EEE482 Computational Neuroscience Final Project Report Visual Object Recognition

Ayhan Okuyan, Emre Donmez, Baris Akcin, Alp Kumbasar
name.surname[at]ug.bilkent.edu.tr
Faculty of Electrical and Electronics Engineering
Bilkent University
06800 Bilkent, Ankara
TURKEY

***Abstract*— This project is focuses on the topic of Visual Object Recognition using the Haxby Dataset for MVPA (Multi-Voxel Pattern Analysis), which provides FMRI records of different subjects' brains while they are being shown black and white photographs of everyday objects. We have approached the application as a two-part pipeline where the first part is the Feature Selection/Reduction phase for which we mapped the data to lower dimensions using PCA and Temporal Masking. The second part of the pipeline consisted of the classification algorithm where we tried several alternatives to obtain the best result we could. Then the best flow is explained and discuss why it became the best one.**

I. INTRODUCTION

The primary goal in the field of Neuroscience is to link specific brain activities with specific tasks that the human performs. With Computational Neuroscience, we analyse these brain activities using Machine Learning tools and map them with specific tasks that the brain performs. We do this with the help of Machine Learning tools and algorithms. In this project, we use MVPA(Multi-Voxel Pattern Analysis) on brain fMRI data in order to map certain visual activities to brain activities. This type of research plays an important role on the field of Computational Neuroscience since it is directly related to the main goal of the field, mapping brain stimulus with performed task.

In this report, we will be introducing the methods and results of analysis on recorded human brain activity when different visual stimuli are shown to subjects. We have implemented 26 different multi-voxel pattern analysis methods based on 9 different classifiers and four altered versions of the dataset that we are assigned in order to decode the category of the shown stimuli to the subject according to their recorded brain activity.

Each classification method with different models are trained and validated for each subject separately and for each pre-processing (Data Reduction) method. The highest validation accuracy giving model for each subject and each data reduction is then evaluated on the test set for their corresponding subjects. The final accuracy for each data reduction model is then obtained by taking the mean of test accuracies of every subject on the corresponding data reduction method results.

II. METHODS

Here, we have decided to implement a Hierarchical Group-Level Multivariate Pattern Analysis (G-MVPA) [16] without cross validation. Instead we have followed with a three-way split of the data, since we were unable to meet (as this report is constructed in 2020 Covid-19 Crisis), we wanted to make sure that the data we used match with each other so that we could compare the results more confidently. We have used a 80%-10%-10% training-validation-test split and shared the split data with the project teammates. (see Appendix).

A. Dataset

In this project, we have used the Haxby Dataset[12] which contains the FMRI data for 6 subjects each having 121 time steps and 12 trials except for subject 5, who has only 11 trials, 12^{th} trial is corrupted. The brain FMRI data is 4 dimensional and has a shape of (x,y,z,t) (64,64,40,72). In total, the data set for each subject except the 5^{th} subject has dimension 12x40x64x64x121 and the 5^{th} subject has data with dimension 11x40x64x64x121. There are 8 different grayscale images that are shown to each subject. These images are scissors, face, cat, scrambled pix, bottle, chair, shoe, and house. Every image is shown for 9 FMRI time steps to each subject and between each image, there is a waiting time step of 6. Also, the first image is shown starting from the 6^{th} time step. We have discarded the pauses between and got only the time steps with actual stimuli shown resulting in 72 time steps to use. After this, for each subject we have flattened the trials and time steps resulting in 792 instances for subject 5 and 864 instances for the rest of subjects. Finally we have flattened the data across brain FMRI's 3D temporal data and obtained a feature set of size 163840 for each subject. The resulting data for every subject other than the 5^{th} is 864x163840 and 792x163849 for the 5^{th} subject.

B. Feature Reduction/Selection Algorithms

In order to deal with the highly dimensional and complex data, we have used Feature Reduction and Selection techniques to decrease the complexity of the data. The Feature Selection technique that we have chosen to implement is the Masking procedure, which uses pre-created masks to filter the locations according to the temporal ventral cortex. For feature reduction, we have used Principle Component Analysis (PCA).

1) Principle Component Analysis - PCA: As a dimensionality reduction method, we chose PCA which is one of the most popular methods. We used sklearn.decomposition library for the PCA method. The reason behind choosing this method is it allows us to fit the data without setting dimension. It automatically reduces dimensions according to the features. For this experiment, we have tried to choose the number of PCs that would yield a 95% explained variance, which corresponded to 4 PCs, which we found to be too small and decided on using 100 Principle Components.

2) Temporal Masking: Original dataset consists of brain FMRI data which covers the whole brain. Since visual object recognition occurs in a specific part of the brain, we decided to make a research about what method we can use to reduce the dimension of this complex data structure. While searching, we found the haxby2001 dataset, the same dataset that we were assigned with, but it also contained temporal masks for each subject highlighting the voxels that are in their temporal ventral cortex region, which is associated with visual recognition tasks [3]. We decided to apply the masks for each subject for dimensionality reduction procedure, to reduce the feauture set of each subject's dataset.

C. Learning Algorithms - Classifiers

1) k-Nearest Neighbors - kNN: kNN is a pattern recognition algorithm which find k-nearest training samples for the input. As a library, sklearn.neighbors' KneighborsClassifier is used [1]. For visual object recognition, we decided to use kNN with various k values which will change from 1 to 25. Each set consists of 6 different subjects, so we will apply these methods for 6 subjects. This process will be applied for 4 different forms of datasets we obtained which are original data, PCA data, masked data and masked PCA data.

2) Support Vector Machine (Classifier) - SVM (SVC): SVM is a learning algorithm that build a model in training to assign new points to either one class or the other, making the assigned points a binary linear decision boundry[12]. Then using what is known as the "kernel trick", one can use SVMs to build nonlinear decision boundaries just by replacing the kernel function by a polynomial, a radial basis function (RBF). In order to implement SVM, we have used sklearn library, which offers extensive decisions to build a general classifier. For each subject, we have performed a series of trials to build the best model by changing the C (punishment coefficient) in between $1, 2, 4, 7, 10$ and the kernel function as linear, RBF, polynomial and sigmoid. The obtained models for each subject are then tested on their respective test sets. The motivation behind using an SVM classifier is that

as stated by Mahmoudi et. al. [12], they became popular in analyzing FMRI data due to their capability of dealing with high dimensional data and being flexible in terms of decision boundaries.

3) Random Forest: The random forest algorithm uses a decision-tree based model. The strength of the random forest over a simple decision tree is the fact that it uses many randomly created decision-trees which use a different portion of the dataset's features randomly selected to build itself. Each decision tree in the forest then performs a prediction on the validation or test data and the prediction that is predicted the most is selected as the final prediction of the random forest model. This gives the Random Forest algorithm advantage over the Decision tree algorithm by increasing the randomness of the build which eliminates overfitting and creates a more general classification model. The Labels are encoded using a dictionary and are converted to integer values in range 0-7.

Implementation

We have generated 7 different Random Forest models for every subject which have different number of estimators, different values for maximum depth of trees, and different values for minimum samples a leaf node can have. Each Random Forest model is tested on the validation set of that subject and the best resulting Random Forest model is used for predicting the test set of that corresponding subject. After all subject accuracies are obtained, the mean of these accuracies was taken as the final accuracy for the used dataset model which consists of the Original Dataset, PCA Dataset, Masked Dataset, and Masked-PCA Dataset.

4) AdaBoost with Random Forest: The Random Forest with AdaBoost method is similar to the Random Forest Algorithm in the way that it creates many decision trees with a random feature set and forms a forest, however what is different is the way that these trees are generated. In AdaBoost Random Forest, the Trees that are generated are almost always stumps which means a tree with a depth of 1. This of course decreases the learning power of a single tree. However, with Ada Boosted Random Forest, each stump is made in consideration of the previous generated stumps and works to eliminate their error by using the Gini index calculation. This results in many stumps with different weights that have different amounts of effect on the final prediction and all of them are directly related with each other, resulting in a more controlled growth of the forest compared to the Random Forest algorithm. The same preprocessing from the Random Forest algorithm is used for labels.

Implementation

The implementation is very similar to the Random Forest algorithm using sklearn's ensemble librarie's AdaBoostClassifier method[8]. We create the same Random Forest models with the Random Forest algorithm and use those models as the base models for AdaBoostClassifier objects which have the same number of estimators as their base models.

5) Logistic Regression: Logistic regression estimates the logistic model parameters. As a library, sklearn.linear-model's LogisticRegression is used [4]. We decided to use different solvers which are newton-cg (Heissian matrix), lbfgs (Limited-memory Broyden-Fletcher-Goldfarb-Shanno), sag (Stochastic Average Gradient descent), saga (extension of sag with L1 regularization) [2]. We will apply these methods for 6 subjects for 4 different forms of dataset (original, PCA, masked, masked PCA).

6) Multilayered Perceptron - MLP: Multilayered Perceptron is a deep learning technique for classifying both binary and multi labeled dataset. Multi-Layer-Perceptron models are neural network models where it uses forward and backward propagation for updating its weights by the mistakes it makes in every iteration. For correcting its mistakes while training in each iteration, it uses 'Adam' loss function in our case for decreasing the loss function by updating the weights according to the result come from backward propagation. By this way, model starts to learn and adapt its weights and optimize itself for creating the best model possible.

Implementation

We have generated 3 different models for each subject. In these 3 different models, we have changed hidden layer number and number of neurons in these hidden layers. For this purpose, we have created $(1000, 750, 500, 250, 100)$, $(1000, 2000, 1000, 100)$, $(200, 200, 200, 200, 100)$ hidden layers in those 3 models. For selecting the best model for test prediction, we have used valida-

tion dataset accuracy results and use the most accurate model on test dataset. Furthermore, we have tested original dataset, original dataset with most significant 100 PCs selected PCA dataset, masked dataset, and masked dataset with most significant 100 PCs selected PCA dataset used. We will also select the most useful dataset by validation accuracy and will use the best dataset version on test dataset and calculate the accuracy.

7) Convolutional Neural Network - CNN: One important Deep Learning algorithm that is used to extract features from image datasets is the Convolutional Neural Network (CNN) architecture proposed by Lecun[15], which uses shared convolutional kernels in order to recognize spatial attributes in the image data. The standard CNN architecture is composed of two parts, first one being the convolutional section where convolution, activation and max-pooling layers are used in conjunction. The convolved information passes through an activation layer (function), since we want to add non-linearity to the learning process. Max-pooling is the layer in which the input is pooled by a kernel so that the algorithm can learn to adjust various sizes of the same image, also it is an important tool in reducing the input dimensions to a smaller dimension, which was useful due to the FMRI data being too large. The second part of the CNN is an MLP structure with fully-connected layers, at the end layer, having number of neurons equal to the number of classes, activated by a softmax function. Through this application, we have treated each FMRI image as a 40x40 image with 64 kernels and made the convolutional kernels move on the XY plane by chose, which we thought would describe the needed features better than the other dimensions. Since, there were many hyperparameters involved, we have used a trial and error approach instead of a search approach to optimize the network, whose end result is as follows. The main motivation behind using CNN was to observe if the conserved temporal data would effect the performance of the model better since the temporal relations between pixels are lost when we apply PCA, use mask or even flatten out the data. We observe three convolution layers with ReLU activations, each with batch normalization right after them and a three layers MLP structure at the end. For training SGD optimizer is used with learning rate 0.0075 and a loss function of cross-entropy.

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 62, 62, 32)	11552
batch_normalization_8 (Batch	(None, 62, 62, 32)	128
max_pooling2d_8 (MaxPooling2	(None, 31, 31, 32)	0
conv2d_9 (Conv2D)	(None, 29, 29, 64)	18496
batch_normalization_9 (Batch	(None, 29, 29, 64)	256
max_pooling2d_9 (MaxPooling2	(None, 14, 14, 64)	0
conv2d_10 (Conv2D)	(None, 12, 12, 64)	36928
batch_normalization_10 (Batc	(None, 12, 12, 64)	256
max_pooling2d_10 (MaxPooling	(None, 6, 6, 64)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_6 (Dense)	(None, 258)	594690
dense_7 (Dense)	(None, 64)	16576
dense_8 (Dense)	(None, 8)	520

Total params: 679,402
Trainable params: 679,082
Non-trainable params: 320

Fig. 1. Optimized CNN Architecture

8) 3D Convolutional Neural Network - 3DCNN: While being an unorthodox technique, as we seen on [17], 3DCNN is used to process and analyze time dependent image data. First published bu Vu et. al.[17], the researchers have created a three dimensional of the LeNet[14] model to classify ADHD disorder as a binary classification task. After observing the results, we have tried the same structure for ourselves and came up with the following structure.

Layer (type)	Output Shape	Param #
conv3d_27 (Conv3D)	(None, 38, 62, 62, 8)	1952
batch_normalization_15 (Batc	(None, 38, 62, 62, 8)	32
max_pooling3d_18 (MaxPooling	(None, 19, 31, 31, 8)	0
flatten_21 (Flatten)	(None, 146072)	0
dense_63 (Dense)	(None, 512)	74789376
dense_64 (Dense)	(None, 128)	65664
dense_65 (Dense)	(None, 8)	1032

Total params: 74,858,056
Trainable params: 74,858,040
Non-trainable params: 16

Fig. 2. Optimized 3D-CNN Structure per Subject

9) Naive Bayes Classifier: Naive Bayes classifier is an estimation model which makes its estimation with respect to prior knowledge it has learnt from train dataset features and labels. Formulation of the Naive Bayes Classifier is as follows:

$$P(class|data) = P(data|class) * P(class)/P(data) \quad (1)$$

As we can observe from equation (eq.1), we are using our pre-known information about classification probabilities in order to predict the result. In equation (eq.1), We obtain the probabilities P(data—class), P(class) and P(data) probabilities from training dataset. By using these prior knowledge, we are able to predict a new data point given to us.

Here, we have chosen a specifically shallow architecture since the number of samples being small led to a high level of overfitting with SGD optimizer, however due to the set being small, validation was also small with the high level of noise in validation accuracy. Hence, we moved towards an Inter-Subject Pattern Analysis style where we trained on 5 of the subject and tested on the 6th. This resulted in a catastrophic way since each person's brain structure was significantly different in terms of specific locations and that experiment yielded no valuable result, the most successful architecture we came up with is as follows. After that we have tried mixing all samples and training on a subsection, however, since we worked on local GPUs (6GB), we were not able to load all the data to GPU space and the experiment failed as a result due to technical difficulties.

Layer (type)	Output Shape	Param #
conv3d (Conv3D)	(None, 38, 62, 62, 16)	3904
batch_normalization (BatchNo	(None, 38, 62, 62, 16)	64
max_pooling3d (MaxPooling3D)	(None, 19, 31, 31, 16)	0
conv3d_1 (Conv3D)	(None, 17, 29, 29, 32)	13856
batch_normalization_1 (Batch	(None, 17, 29, 29, 32)	128
max_pooling3d_1 (MaxPooling3	(None, 8, 14, 14, 32)	0
conv3d_2 (Conv3D)	(None, 6, 12, 12, 64)	55360
spatial_dropout3d (SpatialDr	(None, 6, 12, 12, 64)	0
batch_normalization_2 (Batch	(None, 6, 12, 12, 64)	256
flatten (Flatten)	(None, 55296)	0
dense (Dense)	(None, 512)	28312064
dense_1 (Dense)	(None, 128)	65664
dense_2 (Dense)	(None, 8)	1032

Total params: 28,452,328
Trainable params: 28,452,104
Non-trainable params: 224

Fig. 3. Optimized 3D-CNN Structure for Unified Subjects

III. RESULTS

To sum up, we have created 4 different datasets by using the preprocessed(flattened) original dataset, original dataset with most significant 100 PC values, masked dataset created by using original dataset masking and masked dataset with most significant 100 PC values. We have used 9 different machine learning algorithms in order to classify the FMRI Data. We have evaluated the results of the outcomes of these methods in two seperate groups. The first group contains SVM, MLP, NB, kNN, Logistic Regression, Random Forest, and AdaBoost with Random Forest Base Classifier. The second group consists of CNN and 3DCNN. We have evaluated CNN and 3DCNN separately from the other methods since they use the raw dataset for training rather than the preprocessed(flattened) dataset. The final mean test accuracies for each method in group 1 for every dataset is given in the Figure 4.

