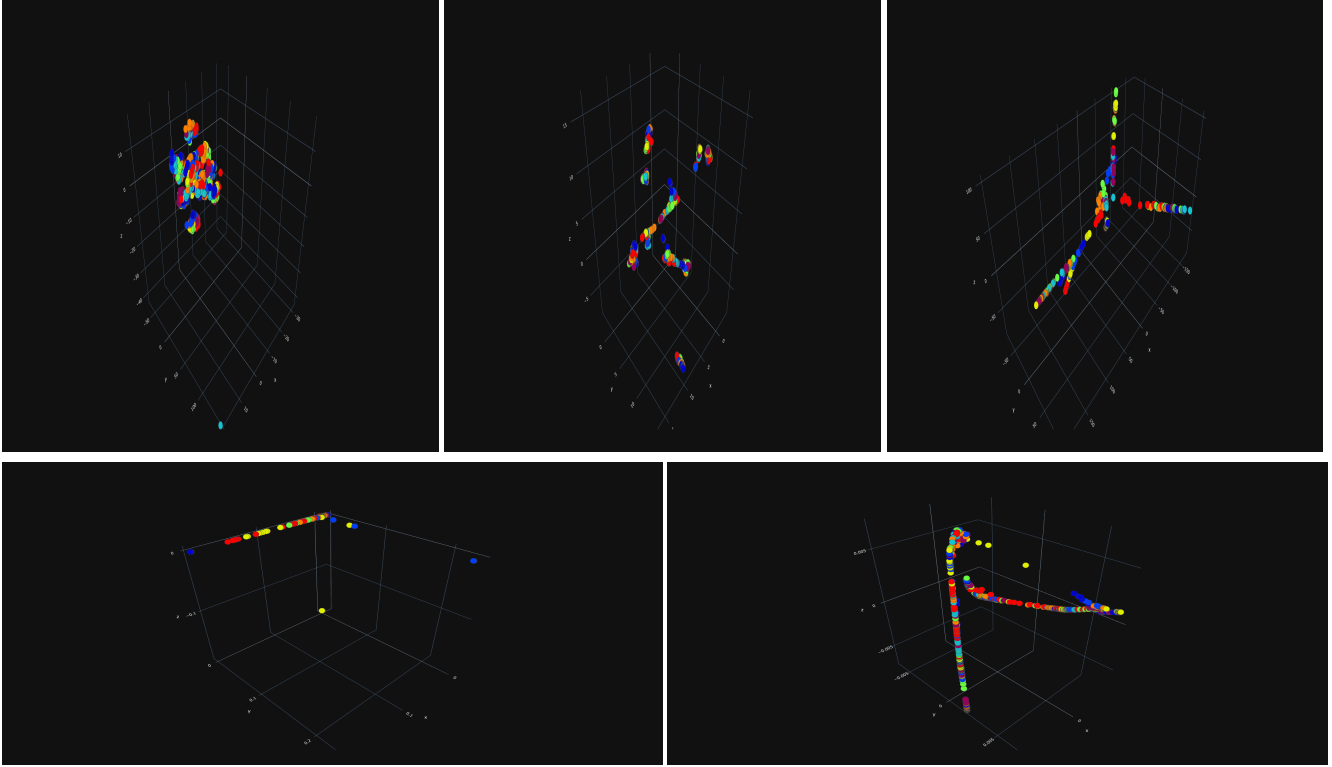


Figure 5: Manifold learning: t-SNE, UMAP, ISOMAP, LLE and Spectral Embedding are performed with 3 component for visualization purposes that are presented in sequential way from top left to bottom right. Unique colors represent unique classes.

### 2.1.2. Region of Interest Analysis

A common way to analyze fMRI data is performing the region of interest (RoI) analysis that involves the extraction of signals from specified areas. The most theoretically agnostic use of ROI analysis is to simply explore the underlying signal behind a whole-brain voxel-wise analysis [17]. After extracting statistically meaningful areas, we can perform severity of correction for multiple statistical tests instead of large number of voxels in brain. Sample RoI visualization is performed, can be found at figure 1. In the top right corner, we visualized the RoI for subject 2 in Haxby dataset. Further, most ML decoders are performed on the RoI's of subjects instead of whole brain medium.

### 2.1.3. Statistical Maps

Statistical Parametric Mapping refers to the construction and assessment of spatially extended statistical processes used to test hypotheses about functional imaging data [5]. Generally, the prior step in statistical fMRI is to create a thresholded statistical map, representing the regions that are active (above a threshold) [17]. Hence, it is useful in examining differences in brain activity recorded during neuroscientific experiments. Here, we performed statistical mapping and visualized it in the figure 1. In the second above left

figure, we visualized the statistical map of subject 3 with threshold value 3.

### 2.1.4. Direct fMRI Visualizations

Simple and compact visualization of fMRI data is quite important topic in the context of neuroimaging since it enables researchers to view cortical brain activity. So, we directly plot the temporally averaged fMRI data of subject 2 to further visualization. It can be found at figure 1 in the second above right part of the figure.

### 2.1.5. Anatomic Visualizations

We visualized the anatomical structure of the fMRI (by default 3 cuts: Frontal, Axial, and Lateral) obtained by the temporally averaged fMRI data of the subject 2 to generate insights before decoding in figure 1 at the second above left part of the figure.

### 2.1.6. Glass Brain

The Glass Brain is a state-of-the-art real-time brain visualization technology that is created on the Unity 3D game-engine and powered by NVIDIA's GPU computing [19]. Its inputs include an individual's brain structure, both tissue and fiber tract architecture, obtained from high-resolution MRI-DTI brain scans. Real-time brain activity and functional interactions among networks are superimposed on the

brain structure using high-density EEG (electroencephalography) [19]. Here, we project frontal, axial, and lateral sides of temporally averaged fMRI data of subject 5 and visualized it in figure 1. The last row of the figure 1 represents the glass brain visualizations.

## 2.2. Functional Connectivity and Similarity Analysis

Functional connectivity is defined as the temporal dependency of neuronal activation patterns of anatomically separated brain regions and in the past years an increasing body of neuroimaging studies has started to explore functional connectivity by measuring the level of co-activation of resting-state fMRI time-series between brain regions [23]. These functional connections are important in establishing statistical connections in brain regions. Functional connectivity can be obtained by estimating a covariance (or correlation) matrix for signals from different brain regions decomposed, for example on resting-state or naturalistic-stimuli datasets. Here, we performed functional connectivity analysis based on the correlation, precision and partial correlation. Then, similarity analysis based on the cosine, minkowski and euclidean distance is performed to further extend statistical findings in masked fMRI data. Please refer to self-explained figures 2 and 3 for depth understanding.

### 2.2.1. Functional Connectivity: Correlation

Functional connectivity based on Pearson correlation is performed on subject 1 and visualized in figure 2 at top left corner. We can see that in the ventral temporal cortex of subject 1, there are correlations when the stimuli of faces are presented.

### 2.2.2. Functional Connectivity: Precision

As shown in the papers [20, 24], it is more interesting to use the inverse covariance matrix, i.e. the precision matrix. It gives only direct connections between regions, as it contains partial covariances, which are covariances between two regions conditioned on all the others. Moreover, we performed functional connectome based on precision score, to extract signals on RoI's of subject 1 and visualized in the figure 2 at top center. Here, with the change in the connectivity measure, we see direct changes in spatial correlations in the ventral cortex of subject 1. With precision measure, we further get understanding in brain organization and brain networks.

### 2.2.3. Functional Connectivity: Partial Correlation

Among the range of network modeling methods, partial correlation has shown great promises in accurately detecting true brain network connections [25]. So, we performed functional connectivity analysis based on partial correlation and visualized it in figure 2 at top right corner. Visualization of partial correlation in RoI fMRI data demonstrate that the ventral temporal cortex of the subject 1 is not much correlated.

### 2.2.4. Similarity Analysis: Cosine Similarity

To facilitate the geodesic understanding in the context of statistical connections in the brain, we performed cosine similarity analysis on subject 1, and the obtained matrix is visualized in figure 3 at the lower left corner. The results demonstrate that there are highly overlapping regions in terms of neural activity when visual stimuli is presented.

### 2.2.5. Similarity Analysis: Minkowski Similarity

To experiment with different similarity metrics, we utilized the minkowski distance that is a generalization of both the Euclidean and the Manhattan distance. Hence, it is useful in fMRI temporal similarity analysis.

### 2.2.6. Similarity Analysis: Euclidean Similarity

Lastly, we performed similarity analysis based on classical euclidean distance. It is a very classical measure of the distance in terms of cartesian coordinates of the points using the Pythagorean theorem [13]. From the statistical and structural patterns exposed by functional connectivity and similarity analysis, we can conclude that the neural activity evoked in the ventral temporal cortex of the human brain is highly overlapping and distributed.

## 2.3. Dimension Reduction to Manifold Learning

Functional MRI data are very high-dimensional if one considers all the voxels or surface coordinates acquired with standard imaging parameters [?]. As in our dataset, with the structure of 4D timeseries image data, we have curve of dimensionality problem. Hence, dimension reduction and manifold learning algorithms can reduce the dimensionality of fMRI space by preserving geodesic relations in the lower representations. We performed PCA, LDA, ICA, NNMF and MDS as dimension reduction algorithms. Besides, t-SNE, UMAP, ISOMAP, LLE and Spectral Embedding are performed to generate lower dimensional manifolds of the fMRI space. Further, we visualized all manifolds in 3-D dimensional space in figure 4 and 5.

### 2.3.1. Dimension Reduction: PCA

PCA is a linear unsupervised dimension reduction algorithm and it computes principal vectors to change the basis of the representation [26]. PCA is used algorithm in broad range of topics from image compression to decorrelation of texts. Here, we performed PCA on RoI's of subject 3, and visualized in figure 4 at left top corner.

### 2.3.2. Dimension Reduction: LDA

LDA is supervised dimensionality reduction algorithm and it is a generalization of Fisher's linear discriminant, aims to find linear subspace that characterize the original data space. Since it is supervised, it is a powerful paradigm in representation learning. Here, we performed LDA on RoI's of subject 3, and visualized in figure 4 at right down corner. From the figures, we can see that LDA is outperforming other methods by uniquely separating geodesic distances in the manifolds.

### 2.3.3. Dimension Reduction: ICA

ICA is a computational approach for separating multi-variate signals into its additive components. It is natural paradigm for unsupervised dimensionality reduction. Here, we performed ICA on RoI's of subject 3, and visualized in figure 4 at right top corner.

### 2.3.4. Dimension Reduction: NMF

NNMF is a iterative non-negative factor analysis to decompose non-negative matrix into its linear subspaces. It is useful in extracting natural linear subspaces of original data samples. Here, we performed NMF on RoI's of subject 3, and visualized in figure 4 at left down corner.

### 2.3.5. Manifold Learning: MDS

MDS is classical approach for extracting non-linear subspaces of the original data space by preserving geodesic distance in the manifold. Lower dimensional embedding is obtained to represent original data in the manifold. Here, we performed MDS on RoI's of subject 3, and visualized in figure 4 at center top.

### 2.3.6. Manifold Learning: t-SNE

T-SNE is iterative statistical approach for producing non-linear embedding of the original data space by preserving small pairwise distances or localized similarities. It minimizes the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. Here, we performed t-SNE on RoI's of subject 3, and visualized in figure 5 at left top corner.

### 2.3.7. Manifold Learning: UMAP

UMAP is a recent approach for non-linear embedding, and it generally outperforms t-SNE by a significant margin. It is very similar to t-SNE but it also preserves the global geodesic structure of the data. Here, we performed UMAP on RoI's of subject 3, and visualized in figure 5 at the center top.

### 2.3.8. Manifold Learning: ISOMAP

ISOMAP map is also non-linear embedding algorithm through isometric mapping for accurately estimating the intrinsic geometry of manifold by preserving geodesic distances in the manifold. Here, we performed ISOMAP on RoI's of subject 3, and visualized in figure 5 at the right top corner.

### 2.3.9. Manifold Learning: LLE

LLE is topology preserving non-linear dimension reduction algorithm, trying to preserve neighbor structure in manifold, and it is generally outperforming ISOMAP in terms of optimization and speed thus it has very practical uses in literature. Here, we performed LLE on RoI's of subject 3, and visualized in figure 5 at right bottom corner.

### 2.3.10. Manifold Learning: Spectral Embedding

Spectral embedding is also non-linear embedding algorithm that forms an affinity matrix and applies spectral decomposition to the laplacian graph. Here, we performed

Spectral embedding on RoI's of subject 3, and visualized in figure 5 at left bottom corner.

## 2.4. Classical Machine Learning to Ensemble Robust Decoding

We applied more than 30 decoding algorithms, here we listed our methods and their brief descriptions.

### 2.4.1. LinearSVC

Support Vector with the linear kernel is an efficient algorithm for linearly separable data spaces by fitting hyper-plane that categorizes the visual stimuli in our context [16].

### 2.4.2. SGD Classifier

SGD classifier is a linear classifier that use stochastic gradient descent with Hinge loss to separate data spaces. We SGD classifier is implemented with l1 regularization [16].

### 2.4.3. MLP

MLP is a simple neural network architecture that consists of a bunch of linear layers with ReLU non-linearity, and optimized by the Stochastic Gradient Descent rule [16].

### 2.4.4. Perceptron

Perceptron is a single layer version of MLP and uses log-loss instead of Cross-Entropy [16].

### 2.4.5. Logistic Regression

Logistic regression is a classical machine learning algorithm, it applies linear transformation on the data followed by sigmoidal activation to generate real valued probabilities [16].

### 2.4.6. Logistic Regression Cross-Validation

We also performed 10-Fold cross-validation with logistic regression. It produces more reliable results compared to conventional logistic regression [16].

### 2.4.7. SVC

Support Vector Classifier is a powerful paradigm in the context of classification. It is classical SVM with radial basis function kernel [16].

### 2.4.8. Calibrated Classifier

It is a cross-validated probability calibration classifier with isotonic logistic regression [16].

### 2.4.9. Passive Aggressive Classifier

Passive Aggressive classifier is margin based online large-scale learning algorithm similar to perceptron but does not require learning rate [4, 16].

### 2.4.10. Label Propagation Classifier

Label Propagation classifier is semi-supervised graph inference algorithm that iterates on the original graph and normalizes the edge weights by graph Laplacian [9, 16].

### 2.4.11. Random Forest Classifier

Random Forest classifier is a meta estimator algorithm that fits parallel decision tree classifiers with bootstrapped samples of the original dataset to improve the predictive accuracy and model reliability [16].

