interpretation-mri

October 11, 2024

[1]:
```
from __future__ import absolute_import, division, print_function,␣
 ↪unicode_literals

%load_ext autoreload
%autoreload 2

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # adapt plots for retina␣
 ↪displays

from IPython.core.debugger import set_trace
```

[2]:
```
import os
# import utils_custom as utils
from tqdm import tqdm_notebook
from sklearn.model_selection import train_test_split
import multiprocessing
import json
```

[3]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
from torch.utils.data import Dataset, DataLoader
```

[4]:
```
import datasets, models, interpretation, utils
```

1 Load model, dataset and brain area masks

[5]:
```
net = models.ClassificationModel3D()
net.load_state_dict(torch.load('output/models/softmax-output_state-dict.pt'))
```

```
C:\Users\amana\AppData\Local\Temp\ipykernel_3796\3196439524.py:2: FutureWarning:
You are using `torch.load` with `weights_only=False` (the current default
value), which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code during
unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  net.load_state_dict(torch.load('output/models/softmax-output_state-dict.pt'))
```

[5]:
```
<All keys matched successfully>
```

[6]:
```
if torch.cuda.is_available():
    net.cuda()
    cuda_device = torch.cuda.current_device()
    print('Moved network to GPU')
else:
    cuda_device = -1
    print('GPU not available')
```

```
Moved network to GPU
```

[7]:
```
# Important: Set model to eval before using any interpretation methods
net.eval();
```

[8]:
```
# Load the data table with all 3 Tesla MRI scans from ADNI.
# df = datasets.load_data_table_15T()
table_train = r'C:\AD_detection_v2\ADNI1_Screening_1.5T_8_25_2024_Train.csv'
image_dir= r'D:\ADNI data'
table_test = r'C:\AD_detection_v2\ADNI1_Screening_1.5T_8_25_2024_Test.csv'

df_train = datasets.load_data_table(table_train,image_dir)
df_test = datasets.load_data_table(table_test,image_dir)
```

```
Loading dataframe for
C:\AD_detection_v2\ADNI1_Screening_1.5T_8_25_2024_Train.csv
Found 224 images in table
```

```
Filtered out 0 of 224 images because of failed preprocessing
Filtered out 0 of 224 images because of missing files
Final dataframe contains 224 images from 224 patients

Loading dataframe for C:\AD_detection_v2\ADNI1_Screening_1.5T_8_25_2024_Test.csv
Found 103 images in table
Filtered out 0 of 103 images because of failed preprocessing
Filtered out 0 of 103 images because of missing files
Final dataframe contains 103 images from 103 patients
```

[9]:
```
df_test.head()
```

[9]:

	Image Data ID	Subject	Group	Sex	Age	Visit	Modality	Description	Type	Acq Date	Format	Downloaded	filepath
0	I45117	002_S_0413	CN	F	76	sc	MRI	MPR; GradWarp; B1 Correction; N3; Scaled	Processed	2006-02-05 00:00:00	NiFTI	2024-01-02 00:00:00	D:\ADNI data/CN and AD/CN/002_S_0413/MPR__Grad…
1	I40683	002_S_0685	CN	F	90	sc	MRI	MPR; GradWarp; B1 Correction; N3; Scaled	Processed	2006-06-07 00:00:00	NiFTI	2024-01-02 00:00:00	D:\ADNI data/CN and AD/CN/002_S_0685/MPR__Grad…
2	I40817	002_S_1018	AD	F	71	sc	MRI	MPR; GradWarp; B1 Correction; N3; Scaled	Processed	11/29/2006	NiFTI	2024-01-02 00:00:00	D:\ADNI data/CN and AD/AD/002_S_1018/MPR__Grad…
3	I32855	005_S_0223	CN	F	78	sc	MRI	MPR; GradWarp; B1 Correction; N3; Scaled	Processed	2/28/2006	NiFTI	2024-01-02 00:00:00	D:\ADNI data/CN and AD/CN/005_S_0223/MPR__Grad…
4	I35790	007_S_0068	CN	F	75	sc	MRI	MPR-R; GradWarp; B1 Correction; N3; Scaled	Processed	11/30/2005	NiFTI	2024-01-02 00:00:00	D:\ADNI data/CN and AD/CN/007_S_0068/MPR-R__Gr…

[10]:
```
patients_train = df_train.to_numpy()
patients_test = df_test.to_numpy()

df = pd.concat([df_train, df_test])
```

[11]:
```
train_dataset, test_dataset = datasets.build_datasets(df, patients_train,␣
↪patients_test, normalize=False)
```

