

Brain Encoding Models for Visual Cortex

INTRODUCTION

Understanding how the human brain processes visual stimuli is a key challenge in **cognitive neuroscience**. Recent advancements in **deep neural networks (DNNs)** have enabled researchers to model and predict neural responses to natural images using encoding models. This assignment aims to evaluate how well different CNN architectures—**ResNet-50, EfficientNet, and AlexNet**—can predict brain responses to visual stimuli across different cortical regions.

To achieve this, we will use **fMRI data** from the **Algonauts 2023 dataset**, where subjects viewed various natural images while their brain activity was recorded. The task involves extracting **feature representations** from deep neural networks and training **encoding models** (Linear or Ridge Regression) to predict **brain responses** in different **Regions of Interest (ROIs)**.

Regions of Interest (ROIs)

In this study, we focus on the following **three ROIs**, each belonging to a different ROI class:

1. **V3v (Ventral Visual Area 3) – Class: prf-visualrois**

Part of the **early visual cortex**, responsible for processing spatial and motion-related information.

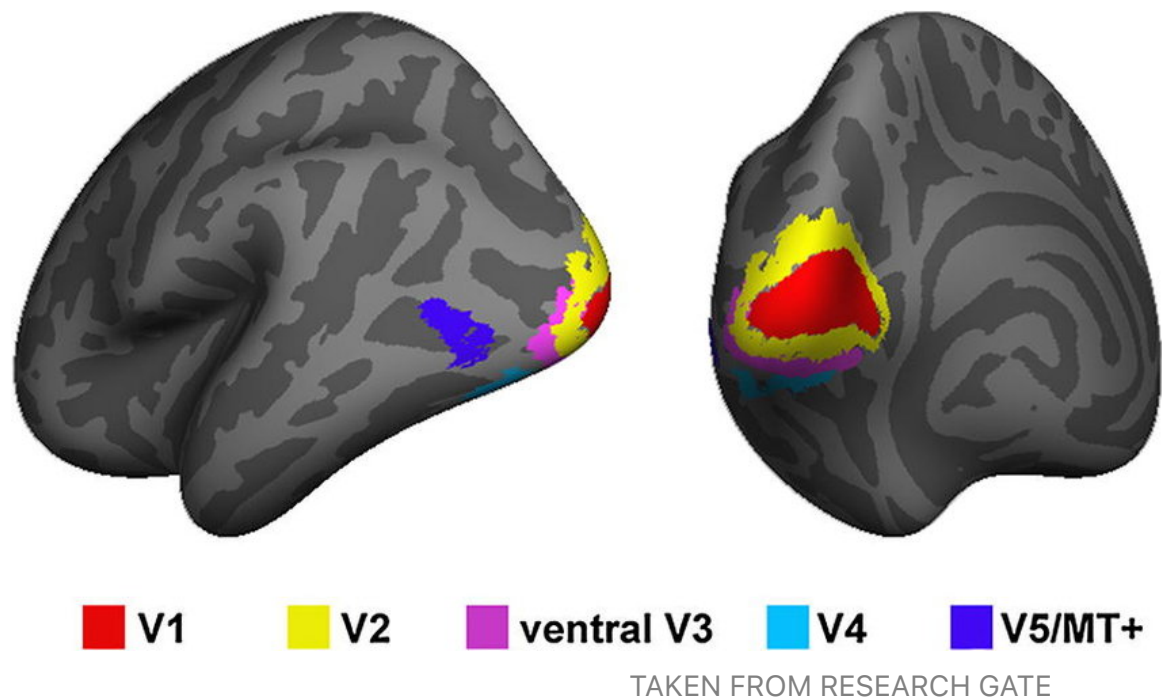
2. **FBA-2 (Fusiform Body Area) – Class: floc-bodies**

A **higher-order visual region** involved in recognizing **body parts and**

biological motion.

3. Early Anatomical Stream – Class: streams

A **broader set of early visual regions** involved in low-level visual processing.



WORKFLOW

The study follows an end-to-end pipeline consisting of the following key steps:

Step 1: Data Preparation

Mounts **Google Drive** (if using Colab) to access dataset files. Imports required Python libraries for **data processing, visualization, and model training**. Loads essential dataset components: **Training images** (visual stimuli shown to subjects), **ROI masks** (brain region mappings for fMRI analysis), **fMRI responses** (neural activity recorded during stimulus

presentation).

```
import os
import numpy as np
from pathlib import Path
from PIL import Image
from tqdm import tqdm
import matplotlib
from matplotlib import pyplot as plt
from nilearn import datasets
from nilearn import plotting
import torch
from torch.utils.data import DataLoader, Dataset
from torchvision.models.feature_extraction import create_feature_extractor, get_graph_node_names
from torchvision import transforms
from sklearn.decomposition import IncrementalPCA
from sklearn.linear_model import LinearRegression
from scipy.stats import pearsonr as corr
import matplotlib.pyplot as plt
from scipy.stats import pearsonr

[2]

from google.colab import drive
drive.mount('/content/drive/', force_remount=True)
data_dir='/content/drive/MyDrive/algonauts_2023_tutorial_data'
device = 'cuda'
device = torch.device(device)
## Enter the Subject Here
subj = 4

class argObj:
    def __init__(self, data_dir, subj):
        self.subj = format(subj, '02')
        self.data_dir = os.path.join(data_dir, 'subj'+self.subj)

args = argObj(data_dir,subj)

... Mounted at /content/drive/
```

Code Used For Importing the Required Libraries and the dataset in google drive

Step 2: Load & Visualize fMRI Training Data

Loads **left hemisphere (LH)** and **right hemisphere (RH)** fMRI data for all stimulus images. Each row represents a **stimulus image**, and each column represents a **brain vertex's response** to that image. Visualizes the **brain surface map** using the fsaverage template.

```

fmri_dir = os.path.join(args.data_dir, 'training_split', 'training_fmri')
lh_fmri = np.load(os.path.join(fmri_dir, 'lh_training_fmri.npy'))
rh_fmri = np.load(os.path.join(fmri_dir, 'rh_training_fmri.npy'))

print('LH training fMRI data shape:')
print(lh_fmri.shape)
print('(Training stimulus images x LH vertices)')

print('\nRH training fMRI data shape:')
print(rh_fmri.shape)
print('(Training stimulus images x RH vertices)')

```

[8]

```

... LH training fMRI data shape:
(8779, 19004)
(Training stimulus images x LH vertices)

RH training fMRI data shape:
(8779, 20544)
(Training stimulus images x RH vertices)