On the contrary of what we have hypothesized, MLP worked worse than the other algorithms in most cases while Random Forest worked better than we thought. We thought the deep learning architectures would be more suitable to work with complex data but the other nonlinear algorithms like SVMs worked better than the others. Over-

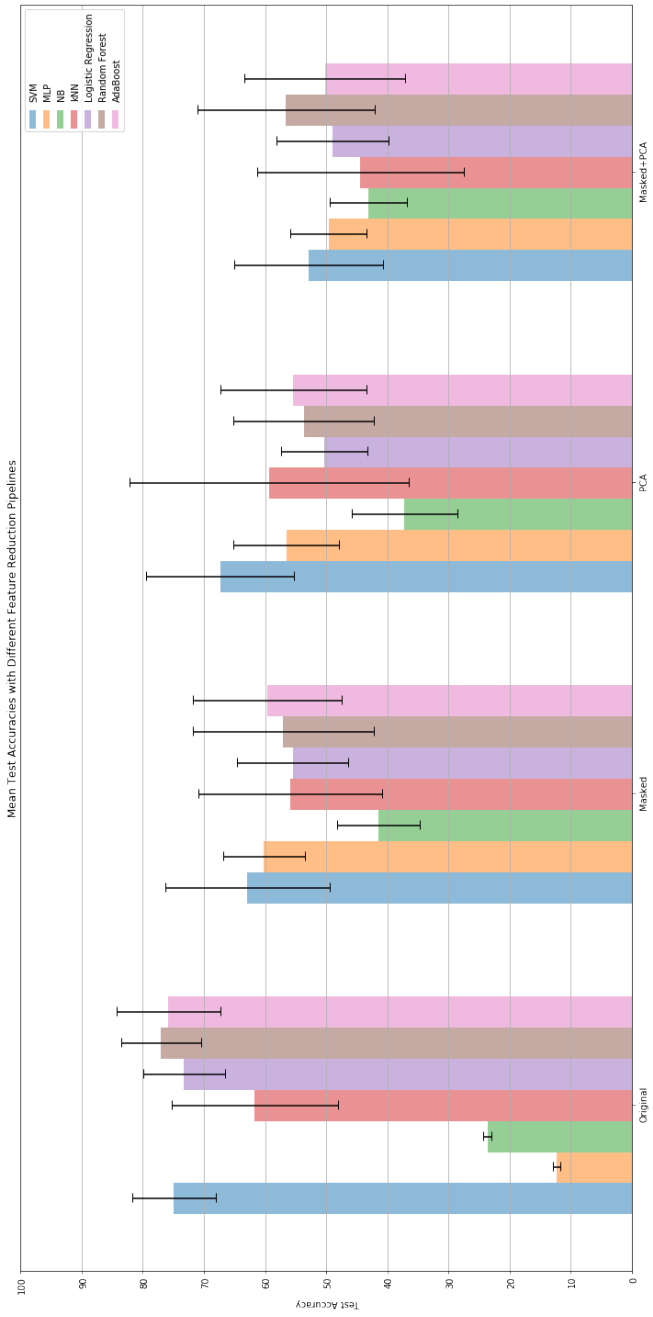

Fig. 4. Mean Test Accuracies and Standard Deviations

all, we see that the original flattened data would not be functional since it contains pixels that are pretty much irrelevant with the temporal ventral cortex area.

IV. DISCUSSION

At the end of the experiment, we have worked through nine different types of classification methods and size different datasets, four of them being prevalent. We can observe that except MLP and NB models, the rest of the models have the best accuracy on the original dataset. The main reason MLP did not perform best in the original dataset is that it gives weight to every pixel of the brain. This creates a lot of unnecessary noise in labeling the samples. Like MLP, NB models are also getting lots of noise in the labeling of samples. These two models are more affected by unnecessary samples with respect to other models. This is the main reason why they have performed better in dimensionally reduced dataset and masked dataset. By this way, they can learn from the most significant features which will decrease the noise of unnecessary features.

Dimensionality reduction method PCA was not useful for the models we have created except MLP and NB models. The main reason for that may be that other than these models, most of the models are trying to separate the samples as much as possible. By this way, they have higher accuracy in dataset which have higher features to evaluate.

When validation accuracies analyzed the best model usually has k value 1. This shows that taking the closest point is an effective way for this dataset. The dataset with the highest accuracy is the original dataset. The reason behind that can be the number of dimensions in the dataset. Original dataset consists of more features compared to others which become advantageous for kNN. Because of using k value as 1, it was expected that it performed best in the original. Since, by this way kNN model can observe the closest sample with respect to features. So, most similar sample in the dataset will classify the test sample.

Overall, the best three models which had eye catching accuracy in our models are SVM, logistic regression and Random Forest. They handled the original dataset near 75% and 80% respectively. Original dataset's mean test accuracy is the highest. The reason behind that can be high dimensionality and most of our models perform better with a big number of features. As a result, we have found that the Random Forest classifier which is an ensemble of various Decision Trees, worked best, and the dataset that was most useful in general was the flattened original data, Here, the result was surprising as the height and depth dimensions were disrupted although row structure was conserved. The only drawback of using this data was that it took extensive space and computational time, the data size was approximately 6.75GB, which was more

than the RAM of capacity of general-use computers. However, we believe that further research in the deep learning algorithms can provide aid in the analysis of the deep learning. The 3D-CNN architecture can be more freely optimized if sufficient GPU is supplied and we believe is a promising method, due to its ability to easily memorize the training data. With a more general and bigger dataset, this algorithm can perform much more successfully. Finally, although we could work with a one vs. all. binary classification setup to show better performance, we have tried to implement a multi-class classification problem.

REFERENCES

[1] S. M., "MachineLearning — KNN using scikit-learn," Medium, 26 October 2018. [Online]. Available: https://towardsdatascience.com/knn-using-scikit-learn-c6bed765be75. [Accessed 28 May 2020].

[2] J. Hale, "Don't Sweat the Solver Stuff," Medium, 27 September 2019. [Online]. Available: https://towardsdatascience.com/dont-sweat-the-solver-stuff-aea7cddc3451. [Accessed 27 May 2020].

[3] J. Haxby and M. Gobbini, "Index of /datasets/haxby2001," Pymvpa, 8 February 2013. [Online]. Available: http://data.pymvpa.org/datasets/haxby2001/. [Accessed 20 May 2020].

[4] S. Li, "Building A Logistic Regression in Python, Step by Step," Medium, 29 September 2017. [Online]. Available: https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8. [Accessed 25 May 2020].

[5] F. Pedregosa, G. Varoquaux and A. Gramfort, "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, 2011.

[6] T. Yiu, "Understanding Random Forest," Medium, 12 June 2019. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2. [Accessed 26 May 2020].

[7] Y. Freund and R. Schapire, A Decision-Theoretic Generalization Of On-Line Learning And An Application To Boosting, 1997.

[8] R. Schapire, Explaining AdaBoost. Machine Learning, 2001, pp. 265-291.

[9] "sklearn.neural-network.MLPClassifier," scikit-learn developers, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural-network.MLPClassifier.html. [Accessed 26 May 2020].

[10] "1.9. Naive Bayes," scikit-learn developers, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/naive-bayes.html. [Accessed 30 May 2020].

[11] "sklearn.model-selection.train-test-split," scikit-learn developers, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model-selection.train-test-split.html. [Accessed 1 June 2020].

[12] "8.3.8. Decoding with ANOVA + SVM: face vs house in the Haxby dataset," Nilearn, 2019. [Online]. Available: https://nilearn.github.io/auto-examples/02-decoding/plot-haxby-anova-svm.htmlsphx-glr-auto-examples-02-decoding-plot-haxby-anova-svm-py. [Accessed 22 May 2020].

[13] A. Mahmoudi, S. Takerkart and F. Regragui, "Multivoxel Pattern Analysis for fMRI Data: A Review," Hindawi, vol. Volume 2012, 2012.

[14] H. Vu, K. Hyun-Chul and J.-H. Lee, "3D convolutional neural network for feature extraction and classification of fMRI volumes," IEEE, 12-14 June 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8423964. [Accessed 26 May 2020].

[15] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," IEEE, vol. November 1998, 1998.

[16] B. C. Wang, C. Thierry and T. Sylvain, "Inter-subject pattern analysis A straightforward and powerful scheme for group-level MVPA," NeuroImage, 2019.

[17] D. Wen, Z. Wei and Y. Zhou, "Deep Learning Methods to Process fMRI Data and Their Application in the Diagnosis of Cognitive Impairment: A Brief Overview and Our Opinion," frontiersin, 2018.

[18] J. V. Haxby, M. I. Gobbini and M. L. Furey, "Distributed and Overlapping Representations of Faces and Objects in Ventral Temporal Cortex," Research Articles - Science, vol. 293, pp. 2425-2430, 2001.

APPENDIX

PreProcess_Save_Data.py

```
import numpy as np
import matplotlib.pyplot as plt
from nilearn.input_data import NiftiMasker
import nibabel as nib
import pandas as pd
import os
from glob import glob
from tqdm import tqdm
from sklearn.decomposition import PCA, NMF

LABELS = ['scissors', 'face', 'cat', 'scrambledpix', 'bottle', 'chair', 'shoe',
'house']
NUM_LABELS = len(LABELS)
DATA_DIR = 'Dataset'

start_indices = np.asarray([6, 21, 35, 49, 63, 78, 92, 106])
all_indices = []
for i in range(9):
    all_indices.append(start_indices + i)
all_indices = np.sort(np.asarray(all_indices).ravel())
print(all_indices)
print(all_indices.shape[0])

SUBJ_COUNT = 6
STIM_LENGTH = 9

func_org = []
func_labels = []

for subj in range(1, SUBJ_COUNT + 1):
    print('Subject ' + str(subj))

    # Get FMRI and Label Directories for Subject
    func_dir = os.path.join(DATA_DIR, 'sub-' + str(subj), 'func\\*.gz')
    func_evt_dir = os.path.join(DATA_DIR, 'sub-' + str(subj), 'func\\*.tsv')
    func_files = glob(func_dir)
    func_evt_dir = glob(func_evt_dir)

    # Temp Lists for Subject
    func_org_temp = []
    func_label_temp = []

    run_cnt = 1
    for file, evt in zip(func_files, func_evt_dir):
        print('Run ', run_cnt)
        run_cnt += 1

        # Original FMRI Data
        func_data = nib.load(file)
        print('Original Data', func_data.get_fdata()[:, :, :, all_indices].shape)
        func_org_temp.append(func_data.get_fdata()[:, :, :, all_indices])

        # Read and Filter Labels for Timepoints
        func_label = pd.read_csv(evt, delimiter='\t')
        filt_labels = []
```

```
        for i in range(NUM_LABELS):
            filt_labels.append([func_label['trial_type'][i * 12]] * STIM_LENGTH)
        filt_labels = np.asarray(filt_labels).flatten()
        print('Label ', filt_labels.shape)
        func_label_temp.append(filt_labels)

    func_org.append(np.asarray(func_org_temp))
    func_labels.append(filt_labels)

func_org = np.asarray(func_org)
func_labels = np.asarray(func_labels)

print('Original ', func_org[0].shape)
print('Labels ', func_labels[0].shape)

np.save('func_org.npy', func_org, allow_pickle=True)
np.save('Labels.npy', func_labels, allow_pickle=True)
```

Data_Prep_Final.py

```
import numpy as np
from sklearn.decomposition import PCA, NMF
from nilearn.input_data import NiftiMasker
import os
import nibabel as nib
from sklearn.model_selection import train_test_split
from glob import glob

func_org = np.load('func_org.npy', allow_pickle=True)
func_labels = np.load('labels.npy', allow_pickle=True)
print(func_org[4].shape)
for i in range(6):
    func_org[i] = np.reshape(func_org[i], (func_org[i].shape[0], 40 * 64 * 64,
72)).astype('float32')
    func_org[i] = np.transpose(func_org[i], (1, 0, 2))
    func_org[i] = np.reshape(func_org[i], (func_org[i].shape[0], -1))
    func_org[i] = func_org[i].T
    print(func_org[i].shape)
print(func_org.shape)

print(func_labels[0].shape)
run_list = [12, 12, 12, 12, 11, 12]

labels = []
i = 0
for runs in run_list:
    temp = np.asarray(([func_labels[i]] * runs)).ravel().T
    print(temp.shape)
    labels.append(temp)
    i += 1
# labels = np.asarray(np.hstack(labels))
labels = np.asarray(labels)
print(labels.shape)

LABELS = ['scissors', 'face', 'cat', 'scrambledpix', 'bottle', 'chair', 'shoe',
'house']
NUM_LABELS = len(LABELS)
```

```
start_indices = np.asarray([6, 21, 35, 49, 63, 78, 92, 106])
all_indices = []
for i in range(9):
    all_indices.append(start_indices + i)
all_indices = np.sort(np.asarray(all_indices).ravel())
print(all_indices)
print(all_indices.shape[0])

DATA_DIR = 'Dataset'
MASK_DIR = 'masks'
SUBJ_COUNT = 6
STIM_LENGTH = 9

masks_list = []
func_masked = []

for subj in range(1, SUBJ_COUNT + 1):
    print('Subject ' + str(subj))

    # Retreive Masks for Subject
    mask_dir = os.path.join(MASK_DIR, 'subj' + str(subj), 'mask4_vt.nii.gz')
    masks_list.append(nib.load(mask_dir))
    masker = NiftiMasker(mask_img=mask_dir, smoothing_fwhm=4,
                         standardize=True, memory="nilearn_cache", memory_level=1)

    # Get FMRI and Label Directories for Subject
    func_dir = os.path.join(DATA_DIR, 'sub-' + str(subj), 'func\\*.gz')
    func_evt_dir = os.path.join(DATA_DIR, 'sub-' + str(subj), 'func\\*.tsv')
    func_files = glob(func_dir)
    func_evt_dir = glob(func_evt_dir)

    # Temp Lists for Subject
    func_masked_temp = []

    run_cnt = 1
    for file, evt in zip(func_files, func_evt_dir):
        print('Run ', run_cnt)
        run_cnt += 1

        # Original FMRI Data
        func_data = nib.load(file)

        # Masked FMRI Data
        func_masked_data = np.asarray(masker.fit_transform(func_data)).T[:,
all_indices]
        print('Masked Data', func_masked_data.shape)
        func_masked_temp.append(func_masked_data)

    func_masked.append(np.asarray(func_masked_temp))

func_masked = np.asarray(func_masked)

for i in range(6):
    print(func_masked[i].shape)
    func_masked[i] = np.transpose(func_masked[i], (0, 2, 1))
    print(func_masked[i].shape)
    func_masked[i] = np.vstack(func_masked[i])
    print(func_masked[i].shape)
```

```
SAVE_DIR = 'data_final'
data_files = ["train", "test", "val"]
if not os.path.exists(SAVE_DIR):
    os.mkdir(SAVE_DIR)

for i in range(6):
    # subject save directory
    SUB_SAVE_DIR = os.path.join(SAVE_DIR, 'subj' + str(i + 1))
    if not os.path.exists(SUB_SAVE_DIR):
        os.mkdir(SUB_SAVE_DIR)

    # split org
    X_train, X_test, y_train, y_test = train_test_split(func_org[i], labels[i],
test_size=0.2, random_state=24)
    X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5,
random_state=24)

    # split masked
    X_train_mask, X_test_mask, y_train_mask, y_test_mask =
train_test_split(func_masked[i], labels[i], test_size=0.2,

random_state=24)
    X_val_mask, X_test_mask, y_val_mask, y_test_mask =
train_test_split(X_test_mask, y_test_mask, test_size=0.5,

random_state=24)

    # PCA construction
    pca = PCA(n_components=100, whiten=True)
    pca.fit(X_train)

    # PCA transforms
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)
    X_val_pca = pca.transform(X_val)

    # PCA masked construction
    pca = PCA(n_components=100, whiten=True)
    pca.fit(X_train_mask)

    # PCA masked transforms
    X_train_pca_mask = pca.transform(X_train_mask)
    X_test_pca_mask = pca.transform(X_test_mask)
    X_val_pca_mask = pca.transform(X_val_mask)

    folders = ['org', 'PCA', 'masked', 'maskedPCA', 'labels']
    for folder in folders:
        TYPE_SAVE_DIR = os.path.join(SUB_SAVE_DIR, folder)
        if not os.path.exists(TYPE_SAVE_DIR):
            os.mkdir(TYPE_SAVE_DIR)

        if (folder == 'org'):
            # save org
            np.save(os.path.join(TYPE_SAVE_DIR, "train_features.npy"), X_train,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "val_features.npy"), X_val,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "test_features.npy"), X_test,
allow_pickle=True)
```

```
        elif (folder == 'PCA'):
            # save PCA
            np.save(os.path.join(TYPE_SAVE_DIR, "train_features.npy"),
X_train_pca, allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "val_features.npy"), X_val_pca,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "test_features.npy"), X_test_pca,
allow_pickle=True)

        elif (folder == 'masked'):
            # save masked
            np.save(os.path.join(TYPE_SAVE_DIR, "train_features"), X_train_mask,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "val_features.npy"), X_val_mask,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "test_features.npy"), X_test_mask,
allow_pickle=True)

        elif (folder == 'maskedPCA'):
            # save masked+PCA
            np.save(os.path.join(TYPE_SAVE_DIR, "train_features.npy"),
X_train_pca_mask, allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "val_features.npy"),
X_val_pca_mask, allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "test_features.npy"),
X_test_pca_mask, allow_pickle=True)

        elif (folder == 'labels'):
            # save labels
            np.save(os.path.join(TYPE_SAVE_DIR, "train_labels.npy"), y_train,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "val_labels.npy"), y_val,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "test_labels.npy"), y_test,
allow_pickle=True)
```