```
+-------+----------+---------+---------+------------+---------+---------+
|       |   Images |   -> AD |   -> CN |   Patients |   -> AD |   -> CN |
+=======+==========+=========+=========+============+=========+=========+
| All   |      327 |     148 |     179 |        327 |     148 |     179 |
+-------+----------+---------+---------+------------+---------+---------+
| Train |      224 |     112 |     112 |        224 |     112 |     112 |
+-------+----------+---------+---------+------------+---------+---------+
| Val   |      103 |      36 |      67 |        103 |      36 |      67 |
+-------+----------+---------+---------+------------+---------+---------+
```

```
Dataset is already normalized
```

[12]:
```
# Set up a binary mask for each brain area based on the AAL atlas.
# This is required for brain area occlusion and to determine the relevance per␣
↪area.
# The AAL atlas can for example be retrieved from here: https://github.com/
↪neurolabusc/MRIcron/blob/master/templates/aal.nii.gz
brain_map = utils.load_nifti('data/aal.nii.gz')
brain_areas = np.unique(brain_map)[1:]  # omit background

area_masks = []
for area in brain_areas:
    area_mask = np.zeros_like(brain_map)
    area_mask[brain_map == area] = 1
    area_mask = utils.resize_image(area_mask, utils.load_nifti(test_dataset.
↪filenames[0]).shape, interpolation=0)
    area_masks.append(area_mask)
```

```
area_names = ['Precentral_L', 'Precentral_R', 'Frontal_Sup_L', 'Frontal_Sup_R',␣
 ↪'Frontal_Sup_Orb_L', 'Frontal_Sup_Orb_R', 'Frontal_Mid_L', 'Frontal_Mid_R',␣
 ↪'Frontal_Mid_Orb_L', 'Frontal_Mid_Orb_R', 'Frontal_Inf_Oper_L',␣
 ↪'Frontal_Inf_Oper_R', 'Frontal_Inf_Tri_L', 'Frontal_Inf_Tri_R',␣
 ↪'Frontal_Inf_Orb_L', 'Frontal_Inf_Orb_R', 'Rolandic_Oper_L',␣
 ↪'Rolandic_Oper_R', 'Supp_Motor_Area_L', 'Supp_Motor_Area_R', 'Olfactory_L',␣
 ↪'Olfactory_R', 'Frontal_Sup_Medial_L', 'Frontal_Sup_Medial_R',␣
 ↪'Frontal_Med_Orb_L', 'Frontal_Med_Orb_R', 'Rectus_L', 'Rectus_R',␣
 ↪'Insula_L', 'Insula_R', 'Cingulum_Ant_L', 'Cingulum_Ant_R',␣
 ↪'Cingulum_Mid_L', 'Cingulum_Mid_R', 'Cingulum_Post_L', 'Cingulum_Post_R',␣
 ↪'Hippocampus_L', 'Hippocampus_R', 'ParaHippocampal_L', 'ParaHippocampal_R',␣
 ↪'Amygdala_L', 'Amygdala_R', 'Calcarine_L', 'Calcarine_R', 'Cuneus_L',␣
 ↪'Cuneus_R', 'Lingual_L', 'Lingual_R', 'Occipital_Sup_L', 'Occipital_Sup_R',␣
 ↪'Occipital_Mid_L', 'Occipital_Mid_R', 'Occipital_Inf_L', 'Occipital_Inf_R',␣
 ↪'Fusiform_L', 'Fusiform_R', 'Postcentral_L', 'Postcentral_R',␣
 ↪'Parietal_Sup_L', 'Parietal_Sup_R', 'Parietal_Inf_L', 'Parietal_Inf_R',␣
 ↪'SupraMarginal_L', 'SupraMarginal_R', 'Angular_L', 'Angular_R',␣
 ↪'Precuneus_L', 'Precuneus_R', 'Paracentral_Lobule_L',␣
 ↪'Paracentral_Lobule_R', 'Caudate_L', 'Caudate_R', 'Putamen_L', 'Putamen_R',␣
 ↪'Pallidum_L', 'Pallidum_R', 'Thalamus_L', 'Thalamus_R', 'Heschl_L',␣
 ↪'Heschl_R', 'Temporal_Sup_L', 'Temporal_Sup_R', 'Temporal_Pole_Sup_L',␣
 ↪'Temporal_Pole_Sup_R', 'Temporal_Mid_L', 'Temporal_Mid_R',␣
 ↪'Temporal_Pole_Mid_L', 'Temporal_Pole_Mid_R', 'Temporal_Inf_L',␣
 ↪'Temporal_Inf_R', 'Cerebelum_Crus1_L', 'Cerebelum_Crus1_R',␣
 ↪'Cerebelum_Crus2_L', 'Cerebelum_Crus2_R', 'Cerebelum_3_L', 'Cerebelum_3_R',␣
 ↪'Cerebelum_4_5_L', 'Cerebelum_4_5_R', 'Cerebelum_6_L', 'Cerebelum_6_R',␣
 ↪'Cerebelum_7b_L', 'Cerebelum_7b_R', 'Cerebelum_8_L', 'Cerebelum_8_R',␣
 ↪'Cerebelum_9_L', 'Cerebelum_9_R', 'Cerebelum_10_L', 'Cerebelum_10_R',␣
 ↪'Vermis_1_2', 'Vermis_3', 'Vermis_4_5', 'Vermis_6', 'Vermis_7', 'Vermis_8',␣
 ↪'Vermis_9', 'Vermis_10']

