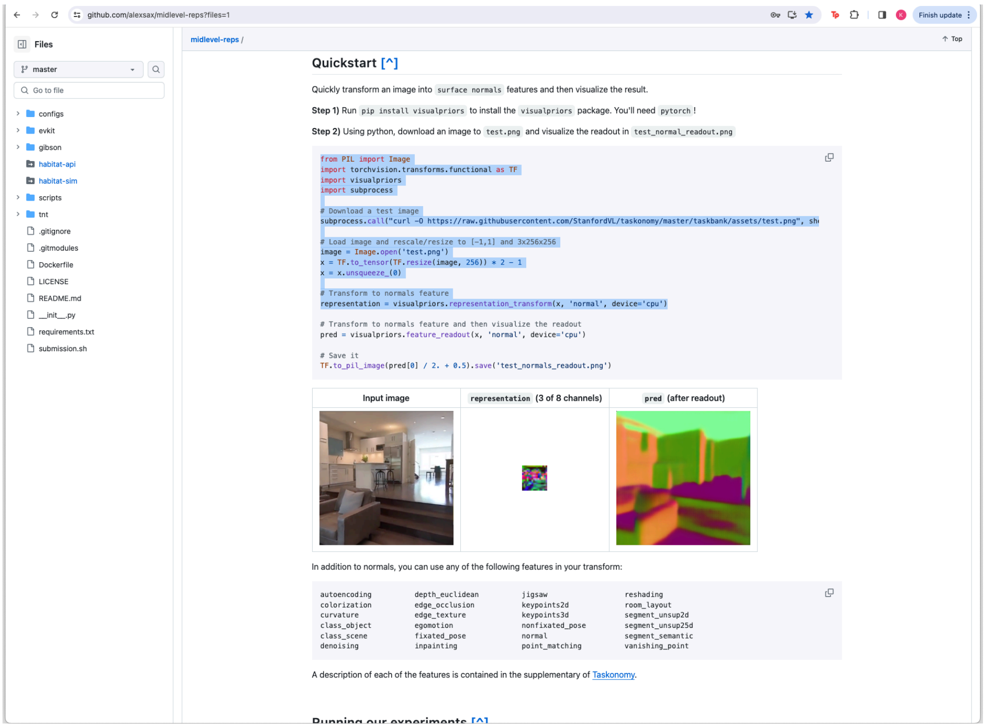
Question 1.1

I'd like to note I ran into some issues when trying to use the visualpriors package. On the GitHub page they only listed 24 representation models, not 25 as stated in the homework. Please see a screenshot of the webpage below for the listed models.



Quickstart [^]

Quickly transform an image into `surface normals` features and then visualize the result.

Step 1) Run `pip install visualpriors` to install the `visualpriors` package. You'll need `pytorch`!

Step 2) Using python, download an image to `test.png` and visualize the readout in `test_normal_readout.png`

```
from PIL import Image
import torchvision.transforms.functional as TF
import visualpriors
import subprocess

# Download a test image
subprocess.call("curl -O https://raw.githubusercontent.com/StanfordVL/taskonomy/master/taskbank/assets/test.png", sh

# Load image and rescale/resize to [-1,1] and 3x256x256
image = Image.open('test.png')
x = TF.to_tensor(TF.resize(image, 256)) * 2 - 1
x = x.unsqueeze_(0)

# Transform to normals feature
representation = visualpriors.representation_transform(x, 'normal', device='cpu')

# Transform to normals feature and then visualize the readout
pred = visualpriors.feature_readout(x, 'normal', device='cpu')

# Save it
TF.to_pil_image(pred[0] / 2. + 0.5).save('test_normals_readout.png')
```

Input image	`representation` (3 of 8 channels)	`pred` (after readout)

