RECONSTRUCTING PERCEIVED IMAGES FROM HUMAN BRAIN ACTIVITIES USING TWIN DEEP NEURAL NETWORK

Team Members

1. Dr. N. Pughazendi

2. Piyush Chouhan M

3. Mohanavel R

4. Kedar R

RECONSTRUCTING PERCEIVED IMAGES FROM HUMAN BRAIN ACTIVITIES USING TWIN DEEP NEURAL NETWORK

ABSTRACT

Understanding the Human Visual System necessitates the use of neural decoding. The human visual system is capable of extracting and comparing features from any object. The majority of the articles are concerned with either the classification of brain function patterns or the recognition of visual stimuli. We implement the Twin Deep Neural Network model for accurate image reconstruction in this project. We present the Twin Deep Neural Network model for accurate image reconstruction from human brain activity using functional Magnetic Resonance Imaging in this research (fMRI). For better visual reconstruction and to make full use of each dataset, the TDNN method may be used to compare the relationship between a sample pair of similar features. The TDNN approach reduces the constraints imposed by high dimensionality and a limited amount of FMRI data. Essentially, this method will increase the training data from N samples to 2N sample pairs, allowing the limited number of training samples to be fully utilised. On an open dataset of handwritten digital images and character datasets, we found that the proposed TDNN approach outperformed all current state-of-the-art methods on the Convolutional Neural Network by about 10%. (CNN).

I. INTRODUCTION

In the past, understanding the human brain's visual processing was a crucial and difficult issue. In the computing field, mathematical and probabilistic models have revolutionised many applications, including identification, computer vision, and classification problems. The human brain's processing is first implemented

using linear models, and then perceptrons are created, which are more useful and have the natural properties of the human brain. Although the feature sample for several types of applications is different and simpler after the emerging of convolution neural network is a more upgradeable version of deep neural network and also has good feature extraction in the model help to improve accuracy and overcome issues, a complex structure of visual stimuli and high dimension data is challenging issues.

Deep neural networks have been used in a variety of fields to solve problems such as reducing high-dimensional data to low dimensions and capturing key features. Deep learning and generative models are being developed to improve image processing and decode high-dimensional data. Autoencoders are a simple generative model that can generate high-dimensional data using a neural network while capturing important features without compromising data originality. As a result, the concept of neural encoding and decoding in a single model emerges. The term "neural encoding" refers to the process of forecasting voxel activation from visual stimuli and vice versa. Voxel activation classification and prediction of visual stimuli are two types of human brain processing. It is possible to recreate visual stimuli using a combination of these decoding and encoding techniques.

II. PROBLEM DEFINITION

Understanding the human visual system and reconstructing recognised images from human brain activity For visual representation, use a convolutional neural network to implement a variational autoencoder, and then use a Gaussian mixture model to cluster the same variants of class in the latent space vector.

III. SYSTEM ARCHITECTURE

3.1 ARCHITECTURE OVERVIEW

Three modules make up the proposed model. The first stage entails collecting input data from the brain linear repository, such as visual handwritten digits and FMRI brain activity, and then fetching the data to the inference model, where GMM is used to cluster.

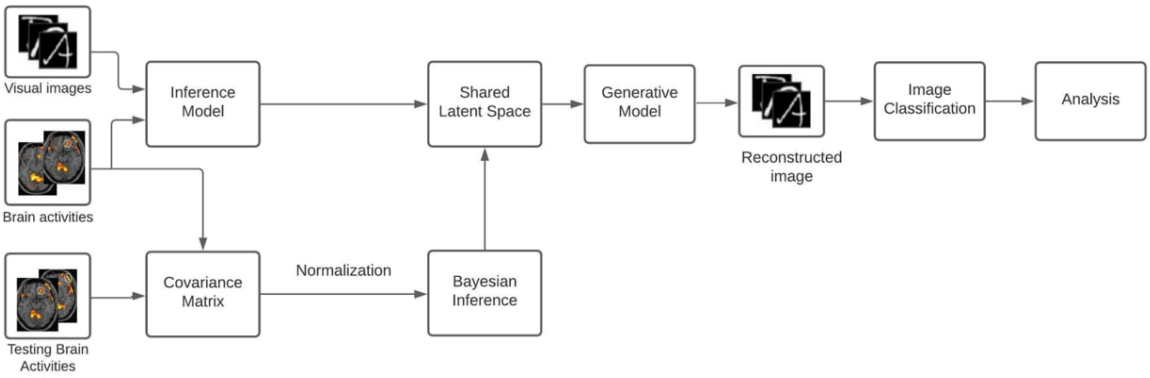


Figure 3.1 Architecture Overview

3.2 NEURAL NETWORK ARCHITECTURE

Model: "TDNN"

_____

_

Layer (type) Output Shape Param # Connected to

==========

=

input_1 (InputLayer) (None, 1, 28, 28) 0

_____

_

Firstlayer (Conv2D) (None, 1, 28, 1) 113 input_1[0][0]

_____

_

Secondlayer (Conv2D) (None, 1, 14, 64) 320 Firstlayer[0][0]

_____

_

Thirdlayer (Conv2D) (None, 1, 14, 64) 36928 Secondlayer[0][0]

_____

Fourthlayer (Conv2D) (None, 1, 14, 64) 36928 Thirdlayer[0][0]

_____

flatten_1 (Flatten) (None, 896) 0 Fourthlayer[0][0]

hiddenlayer (Dense) (None, 128) 114816 flatten_1[0][0]