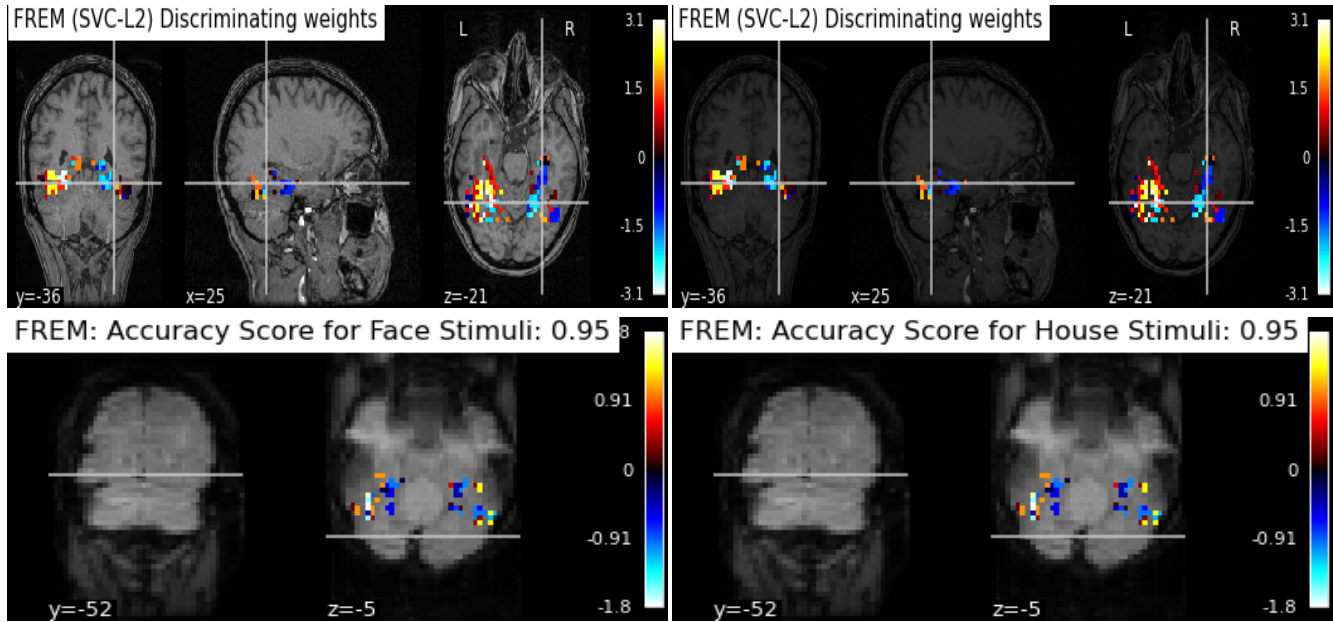


Figure 6: Visualizations of discriminate power of FREM decoder at first row. In second row, the visuals represent the discrimination of neural responses when face and house stimuli is presented.

#### 2.4.12. Gradient Boosting Classifier

Gradient boosting classifier is boosting algorithm that ensembles a weak learnings, generally decision trees. It constructs an additive model in a forward stage-wise fashion that enables model to optimize the parameters on any differentiable loss functions. We utilized 100 weak learners to construct end classifier [16].

#### 2.4.13. Quadratic Discriminant Classifier

Quadratic Discriminant classifier is class conditional density algorithm with quadratic decision boundaries to fit separate Gaussian to each classes. It is a powerful method when we have priory knowledge that individual classes exhibit distinct covariance [16].

#### 2.4.14. Ridge Classifier

Ridge classifier is L2 penalized version of logistic regression that helps robust decoding with lower degrees of freedom [16].

#### 2.4.15. Ridge Classifier Cross-Validation

Ridge classifier cross-validation is more reliable version of Ridge Classifier by estimating its model parameters by 10-Fold cross validation [16].

#### 2.4.16. AdaBoost Classifier

AdaBoost classifier is type of ensemble learning algorithm that fits weak classifier on the dataset then fits additional copies of the classifier with adjusted parameters based on the incorrectly classified objects [16].

#### 2.4.17. Extra Trees Classifier

Extra Trees classifier is another ensemble learning method based on randomized decision trees on the sub-sampled ver-

sion of the datasets to improve predictive quality [16].

#### 2.4.18. K-Neighbors Classifier

K-Neighbors classifier is an instance-based learning method based on  $k$  neighbors to consider. We consider 5 neighbors with minkowski distance metric to vote class predictions [16].

#### 2.4.19. Bernoulli Naive Bayes Classifier

Bernoulli Naive Bayes classifier applies Bayesian rule to dataset with prior assumption that the data comes form multivariate Bernoulli distribution [16].

#### 2.4.20. Gaussian Naive Bayes Classifier

Gaussian Naive Bayes classifier applies Bayesian rule to dataset with prior assumption that the data comes form multivariate Gaussian distribution [16].

#### 2.4.21. Nu-Support Vector Classifier

Nu-Support Vector classifier is type of SVM algorithm. It is very similar to SVC but it differs from its controllability of a number of support vectors [16].

#### 2.4.22. Nearest Centroid Classifier

In Nearest Centroid classifier, each class is represented by with class centroid. New samples are classified based on the distance to class centroid, so it is very similar to the supervised version of K-means [16].

#### 2.4.23. Bagging Classifier

Bagging classifier is an ensemble learning method that fits base classifiers to random subsets of the original dataset. Then, the predictions are aggregated either by voting or averaging to produce end class-wise predictions [16].



Model	Accuracy	Balanced Accuracy	F1 Score	Time Taken (sec)
<b>FREM:LR-L2</b>	0.95	0.96	0.94	-
<b>FREM:SVC-L2</b>	0.94	0.94	0.94	-
CalibratedClassifierCV	0.87	0.88	0.87	2.78
PassiveAggressiveClassifier	0.86	0.86	0.86	1.55
LogisticRegression	0.85	0.85	0.85	0.64
RidgeClassifierCV	0.85	0.85	0.85	2.14
LinearSVC	0.84	0.84	0.84	0.49
Perceptron	0.83	0.83	0.83	0.67
Twin-SVT	0.82	-	-	-
LinearDiscriminantAnalysis	0.81	0.81	0.81	0.23
3D CNN	0.8	-	-	-
SGDClassifier	0.79	0.79	0.79	0.26
NuSVC	0.75	0.761	0.75	0.53
RidgeClassifier	0.74	0.74	0.74	1.23
SVC	0.73	0.74	0.73	0.42
LGBMClassifier	0.71	0.72	0.71	5.62
XGBClassifier	0.70	0.71	0.70	6.32
2D CNN	0.7	-	-	-
RandomForestClassifier	0.69	0.69	0.69	6.60
ExtraTreesClassifier	0.66	0.67	0.66	3.47
KNeighborsClassifier	0.60	0.61	0.60	0.47
BaggingClassifier	0.50	0.50	0.49	0.96
NearestCentroid	0.39	0.40	0.40	0.17
DecisionTreeClassifier	0.37	0.38	0.38	0.07
BernoulliNB	0.36	0.36	0.35	0.08
GaussianNB	0.35	0.35	0.34	0.05
ExtraTreeClassifier	0.30	0.31	0.29	0.04
AdaBoostClassifier	0.24	0.25	0.21	1.41
QuadraticDiscriminantAnalysis	0.15	0.16	0.15	0.17
LabelPropagation	0.14	0.12	0.03	0.08

Table 1: **fMRI Decoding Accuracy Scores.** Decoding winner is FREM algorithm with Logistic Regression. Deep learning based methods could not outperform the ensemble ones since classical ML algorithms performed on the masked ventral temporal cortex region whereas DL algorithms performed in whole brain area of interest. Best results are bolded.

#### 2.4.24. FREM: Cross-Validated Ensemble of L2 regularized SVCs and Logistic Regressions

FREM uses an implicit spatial regularization through fast clustering and aggregates a high number of estimators trained on various splits of the training set, thus returning a very robust decoder at a lower computational cost than other spatially regularized methods [2] that improve decoding accuracy with lower degree of computation. We both construct FREM based on SVCs and Logistic Regression units [2].

#### 2.5. Convolutional Neural Networks to Vision Transformers

We further performed deep learning algorithms from CNNs to vision transformers. All models are optimized with Cross Entropy loss, Adam optimizer with starting learning rate as 1e-3. Scheduled learning rate scaler is per-

formed to decrease the learning rate by % 80 at every epoch. Batch size is varied from 16 to 64. One single Tesla K80 GPU is used for all experiments. Architecture details and their visuals of both CNNs and vision transformer are depicted in appendix C, D and E, respectively. Please refer for further information. All deep learning algorithm are performed in the whole brain area represented as fMRI form instead of RoI's of subjects.

#### 2.6. 2D CNN

We developed 2D CNN architecture to decode fMRI samples. In the architecture, there are 4 special convolutional blocks, each block is consists of a sequel of Convolution, Batch Normalization, ReLU, Max Pooling and Dropout layer. To classify, the representative feature vector is propagated to linear blocks. There are two linear blocks, each block consist of sequel of linear layer, batch normalization, ReLU and dropout layer. We provide details of the archi-

texture in appendix C.

### 2.7. 3D CNN

3D CNN is developed to perform spatio-temporal decoding of fMRI's of whole brain region. The architecture is nearly the same as the 2D case except all layers are 3D, i.e., 2D convolutions are interchanged with 3D's, then Batch Normalization, Max Pooling and Dropout layers are now extended to 3D case. We provide details of the architecture and its visualization in appendix D.

### 2.8. Vision Transformers

Vision transformers are recently proposed paradigm that replaced with CNN blocks with multi-head self attention mechanism. As vision transformer, we performed experiments in twin-svt [3] that is state-of-the-art vision transformer with its unique attention mechanism called spatially separable self-attention (SSSA) module SSSA module consists of spatially oriented both global and local attention mechanism by taking the advantage of interactions between different streams of visual representations. We provide details of the transformer in appendix E. We highly recommend reader to refer to appendix E for a detailed understanding of twin-svt.

## 3. Results

Decoding results are presented in table 1. Winner brain decoder is FREM based methods. FREM is a powerful paradigm in the context of brain decoding since it aggregates similar voxels together to reduce the dimensionality of the RoI of subjects with an ensemble of different methods. It is highly spatially regularized method and combines high number of parallel estimators trained on different sub-spaces of the original data. We visualized discriminate weights of FREMs on fMRI images with corresponding RoIs of subjects in figure 6 at the top row. Further, we visualized the decoding accuracy of FREMs when face and house stimuli is presented figure 6 at bottom row. When visuals of faces and houses are presented to the subjects, FREMs perform better with respect to other stimulus types. Then, Calibrated classifier is placed as third among decoders by achieving % 87 accuracy. Then, Passive Aggressive classifier and Logistic Regression are also performed quite well in brain decoding. In terms of deep learning methods, twin transformed outperforms both 2D and 3D CNNs by a considerable margin thanks to its spatially oriented self-separable attention mechanism. Another big plus of the transformer is that it is far away explainable than any DL based approaches that will quite an important step in further discovery of attention mechanism, especially covert attention, in human brains. It may demonstrate interactions of important regions of spatial connections of human brain that play a significant role in spatio-temporal decoding.

## 4. Discussion

Our experiments are motivated by addressing the issue of whether patterns of neural responses recorded by fMRI

when visuals stimuli is presented are discriminate or not. This paper addresses the advances of understanding in distributed and overlapping patterns of neural activity to infer the functional role of brain areas and networks. To expose that, we performed functional connectivity and similarity analysis, and concluded that responses evoked in RoIs of the ventral temporal cortex are highly correlated with all measures except for partial correlation. Partial correlation is relatively less in that regions. Then, we performed a bunch of dimension reduction and manifold learning algorithm to visualize the underlying geodesic manifolds of the regions. All methods except for LDA demonstrate that there are highly overlapping and distributed representations in that areas. However, LDA show that even there are highly correlated and overlapping responses, we can discriminate them by powerful decoding algorithms. Further, we run more than 30 different fMRI decoding algorithm on the ventral temporal masked regions, and conclude that state-of-the-art robust decoding algorithm FREM with either SVCs and Logistic Regression, outperforms all both ML and DL based methods as it achieves more than % 95 accuracy. The reason behind the deep learning based methods, i.e., 2D-3D CNNs and vision transformers, could not outperform the ML based ones is that ML decoders are performed on the ventral temporal masked regions of the subjects whereas DL decoders are performed in whole brain areas. However, the vision transformer outperformed other DL decoders and demonstrate the power of spatially oriented attention mechanism that mimics the human attention mechanism in deeper way. To our best knowledge, we either achieved either competitive accuracy on current state-of-the-art or outperformed by a small margin [22, 14]. We further demonstrate that with the spatially oriented vision transformers that expose both global and local attention in distributed regions in the brain, we explain and enlight the covert attention mechanisms in the human brain by its encoded neural response, as it achieves % 82 accuracy even the algorithm scans the whole brain region. Further research along with the decoding of brain by spatially oriented vision transformers can expose the underlying reasons regarding the why, when and how human attention mechanism is oriented in a computational manner. We further conclude that multi-voxel pattern recognition methods with cutting-edge techniques, e.g., vision transformers, may advance our understanding of neural information processing and neural coding from visual perception to memory search. Code is available at appendix F.

## References

- [1] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.
- [2] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt,

- and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.
- [3] X. Chu, Z. Tian, Y. Wang, B. Zhang, H. Ren, X. Wei, H. Xia, and C. Shen. Twins: Revisiting the design of spatial attention in vision transformers, 2021.
- [4] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive aggressive algorithms. 2006.
- [5] K. J. Friston. Statistical parametric mapping. 1994.
- [6] C. G. Gross, C. d. Rocha-Miranda, and D. Bender. Visual properties of neurons in inferotemporal cortex of the macaque. *Journal of neurophysiology*, 35(1):96–111, 1972.
- [7] S. J. Hanson, T. Matsuka, and J. V. Haxby. Combinatorial codes in ventral temporal lobe for object recognition.
- [8] J. Haxby, M. Gobbini, M. Furey, A. Ishai, J. Schouten, and P. Pietrini. "visual object recognition", 2018.
- [9] R. A. Heckemann, J. V. Hajnal, P. Aljabar, D. Rueckert, and A. Hammers. Automatic anatomical brain mri segmentation combining label propagation and decision fusion. *NeuroImage*, 33(1):115–126, 2006.
- [10] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2020.
- [11] S. Huang, W. Shao, M.-L. Wang, and D.-Q. Zhang. fmri-based decoding of visual information from human brain activity: A brief review. *International Journal of Automation and Computing*, pages 1–15, 2021.
- [12] R. Koster, M. J. Chadwick, Y. Chen, D. Berron, A. Banino, E. Düzel, D. Hassabis, and D. Kumaran. Big-loop recurrence within the hippocampal system supports integration of information across episodes. *Neuron*, 99(6):1342–1354, 2018.
- [13] E. Maor. *The Pythagorean theorem: a 4,000-year history*. Princeton University Press, 2019.
- [14] K. A. Norman, S. M. Polyn, G. J. Detre, and J. V. Haxby. Beyond mind-reading: multi-voxel pattern analysis of fmri data. *Trends in cognitive sciences*, 10(9):424–430, 2006.
- [15] A. J. O’toole, F. Jiang, H. Abdi, and J. V. Haxby. Partially distributed representations of objects and faces in ventral temporal cortex. *Journal of cognitive neuroscience*, 17(4):580–590, 2005.
- [16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [17] R. A. Poldrack. Region of interest analysis for fmri. *Social cognitive and affective neuroscience*, 2(1):67–70, 2007.
- [18] M. Poustchi-Amin, S. A. Mirowski, J. J. Brown, R. C. McKinstry, and T. Li. Principles and applications of echo-planar imaging: a review for the general radiologist. *Radiographics*, 21(3):767–779, 2001.
- [19] R. P. Reddy, A. R. Mathulla, and J. Rajeswaran. A pilot study of perspective taking and emotional contagion in mental health professionals: Glass brain view of empathy. *Indian Journal of Psychological Medicine*, page 0253717620973380, 2021.
- [20] S. M. Smith, K. L. Miller, G. Salimi-Khorshidi, M. Webster, C. F. Beckmann, T. E. Nichols, J. D. Ramsey, and M. W. Woolrich. Network modelling methods for fmri. *Neuroimage*, 54(2):875–891, 2011.
- [21] K. Tanaka. Inferotemporal cortex and object vision. *Annual review of neuroscience*, 19(1):109–139, 1996.
- [22] M. S. Treder. Mypa-light: a classification and regression toolbox for multi-dimensional data. *Frontiers in Neuroscience*, 14:289, 2020.
- [23] M. P. Van Den Heuvel and H. E. H. Pol. Exploring the brain network: a review on resting-state fmri functional connectivity. *European neuropsychopharmacology*, 20(8):519–534, 2010.
- [24] G. Varoquaux, A. Gramfort, J. B. Poline, and B. Thirion. Brain covariance selection: better individual functional connectivity models using population prior. *arXiv preprint arXiv:1008.5071*, 2010.
- [25] Y. Wang, J. Kang, P. B. Kemmer, and Y. Guo. An efficient and reliable statistical method for estimating functional connectivity in large scale brain networks using partial correlation. *Frontiers in neuroscience*, 10:123, 2016.
- [26] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

## A. Samples of Visual Stimuli

Example of stimuli are presented in figure 7.