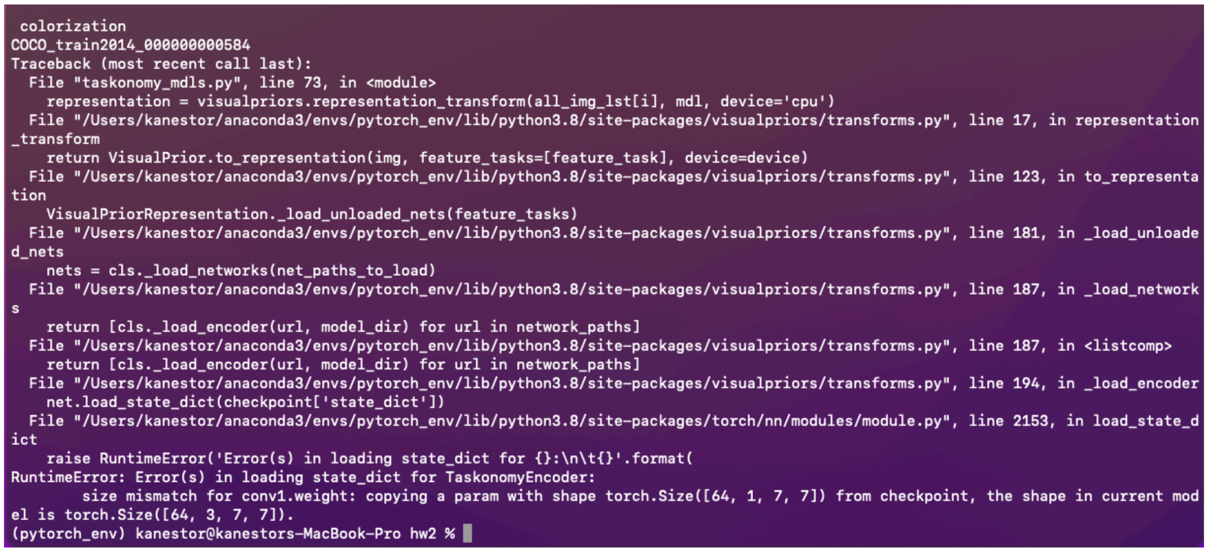
In addition to normals, you can use any of the following features in your transform:

```
autoencoding        depth_euclidean       jigsaw              reshading
colorization        edge_occlusion        keypoints2d         room_layout
curvature           edge_texture          keypoints3d         segment_unsup2d
class_object        egomotion             nonfixated_pose     segment_unsup25d
class_scene         fixated_pose          normal              segment_semantic
denoising           inpainting            point_matching      vanishing_point
```

A description of each of the features is contained in the supplementary of Taskonomy.

Running our experiments [^]