Z_1 (Dense) (None, 6) 774 hiddenlayer[0][0]

Z_2 (Dense) (None, 6) 774 hiddenlayer[0][0]

lambda_1 (Lambda) (None, 6) 0 Z_1[0][0]
Z_2[0][0]

dense_1 (Dense) (None, 128) 896 lambda_1[0][0]

dense_2 (Dense) (None, 12544) 1618176 dense_1[0][0]

reshape_1 (Reshape) (None, 14, 14, 64) 0 dense_2[0][0]

conv2d_transpose_1 (Conv2DTrans (None, 14, 14, 64) 36928 reshape_1[0][0]

conv2d_transpose_2 (Conv2DTrans (None, 14, 14, 64) 36928 conv2d_transpose_1[0][0]

conv2d_transpose_3 (Conv2DTrans (None, 29, 29, 64) 36928 conv2d_transpose_2[0][0]

conv2d_1 (Conv2D) (None, 28, 28, 1) 257 conv2d_transpose_3[0][0]

==========
=
Total params: 1,920,766
Trainable params: 1,920,766
Non-trainable params: 0

IV. REQUIREMENT ANALYSIS

4.1 INPUT REQUIREMENTS

Visual stimulation and brain mappings are two types of datasets that the proposed system is supposed to obtain. A working magnetic resonance imaging system and a head-mounted image visualizer device are used to build the dataset. The subject is placed on a FMRI computer with a head-mounted system and various visual image frames are projected on the screen. The FMRI system is then turned on and a magnetic resonance image of the human subject is taken. The individual cross-section views of the brains are plotted and then combined into a 3D brain voxel

of 3092 parts. This 3D brain now represents two brain states: active and dormant. The blood flow on a specific part of the cerebrum is represented by these active regions. After that, the 3D brain is converted to a 2D flat map surface, which is then streamlined into a 1D plane. This 3D to 1D conversion process allows us to catch the bulk of brain activity. These mental processes are then classified as one of the input dataset forms. The visual frames that the person is exposed to are viewed as a different form of dataset during this phase.

4.2 FUNCTIONAL REQUIREMENTS

As inputs, the device is supposed to obtain two types of datasets: visual images and brain mappings. It is assumed that an inference model fed with perception datasets would derive an encoding schema from the input visual stimulus dataset. The encoding model's inference is subjected to a mutual latent space test. This shared latent space serves as a popular knowledge base, with inferences modified on a regular basis. The generative multi-view model also makes use of the knowledge-based structure. From the shared knowledge-based and input visual stimulus datasets, this generative model will recreate photos. To assess the consistency of the reconstructed images further, they are subjected to various classification algorithms to determine the accuracy of label classification. Bypassing the standard MNIST dataset and the restored image dataset in the classifier algorithm, this can be accomplished.

4.3 TECHNOLOGY STACK

The major tools that have been used in our project are

- ANACONDA
- ANGULAR 10
- TENSORFLOW
- KERAS

- FLASK
- BOOTSTRAP
- NODE

Anaconda Navigator is the latest python repository and also it has been used to initiate applications with a major ide provide environment. It is free software for all users with python and r programming languages that are emerging in mathematical models and computer vision. It has been provided with the user interface as well as a command-line that is greatly helpful to users. It has integrated with the pre-built library packages and channels for programming languages.

Spyder is a cross platform python programming ide for mathematical models and computer vision. It is free software for all users integrates with basic package of python and also includes mathematical models packages like numpy, pandas, etc. It is best ide for project work and included in anaconda navigator for easy to install packages at once.

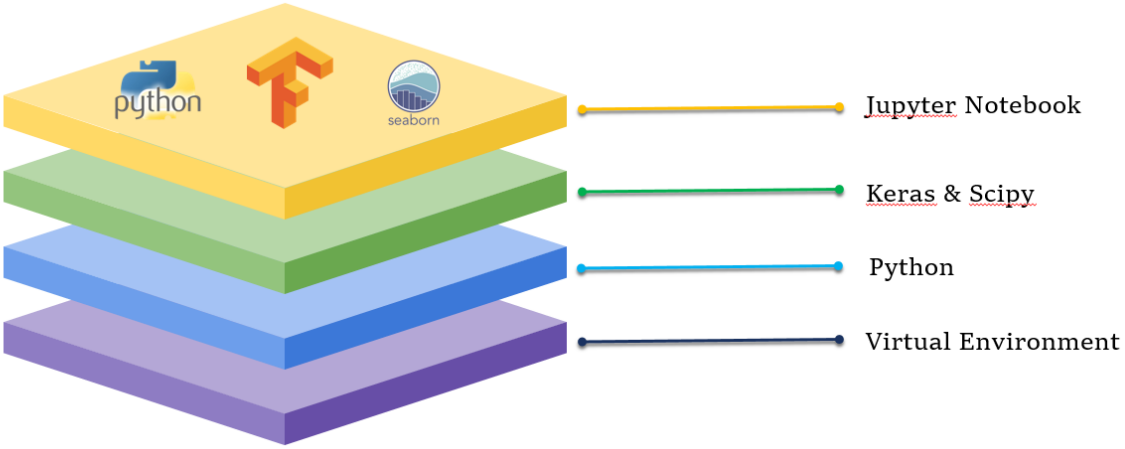


Figure 4.1 Technology Stack

V. CONCLUSION

Brain decoding plays an important role in brain machine interfaces by helping disabled persons in expressing and promoting the development of the brain mechanism. The development of functional Magnetic Resonance Imaging (FMRI), it has become an effective tool in understanding brain activity. Our future work includes applications of our method to FMRI mental illness classification problem.