```

```

hemisphere = 'left' # ['left', 'right']

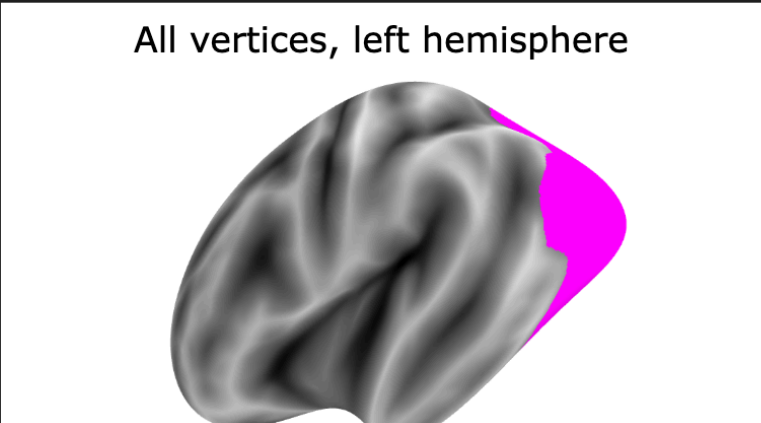
# Load the brain surface map of all vertices: file lh.all-vertices_fsaverage_space.npy
roi_dir = os.path.join(args.data_dir, 'roi_masks',
                        hemisphere[0]+'h.all-vertices_fsaverage_space.npy')
fsaverage_all_vertices = np.load(roi_dir)
print("Length total Vertices ", len(fsaverage_all_vertices))

# Create the interactive brain surface map
fsaverage = datasets.fetch_surf_fsaverage('fsaverage')
view = plotting.view_surf(
    surf_mesh=fsaverage['infl_'+hemisphere],
    surf_map=fsaverage_all_vertices,
    bg_map=fsaverage['sulc_'+hemisphere],
    threshold=1e-14,
    cmap='cool',
    colorbar=False,
    title='All vertices, '+hemisphere+' hemisphere'
)
view

```

... Length total Vertices , 163842

...



Here is the code to visualise the all vertices in the left hemisphere and the loading of fmri data

Step 3: Select Region of Interest (ROI)

Selects a **specific brain region** (e.g., V3v, FBA-2, Early) from fMRI data. Maps the selected ROI's indices onto the **brain surface template**. Visualizes the **selected ROI** on a 3D interactive brain model.

```
hemisphere = 'left' # ['left', 'right'] {allow-input: true}
roi = "V3v" # [ "V3v", "FBA-2", "early"] {allow-input: true}
roi_class='prf-visualrois'

# Load the ROI brain surface maps
roi_class_dir = os.path.join(args.data_dir, 'roi_masks',
                             hemisphere[0]+'h.'+roi_class+'_fsaverage_space.npy')
roi_map_dir = os.path.join(args.data_dir, 'roi_masks',
                             'mapping_'+roi_class+'.npy')
fsaverage_roi_class = np.load(roi_class_dir)
roi_map = np.load(roi_map_dir, allow_pickle=True).item()

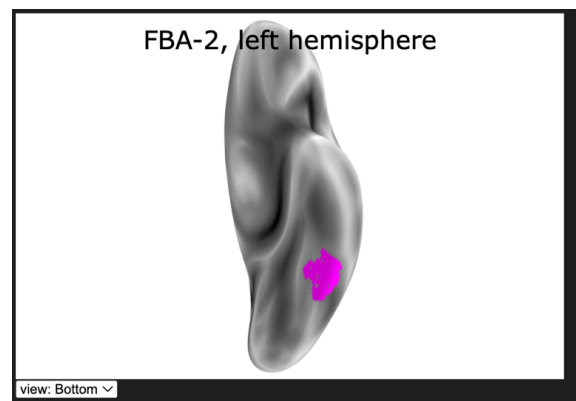
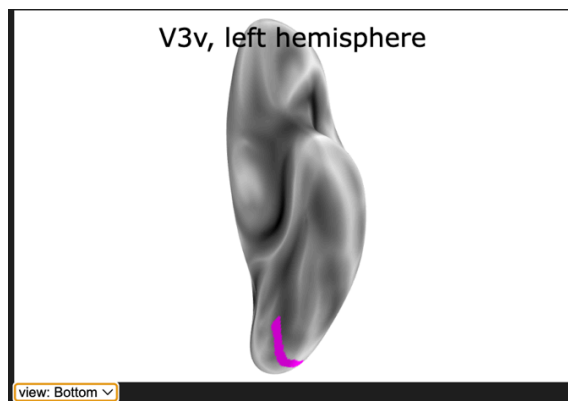
# Select the vertices corresponding to the ROI of interest
roi_mapping = list(roi_map.keys())[list(roi_map.values()).index(roi)]
print("Value of ROI Mapping using mapping file:", roi_mapping)

# Convert to binary mask (1 where the value matches, else 0)
fsaverage_roi = np.asarray(fsaverage_roi_class == roi_mapping, dtype=int)

# Count the number of 1s (vertices belonging to ROI)
roi_vertex_count = np.sum(fsaverage_roi)

print(f"Total Vertices belonging to ROI {roi_class} are {roi_vertex_count}")
# Create the interactive brain surface map
fsaverage = datasets.fetch_surf_fsaverage('fsaverage')
view = plotting.view_surf(
    surf_mesh=fsaverage['infl_'+hemisphere],
    surf_map=fsaverage_roi,
    bg_map=fsaverage['sulc_'+hemisphere],
    threshold=1e-14,
    cmap='cool',
    colorbar=False,
    title=roi+' '+hemisphere+' hemisphere'
)
view
```

Code for the **visualization of a specific ROI**. Using the **mapping file**, we obtain the **ID** to which the given ROI maps within its **ROI class** (e.g., V3v maps to ID 5). This allows us to extract the corresponding **voxels** from the **fsaverage file** for that ROI.



Step 4: Load & Preprocess Stimulus Images

Loads the **natural images** shown to subjects during fMRI scans. Splits images into **Training (90%)**, **Validation (10%)**, and **Test** sets. Applies preprocessing: **Convert to Tensor** and **Normalize** (standard preprocessing for deep learning).

```

Stimulus images

train_img_dir = os.path.join(args.data_dir, 'training_split', 'training_images')
test_img_dir = os.path.join(args.data_dir, 'test_split', 'test_images')

# Create lists with all training and test image file names, sorted
train_img_list = os.listdir(train_img_dir)
train_img_list.sort()
test_img_list = os.listdir(test_img_dir)
test_img_list.sort()
print('Training images: ' + str(len(train_img_list)))
print('Test images: ' + str(len(test_img_list)))

[13]
... Training images: 8779
    Test images: 395

▶
train_img_file = train_img_list[0]
print('Training image file name: ' + train_img_file)
print('73k NSD images ID: ' + train_img_file[-9:-4])

[14]
... Training image file name: train-0001_nsd-00012.png
    73k NSD images ID: 00012