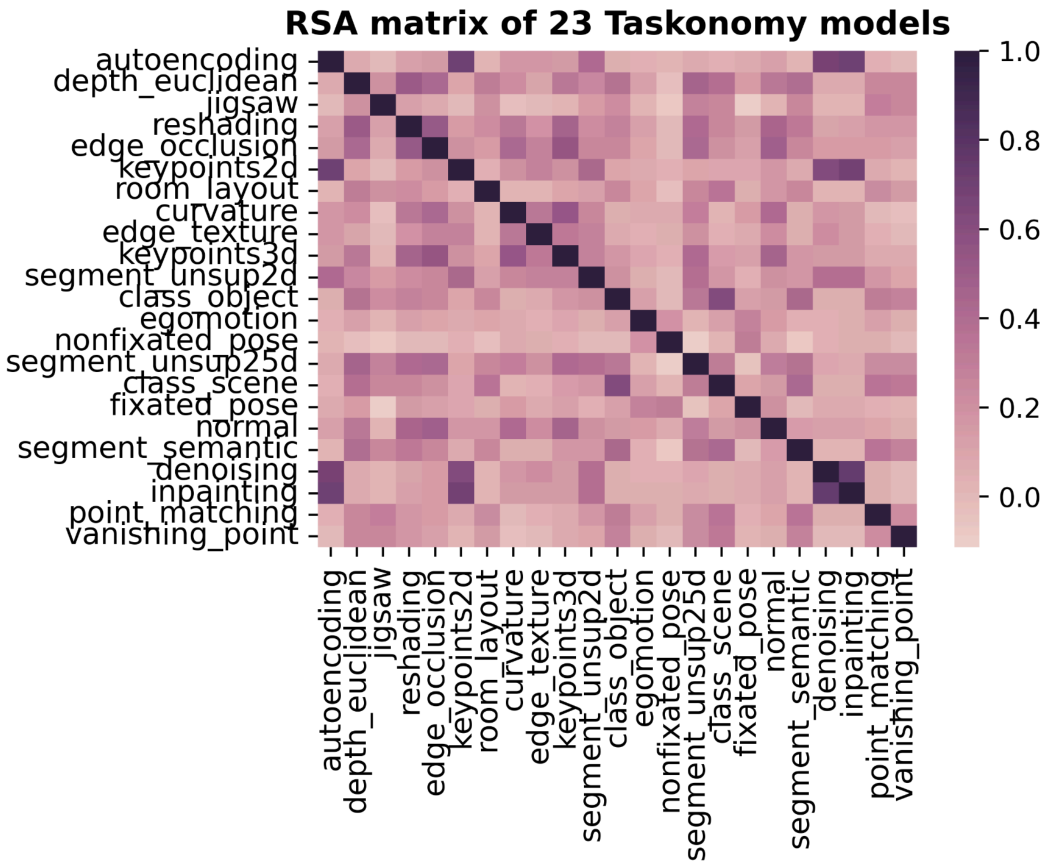
I was able to run 23 of the models except for the **colorization** model where I got the errors noted in the terminal screenshow below. I'm not sure why this model won't run since all the input stimuli were reshaped to [3, 256, 256] and normalized to [-1,1] and the other 23 models seemed to run just fine. As such my homework results that follow encompass the 23 other models I was able to run from the **visualpriors** package. On Slack Joel Ye (TA) mentioned that it was ok to skip models that did not run.



```
 colorization
COCO_train2014_000000000584
Traceback (most recent call last):
  File "taskonomy_mdls.py", line 73, in <module>
    representation = visualpriors.representation_transform(all_img_lst[i], mdl, device='cpu')
  File "/Users/kanestor/anaconda3/envs/pytorch_env/lib/python3.8/site-packages/visualpriors/transforms.py", line 17, in representation
_transform
    return VisualPrior.to_representation(img, feature_tasks=[feature_task], device=device)
  File "/Users/kanestor/anaconda3/envs/pytorch_env/lib/python3.8/site-packages/visualpriors/transforms.py", line 123, in to_representa
tion
    VisualPriorRepresentation._load_unloaded_nets(feature_tasks)
  File "/Users/kanestor/anaconda3/envs/pytorch_env/lib/python3.8/site-packages/visualpriors/transforms.py", line 181, in _load_unloade
d_nets
    nets = cls._load_networks(net_paths_to_load)
  File "/Users/kanestor/anaconda3/envs/pytorch_env/lib/python3.8/site-packages/visualpriors/transforms.py", line 187, in _load_network
s
    return [cls._load_encoder(url, model_dir) for url in network_paths]
  File "/Users/kanestor/anaconda3/envs/pytorch_env/lib/python3.8/site-packages/visualpriors/transforms.py", line 187, in <listcomp>
    return [cls._load_encoder(url, model_dir) for url in network_paths]
  File "/Users/kanestor/anaconda3/envs/pytorch_env/lib/python3.8/site-packages/visualpriors/transforms.py", line 194, in _load_encoder
    net.load_state_dict(checkpoint['state_dict'])
  File "/Users/kanestor/anaconda3/envs/pytorch_env/lib/python3.8/site-packages/torch/nn/modules/module.py", line 2153, in load_state_d
ict
    raise RuntimeError('Error(s) in loading state_dict for {}:\n\t{}'.format(
RuntimeError: Error(s) in loading state_dict for TaskonomyEncoder:
        size mismatch for conv1.weight: copying a param with shape torch.Size([64, 1, 7, 7]) from checkpoint, the shape in current mod
el is torch.Size([64, 3, 7, 7]).
(pytorch_env) kanestor@kanestors-MacBook-Pro hw2 %
```