This will help the neurologist to better understand the health conditions of a paralyzed patient. It can be evolved to produce the dream visuals of a person which is still a mystery to solve. This can open doors to several mysteries behind neurology. Still we are at 21st century but storing and playing a dream is still an issue. With the help of this technology, we can be able to reconstruct the dream using Generative Adversarial Neural network (GAN) networks.

CODING

- **PROGRAM DESIGN LANGUAGE**

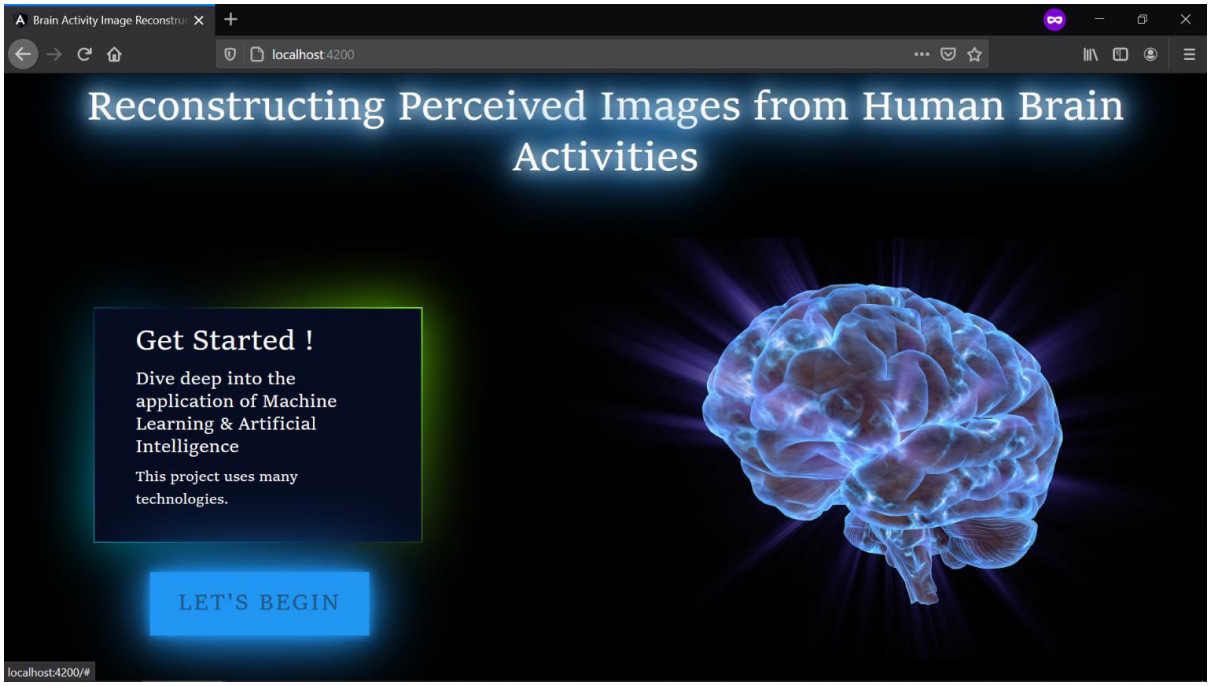
```
import sys
from inference_model import InferenceModel
from shared_latent_space import SharedLatentSpace
from decoder_train import DecoderTrain
from training import Training
from testing import Testing
from reconstructor import Reconstructor
from visualizer import Visualizer
import warnings
warnings.filterwarnings("ignore")
arguments = sys.argv

inference_model = InferenceModel()
if len(arguments)>1:
    iteration = arguments[1]
    epochs = arguments[2]
    inference_model.Iteration= int(iteration)
    inference_model.Epoch = int(epochs)

visualizer = Visualizer(inference_model)
shared_latent_space = SharedLatentSpace()
shared_latent_space.calculateZ(inference_model)
decoder = DecoderTrain(inference_model,shared_latent_space)
training = Training(inference_model,decoder)
```