# Merge left and right areas.
merged_area_names = [name[:-2] for name in area_names[:108:2]] + area_names[108:
 ↪]
```

[13]:
```
def get_relevance_per_area(relevance_map, normalize=True):
    """
    Calculate and return the relevance per area based on the provided relevance␣
 ↪map.

    Args:
        relevance_map (numpy.ndarray): A 2D array representing the relevance␣
 ↪scores across different areas.
        normalize (bool): If True, normalize the relevance scores so that their␣
 ↪sum equals 1.

    Returns:
```

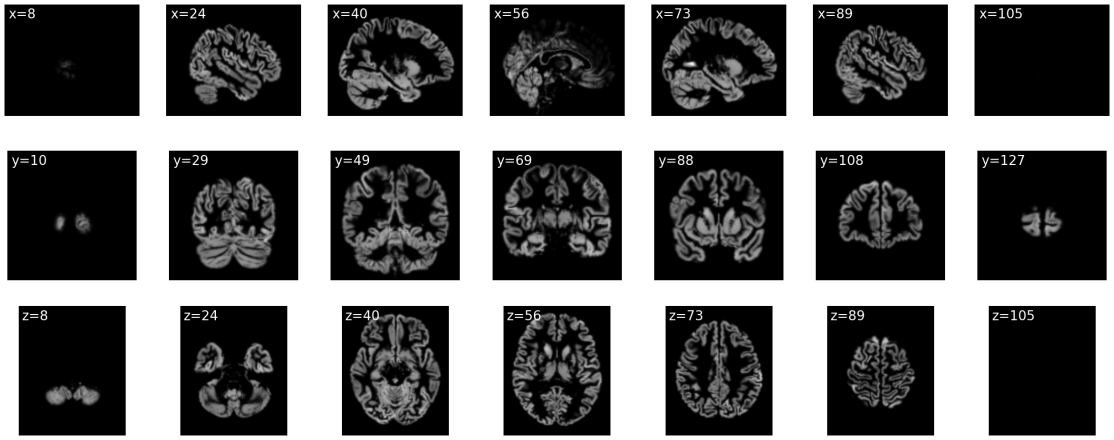
```
        list of tuples: A sorted list of tuples where each tuple contains an␣
↪area name and its normalized relevance score,
                        sorted in descending order of relevance.
    """

    # Assuming `area_masks` is a predefined list of masks for different areas
    relevances = np.zeros(len(area_masks))

    for i, area_mask in enumerate(area_masks):
        relevances[i] = np.sum(relevance_map * area_mask)  # Compute relevance␣
↪for each area

    if normalize:
        relevances /= relevances.sum()  # Normalize so all areas sum to 1