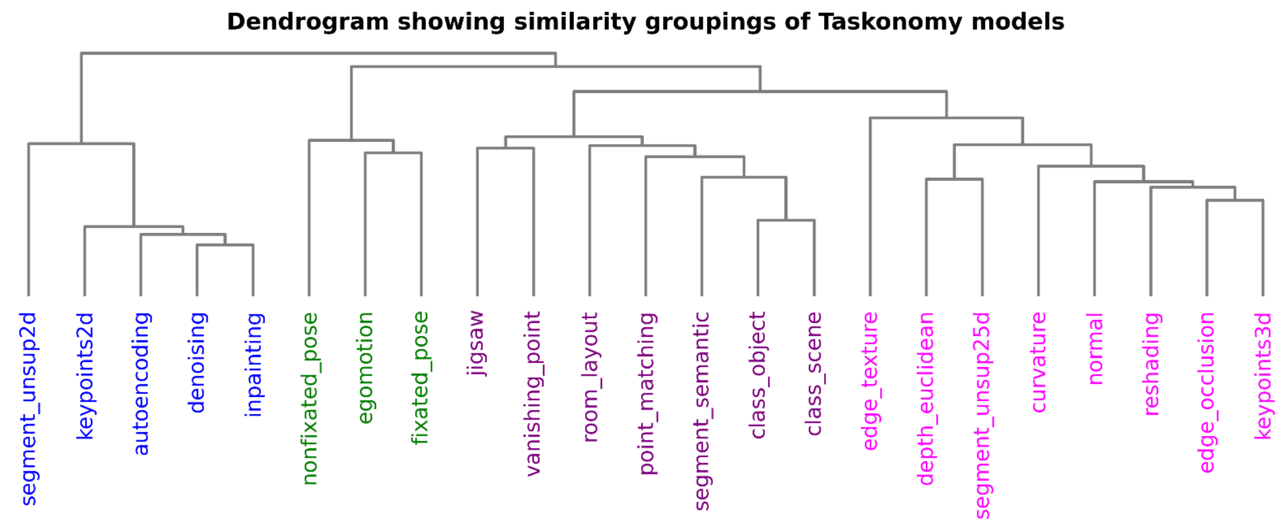
In the figure obtained for this homework, most notably the following models have a high degree of similarity based on the computed RSA matrix from the stimulus figures: autoencoding, inpainting, denoising, keypoints2d, reshading, edge_occlusion, depth_euclidean, class_object, class_scene. These were the model instances with RSA closest to 1, though there are other diffuse block like patterns throughout the matrix, indicating models with lower similarity ranges of 0.3-0.6.

The similarity matrix from Wang et al. (2019) Fig. 6, has a more clearly defined blockwise structure, indicating more instances of similarity in the taskonomy encoding brain models that are being examined in that paper. Since this figure was based on actual neural data this is a positive result to obtain that task representations across participants are similarly encoded.