## B. Echo-Planar Imaging for 4D Visualization of the fMRI

Echo-planar, temporally averaged fMRI visualization is presented at figure 8

## C. 2D CNN Details

Detailed architecture of 2D CNN is depicted in figure 9.

## D. 3D CNN Details

Detailed architecture of 3D CNN is depicted in figure 10.

## E. Twin Transformer Details

Recently, twins are proposed in the paper [3], and demonstrate that the spatially oriented vision transformers can outperform the classical CNNs [3]. Here, we integrated Twins-SVT network to our case to produce high quality decoding. There twin transformer is based on a spatially separable self-attention (SSSA) network that consist of locally-grouped self-attention (LSA) and global sub-sampled attention (GSA) [3]. Thanks to its spatially separable module, the quality of features are increased by a significant margin. In the subsections, we describe the SSSA module in detail.

### E.1. Locally-grouped self-attention (LSA)

In LSA, 2-D feature maps are divided into sub-windows that enable the self-attention within each sub-window. Features maps are divided into  $m \times n$  sub-windows, that lead to each window consist of  $\frac{HW}{mn}$  elements where H,W represents image dimensions. By dividing the image into  $m \times n$  region the computational cost is decreased from

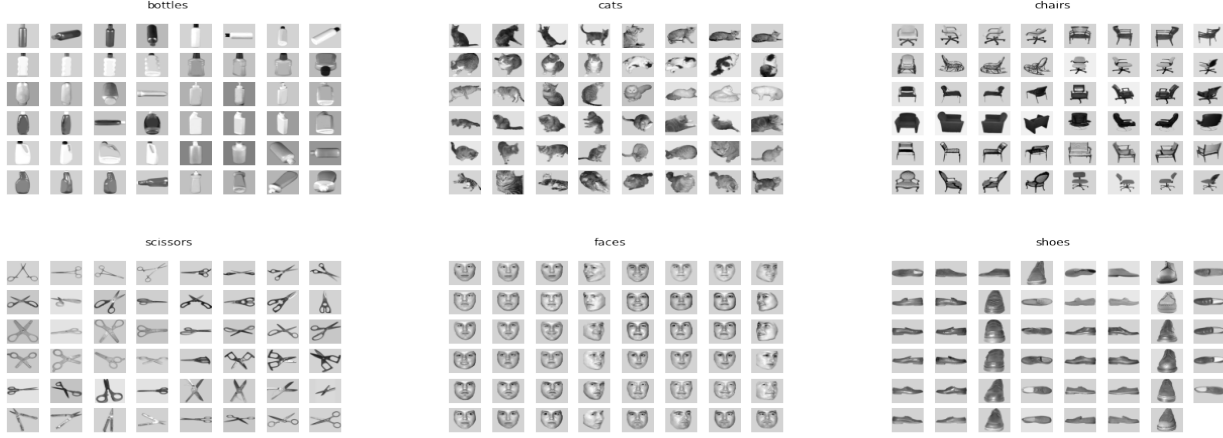


Figure 7: Examples of stimuli visuals. They are gray scale images from different kind of nature.

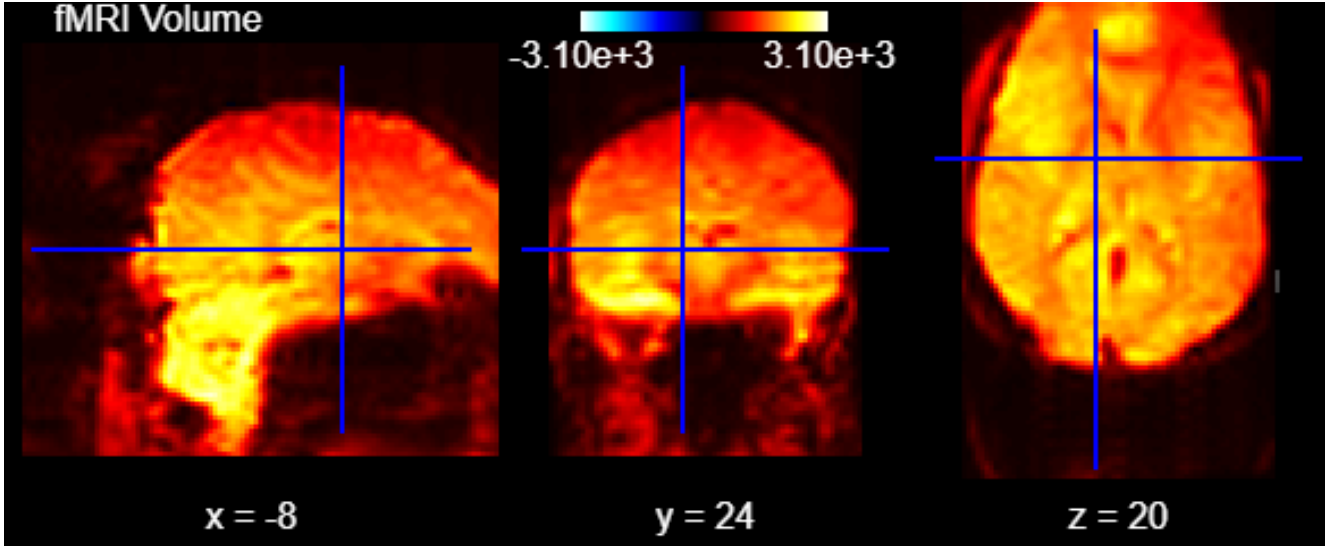


Figure 8: Echo-Planar Imaging for 4D Visualization of the fMRI on subject, averaged for 4D

$O(H^2W^2d)$  to  $O(\frac{H^2W^2}{mn}d)$  where  $d$  is the self-attention dimension. At that point, we did not make any further relation to non-overlapping regions in the windows. Hence, here GSA module comes into play.

## E.2. Global sub-sampled attention (GSA)

As we need further localization in the self-attention mechanism, global self-attention is required to make connections in non-overlapping regions. In GSA module, a single representative key in formations from the locally attended windows are used to compute global attention. However, with the computation of global-attention, the computation cost would increase to  $O(H^2W^2d)$ . To prevent this, locally attended features are sub-sampled via average pooling, depth-wise strided convolutions and regular strided convolutions.

The results show that regular strided convolutions perform best [3]. Mathematically, SSSA module performs the following computations.

$$\begin{aligned} a_{i,j}^l &= LSA(LayerNorm(a_{i,j}^{l-1})) + a_{i,j}^{l-1}, \\ a_{i,j}^l &= FFN(LayerNorm(a_{i,j}^l)) + a_{i,j}^l, \\ a^{l+1} &= GSA(LayerNorm(a^l)) + a^l, \\ a^{l+1} &= FFN(LayerNorm(a^{l+1})) + a^{l+1} \end{aligned} \quad (1)$$

for  $i = 1, \dots, m$  and  $j = 1, \dots, n$  where LSA denotes locally-grouped self-attention, GSA denotes global sub-sampled attention, FFN denotes feed-forward network and LayerNorm denotes layer normalization layer [1].

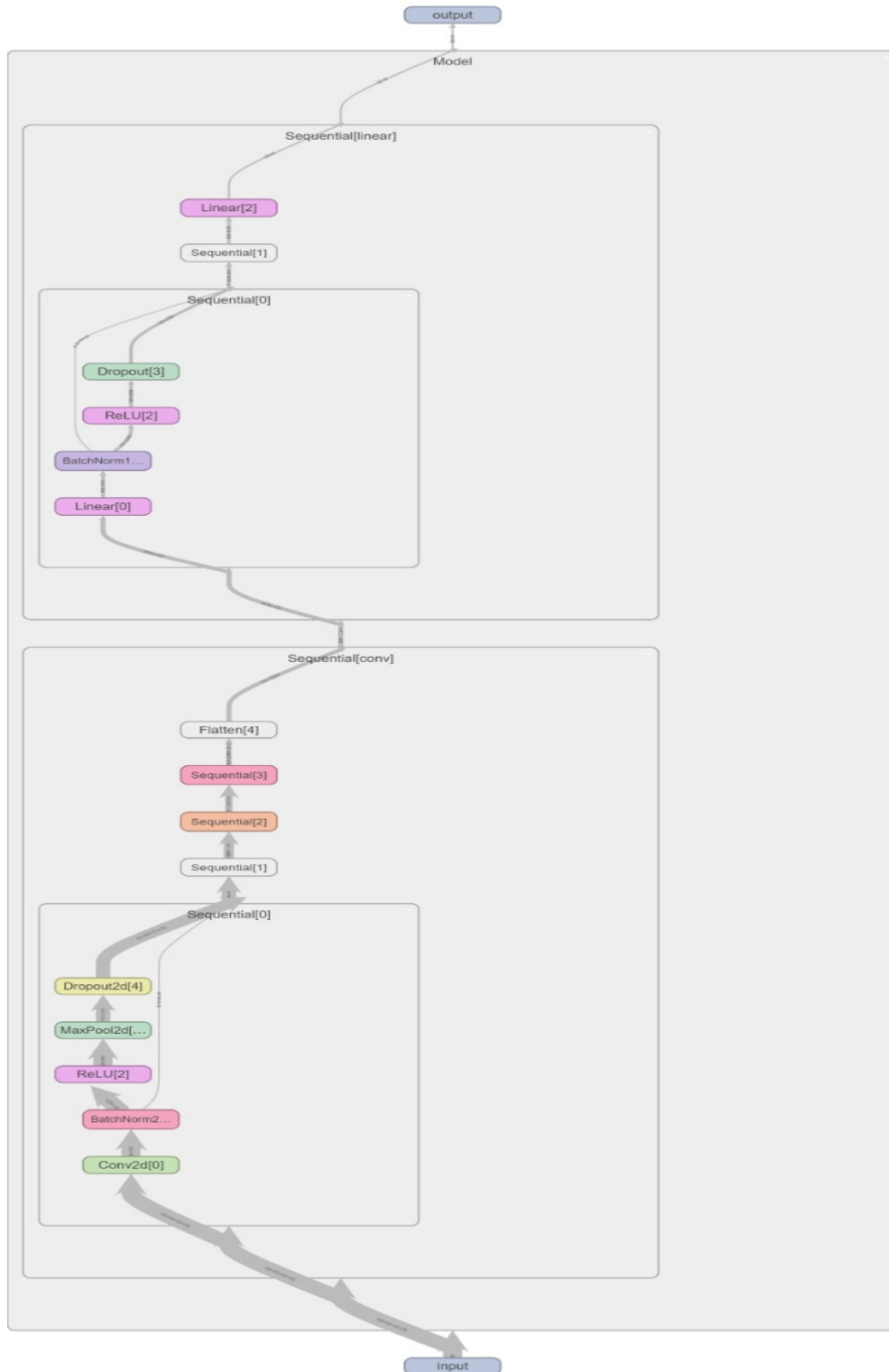


Figure 9: **Architecture of 2D CNN.** In the architecture, there are 4 special convolutional blocks, each block is consist of sequel of Convolution, Batch Normalization, ReLU, Max Pooling and dropout layer. To classify, the representative feature vector is propagated to linear blocks. There are two linear blocks, each block consist of sequel of linear layer, batch normalization, ReLU and dropout layer.



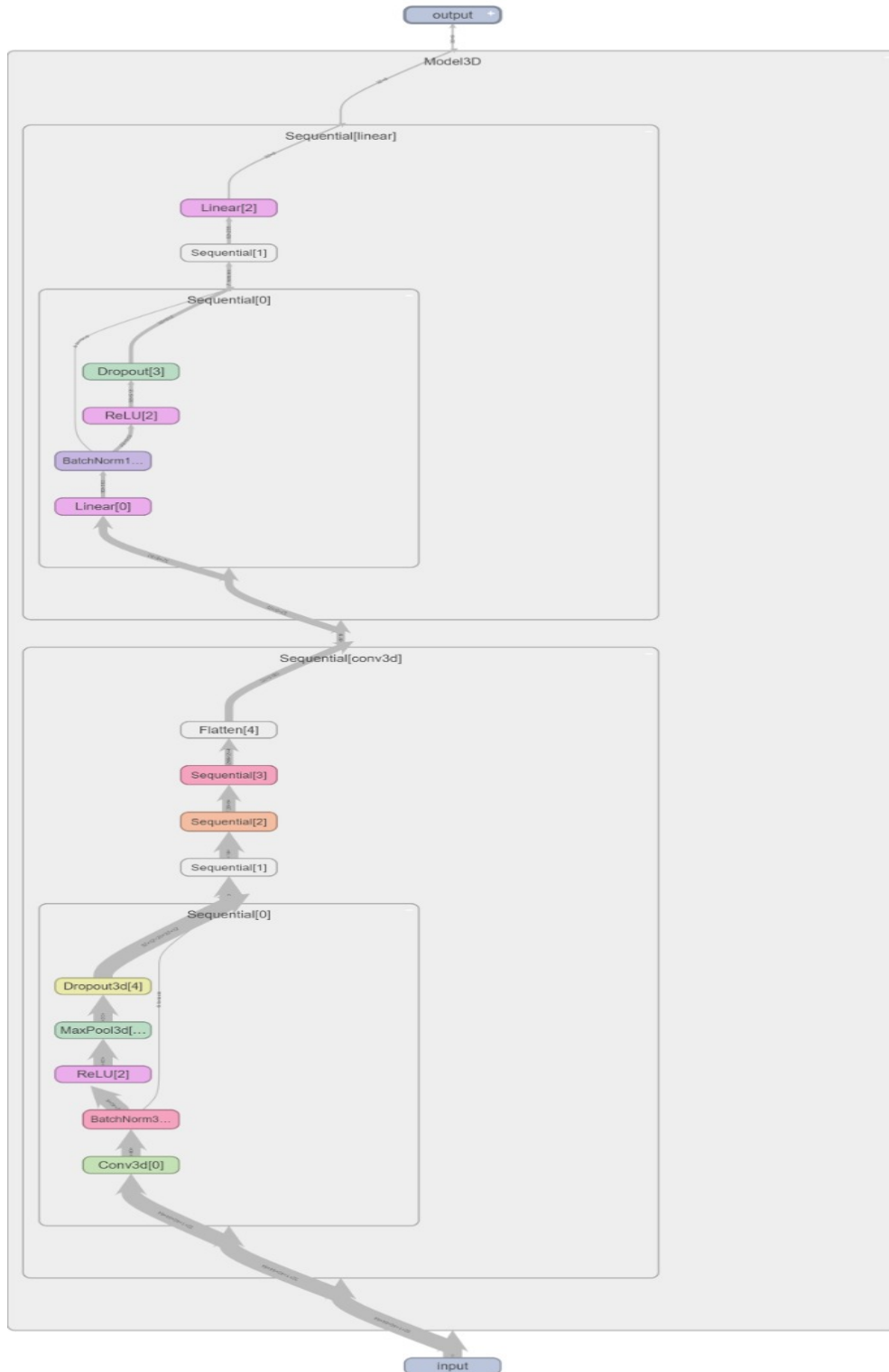


Figure 10: **Architecture of 3D CNN**. In the architecture, there are 4 special convolutional blocks, each block is consist of sequel of Convolution, Batch Normalization, ReLU, Max Pooling and dropout layer in 3D. To classify, the representative feature vector is propagated to linear blocks. There are two linear blocks, each block consist of sequel of linear layer, batch normalization, ReLU and dropout layer in 3D.