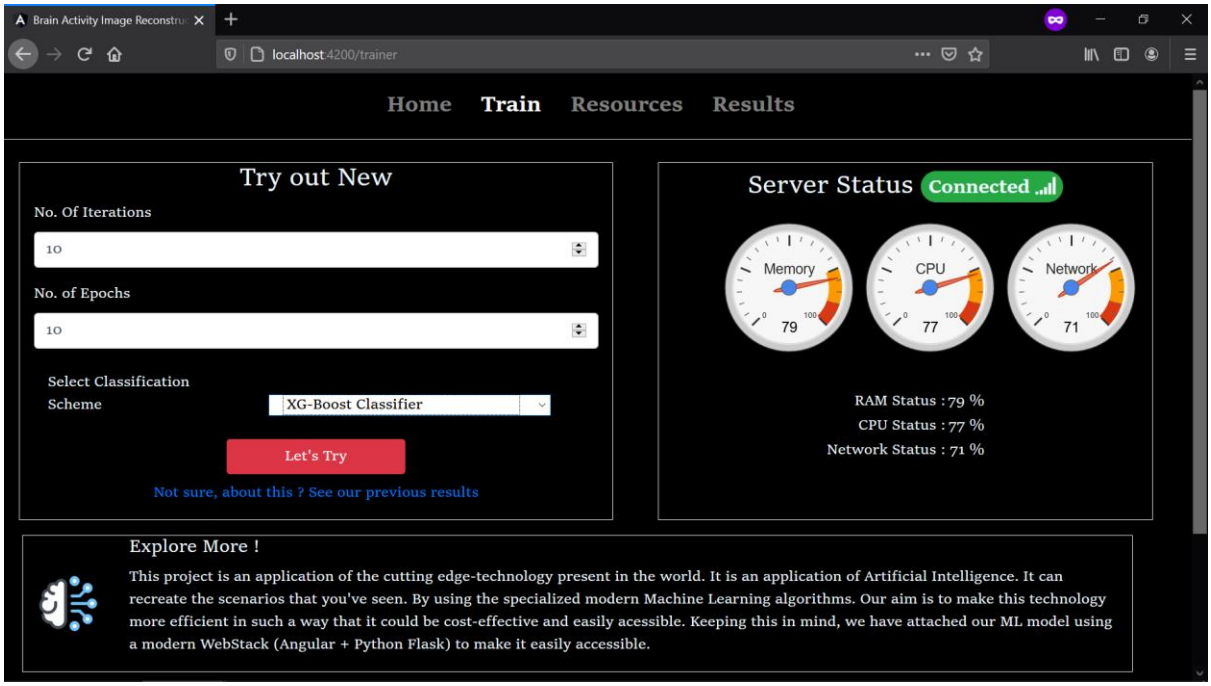
```
testing = Testing(inference_model,decoder)

reconstructor = Reconstructor()

reconstructor.constructXtest(inference_model,training,testing)

reconstructor.constructXtrain(inference_model,training,testing)


visualizer.data_display(inference_model.XY_TrainLength,reconstructor.
X_reconstructed_train,'X_reconstructed_train.png',False,False)

visualizer.data_display(inference_model.XY_TrainLength,inference_mo
del.X_train,'X_train.png',False,False)

visualizer.data_display(inference_model.XY_TestLength,reconstructor.X
_reconstructed_test,'X_reconstructed_test.png',False,True)

visualizer.data_display(inference_model.XY_TestLength,inference_mod
el.X_test,'X_test.png',False,True)

visualizer.plotter(inference_model,reconstructor)


class DecoderTrain():

   def __init__(self,inference_model,shared_latent_space):

      self.decoder_hiddenlayer                                    =
Dense(inference_model.CenterDimension, activation='relu')

      self.decoder_up = Dense(inference_model.filters * 14 * 14,
activation='relu')


      if backend.image_data_format() == 'channels_first':

        output_size=(inference_model.batch_size,inference_model.filters,
4, 14)

      else:

         output_size = (inference_model.batch_size, 14, 14,
inference_model.filters)
```

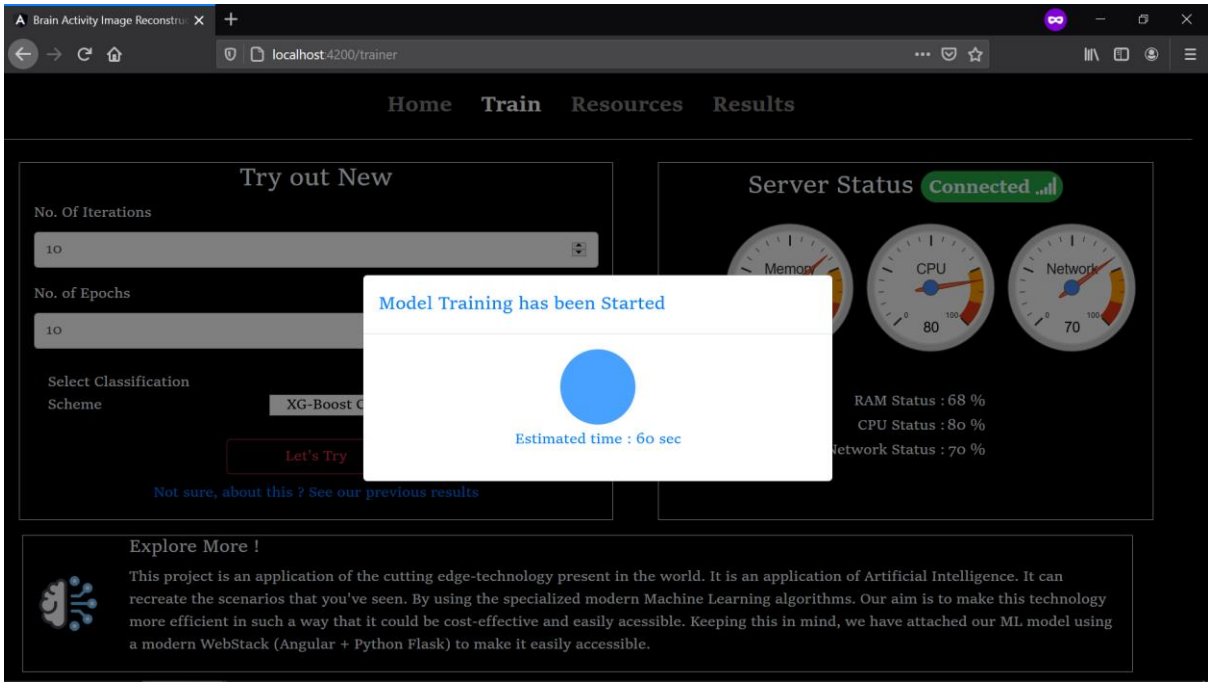
```
        self.decoder_reshape = Reshape(output_size[1:])

        self.decoder_Firstlayer = Conv2DTranspose(inference_model.filters, kernel_size=inference_model.convsize, padding='same', strides=1, activation='relu')

        self.decoder_Secondlayer = Conv2DTranspose(inference_model.filters, kernel_size=inference_model.convsize, padding='same', strides=1, activation='relu')


        if backend.image_data_format() == 'channels_first':

            output_size = (inference_model.batch_size, 29, 29, inference_model.filters)

        else:

            output_size = (inference_model.batch_size, inference_model.filters, 29, 29)


        self.decoder_Thirdlayer = Conv2DTranspose(inference_model.filters, kernel_size=(3, 3), strides=(2, 2), padding='valid', activation='relu')

        self.decoder_1 = Conv2D(inference_model.channel, kernel_size=2, padding='valid', activation='sigmoid')


        x_hiddenlayer = self.decoder_hiddenlayer(shared_latent_space.Z)

        x_up = self.decoder_up(x_hiddenlayer)

        x_reshape = self.decoder_reshape(x_up)

        x_Firstlayer = self.decoder_Firstlayer(x_reshape)

        x_Secondlayer = self.decoder_Secondlayer(x_Firstlayer)
```