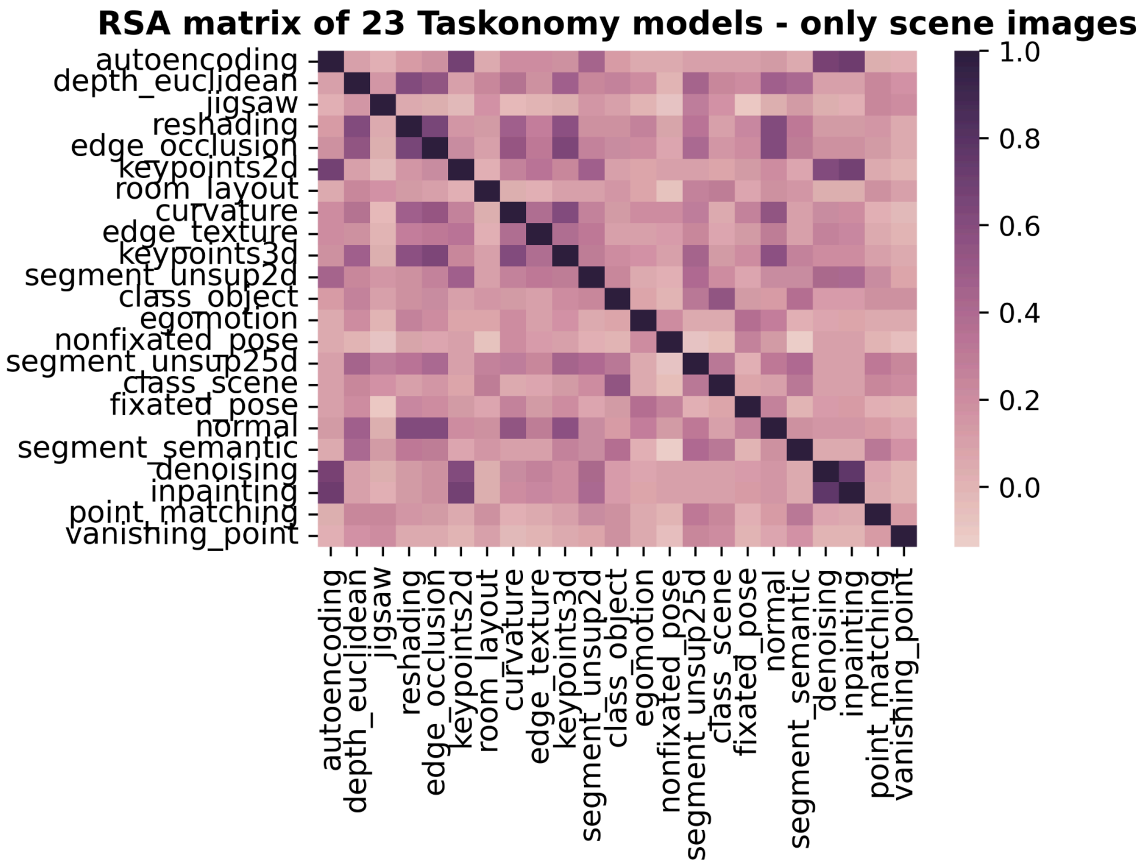
Question 1.2

The dendrogram tree generated for this homework (below) is based on the 23 Taskonomy models, with Bold5000 stimuli images as input. It uses average clustering on the RSA matrix to show which of the Taskonomy models can be grouped together based on their similarity values. This output figure is most similar to the original Tasknomy paper by Zamir et al. (2019), than to the neural Tasknomoy paper by Wang et al. (2019). This is because the original paper uses training data such as internal scenes to train models to better navigate those terrains and does not incorporate neural data. So the output representations are the semantic meanings these models were able to converge to in order to learn that terrain and transfer this knowledge to other similar terrains. Whereas the neural Taskonomy dendrogram tree is based on a comparison of brain representations of these terrains, so intuitively it makes most sense for the dendrogram output for this paper which is not based on neural data but on model task representations of various terrains to be most similar to another model based terrain representation.

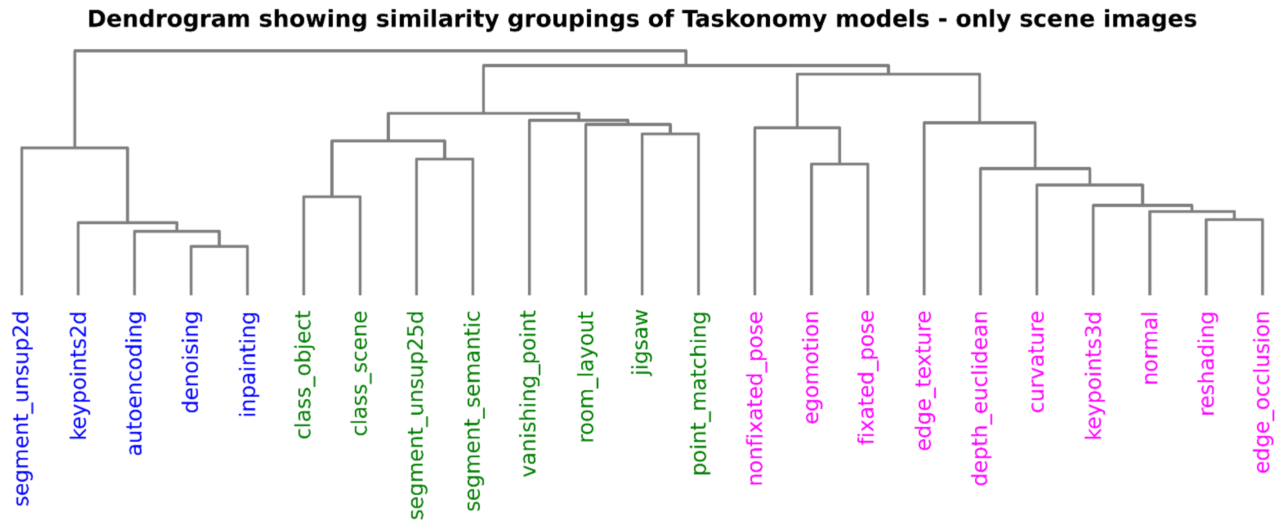


Question 1.3

The RSA matrix below was generated using the Taskonomy models with only scene images used as input. The matrix output is similar to the previously generated matrix including all the presented stimuli images, though there was stronger similarity relationships, noted by the darker blocks of colour in the image. This indicates that while the taskonomy models are only trained on indoor images there is significant transfer of learning to enable the models to effectively navigate if not perform better at outdoor scene representations.