    # Merge left and right areas
    merged_relevances = np.concatenate([relevances[:108].reshape(-1, 2).sum(1),␣
↪relevances[108:]])

    # Assuming `merged_area_names` is a predefined list of area names
    return sorted(zip(merged_area_names, merged_relevances), key=lambda x:␣
↪x[1], reverse=True)
```

2 Relevance heatmaps for single images

2.1 Raw image

[14]:
```
which = 7

image_tensor = test_dataset[which][0]
raw_image = test_dataset.get_raw_image(which)

utils.plot_slices(raw_image, num_slices=7)
```

```
0.0 1.2856388 None None
```



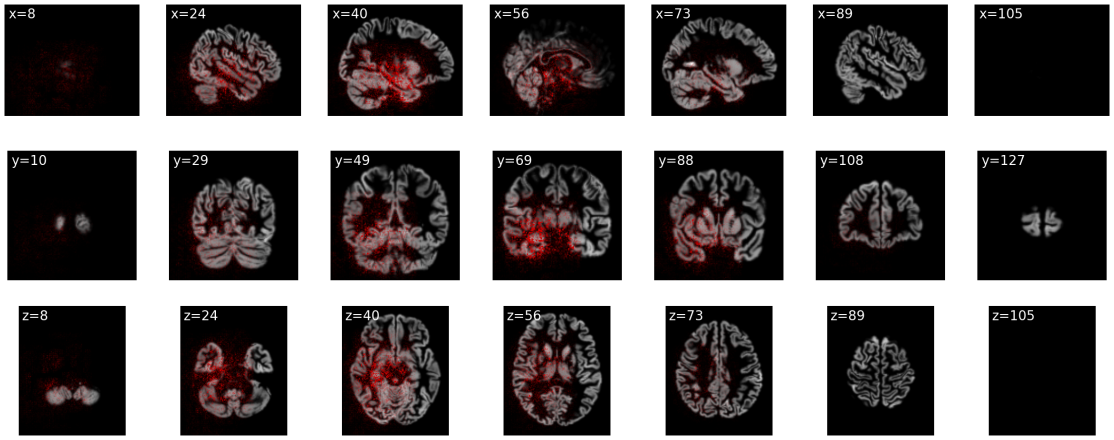
2.2 Sensitivity Analysis

[15]:
```
relevance_map_backprop = interpretation.sensitivity_analysis(net, image_tensor,␣
↪cuda=True, verbose=True)
```

```
c:\AD_Detection_v2\interpretation.py:62: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  output = F.softmax(output)

Image was classified as tensor(0, device='cuda:0') with probability
tensor(0.8311, device='cuda:0')
```

[16]:
```
utils.plot_slices(raw_image, overlay=relevance_map_backprop[0], overlay_vmax=np.
↪percentile(relevance_map_backprop, 99.9), overlay_cmap=utils.
↪alpha_to_red_cmap)
```

```
0.0 1.2856388 0.0 0.0037466600108892884
```



[17]:
```
get_relevance_per_area(relevance_map_backprop[0])[:10]
```

[17]:
```
[('Temporal_Mid', 0.0659991365192679),
 ('Temporal_Inf', 0.04805441105614506),
 ('Fusiform', 0.04778232206385523),
 ('Lingual', 0.03972294268466282),
 ('Temporal_Sup', 0.03599013419454606),
 ('ParaHippocampal', 0.03582749708552674),
 ('Hippocampus', 0.03381022005015622),
 ('Thalamus', 0.033561680108308775),
 ('Cerebelum_8', 0.033085078734686225),
 ('Cerebelum_6', 0.03282955439679061)]
```

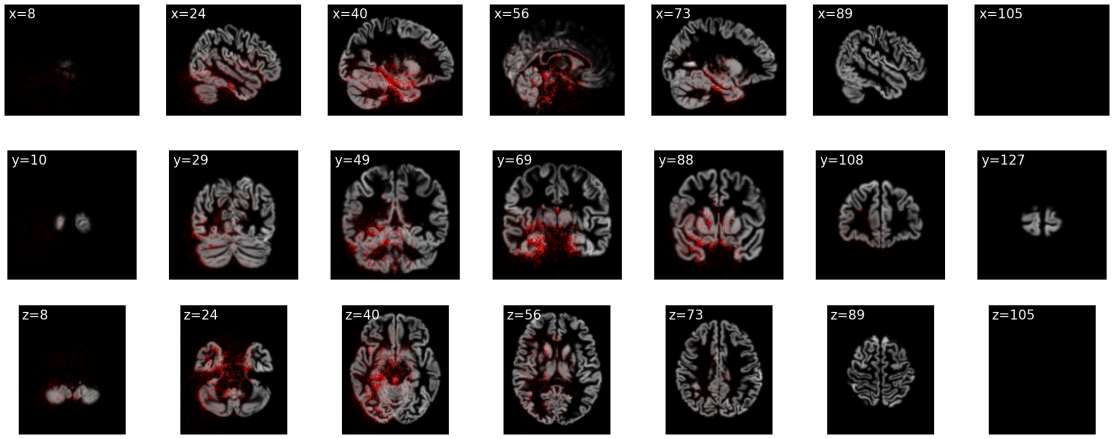
2.3 Guided Backpropagation

[18]:
```
relevance_map_guided = interpretation.guided_backprop(net, image_tensor,␣
↪cuda=True, verbose=True)
```