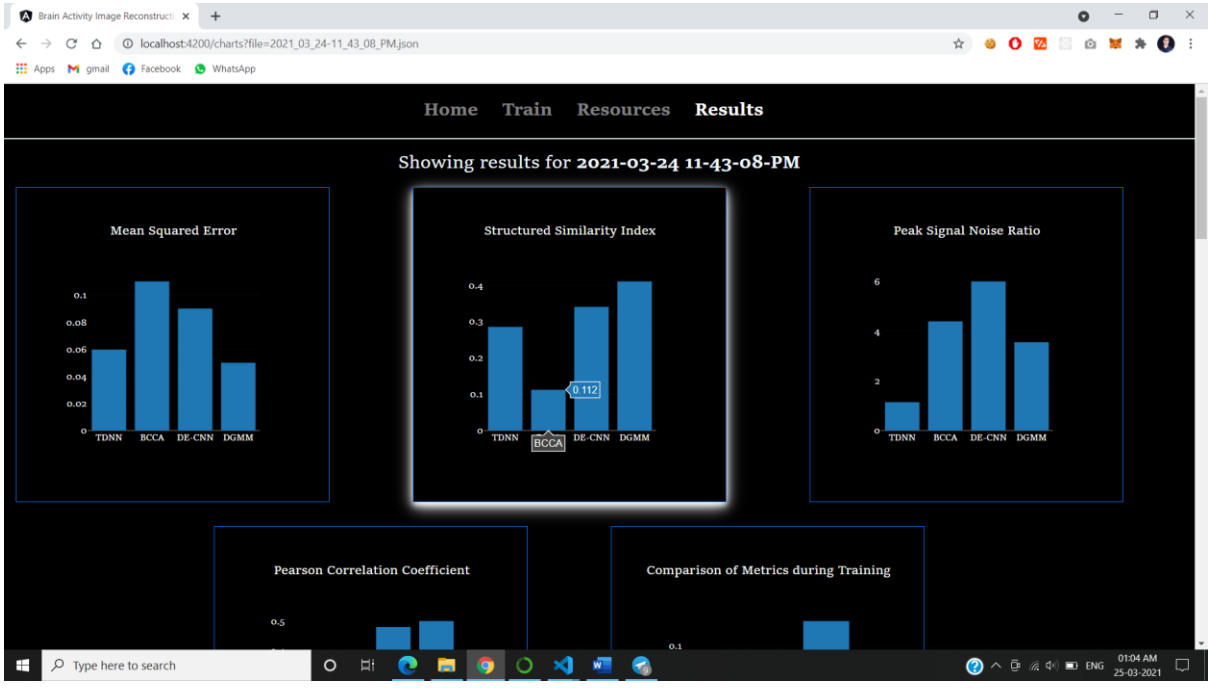
```
x_Thirdlayer = self.decoder_Thirdlayer(x_Secondlayer)

X_1 = self.decoder_1 (x_Thirdlayer)


logc = np.log(2 * np.pi)

def GMM(Y, Y_1, Y_2):

    return backend.mean(-(0.5 * logc + 0.5 * Y_2) - 0.5 * ((Y - Y_1)**2 / backend.exp(Y_2)), axis=-1)

def LossFunction(X, X_1):

    X = backend.flatten(X)

    X_1 = backend.flatten(X_1)

    Lp = 0.5 * backend.mean( 1 + inference_model.Z_2 - backend.square(inference_model.Z_1) - backend.exp(inference_model.Z_2), axis=-1)

    Lx =  - metrics.binary_crossentropy(X, X_1) # Pixels have a Bernoulli distribution

    Ly = GMM(inference_model.Y, inference_model.Y_1, inference_model.Y_2) # Voxels have a Gaussian distribution

    loss = backend.mean(Lp + 10000 * Lx + Ly)

    totalloss = - loss

    return  totalloss


self.TDNN= Model(inputs=[inference_model.X, inference_model.Y, inference_model.Y_1, inference_model.Y_2], outputs=X_1,name = 'TDNN')

opt_method = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

self.TDNN.compile(optimizer = opt_method, loss = LossFunction)

self.TDNN.summary()
```