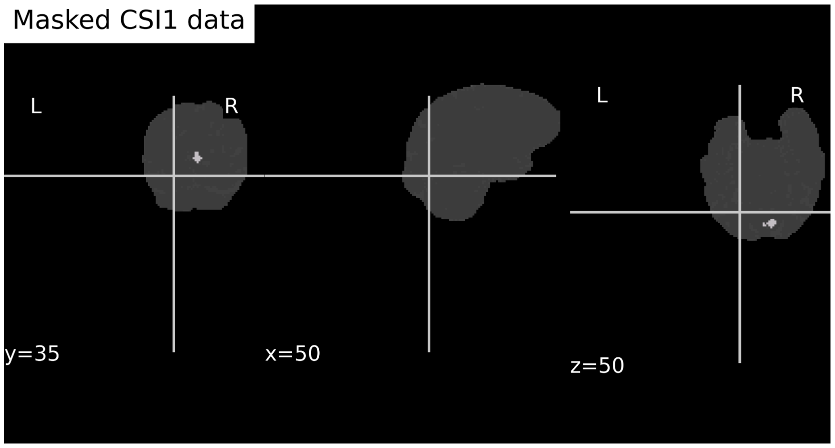


The dendogram below, generated using only the scene images, has a different hierarchical structure in comparison to the one above encompassing all the stimuli images. In this tree there are three main groupings, where previously there were four. The models nonfixated_pose, egomotion and fixated_pose are no longer in an independent branch of the tree, but has now joined the branch including the following models: edge_texture, depth_euclidean, curvature, keypoints3d, normal, reshading and edge_occlusion. Since the inputs are only scene images, they most likely have fewer animated and fixed humans and animal objects, so this explains the removal of a separate branch for nonfixated_pose, egomotion and fixated_pose. These motion centric models then became grouped with models that determine edges, depth and distance which would be used prominently to analyse images of scenes and landscapes.