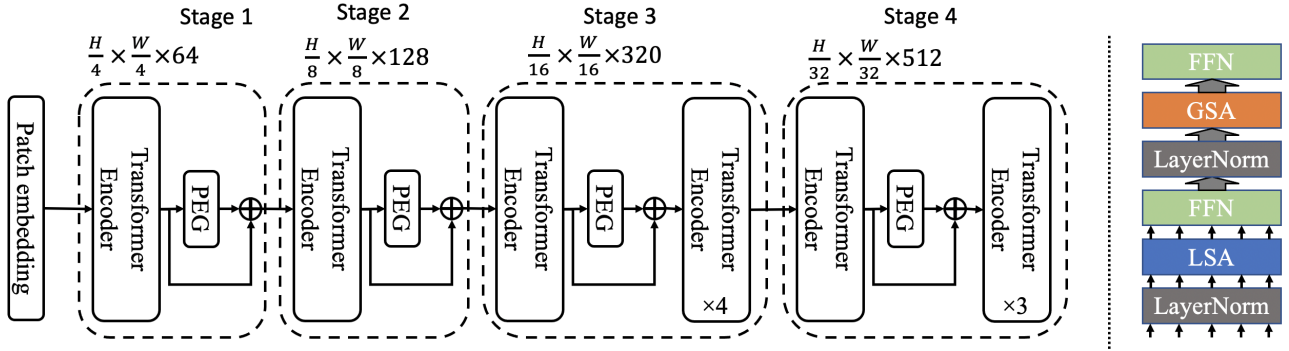


Figure 11: **Architecture of Twin Transformer.** Given the input, Twins-SVT interleaves locally-grouped attention (LSA) and global sub-sampled attention (GSA) in hierarchical stages. PEG stands for position encoding generator.

## F. Code

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # - Computational Neuroscience 2021-2022 Final Project -
5
6 # ## Project Name: Combinatorial Codes in Ventral Temporal Lobe for Visual Object Recognition
7 #
8 #
9
10 # Filename : CompNeuro_2021-2022_Final_Project.ipynb
11 #
12 # Authors : Can Kocagil, Emirhan Ilhan and Arman Vural Budunoglu,
13 #
14 # Institution : Bilkent University Department of Electric & Electronical Engineering
15 #
16 # Class : EEE482/582 - Computational Neuroscience
17 #
18 # Project Goal : Implement multi-voxel pattern analyses methods (based on some type of classifier) to
19 # decode the category of visual stimuli viewed by a human subject based on their
20 # recorded brain activity
21 #
22 # Dataset Link : https://openfmri.org/dataset/ds000105.
23 #
24 # Related Papers : Distributed and overlapping representations of faces and objects in ventral temporal
25 # cortex
26 #
27 # Pipeline:
28 #
29 # 1) Necessary Installations (If necessary)
30 # 2) Imports
31 # 3) Visual Stimuli and Category Loading
32 # 4) Visual Stimuli Transformations
33 # 5) Explanatory Visual Stimuli Analysis
34 #
35 # * PCA
36 # * T-Stochastic Neighbor Embedding (t-SNE)
37 # * Linear Discriminate Analysis
38 # * Uniform Manifold Approximation and Projection (UMAP)
39 # * Independent Component Analysis (ICA)
40 # * Non-Negative Matrix Factorization
41 # * Masking
42 #
43 # 6) Visual Stimuli Similarity Analysis
44 #
45 # * Euclidean Similarity
46 # * Cosine Similarity
47 # * Pearson Correlation
48 #
49 # 7) Classical ML Algorithms:
50 #
51 # * LinearSVC
52 # * SGDClassifier
53 # * MLPClassifier
54 # * Perceptron
55 # * LogisticRegression
56 # * LogisticRegressionCV
57 # * SVC
58 # * CalibratedClassifierCV
59 # * PassiveAggressiveClassifier
60 # * LabelPropagation
61 # * LabelSpreading
62 # * RandomForestClassifier
63 # * GradientBoostingClassifier
64 # * QuadraticDiscriminantAnalysis
65 # * RidgeClassifierCV
```

```

64 #         * RidgeClassifier
65 #         * AdaBoostClassifier
66 #         * ExtraTreesClassifier
67 #         * KNeighborsClassifier
68 #         * BaggingClassifier
69 #         * BernoulliNB
70 #         * LinearDiscriminantAnalysis
71 #         * GaussianNB
72 #         * NuSVC
73 #         * DecisionTreeClassifier
74 #         * NearestCentroid
75 #         * ExtraTreeClassifier
76 #         * CheckingClassifier
77 #         * DummyClassifier
78 #
79 #     7) Reported Metrics
80 #         * Accuracy
81 #         * Balanced Accuracy
82 #         * ROC AUC
83 #         * F1-Score
84 #         * Time Taken
85 #
86 #     8) Deep Learning Algorithms
87 #         * 3-D Convolutional Neural Networks
88 #         * Visual Transformers
89 #         * ...
90 #
91 #     9) Results Interpretation
92 #
93 #
94 #
95 #
96 #
97
98 # # Necessary Installations (If necessary)
99
100 # In[1]:
101
102
103 get_ipython().system('pip install umap')
104 get_ipython().system('pip install pipreqs')
105 get_ipython().system('pip install lazypredict')
106 get_ipython().system('pip install nibabel')
107 get_ipython().system('pip install nilearn')
108 get_ipython().system('pip install -U kaleido')
109
110
111 try:
112     import sklearn
113     print('Scikit-learn is available, version', sklearn.__version__)
114
115 except:
116     get_ipython().system('pip install scikit-learn')
117
118
119 try:
120     import cv2
121     print('Open-CV is available, version', cv2.__version__)
122
123 except:
124     get_ipython().system('pip install opencv-python')
125
126
127 try:
128     import seaborn
129     print('Seaborn is available, version', seaborn.__version__)
130

```

```

131 except:
132     get_ipython().system('pip install seaborn')
133
134
135 # # Imports
136
137 # In[1]:
138
139
140 from __future__ import print_function, division
141
142 # Basics:
143 import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
144 import os, random, time, sys, copy, math, pickle
145
146 # interactive mode
147 plt.ion()
148
149 # Ignore warnings
150 import warnings
151 warnings.filterwarnings("ignore")
152
153 # For plotting
154 import plotly.io as plt_io
155 import plotly.graph_objects as go
156 get_ipython().run_line_magic('matplotlib', 'inline')
157
158 # Dimension Reduction Algorithms:
159 from sklearn.decomposition import PCA
160 from sklearn.manifold import TSNE
161 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
162 from sklearn.decomposition import FastICA
163 from sklearn.decomposition import NMF
164 import umap
165
166 # Transformations
167 from sklearn.preprocessing import StandardScaler
168 from sklearn.preprocessing import MinMaxScaler
169
170 # Metrics:
171 from sklearn.metrics import classification_report
172
173 # Train-Test Splitter:
174 from sklearn.model_selection import train_test_split
175
176 # For Classical ML algorithms:
177 from lazypredict.Supervised import LazyClassifier
178
179 # Utilies:
180 from tqdm import tqdm
181
182 # For distance measurements:
183 from scipy.spatial.distance import cdist
184
185 # Extras:
186 from abc import abstractmethod
187 from typing import Callable, Iterable, List, Tuple
188
189 # Set true for Google Colab:
190 COLAB = False
191
192 if COLAB:
193     # To access Google Drive:
194     from google.colab import drive
195     drive.mount("/content/gdrive")
196
197

```



```

198 # For neuroimaging:
199 from nibabel.testing import data_path
200 from nilearn import plotting as nplt
201 from nilearn.input_data import NiftiMasker
202 from nilearn import datasets
203 from nilearn import plotting
204 from nilearn.image import mean_img
205 from nilearn.image import index_img
206 import nibabel as nib
207 from nilearn import image
208
209
210
211 print("NumPy Version: ", np.__version__)
212
213
214 root_dir = os.getcwd()
215 image_results_dir = os.path.join(root_dir, 'images')
216 results_dir = os.path.join(root_dir, 'results')
217
218 print('Working Directory: \n ', root_dir)
219
220
221 # Creating requirements.txt file
222 get_ipython().system('pip3 freeze > requirements.txt ')
223
224
225 # # Utilities
226
227 # In[2]:
228
229
230 from utils.timers import timeit
231 from utils.metrics import accuracy, confusion_matrix, visualize_confusion_matrix
232 from utils.savers import save, save_obj, load, load_obj
233 from utils.reproduce import random_seed
234 from dataset.fetch_data_matrix import fetch_from_haxby
235 from visualizer.plot2D import plot_2d
236 from visualizer.plot3D import plot_3d
237
238
239
240
241 # In[3]:
242
243
244 # There are 6 number of subjects in the experiment:
245 haxby_dataset = datasets.fetch_haxby(subjects= [1,2,3,4,5,6])
246
247
248 # In[4]:
249
250
251 haxby_dataset
252
253
254 # In[5]:
255
256
257 num_subjects = 6
258
259 for subject in range(num_subjects):
260
261     # 'func' is a list of filenames: one for each subject
262     fmri_filename = haxby_dataset.func[subject]
263
264     # print basic information on the dataset

```

```

265     print('First subject functional nifti images (4D) are at: %s' %
266           fmri_filename) # 4D data
267
268
269 # # Explanatory Visual Stimuli Analysis
270
271 # ## Echo-planar imaging (EPI) Averaging for 4-D Visualization of the fMRI Nifti Image
272
273 # In[7]:
274
275
276 explanatory_fMRI_dir = os.path.join(image_results_dir, 'explanatory')
277
278 # cut in x-direction
279 sagittal = -25
280 # cut in y-direction
281 coronal = -37
282 # cut in z-direction
283 axial = -6
284
285 # coordinates displaying should be prepared as a list
286 cut_coords = [sagittal, coronal, axial]
287
288
289 # Echo-planar imaging (EPI) Averaged for 4-D
290 epi_image = mean_img(fmri_filename)
291
292 plotting.view_img(epi_image,
293                  threshold=None,
294                  title = 'fMRI Volume',
295                  output_file = os.path.join(explanatory_fMRI_dir + 'fMRI_volume.png'),
296                  )
297
298
299 # ## A mask of the Ventral Temporal (VT) cortex with egion of Interest (RoI)
300
301 # In[14]:
302
303
304 from visualizer.roi import RoI_visualizer
305
306
307 # In[15]:
308
309
310 RoI_visualizer(haxby_dataset, subject_id=2)
311
312
313 # ## Statistical Maps
314
315 # In[16]:
316
317
318 subject_id = 3
319
320 plotting.plot_stat_map(mean_img(fmri_filename),
321                       threshold=3,
322                       figure=plt.figure(figsize=(12,4)),
323                       title=f'Statistical Map of fMRI images of subject {subject_id}',
324                       #output_file = os.path.join(explanatory_fMRI_dir, 'stats_map.png')
325                       )
326 plt.show()
327
328
329 # ## Simple, Compact, fMRI Visualizations
330
331 # In[27]:

```

```

332
333
334 plotting.plot_img(mean_img(fmri_filename),
335                   cut_coords=None,
336                   #output_file= os.path.join(explanatory_fmri_dir, 'fMRI.png'),
337                   display_mode='ortho',
338                   figure=plt.figure(figsize = (12,4)),
339                   axes=None,
340                   title='Visualization of fMRI Data',
341                   threshold=3,
342                   annotate=True,
343                   draw_cross=True,
344                   black_bg=False,
345                   colorbar=False)
346 plt.show()
347
348
349 # ## EPI Plotting
350
351 # In[28]:
352
353
354 plotting.plot_epi(mean_img(fmri_filename),
355                  title='Smoothed mean EPI',
356                  cut_coords=cut_coords,
357                  #output_file= os.path.join(explanatory_fmri_dir, 'epi.png')
358                  )
359
360
361 # ## Anatomic fMRI Visualizations
362
363 # In[31]:
364
365
366 plotting.plot_anat(haxby_dataset.anat[0],
367                  cut_coords=cut_coords,
368                  #output_file= os.path.join(explanatory_fmri_dir, 'anat.png'),
369                  display_mode='ortho',
370                  figure=plt.figure(figsize = (12,4)),
371                  axes=None,
372                  title='Visualization of fMRI Data',
373                  threshold=None,
374                  annotate=True,
375                  draw_cross=True,
376                  black_bg=False,
377                  colorbar=False)
378 plotting.show()
379
380
381 # In[ ]:
382
383
384 plotting.plot_anat(mean_img(fmri_filename),
385                  cut_coords=None,
386                  output_file=None,
387                  display_mode='ortho',
388                  figure=plt.figure(figsize = (12,4)),
389                  axes=None,
390                  title='Visualization of fMRI Data',
391                  threshold=None,
392                  annotate=True,
393                  draw_cross=True,
394                  black_bg=False,
395                  colorbar=False)
396 plotting.show()
397
398

```