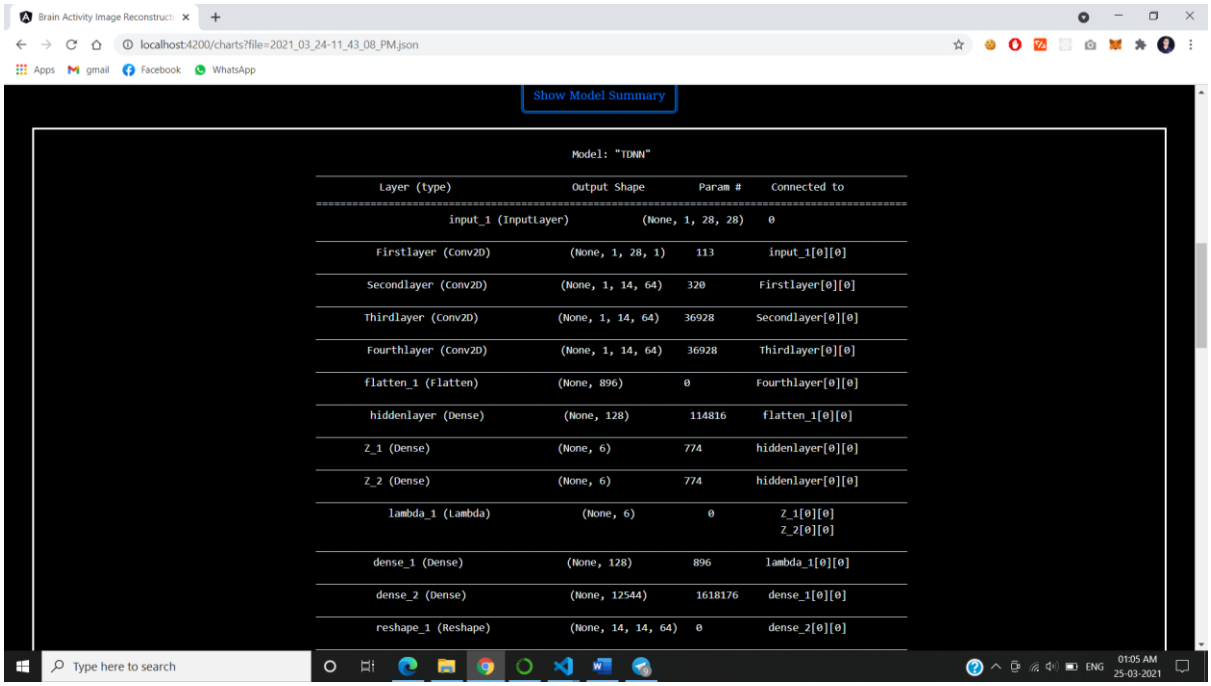
SCREENSHOTS

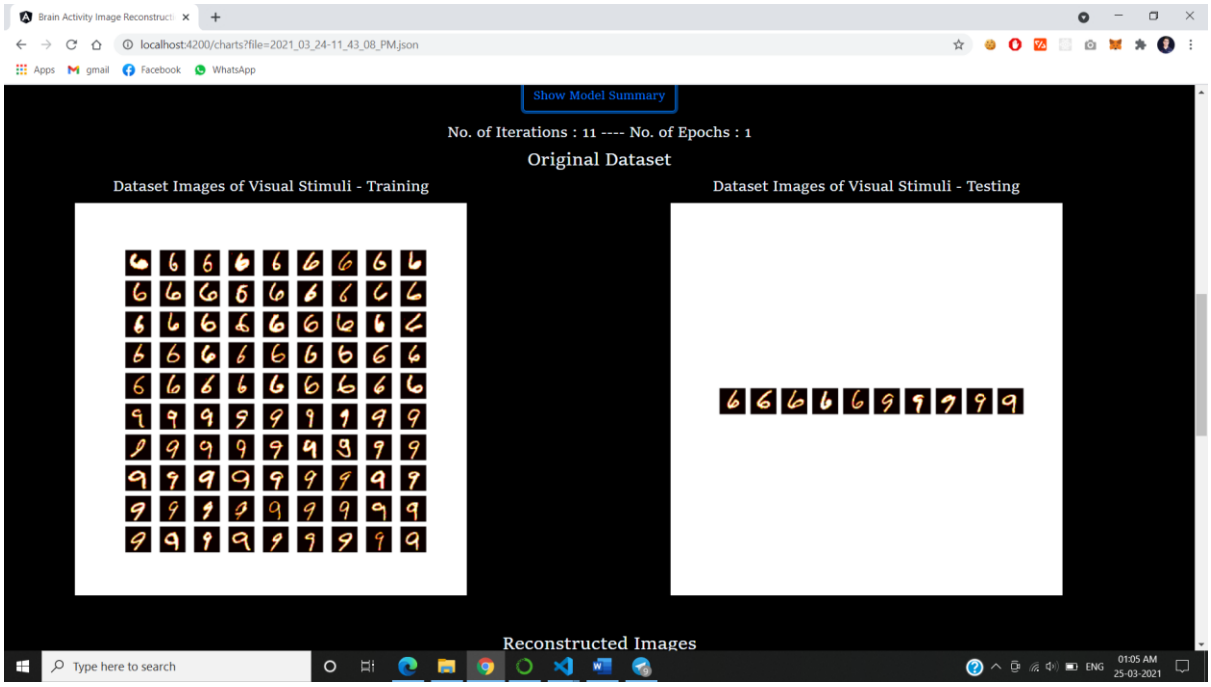


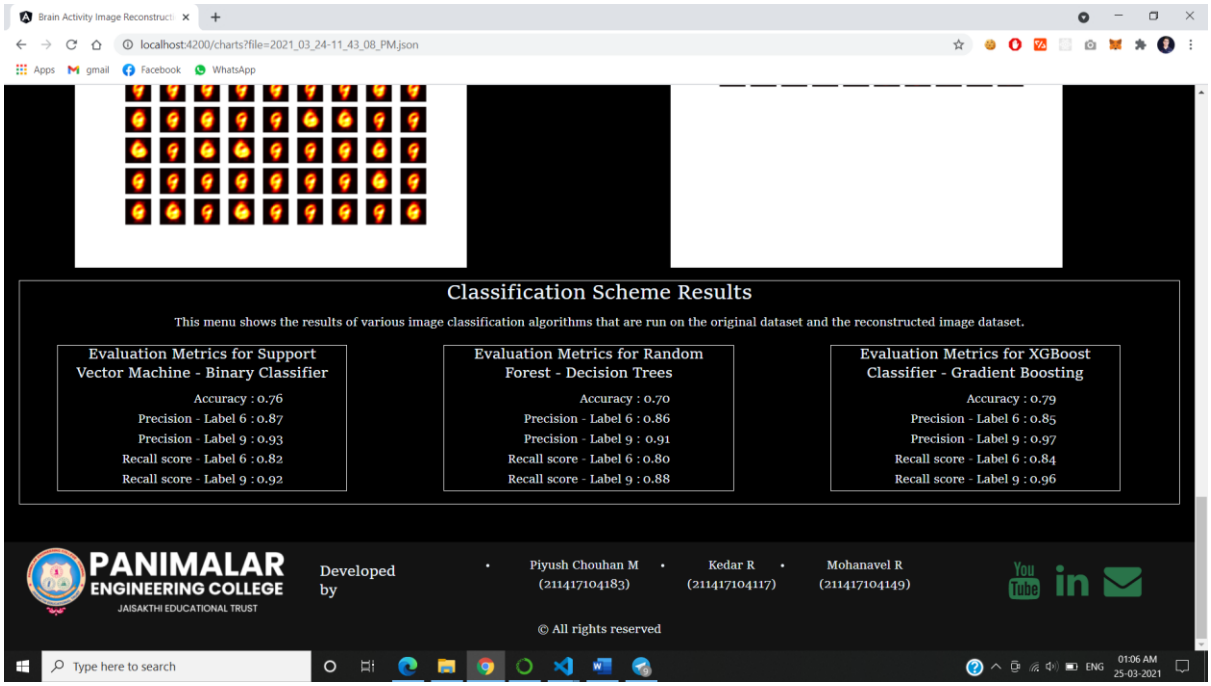


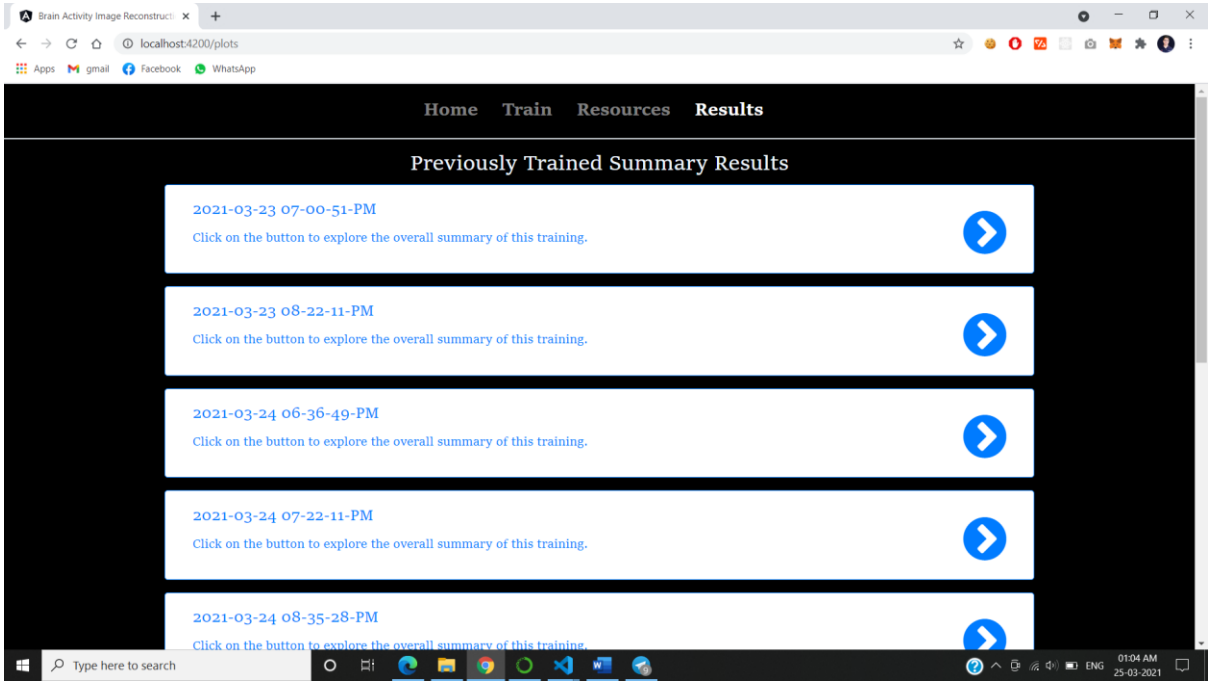

Brain Activity Image Reconstru
localhost:4200/trainer
Home
Train
Resources
Results
Try out New
No. Of Iterations
10
No. of Epochs
10
Select Classification Scheme
XG-Boost C
Let's Try
Not sure, about this ? See our previous results
Model Training has been Started
Estimated time : 60 sec
Server Status
Connected
Memory
CPU
80
Network
70
RAM Status : 68 %
CPU Status : 80 %
Network Status : 70 %
Explore More !
This project is an application of the cutting edge-technology present in the world. It is an application of Artificial Intelligence. It can recreate the scenarios that you've seen. By using the specialized modern Machine Learning algorithms. Our aim is to make this technology more efficient in such a way that it could be cost-effective and easily acessible. Keeping this in mind, we have attached our ML model using a modern WebStack (Angular + Python Flask) to make it easily accessible.


Brain Activity Image Reconstruct
localhost:4200/charts?file=2021_03_24-11_43_08_PM.json
Apps
gmail
Facebook
WhatsApp
Home
Train
Resources
Results
Showing results for 2021-03-24 11-43-08-PM
Mean Squared Error
TDNN
BCCA
DE-CNN
DGMM
Structured Similarity Index
0.112
Peak Signal Noise Ratio
Pearson Correlation Coefficient
Comparison of Metrics during Training
Type here to search
ENG
01:04 AM
25-03-2021


