PIXEL TO VOXEL AND BACK

Self supervision in natural image reconstruction from FMRI

Submitted BY:

Ashish Kempwad : 2019201091 [MTECH CSE,2019-2021][IIIT Hyderabad]

Swapnil Satpathy : 2019201078 [MTECH CSE,2019-2021][IIIT Hyderabad]

INTRODUCTION

The Problem Statement which we are dealing with in this project is the task natural Image reconstruction from FMRI which can have the below uses:

1) Study consciousness
2) Reconstruct Dreams

Although there are many supervised learning models available in this space but these models have a drawback in way that acquiring sufficient labelled pairs of **IMAGE:FMRI** is difficult.This lack in amounts of training data limits the generalization power of today's FMRI decoders(Input-> FMRI, Output -> Image).Purely supervised models are prone to poor generalization to new test data(FMRI of new images).

Hence in this project we are trying to achieve a self-supervised approach.The proposed model in the paper concatenates the encoder and decoder networks back-to-back which allows augmenting the

training with both types of unlabelled data i.e unlabelled FMRI and unlabelled images and this is what we are trying to achieve in this paper.

MODEL OVERVIEW

There are three phases in our model:

First Phase:

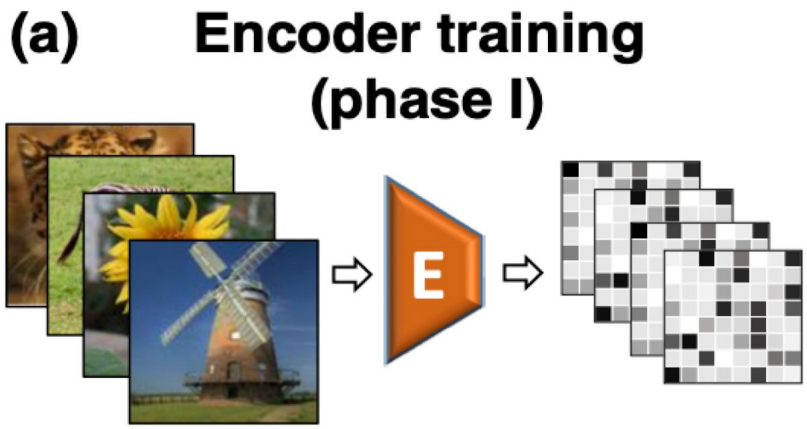
Supervised Learning of the Encoder E alone

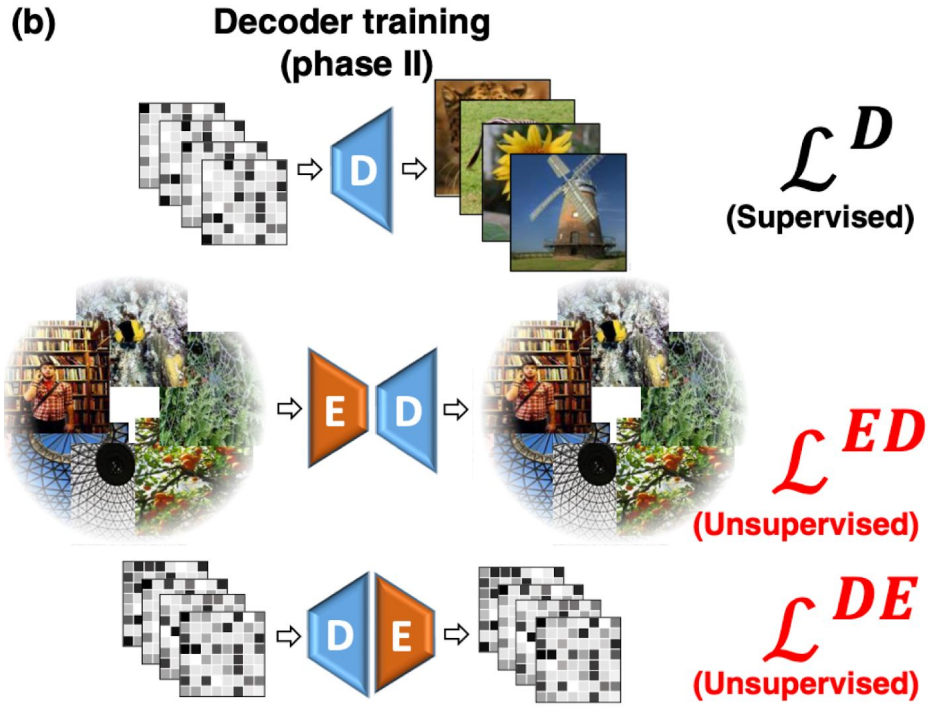
Second Phase:

Supervised Learning of the Decoder D alone

Third Phase:

Unsupervised Learning of the concatenated models i.e unsupervised learning with the Encoder-Decoder Concatenated model and unsupervised Learning with Decoder-Encoder Concatenated Model.





The above two figures show the phases used in this project. The first figure shows the supervised Training with the encoder alone

and the second figure shows the supervised training with the decoder, and unsupervised training with the concatenated encoder-decoder models.

PHASE 1

Encoder Training (SuperVised Training)

INPUT DATA:

For the supervised training of the encoder we will be using Imagenet Images rather to be specific we will be using Imagenet Images along with their corresponding FMRI data[voxel activations] prepared by the kamitani group. This model used here is called the GENERAL OBJECT DECODING MODEL [FMRI ON IMAGENET] prepared by kamitani Group.

For this intent of purpose, they have used 1200 imagenet images for training and 50 imagenet images for testing.

FMRI on IMAGENET (General Object Decoding)

Below are the details of how the FMRI(BOLD) data is collected by the Kamitani Group:-

FMRI data was recorded while subjects were viewing object images(image presentation experiment).

The image presentation experiment consisted of two distinct types of sessions : training image sessions and test image sessions. In the training image session, a total of 1200 images from 150 object categories(8 images from each category) were presented only once(24 runs).

In the test image session a total of 50 images from 50 object categories(1 image from each category) were presented 35 times each (35 runs).

Each scanning session consisted of functional(EPI) and anatomical(T2) data. The functional EPI images covered the entire brain(TR, 3000ms; TE,30ms; flip angle, 80 degrees; voxel size, 3 X 3 X 3 mm; FOV ,192 X 192 mm; number of slices ,50,slice gap , 0 mm)

The ROI (Region of Interest) we will be concentrating on in this project is ROI_VC

THE FMRI MAT FILE:

```
meta_keys = list(l[0] for l in meta[0][0][0][0])
meta_keys
```

```
['DataType',
 'Run',
 'Label',
 'VoxelData',
 'voxel_x',
 'voxel_y',
 'voxel_z',
 'VolInds',
 'ROI_V1',
 'ROI_V2',
 'ROI_V3',
 'ROI_V4',
 'ROI_LOC',
 'ROI_FFA',
 'ROI_PPA',
 'ROI_LVC',
 'ROI_HVC',
 'ROI_VC']
```

```
meta_desc = list(l[0] for l in meta[0][0][1][0])
meta_desc
```

```
['1 = Data type (1 = Training data; 2 = Perception test data; 3 = Imagery test data',
 '1 = Run number',
 '1 = Label (image ID)',
 '1 = Voxel data',
 'Voxel x coordinate',
 'Voxel y coordinate',
 'Voxel z coordinate',
 '1 = Volume index',
 '1 = ROI V1',
 '1 = ROI V2',
 '1 = ROI V3',
 '1 = ROI V4',
 '1 = ROI LOC',
 '1 = ROI FFA',
 '1 = ROI PPA',
 '1 = ROI LVC',
 '1 = ROI HVC',
 '1 = ROI VC']
```

DATA PREPROCESSING:

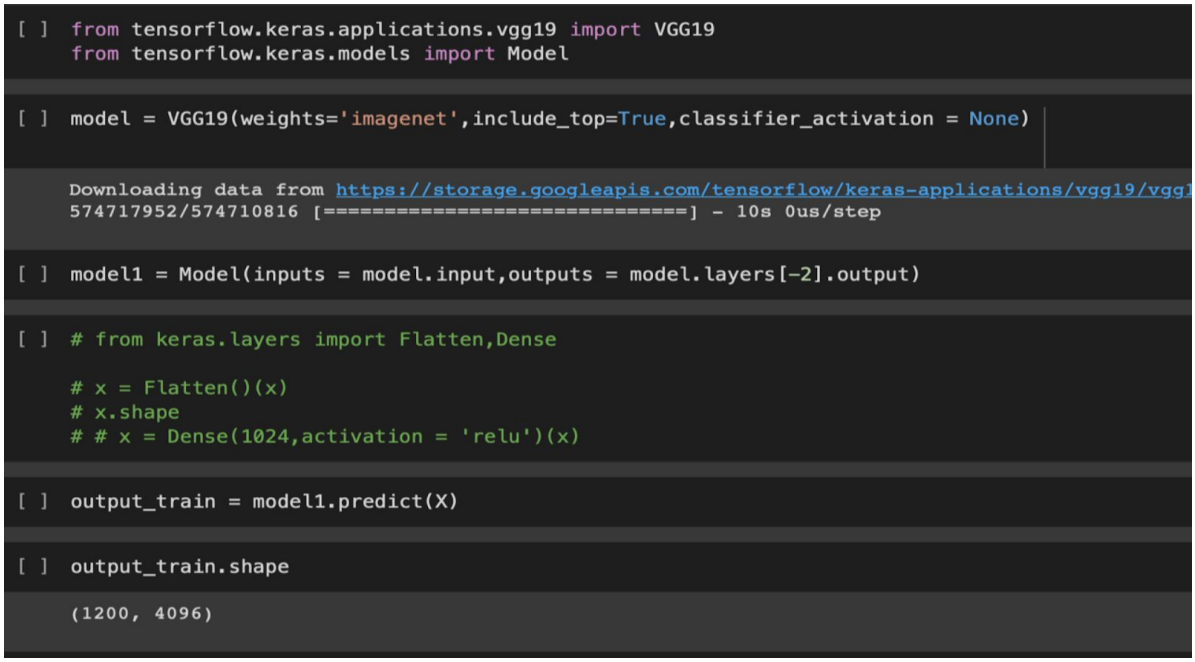
We are doing a central crop on images in case the row dimensions and column dimensions of the image are not the same. We are resizing the imagenet images which are originally 500 X 500 to 224 X 224 size.

Training the Encoder:

For the purpose of training the encoder, we have used many supervised learning models and compared between them. The inputs to these encoder models will be the images and output will be fmri voxel activations(Hence a Regression Problem).

Initially we used a VGG-19 model[The last layer removed] to extract features . We convert the input from 1200 X 224 X 224 X 3[As we have 1200 images and each image

has dimension of 224 X 224 X 3] space to 1200 X 4096 space. Below is the figure showing the VGG-19 model used to transform the input and extract features.



```
[ ]  from tensorflow.keras.applications.vgg19 import VGG19
     from tensorflow.keras.models import Model

[ ]  model = VGG19(weights='imagenet',include_top=True,classifier_activation = None)

     Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg1
     574717952/574710816 [==============================] - 10s 0us/step

[ ]  model1 = Model(inputs = model.input,outputs = model.layers[-2].output)

[ ]  # from keras.layers import Flatten,Dense

     # x = Flatten()(x)
     # x.shape
     # # x = Dense(1024,activation = 'relu')(x)

[ ]  output_train = model1.predict(X)

[ ]  output_train.shape

     (1200, 4096)
```

We used some supervised learning models:

Input (X) for training-> 1200 X 4096

Output (Y) for training -> 1200 X 4643[Here **4643 is the number of voxels**]

Input for testing -> 50 X 4096

Output for testing -> 50 X 4643

The loss we used is MSE loss(Mean Squared

Error Loss)

Below are the Regression Models used along with their MSE losses associated.