kNN.py

```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

DATA_DIR = 'data_final'

def accuracy(y_pred, y_real):
    num_true = np.sum(y_pred == y_real)
    return num_true / y_real.shape[0]

def barplot(k_acc, k, string):
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1])
    ax.bar(k, k_acc)
    plt.xlabel('k values')
```

```
    plt.ylabel('Validation Accuracies')
    plt.title(string)
    plt.show()

# PCA training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'PCA')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    k_range = range(1, 26)
    best_val_acc = 0
    k_acc = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn = knn.fit(X_train, y_train)
        predictions = knn.predict(X_val)
        val_acc = accuracy(predictions, y_val)
        k_acc.append(val_acc)

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = knn

    best_models.append(best_model)
    barplot(k_acc, k_range, 'Validation Accuracies for PCA for Subject ' + str(i))
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_pca = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'PCA')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_pca.append(accuracy(predictions, y_test))
```

```
print('Test Accuracies for Each subject', test_accuracies_pca)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_pca),
np.std(test_accuracies_pca)))

# Masked training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'masked')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    k_range = range(1, 26)
    best_val_acc = 0
    k_acc = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn = knn.fit(X_train, y_train)
        predictions = knn.predict(X_val)
        val_acc = accuracy(predictions, y_val)
        k_acc.append(val_acc)

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = knn

    best_models.append(best_model)
    barplot(k_acc, k_range, 'Validation Accuracies for Masked for Subject ' +
str(i))
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_masked = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'masked')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
```

```
    test_accuracies_masked.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_masked)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_masked),
np.std(test_accuracies_masked)))

# Masked-PCA training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'maskedPCA')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    k_range = range(1, 26)
    best_val_acc = 0
    k_acc = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn = knn.fit(X_train, y_train)
        predictions = knn.predict(X_val)
        val_acc = accuracy(predictions, y_val)
        k_acc.append(val_acc)

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = knn

    best_models.append(best_model)
    barplot(k_acc, k_range, 'Validation Accuracies for Masked PCA for Subject ' +
str(i))
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_maskedpca = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'maskedPCA')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)
```

```
    predictions = model.predict(X_test)
    test_accuracies_maskedpca.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_maskedpca)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_maskedpca),
np.std(test_accuracies_maskedpca)))

# Original training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'org')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    k_range = range(1, 26)
    best_val_acc = 0
    k_acc = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn = knn.fit(X_train, y_train)
        predictions = knn.predict(X_val)
        val_acc = accuracy(predictions, y_val)
        k_acc.append(val_acc)

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = knn

    best_models.append(best_model)
    barplot(k_acc, k_range, 'Validation Accuracies for Original Data for Subject '
+ str(i))
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_org = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'org')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)
```

```
    predictions = model.predict(X_test)
    test_accuracies_org.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_org)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_org),
np.std(test_accuracies_org)))

# plot means and stds
means = 100 * np.asarray([np.mean(test_accuracies_org),
np.mean(test_accuracies_masked), np.mean(test_accuracies_pca),
                          np.mean(test_accuracies_maskedpca)])
stds = 100 * np.asarray([np.std(test_accuracies_org),
np.std(test_accuracies_masked), np.std(test_accuracies_pca),
                         np.std(test_accuracies_maskedpca)])
reducs = ['Original', 'Masked', 'PCA', 'Masked+PCA']

fig, ax = plt.subplots()
ax.bar(np.arange(4), means, yerr=stds, align='center', alpha=0.5, ecolor='black',
capsize=10)
ax.set_yticks(np.arange(0, 105, 10))
ax.set_ylabel('Test Accuracy')
ax.set_xticks(np.arange(4))
ax.set_xticklabels(reducs)
ax.set_title('Mean Test Accuracies for knn with different k values')
ax.grid(True)

# Save the figure and show
plt.tight_layout()
plt.show()
```

SVM_Subject.py

```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn import svm
from sklearn.model_selection import train_test_split

DATA_DIR = 'data_final'

def accuracy(y_pred, y_real):
    num_true = np.sum(y_pred == y_real)
    return num_true / y_real.shape[0]

# PCA training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'PCA')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
```

```
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    KERNEL_LIST = ['linear', 'poly', 'rbf', 'sigmoid']
    best_val_acc = 0
    for kernel in KERNEL_LIST:
        classifier = svm.SVC(C=5, kernel=kernel, decision_function_shape='ovr',
gamma='auto')
        predictions = classifier.fit(X_train, y_train).predict(X_val)
        val_acc = accuracy(predictions, y_val)
        print('Validation Accuracy for %s Kernel with One vs Rest
Classification: %f' % (kernel, val_acc))

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = classifier

    best_models.append(best_model)
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_pca = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'PCA')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_pca.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_pca)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_pca),
np.std(test_accuracies_pca)))

# Masked training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'masked')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
```

```
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    KERNEL_LIST = ['linear', 'poly', 'rbf', 'sigmoid']
    best_val_acc = 0
    for kernel in KERNEL_LIST:
        classifier = svm.SVC(C=5, kernel=kernel, decision_function_shape='ovr',
gamma='auto')
        predictions = classifier.fit(X_train, y_train).predict(X_val)
        val_acc = accuracy(predictions, y_val)
        print('Validation Accuracy for %s Kernel with One vs Rest
Classification: %f' % (kernel, val_acc))

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = classifier

    best_models.append(best_model)
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_masked = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'masked')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_masked.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_masked)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_masked),
np.std(test_accuracies_masked)))

# Masked-PCA training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'maskedPCA')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)
```

```
    KERNEL_LIST = ['linear', 'poly', 'rbf', 'sigmoid']
    best_val_acc = 0
    for kernel in KERNEL_LIST:
        classifier = svm.SVC(C=5, kernel=kernel, decision_function_shape='ovr',
gamma='auto')
        predictions = classifier.fit(X_train, y_train).predict(X_val)
        val_acc = accuracy(predictions, y_val)
        print('Validation Accuracy for %s Kernel with One vs Rest
Classification: %f' % (kernel, val_acc))

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = classifier

    best_models.append(best_model)
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_maskedpca = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'maskedPCA')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_maskedpca.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_maskedpca)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_maskedpca),
np.std(test_accuracies_maskedpca)))

# Original training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'org')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)
```

```
    KERNEL_LIST = ['linear', 'poly', 'rbf', 'sigmoid']
    best_val_acc = 0
    for kernel in KERNEL_LIST:
        classifier = svm.SVC(C=5, kernel=kernel, decision_function_shape='ovr',
gamma='auto')
        predictions = classifier.fit(X_train, y_train).predict(X_val)
        val_acc = accuracy(predictions, y_val)
        print('Validation Accuracy for %s Kernel with One vs Rest
Classification: %f' % (kernel, val_acc))

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = classifier

    best_models.append(best_model)
print('\n\n-----------------------------------------\n\n')
# PCA test
test_accuracies_org = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'org')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_org.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_org)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_org),
np.std(test_accuracies_org)))

# plot means and stds
means = 100 * np.asarray([np.mean(test_accuracies_org),
np.mean(test_accuracies_masked), np.mean(test_accuracies_pca),
                          np.mean(test_accuracies_maskedpca)])
stds = 100 * np.asarray([np.std(test_accuracies_org),
np.std(test_accuracies_masked), np.std(test_accuracies_pca),
                         np.std(test_accuracies_maskedpca)])
reducs = ['Original', 'Masked', 'PCA', 'Masked+PCA']

fig, ax = plt.subplots()
ax.bar(np.arange(4), means, yerr=stds, align='center', alpha=0.5, ecolor='black',
capsize=10)
ax.set_yticks(np.arange(0, 105, 10))
ax.set_ylabel('Test Accuracy')
ax.set_xticks(np.arange(4))
ax.set_xticklabels(reducs)
ax.set_title('Mean Test Accuracies for SVM \nwith Different Feature Reduction
Pipelines')
ax.grid(True)

# Save the figure and show
```

```
plt.tight_layout()
plt.show()
```

Random_Forest_Final.py

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import os
import matplotlib.pyplot as plt

def create_dictionary(labels, names):
    dictLabels = np.zeros(labels.shape)
    for i in range(len(labels)):
        val = names.index(labels[i])
        # val = np.where(names == str(labels[i][j]))
        dictLabels[i] = val
    return dictLabels

names = ['scissors', 'face', 'cat', 'scrambledpix', 'bottle', 'chair', 'shoe',
'house']
FOLDER_DIR = 'data_final'

# PCA for all subjects
FINAL_ACCS1 = []
for i in range(6):
    SUB_DIR = os.path.join(FOLDER_DIR, 'subj' + str(i + 1))
    PCA_DIR = os.path.join(SUB_DIR, 'PCA')
    LABELS_DIR = os.path.join(SUB_DIR, 'labels')

    trainf_dir = os.path.join(PCA_DIR, 'train_features.npy')
    testf_dir = os.path.join(PCA_DIR, 'test_features.npy')
    valf_dir = os.path.join(PCA_DIR, 'val_features.npy')

    trainl_dir = os.path.join(LABELS_DIR, 'train_labels.npy')
    testl_dir = os.path.join(LABELS_DIR, 'test_labels.npy')
    vall_dir = os.path.join(LABELS_DIR, 'val_labels.npy')

    train_f = np.load(trainf_dir, allow_pickle=True)
    test_f = np.load(testf_dir, allow_pickle=True)
    val_f = np.load(valf_dir, allow_pickle=True)
    train_l = np.load(trainl_dir, allow_pickle=True)
    test_l = np.load(testl_dir, allow_pickle=True)
    val_l = np.load(vall_dir, allow_pickle=True)

    train_l = create_dictionary(np.asarray(train_l), names)
    test_l = create_dictionary(np.asarray(test_l), names)
    val_l = create_dictionary(np.asarray(val_l), names)

    # Start Random Forest Training
    # Forest 1
    forest1 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    forest1.fit(train_f, train_l)
    pred_valid1 = forest1.predict(val_f)
    acc1 = metrics.accuracy_score(val_l, pred_valid1)
    print("Accuracy of subject {}, with Forest 1: ".format(str(i + 1)), acc1)
```

```
# Forest 2
forest2 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=100, max_features='sqrt',
                                 max_depth=8, min_samples_leaf=4,
                                 min_samples_split=2, random_state=1)
forest2.fit(train_f, train_l)
pred_valid2 = forest2.predict(val_f)
acc2 = metrics.accuracy_score(val_l, pred_valid2)
print("Accuracy of subject {}, with Forest 2: ".format(str(i + 1)), acc2)

# Forest 3
forest3 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=50, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest3.fit(train_f, train_l)
pred_valid3 = forest3.predict(val_f)
acc3 = metrics.accuracy_score(val_l, pred_valid3)
print("Accuracy of subject {}, with Forest 3: ".format(str(i + 1)), acc3)

# Forest 4
forest4 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)
forest4.fit(train_f, train_l)
pred_valid4 = forest4.predict(val_f)
acc4 = metrics.accuracy_score(val_l, pred_valid4)
print("Accuracy of subject {}, with Forest 4: ".format(str(i + 1)), acc4)

# Forest 5
forest5 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=200, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)
forest5.fit(train_f, train_l)
pred_valid5 = forest5.predict(val_f)
acc5 = metrics.accuracy_score(val_l, pred_valid5)
print("Accuracy of subject {}, with Forest 5: ".format(str(i + 1)), acc5)

# Forest 6
forest6 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest6.fit(train_f, train_l)
pred_valid6 = forest6.predict(val_f)
acc6 = metrics.accuracy_score(val_l, pred_valid6)
print("Accuracy of subject {}, with Forest 6: ".format(str(i + 1)), acc6)

# Forest 7
forest7 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest7.fit(train_f, train_l)
pred_valid7 = forest7.predict(val_f)
acc7 = metrics.accuracy_score(val_l, pred_valid7)
```

```
    print("Accuracy of subject {}, with Forest 7: ".format(str(i + 1)), acc7)

    # Try the best validation model on the test set
    Forests = [forest1, forest2, forest3, forest4, forest5, forest6, forest7]
    accuracies = np.asarray([acc1, acc2, acc3, acc4, acc5, acc6, acc7])
    max_acc = np.argmax(accuracies)
    best_forest = Forests[max_acc]

    # Try on test set
    pred_test = best_forest.predict(test_f)
    test_acc = metrics.accuracy_score(test_l, pred_test)
    print("Accuracy of subject {}, on test with Forest {}: ".format(str(i + 1),
str(max_acc + 1)), test_acc)
    FINAL_ACCS1.append(test_acc)
final1 = np.asarray(FINAL_ACCS1)
print("Mean of Final PCA Accuracies: " + str(np.mean(final1)))

# Masked Data for all subjects
FINAL_ACCS2 = []
for i in range(6):
    SUB_DIR = os.path.join(FOLDER_DIR, 'subj' + str(i + 1))
    masked_DIR = os.path.join(SUB_DIR, 'masked')
    LABELS_DIR = os.path.join(SUB_DIR, 'labels')

    trainf_dir = os.path.join(masked_DIR, 'train_features.npy')
    testf_dir = os.path.join(masked_DIR, 'test_features.npy')
    valf_dir = os.path.join(masked_DIR, 'val_features.npy')

    trainl_dir = os.path.join(LABELS_DIR, 'train_labels.npy')
    testl_dir = os.path.join(LABELS_DIR, 'test_labels.npy')
    vall_dir = os.path.join(LABELS_DIR, 'val_labels.npy')

    train_f = np.load(trainf_dir, allow_pickle=True)
    test_f = np.load(testf_dir, allow_pickle=True)
    val_f = np.load(valf_dir, allow_pickle=True)
    train_l = np.load(trainl_dir, allow_pickle=True)
    test_l = np.load(testl_dir, allow_pickle=True)
    val_l = np.load(vall_dir, allow_pickle=True)

    train_l = create_dictionary(np.asarray(train_l), names)
    test_l = create_dictionary(np.asarray(test_l), names)
    val_l = create_dictionary(np.asarray(val_l), names)

    # Start Random Forest Training
    # Forest 1
    forest1 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    forest1.fit(train_f, train_l)
    pred_valid1 = forest1.predict(val_f)
    acc1 = metrics.accuracy_score(val_l, pred_valid1)
    print("Accuracy of subject {}, with Forest 1: ".format(str(i + 1)), acc1)

    # Forest 2
    forest2 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=8, min_samples_leaf=4,
                                     min_samples_split=2, random_state=1)
```

```
forest2.fit(train_f, train_l)
pred_valid2 = forest2.predict(val_f)
acc2 = metrics.accuracy_score(val_l, pred_valid2)
print("Accuracy of subject {}, with Forest 2: ".format(str(i + 1)), acc2)

# Forest 3
forest3 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=50, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest3.fit(train_f, train_l)
pred_valid3 = forest3.predict(val_f)
acc3 = metrics.accuracy_score(val_l, pred_valid3)
print("Accuracy of subject {}, with Forest 3: ".format(str(i + 1)), acc3)

# Forest 4
forest4 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)
forest4.fit(train_f, train_l)
pred_valid4 = forest4.predict(val_f)
acc4 = metrics.accuracy_score(val_l, pred_valid4)
print("Accuracy of subject {}, with Forest 4: ".format(str(i + 1)), acc4)

# Forest 5
forest5 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=200, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)
forest5.fit(train_f, train_l)
pred_valid5 = forest5.predict(val_f)
acc5 = metrics.accuracy_score(val_l, pred_valid5)
print("Accuracy of subject {}, with Forest 5: ".format(str(i + 1)), acc5)

# Forest 6
forest6 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest6.fit(train_f, train_l)
pred_valid6 = forest6.predict(val_f)
acc6 = metrics.accuracy_score(val_l, pred_valid6)
print("Accuracy of subject {}, with Forest 6: ".format(str(i + 1)), acc6)

# Forest 7
forest7 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest7.fit(train_f, train_l)
pred_valid7 = forest7.predict(val_f)
acc7 = metrics.accuracy_score(val_l, pred_valid7)
print("Accuracy of subject {}, with Forest 7: ".format(str(i + 1)), acc7)

# Try the best validation model on the test set
Forests = [forest1, forest2, forest3, forest4, forest5, forest6, forest7]
accuracies = np.asarray([acc1, acc2, acc3, acc4, acc5, acc6, acc7])
max_acc = np.argmax(accuracies)
```

```
    best_forest = Forests[max_acc]

    # Try on test set
    pred_test = best_forest.predict(test_f)
    test_acc = metrics.accuracy_score(test_l, pred_test)
    print("Accuracy of subject {}, on test with Forest {}: ".format(str(i + 1),
str(max_acc + 1)), test_acc)
    FINAL_ACCS2.append(test_acc)
final2 = np.asarray(FINAL_ACCS2)
print("Mean of Final Masked Accuracies: " + str(np.mean(final2)))

# Masked PCA Data for all subjects
FINAL_ACCS3 = []
for i in range(6):
    SUB_DIR = os.path.join(FOLDER_DIR, 'subj' + str(i + 1))
    maskedPCA_DIR = os.path.join(SUB_DIR, 'maskedPCA')
    LABELS_DIR = os.path.join(SUB_DIR, 'labels')

    trainf_dir = os.path.join(maskedPCA_DIR, 'train_features.npy')
    testf_dir = os.path.join(maskedPCA_DIR, 'test_features.npy')
    valf_dir = os.path.join(maskedPCA_DIR, 'val_features.npy')

    trainl_dir = os.path.join(LABELS_DIR, 'train_labels.npy')
    testl_dir = os.path.join(LABELS_DIR, 'test_labels.npy')
    vall_dir = os.path.join(LABELS_DIR, 'val_labels.npy')

    train_f = np.load(trainf_dir, allow_pickle=True)
    test_f = np.load(testf_dir, allow_pickle=True)
    val_f = np.load(valf_dir, allow_pickle=True)
    train_l = np.load(trainl_dir, allow_pickle=True)
    test_l = np.load(testl_dir, allow_pickle=True)
    val_l = np.load(vall_dir, allow_pickle=True)

    train_l = create_dictionary(np.asarray(train_l), names)
    test_l = create_dictionary(np.asarray(test_l), names)
    val_l = create_dictionary(np.asarray(val_l), names)