Brain Activity Image Reconstructi
localhost:4200/charts?file=2021_03_24-11_43_08_PM.json
Apps
gmail
Facebook
WhatsApp
Show Model Summary
Model: "TDNN"
Layer (type) Output Shape Param # Connected to
input_1 (InputLayer) (None, 1, 28, 28) 0
Firstlayer (Conv2D) (None, 1, 28, 1) 113 input_1[0][0]
Secondlayer (Conv2D) (None, 1, 14, 64) 320 Firstlayer[0][0]
Thirdlayer (Conv2D) (None, 1, 14, 64) 36928 Secondlayer[0][0]
Fourthlayer (Conv2D) (None, 1, 14, 64) 36928 Thirdlayer[0][0]
flatten_1 (Flatten) (None, 896) 0 Fourthlayer[0][0]
hiddenlayer (Dense) (None, 128) 114816 flatten_1[0][0]
Z_1 (Dense) (None, 6) 774 hiddenlayer[0][0]
Z_2 (Dense) (None, 6) 774 hiddenlayer[0][0]
lambda_1 (Lambda) (None, 6) 0 Z_1[0][0] Z_2[0][0]
dense_1 (Dense) (None, 128) 896 lambda_1[0][0]
dense_2 (Dense) (None, 12544) 1618176 dense_1[0][0]
reshape_1 (Reshape) (None, 14, 14, 64) 0 dense_2[0][0]
Type here to search
ENG
01:05 AM
25-03-2021


Brain Activity Image Reconstructi
localhost:4200/charts?file=2021_03_24-11_43_08_PM.json
Apps
gmail
Facebook
WhatsApp
Show Model Summary
No. of Iterations : 11 ---- No. of Epochs : 1
Original Dataset
Dataset Images of Visual Stimuli - Training
Dataset Images of Visual Stimuli - Testing
Reconstructed Images
Type here to search
ENG
01:05 AM
25-03-2021


Brain Activity Image Reconstructi
localhost:4200/charts?file=2021_03_24-11_43_08_PM.json
Apps
gmail
Facebook
WhatsApp
Classification Scheme Results
This menu shows the results of various image classification algorithms that are run on the original dataset and the reconstructed image dataset.