```
Registered hook for layer: ReLU()
Image was classified as tensor(0, device='cuda:0') with probability
tensor(0.8311, device='cuda:0')
Removing 1 hook(s)
```

[19]:
```
utils.plot_slices(raw_image, overlay=relevance_map_guided[0], overlay_vmax=np.
↪percentile(relevance_map_guided, 99.9))
```

```
0.0 1.2856388 0.0 0.0015516546228900557
```



[20]:
```
get_relevance_per_area(relevance_map_guided[0])[:10]
```

[20]:
```
[('Fusiform', 0.059442254286864143),
 ('ParaHippocampal', 0.05827792686041908),
 ('Temporal_Inf', 0.05314069887610396),
 ('Hippocampus', 0.05075693529506875),
 ('Temporal_Mid', 0.047198856347454854),
 ('Lingual', 0.04398118032990446),
 ('Cerebelum_4_5', 0.03627929486938075),
 ('Cerebelum_8', 0.03504120409466398),
 ('Cerebelum_Crus1', 0.03398235244466915),
 ('Thalamus', 0.03335072957838977)]
```

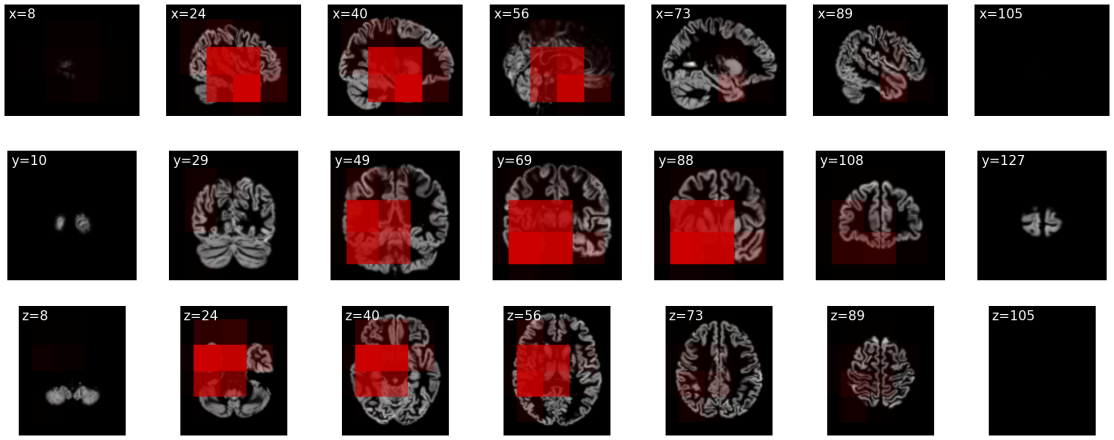
2.4 Occlusion

[21]:
```
relevance_map_occlusion = interpretation.occlusion(net, image_tensor, size=40,␣
↪stride=25, cuda=True, resize=True, verbose=True)
```