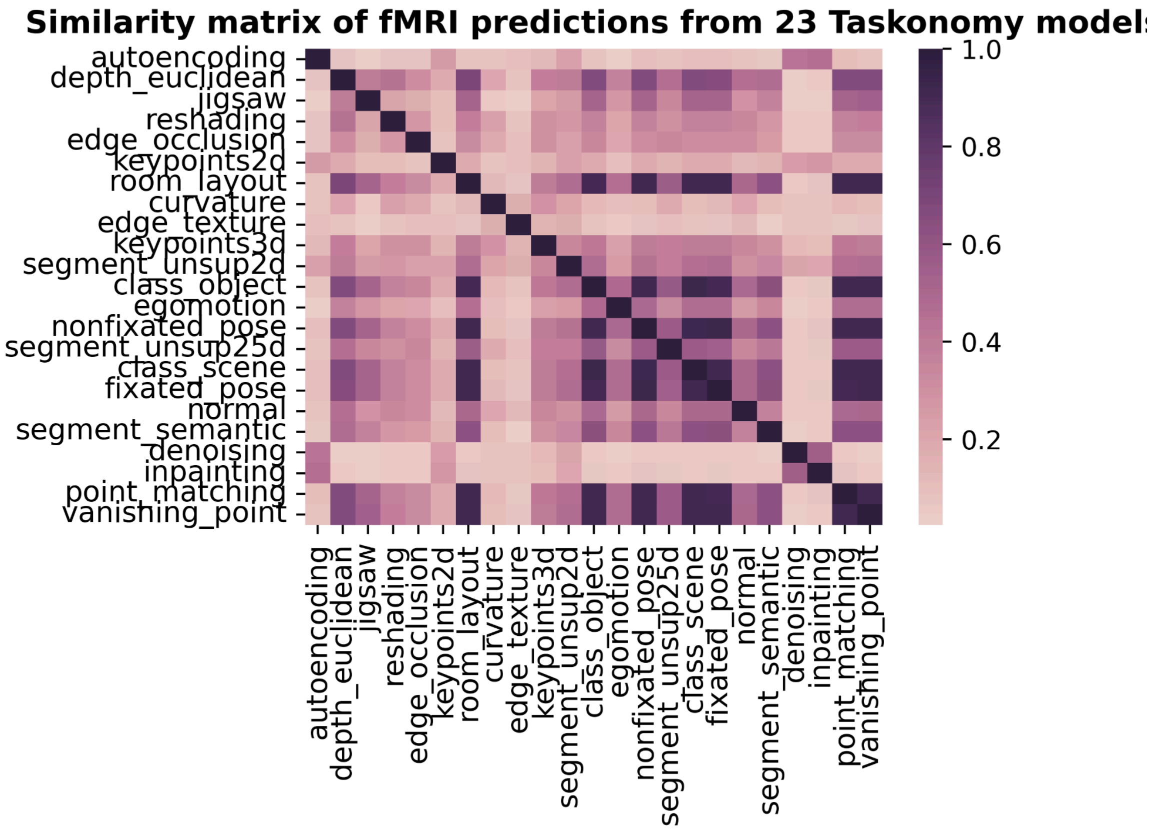


Question 2.1

I tried using the binarized version of the given brain mask from /sub-CS1/anatomicals/brainmask.nii.gz using Nilearn, however I kept getting errors and warnings that processing would take too long because of nans in the original fMRI data. Eventually the jobs would get killed without processing. I used the NiftiMasker automatic brain mask from Nilearn instead with the mask_strategy='epi', which produced a good mask of brain voxels only. An image of the resulting mask is shown below.



Shown below is the similarity matrix constructed from correlations of fMRI predictions based on taskonomy model representations.



Question 2.2

In my permutation analysis I did not have to drop any voxels of the representation model predictions of fMRI responses since all 2048 voxels for each model were significant. This could in part be due to how accurately the modelsl are aat predicting the fMRI responses as well as the fact I used Lanczos resampling to downsize the fMRI voxels to match that of the representational model vectors before training the encoding model. This resampling could have already handled potentially non-significant voxels that would have been removed. Considering all that and the fact that the similarity matrix based on the fMRI predictions from the representational models have more more instances of higher similarity in comparison to the RSA matrix comparing the representation model embeddings, this indicates that many of the predictions overall have a high similarity to each other and by implication the brain since all voxels were significant in comparison to the true fMRI data - session 15 was used as the test set.

This similarity model - compute correlations, get dot product, calculate cosine similarity - of the entire fMRI matrix, gives us an overall comparison of similarity across the different models for the fMRI predictions. This metric does not however allow us to get an in depth understanding of how the representation models are related to each other based on the individual images presented during the fMRI timescale. Models that have an overall low similarity score i.e. dissimilar may have high similarity scores for specific types of images presented during the imaging session.

Question 2.3

Shown below is the R^2 for five of the representation models, comparing the fMRI predictions from those models to the true fMRI test from session 15. All values are negative indicating that the fMRI predictions from these models are not good predictions in comparison to the true fMRI response. The R^2 from the autoencoding model is slightly less negative in comparison to the other models.

```
{'autoencoding': -0.25323553942092103,
'depth_euclidean': -0.26807558917003294,
'jigsaw': -0.39874081584738796,
'reshading': -0.2584641838481489,
'edge_occlusion': -0.29464452869789587}
```