```

Load the stimulus images

STEP 5 : Visualizing the fMRI Response in the Given ROI

Maps the **fMRI response data** onto the selected **Region of Interest (ROI)** in the brain. Helps in understanding **how strongly different voxels within the ROI respond** to visual stimuli.

```
# Select the vertices corresponding to the ROI of interest
roi_mapping = list(roi_map.keys())[list(roi_map.values()).index(roi)]
print(f" Selected ROI '{roi}' maps to integer value: {roi_mapping}")

challenge_roi = np.asarray(challenge_roi_class == roi_mapping, dtype=int)
fsaverage_roi = np.asarray(fsaverage_roi_class == roi_mapping, dtype=int)

# Debug print: Number of vertices selected in each space
print(f" Total vertices in Challenge Space: {challenge_roi_class.shape[0]}")
print(f" Selected vertices in Challenge Space (ROI): {np.sum(challenge_roi)}")
print(f" Selected vertices in Fsaverage Space (ROI): {np.sum(fsaverage_roi)}")

# Debug print: Show first few selected vertex indices
print(f" First 10 Challenge ROI Vertex Indices: {np.where(challenge_roi)[0][:10]}")
print(f" First 10 Fsaverage ROI Vertex Indices: {np.where(fsaverage_roi)[0][:10]}")

# Initialize fMRI response map
fsaverage_response = np.zeros(len(fsaverage_roi))
print(f" Initialized fsaverage_response with shape: {fsaverage_response.shape}")

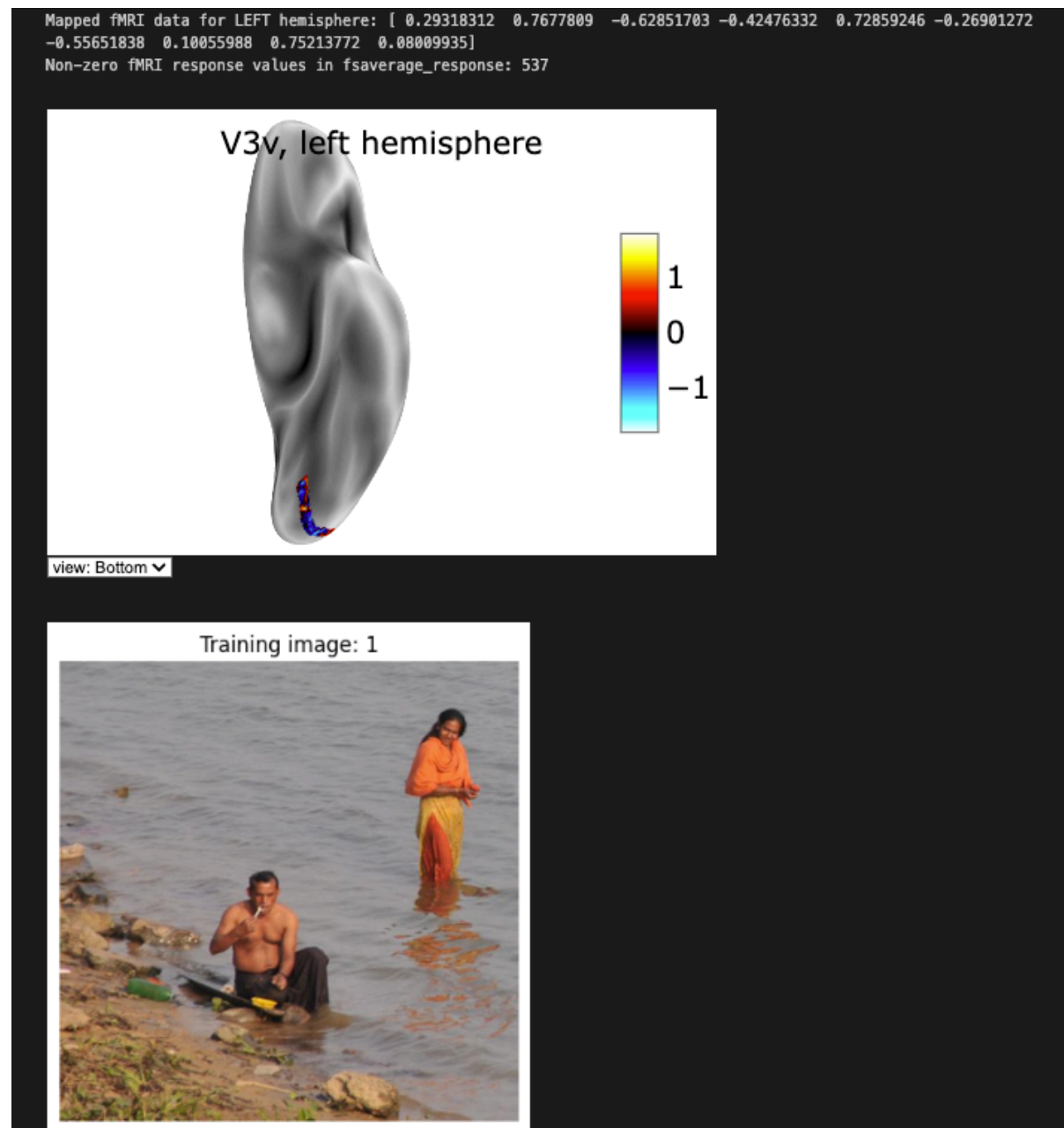
# Map fMRI data onto the brain surface
if hemisphere == 'left':
    fsaverage_response[np.where(fsaverage_roi)[0]] = lh_fmri[img, np.where(challenge_roi)[0]]
    print(f" Mapped fMRI data for LEFT hemisphere: {fsaverage_response[np.where(fsaverage_roi)[0]][:10]}")
elif hemisphere == 'right':
    fsaverage_response[np.where(fsaverage_roi)[0]] = rh_fmri[img, np.where(challenge_roi)[0]]
    print(f" Mapped fMRI data for RIGHT hemisphere: {fsaverage_response[np.where(fsaverage_roi)[0]][:10]}")

# Final check
print(f" Non-zero fMRI response values in fsaverage_response: {np.count_nonzero(fsaverage_response)}")

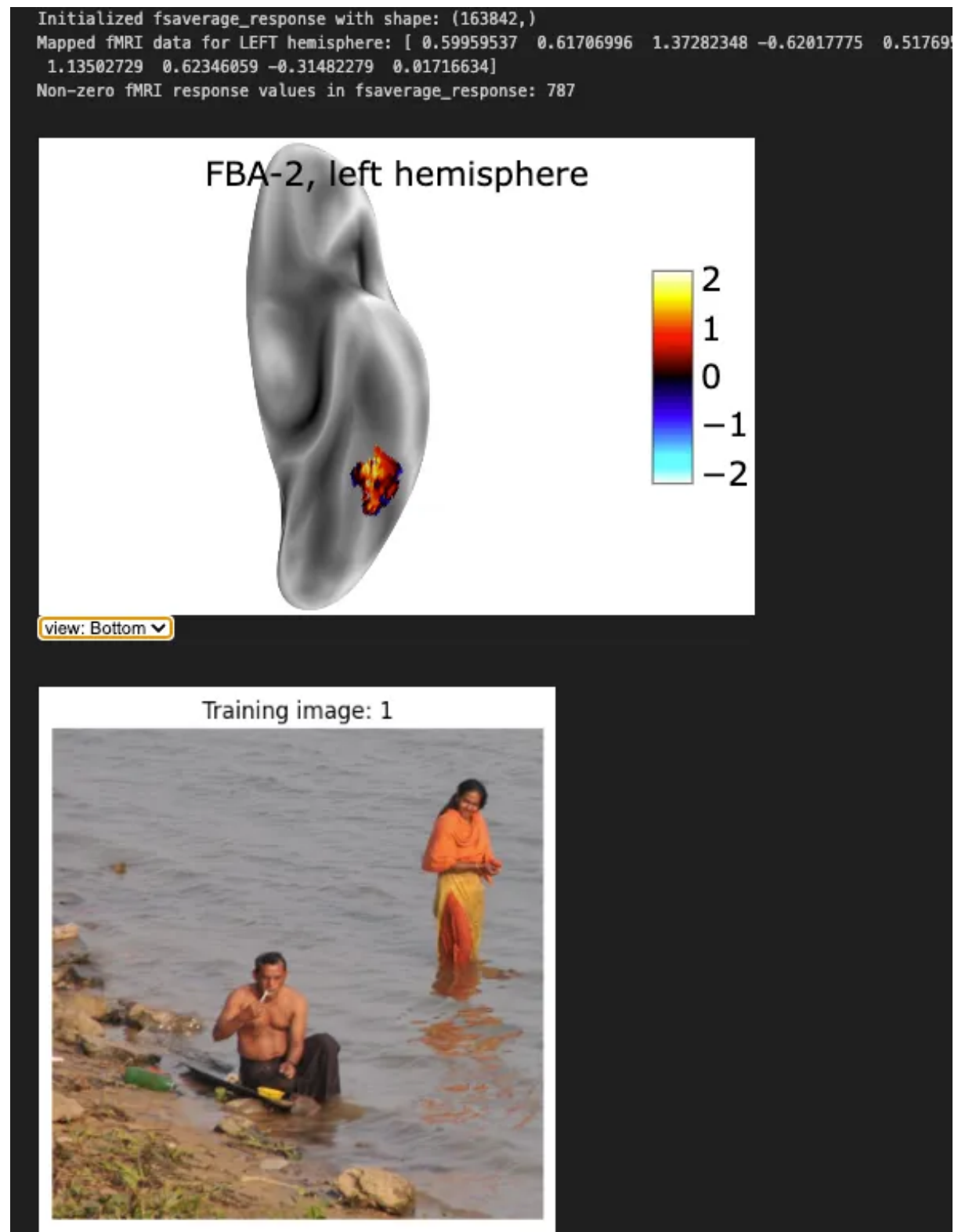
# Create the interactive brain surface map
fsaverage = datasets.fetch_surf_fsaverage('fsaverage')
view = plotting.view_surf(
    surf_mesh=fsaverage['infl_'+hemisphere],
    surf_map=fsaverage_response,
    bg_map=fsaverage['sulc_'+hemisphere],
    threshold=1e-14,
    cmap='cold_hot',
    colorbar=True,
    title=roi+' '+hemisphere+' hemisphere'
)
view
```

```
... Loaded challenge ROI class shape: (19004,)
Loaded fsaverage ROI class shape: (163842,)
ROI mapping dictionary keys (first 10): [0, 1, 2, 3, 4, 5, 6, 7]
ROI mapping dictionary values (first 10): ['Unknown', 'V1v', 'V1d', 'V2v', 'V2d', 'V3v', 'V3d', 'hV4']
Selected ROI 'V3v' maps to integer value: 5
Total vertices in Challenge Space: 19004
Selected vertices in Challenge Space (ROI): 537
Selected vertices in Fsaverage Space (ROI): 537
```