    # Start Random Forest Training
    # Forest 1
    forest1 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    forest1.fit(train_f, train_l)
    pred_valid1 = forest1.predict(val_f)
    acc1 = metrics.accuracy_score(val_l, pred_valid1)
    print("Accuracy of subject {}, with Forest 1: ".format(str(i + 1)), acc1)

    # Forest 2
    forest2 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=8, min_samples_leaf=4,
                                     min_samples_split=2, random_state=1)
    forest2.fit(train_f, train_l)
    pred_valid2 = forest2.predict(val_f)
    acc2 = metrics.accuracy_score(val_l, pred_valid2)
    print("Accuracy of subject {}, with Forest 2: ".format(str(i + 1)), acc2)

    # Forest 3
```

```
forest3 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=50, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest3.fit(train_f, train_l)
pred_valid3 = forest3.predict(val_f)
acc3 = metrics.accuracy_score(val_l, pred_valid3)
print("Accuracy of subject {}, with Forest 3: ".format(str(i + 1)), acc3)

# Forest 4
forest4 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)
forest4.fit(train_f, train_l)
pred_valid4 = forest4.predict(val_f)
acc4 = metrics.accuracy_score(val_l, pred_valid4)
print("Accuracy of subject {}, with Forest 4: ".format(str(i + 1)), acc4)

# Forest 5
forest5 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=200, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)
forest5.fit(train_f, train_l)
pred_valid5 = forest5.predict(val_f)
acc5 = metrics.accuracy_score(val_l, pred_valid5)
print("Accuracy of subject {}, with Forest 5: ".format(str(i + 1)), acc5)

# Forest 6
forest6 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest6.fit(train_f, train_l)
pred_valid6 = forest6.predict(val_f)
acc6 = metrics.accuracy_score(val_l, pred_valid6)
print("Accuracy of subject {}, with Forest 6: ".format(str(i + 1)), acc6)

# Forest 7
forest7 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
forest7.fit(train_f, train_l)
pred_valid7 = forest7.predict(val_f)
acc7 = metrics.accuracy_score(val_l, pred_valid7)
print("Accuracy of subject {}, with Forest 7: ".format(str(i + 1)), acc7)

# Try the best validation model on the test set
Forests = [forest1, forest2, forest3, forest4, forest5, forest6, forest7]
accuracies = np.asarray([acc1, acc2, acc3, acc4, acc5, acc6, acc7])
max_acc = np.argmax(accuracies)
best_forest = Forests[max_acc]

# Try on test set
pred_test = best_forest.predict(test_f)
test_acc = metrics.accuracy_score(test_l, pred_test)
print("Accuracy of subject {}, on test with Forest {}: ".format(str(i + 1),
```

```
str(max_acc + 1)), test_acc)
    FINAL_ACCS3.append(test_acc)
final3 = np.asarray(FINAL_ACCS3)
print("Mean of Final Masked PCA Accuracies: " + str(np.mean(final3)))

# ORG Data for all subjects
FINAL_ACCS4 = []
for i in range(6):
    SUB_DIR = os.path.join(FOLDER_DIR, 'subj' + str(i + 1))
    ORG_DIR = os.path.join(SUB_DIR, 'org')
    LABELS_DIR = os.path.join(SUB_DIR, 'labels')

    trainf_dir = os.path.join(ORG_DIR, 'train_features.npy')
    testf_dir = os.path.join(ORG_DIR, 'test_features.npy')
    valf_dir = os.path.join(ORG_DIR, 'val_features.npy')

    trainl_dir = os.path.join(LABELS_DIR, 'train_labels.npy')
    testl_dir = os.path.join(LABELS_DIR, 'test_labels.npy')
    vall_dir = os.path.join(LABELS_DIR, 'val_labels.npy')

    train_f = np.load(trainf_dir, allow_pickle=True)
    test_f = np.load(testf_dir, allow_pickle=True)
    val_f = np.load(valf_dir, allow_pickle=True)
    train_l = np.load(trainl_dir, allow_pickle=True)
    test_l = np.load(testl_dir, allow_pickle=True)
    val_l = np.load(vall_dir, allow_pickle=True)

    train_l = create_dictionary(np.asarray(train_l), names)
    test_l = create_dictionary(np.asarray(test_l), names)
    val_l = create_dictionary(np.asarray(val_l), names)

    # Start Random Forest Training
    # Forest 1
    forest1 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    forest1.fit(train_f, train_l)
    pred_valid1 = forest1.predict(val_f)
    acc1 = metrics.accuracy_score(val_l, pred_valid1)
    print("Accuracy of subject {}, with Forest 1: ".format(str(i + 1)), acc1)

    # Forest 2
    forest2 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=8, min_samples_leaf=4,
                                     min_samples_split=2, random_state=1)
    forest2.fit(train_f, train_l)
    pred_valid2 = forest2.predict(val_f)
    acc2 = metrics.accuracy_score(val_l, pred_valid2)
    print("Accuracy of subject {}, with Forest 2: ".format(str(i + 1)), acc2)

    # Forest 3
    forest3 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=50, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    forest3.fit(train_f, train_l)
    pred_valid3 = forest3.predict(val_f)
```

```
    acc3 = metrics.accuracy_score(val_l, pred_valid3)
    print("Accuracy of subject {}, with Forest 3: ".format(str(i + 1)), acc3)

    # Forest 4
    forest4 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=500, max_features='sqrt',
                                     max_depth=30, min_samples_leaf=2,
                                     min_samples_split=4, random_state=1)
    forest4.fit(train_f, train_l)
    pred_valid4 = forest4.predict(val_f)
    acc4 = metrics.accuracy_score(val_l, pred_valid4)
    print("Accuracy of subject {}, with Forest 4: ".format(str(i + 1)), acc4)

    # Forest 5
    forest5 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=200, max_features='sqrt',
                                     max_depth=30, min_samples_leaf=2,
                                     min_samples_split=4, random_state=1)
    forest5.fit(train_f, train_l)
    pred_valid5 = forest5.predict(val_f)
    acc5 = metrics.accuracy_score(val_l, pred_valid5)
    print("Accuracy of subject {}, with Forest 5: ".format(str(i + 1)), acc5)

    # Forest 6
    forest6 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=500, max_features='sqrt',
                                     max_depth=30, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    forest6.fit(train_f, train_l)
    pred_valid6 = forest6.predict(val_f)
    acc6 = metrics.accuracy_score(val_l, pred_valid6)
    print("Accuracy of subject {}, with Forest 6: ".format(str(i + 1)), acc6)

    # Forest 7
    forest7 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=500, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    forest7.fit(train_f, train_l)
    pred_valid7 = forest7.predict(val_f)
    acc7 = metrics.accuracy_score(val_l, pred_valid7)
    print("Accuracy of subject {}, with Forest 7: ".format(str(i + 1)), acc7)

    # Try the best validation model on the test set
    Forests = [forest1, forest2, forest3, forest4, forest5, forest6, forest7]
    accuracies = np.asarray([acc1, acc2, acc3, acc4, acc5, acc6, acc7])
    max_acc = np.argmax(accuracies)
    best_forest = Forests[max_acc]

    # Try on test set
    pred_test = best_forest.predict(test_f)
    test_acc = metrics.accuracy_score(test_l, pred_test)
    print("Accuracy of subject {}, on test with Forest {}: ".format(str(i + 1),
str(max_acc + 1)), test_acc)
    FINAL_ACCS4.append(test_acc)
final4 = np.asarray(FINAL_ACCS4)
print("Mean of Final ORG Accuracies: " + str(np.mean(final4)))

plt.figure()
```

```
objects = ('Original', 'Masked', 'PCA', 'Masked PCA')
y_pos = np.arange(len(objects))
# Calculate Mean
mean1 = np.mean(final1)
mean2 = np.mean(final2)
mean3 = np.mean(final3)
mean4 = np.mean(final4)
# Calculate Standard deviation
std1 = np.std(final1)
std2 = np.std(final2)
std3 = np.std(final3)
std4 = np.std(final4)
performance = [mean4, mean2, mean1, mean3]
error = [std4, std2, std1, std3]

plt.bar(y_pos, performance, yerr=error, align='center', alpha=0.5, capsize=10)
plt.grid(True)
plt.xticks(y_pos, objects)
plt.ylabel('Test Accuracy')
plt.title('Mean Test Accuracies for Random Forest')

plt.show()

print("Random Forest:")
print("1-Org data mean test accuracy and std: mean: {} std: {}".format(str(mean4),
str(std4)))
print("")
print("2-pca data mean test accuracy and std: mean: {} std: {}".format(str(mean1),
str(std1)))
print("")
print("3-masked data mean test accuracy and std: mean: {} std:
{}".format(str(mean2), str(std2)))
print("")
print("4-masked pca data mean test accuracy and std: mean: {} std:
{}".format(str(mean3), str(std3)))