1) Ridge Regression Model -> 0.7648390111000501
2) Linear Regression Model -> 1.4276617705500056
3) Decision Tree Model -> 0.2913179000360818
4) MLP Regression Model -> 0.287392176304154

Out of all the Models we can see that the best performing model is the MLP Regressor Model with random state=1 and max_iterations=500 as hyperparameters

Below figures show the working of the above mentioned models:

Linear Regression:

```
[ ] from sklearn.linear_model import LinearRegression
```

```
[ ] reg = LinearRegression().fit(output_train,Y)
```

```
[ ] predictions = reg.predict(output_test)
```

```
[ ] predictions.shape
```

```
(50, 4643)
```

```
[ ] from sklearn.metrics import mean_squared_error
    print(np.sqrt(mean_squared_error(Y_test_avg,predictions)))
```

```
1.4276617705500056
```

Decision Tree Model:

```
[ ] from sklearn.tree import DecisionTreeRegressor
```

```
[ ] dtree_mse = {}
    for i in range(5,10,5):
      dtree = DecisionTreeRegressor(max_depth=i, min_samples_leaf=0.13, random_state=3)
      dtree.fit(output_train,Y)
      pred_tree= dtree.predict(output_test)
      dtree_mse[i] = (np.sqrt(mean_squared_error(pred_tree,Y_test_avg)))
```

```
[ ] max_key = max(dtree_mse, key=dtree_mse.get)

    print(max_key,dtree_mse[max_key])
```

```
5 0.2913179000360818
```

```
[ ] pred_tree= dtree.predict(output_test)
    print(np.sqrt(mean_squared_error(pred_tree,Y_test_avg)))
```

```
0.2913179000360818
```

Ridge Regression Model:

```
[ ]  from sklearn.linear_model import Ridge



     ridge = Ridge(alpha=1)
     ridge.fit(output_train,Y)
     predictions3 = ridge.predict(output_test)
     print(np.sqrt(mean_squared_error(predictions3,Y_test_avg)))
```

```
0.7648390111000501
```

```
[ ]  # max_key = max(mse, key=mse.get)

     # print(max_key,mse[max_key])
```

```
1 0.7648390111000501
```

MLP Regression Model:

```
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
encoder_regr = MLPRegressor(random_state=1,max_iter=500).fit(output_train,Y)
predictions4 = encoder_regr.predict(output_test)
print(np.sqrt(mean_squared_error(predictions4,Y_test_avg)))
```

```
0.287392176304154
```

PHASE 2

Decoder Training (SuperVised Training)

For the purpose of training the decoder, we have used many supervised learning models and compared between them. The inputs to these encoder models will be the fmri voxel activations and output will be image features (Hence a Regression Problem).

Input (X) for training-> 1200 X 4643[Here **4643 is the number of voxels**]

Output (Y) for training -> 1200 X 4096

Input for testing -> 50 X 4643

Output for testing -> 50 X 4096

The Loss Used in these models is a combination of MSE plus Cosine Similarity which is calculated as 0.2 * MSE + 0.8*Cosine Similarity.

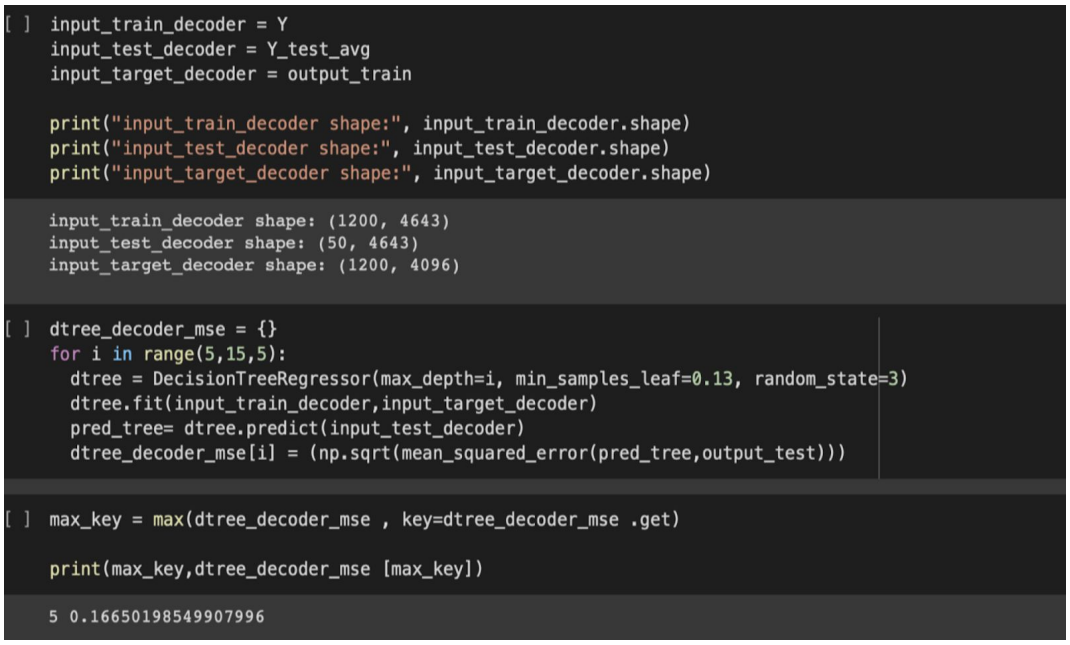
Below are the Decoder regression Models:

1) MLP Regression Model:- 0.4309026462890331
2) Ridge Regression model:- 0.15066884566512978

3) Decision Tree Model:- 0.16650

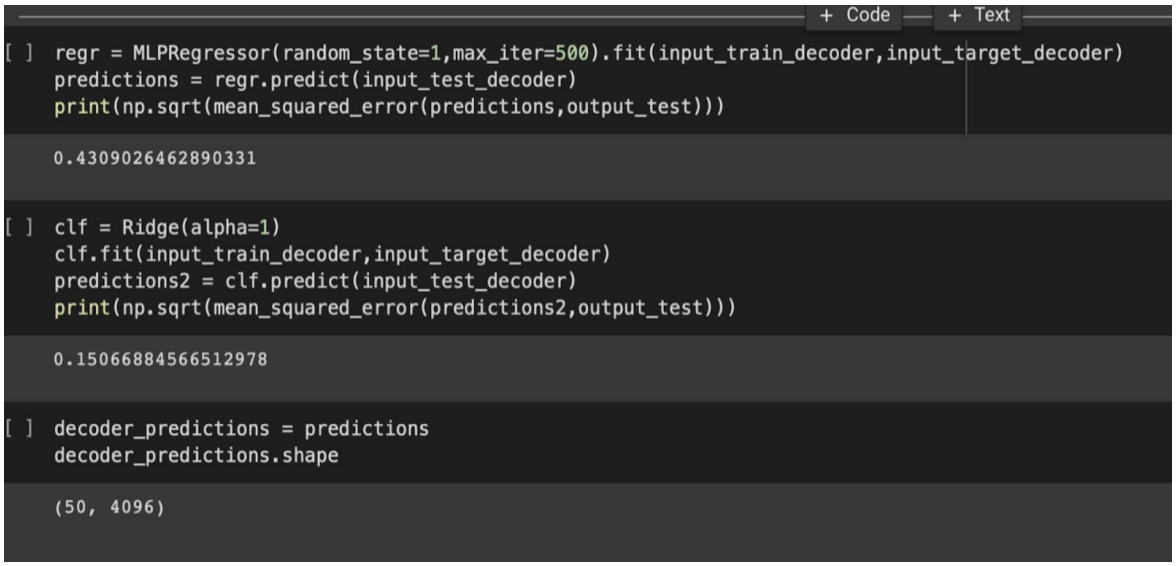
Below figures show the Regression Models used for the decoder task:

Decision Tree Model:



```
[ ] input_train_decoder = Y
    input_test_decoder = Y_test_avg
    input_target_decoder = output_train

    print("input_train_decoder shape:", input_train_decoder.shape)
    print("input_test_decoder shape:", input_test_decoder.shape)
    print("input_target_decoder shape:", input_target_decoder.shape)

    input_train_decoder shape: (1200, 4643)
    input_test_decoder shape: (50, 4643)
    input_target_decoder shape: (1200, 4096)

[ ] dtree_decoder_mse = {}
    for i in range(5,15,5):
      dtree = DecisionTreeRegressor(max_depth=i, min_samples_leaf=0.13, random_state=3)
      dtree.fit(input_train_decoder,input_target_decoder)
      pred_tree= dtree.predict(input_test_decoder)
      dtree_decoder_mse[i] = (np.sqrt(mean_squared_error(pred_tree,output_test)))

[ ] max_key = max(dtree_decoder_mse , key=dtree_decoder_mse .get)

    print(max_key,dtree_decoder_mse [max_key])

    5 0.16650198549907996
```

MLP Regression and Ridge Regression Model:



We can see that the best performing model in this decoder task is the Ridge Regression Model with loss of 0.1506 and hyperparameter alpha =1.