```

399 # ## Plot Haxby masks
400
401 # In[33]:
402
403
404 # Build the mean image because we have no anatomic data
405
406
407 func_filename = haxby_dataset.func[0]
408 _mean_img = image.mean_img(func_filename)
409
410 z_slice = -14
411
412 fig = plt.figure(figsize=(4, 5.4), facecolor='k')
413
414 from nilearn.plotting import plot_anat, show
415
416 display = plot_anat(_mean_img, display_mode='z', cut_coords=[z_slice],
417                     figure=fig)
418
419 mask_vt_filename = haxby_dataset.mask_vt[0]
420 mask_house_filename = haxby_dataset.mask_house[0]
421 mask_face_filename = haxby_dataset.mask_face[0]
422
423 display.add_contours(mask_vt_filename,
424                     contours=1,
425                     antialiased=False,
426                     linewidths=4.,
427                     levels=[0],
428                     colors=['red'])
429 display.add_contours(mask_house_filename,
430                     contours=1,
431                     antialiased=False,
432                     linewidths=4.,
433                     levels=[0],
434                     colors=['blue'])
435 display.add_contours(mask_face_filename,
436                     contours=1,
437                     antialiased=False,
438                     linewidths=4.,
439                     levels=[0],
440                     colors=['limegreen'])
441
442 # We generate a legend using the trick described on
443 # http://matplotlib.sourceforge.net/users/legend\_guide.html#using-proxy-artist
444 from matplotlib.patches import Rectangle
445 p_v = Rectangle((0, 0), 1, 1, fc="red")
446 p_h = Rectangle((0, 0), 1, 1, fc="blue")
447 p_f = Rectangle((0, 0), 1, 1, fc="limegreen")
448 plt.legend([p_v, p_h, p_f], ["vt", "house", "face"])
449
450 plt.show()
451
452
453
454 #display.savefig(os.path.join(explanatory_fMRI_dir, 'pretty_brain_response.png'))
455
456
457 # ## Glass Brain Plotting
458
459 # In[35]:
460
461
462 plotting.plot_glass_brain(mean_img(fmri_filename),
463                           threshold=3,
464                           output_file= os.path.join(explanatory_fMRI_dir, 'glass_brain_white.png')
465                           )

```

```

466 plotting.show()
467
468
469 # In[37]:
470
471
472 plotting.plot_glass_brain(
473     mean_img(fmri_filename),
474     black_bg=True,
475     display_mode='xz',
476     threshold=None,
477     #output_file= os.path.join(explanatory_fMRI_dir, 'glass_brain_black.png')
478 )
479
480 plotting.show()
481
482
483 # ## Stimuli Visualizations
484
485 # In[38]:
486
487
488 haxby_dataset_stimuli = datasets.fetch_haxby(subjects=[], fetch_stimuli=True)
489 stimulus_information = haxby_dataset_stimuli.stimuli
490
491 for stimulus_type in [*stimulus_information]:
492
493     if stimulus_type != 'controls':
494
495         img_paths = stimulus_information[stimulus_type]
496
497         fig, axes = plt.subplots(6, 8)
498         fig.suptitle(stimulus_type)
499
500         for img_path, ax in zip(img_paths, axes.ravel()):
501             image = plt.imread(img_path)
502             ax.imshow(image, cmap='gray')
503
504         for ax in axes.ravel():
505             ax.axis("off")
506
507
508         #fig.savefig(os.path.join(explanatory_fMRI_dir, f'{stimulus_type}.png'))
509
510 plt.show()
511
512
513
514
515 # # Interactive Brain Visualizations
516
517 # ## 3D Plots of statistical maps on the cortical surface
518
519 # In[ ]:
520
521
522 plotting.view_img_on_surf(mean_img(fmri_filename), threshold='90%', surf_mesh='fsaverage')
523
524
525 # In[ ]:
526
527
528 plotting.view_img_on_surf(mean_img(fmri_filename), threshold='70%', surf_mesh='fsaverage')
529
530
531 # ## Brain Marking
532

```



```

533 # In[ ]:
534
535
536 plotting.view_markers(
537     [(0, -52, 18), (-46, -68, 32), (46, -68, 32), (1, 50, -5)],
538     ['red', 'cyan', 'magenta', 'orange'],
539     marker_size=10)
540
541
542 # ## Decoding Label Analysis and Masking
543
544 # In[42]:
545
546
547 # Load behavioral information
548 behavioral = pd.read_csv(haxby_dataset.session_target[0], delimiter=' ')
549 behavioral.head()
550
551
552 # In[43]:
553
554
555 # Visual Stimuli Categories:
556 for stimuli in np.unique(behavioral['labels']).tolist():
557     print(stimuli)
558
559
560 # In[6]:
561
562
563 stimuli_categories = [
564     'scissors',
565     'face',
566     'cat',
567     'scrambledpix',
568     'bottle',
569     'chair',
570     'shoe',
571     'house'
572 ]
573
574
575 # ### Masking Spatio Temporal Code and Its Target
576
577 # In[45]:
578
579
580 # Creating conditional categories:
581 conditions = behavioral['labels']
582
583 # We ignore rest condition:
584 condition_mask = conditions.isin(stimuli_categories).tolist()
585
586
587 fmri_niimgs = index_img(fmri_filename, condition_mask)
588
589 conditions = conditions[condition_mask]
590
591 # Convert to numpy array
592 conditions = conditions.values
593 print(conditions.shape)
594
595
596 # Spatio-temporal Masked data shape: (temporal dimension, spatial dimension 1, spatial dimension 2, #
    of experiments)
597
598 # In[48]:

```

```

599
600
601 # (temporal dimension, spatial dimension 1, spatial dimension 2, # of experiments)
602 fmri_niimgs.get_data().shape
603
604
605 # Spatio-temporal Un-masked data shape: (temporal dimension, spatial dimension 1, spatial dimension 2,
606     # of experiments)
607
608 # In[50]:
609
610 spatio_temporal_data = fetch_from_haxby(haxby_dataset.func[subject_id])
611 spatio_temporal_data.shape
612
613
614 # In[51]:
615
616
617 for subject_id in range(num_subjects):
618     label = pd.read_csv(haxby_dataset.session_target[subject_id], delimiter=' ')
619
620     # Creating conditional categories:
621     conditions = behavioral['labels']
622
623     condition_mask = conditions.isin(stimuli_categories).tolist()
624     conditions = conditions[condition_mask]
625
626     # Convert to numpy array
627     conditions = conditions.values
628     print(conditions.shape)
629
630
631 # # Creatining fMRI Data Matrices for each Subject
632
633 # In[7]:
634
635
636 # Creating stimuli to category and category to stimuli:
637 stimuli2category = {
638     'scissors'      : 0,
639     'face'          : 1,
640     'cat'           : 2,
641     'scrambledpix' : 3,
642     'bottle'        : 4,
643     'chair'         : 5,
644     'shoe'          : 6,
645     'house'         : 7
646 }
647
648 category2stimuli = {category:stimuli for stimuli, category in stimuli2category.items()}
649
650
651 # ## Spatio-Temporal Masking
652
653 # In[8]:
654
655
656 def fetch_haxby_per_subject(subject_id:int = None,standardize:bool = True) -> Tuple[np.ndarray, np.
657     ndarray, np.ndarray]:
658     """
659     Given the subject id, fetch the haxby data in matrix format.
660
661     Arguments:
662     - subject_id (int) : Subject number from [1,6]
663     - standardize (bool): If true, masks are standardized

```

```

664     Returns:
665         - data (Tuple[np.ndarray, np.ndarray, np.ndarray]) = Original 4-D data, Flattened + Masked
666           Data, Label
667
668     """
669
670     # Getting the data file name:
671     spatio_temporal_data_path = haxby_dataset.func[subject_id]
672
673     # Getting labels:
674     behavioral = pd.read_csv(haxby_dataset.session_target[subject_id], delimiter = ' ')
675
676     # Creating conditional categories:
677     conditions = behavioral['labels']
678
679     # Creating masks for stimuli categories, (ignores rest conditions)
680     condition_mask = conditions.isin([*stimuli2category]).tolist()
681
682     # Appplying masks to labels (categorical):
683     conditions = conditions[condition_mask]
684
685     # Creating labels series (numerical):
686     categories = np.array([stimuli2category[stimulus] for stimulus in conditions])
687
688     # Masking fMRI images: (shape = (40, 64, 64, 864))
689     fmri_niimgs = index_img(spatio_temporal_data_path, condition_mask)
690
691     # Converting NumPy and transposing to (864, 40, 64, 64):
692     numpy_fmri = fmri_niimgs.get_data().transpose(3,0,1,2)
693
694     masker = NiftiMasker(mask_img=haxby_dataset.mask_vt[subject_id],
695                          smoothing_fwhm=4,
696                          standardize=standardize,
697                          memory='nilearn_cache',
698                          memory_level=1)
699
700     masked = masker.fit_transform(fmri_niimgs)
701
702
703     return numpy_fmri, masked, categories
704
705
706 # In[105]:
707
708
709 data = [fetch_haxby_per_subject(subject_id) for subject_id in range(num_subjects)]
710 fmri_imgs_mat, masks, categories = list(zip(*data))
711
712 # Saving the data for future use:
713 save(fmri_imgs_mat, 'fMRI_data')
714 save(masks, 'masked_data')
715 save(categories, 'labels')
716
717
718 # In[102]:
719
720
721 # Loading:
722 fmri_imgs_mat, masks, categories = load('fMRI_data'), load('masked_data'), load('labels')
723
724
725 # # 4-D fMRI Data Similarity Analysis
726
727 # ## Functional Connectivity
728
729 # ### Correlation

```

```

730
731 # In[59]:
732
733
734 from nilearn.connectome import ConnectivityMeasure
735 correlation_measure = ConnectivityMeasure(kind='correlation')
736 correlation_matrix = correlation_measure.fit_transform([masks[subject_id]])[0]
737
738 fig = plt.figure()
739
740 # Mask out the major diagonal
741 np.fill_diagonal(correlation_matrix, 0)
742 plotting.plot_matrix(correlation_matrix,
743                     colorbar=True,
744                     vmax=0.8, vmin=-0.8,
745                     figure = fig)
746 plotting.show()
747
748 fig.savefig(os.path.join(explanatory_fMRI_dir, 'correlation.png'))
749
750
751 # ### Precision
752
753 # In[1]:
754
755
756 correlation_measure = ConnectivityMeasure(kind='precision')
757 correlation_matrix = correlation_measure.fit_transform([masks[subject_id]])[0]
758
759 fig = plt.figure()
760
761 # Mask out the major diagonal
762 np.fill_diagonal(correlation_matrix, 0)
763 plotting.plot_matrix(correlation_matrix, colorbar=True,
764                     vmax=0.8,
765                     vmin=-0.8,
766                     figure = fig)
767 plotting.show()
768
769
770 fig.savefig(os.path.join(explanatory_fMRI_dir, 'precision.png'))
771
772
773 # ### Partial Correlation
774
775 # In[62]:
776
777
778 correlation_measure = ConnectivityMeasure(kind='partial correlation')
779 correlation_matrix = correlation_measure.fit_transform([masks[subject_id]])[0]
780 fig = plt.figure()
781
782 # Mask out the major diagonal
783 np.fill_diagonal(correlation_matrix, 0)
784 plotting.plot_matrix(correlation_matrix, colorbar=True,
785                     vmax=0.8, vmin=-0.8, figure = fig)
786 plotting.show()
787 fig.savefig(os.path.join(explanatory_fMRI_dir, 'partial_correlation.png'))
788
789
790 # ### Cosine
791
792 # In[65]:
793
794
795 fig = plt.figure(figsize=(8,6))
796 plt.imshow(cdist(masks[subject_id], masks[subject_id], metric='cosine'))

```

```

797 plt.colorbar()
798 plt.title('Cosine Similarity of Masked fMRI Samples')
799 plt.show()
800 fig.savefig(os.path.join(explanatory_fMRI_dir, 'cosine.png'))
801
802
803 # ### Minkowski
804
805 # In[66]:
806
807
808 fig = plt.figure(figsize=(8,6))
809 plt.imshow(cdist(masks[subject_id], masks[subject_id], metric='minkowski'))
810 plt.colorbar()
811 plt.title('Minkowski Similarity of Masked fMRI Samples')
812 plt.show()
813 fig.savefig(os.path.join(explanatory_fMRI_dir, 'minkowski.png'))
814
815
816 # ### Euclidean
817
818 # In[67]:
819
820
821 fig = plt.figure(figsize=(8,6))
822 plt.imshow(cdist(masks[subject_id], masks[subject_id]))
823 plt.colorbar()
824 plt.title('Euclidean Similarity of Masked fMRI Samples')
825 plt.show()
826 fig.savefig(os.path.join(explanatory_fMRI_dir, 'euclidean.png'))
827
828
829 # # Visual Stimuli Transformations
830 #
831
832 # In[88]:
833
834
835 # Standardizing the data
836 scaler = StandardScaler()
837
838 # Normalizing data:
839 minmax_scaler = MinMaxScaler()
840
841
842 # In[114]:
843
844
845 def plot_2d(component1:np.ndarray, component2:np.ndarray,path:str, y = None, ) -> None:
846
847     fig = go.Figure(data=go.Scatter(
848         x = component1,
849         y = component2,
850         mode='markers',
851         marker=dict(
852             size=20,
853             color=y, #set color equal to a variable
854             colorscale='Rainbow', # one of plotly colorscales
855             showscale=True,
856             line_width=1
857         )
858     ))
859     fig.update_layout(margin=dict(l=100,r=100,b=100,t=100),width=2000,height=1200)
860     fig.layout.template = 'plotly_dark'
861
862     fig.show()
863

```



```

864     fig.write_image(path)
865
866 def plot_3d(component1 : np.ndarray,
867             component2 : np.ndarray,
868             component3 : np.ndarray,
869             path:str,
870             y = None) -> None:
871
872
873     fig = go.Figure(data=[go.Scatter3d(
874         x=component1,
875         y=component2,
876         z=component3,
877         mode='markers',
878         marker=dict(
879             size=10,
880             color=y,          # set color to an array/list of desired values
881             colorscale='Rainbow', # choose a colorscale
882             opacity=1,
883             line_width=1
884         )
885     )])
886     # tight layout
887     fig.update_layout(margin=dict(l=50,r=50,b=50,t=50),width=1800,height=1000)
888     fig.layout.template = 'plotly_dark'
889
890     fig.show()
891     fig.write_image(path)
892
893 def save_obj(obj:object, path:str = None) -> None:
894     with open(path + '.pkl', 'wb') as f:
895         pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)
896
897
898 def load_obj(path:str = None) -> object:
899     with open(path + '.pkl', 'rb') as f:
900         return pickle.load(f)
901
902
903 # ## PCA
904
905 # In[ ]:
906
907
908
909
910 x = masks[subject_id]
911 pca = PCA(n_components=3)
912 principalComponents = pca.fit_transform(x)
913
914 principal = pd.DataFrame(data = principalComponents
915                          ,columns = ['principal component 1',
916                                     'principal component 2',
917                                     'principal component 3'])
918
919 plot_2d(principalComponents[:, 0],
920         principalComponents[:, 1],
921         y = categories[subject_id],
922         path = os.path.join(explanatory_fmri_dir, 'pca_2d.png')
923     )
924
925
926 # In[ ]:
927
928
929 plot_3d(principalComponents[:, 0],
930         principalComponents[:, 1],

```