methods = [final1, final2, final3, final4]
datas = ['PCA', 'Masked', 'Masked PCA', 'Original']
i = 0
for final in methods:
    plt.figure()
    plt.plot(final)
    plt.xlabel('Subjects')
    plt.xticks(np.arange(len(final)), ['1', '2', '3', '4', '5', '6'])
    plt.ylabel('Accuracy on Test')
    plt.title('The Validation Accuracies of Random Forest\n on {}
Data'.format(datas[i]))
    plt.grid(True)
    plt.show()
    i = i + 1
```

AdaBoost_RandomForest.py

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import os
```

```
import matplotlib.pyplot as plt

def create_dictionary(labels, names):
    dictLabels = np.zeros(labels.shape)
    for i in range(len(labels)):
        val = names.index(labels[i])
        # val = np.where(names == str(labels[i][j]))
        dictLabels[i] = val
    return dictLabels

names = ['scissors', 'face', 'cat', 'scrambledpix', 'bottle', 'chair', 'shoe',
'house']
FOLDER_DIR = 'data_final'

# PCA for all subjects
FINAL_ACCS1 = []
for i in range(6):
    SUB_DIR = os.path.join(FOLDER_DIR, 'subj' + str(i + 1))
    PCA_DIR = os.path.join(SUB_DIR, 'PCA')
    LABELS_DIR = os.path.join(SUB_DIR, 'labels')

    trainf_dir = os.path.join(PCA_DIR, 'train_features.npy')
    testf_dir = os.path.join(PCA_DIR, 'test_features.npy')
    valf_dir = os.path.join(PCA_DIR, 'val_features.npy')

    trainl_dir = os.path.join(LABELS_DIR, 'train_labels.npy')
    testl_dir = os.path.join(LABELS_DIR, 'test_labels.npy')
    vall_dir = os.path.join(LABELS_DIR, 'val_labels.npy')

    train_f = np.load(trainf_dir, allow_pickle=True)
    test_f = np.load(testf_dir, allow_pickle=True)
    val_f = np.load(valf_dir, allow_pickle=True)
    train_l = np.load(trainl_dir, allow_pickle=True)
    test_l = np.load(testl_dir, allow_pickle=True)
    val_l = np.load(vall_dir, allow_pickle=True)

    train_l = create_dictionary(np.asarray(train_l), names)
    test_l = create_dictionary(np.asarray(test_l), names)
    val_l = create_dictionary(np.asarray(val_l), names)

    # Start Random Forest with AdaBoost Training
    # Forest 1
    forest1 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)

    ada1 = AdaBoostClassifier(base_estimator=forest1,
                              n_estimators=100, random_state=1)

    ada1.fit(train_f, train_l)
    pred_valid1 = ada1.predict(val_f)
    acc1 = metrics.accuracy_score(val_l, pred_valid1)
    print("Accuracy of subject {}, with Forest 1: ".format(str(i + 1)), acc1)

    # Forest 2
    forest2 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=8, min_samples_leaf=4,
```

```
                                    min_samples_split=2, random_state=1)

ada2 = AdaBoostClassifier(base_estimator=forest2,
                          n_estimators=100, random_state=1)

ada2.fit(train_f, train_l)
pred_valid2 = ada2.predict(val_f)
acc2 = metrics.accuracy_score(val_l, pred_valid2)
print("Accuracy of subject {}, with Forest 2: ".format(str(i + 1)), acc2)

# Forest 3
forest3 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=50, max_features='sqrt',
                                    max_depth=20, min_samples_leaf=4,
                                    min_samples_split=4, random_state=1)

ada3 = AdaBoostClassifier(base_estimator=forest3,
                          n_estimators=50, random_state=1)

ada3.fit(train_f, train_l)
pred_valid3 = ada3.predict(val_f)
acc3 = metrics.accuracy_score(val_l, pred_valid3)
print("Accuracy of subject {}, with Forest 3: ".format(str(i + 1)), acc3)

# Forest 4
forest4 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=500, max_features='sqrt',
                                    max_depth=30, min_samples_leaf=2,
                                    min_samples_split=4, random_state=1)

ada4 = AdaBoostClassifier(base_estimator=forest4,
                          n_estimators=500, random_state=1)

ada4.fit(train_f, train_l)
pred_valid4 = ada4.predict(val_f)
acc4 = metrics.accuracy_score(val_l, pred_valid4)
print("Accuracy of subject {}, with Forest 4: ".format(str(i + 1)), acc4)

# Forest 5
forest5 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=200, max_features='sqrt',
                                    max_depth=30, min_samples_leaf=2,
                                    min_samples_split=4, random_state=1)

ada5 = AdaBoostClassifier(base_estimator=forest5,
                          n_estimators=200, random_state=1)

ada5.fit(train_f, train_l)
pred_valid5 = ada5.predict(val_f)
acc5 = metrics.accuracy_score(val_l, pred_valid5)
print("Accuracy of subject {}, with Forest 5: ".format(str(i + 1)), acc5)

# Forest 6
forest6 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=500, max_features='sqrt',
                                    max_depth=30, min_samples_leaf=4,
                                    min_samples_split=4, random_state=1)
ada6 = AdaBoostClassifier(base_estimator=forest6,
                          n_estimators=500, random_state=1)
```

```
    ada6.fit(train_f, train_l)
    pred_valid6 = ada6.predict(val_f)
    acc6 = metrics.accuracy_score(val_l, pred_valid6)
    print("Accuracy of subject {}, with Forest 6: ".format(str(i + 1)), acc6)

    # Forest 7
    forest7 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=500, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    ada7 = AdaBoostClassifier(base_estimator=forest7,
                              n_estimators=500, random_state=1)
    ada7.fit(train_f, train_l)
    pred_valid7 = ada7.predict(val_f)
    acc7 = metrics.accuracy_score(val_l, pred_valid7)
    print("Accuracy of subject {}, with Forest 7: ".format(str(i + 1)), acc7)

    # Try the best validation model on the test set
    Adas = [ada1, ada2, ada3, ada4, ada5, ada6, ada7]
    accuracies = np.asarray([acc1, acc2, acc3, acc4, acc5, acc6, acc7])
    max_acc = np.argmax(accuracies)
    best_forest = Adas[max_acc]

    # Try on test set
    pred_test = best_forest.predict(test_f)
    test_acc = metrics.accuracy_score(test_l, pred_test)
    print("Accuracy of subject {}, on test with AdaBoosted Forest {}:
".format(str(i + 1), str(max_acc + 1)), test_acc)
    FINAL_ACCS1.append(test_acc)
final1 = np.asarray(FINAL_ACCS1)
print("Mean of Final PCA Data Accuracies: " + str(np.mean(final1)))

# Masked data for all subjects
FINAL_ACCS2 = []
for i in range(6):
    SUB_DIR = os.path.join(FOLDER_DIR, 'subj' + str(i + 1))
    masked_DIR = os.path.join(SUB_DIR, 'masked')
    LABELS_DIR = os.path.join(SUB_DIR, 'labels')

    trainf_dir = os.path.join(masked_DIR, 'train_features.npy')
    testf_dir = os.path.join(masked_DIR, 'test_features.npy')
    valf_dir = os.path.join(masked_DIR, 'val_features.npy')

    trainl_dir = os.path.join(LABELS_DIR, 'train_labels.npy')
    testl_dir = os.path.join(LABELS_DIR, 'test_labels.npy')
    vall_dir = os.path.join(LABELS_DIR, 'val_labels.npy')

    train_f = np.load(trainf_dir, allow_pickle=True)
    test_f = np.load(testf_dir, allow_pickle=True)
    val_f = np.load(valf_dir, allow_pickle=True)
    train_l = np.load(trainl_dir, allow_pickle=True)
    test_l = np.load(testl_dir, allow_pickle=True)
    val_l = np.load(vall_dir, allow_pickle=True)

    train_l = create_dictionary(np.asarray(train_l), names)
    test_l = create_dictionary(np.asarray(test_l), names)
    val_l = create_dictionary(np.asarray(val_l), names)
```

```
# Start Random Forest with AdaBoost Training
# Forest 1
forest1 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=100, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)

ada1 = AdaBoostClassifier(base_estimator=forest1,
                          n_estimators=100, random_state=1)

ada1.fit(train_f, train_l)
pred_valid1 = ada1.predict(val_f)
acc1 = metrics.accuracy_score(val_l, pred_valid1)
print("Accuracy of subject {}, with Forest 1: ".format(str(i + 1)), acc1)

# Forest 2
forest2 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=100, max_features='sqrt',
                                 max_depth=8, min_samples_leaf=4,
                                 min_samples_split=2, random_state=1)

ada2 = AdaBoostClassifier(base_estimator=forest2,
                          n_estimators=100, random_state=1)

ada2.fit(train_f, train_l)
pred_valid2 = ada2.predict(val_f)
acc2 = metrics.accuracy_score(val_l, pred_valid2)
print("Accuracy of subject {}, with Forest 2: ".format(str(i + 1)), acc2)

# Forest 3
forest3 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=50, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)

ada3 = AdaBoostClassifier(base_estimator=forest3,
                          n_estimators=50, random_state=1)

ada3.fit(train_f, train_l)
pred_valid3 = ada3.predict(val_f)
acc3 = metrics.accuracy_score(val_l, pred_valid3)
print("Accuracy of subject {}, with Forest 3: ".format(str(i + 1)), acc3)

# Forest 4
forest4 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)

ada4 = AdaBoostClassifier(base_estimator=forest4,
                          n_estimators=500, random_state=1)

ada4.fit(train_f, train_l)
pred_valid4 = ada4.predict(val_f)
acc4 = metrics.accuracy_score(val_l, pred_valid4)
print("Accuracy of subject {}, with Forest 4: ".format(str(i + 1)), acc4)

# Forest 5
forest5 = RandomForestClassifier(criterion="entropy", bootstrap=False,
```

```
                                    n_estimators=200, max_features='sqrt',
                                    max_depth=30, min_samples_leaf=2,
                                    min_samples_split=4, random_state=1)

    ada5 = AdaBoostClassifier(base_estimator=forest5,
                              n_estimators=200, random_state=1)

    ada5.fit(train_f, train_l)
    pred_valid5 = ada5.predict(val_f)
    acc5 = metrics.accuracy_score(val_l, pred_valid5)
    print("Accuracy of subject {}, with Forest 5: ".format(str(i + 1)), acc5)

    # Forest 6
    forest6 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=500, max_features='sqrt',
                                    max_depth=30, min_samples_leaf=4,
                                    min_samples_split=4, random_state=1)
    ada6 = AdaBoostClassifier(base_estimator=forest6,
                              n_estimators=500, random_state=1)

    ada6.fit(train_f, train_l)
    pred_valid6 = ada6.predict(val_f)
    acc6 = metrics.accuracy_score(val_l, pred_valid6)
    print("Accuracy of subject {}, with Forest 6: ".format(str(i + 1)), acc6)

    # Forest 7
    forest7 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=500, max_features='sqrt',
                                    max_depth=20, min_samples_leaf=4,
                                    min_samples_split=4, random_state=1)
    ada7 = AdaBoostClassifier(base_estimator=forest7,
                              n_estimators=500, random_state=1)
    ada7.fit(train_f, train_l)
    pred_valid7 = ada7.predict(val_f)
    acc7 = metrics.accuracy_score(val_l, pred_valid7)
    print("Accuracy of subject {}, with Forest 7: ".format(str(i + 1)), acc7)

    # Try the best validation model on the test set
    Adas = [ada1, ada2, ada3, ada4, ada5, ada6, ada7]
    accuracies = np.asarray([acc1, acc2, acc3, acc4, acc5, acc6, acc7])
    max_acc = np.argmax(accuracies)
    best_forest = Adas[max_acc]

    # Try on test set
    pred_test = best_forest.predict(test_f)
    test_acc = metrics.accuracy_score(test_l, pred_test)
    print("Accuracy of subject {}, on test with AdaBoosted Masked Forest {}:
".format(str(i + 1), str(max_acc + 1)),
          test_acc)
    FINAL_ACCS2.append(test_acc)
final2 = np.asarray(FINAL_ACCS2)
print("Mean of Final Accuracies for AdaBoosted Masked data: " +
str(np.mean(final2)))

# Masked PCA data for all subjects
FINAL_ACCS3 = []
for i in range(6):
    SUB_DIR = os.path.join(FOLDER_DIR, 'subj' + str(i + 1))
    maskedPCA_DIR = os.path.join(SUB_DIR, 'maskedPCA')
```

```
LABELS_DIR = os.path.join(SUB_DIR, 'labels')

trainf_dir = os.path.join(maskedPCA_DIR, 'train_features.npy')
testf_dir = os.path.join(maskedPCA_DIR, 'test_features.npy')
valf_dir = os.path.join(maskedPCA_DIR, 'val_features.npy')

trainl_dir = os.path.join(LABELS_DIR, 'train_labels.npy')
testl_dir = os.path.join(LABELS_DIR, 'test_labels.npy')
vall_dir = os.path.join(LABELS_DIR, 'val_labels.npy')

train_f = np.load(trainf_dir, allow_pickle=True)
test_f = np.load(testf_dir, allow_pickle=True)
val_f = np.load(valf_dir, allow_pickle=True)
train_l = np.load(trainl_dir, allow_pickle=True)
test_l = np.load(testl_dir, allow_pickle=True)
val_l = np.load(vall_dir, allow_pickle=True)

train_l = create_dictionary(np.asarray(train_l), names)
test_l = create_dictionary(np.asarray(test_l), names)
val_l = create_dictionary(np.asarray(val_l), names)

# Start Random Forest with AdaBoost Training
# Forest 1
forest1 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=100, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)

ada1 = AdaBoostClassifier(base_estimator=forest1,
                          n_estimators=100, random_state=1)

ada1.fit(train_f, train_l)
pred_valid1 = ada1.predict(val_f)
acc1 = metrics.accuracy_score(val_l, pred_valid1)
print("Accuracy of subject {}, with Forest 1: ".format(str(i + 1)), acc1)

# Forest 2
forest2 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=100, max_features='sqrt',
                                 max_depth=8, min_samples_leaf=4,
                                 min_samples_split=2, random_state=1)

ada2 = AdaBoostClassifier(base_estimator=forest2,
                          n_estimators=100, random_state=1)

ada2.fit(train_f, train_l)
pred_valid2 = ada2.predict(val_f)
acc2 = metrics.accuracy_score(val_l, pred_valid2)
print("Accuracy of subject {}, with Forest 2: ".format(str(i + 1)), acc2)

# Forest 3
forest3 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=50, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)

ada3 = AdaBoostClassifier(base_estimator=forest3,
                          n_estimators=50, random_state=1)
```

```
ada3.fit(train_f, train_l)
pred_valid3 = ada3.predict(val_f)
acc3 = metrics.accuracy_score(val_l, pred_valid3)
print("Accuracy of subject {}, with Forest 3: ".format(str(i + 1)), acc3)

# Forest 4
forest4 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)

ada4 = AdaBoostClassifier(base_estimator=forest4,
                          n_estimators=500, random_state=1)

ada4.fit(train_f, train_l)
pred_valid4 = ada4.predict(val_f)
acc4 = metrics.accuracy_score(val_l, pred_valid4)
print("Accuracy of subject {}, with Forest 4: ".format(str(i + 1)), acc4)

# Forest 5
forest5 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=200, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=2,
                                 min_samples_split=4, random_state=1)

ada5 = AdaBoostClassifier(base_estimator=forest5,
                          n_estimators=200, random_state=1)

ada5.fit(train_f, train_l)
pred_valid5 = ada5.predict(val_f)
acc5 = metrics.accuracy_score(val_l, pred_valid5)
print("Accuracy of subject {}, with Forest 5: ".format(str(i + 1)), acc5)

# Forest 6
forest6 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=30, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
ada6 = AdaBoostClassifier(base_estimator=forest6,
                          n_estimators=500, random_state=1)

ada6.fit(train_f, train_l)
pred_valid6 = ada6.predict(val_f)
acc6 = metrics.accuracy_score(val_l, pred_valid6)
print("Accuracy of subject {}, with Forest 6: ".format(str(i + 1)), acc6)

# Forest 7
forest7 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                 n_estimators=500, max_features='sqrt',
                                 max_depth=20, min_samples_leaf=4,
                                 min_samples_split=4, random_state=1)
ada7 = AdaBoostClassifier(base_estimator=forest7,
                          n_estimators=500, random_state=1)
ada7.fit(train_f, train_l)
pred_valid7 = ada7.predict(val_f)
acc7 = metrics.accuracy_score(val_l, pred_valid7)
print("Accuracy of subject {}, with Forest 7: ".format(str(i + 1)), acc7)

# Try the best validation model on the test set
```

```
    Adas = [ada1, ada2, ada3, ada4, ada5, ada6, ada7]
    accuracies = np.asarray([acc1, acc2, acc3, acc4, acc5, acc6, acc7])
    max_acc = np.argmax(accuracies)
    best_forest = Adas[max_acc]

    # Try on test set
    pred_test = best_forest.predict(test_f)
    test_acc = metrics.accuracy_score(test_l, pred_test)
    print("Accuracy of subject {}, on test with AdaBoosted Masked PCA Forest {}:
".format(str(i + 1), str(max_acc + 1)),
          test_acc)
    FINAL_ACCS3.append(test_acc)
final3 = np.asarray(FINAL_ACCS3)
print("Mean of Final Accuracies for AdaBoosted Masked PCA data: " +
str(np.mean(final3)))

# ORG data for all subjects
FINAL_ACCS4 = []
for i in range(6):
    SUB_DIR = os.path.join(FOLDER_DIR, 'subj' + str(i + 1))
    ORG_DIR = os.path.join(SUB_DIR, 'org')
    LABELS_DIR = os.path.join(SUB_DIR, 'labels')

    trainf_dir = os.path.join(ORG_DIR, 'train_features.npy')
    testf_dir = os.path.join(ORG_DIR, 'test_features.npy')
    valf_dir = os.path.join(ORG_DIR, 'val_features.npy')

    trainl_dir = os.path.join(LABELS_DIR, 'train_labels.npy')
    testl_dir = os.path.join(LABELS_DIR, 'test_labels.npy')
    vall_dir = os.path.join(LABELS_DIR, 'val_labels.npy')

    train_f = np.load(trainf_dir, allow_pickle=True)
    test_f = np.load(testf_dir, allow_pickle=True)
    val_f = np.load(valf_dir, allow_pickle=True)
    train_l = np.load(trainl_dir, allow_pickle=True)
    test_l = np.load(testl_dir, allow_pickle=True)
    val_l = np.load(vall_dir, allow_pickle=True)

    train_l = create_dictionary(np.asarray(train_l), names)
    test_l = create_dictionary(np.asarray(test_l), names)
    val_l = create_dictionary(np.asarray(val_l), names)

    # Start Random Forest with AdaBoost Training
    # Forest 1
    forest1 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=100, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)

    ada1 = AdaBoostClassifier(base_estimator=forest1,
                              n_estimators=100, random_state=1)

    ada1.fit(train_f, train_l)
    pred_valid1 = ada1.predict(val_f)
    acc1 = metrics.accuracy_score(val_l, pred_valid1)
    print("Accuracy of subject {}, with Forest 1: ".format(str(i + 1)), acc1)

    # Forest 2
    forest2 = RandomForestClassifier(criterion="entropy", bootstrap=False,
```

```
                                    n_estimators=100, max_features='sqrt',
                                    max_depth=8, min_samples_leaf=4,
                                    min_samples_split=2, random_state=1)

ada2 = AdaBoostClassifier(base_estimator=forest2,
                          n_estimators=100, random_state=1)

ada2.fit(train_f, train_l)
pred_valid2 = ada2.predict(val_f)
acc2 = metrics.accuracy_score(val_l, pred_valid2)
print("Accuracy of subject {}, with Forest 2: ".format(str(i + 1)), acc2)

# Forest 3
forest3 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=50, max_features='sqrt',
                                    max_depth=20, min_samples_leaf=4,
                                    min_samples_split=4, random_state=1)

ada3 = AdaBoostClassifier(base_estimator=forest3,
                          n_estimators=50, random_state=1)

ada3.fit(train_f, train_l)
pred_valid3 = ada3.predict(val_f)
acc3 = metrics.accuracy_score(val_l, pred_valid3)
print("Accuracy of subject {}, with Forest 3: ".format(str(i + 1)), acc3)

# Forest 4
forest4 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=500, max_features='sqrt',
                                    max_depth=30, min_samples_leaf=2,
                                    min_samples_split=4, random_state=1)

ada4 = AdaBoostClassifier(base_estimator=forest4,
                          n_estimators=500, random_state=1)

ada4.fit(train_f, train_l)
pred_valid4 = ada4.predict(val_f)
acc4 = metrics.accuracy_score(val_l, pred_valid4)
print("Accuracy of subject {}, with Forest 4: ".format(str(i + 1)), acc4)

# Forest 5
forest5 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=200, max_features='sqrt',
                                    max_depth=30, min_samples_leaf=2,
                                    min_samples_split=4, random_state=1)

ada5 = AdaBoostClassifier(base_estimator=forest5,
                          n_estimators=200, random_state=1)

ada5.fit(train_f, train_l)
pred_valid5 = ada5.predict(val_f)
acc5 = metrics.accuracy_score(val_l, pred_valid5)
print("Accuracy of subject {}, with Forest 5: ".format(str(i + 1)), acc5)

# Forest 6
forest6 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                    n_estimators=500, max_features='sqrt',
                                    max_depth=30, min_samples_leaf=4,
                                    min_samples_split=4, random_state=1)
```

```
    ada6 = AdaBoostClassifier(base_estimator=forest6,
                              n_estimators=500, random_state=1)

    ada6.fit(train_f, train_l)
    pred_valid6 = ada6.predict(val_f)
    acc6 = metrics.accuracy_score(val_l, pred_valid6)
    print("Accuracy of subject {}, with Forest 6: ".format(str(i + 1)), acc6)

    # Forest 7
    forest7 = RandomForestClassifier(criterion="entropy", bootstrap=False,
                                     n_estimators=500, max_features='sqrt',
                                     max_depth=20, min_samples_leaf=4,
                                     min_samples_split=4, random_state=1)
    ada7 = AdaBoostClassifier(base_estimator=forest7,
                              n_estimators=500, random_state=1)
    ada7.fit(train_f, train_l)
    pred_valid7 = ada7.predict(val_f)
    acc7 = metrics.accuracy_score(val_l, pred_valid7)
    print("Accuracy of subject {}, with Forest 7: ".format(str(i + 1)), acc7)

    # Try the best validation model on the test set
    Adas = [ada1, ada2, ada3, ada4, ada5, ada6, ada7]
    accuracies = np.asarray([acc1, acc2, acc3, acc4, acc5, acc6, acc7])
    max_acc = np.argmax(accuracies)
    best_forest = Adas[max_acc]

    # Try on test set
    pred_test = best_forest.predict(test_f)
    test_acc = metrics.accuracy_score(test_l, pred_test)
    print("Accuracy of subject {}, on test with AdaBoosted Original Data Forest
{}: ".format(str(i + 1),

str(max_acc + 1)),
          test_acc)
    FINAL_ACCS4.append(test_acc)
final4 = np.asarray(FINAL_ACCS4)
print("Mean of Final Accuracies for AdaBoosted Original data: " +
str(np.mean(final4)))

plt.figure()
objects = ('Original', 'Masked', 'PCA', 'Masked PCA')
y_pos = np.arange(len(objects))
# Calculate Mean
mean1 = np.mean(final1)
mean2 = np.mean(final2)
mean3 = np.mean(final3)
mean4 = np.mean(final4)
# Calculate Standard deviation
std1 = np.std(final1)
std2 = np.std(final2)
std3 = np.std(final3)
std4 = np.std(final4)
performance = [mean4, mean2, mean1, mean3]
error = [std4, std2, std1, std3]

plt.bar(y_pos, performance, yerr=error, align='center', alpha=0.5, capsize=10)
plt.grid(True)
plt.xticks(y_pos, objects)
plt.ylabel('Test Accuracy')
```

```
plt.title('Mean Test Accuracies for Random Forest with Ada Boost')

plt.show()

print("Random Forest with AdaBoost:")
print("1-Org data mean test accuracy and std: mean: {} std: {}".format(str(mean4),
str(std4)))
print("")
print("2-pca data mean test accuracy and std: mean: {} std: {}".format(str(mean1),
str(std1)))
print("")
print("3-masked data mean test accuracy and std: mean: {} std:
{}".format(str(mean2), str(std2)))
print("")
print("4-masked pca data mean test accuracy and std: mean: {} std:
{}".format(str(mean3), str(std3)))

methods = [final1, final2, final3, final4]
datas = ['PCA', 'Masked', 'Masked PCA', 'Original']
i = 0
for final in methods:
    plt.figure()
    plt.plot(final)
    plt.xlabel('Subjects')
    plt.xticks(np.arange(len(final)), ['1', '2', '3', '4', '5', '6'])
    plt.ylabel('Accuracy on Test')
    plt.title('The Validation Accuracies of Random Forest\n on {}
Data'.format(datas[i]))
    plt.grid(True)
    plt.show()
    i = i + 1
```