```
Image was classified as 0 with probability tensor(0.8311)

c:\AD_Detection_v2\interpretation.py:184: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  output = F.softmax(output)

x:   0%|          | 0/5 [00:00<?, ?it/s]

y:   0%|          | 0/6 [00:00<?, ?it/s]

z:   0%|          | 0/5 [00:00<?, ?it/s]

c:\AD_Detection_v2\interpretation.py:238: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  output = F.softmax(output)
```

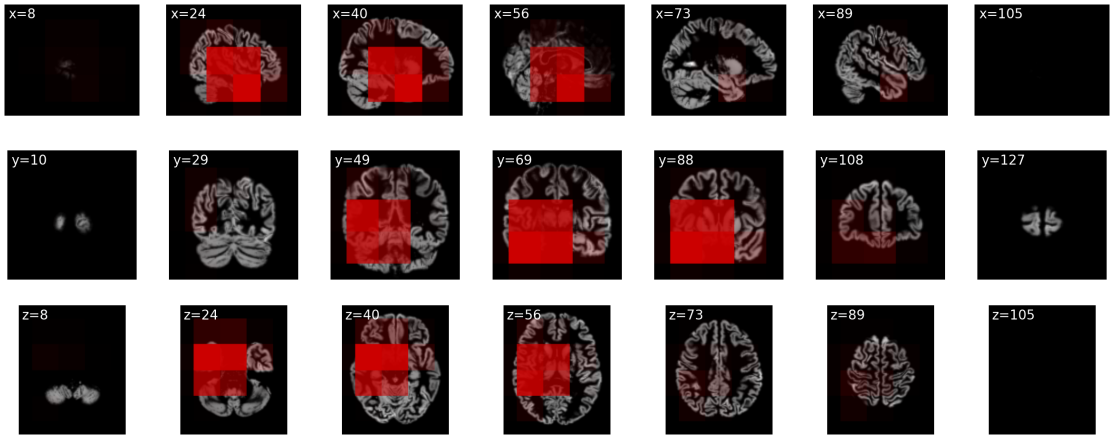
[22]:
```
utils.plot_slices(raw_image, overlay=relevance_map_occlusion,␣
↪overlay_cmap=utils.alpha_to_red_cmap)
```

```
0.0 1.2856388 0.0 0.7370771169662476
```



[23]:
```
utils.plot_slices(raw_image, overlay=relevance_map_occlusion,␣
↪overlay_cmap=utils.alpha_to_red_cmap)
```

```
0.0 1.2856388 0.0 0.7370771169662476
```



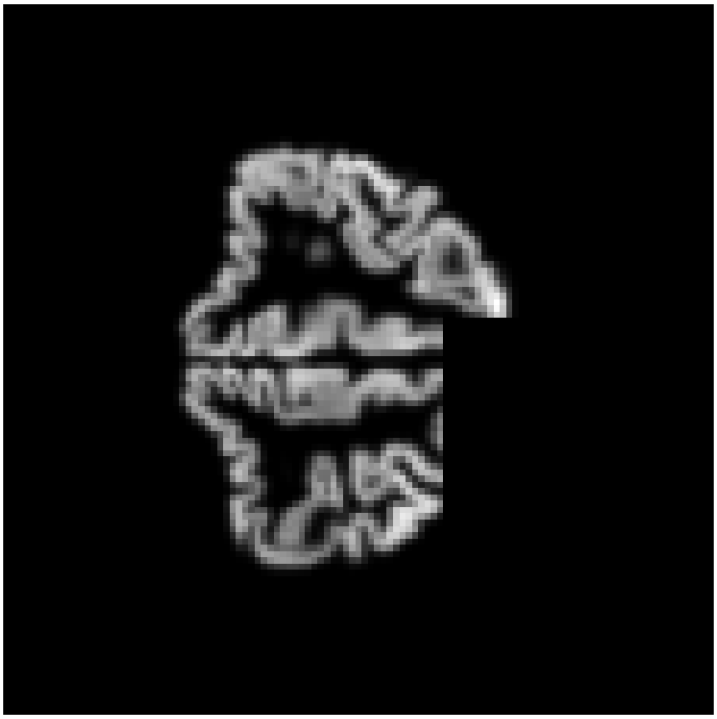
[24]:
```
get_relevance_per_area(relevance_map_occlusion)[:10]
```

[24]:
```
[('Temporal_Mid', 0.08526314912577945),
 ('Temporal_Inf', 0.05968699347135822),
 ('Temporal_Sup', 0.047999872455860605),
 ('Insula', 0.03724190145494688),
 ('Fusiform', 0.035189638481612884),
 ('Cingulum_Mid', 0.03382953382942833),
 ('Postcentral', 0.03345033551473873),
 ('Caudate', 0.03221959983874956),
 ('Temporal_Pole_Sup', 0.031835000157628954),
```