```

931     principalComponents[:, 2],
932     path = os.path.join(explanatory_fmri_dir, 'pca_3d.png'),
933     y = categories[subject_id])
934
935
936 # ## T-Stochastic Neighbor Embedding (t-SNE)
937
938 # In[ ]:
939
940
941
942
943 x = masks[subject_id]
944
945 tsne = TSNE(random_state = 42,
946             n_components=3,
947             verbose=0,
948             perplexity=40,
949             n_iter=400).fit_transform(x)
950
951 plot_2d(tsne[:, 0],
952        tsne[:, 1],
953        path = os.path.join(explanatory_fmri_dir, 'tsene_2d.png'),
954        y = categories[subject_id])
955
956
957 # In[ ]:
958
959
960 plot_3d(tsne[:, 0],
961        tsne[:, 1],
962        tsne[:, 2],
963        path = os.path.join(explanatory_fmri_dir, 'tsene_3d.png'),
964        y = categories[subject_id])
965
966
967 # ## Linear Discriminate Analysis
968
969 # In[ ]:
970
971
972
973
974
975 x = masks[subject_id]
976 y = categories[subject_id]
977
978 X_LDA = LDA(n_components=3).fit_transform(x,y)
979
980 plot_3d(X_LDA[:, 0],
981        X_LDA[:, 1],
982        X_LDA[:, 2],
983        path = os.path.join(explanatory_fmri_dir, 'lda_3d.png'),
984        y = categories[subject_id])
985
986
987 # ## Uniform Manifold Approximation and Projection (UMAP)
988
989 # In[ ]:
990
991
992
993 #!pip uninstall umap
994 #!pip install umap-learn
995
996 import umap.umap_ as umap
997

```

```

998 reducer = umap.UMAP(random_state=42,n_components=3)
999 embedding = reducer.fit_transform(x)
1000
1001
1002 plot_3d(embedding[:, 0],
1003         embedding[:, 1],
1004         embedding[:, 2],
1005         path = os.path.join(explanatory_fmri_dir, 'umap_3d.png'),
1006         y = categories[subject_id])
1007
1008
1009 # ## Independent Component Analysis (ICA)
1010
1011 # In[ ]:
1012
1013
1014
1015
1016 fast_ica = FastICA(n_components = 3)
1017 ICs = fast_ica.fit_transform(x)
1018
1019
1020 plot_3d(ICs[:, 0],
1021         ICs[:, 1],
1022         ICs[:, 2],
1023         path = os.path.join(explanatory_fmri_dir, 'ica_3d.png'),
1024         y = categories[subject_id])
1025
1026
1027 # ## Non-Negative Matrix Factorization
1028
1029 # In[ ]:
1030
1031
1032
1033
1034 nmf = NMF(n_components = 3, max_iter=500)
1035 MFs = nmf.fit_transform(minmax_scaler.fit_transform(x))
1036
1037 plot_3d(MFs[:, 0],
1038         MFs[:, 1],
1039         MFs[:, 2],
1040         path = os.path.join(explanatory_fmri_dir, 'nmf_3d.png'),
1041         y = categories[subject_id])
1042
1043
1044 # ## ISOMAP
1045
1046 # In[ ]:
1047
1048
1049 from sklearn.manifold import Isomap
1050 x = masks[subject_id]
1051
1052 embedding = Isomap(n_components=3)
1053 manifold = embedding.fit_transform(x)
1054
1055
1056 plot_3d(manifold[:, 0],
1057         manifold[:, 1],
1058         manifold[:, 2],
1059         path = os.path.join(explanatory_fmri_dir, 'isomap_3d.png'),
1060         y = categories[subject_id])
1061
1062
1063 # ## Locally Linear Embedding
1064

```

```

1065 # In[ ]:
1066
1067
1068 from sklearn.manifold import LocallyLinearEmbedding
1069
1070
1071 embedding = LocallyLinearEmbedding(n_components=3)
1072 manifold = embedding.fit_transform(x, categories[subject_id])
1073
1074
1075 plot_3d(manifold[:, 0],
1076         manifold[:, 1],
1077         manifold[:, 2],
1078         path = os.path.join(explanatory_fmri_dir, 'lle_3d.png'),
1079         y = categories[subject_id])
1080
1081
1082 # ## Multidimensional scaling
1083
1084 # In[ ]:
1085
1086
1087 from sklearn.manifold import MDS
1088
1089
1090 embedding = MDS(n_components=3)
1091 manifold = embedding.fit_transform(x, categories[subject_id])
1092
1093
1094 plot_3d(manifold[:, 0],
1095         manifold[:, 1],
1096         manifold[:, 2],
1097         path = os.path.join(explanatory_fmri_dir, 'mds_3d.png'),
1098         y = categories[subject_id])
1099
1100
1101 # ## Spectral Embedding
1102
1103 # In[ ]:
1104
1105
1106 from sklearn.manifold import SpectralEmbedding
1107
1108
1109 embedding = SpectralEmbedding(n_components=3)
1110 manifold = embedding.fit_transform(x)
1111
1112
1113 plot_3d(manifold[:, 0],
1114         manifold[:, 1],
1115         manifold[:, 2],
1116         path = os.path.join(explanatory_fmri_dir, 'SpectralEmbedding_3d.png'),
1117         y = categories[subject_id])
1118
1119
1120 # We can see among the linear and non-linear manifold learning algorithms, best separation is found
1121 # with LDA
1122
1123 # # Classical ML Algorithms
1124
1125 # ## One Shot ML Classifiers
1126
1127 # Applied Algorithms:
1128
1129 # * LinearSVC
1130 # * SGDClassifier
1131 # * MLPClassifier

```

```

1131 #      * Perceptron
1132 #      * LogisticRegression
1133 #      * LogisticRegressionCV
1134 #      * SVC
1135 #      * CalibratedClassifierCV
1136 #      * PassiveAggressiveClassifier
1137 #      * LabelPropagation
1138 #      * LabelSpreading
1139 #      * RandomForestClassifier
1140 #      * GradientBoostingClassifier
1141 #      * QuadraticDiscriminantAnalysis
1142 #      * RidgeClassifierCV
1143 #      * RidgeClassifier
1144 #      * AdaBoostClassifier
1145 #      * ExtraTreesClassifier
1146 #      * KNeighborsClassifier
1147 #      * BaggingClassifier
1148 #      * BernoulliNB
1149 #      * LinearDiscriminantAnalysis
1150 #      * GaussianNB
1151 #      * NuSVC
1152 #      * DecisionTreeClassifier
1153 #      * NearestCentroid
1154 #      * ExtraTreeClassifier
1155 #      * CheckingClassifier
1156 #      * DummyClassifier
1157
1158 # In[ ]:
1159
1160
1161
1162
1163 # Loading:
1164 fmri_imgs_mat, masks, categories = load('fMRI_data'), load('masked_data'), load('labels')
1165
1166
1167 predictions_per_subject = list()
1168
1169
1170 for subject_id, (mask, category) in enumerate(zip(masks, categories)):
1171
1172     print(f'Subject id: {subject_id}')
1173
1174     X_train, X_test, y_train, y_test = train_test_split(mask, category, test_size=0.3, random_state=42)
1175
1176     clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)
1177     models, predictions = clf.fit(X_train, X_test, y_train, y_test)
1178
1179     models.to_csv(os.path.join(results_dir, f'Subject_{subject_id}_lazy_results.csv'))
1180
1181     print(models)
1182
1183
1184 # ## FREM : Ensembling of Regularized Models for Robust Decoding (SVC - L2)
1185
1186 # FREM uses an implicit spatial regularization through fast clustering and aggregates a high number of
1187 # estimators trained on various splits of the training set, thus returning a very robust decoder at a
1188 # lower computational cost than other spatially regularized methods
1189
1190 # ---
1191
1192 # FREM ensembling procedure yields an important improvement of decoding accuracy on this simple example
1193 # compared to fitting only one model per fold and the clustering mechanism keeps its computational
1194 # cost reasonable even on heavier examples. Here we ensembled several instances of l2-SVC, but
1195 # FREMClassifier also works with ridge or logistic.
1196
1197 # In[ ]:

```

```

1193
1194
1195 from nilearn.decoding import FREMClassifier
1196 from nilearn.image import index_img
1197
1198 models_path = os.path.join(root_dir, 'models')
1199 num_subjects = 6
1200
1201 for subject_id in range(num_subjects):
1202
1203     print(f'Subject id: {subject_id}')
1204
1205     behavioral = pd.read_csv(haxby_dataset.session_target[subject_id], sep=" ")
1206
1207     conditions = behavioral['labels']
1208     condition_mask = conditions.isin([*stimuli2category])
1209
1210     # Split data into train and test samples, using the chunks
1211     condition_mask_train = (condition_mask) & (behavioral['chunks'] <= 8)
1212     condition_mask_test = (condition_mask) & (behavioral['chunks'] > 8)
1213
1214
1215     filenames = haxby_dataset.func[subject_id]
1216     X_train = index_img(filenames, condition_mask_train)
1217     X_test = index_img(filenames, condition_mask_test)
1218     y_train = conditions[condition_mask_train].values
1219     y_test = conditions[condition_mask_test].values
1220
1221     masker = NiftiMasker(mask_img=haxby_dataset.mask_vt[subject_id],
1222                          smoothing_fwhm=4,
1223                          standardize=True,
1224                          memory='nilearn_cache',
1225                          memory_level=1)
1226
1227     #masked = masker.fit_transform(fmri_niimgs)
1228
1229
1230     decoder = FREMClassifier(estimator='svc', cv=10, mask = masker)
1231
1232     # Fit model on train data and predict on test data
1233     decoder.fit(X_train, y_train)
1234
1235     y_pred = decoder.predict(X_test)
1236
1237     report = pd.DataFrame(classification_report(y_test, y_pred, output_dict = True)).T
1238     report.to_csv(os.path.join(results_dir, f'Subject_{subject_id}_FREM_results.csv'))
1239
1240     scores = pd.DataFrame(decoder.cv_scores_).T
1241     scores.to_csv(os.path.join(results_dir, f'Subject_{subject_id}_FREMCV_results.csv'))
1242
1243     save_obj(decoder, os.path.join(models_path, f'Subject_{subject_id}_FREM_model'))
1244
1245
1246 # ## FREM : Ensembling of Regularized Models for Robust Decoding (Logistic Regression - L2)
1247
1248 # In[446]:
1249
1250
1251 from nilearn.decoding import FREMClassifier
1252 from nilearn.image import index_img
1253 from sklearn.model_selection import LeaveOneGroupOut
1254 cv = LeaveOneGroupOut()
1255 models_path = os.path.join(root_dir, 'models')
1256 num_subjects = 6
1257
1258 for subject_id in range(num_subjects):
1259

```

```

1260 print(f'Subject id: {subject_id}')
1261
1262 behavioral = pd.read_csv(haxby_dataset.session_target[subject_id], sep=" ")
1263
1264 conditions = behavioral['labels']
1265 condition_mask = conditions.isin([*stimuli2category])
1266
1267 filenames = haxby_dataset.func[subject_id]
1268 X_train = index_img(filenames, condition_mask)
1269 y_train = conditions[condition_mask].values
1270
1271 decoder = FREMClassifier(estimator='logistic_l2',
1272                          cv=10,
1273                          mask = NiftiMasker(mask_img=haxby_dataset.mask_vt[subject_id],
1274                                              smoothing_fwhm=4,
1275                                              standardize=True,
1276                                              memory='nilearn_cache',
1277                                              memory_level=1)
1278                          )
1279
1280 # Fit model on train data and predict on test data:
1281 decoder.fit(X_train, y_train)
1282
1283 # Saving:
1284 scores = pd.DataFrame(decoder.cv_scores_).T
1285 scores.to_csv(os.path.join(results_dir, f'Subject_{subject_id}_FREMLogisticRegressionCV_results.csv
1286 '))
1287 save_obj(decoder, os.path.join(models_path, f'Subject_{subject_id}_FREMLogisticRegressionCV_model')
1288 )
1289
1290 # # ML Visualizations
1291 # ## Statistical Map Visualizations for ML Classifiers
1292
1293 # In[8]:
1294
1295 image_results_dir = os.path.join(root_dir, 'images/results')
1296 models_path = os.path.join(root_dir, 'models')
1297
1298 subject_id = 5
1299 decoder = load_obj(os.path.join(models_path, f'Subject_{subject_id}_FREM_model'))
1300
1301 weight_img = decoder.coef_img_["face"]
1302 filenames = haxby_dataset.func[subject_id]
1303
1304 plotting.plot_stat_map(weight_img,
1305                        bg_img = mean_img(filenames),
1306                        title=f"FREM: Accuracy Score for Face Stimuli: {np.mean(decoder.cv_scores_['face
1307 ']).round(2)}",
1308                        cut_coords=(-52, -5),
1309                        display_mode="yz",
1310                        #output_file= os.path.join(image_results_dir, 'FREM_face.png'),
1311                        )
1312
1313 plotting.show()
1314
1315 # In[9]:
1316
1317 subject_id = 5
1318 decoder = load_obj(os.path.join(models_path, f'Subject_{subject_id}_FREM_model'))
1319
1320 weight_img = decoder.coef_img_["house"]

```

```

1324 filenames = haxby_dataset.func[subject_id]
1325
1326 plotting.plot_stat_map(weight_img,
1327                        bg_img = mean_img(filenames),
1328                        title=f"FREM: Accuracy Score: {np.mean(decoder.cv_scores_['house']).round(2)}",
1329                        cut_coords=(-52, -5),
1330                        #output_file= os.path.join(image_results_dir, 'FREM_house.png'),
1331                        display_mode="yz")
1332
1333
1334
1335 plotting.show()
1336
1337
1338 # In[432]:
1339
1340
1341 subject_id = 0
1342 decoder = load_obj(os.path.join(models_path, f'Subject_{subject_id}_FREM_model'))
1343
1344 weight_img = decoder.coef_img_["face"]
1345
1346
1347 plotting.plot_stat_map(weight_img,
1348                        bg_img=haxby_dataset.anat[subject_id],
1349                        title='FREM (SVC-L2) Discriminating weights',
1350                        #output_file= os.path.join(image_results_dir, 'FREM (SVC-L2) Discriminating
1351                        weights.png'),
1352                        )
1353
1354 plotting.show()
1355
1356 # In[437]:
1357
1358
1359 subject_id = 0
1360 decoder = load_obj(os.path.join(models_path, f'Subject_{subject_id}_FREM_model'))
1361
1362 weight_img = decoder.coef_img_["face"]
1363
1364
1365 plotting.plot_stat_map(weight_img,
1366                        bg_img=haxby_dataset.anat[subject_id],
1367                        title='FREM (SVC-L2) Discriminating weights',
1368                        dim = -1,
1369                        #output_file= os.path.join(image_results_dir, 'FREM (SVC-L2) Discriminating
1370                        weights anat.png')
1371                        )
1372
1373 plotting.show()
1374
1375 # # ML Classifiers Accuracy Visualizations
1376
1377 # In[192]:
1378
1379
1380 def list2df(iterable):
1381     df = pd.DataFrame(iterable).T
1382     df.columns = cols
1383     return df
1384
1385
1386 # In[191]:
1387
1388

```