Visualising FMRI responses corresponding to Image 1 for ROi -v3v



Visualization Output for Training Image 1 Showing the Activated V3v in the fMRI Data



Visualization Output for Training Image 1 Showing the Activated FBA-2 in the fMRI Data

STEP 6: Preparing the DataLoaders

Once the **stimulus images** have been loaded, the next step is to **divide the**

dataset into **training, validation, and test sets**. Along with this, we extract the corresponding **fMRI signals** for each partition. At this stage, we have: **Input DataLoaders** (for feeding images into model), **Expected fMRI responses** (for training the encoding models).

```

rand_seed = 42
np.random.seed(rand_seed)

# Calculate how many stimulus images correspond to 90% of the training data
num_train = int(np.round(len(train_img_list) / 100 * 90))

# Shuffle all training stimulus images
idxs = np.arange(len(train_img_list))
np.random.shuffle(idxs)
# Assign 90% of the shuffled stimulus images to the training partition,
# and 10% to the test partition
idxs_train, idxs_val = idxs[:num_train], idxs[num_train:]
# No need to shuffle or split the test stimulus images
idxs_test = np.arange(len(test_img_list))

print('Training stimulus images: ' + format(len(idxs_train)))
print('\nValidation stimulus images: ' + format(len(idxs_val)))
print('\nTest stimulus images: ' + format(len(idxs_test)))

Training stimulus images: 7901

Validation stimulus images: 878

Test stimulus images: 395

transform = transforms.Compose([
    transforms.Resize((224,224)), # resize the images to 224x224 pixels
    transforms.ToTensor(), # convert the images to a PyTorch tensor
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # normalize the images color channels
])

class ImageDataset(Dataset):
    def __init__(self, imgs_paths, idxs, transform):
        self.imgs_paths = np.array(imgs_paths)[idxs]
        self.transform = transform

    def __len__(self):
        return len(self.imgs_paths)

    def __getitem__(self, idx):
        # Load the image
        img_path = self.imgs_paths[idx]
        img = Image.open(img_path).convert('RGB')
        # Preprocess the image and send it to the chosen device ('cpu' or 'cuda')
        if self.transform:
            img = self.transform(img).to(device)
        return img

```

Code to partition the dataset into train, validation and test and creating the tensor dataset.

```

lh_fmri_train = lh_fmri[idxs_train]
lh_fmri_val = lh_fmri[idxs_val]
rh_fmri_train = rh_fmri[idxs_train]
rh_fmri_val = rh_fmri[idxs_val]
lh_fmri_test = lh_fmri[idxs_test]
rh_fmri_test = rh_fmri[idxs_test]

[22]

print('Training fMRI data shape:')
print(lh_fmri_train.shape)
print('(Training stimulus images x LH vertices)')

print('\nValidation fMRI data shape:')
print(lh_fmri_val.shape)
print('(Validation stimulus images x LH vertices)')

print('\nTest fMRI data shape:')
print(lh_fmri_test.shape)
print('(Test stimulus images x LH vertices)')

[23]

... Training fMRI data shape:
(7901, 19004)
(Training stimulus images x LH vertices)

Validation fMRI data shape:
(878, 19004)
(Validation stimulus images x LH vertices)

Test fMRI data shape:
(395, 19004)
(Test stimulus images x LH vertices)

```

Extract FMRI data corresponding to them.

Step 7: Extract Features Using CNN Architecture

Loads **pretrained models** and extracts features from the **last layer** of **ResNet-50, EfficientNet, and AlexNet**. Each model is loaded individually, and features are extracted from their respective final layers.

```

import torch
from torchvision.models import efficientnet_b0
from torchvision.models import resnet50
from torchvision.models.feature_extraction import create_feature_extractor, get_graph_node_names

# Load the pre-trained model- AlexNet
model = torch.hub.load('pytorch/vision:v0.10.0', 'alexnet')
model.to(device) # send the model to the chosen device ('cpu' or 'cuda')
model.eval() # set the model to evaluation mode, since you are not training it
model

## Load Pretrained EfficientNet Model
# model = efficientnet_b0(pretrained=True) # Choose from efficientnet_b0 to efficientnet_b7
# model.to(device)
# model.eval()
# model

## Load Pretrained ResNet Model
# model = resnet50(pretrained=True)
# model.to(device)
# model.eval()
# model

Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /root/.cache/torch/hub/v0.10.0.zip

AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
  )
)

```

Code for loading the pretrained models- Resnet,Alexnet,EfficientNet

Step 8: Apply PCA for Dimensionality Reduction

Applies **Principal Component Analysis (PCA)** to reduce **high-dimensional feature vectors** to **100 principal components**. This helps in **reducing computational cost** while retaining **essential information**.