Logistic_Regression.py

```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

DATA_DIR = 'data_final'

def accuracy(y_pred, y_real):
    num_true = np.sum(y_pred == y_real)
    return num_true / y_real.shape[0]

def barplot(val_acc_tot, string):
    X = np.arange(1, 7)
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1])
    ax.bar(X - 0.30, val_acc_tot[0], color='b', width=0.25)
    ax.bar(X - 0.10, val_acc_tot[1], color='g', width=0.25)
    ax.bar(X + 0.10, val_acc_tot[2], color='r', width=0.25)
    ax.bar(X + 0.30, val_acc_tot[3], color='y', width=0.25)
    plt.xlabel('Subjects')
    plt.ylabel('Validation Accuracies')
    plt.title(string)
    plt.legend(["Newton-cg", "Sag", "Saga", "Lbfgs"], loc="upper right")
```

```
    plt.grid()
    plt.show()

def barplot2(val_acc_tot, string):
    X = np.arange(1, 7)
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1])
    ax.bar(X - 0.10, val_acc_tot[0], color='b', width=0.25)
    ax.bar(X + 0.10, val_acc_tot[1], color='g', width=0.25)
    plt.xlabel('Subjects')
    plt.ylabel('Validation Accuracies')
    plt.title(string)
    plt.legend(["Newton-cg", "Lbfgs"], loc="upper right")
    plt.grid()
    plt.show()

# PCA training and validation
best_models = []
val_acc_tot = np.zeros((4, 6))
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'PCA')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    solver = ['newton-cg', 'sag', 'saga', 'lbfgs']
    best_val_acc = 0
    k_acc = []
    for j in range(len(solver)):
        logisticRegr = LogisticRegression(solver=solver[j])  # ,max_iter=1000)
        logisticRegr = logisticRegr.fit(X_train, y_train)
        predictions = logisticRegr.predict(X_val)
        val_acc = accuracy(predictions, y_val)
        k_acc.append(val_acc)
        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = logisticRegr
        print('Validation Accuracy for solver ' + solver[j] + 'is: ' +
str(val_acc))
        val_acc_tot[j][i - 1] = val_acc
    best_models.append(best_model)
barplot(val_acc_tot, 'Validation Accuracies for PCA')
print('\n\n-----------------------------------------\n\n')

# PCA test
test_accuracies_pca = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
```

```
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'PCA')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_pca.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_pca)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_pca),
np.std(test_accuracies_pca)))

# Masked training and validation
best_models = []
val_acc_tot = np.zeros((4, 6))
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'masked')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    solver = ['newton-cg', 'sag', 'saga', 'lbfgs']
    best_val_acc = 0
    k_acc = []
    for j in range(len(solver)):
        logisticRegr = LogisticRegression(solver=solver[j])  # ,max_iter=1000)
        logisticRegr = logisticRegr.fit(X_train, y_train)
        predictions = logisticRegr.predict(X_val)
        val_acc = accuracy(predictions, y_val)
        k_acc.append(val_acc)
        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = logisticRegr
        print('Validation Accuracy for solver ' + solver[j] + 'is: ' +
str(val_acc))
        val_acc_tot[j][i - 1] = val_acc
    best_models.append(best_model)
barplot(val_acc_tot, 'Validation Accuracies for Masked')
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_masked = []
```

```
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'masked')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_masked.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_masked)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_masked),
np.std(test_accuracies_masked)))

# Masked-PCA training and validation
best_models = []
val_acc_tot = np.zeros((4, 6))
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'maskedPCA')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    solver = ['newton-cg', 'sag', 'saga', 'lbfgs']
    best_val_acc = 0
    k_acc = []
    for j in range(len(solver)):
        logisticRegr = LogisticRegression(solver=solver[j])  # ,max_iter=1000)
        logisticRegr = logisticRegr.fit(X_train, y_train)
        predictions = logisticRegr.predict(X_val)
        val_acc = accuracy(predictions, y_val)
        k_acc.append(val_acc)
        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = logisticRegr
        print('Validation Accuracy for solver ' + solver[j] + 'is: ' +
str(val_acc))
        val_acc_tot[j][i - 1] = val_acc
    best_models.append(best_model)
barplot(val_acc_tot, 'Validation Accuracies for Masked PCA')
print('\n\n-----------------------------------------\n\n')
```

```
# PCA test
test_accuracies_maskedpca = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'maskedPCA')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_maskedpca.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_maskedpca)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_maskedpca),
np.std(test_accuracies_maskedpca)))

# Original training and validation
best_models = []
val_acc_tot = np.zeros((2, 6))
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'org')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))

    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)

    solver = ['newton-cg', 'lbfgs']
    best_val_acc = 0
    k_acc = []
    for j in range(len(solver)):
        logisticRegr = LogisticRegression(solver=solver[j])  # ,max_iter=1000)
        logisticRegr = logisticRegr.fit(X_train, y_train)
        predictions = logisticRegr.predict(X_val)
        val_acc = accuracy(predictions, y_val)
        k_acc.append(val_acc)
        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = logisticRegr
        print('Validation Accuracy for solver ' + solver[j] + 'is: ' +
str(val_acc))
        val_acc_tot[j][i - 1] = val_acc
    best_models.append(best_model)
barplot2(val_acc_tot, 'Validation Accuracies for Original Data')
```

```
print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies_org = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'org')

    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies_org.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies_org)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies_org),
np.std(test_accuracies_org)))

# plot means and stds
means = 100 * np.asarray([np.mean(test_accuracies_org),
np.mean(test_accuracies_masked), np.mean(test_accuracies_pca),
                          np.mean(test_accuracies_maskedpca)])
stds = 100 * np.asarray([np.std(test_accuracies_org),
np.std(test_accuracies_masked), np.std(test_accuracies_pca),
                         np.std(test_accuracies_maskedpca)])
reducs = ['Original', 'Masked', 'PCA', 'Masked+PCA']

fig, ax = plt.subplots()
ax.bar(np.arange(4), means, yerr=stds, align='center', alpha=0.5, ecolor='black',
capsize=10)
ax.set_yticks(np.arange(0, 105, 10))
ax.set_ylabel('Test Accuracy')
ax.set_xticks(np.arange(4))
ax.set_xticklabels(reducs)
ax.set_title('Mean Test Accuracies for Logistic Regression with Different
Solvers')
ax.grid(True)

# Save the figure and show
plt.tight_layout()
plt.show()
```

MLP_NB.py

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
import math
```

```
data_path = ""
os.chdir("data_final/")

subj = "subj"
ways = ["labels", "masked", "maskedPCA", "org", "PCA"]

labels = []
masked = []
maskedPCA = []
org = []
PCA = []

for i in range(6):
    subj_h = subj + str(i + 1)
    p_h = data_path + subj_h + "/"
    for j in (ways):
        path_holder = p_h + j + "/"
        if j == "labels":
            path_h = p_h + j
            labels.append(np.load(path_holder + "train_labels.npy",
allow_pickle=True))
            labels.append(np.load(path_holder + "val_labels.npy",
allow_pickle=True))
            labels.append(np.load(path_holder + "test_labels.npy",
allow_pickle=True))

        if j == "masked":
            path_h = p_h + j
            masked.append(np.load(path_holder + "train_features.npy",
allow_pickle=True))
            masked.append(np.load(path_holder + "val_features.npy",
allow_pickle=True))
            masked.append(np.load(path_holder + "test_features.npy",
allow_pickle=True))

        if j == "maskedPCA":
            path_h = p_h + j
            maskedPCA.append(np.load(path_holder + "train_features.npy",
allow_pickle=True))
            maskedPCA.append(np.load(path_holder + "val_features.npy",
allow_pickle=True))
            maskedPCA.append(np.load(path_holder + "test_features.npy",
allow_pickle=True))

        if j == "org":
            path_h = p_h + j
            org.append(np.load(path_holder + "train_features.npy",
allow_pickle=True))
            org.append(np.load(path_holder + "val_features.npy",
allow_pickle=True))
            org.append(np.load(path_holder + "test_features.npy",
allow_pickle=True))

        if j == "PCA":
            path_h = p_h + j
            PCA.append(np.load(path_holder + "train_features.npy",
allow_pickle=True))
            PCA.append(np.load(path_holder + "val_features.npy",
allow_pickle=True))
```

```
            PCA.append(np.load(path_holder + "test_features.npy",
allow_pickle=True))

np_labels = np.array(labels)
np_masked = np.array(masked)
np_masked_PCA = np.array(maskedPCA)
np_org = np.array(org)
np_PCA = np.array(PCA)

def mlp_hyper(train_f, train_l, val_f, val_l, lr, hid_lay, batch_size, k_fold=10):
    accs = []
    models = []
    counter = 1

    for i in (lr):
        for j in (hid_lay):
            for k in (batch_size):
                print(j)
                clf = MLPClassifier(max_iter=500, hidden_layer_sizes=j,
learning_rate_init=i, early_stopping=True,
                                    alpha=0.00001, batch_size=k, random_state=1)

                mlp_model = clf.fit(train_f, train_l)

                mlp_preds = mlp_model.predict(val_f)

                correct = 0

                for n in range(val_l.shape[0]):
                    if val_l[n] == mlp_preds[n]:
                        correct += 1

                acc = correct / val_l.shape[0]
                print(acc)
                accs.append(acc)
                models.append(mlp_model)
        counter += 1

    return np.array(models), np.array(accs)

acc_subjs = []
lr_s = [0.0005]
batch_s = [200]
hid_s = [(1000, 750, 500, 250, 100), (1000, 2000, 1000, 100), (200, 200, 200, 200,
100)]
# , (1000,500,250), (1000,750,250,100), (1000,750,100), (1000,2000,1000, 100),
(200,200,200,200,100)

accs = []
models = []

for i in range(6):
    print("Accuracy of Subject " + str(i + 1) + " is as follows:")
    print("Masked Accs of subj" + str(i + 1) + ":")
    train_f = np_masked[i * 3]
    val_f = np_masked[(i * 3) + 1]

    train_l = np_labels[i * 3]
    val_l = np_labels[(i * 3) + 1]
```

```
    models_h, acc_h = mlp_hyper(train_f, train_l, val_f, val_l, lr_s, hid_s,
batch_s)
    accs.append(acc_h)
    models.append(models_h)

    print("MaskedPCA Accs of subj" + str(i + 1) + ":")
    train_f = np_masked_PCA[i * 3]
    val_f = np_masked_PCA[(i * 3) + 1]

    models_h, acc_h = mlp_hyper(train_f, train_l, val_f, val_l, lr_s, hid_s,
batch_s)
    accs.append(acc_h)
    models.append(models_h)

    print("Orginal Accs of subj" + str(i + 1) + ":")
    train_f = np_org[i * 3]
    val_f = np_org[(i * 3) + 1]

    models_h, acc_h = mlp_hyper(train_f, train_l, val_f, val_l, lr_s, hid_s,
batch_s)
    accs.append(acc_h)
    models.append(models_h)

    print("PCA Accs of subj" + str(i + 1) + ":")
    train_f = np_PCA[i * 3]
    val_f = np_PCA[(i * 3) + 1]

    models_h, acc_h = mlp_hyper(train_f, train_l, val_f, val_l, lr_s, hid_s,
batch_s)
    accs.append(acc_h)
    models.append(models_h)

accs = np.array(accs)
models = np.array(models)

accs_1 = []
models_1 = []

accs_2 = []
models_2 = []

accs_3 = []
models_3 = []

accs_4 = []
models_4 = []

accs_5 = []
models_5 = []

accs_6 = []
models_6 = []

for i in range(24):
    t = math.ceil((i + 1) / 4)
    # print(t)
    if t == 1:
        accs_1.append(accs[i])
```

```
            models_1.append(models[i])

        elif t == 2:
            accs_2.append(accs[i])
            models_2.append(models[i])

        elif t == 3:
            accs_3.append(accs[i])
            models_3.append(models[i])

        elif t == 4:
            accs_4.append(accs[i])
            models_4.append(models[i])

        elif t == 5:
            accs_5.append(accs[i])
            models_5.append(models[i])

        elif t == 6:
            accs_6.append(accs[i])
            models_6.append(models[i])

accs_1 = np.array(accs_1)
models_1 = np.array(models_1)

accs_2 = np.array(accs_2)
models_2 = np.array(models_2)

accs_3 = np.array(accs_3)
models_3 = np.array(models_3)

accs_4 = np.array(accs_4)
models_4 = np.array(models_4)

accs_5 = np.array(accs_5)
models_5 = np.array(models_5)

accs_6 = np.array(accs_6)
models_6 = np.array(models_6)

table_titles = ["Model 1", "Model 2", "Model 3"]
table_rows = ["Maked", "Masked PCA", "Orginal", "Orginal PCA"]

print("SUBJECT 1 VALIDATION RESULTS")
sub1_table = pd.DataFrame(accs_1, table_rows, table_titles)
sub1_table.head()

print("SUBJECT 2 VALIDATION RESULTS")
sub2_table = pd.DataFrame(accs_2, table_rows, table_titles)
sub2_table.head()

print("SUBJECT 3 VALIDATION RESULTS")
sub3_table = pd.DataFrame(accs_3, table_rows, table_titles)
sub3_table.head()

print("SUBJECT 4 VALIDATION RESULTS")
sub4_table = pd.DataFrame(accs_4, table_rows, table_titles)
sub4_table.head()
```

```
print("SUBJECT 5 VALIDATION RESULTS")
sub5_table = pd.DataFrame(accs_5, table_rows, table_titles)
sub5_table.head()

print("SUBJECT 6 VALIDATION RESULTS")
sub6_table = pd.DataFrame(accs_6, table_rows, table_titles)
sub6_table.head()

accs_1 = accs_1.T
X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0.00, accs_1[0], color='b', width=0.25, label="Model 1")
ax.bar(X + 0.25, accs_1[1], color='g', width=0.25, label="Model 2")
ax.bar(X + 0.50, accs_1[2], color='r', width=0.25, label="Model 3")
# ax.bar(X + 0.75, accs_1[3], color = 'o', width = 0.25, label="Orginal PCA")
ax.legend()
ax.set_xticklabels(("", "Maked", "", "Masked PCA", '', "Orginal", "", "Orginal
PCA"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Accuracy Comparison of Subject 1 Validation Accuracy for Models
1,2,3...")
ax.set_xlabel("Dataset")

accs_2 = accs_2.T
X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0.00, accs_2[0], color='b', width=0.25, label="Model 1")
ax.bar(X + 0.25, accs_2[1], color='g', width=0.25, label="Model 2")
ax.bar(X + 0.50, accs_2[2], color='r', width=0.25, label="Model 3")
# ax.bar(X + 0.75, accs_1[3], color = 'o', width = 0.25, label="Orginal PCA")
ax.legend()
ax.set_xticklabels(("", "Maked", "", "Masked PCA", '', "Orginal", "", "Orginal
PCA"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Accuracy Comparison of Subject 2 Validation Accuracy for Models
1,2,3...")
ax.set_xlabel("Dataset")

accs_3 = accs_3.T
X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0.00, accs_3[0], color='b', width=0.25, label="Model 1")
ax.bar(X + 0.25, accs_3[1], color='g', width=0.25, label="Model 2")
ax.bar(X + 0.50, accs_3[2], color='r', width=0.25, label="Model 3")
# ax.bar(X + 0.75, accs_1[3], color = 'o', width = 0.25, label="Orginal PCA")
ax.legend()
ax.set_xticklabels(("", "Maked", "", "Masked PCA", '', "Orginal", "", "Orginal
PCA"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Accuracy Comparison of Subject 3 Validation Accuracy for Models
1,2,3...")
ax.set_xlabel("Dataset")

accs_4 = accs_4.T
X = np.arange(4)
fig = plt.figure()
```

```
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0.00, accs_4[0], color='b', width=0.25, label="Model 1")
ax.bar(X + 0.25, accs_4[1], color='g', width=0.25, label="Model 2")
ax.bar(X + 0.50, accs_4[2], color='r', width=0.25, label="Model 3")
# ax.bar(X + 0.75, accs_1[3], color = 'o', width = 0.25, label="Orginal PCA")
ax.legend()
ax.set_xticklabels(("", "Maked", "", "Masked PCA", '', "Orginal", "", "Orginal
PCA"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Accuracy Comparison of Subject 4 Validation Accuracy for Models
1,2,3...")
ax.set_xlabel("Dataset")

accs_5 = accs_5.T
X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0.00, accs_5[0], color='b', width=0.25, label="Model 1")
ax.bar(X + 0.25, accs_5[1], color='g', width=0.25, label="Model 2")
ax.bar(X + 0.50, accs_5[2], color='r', width=0.25, label="Model 3")
# ax.bar(X + 0.75, accs_1[3], color = 'o', width = 0.25, label="Orginal PCA")
ax.legend()
ax.set_xticklabels(("", "Maked", "", "Masked PCA", '', "Orginal", "", "Orginal
PCA"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Accuracy Comparison of Subject 5 Validation Accuracy for Models
1,2,3...")
ax.set_xlabel("Dataset")

accs_6 = accs_6.T
X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0.00, accs_6[0], color='b', width=0.25, label="Model 1")
ax.bar(X + 0.25, accs_6[1], color='g', width=0.25, label="Model 2")
ax.bar(X + 0.50, accs_6[2], color='r', width=0.25, label="Model 3")
# ax.bar(X + 0.75, accs_1[3], color = 'o', width = 0.25, label="Orginal PCA")
ax.legend()
ax.set_xticklabels(("", "Maked", "", "Masked PCA", '', "Orginal", "", "Orginal
PCA"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Accuracy Comparison of Subject 6 Validation Accuracy for Models
1,2,3...")
ax.set_xlabel("Dataset")

accs_1 = accs_1.T
accs_2 = accs_2.T
accs_3 = accs_3.T
accs_4 = accs_4.T
accs_5 = accs_5.T
accs_6 = accs_6.T

b_models_1 = []
b_models_2 = []
b_models_3 = []
b_models_4 = []
b_models_5 = []
b_models_6 = []
```