```

1389 average_df = None
1390 cols = ['Model', 'Accuracy', 'Balanced Accuracy', 'F1 Score', 'Time Taken']
1391 accuracy_svc = 0
1392 accuracy_logistic = 0
1393 num_subjects = 6
1394
1395 for subject_id in range(num_subjects):
1396     results_df = pd.read_csv(os.path.join(results_dir, f'Subject_{subject_id}_lazy_results.csv')).
1397     sort_values(by = 'Model')
1398     if subject_id == 0:
1399         average_df = results_df
1400     else:
1401         average_df += results_df
1402
1403     results_df_frem = pd.read_csv(os.path.join(results_dir, f'Subject_{subject_id}_FREMCV_results.csv')
1404     )
1405     accuracy_svc += results_df_frem.mean(1).mean()
1406
1407     results_df_frem_lr = pd.read_csv(os.path.join(results_dir, f'Subject_{subject_id}
1408     _FREMLogisticRegressionCV_results.csv'))
1409     accuracy_logistic += results_df_frem_lr.mean(1).mean()
1410
1411 accuracy_svc /= num_subjects
1412 accuracy_svc = round(accuracy_svc, 2)
1413 frem_col = ['FREM:SVCL2', accuracy_svc, accuracy_svc, accuracy_svc, '-']
1414 df_frem = list2df(frem_col)
1415
1416 accuracy_logistic /= num_subjects
1417 accuracy_logistic = round(accuracy_logistic, 2)
1418 frem_col_lr = ['FREM:LRL2', accuracy_logistic, accuracy_logistic, accuracy_logistic, '-']
1419 df_frem_lr = list2df(frem_col_lr)
1420
1421 cnn1 = ['2D CNN', 0.70, '-', '-', '-']
1422 cnn2 = ['3D CNN', 0.80, '-', '-', '-']
1423 twin = ['Twin-SVT', 0.82, '-', '-', '-']
1424 df_cnn1 = list2df(cnn1)
1425 df_cnn2 = list2df(cnn2)
1426 df_twin = list2df(twin)
1427
1428 average_df = average_df.drop('Model', axis = 1) / num_subjects
1429 average_df['Model'] = results_df['Model']
1430 average_df.drop('ROC AUC', axis = 1, inplace = True)
1431 average_df = average_df[cols]
1432 average_df = pd.concat([average_df, df_frem, df_frem_lr, df_cnn1, df_cnn2, df_twin])
1433
1434 average_df.sort_values('Accuracy', inplace = True, ascending = False)
1435 average_df.index = range(len(average_df))
1436 average_df.to_csv(os.path.join(results_dir, 'LazyAveragedResults.csv'), index=False)
1437 average_df
1438
1439 # In[194]:
1440
1441
1442 get_ipython().system('pip install graphviz')
1443 get_ipython().system('pip install torchviz')
1444
1445
1446 # In[195]:
1447
1448
1449 from graphviz import Digraph
1450
1451
1452

```

```

1453 # In[221]:
1454
1455
1456 from torch.utils.tensorboard import SummaryWriter
1457
1458 # default 'log_dir' is "runs" - we'll be more specific here
1459 writer = SummaryWriter('runs/2DVIS')
1460
1461
1462 # In[224]:
1463
1464
1465 writer.add_graph(net, x)
1466 writer.close()
1467
1468
1469 # In[229]:
1470
1471
1472 get_ipython().run_line_magic('load_ext', 'tensorboard')
1473
1474
1475 # In[230]:
1476
1477
1478 tensorboard --logdir=runs
1479
1480
1481 # # Deep Learning Algorithms
1482
1483 # In[12]:
1484
1485
1486 get_ipython().system('pip install vit-pytorch')
1487
1488
1489 # In[44]:
1490
1491
1492 get_ipython().system('conda install pytorch torchvision torchaudio cpuonly -c pytorch')
1493
1494
1495 # In[196]:
1496
1497
1498 import torch
1499 import torchvision
1500 import torch.nn as nn
1501 import torch.optim as optim
1502 import torch.utils.data
1503 import torchvision.datasets as dataset
1504 import torchvision.transforms as transforms
1505 import torch.nn.functional as F
1506 from PIL import Image
1507
1508
1509 # PyTorch's versions:
1510 print("PyTorch Version: ",torch.__version__)
1511 print("Torchvision Version: ",torchvision.__version__)
1512
1513 # We will be working with GPU:
1514 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
1515 print('Device : ' , device)
1516
1517 # Number of GPUs available.
1518 num_GPU = torch.cuda.device_count()
1519 print('Number of GPU : ' , num_GPU)

```

```

1520
1521
1522 # Creating stimuli to category and category to stimuli:
1523 stimuli2category = {
1524     'scissors'      : 0,
1525     'face'          : 1,
1526     'cat'           : 2,
1527     'scrambledpix'  : 3,
1528     'bottle'        : 4,
1529     'chair'         : 5,
1530     'shoe'          : 6,
1531     'house'         : 7
1532 }
1533
1534 category2stimuli = {category:stimuli for stimuli, category in stimuli2category.items()}
1535
1536
1537 # # Preparing fMRI Data for Batch Processing
1538
1539 # In[197]:
1540
1541
1542 class fMRIDataset(torch.utils.data.Dataset):
1543     scaler = MinMaxScaler()
1544     def __init__(self,
1545                 mode:str = 'fMRI',
1546                 transforms = None,
1547                 fetch_from_path:bool = True,
1548                 prepare_for_transformer:bool = False):
1549
1550         assert mode in ['fMRI','mask'], 'Please provide fMRI or Mask type of mode!'
1551
1552         self.transforms = transforms
1553         self.num_class = len(stimuli2category) or len(category2stimuli)
1554
1555         self.batch_data_path = 'batch_fMRI'
1556         self.batch_label_path = 'batch_label'
1557         self.batch_mask_path = 'batch_masks'
1558
1559         if prepare_for_transformer:
1560             self.batch_data_path = 'batch_fMRI_transformer'
1561             self.batch_label_path = 'batch_label_transformer'
1562
1563
1564         batched_data_path = os.path.join(root_dir, self.batch_data_path)
1565         batched_label_path = os.path.join(root_dir, self.batch_label_path)
1566         batched_mask_path = os.path.join(root_dir, self.batch_mask_path)
1567
1568
1569         if mode == 'fMRI':
1570             if fetch_from_path:
1571                 if os.path.exists(batched_data_path + '.npy') and os.path.exists(batched_label_path +
1572                 '.npy'):
1573
1574                     print(f'Data is fetching from {root_dir}')
1575                     self.data = load(batched_data_path)
1576                     self.labels = load(batched_label_path)
1577
1578                 else:
1579                     raise NoneError("Object not constructed. Cannot access a 'None' object.")
1580             else:
1581                 self.data = np.concatenate(load('fMRI_data'), axis = 0)
1582                 self.labels = np.concatenate(load('labels'), axis = 0)
1583
1584         if prepare_for_transformer:
1585             self.prepare_transformer()

```

```

1586         save(self.data, batched_data_path)
1587         save(self.labels, batched_label_path)
1588
1589     else:
1590         pass
1591
1592
1593
1594
1595     assert self.labels.shape[0] == self.data.shape[0], ' # of Targets and Data samples does not
match!'
1596
1597
1598     def prepare_transformer(self):
1599         self.data = self.data[:, 1:, :, :, ].reshape(-1, 64, 64, 3)
1600         self.labels = np.repeat(self.labels, repeats = 13, axis = 0)
1601
1602     def __len__(self):
1603         return len(self.data)
1604
1605     def __getitem__(self, idx):
1606         image = self.data[idx]
1607
1608
1609         if image.shape == torch.Size([64, 3, 64]):
1610             image = image.permute(1,0,2)
1611
1612         #assert image.shape == torch.Size([3, 64, 64]), 'Mismatch Image Dimension!'
1613
1614         label = self.labels[idx].reshape(1,)
1615         label = torch.as_tensor(label, dtype=torch.int, device=device)
1616
1617         if self.transforms is not None:
1618             image = self.transforms(image)
1619
1620         return image, label
1621
1622
1623 # In[198]:
1624
1625
1626 class Normalize():
1627     def __call__(self, image):
1628         max_val = image.max()
1629         return image / max_val
1630
1631 class TorchTensor():
1632     def __call__(self, image):
1633         return torch.as_tensor(image, dtype=torch.float, device=device)
1634
1635 class MeanNormalize():
1636     def __call__(self, image):
1637         return F.normalize(image)
1638
1639 # [batch * channel(# of channels of each image) * depth(# of frames) * height * width]
1640 class Make3D():
1641     def __call__(self, image):
1642         return image.unsqueeze(0)
1643
1644 class MinMax():
1645     def __call__(self, image):
1646         min_val = image.min(axis = 0)
1647         max_val = image.max(axis = 0)
1648         return (image - min_val)/(max_val - min_val)
1649
1650 class Clamp():
1651     def __call__(self, image):

```

```

1652         return torch.clamp(image, max=2000)
1653
1654 class Log():
1655     def __call__(self, image):
1656         return torch.log10(image+1)
1657
1658
1659 # In[207]:
1660
1661
1662 transform = transforms.Compose([
1663     #transforms.ColorJitter([0.9,0.9]),
1664     #transforms.RandomGrayscale(p = 0.3),
1665     #transforms.RandomAffine((-30,30)),
1666     #transforms.RandomPerspective(),
1667     #transforms.GaussianBlur(3),
1668     #transforms.RandomHorizontalFlip(p = 0.2),
1669     #transforms.RandomVerticalFlip(p = 0.2),
1670
1671     #Important parts, above can be ignored
1672     #transforms.Resize((224,224)),
1673     #transforms.CenterCrop(224),
1674     Normalize(),
1675     TorchTensor(),
1676     #transforms.ToTensor(),
1677 ])
1678
1679
1680 fMRI_dataset = fMRIDataset(transforms = transform, fetch_from_path = True)
1681
1682
1683 break_point = len(fMRI_dataset) - 100
1684 train_dataset = torch.utils.data.Subset(fMRI_dataset, indices = range(break_point))
1685 val_dataset = torch.utils.data.Subset(fMRI_dataset, indices = range(break_point, len(fMRI_dataset)))
1686
1687
1688 # In[223]:
1689
1690
1691 batch_size = 16
1692 train_loader = torch.utils.data.DataLoader(dataset = train_dataset,
1693     shuffle = False,
1694     batch_size = batch_size,
1695     drop_last = True,
1696 )
1697
1698 val_loader = torch.utils.data.DataLoader(dataset = val_dataset,
1699     shuffle = False,
1700     batch_size = batch_size,
1701     drop_last = True,
1702 )
1703
1704 x, y = next(iter(train_loader))
1705
1706 print(x.shape, x.dtype)
1707 print(y.shape, y.dtype)
1708
1709
1710 # In[159]:
1711
1712
1713 from torch_utils import utils_torch
1714 def train_one_epoch(model, criterion, optimizer, data_loader, device, epoch, print_freq, apex=False):
1715     model.train()
1716     metric_logger = utils_torch.MetricLogger(delimiter=" ")
1717     metric_logger.add_meter('lr', utils_torch.SmoothedValue(window_size=1, fmt='{value}'))
1718     metric_logger.add_meter('img/s', utils_torch.SmoothedValue(window_size=10, fmt='{value}'))

```

```

1719 header = 'Epoch: [{}].format(epoch)
1720
1721 for image, target in metric_logger.log_every(data_loader, print_freq, header):
1722     start_time = time.time()
1723     image, target = image.to(device), target.to(device).squeeze(-1).long()
1724     output = model(image)
1725     loss = criterion(output, target)
1726
1727     optimizer.zero_grad()
1728     loss.backward()
1729     optimizer.step()
1730
1731     acc1, acc5 = utils_torch.accuracy(output, target, topk=(1, 5))
1732     batch_size = image.shape[0]
1733     metric_logger.update(loss=loss.item(), lr=optimizer.param_groups[0]["lr"])
1734     metric_logger.meters['acc1'].update(acc1.item(), n=batch_size)
1735     metric_logger.meters['acc5'].update(acc5.item(), n=batch_size)
1736     metric_logger.meters['img/s'].update(batch_size / (time.time() - start_time))
1737
1738
1739 def evaluate(model, criterion, data_loader, device, print_freq=100):
1740     model.eval()
1741     metric_logger = utils_torch.MetricLogger(delimiter=" ")
1742     header = 'Test:'
1743     with torch.no_grad():
1744         for image, target in metric_logger.log_every(data_loader, print_freq, header):
1745             image = image.to(device, non_blocking=True)
1746             target = target.to(device, non_blocking=True).squeeze(-1).long()
1747             output = model(image)
1748             loss = criterion(output, target)
1749
1750             acc1, acc5 = utils_torch.accuracy(output, target, topk=(1, 5))
1751             # FIXME need to take into account that the datasets
1752             # could have been padded in distributed setup
1753             batch_size = image.shape[0]
1754             metric_logger.update(loss=loss.item())
1755             metric_logger.meters['acc1'].update(acc1.item(), n=batch_size)
1756             metric_logger.meters['acc5'].update(acc5.item(), n=batch_size)
1757     # gather the stats from all processes
1758     metric_logger.synchronize_between_processes()
1759
1760     print(' * Acc@1 {top1.global_avg:.3f} Acc@5 {top5.global_avg:.3f}'
1761           .format(top1=metric_logger.acc1, top5=metric_logger.acc5))
1762     return metric_logger.acc1.global_avg
1763
1764
1765 # ## Models
1766
1767 # In[199]:
1768
1769
1770 class Model(nn.Module):
1771     def __init__(self, model = None):
1772         super(Model, self).__init__()
1773         if model is not None:
1774             self.model = model
1775         else:
1776             self.model = nn.Sequential(
1777                 self.conv_block(40, 60, 0.1),
1778                 self.conv_block(60, 80, 0.15),
1779                 self.conv_block(80, 128, 0.25),
1780                 self.conv_block(128, 256, 0.3),
1781                 nn.Flatten(),
1782                 self.linear_block(1024, 256, 0.4),
1783                 self.linear_block(256, 128, 0.4),
1784                 nn.Linear(128, 8)
1785 )

```

```

1786
1787     def forward(self, img):
1788         return self.model(img)
1789
1790     @staticmethod
1791     def conv_block(in_channel, out_channel, p):
1792         return nn.Sequential(
1793             nn.Conv2d(in_channel, out_channel, 3),
1794             nn.BatchNorm2d(out_channel),
1795             nn.ReLU(),
1796             nn.MaxPool2d(2, 2),
1797             nn.Dropout2d(p)
1798         )
1799
1800     @staticmethod
1801     def linear_block(in_ftrs, out_ftrs, p):
1802         return nn.Sequential(
1803             nn.Linear(in_ftrs, out_ftrs),
1804             nn.BatchNorm1d(num_features=out_ftrs),
1805             nn.ReLU(),
1806             nn.Dropout(p)
1807         )
1808
1809 net = Model().to(device)
1810 print('Trainable parameter of the model: ', sum(param.numel() for param in net.parameters() if param.
1811       requires_grad == True))
1812 print(net)
1813
1814 # ## Creating Loss function, Optimizer, Scheduler (If any)
1815
1816 # In[31]:
1817
1818
1819 # loss function
1820 criterion = nn.CrossEntropyLoss()
1821 # optimizer
1822 optimizer = optim.Adam(net.parameters())
1823 # scheduler
1824 scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma = 0.7)
1825
1826
1827 # In[ ]:
1828
1829
1830 # let's train it for 10 epochs
1831 num_epochs = 10
1832 print_freq = 10
1833
1834 for epoch in range(num_epochs):
1835     # train for one epoch, printing every 10 iterations
1836     train_one_epoch(net, criterion, optimizer, train_loader, device, epoch, print_freq, apex=False)
1837
1838     # update the learning rate
1839     scheduler.step()
1840     # evaluate on the test dataset
1841     evaluate(net, criterion, val_loader, device, print_freq)
1842
1843 print("That's it!")
1844
1845
1846 # ## MLPs for Masks
1847
1848 # In[160]:
1849
1850
1851 class MaskDataset(torch.utils.data.Dataset):

```