First, features are extracted from the model using a **feature extractor**, as shown in the code. These extracted features are then passed to **Incremental PCA**, which is trained on the data to perform dimensionality reduction efficiently.

```

def fit_pca(feature_extractor, dataloader, n_components=100, batch_size=32):
    """Fit PCA on extracted features in mini-batches."""

    # Move model to GPU if available
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    feature_extractor.to(device)

    pca = IncrementalPCA(n_components=n_components, batch_size=batch_size)

    for d in tqdm(dataloader, total=len(dataloader), desc="Extracting features for PCA"):
        d = d.to(device) # Move data to GPU
        with torch.no_grad(): # Disable gradients for speed
            ft = feature_extractor(d)
            ft = torch.hstack([torch.flatten(l, start_dim=1) for l in ft.values()])
            pca.partial_fit(ft.cpu().numpy()) # Fit PCA in batches

    return pca

pca = fit_pca(feature_extractor, train_imgs_dataloader)

def extract_features(feature_extractor, dataloader, pca):
    """Extract and transform features using PCA."""

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    feature_extractor.to(device)

    features = []
    for d in tqdm(dataloader, total=len(dataloader), desc="Extracting & Transforming Features"):
        d = d.to(device)
        with torch.no_grad():
            ft = feature_extractor(d)
            ft = torch.hstack([torch.flatten(l, start_dim=1) for l in ft.values()])
            ft = pca.transform(ft.cpu().numpy()) # Apply PCA
            features.append(ft)

    return np.vstack(features)

# Run feature extraction with PCA
features_train = extract_features(feature_extractor, train_imgs_dataloader, pca)
features_val = extract_features(feature_extractor, val_imgs_dataloader, pca)
features_test = extract_features(feature_extractor, test_imgs_dataloader, pca)

# Print results
print('\nTraining images features:', features_train.shape)
print('(Training stimulus images & PCA features)')

print('\nValidation images features:', features_val.shape)
print('(Validation stimulus images & PCA features)')

print('\nTest images features:', features_test.shape)
print('(Test stimulus images & PCA features)')

```

Code to get dimensionality reduced features using the CNN model feature extractor and PCA

```
Extracting features for PCA: 100%|██████████| 16/16 [04:18<00:00, 16.18s/it]
Extracting & Transforming Features: 100%|██████████| 16/16 [02:06<00:00, 7.90s/it]
Extracting & Transforming Features: 100%|██████████| 2/2 [00:13<00:00, 6.98s/it]
Extracting & Transforming Features: 100%|██████████| 1/1 [02:49<00:00, 169.56s/it]

Training images features: (7901, 100)
(Training stimulus images × PCA features)

Validation images features: (878, 100)
(Validation stimulus images × PCA features)

Test images features: (395, 100)
(Test stimulus images × PCA features)
```

Final features with reduced dimensionality

Step 9: Train Ridge Regression Model

Randomly selects **10 vertices** and uses **Linear Regression** to learn a mapping from **PCA-transformed image features** to **fMRI responses (ROI vertices)**.

```

# Set number of voxels to randomly select
num_voxels = 10

# Get all voxel indices in the ROI
roi_voxel_indices = np.where(challenge_roi)[0]

# Randomly select `num_voxels` from the available ROI vertices
if len(roi_voxel_indices) >= num_voxels:
    selected_voxels = np.random.choice(roi_voxel_indices, num_voxels, replace=False)
else:
    selected_voxels = roi_voxel_indices # If fewer than 10 exist, take all

print(f" Randomly selected {len(selected_voxels)} voxels from ROI: {roi}")

# Extract fMRI responses for selected voxels
lh_roi_train_subset = lh_fmri_train[:, selected_voxels]
lh_roi_val_subset = lh_fmri_val[:, selected_voxels]
lh_roi_test_subset = lh_fmri_test[:, selected_voxels]

print(f" Selected Voxels Shape (Train): {lh_roi_train_subset.shape}")
print(f" Selected Voxels Shape (Validation): {lh_roi_val_subset.shape}")
print(f" Selected Voxels Shape (Test): {lh_roi_test_subset.shape}")

# Train a single Linear Regression model
reg = LinearRegression().fit(features_train, lh_roi_train_subset)

# Predict fMRI responses
lh_fmri_val_pred = reg.predict(features_val)
lh_fmri_test_pred = reg.predict(features_test)

print(f" Model trained on {num_voxels} randomly selected voxels in ROI: {roi}")
print(f" Validation Predictions Shape: {lh_fmri_val_pred.shape}")
print(f" Test Predictions Shape: {lh_fmri_test_pred.shape}")

```

[35]

```

... Randomly selected 10 voxels from ROI: V3v
Selected Voxels Shape (Train): (7901, 10)
Selected Voxels Shape (Validation): (878, 10)
Selected Voxels Shape (Test): (395, 10)
Model trained on 10 randomly selected voxels in ROI: V3v
Validation Predictions Shape: (878, 10)
Test Predictions Shape: (395, 10)

```

Randomly selects the 10 voxels and fits the linear regression model and does the prediction on the test and validation set

Step 10: Visualize Encoding Accuracy

Plots a histogram of **Pearson Correlation scores** for all vertices in the ROI. Highlights the **mean encoding accuracy** across all voxels.

```

# Function to compute correlation
def compute_correlation(pred, actual):
    """Computes Pearson correlation for each vertex."""
    correlation = np.zeros(pred.shape[1]) # Initialize correlation array
    for v in range(pred.shape[1]): # Iterate over all ROI vertices
        correlation[v] = pearsonr(pred[:, v], actual[:, v])[0] # Compute Pearson correlation
    return correlation

# Compute correlation for validation set
lh_correlation = compute_correlation(lh_fmri_val_pred, lh_roi_val_subset)

# Compute summary statistics
mean_corr = np.mean(lh_correlation)
max_corr = np.max(lh_correlation)

# Print performance evaluation
print(f"\n Mean Pearson Correlation for ROI {roi}: {mean_corr:.4f}")
print(f" Max Pearson Correlation for ROI {roi}: {max_corr:.4f}")

# ==== PLOT DISTRIBUTION OF CORRELATIONS ====
plt.figure(figsize=(5, 5))
plt.hist(lh_correlation, bins=50, color='skyblue', edgecolor='black')
plt.xlabel("Pearson Correlation")
plt.ylabel("Number of Vertices")
plt.title(f"Encoding Accuracy Distribution for ROI: {roi}")
plt.axvline(mean_corr, color='red', linestyle='dashed', label=f"Mean: {mean_corr:.4f}")
plt.legend()
plt.show()

```

[38]

...