PHASE 3

Unsupervised Training of Encoder-Decoder and Decoder-Encoder Concatenated.

a) Encoder-Decoder concatenated:

We concatenate the MLP Regressor Model as Encoder and Ridge Regression Model as Decoder together. The input to this model

will be 1200 X 4096 image vectors with no corresponding FMRI data. We will use the previously trained MLP Regressor Model and Ridge Regressor Models for this task.

Below figure shows the working of this particular concatenated model:

```
[ ] from sklearn.neural_network import MLPRegressor
    middle_layer = encoder_regr.predict(output_train)
```

```
[ ] middle_layer.shape

    (1200, 4643)
```

```
[ ] output_encoder_decoder = decoder_reg.predict(middle_layer)
```

```
[ ] output_encoder_decoder.shape

    (1200, 4096)
```

```
[ ] print(np.sqrt(mean_squared_error(output_encoder_decoder,output_train)))

    0.17795101012573947
```

We give our image vectors as input to the pretrained MLP Regressor Encoder Model for which we get a middle output of 1200 X 4643 which we again give it to the pretrained Ridge Regressor Decoder Model to get the image output back.

The loss we are getting in this model is

0.177951.

b) Decoder-Encoder Task:

Experimentation with various models concatenated:

1) **FMRI -> RIDGE REGRESSOR -> Latent Image -> MLP REGRESSOR**
We are getting an MSE of 0.9828046138457656 with this model.

2) **FMRI -> MLP REGRESSOR -> Latent Image -> MLP REGRESSOR**

We are getting the same MSE of 0.9828046138457656 with this model.

3) **FMRI -> RIDGE REGRESSOR -> Latent Image -> RIDGE REGRESSOR**

We are getting the MSE of 0.4166018575484804 with this model.

Hence we will be using Ridge-Ridge Concatenation for this purpose

Below figure shows the working of the Decoder-Encoder Concatenated Model.

▾ Decoder-Encoder

```
[ ] middle_representation=decoder_reg.predict(Y)

[ ] middle_representation.shape

    (1200, 4096)

[ ] output_decoder_encoder=encoder_regr.predict(middle_representation)

[ ] output_decoder_encoder.shape

    (1200, 4643)

[ ] print(np.sqrt(mean_squared_error(output_decoder_encoder,Y)))

    0.9828046138457656
```

With MLP-MLP Concatenation

```
[ ] temp=regr.predict(Y)
```

```
[ ] temp2=encoder_regr.predict(temp)
```

```
[ ] print(np.sqrt(mean_squared_error(temp2,Y)))
```

```
0.9828046138457656
```

With Ridge-Ridge Regressor

```
[ ] temp=decoder_reg.predict(Y)
    temp2=ridge.predict(temp)
    print(np.sqrt(mean_squared_error(temp2,Y)))
```

```
0.4166018575484804
```

We are getting a loss of 0.416601.

IMPROVEMENTS OVER THE LOSS

As per the original paper they have used complex Neural Network models with many famous architecture components due to which lead them to a loss of 0.34567 in Decoder-Encoder Model and 0.1546 with

Encoder-Decoder Model. We tried using the model suggested by the original paper,but we were unable to understand many of the components in the network, hence we simplified our model and experimented with simple Ridge Regressor and MLP Regressor Models. Apart from that we can experiment with different datasets to see if our models work fine with these different datasets.

Link to Our Codes (Written on Google Colab):

https://colab.research.google.com/drive/1IbeS88V-ydsSUKMPhBbzD2JrtShpMGeI?usp=sharing

https://colab.research.google.com/drive/1xlmTp-QOrv6azDbeeUQK9E-b5MUocW5Y?usp=sharing

REFERENCES

1) https://arxiv.org/abs/1907.02431
2) https://github.com/WeizmannVision/ssfmri2im

3) http://www.wisdom.weizmann.ac.il/~vision/ssfmri2im/
4) https://www.youtube.com/watch?v=tQyMqRqHwao

Regarding the HCP data, we are unable to get the proper scripts to go ahead and hence download the data. We came in touch with Shanu and Shubham's team and got to know that they have already downloaded and submitted it for all.