```
for i in range(6):
    for j in range(4):
        if i == 0:
            b_h_i = np.argmax(accs_1[j])
            b_models_1.append(models_1[j, b_h_i])

        if i == 1:
            b_h_i = np.argmax(accs_2[j])
            b_models_2.append(models_2[j, b_h_i])

        if i == 2:
            b_h_i = np.argmax(accs_3[j])
            b_models_3.append(models_3[j, b_h_i])

        if i == 3:
            b_h_i = np.argmax(accs_4[j])
            b_models_4.append(models_4[j, b_h_i])

        if i == 4:
            b_h_i = np.argmax(accs_5[j])
            b_models_5.append(models_5[j, b_h_i])

        if i == 5:
            b_h_i = np.argmax(accs_6[j])
            b_models_6.append(models_6[j, b_h_i])

b_models_1 = np.array(b_models_1)
b_models_2 = np.array(b_models_2)
b_models_3 = np.array(b_models_3)
b_models_4 = np.array(b_models_4)
b_models_5 = np.array(b_models_5)
b_models_6 = np.array(b_models_6)
b_models_1[0].get_params()

accs_masked = []
accs_maskedPCA = []
accs_org = []
accs_orgPCA = []

for i in range(6):
    if i == 0:
        model_h = b_models_1
    elif i == 1:
        model_h = b_models_2
    elif i == 2:
        model_h = b_models_3
    elif i == 3:
        model_h = b_models_4
    elif i == 4:
        model_h = b_models_5
    else:
        model_h = b_models_6

    test_f = np_masked[(i * 3) + 2]
    test_l = np_labels[(i * 3) + 2]

    masked_m = model_h[0]

    pre_test = masked_m.predict(test_f)
```

```
    correct = 0
    for j in range(test_l.shape[0]):
        if pre_test[j] == test_l[j]:
            correct += 1

    accs_masked.append((correct / test_l.shape[0]))

    test_f = np_masked_PCA[(i * 3) + 2]
    test_l = np_labels[(i * 3) + 2]

    maskedPCA_m = model_h[1]

    pre_test = maskedPCA_m.predict(test_f)

    correct = 0
    for j in range(test_l.shape[0]):
        if pre_test[j] == test_l[j]:
            correct += 1

    accs_maskedPCA.append((correct / test_l.shape[0]))

    test_f = np_org[(i * 3) + 2]
    test_l = np_labels[(i * 3) + 2]

    org_m = model_h[2]

    pre_test = org_m.predict(test_f)

    correct = 0
    for j in range(test_l.shape[0]):
        if pre_test[j] == test_l[j]:
            correct += 1
    accs_org.append((correct / test_l.shape[0]))

    test_f = np_PCA[(i * 3) + 2]
    test_l = np_labels[(i * 3) + 2]

    pca_m = model_h[3]

    pre_test = pca_m.predict(test_f)

    correct = 0
    for j in range(test_l.shape[0]):
        if pre_test[j] == test_l[j]:
            correct += 1

    accs_orgPCA.append(((correct) / test_l.shape[0]))

accs_masked = np.array(accs_masked)
accs_maskedPCA = np.array(accs_maskedPCA)
accs_org = np.array(accs_org)
accs_orgPCA = np.array(accs_orgPCA)

X = np.arange(6)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0.05, accs_masked, color='b', width=0.20, label="Masked Dataset")
ax.bar(X + 0.25, accs_maskedPCA, color='g', width=0.25, label="Masked PCA
```

```
Dataset")
ax.bar(X + 0.50, accs_org, color='r', width=0.25, label="Orginal Dataset")
ax.bar(X + 0.70, accs_orgPCA, color='orange', width=0.20, label="Original PCA
Dataset")
ax.legend()
ax.set_xticklabels(("", "Subject 1", "Subject 2", "Subject 3", "Subject 4",
"Subject 5", "Subject 6"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Test Dataset Accuracy Results for All Subjects for Masked, Masked
PCA, Original, Original PCA Datasets")
ax.set_xlabel("Subject Number")

X = np.arange(6)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X, [accs_masked[0], accs_masked[1], accs_maskedPCA[2], accs_masked[3],
accs_masked[4], accs_masked[5]],
       color='green', width=0.5)
ax.set_xticklabels(("", "Subject 1", "Subject 2", "Subject 3", "Subject 4",
"Subject 5", "Subject 6"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Test Dataset Accuracy Results for All Subjects For Best Models")
ax.set_xlabel("Subject Number")

table_mlp_test = np.array(
    [accs_masked[0], accs_masked[1], accs_masked[2], accs_masked[3],
accs_masked[4], accs_masked[5]])

table_mlp_test = table_mlp_test.reshape(6, 1).T

table_titles = ["Best Model Acc"]
table_rows = ["Subject 1", "Subject 2", "Subject 3", "Subject 4", "Subject 5",
"Subject 6"]
test_mlp_accs = np.concatenate(
    (accs_masked.reshape(6, 1), accs_maskedPCA.reshape(6, 1), accs_org.reshape(6,
1), accs_orgPCA.reshape(6, 1)),
    axis=1)

sub3_table = pd.DataFrame(table_mlp_test, table_titles, table_rows)
sub3_table.head()

masked_m = np.mean(accs_masked)
masked_s = np.std(accs_masked) / 2

maskedPCA_m = np.mean(accs_maskedPCA)
maskedPCA_s = np.std(accs_maskedPCA) / 2

org_m = np.mean(accs_org)
org_s = np.std(accs_org) / 2

orgPCA_m = np.mean(accs_orgPCA)
orgPCA_s = np.std(accs_orgPCA) / 2

error = [masked_s, maskedPCA_s, org_s, orgPCA_s]
means = [masked_m, maskedPCA_m, org_m, orgPCA_m]

X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
```

```
ax.bar(X, [masked_m, maskedPCA_m, org_m, orgPCA_m], yerr=error, color='green',
align='center', width=0.5, alpha=0.5,
       capsize=10)
ax.set_xticklabels(("", "Masked", "", "Masked PCA", "", "Original", "", "Original
PCA"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Test Dataset Mean Accuracy Results")
ax.set_xlabel("Subject Number")

print(np.array(error * 2))
print(means)

#_____________________________________________
# NB Classifier

def naiveBayes_Train(train_f, train_l, val_f, val_l, ):
    clf = GaussianNB()
    model_nb = clf.fit(train_f, train_l)
    pre = model_nb.predict(val_f)

    correct = 0
    for j in range(val_l.shape[0]):
        if val_l[j] == pre[j]:
            correct += 1

    acc_h = correct / val_l.shape[0]
    print(acc_h)
    model = model_nb
    acc = acc_h

    return model, acc

acc_subjs_nb = []
accs_nb = []
models_nb = []

for i in range(6):
    print("Accuracy of Subject " + str(i + 1) + " is as follows:")
    print("Masked Accs of subj" + str(i + 1) + ":")
    train_f = np_masked[i * 3]
    val_f = np_masked[(i * 3) + 1]

    train_l = np_labels[i * 3]
    val_l = np_labels[(i * 3) + 1]

    models_h, acc_h = naiveBayes_Train(train_f, train_l, val_f, val_l)
    accs_nb.append(acc_h)
    models_nb.append(models_h)

    print("MaskedPCA Accs of subj" + str(i + 1) + ":")
    train_f = np_masked_PCA[i * 3]
    val_f = np_masked_PCA[(i * 3) + 1]

    models_h, acc_h = naiveBayes_Train(train_f, train_l, val_f, val_l)
    accs_nb.append(acc_h)
    models_nb.append(models_h)

    print("Orginal Accs of subj" + str(i + 1) + ":")
    train_f = np_org[i * 3]
```

```
    val_f = np_org[(i * 3) + 1]

    models_h, acc_h = naiveBayes_Train(train_f, train_l, val_f, val_l)
    accs_nb.append(acc_h)
    models_nb.append(models_h)

    print("PCA Accs of subj" + str(i + 1) + ":")
    train_f = np_PCA[i * 3]
    val_f = np_PCA[(i * 3) + 1]

    models_h, acc_h = naiveBayes_Train(train_f, train_l, val_f, val_l)
    accs_nb.append(acc_h)
    models_nb.append(models_h)

accs_1_nb = []
models_1_nb = []

accs_2_nb = []
models_2_nb = []

accs_3_nb = []
models_3_nb = []

accs_4_nb = []
models_4_nb = []

accs_5_nb = []
models_5_nb = []

accs_6_nb = []
models_6_nb = []

for i in range(24):
    t = math.ceil((i + 1) / 4)
    # print(t)
    if t == 1:
        accs_1_nb.append(accs_nb[i])
        models_1_nb.append(models_nb[i])

    elif t == 2:
        accs_2_nb.append(accs_nb[i])
        models_2_nb.append(models_nb[i])

    elif t == 3:
        accs_3_nb.append(accs_nb[i])
        models_3_nb.append(models_nb[i])

    elif t == 4:
        accs_4_nb.append(accs_nb[i])
        models_4_nb.append(models_nb[i])

    elif t == 5:
        accs_5_nb.append(accs_nb[i])
        models_5_nb.append(models_nb[i])

    elif t == 6:
        accs_6_nb.append(accs_nb[i])
        models_6_nb.append(models_nb[i])
```

```
accs_1_nb = np.array(accs_1_nb)
models_1_nb = np.array(models_1_nb)

accs_2_nb = np.array(accs_2_nb)
models_2_nb = np.array(models_2_nb)

accs_3_nb = np.array(accs_3_nb)
models_3_nb = np.array(models_3_nb)

accs_4_nb = np.array(accs_4_nb)
models_4_nb = np.array(models_4_nb)

accs_5_nb = np.array(accs_5_nb)
models_5_nb = np.array(models_5_nb)

accs_6_nb = np.array(accs_6_nb)
models_6_nb = np.array(models_6_nb)

all_val_nb = np.concatenate((accs_1_nb.reshape(4, 1), accs_2_nb.reshape(4, 1),
accs_3_nb.reshape(4, 1),
                             accs_4_nb.reshape(4, 1), accs_5_nb.reshape(4, 1),
accs_6_nb.reshape(4, 1)), axis=1)
print(all_val_nb.shape)

print("Validation Dataset Accuracy Subject vs. Dataset Table")
table_titles = ["Subject 1", "Subject 2", "Subject 3", "Subject 4", "Subject 5",
"Subject 6"]
table_rows = ["Maked", "Masked PCA", "Orginal", "Orginal PCA"]
sub2_table = pd.DataFrame(all_val_nb, table_rows, table_titles)
sub2_table.head()
print(accs_1_nb.shape)

X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0, accs_1_nb, color='b', width=0.15, label="Subject 1")
ax.bar(X + 0.15, accs_2_nb, color='g', width=0.15, label="Subject 2")
ax.bar(X + 0.30, accs_3_nb, color='r', width=0.15, label="Subject 3")
ax.bar(X + 0.45, accs_4_nb, color='orange', width=0.15, label="Subject 4")
ax.bar(X + 0.6, accs_5_nb, color='black', width=0.15, label="Subject 5")
ax.bar(X + 0.75, accs_6_nb, color='yellow', width=0.15, label="Subject 6")
ax.legend()
ax.set_xticklabels(('', "Masked", '', "Masked PCA", '', "Original Dataset", '',
'', "Original PCA Dataset"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title(
    "Validation Dataset Accuracy Results for All Subjects for Masked, Masked PCA,
Original, Original PCA Datasets")
ax.set_xlabel("Subject Number")

accs_masked_nb = []
accs_maskedPCA_nb = []
accs_org_nb = []
accs_orgPCA_nb = []

for i in range(6):
    if i == 0:
        model_h = models_1_nb
    elif i == 1:
```

```
    model_h = models_2_nb
elif i == 2:
    model_h = models_3_nb
elif i == 3:
    model_h = models_4_nb
elif i == 4:
    model_h = models_5_nb
else:
    model_h = models_6_nb

test_f = np_masked[(i * 3) + 2]
test_l = np_labels[(i * 3) + 2]

masked_m = model_h[0]

pre_test = masked_m.predict(test_f)

correct = 0
for j in range(test_l.shape[0]):
    if pre_test[j] == test_l[j]:
        correct += 1

accs_masked_nb.append((correct / test_l.shape[0]))

test_f = np_masked_PCA[(i * 3) + 2]
test_l = np_labels[(i * 3) + 2]

maskedPCA_m = model_h[1]

pre_test = maskedPCA_m.predict(test_f)

correct = 0
for j in range(test_l.shape[0]):
    if pre_test[j] == test_l[j]:
        correct += 1

accs_maskedPCA_nb.append((correct / test_l.shape[0]))

test_f = np_org[(i * 3) + 2]
test_l = np_labels[(i * 3) + 2]

org_m = model_h[2]

pre_test = org_m.predict(test_f)

correct = 0
for j in range(test_l.shape[0]):
    if pre_test[j] == test_l[j]:
        correct += 1

accs_org_nb.append((correct / test_l.shape[0]))

test_f = np_PCA[(i * 3) + 2]
test_l = np_labels[(i * 3) + 2]

pca_m = model_h[3]

pre_test = pca_m.predict(test_f)
```

```
    correct = 0
    for j in range(test_l.shape[0]):
        if pre_test[j] == test_l[j]:
            correct += 1

    accs_orgPCA_nb.append((correct / test_l.shape[0]))

X = np.arange(6)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X + 0.05, accs_masked_nb, color='b', width=0.20, label="Masked Dataset")
ax.bar(X + 0.25, accs_maskedPCA_nb, color='g', width=0.25, label="Masked PCA
Dataset")
ax.bar(X + 0.50, accs_org_nb, color='r', width=0.25, label="Orginal Dataset")
ax.bar(X + 0.70, accs_orgPCA_nb, color='orange', width=0.20, label="Original PCA
Dataset")
ax.legend()
ax.set_xticklabels(("", "Subject 1", "Subject 2", "Subject 3", "Subject 4",
"Subject 5", "Subject 6"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Test Dataset Accuracy Results for All Subjects for Masked, Masked
PCA, Original, Original PCA Datasets")
ax.set_xlabel("Subject Number")

accs_masked_nb = np.array(accs_masked_nb)
all_nb_test = np.array(
    [accs_masked_nb[0], accs_org_nb[1], accs_masked_nb[2], accs_masked_nb[3],
accs_masked_nb[4], accs_masked_nb[5]])

print(all_nb_test)
all_nb_test = all_nb_test.reshape(6, 1).T

table_titles = ["Subject 1", "Subject 2", "Subject 3", "Subject 4", "Subject 5",
"Subject 6"]
table_rows = ["Best Model Acc"]
sub2_table = pd.DataFrame(all_nb_test, table_rows, table_titles)
sub2_table.head()

X = np.arange(6)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X,
       [accs_masked_nb[0], accs_org_nb[1], accs_masked_nb[2], accs_masked_nb[3],
accs_masked_nb[4], accs_masked_nb[5]],
       color='green', width=0.5)
ax.set_xticklabels(("", "Subject 1", "Subject 2", "Subject 3", "Subject 4",
"Subject 5", "Subject 6"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Test Dataset Accuracy Results for All Subjects For Best Models")
ax.set_xlabel("Subject Number")

masked_m_nb = np.mean(accs_masked_nb)
masked_s_nb = np.std(accs_masked_nb) / 2

maskedPCA_m_nb = np.mean(accs_maskedPCA_nb)
maskedPCA_s_nb = np.std(accs_maskedPCA_nb) / 2

org_m_nb = np.mean(accs_org_nb)
org_s_nb = np.std(accs_org_nb) / 2
```

```
orgPCA_m_nb = np.mean(accs_orgPCA_nb)
orgPCA_s_nb = np.std(accs_orgPCA_nb) / 2

error_nb = [masked_s_nb, maskedPCA_s_nb, org_s_nb, orgPCA_s_nb]
means_nb = [masked_m_nb, maskedPCA_m_nb, org_m_nb, orgPCA_m_nb]

X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(X, [masked_m_nb, maskedPCA_m_nb, org_m_nb, orgPCA_m_nb], yerr=error_nb,
color='green', align='center', width=0.5,
       alpha=0.5, capsize=10)
ax.set_xticklabels(("", "Masked", "", "Masked PCA", "", "Original", "", "Original
PCA"))
ax.set_ylabel("Estimation Accuracy in Percent (%)")
ax.set_title("Test Dataset Mean Accuracy Results")
ax.set_xlabel("Subject Number")
```

Preprocess_for_2DCNN.py