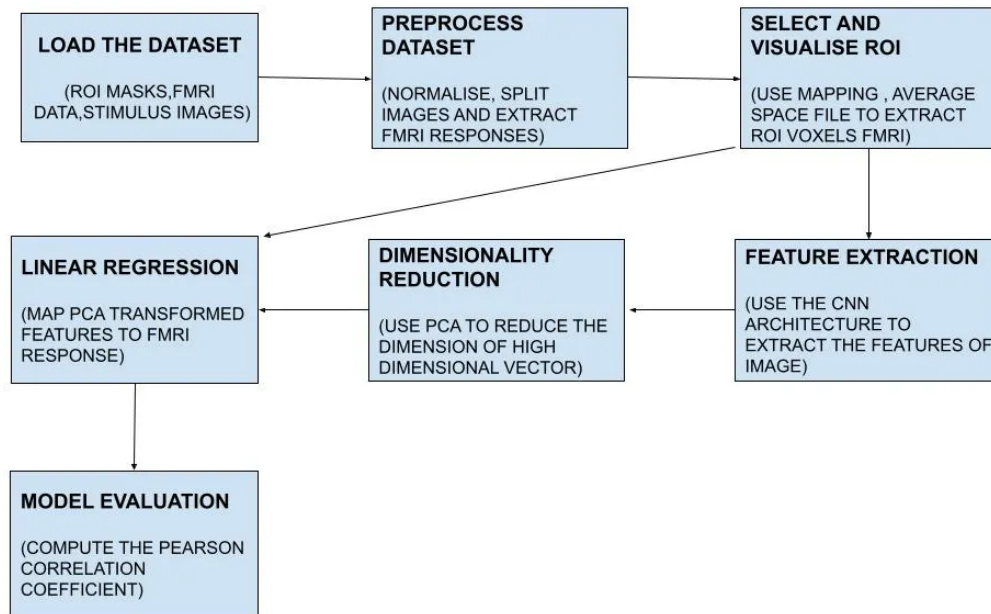
```

Mean Pearson Correlation for ROI FBA-2: 0.2691
Max Pearson Correlation for ROI FBA-2: 0.5236

```

Code for finding and plotting the pearson correlation coefficient

OVERALL PIPELINE:



RESULTS :

Now, we will compare the **correlation coefficient values** (mean and maximum) for both **Regions of Interest (V3v and FBA-2)** across the three models used—**AlexNet, ResNet, and EfficientNet**.

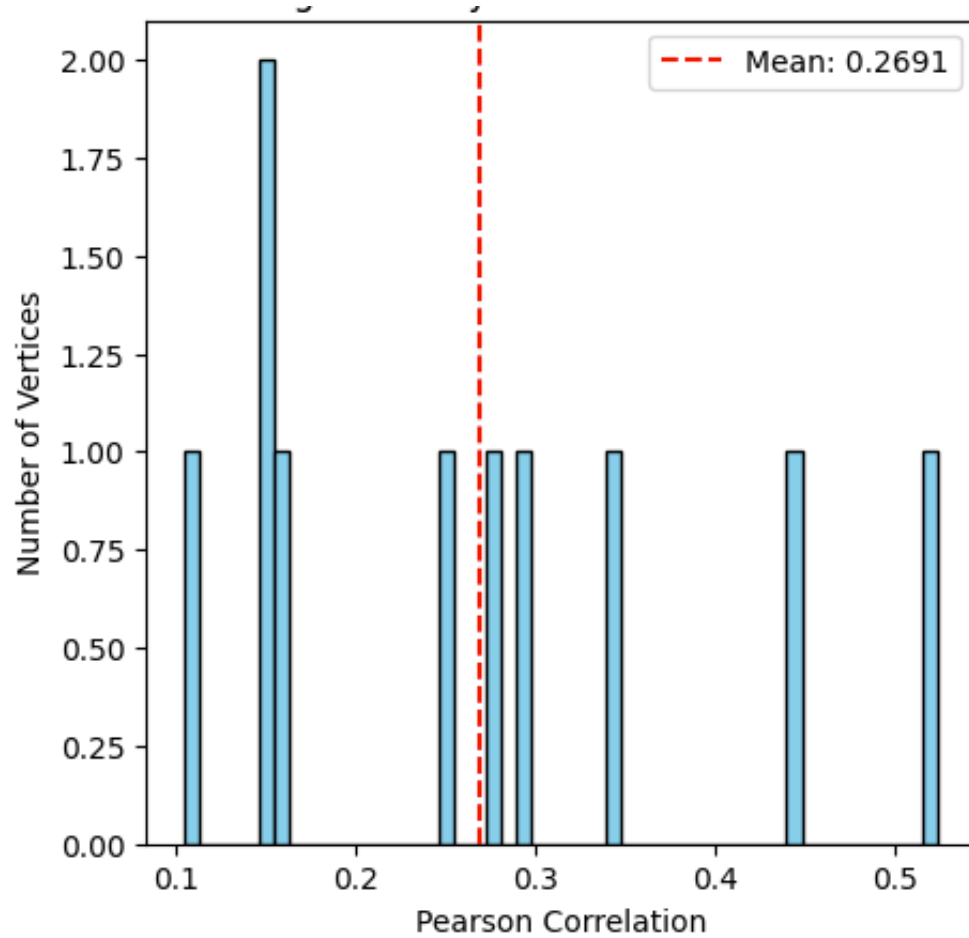
We will analyze the results **region-wise** and also compare **inter-region values**, providing reasons and hypotheses for the observed differences.

V3V :