Evaluation Metrics for Support Vector Machine - Binary Classifier
Accuracy : 0.76
Precision - Label 6 : 0.87
Precision - Label 9 : 0.93
Recall score - Label 6 : 0.82
Recall score - Label 9 : 0.92
Evaluation Metrics for Random Forest - Decision Trees
Accuracy : 0.70
Precision - Label 6 : 0.86
Precision - Label 9 : 0.91
Recall score - Label 6 : 0.80
Recall score - Label 9 : 0.88
Evaluation Metrics for XGBoost Classifier - Gradient Boosting
Accuracy : 0.79
Precision - Label 6 : 0.85
Precision - Label 9 : 0.97
Recall score - Label 6 : 0.84
Recall score - Label 9 : 0.96
PANIMALAR ENGINEERING COLLEGE
JAISAKTHI EDUCATIONAL TRUST
Developed by
Piyush Chouhan M (211417104183)
Kedar R (211417104117)
Mohanavel R (211417104149)
© All rights reserved
Type here to search
ENG
01:06 AM
25-03-2021


Brain Activity Image Reconstructi
localhost:4200/plots
Apps
gmail
Facebook
WhatsApp
Home
Train
Resources
Results
Previously Trained Summary Results
2021-03-23 07-00-51-PM
Click on the button to explore the overall summary of this training.
2021-03-23 08-22-11-PM
Click on the button to explore the overall summary of this training.
2021-03-24 06-36-49-PM
Click on the button to explore the overall summary of this training.
2021-03-24 07-22-11-PM
Click on the button to explore the overall summary of this training.
2021-03-24 08-35-28-PM
Click on the button to explore the overall summary of this training.
Type here to search
ENG
01:04 AM
25-03-2021