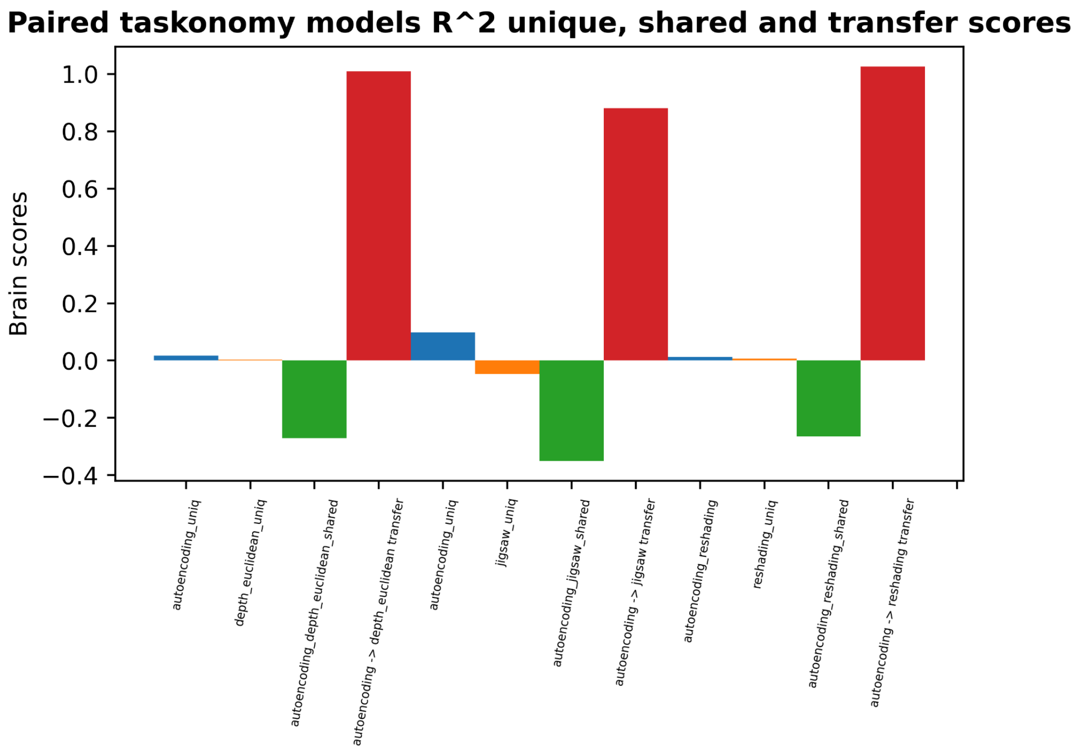
Question 2.4
Shown below are the R^2 values obtained from concatenating the embeddings of three model pairs before training an encoding model, and then comparing the true and prediction fMRI responses. **autoencoding** and **depth_euclidian** concatenation have about the same R^2 value as the single models, shown in the question above. This makes sense since they are a concatenation. **autoencoding** and **jigsaw** concatenation seem to be an average of the two R^2 shown in the single models. Again the **autoencoding** and **reshading** concatenations have about the same R^2 as the single models. None of these values are surprising since we only combined the embeddings of the two models before training and did not perform any other operations.

```
{'autoencoding_depth_euclidean': -0.2505617174037627,
'autoencoding_jigsaw': -0.30088393236056754,
'autoencoding_reshading': -0.24643125184870274}
```

Question 2.5

Shown below are the $$R^2$$ unique scores for each of the pair taskonomy models used for encoding, the shared $$R^2$$ performance of each model, and the asymmetric model transfer scores highlighting how much influence one model has on over another in the fMRI predictions. The model transfer scores are in line with those shown in Fig. 7 of the Zamir et al. (2019) paper, and there was no clear difference between the models analysed here and those from the paper. So there is no clear front runner model that is better in encoding predictions, even when concatenating two models they both perform at about the same level. This is not surprising since the $$R^2$$ results from previous sections show values to be about the same. We can hypothesise this is the case since the $$R^2$$ values comparing true versus prediction fMRI responses did not indicate the models were good predictors of the fMRI response.

```
# [mdl1, mdl2] R^2 unique scores
{'autoencoding_depth_euclidean': [0.017513871766270228, 0.0026738220171583227],
'autoencoding_jigsaw': [0.09785688348682042, -0.04764839293964651],
'autoencoding_reshading': [0.012032931999446161, 0.006804287572218293]}
# shared R^2 performance
{'autoencoding_depth_euclidean': -0.27074941118719126,
'autoencoding_jigsaw': -0.35109242290774145,
'autoencoding_reshading': -0.26526847142036714}
# model transfer scores - mdl1 -> mdl2
{'autoencoding_depth_euclidean': 1.0099741346291042,
'autoencoding_jigsaw': 0.8805028453423157,
'autoencoding_reshading': 1.0263258431822642}
```


Paired taskonomy models R^2 unique, shared and transfer scores
Brain scores
autoencoding_uniq
depth_euclidean_uniq
autoencoding_depth_euclidean_shared
autoencoding -> depth_euclidean transfer
autoencoding_uniq
jigsaw_uniq
autoencoding_jigsaw_shared
autoencoding -> jigsaw transfer
autoencoding_reshading
reshading_uniq
autoencoding_reshading_shared
autoencoding -> reshading transfer