1. ALEXNET

Mean Pearson Correlation: 0.2691

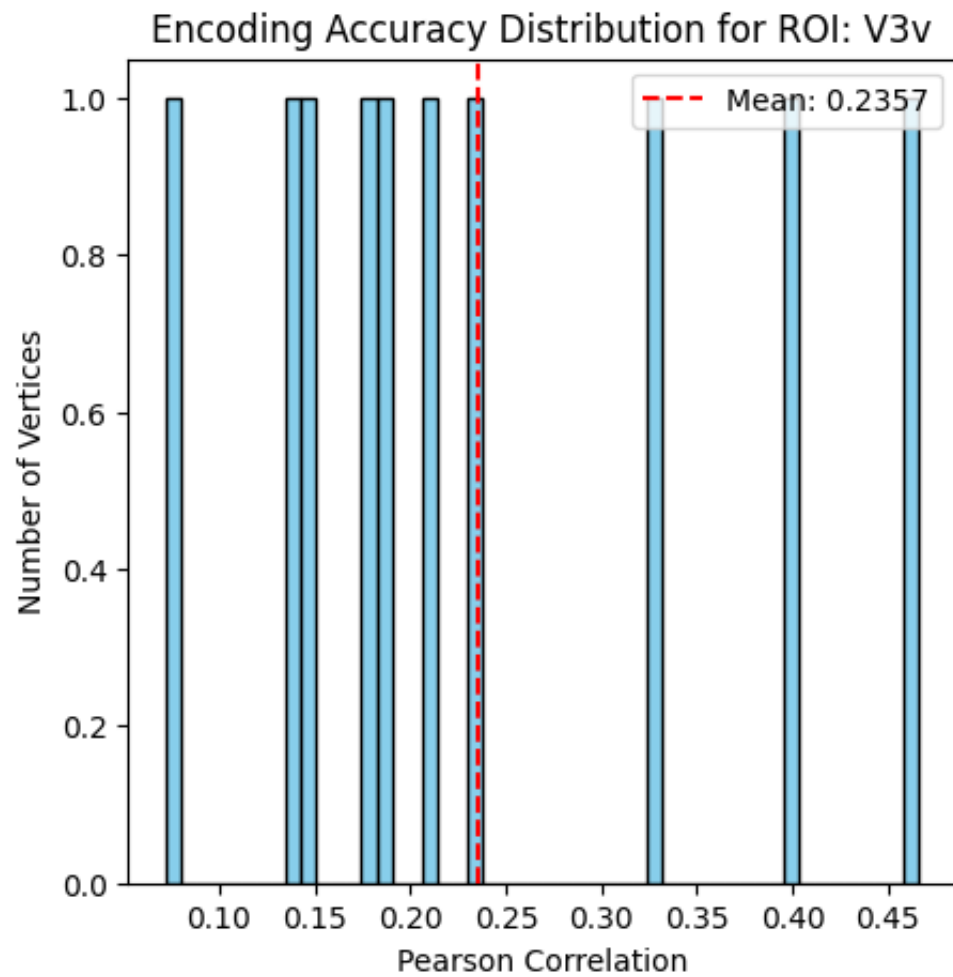
Max Pearson Correlation: 0.5236



2. RESNET:

Mean Pearson Correlation : 0.2357

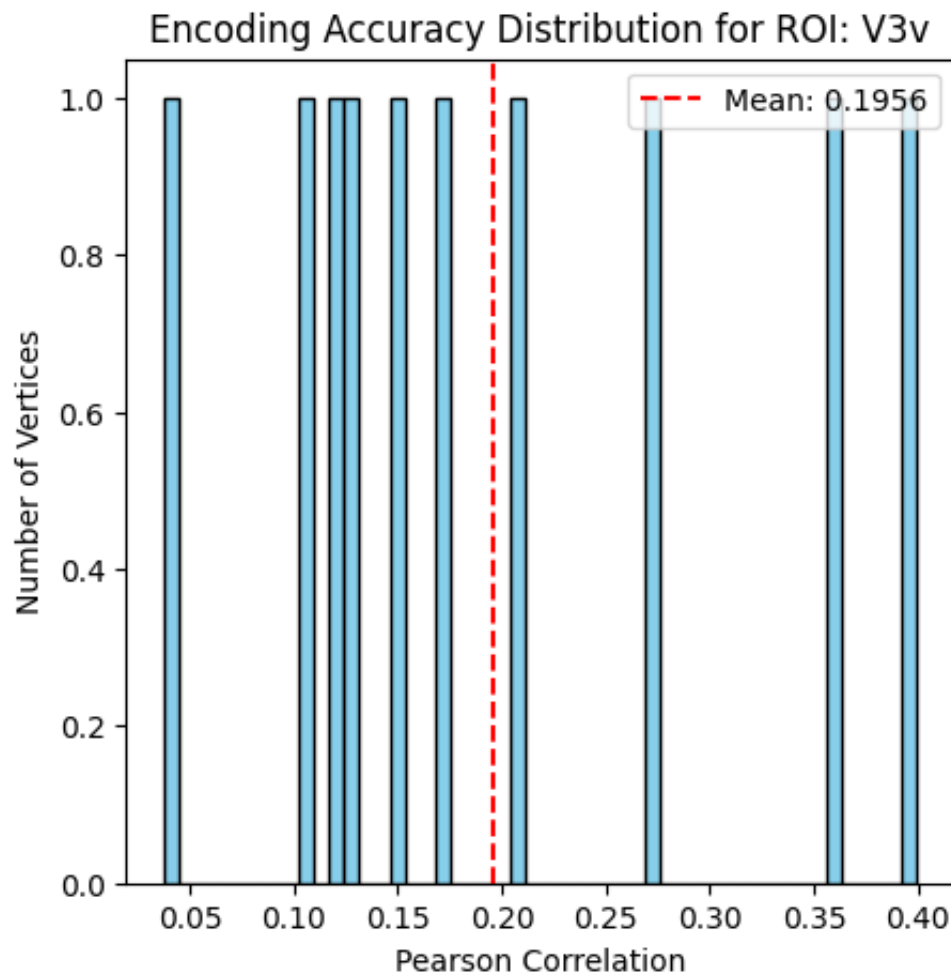
Max Pearson Correlation : 0.4660



3. EFFICIENTNET:

Mean Pearson Correlation : 0.1956

Max Pearson Correlation: 0.3995



Conclusion

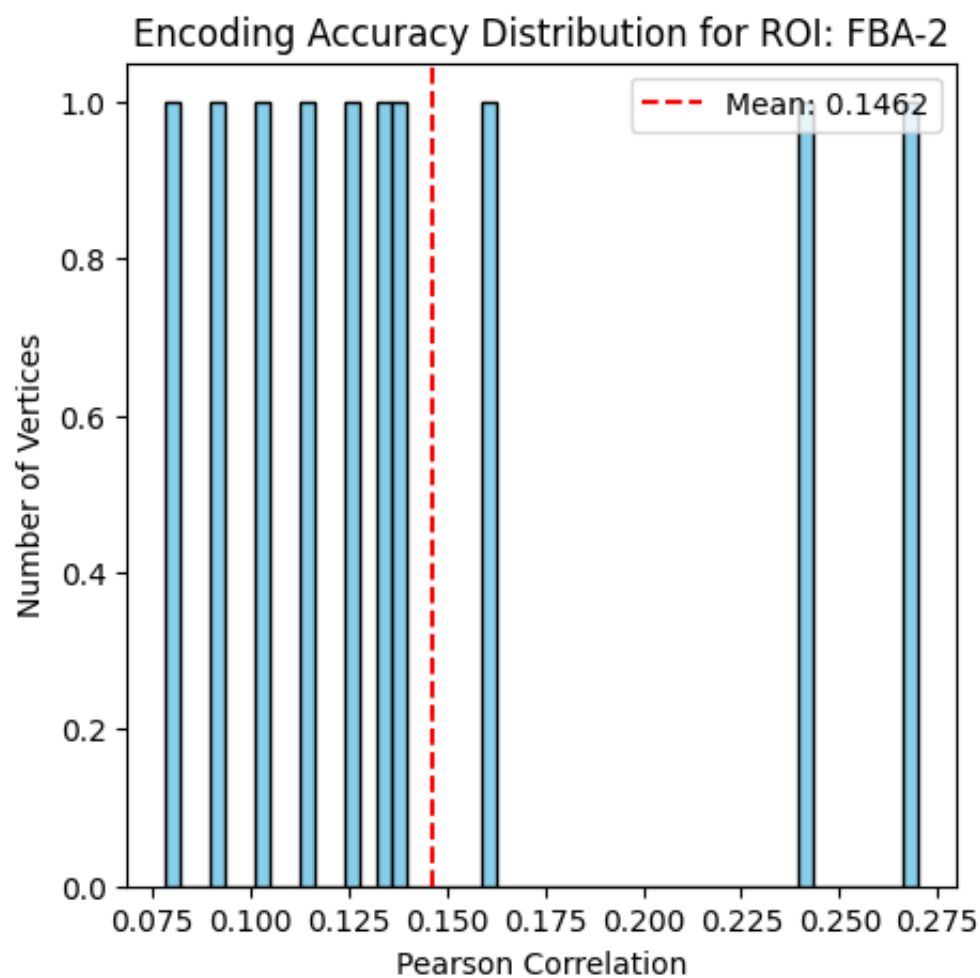
- **AlexNet performed best** (Mean: 0.2691, Max: 0.5236), suggesting its lower-level feature extraction aligns better with V3v.
- **ResNet had slightly lower correlations** (Mean: 0.2357, Max: 0.4660), likely due to its deeper hierarchical processing.
- **EfficientNet showed the weakest correlation** (Mean: 0.1956, Max: 0.3995), possibly because of its compact, highly optimized feature representations.
- **Overall, correlation values are low**, indicating CNN features capture some neural patterns but do not fully explain brain responses in V3v.