```
import numpy as np
import matplotlib.pyplot as plt
import os

func_labels = np.load('labels.npy', allow_pickle=True)

func_org = np.load('func_org.npy', allow_pickle=True)
print(func_org[0].shape)
for i in range(6):
    func_org[i] = func_org[i].astype('float32')
    func_org[i] = np.transpose(func_org[i], (0,4,1,2,3))
    func_org[i] = np.vstack(func_org[i])
    print(func_org[i].shape)
print(func_org.shape)
func_org = np.vstack(func_org)
print(func_org.shape)

SAVE_DIR = '3D_Data_for_CNN'
if not os.path.exists(SAVE_DIR):
    os.mkdir(SAVE_DIR)
np.save(os.path.join(SAVE_DIR, '3d_features.npy'), func_org, allow_pickle=True)
```

Preprocess_for_3DCNN.py

```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split

func_labels = np.load('labels.npy', allow_pickle=True)
print(func_labels[0].shape)

all_unique_labels = []
for i in range(6):
    unique_labels = []
    pres_label = ''
```

```
    for j in range(len(func_labels[2])):
        if (pres_label != func_labels[2][j]):
            pres_label = func_labels[2][j]
            unique_labels.append(pres_label)
    unique_labels = np.asarray(unique_labels)
    all_unique_labels.append(unique_labels)
all_unique_labels = np.asarray(all_unique_labels)
print(all_unique_labels.shape)

print(func_labels[0])

func_org = np.load('func_org.npy', allow_pickle=True)
print(func_org[0].shape)
for i in range(6):
    func_org[i] = func_org[i].astype('float32')
    func_org[i] = np.transpose(func_org[i], (0, 4, 1, 2, 3))
    func_org[i] = np.asarray(np.split(func_org[i], indices_or_sections=8, axis=1))
    print(func_org[i].shape)
    func_org[i] = np.vstack(func_org[i])
    print(func_org[i].shape)
print(func_org[0].shape)

print(all_unique_labels.shape)
run_list = [12, 12, 12, 12, 11, 12]

labels = []
i = 0
for runs in run_list:
    temp = np.asarray(([all_unique_labels[i]] * runs)).ravel().T
    print(temp.shape)
    labels.append(temp)
    i += 1
# labels = np.asarray(np.hstack(labels))
labels = np.asarray(labels)
print(labels.shape)

SAVE_DIR = 'data_final'
if not os.path.exists(SAVE_DIR):
    os.mkdir(SAVE_DIR)

for i in range(6):
    # subject save directory
    SUB_SAVE_DIR = os.path.join(SAVE_DIR, 'subj' + str(i + 1))
    if not os.path.exists(SUB_SAVE_DIR):
        os.mkdir(SUB_SAVE_DIR)

    # split org
    X_train, X_test, y_train, y_test = train_test_split(func_org[i], labels[i],
test_size=0.2, random_state=24)
    X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5,
random_state=24)

    folders = ['temporal', 'labels_temporal']
    for folder in folders:
        TYPE_SAVE_DIR = os.path.join(SUB_SAVE_DIR, folder)
        if not os.path.exists(TYPE_SAVE_DIR):
            os.mkdir(TYPE_SAVE_DIR)

        if (folder == 'temporal'):
```

```
            # save org
            np.save(os.path.join(TYPE_SAVE_DIR, "train_features.npy"), X_train,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "val_features.npy"), X_val,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "test_features.npy"), X_test,
allow_pickle=True)
        elif (folder == 'labels_temporal'):
            # save labels
            np.save(os.path.join(TYPE_SAVE_DIR, "train_labels.npy"), y_train,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "val_labels.npy"), y_val,
allow_pickle=True)
            np.save(os.path.join(TYPE_SAVE_DIR, "test_labels.npy"), y_test,
allow_pickle=True)

# unified subjects (5training 1test subject)
func_org_train = np.vstack(func_org[1:5])
func_org_test = func_org[0]
del func_org
print(func_org_train.shape, func_org_test.shape)
labels_train = np.hstack(labels[1:5])
labels_test = labels[0]
print(labels_train.shape, labels_test.shape)
shuffle_indices = np.random.permutation(labels_train.shape[0])
func_org_train = func_org_train[shuffle_indices]
labels_train = labels_train[shuffle_indices]
SAVE_DIR = os.path.join('data_final', 'subj_uni')
if not os.path.exists(SAVE_DIR):
    os.mkdir(SAVE_DIR)
np.save(os.path.join(SAVE_DIR, "train_features.npy"), func_org_train,
allow_pickle=True)
np.save(os.path.join(SAVE_DIR, "test_features.npy"), func_org_test,
allow_pickle=True)
np.save(os.path.join(SAVE_DIR, "train_labels.npy"), labels_train,
allow_pickle=True)
np.save(os.path.join(SAVE_DIR, "test_labels.npy"), labels_test, allow_pickle=True)

# mixed subjects
func_org = np.vstack(func_org)
labels = np.hstack(labels)
shuffle_indices = np.random.permutation(func_org.shape[0])
func_org = func_org[shuffle_indices]
labels = labels[shuffle_indices]

X_train, X_test, y_train, y_test = train_test_split(func_org, labels,
test_size=0.2, random_state=24)

SAVE_DIR = os.path.join('data_final', 'subj_uni_mixed')
if not os.path.exists(SAVE_DIR):
    os.mkdir(SAVE_DIR)

np.save(os.path.join(SAVE_DIR, "train_features.npy"), X_train, allow_pickle=True)
np.save(os.path.join(SAVE_DIR, "test_features.npy"), X_test, allow_pickle=True)
np.save(os.path.join(SAVE_DIR, "train_labels.npy"), y_train, allow_pickle=True)
np.save(os.path.join(SAVE_DIR, "test_labels.npy"), y_test, allow_pickle=True)
```

CNN_on_Original_without_Time.py

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential, Model
import tensorflow.keras.layers as layers
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.losses import SparseCategoricalCrossentropy

DATA_DIR = '3D_Data_for_CNN/'
features = np.load(os.path.join(DATA_DIR,'3d_features.npy'), allow_pickle=True)
labels = np.load(os.path.join(DATA_DIR, '3d_labels.npy'), allow_pickle=True)
print(features.shape)
print(labels.shape)

LABELS = ['scissors', 'face', 'cat', 'scrambledpix', 'bottle', 'chair', 'shoe',
'house']
LABELS_INT = [0,1,2,3,4,5,6,7]
label_dict = dict()
rev_label_dict = dict()
i=0
for label in LABELS:
    label_dict[label] = i
    rev_label_dict[i] = label
    i+=1
print(label_dict)
print(rev_label_dict)
int_labels = []
for i in range(labels.shape[0]):
    int_labels.append(label_dict[labels[i]])
print(labels[0:20])
print(int_labels[0:20])
int_labels = np.asarray(int_labels)
print(int_labels.shape)

print(features.shape)
print(int_labels.shape)
features = np.transpose(features, (0,2,3,1))

X_train, X_test, y_train, y_test = train_test_split(features, int_labels,
test_size=0.1, random_state=24)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

model1 = Sequential()
model1.add(layers.Conv2D(32, kernel_size=(3,3),
activation='relu',kernel_initializer='he_uniform', input_shape=(64,64,40)))
model1.add(layers.BatchNormalization())
model1.add(layers.MaxPooling2D((2,2)))
model1.add(layers.Conv2D(64, kernel_size=(3,3), activation='relu',
kernel_initializer='he_uniform'))
model1.add(layers.BatchNormalization())
model1.add(layers.MaxPooling2D((2,2)))
model1.add(layers.Conv2D(64, kernel_size=(3,3), activation='relu',
kernel_initializer='he_uniform'))
model1.add(layers.BatchNormalization())
model1.add(layers.MaxPooling2D((2,2)))
```

```
model1.add(layers.Flatten())
model1.add(layers.Dense(258, activation='relu', kernel_initializer='he_uniform'))
model1.add(layers.Dense(64, activation='relu', kernel_initializer='he_uniform'))
model1.add(layers.Dense(len(LABELS), activation='softmax'))
model1.summary()

model1.compile(loss=SparseCategoricalCrossentropy(),
optimizer=SGD(learning_rate=0.00075), metrics=['accuracy'])
history = model1.fit(X_train, y_train, batch_size=64, epochs=95,
validation_split=0.1113)

print(features.shape)
plt.imshow(features[0,:,:,20], cmap='gray')
plt.show()
plt.imshow(features[0,32,:,:], cmap='gray')
plt.show()
plt.imshow(features[0,:,32,:], cmap='gray')
plt.show()
```

3D_CNN_Subject.py

```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn import svm
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, optimizers, losses, models
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

DATA_DIR = 'data_final'

def accuracy(y_pred, y_real):
    num_true = np.sum(y_pred == y_real)
    return num_true / y_real.shape[0]

le = LabelEncoder()
LABELS = ['scissors', 'face', 'cat', 'scrambledpix', 'bottle', 'chair', 'shoe',
'house']
NUM_LABELS = len(LABELS)
le.fit(LABELS)

# PCA training and validation
best_models = []
for i in range(1, 7):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels_temporal')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'temporal')

    X_train = np.load(os.path.join(FEATURES_DIR, 'train_features.npy'),
allow_pickle=True)
    X_train = np.transpose(X_train, (0, 2, 3, 4, 1))
    y_train = np.load(os.path.join(LABELS_DIR, 'train_labels.npy'),
allow_pickle=True)
    y_train = le.transform(y_train)

    print('Train data for subject %d loaded: %s %s' % (i, X_train.shape,
y_train.shape))
```

```
    X_val = np.load(os.path.join(FEATURES_DIR, 'val_features.npy'),
allow_pickle=True)
    X_val = np.transpose(X_val, (0, 2, 3, 4, 1))
    y_val = np.load(os.path.join(LABELS_DIR, 'val_labels.npy'), allow_pickle=True)
    y_val = le.transform(y_val)
    # print(y_val)

    batch_size = 1
    no_epochs = 45
    learning_rates = [0.00005]
    best_val_acc = 0
    for lr in learning_rates:
        model = models.Sequential()
        model.add(layers.Conv3D(8, kernel_size=(3, 3, 3), strides=(1, 1, 1),
activation='relu',
                                kernel_initializer='he_uniform', input_shape=(40,
64, 64, 9)))
        model.add(layers.BatchNormalization())
        model.add(layers.MaxPooling3D(pool_size=(2, 2, 2)))
        # model.add(layers.Dropout(0.35))
        # model.add(layers.Conv3D(64, kernel_size=(3, 3, 3),  activation='relu',
kernel_initializer='he_uniform'))
        # model.add(layers.BatchNormalization())
        # model.add(layers.MaxPooling3D(pool_size=(2, 2, 2)))
        # model.add(layers.Dropout(0.35))
        model.add(layers.Flatten())
        model.add(layers.Dense(512, activation='relu',
kernel_initializer='he_uniform'))
        model.add(layers.Dense(128, activation='relu',
kernel_initializer='he_uniform'))
        model.add(layers.Dense(NUM_LABELS, activation='softmax'))

        model.summary()

        model.compile(loss=losses.sparse_categorical_crossentropy,
optimizer=optimizers.SGD(lr=lr, momentum=0.3),
                      metrics=['accuracy'])
        model.fit(X_train, y_train, batch_size=batch_size, epochs=no_epochs,
validation_data=(X_val, y_val))
        predictions = np.argmax(model.predict(X_val), axis=1)
        print(predictions, y_val)
        val_acc = accuracy(predictions, y_val)
        print('Validation Accuracy for Learning Rate %0.4f: %f' % (lr, val_acc))

        if (val_acc >= best_val_acc):
            best_val_acc = val_acc
            best_model = model

    best_models.append(best_model)

print('\n\n----------------------------------------\n\n')

# PCA test
test_accuracies = []
for i, model in zip(range(1, 7), best_models):
    SUBJ_DIR = os.path.join(DATA_DIR, 'subj' + str(i))
    LABELS_DIR = os.path.join(SUBJ_DIR, 'labels_temporal')
    FEATURES_DIR = os.path.join(SUBJ_DIR, 'temporal')
```

```
    X_test = np.load(os.path.join(FEATURES_DIR, 'test_features.npy'),
allow_pickle=True)
    X_test = np.transpose(X_test, (0, 2, 3, 4, 1))
    y_test = np.load(os.path.join(LABELS_DIR, 'test_labels.npy'),
allow_pickle=True)

    print('Test data for subject %d loaded: %s %s' % (i, X_test.shape,
y_test.shape))
    print('Best Model: %s' % model)

    predictions = model.predict(X_test)
    test_accuracies.append(accuracy(predictions, y_test))
print('Test Accuracies for Each subject', test_accuracies)
print('Mean and Std of Accuracies: %f, %f' % (np.mean(test_accuracies),
np.std(test_accuracies)))

# plot means and stds
means = 100 * np.asarray([np.mean(test_accuracies_org),
np.mean(test_accuracies_masked), np.mean(test_accuracies_pca),
                          np.mean(test_accuracies_maskedpca)])
stds = 100 * np.asarray([np.std(test_accuracies_org),
np.std(test_accuracies_masked), np.std(test_accuracies_pca),
                         np.std(test_accuracies_maskedpca)])
reducs = ['Original', 'Masked', 'PCA', 'Masked+PCA']

fig, ax = plt.subplots()
ax.bar(np.arange(4), means, yerr=stds, align='center', alpha=0.5, ecolor='black',
capsize=10)
ax.set_yticks(np.arange(0, 105, 10))
ax.set_ylabel('Test Accuracy')
ax.set_xticks(np.arange(4))
ax.set_xticklabels(reducs)
ax.set_title('Mean Test Accuracies for SVM \nwith Different Feature Reduction
Pipelines')
ax.grid(True)

# Save the figure and show
plt.tight_layout()
plt.show()
```

3D_CNN_Unified.py

```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn import svm
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, optimizers, losses, models
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

DATA_DIR = 'data_final'

def accuracy(y_pred, y_real):
    num_true = np.sum(y_pred == y_real)
    return num_true / y_real.shape[0]
```

```
le = LabelEncoder()
LABELS = ['scissors', 'face', 'cat', 'scrambledpix', 'bottle', 'chair', 'shoe',
'house']
NUM_LABELS = len(LABELS)
le.fit(LABELS)

SUBJ_DIR = os.path.join(DATA_DIR, 'subj_uni_mixed')

X_train = np.load(os.path.join(SUBJ_DIR, 'train_features.npy'), allow_pickle=True)
X_train = np.transpose(X_train, (0, 2, 3, 4, 1))
y_train = np.load(os.path.join(SUBJ_DIR, 'train_labels.npy'), allow_pickle=True)
y_train = le.transform(y_train)

print('Train data is loaded: %s %s' % (X_train.shape, y_train.shape))
# del model

batch_size = 1
no_epochs = 40
learning_rate = 0.0005
best_val_acc = 0

model = models.Sequential()
model.add(
    layers.Conv3D(16, kernel_size=(3, 3, 3), strides=(1, 1, 1), activation='relu',
kernel_initializer='he_uniform',
                  input_shape=(40, 64, 64, 9)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling3D(pool_size=(2, 2, 2)))
model.add(
    layers.Conv3D(32, kernel_size=(3, 3, 3), strides=(1, 1, 1), activation='relu',
kernel_initializer='he_uniform'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling3D(pool_size=(2, 2, 2)))
model.add(
    layers.Conv3D(64, kernel_size=(3, 3, 3), strides=(1, 1, 1), activation='relu',
kernel_initializer='he_uniform'))
model.add(layers.SpatialDropout3D(0.2))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling3D(pool_size=(2, 2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu', kernel_initializer='he_uniform'))
model.add(layers.Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(layers.Dense(NUM_LABELS, activation='softmax'))

model.summary()

model.compile(loss=losses.sparse_categorical_crossentropy,
              optimizer=optimizers.Adam(lr=learning_rate, beta_1=0.4, beta_2=0.8),
metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=batch_size, epochs=no_epochs,
validation_split=0.1)

X_test = np.load(os.path.join(SUBJ_DIR, 'test_features.npy'), allow_pickle=True)
X_test = np.transpose(X_test, (0, 2, 3, 4, 1))
y_test = np.load(os.path.join(SUBJ_DIR, 'test_labels.npy'), allow_pickle=True)
y_test = le.transform(y_test)

print('Test data is loaded: %s %s' % (X_test.shape, y_test.shape))
```

```
pred = model.predict(X_test)
print(accuracy(np.argmax(pred, axis=1), y_test))

plt.imshow(X_train[0, :, :, 20, 0])
plt.show()
plt.imshow(X_train[0, :, 20, :, 0])
plt.show()
plt.imshow(X_train[0, 20, :, :, 0])
plt.show()

plt.imshow(X_test[0, :, :, 20, 0])
plt.show()
plt.imshow(X_test[0, :, 20, :, 0])
plt.show()
plt.imshow(X_test[0, 20, :, :, 0])
plt.show()
```