```

1852 def __init__(self, mask, category, transforms = None):
1853     self.mask = mask
1854     self.category = category
1855     self.transforms = transforms
1856
1857 def __len__(self):
1858     return len(self.mask)
1859
1860 def __getitem__(self, idx):
1861     mask = self.mask[idx]
1862     label = self.category[idx]
1863     label = torch.as_tensor(label, dtype = torch.int, device = device)
1864
1865     if self.transforms is not None:
1866         mask = self.transforms(mask)
1867
1868     return mask, label
1869
1870 class MLP(nn.Module):
1871     def __init__(self, in_fters, hidden1_dim, hidden2_dim, num_class):
1872         super(MLP, self).__init__()
1873
1874         self.fc1 = nn.Linear(in_fters, hidden1_dim)
1875         self.dropout1 = nn.Dropout2d(0.5)
1876         self.gelu1 = nn.GELU()
1877         self.fc2 = nn.Linear(hidden1_dim, hidden2_dim)
1878         self.dropout2 = nn.Dropout2d(0.25)
1879         self.gelu2 = nn.GELU()
1880         self.fc3 = nn.Linear(hidden2_dim, num_class)
1881
1882     def forward(self, x):
1883         x = self.gelu1(self.dropout1(self.fc1(x)))
1884         x = self.gelu2(self.dropout2(self.fc2(x)))
1885         return self.fc3(x)
1886
1887 # loss function
1888 criterion = nn.CrossEntropyLoss()
1889 # optimizer
1890 optimizer = optim.Adam(net.parameters())
1891 # scheduler
1892 scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma = 0.7)
1893
1894
1895 # In[161]:
1896
1897
1898 masks[0].shape, 864 /16
1899
1900
1901 # In[ ]:
1902
1903
1904 masks, categories = load('masked_data'), load('labels')
1905 num_epochs = 5
1906 print_freq = 5
1907 batch_size = 16
1908
1909 for subject_id in range(num_subjects):
1910
1911
1912     transform = torchvision.transforms.Compose([
1913         MinMax(),
1914         TorchTensor(),
1915
1916
1917     ])
1918

```



```

1919 maskdata = MaskDataset(masks[subject_id], categories[subject_id], transform)
1920
1921 break_point = len(maskdata) - 50
1922 train_dataset = torch.utils.data.Subset(maskdata, indices = range(break_point))
1923 val_dataset = torch.utils.data.Subset(maskdata, indices = range(break_point, len(maskdata)))
1924 batch_size = 16
1925
1926 train_loader = torch.utils.data.DataLoader(dataset = train_dataset,
1927                                           shuffle = True,
1928                                           batch_size = batch_size,
1929                                           drop_last = True,
1930                                           )
1931
1932 val_loader = torch.utils.data.DataLoader(dataset = val_dataset,
1933                                           shuffle = False,
1934                                           batch_size = batch_size,
1935                                           drop_last = True,
1936                                           )
1937
1938
1939 x, _ = next(iter(train_loader))
1940
1941 mlp_kwargs = dict(in_fts = x.size(1), hidden1_dim = 256, hidden2_dim = 128, num_class = 8)
1942 net = MLP(**mlp_kwargs)
1943
1944
1945 for epoch in range(num_epochs):
1946
1947     # train for one epoch, printing every 10 iterations
1948     train_one_epoch(net, criterion, optimizer, train_loader, device, epoch, print_freq, apex=False)
1949     # update the learning rate
1950     scheduler.step()
1951     # evaluate on the test dataset
1952     evaluate(net, criterion, val_loader, device, print_freq)
1953
1954
1955
1956
1957
1958
1959
1960 # ## 3-D Convolutional Neural Network
1961
1962 # In[234]:
1963
1964
1965 class Model3D(nn.Module):
1966     def __init__(self, model = None):
1967         super(Model3D, self).__init__()
1968         if model is not None:
1969             self.model = model
1970         else:
1971             self.conv3d = nn.Sequential(
1972                 self.conv_block(1, 32, 0),
1973                 self.conv_block(32, 64, 0),
1974                 self.conv_block(64, 128, 0),
1975                 #self.conv_block(128, 256, 0.3),
1976                 nn.Flatten()
1977             )
1978             conv_out_size = self._get_conv_out((1, 1, 40, 64, 64))
1979
1980             lin = nn.Linear(256, 8)
1981             torch.nn.init.xavier_uniform_(lin.weight)
1982
1983             self.linear = nn.Sequential(
1984                 self.linear_block(conv_out_size, 512, 0.1),
1985                 self.linear_block(512, 256, 0.1),

```

```

1986         lin
1987     )
1988
1989
1990     self.model = nn.Sequential(
1991         self.conv3d,
1992         self.linear
1993     )
1994
1995     def forward(self, img):
1996         return self.model(img.unsqueeze(1))
1997
1998     @staticmethod
1999     def conv_block(in_channel, out_channel, p):
2000         cnn = nn.Conv3d(in_channel, out_channel, (3,3,3), padding=(1,1,1))
2001         torch.nn.init.xavier_normal_(cnn.weight)
2002         return nn.Sequential(
2003             cnn,
2004             nn.BatchNorm3d(out_channel),
2005             nn.ReLU(),
2006             nn.MaxPool3d(2),
2007             #nn.Dropout3d(p)
2008         )
2009
2010     def _get_conv_out(self, shape):
2011         o = self.conv3d(torch.zeros(*shape))
2012         return int(np.prod(o.size()))
2013
2014     @staticmethod
2015     def linear_block(in_ftrs, out_ftrs, p):
2016         linear = nn.Linear(in_ftrs, out_ftrs)
2017         torch.nn.init.xavier_uniform_(linear.weight)
2018         return nn.Sequential(
2019             linear,
2020             nn.BatchNorm1d(num_features=out_ftrs),
2021             nn.ReLU(),
2022             nn.Dropout(p)
2023         )
2024
2025 net = Model3D().to(device)
2026 #net = torch.load("best_model.pkl")
2027
2028
2029 criterion = nn.CrossEntropyLoss()
2030 # optimizer
2031 optimizer = optim.Adam(net.parameters(), lr=1e-4, weight_decay=1e-4)
2032 # scheduler
2033 scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma = 0.8)
2034
2035
2036 num_epochs = 200
2037 print_freq = 3
2038
2039 for epoch in range(num_epochs):
2040     # train for one epoch, printing every 10 iterations
2041     train_one_epoch(net, criterion, optimizer, train_loader, device, epoch, print_freq, apex=False)
2042
2043     # update the learning rate
2044     scheduler.step()
2045     # evaluate on the test dataset
2046     evaluate(net, criterion, val_loader, device, print_freq)
2047
2048     #torch.save(net, "best_model.pkl")
2049
2050 print("That's it!")
2051
2052 # Validation

```

```

2053 preds = []
2054
2055 with torch.no_grad():
2056     for image, target in val_loader:
2057         image = image.to(device, non_blocking=True)
2058         target = target.to(device, non_blocking=True).squeeze(-1).long()
2059         output = net(image)
2060
2061         preds += output.argmax(dim=1).tolist()
2062
2063
2064 report = classification_report(fMRI_dataset.labels[break_point:break_point+len(preds)], preds,
2065                               target_names=list(stimuli2category.keys()), digits=4)
2066 acc = sklearn.metrics.accuracy_score(fMRI_dataset.labels[break_point:break_point+len(preds)], preds)
2067
2068
2069 # ## Visual Transformers
2070
2071 # In[3]:
2072
2073 import torch
2074 from vit_pytorch.twins_svt import TwinsSVT
2075
2076 net = TwinsSVT(
2077     num_classes = 8,          # number of output classes
2078     s1_emb_dim = 64,          # stage 1 - patch embedding projected dimension
2079     s1_patch_size = 4,        # stage 1 - patch size for patch embedding
2080     s1_local_patch_size = 7,   # stage 1 - patch size for local attention
2081     s1_global_k = 7,          # stage 1 - global attention key / value reduction factor, defaults to 7
2082     as_specified_in_paper
2083     s1_depth = 1,             # stage 1 - number of transformer blocks (local attn -> ff -> global attn
2084                               -> ff)
2085     s2_emb_dim = 128,          # stage 2 (same as above)
2086     s2_patch_size = 2,
2087     s2_local_patch_size = 7,
2088     s2_global_k = 7,
2089     s2_depth = 1,
2090     s3_emb_dim = 256,          # stage 3 (same as above)
2091     s3_patch_size = 2,
2092     s3_local_patch_size = 7,
2093     s3_global_k = 7,
2094     s3_depth = 5,
2095     s4_emb_dim = 512,          # stage 4 (same as above)
2096     s4_patch_size = 2,
2097     s4_local_patch_size = 7,
2098     s4_global_k = 7,
2099     s4_depth = 4,
2100     peg_kernel_size = 3,      # positional encoding generator kernel size
2101     dropout = 0.              # dropout
2102 )
2103
2104 img = torch.randn(1, 3, 224, 224)
2105
2106 pred = net(img) # (1, 8)
2107
2108 criterion = nn.CrossEntropyLoss()
2109 # optimizer
2110 optimizer = optim.Adam(net.parameters(), lr=1e-4, weight_decay=1e-4)
2111 # scheduler
2112 scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma = 0.8)
2113
2114 num_epochs = 200
2115 print_freq = 3
2116

```

```

2117 for epoch in range(num_epochs):
2118     # train for one epoch, printing every 10 iterations
2119     train_one_epoch(net, criterion, optimizer, train_loader, device, epoch, print_freq, apex=False)
2120
2121     # update the learning rate
2122     scheduler.step()
2123     # evaluate on the test dataset
2124     evaluate(net, criterion, val_loader, device, print_freq)
2125
2126 # Note that the code below are utils.
2127 class PDF(object):
2128     def __init__(self, pdf, size=(200,200)):
2129         self.pdf = pdf
2130         self.size = size
2131
2132     def _repr_html_(self):
2133         return '<iframe src={0} width={1[0]} height={1[1]}></iframe>'.format(self.pdf, self.size)
2134
2135     def _repr_latex_(self):
2136         return r'\includegraphics[width=1.0\textwidth]{{{0}}}'.format(self.pdf)
2137
2138 if __name__ == "__main__":
2139     #PDF('Haxby_etal01.pdf', size=(300,250))
2140     pass
2141
2142 import nibabel as nib
2143 import numpy as np
2144
2145
2146 def fetch_from_haxby(haxby_dataset_path:str = None) -> np.ndarray:
2147     return nib.load(haxby_dataset_path).get_data()
2148
2149 from __future__ import print_function, division
2150
2151 # Basics:
2152 import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
2153
2154 # Ignore warnings
2155 import warnings
2156 warnings.filterwarnings("ignore")
2157
2158 # Extras:
2159 from abc import abstractmethod
2160 from typing import Callable, Iterable, List
2161
2162
2163 def confusion_matrix(labels:Iterable[list or np.ndarray],
2164                     preds:Iterable[list or np.ndarray]) -> pd.DataFrame:
2165     """
2166     Takes desireds/labels and softmax predictions,
2167     return a confusion matrix.
2168
2169     """
2170     label = pd.Series(labels, name='Actual')
2171     pred = pd.Series(preds, name='Predicted')
2172     return pd.crosstab(label, pred)
2173
2174
2175
2176
2177
2178 def accuracy(labels, preds):
2179     return (np.sum(preds == labels) / labels.shape) * 100
2180
2181
2182
2183 def visualize_confusion_matrix(data:np.ndarray,

```

```

2184         normalize:bool = True,
2185         title:str = " ") -> None:
2186
2187     if normalize:
2188
2189         data /= np.sum(data)
2190
2191     plt.figure(figsize=(15,15))
2192     sns.heatmap(data,
2193                 fmt='.2%',
2194                 cmap = 'Greens')
2195
2196     plt.title(title)
2197     plt.show()
2198
2199
2200 import numpy as np
2201 from typing import Callable, Iterable, List
2202
2203
2204
2205
2206 def random_seed(Func:Callable):
2207     """
2208
2209     Decorator random seeder.
2210
2211     """
2212     def _random_seed(*args, **kwargs):
2213         np.random.seed(42)
2214         random.seed(42)
2215         result = Func(*args, **kwargs)
2216         return result
2217     return _random_seed
2218
2219
2220
2221 import pickle
2222 import numpy as np
2223
2224 def save_obj(obj:object, path:str = None) -> None:
2225     with open(path + '.pkl', 'wb') as f:
2226         pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)
2227
2228
2229 def load_obj(path:str = None) -> object:
2230     with open(path + '.pkl', 'rb') as f:
2231         return pickle.load(f)
2232
2233
2234 def save(data:np.ndarray = None, path:str = None) -> None:
2235     np.save(path + '.npy', data, allow_pickle=True)
2236
2237
2238 def load(path:str = None) -> np.ndarray:
2239     return np.load(path + '.npy', allow_pickle=True)
2240
2241
2242 import time
2243 from abc import abstractmethod
2244 from typing import Callable, Iterable, List
2245
2246
2247 def timeit(Func:Callable):
2248     def _timeStamp(*args, **kwargs):
2249         since = time.time()
2250         result = Func(*args, **kwargs)

```

```

2251         time_elapsed = time.time() - since
2252
2253         if time_elapsed > 60:
2254             print('Time Consumed : {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
2255         else:
2256             print('Time Consumed : ' , round((time_elapsed),4) , 's')
2257         return result
2258     return _timeStamp
2259
2260
2261 from nilearn import plotting
2262 from nilearn import image
2263 import random
2264 import matplotlib.pyplot as plt, seaborn as sns
2265
2266
2267 def RoI_visualizer(haxby_dataset, subject_id:int = random.randint(0,5)) -> None:
2268     """
2269     Given the subject id from i = 1,...,6, visualize the a mask of the Ventral Temporal (VT) cortex
2270     ,
2271     coming from the Haxby with the Region of Interest (RoI)
2272
2273     Arguments:
2274
2275         subject_id (int) = Subject number
2276
2277     Returns:
2278         - None
2279     """
2280     # Subject ID from i = 0,...,5:
2281     # subject_id = 3
2282
2283     # Get mask filename:
2284     mask_filename = haxby_dataset.mask_vt[subject_id]
2285
2286
2287     # Region of Interest Visualizations:
2288     plotting.plot_roi(mask_filename,
2289                       bg_img=haxby_dataset.anat[subject_id],
2290                       cmap='Paired',
2291                       title = f'Region of Interest of subject {subject_id}',
2292                       figure= plt.figure(figsize=(12,4)),
2293                       alpha=0.7)
2294
2295     plotting.show()

```