FBA-2:

1. ALEXNET :

Mean Pearson Correlation : 0.1462

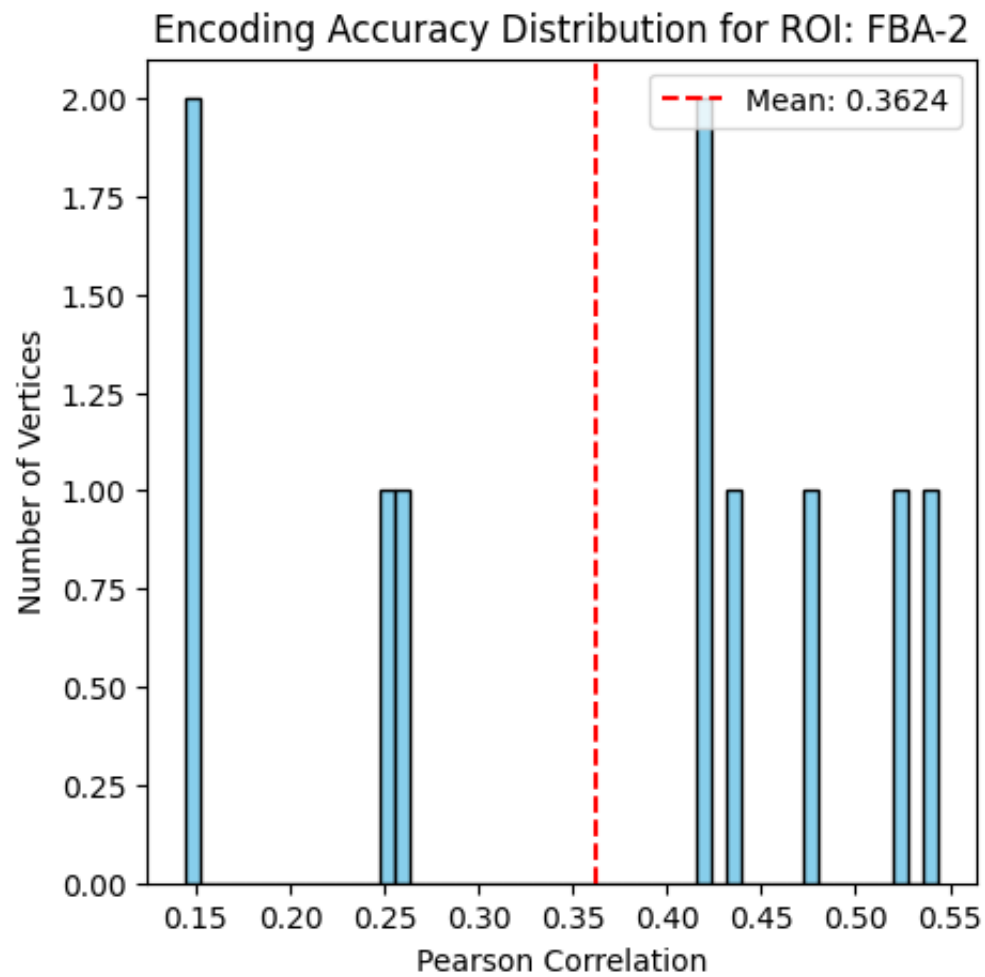
Max Pearson Correlation : 0.2701



2. RESNET:

Mean Pearson Correlation: 0.3624

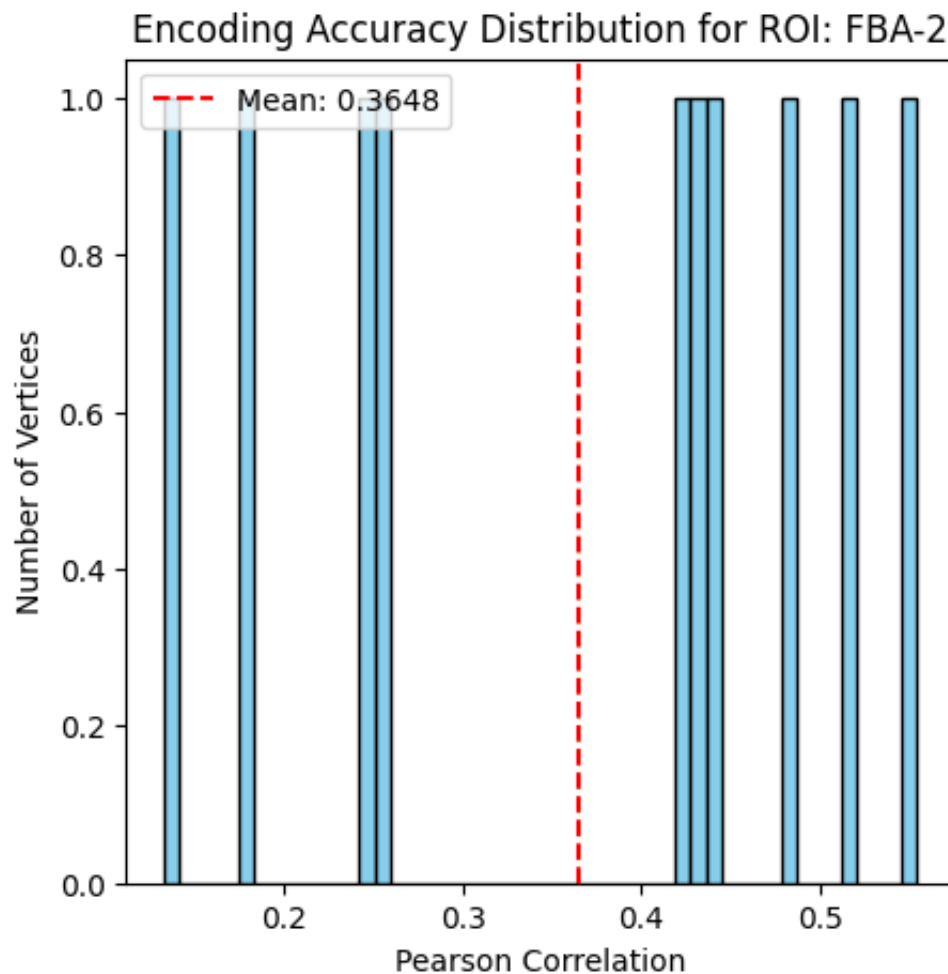
Max Pearson Correlation : 0.5436



3. EFFICIENTNET:

Mean Pearson Correlation :0.3648

Max Pearson Correlation: 0.5536



Conclusion (FBA-2)

- **ResNet and EfficientNet performed best**, with similar mean correlations (~0.36) and max values (~0.55), indicating that deeper architectures capture features relevant to FBA-2.
- **AlexNet performed significantly worse** (Mean: 0.1462, Max: 0.2701), suggesting its lower-level features are less aligned with FBA-2.
- **Higher correlations in FBA-2** compared to V3v suggest that deeper networks better represent **higher-order visual processing** involved in body recognition.

Inter-ROI Observations

- **V3v favored AlexNet**, while **FBA-2 favored deeper architectures** (ResNet, EfficientNet), aligning with their roles in **early vs. higher-level visual processing**.
- **Overall correlation values remain low**, indicating CNNs capture some neural representations but may lack temporal/contextual encoding needed for precise fMRI predictions.