```
('Thalamus', 0.031221903750326817)]
```

[25]:
```
# Plot occlusion patch on image.
occluded_image = raw_image[:, 114, :].copy()
size = 40
occluded_image[50:50+size, 70:70+size] = 0
plt.imshow(occluded_image, cmap='gray')
plt.axis('off')
```

[25]:
```
(-0.5, 112.5, 112.5, -0.5)
```



2.5 Area Occlusion

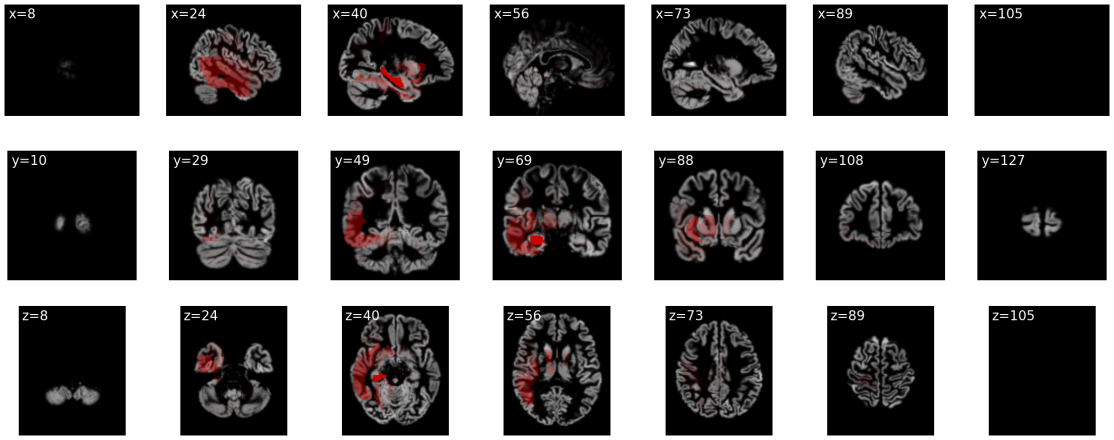
[26]:
```
relevance_map_area_occlusion = interpretation.area_occlusion(net, image_tensor,␣
↪area_masks, cuda=True, verbose=True)
```

```
Image was classified as 0 with probability tensor(0.8311, device='cuda:0')

c:\AD_Detection_v2\interpretation.py:289: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  output = F.softmax(output)

  0%|          | 0/116 [00:00<?, ?it/s]
```

```
c:\AD_Detection_v2\interpretation.py:310: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  output = F.softmax(output)
```

[27]:
```
utils.plot_slices(raw_image, overlay=relevance_map_area_occlusion,␣
↪overlay_cmap=utils.alpha_to_red_cmap)#, overlay_vmin=0, overlay_vmax=1)
```

```
0.0 1.2856388 0.0 0.37365428
```



[28]:
```
get_relevance_per_area(relevance_map_area_occlusion)[:10]
```

[28]:
```
[('Temporal_Mid', 0.20853672653240646),
 ('Temporal_Inf', 0.12808696295874225),
 ('Hippocampus', 0.08742254746367092),
 ('Insula', 0.08365421352881586),
 ('Fusiform', 0.077365693047491),
 ('Temporal_Sup', 0.07451936107328334),
 ('Caudate', 0.04177726297340908),
 ('Postcentral', 0.033649430625157455),
 ('Thalamus', 0.02474329190598219),
 ('Putamen', 0.021284716684372092)]
```

2.6 Grad-CAM

Note: Grad-CAM doesn't work on this data. It's only included here for completeness, but I couldn't figure out if there's a bug in the implementation or if the method is not applicable to this kind of data and/or model.

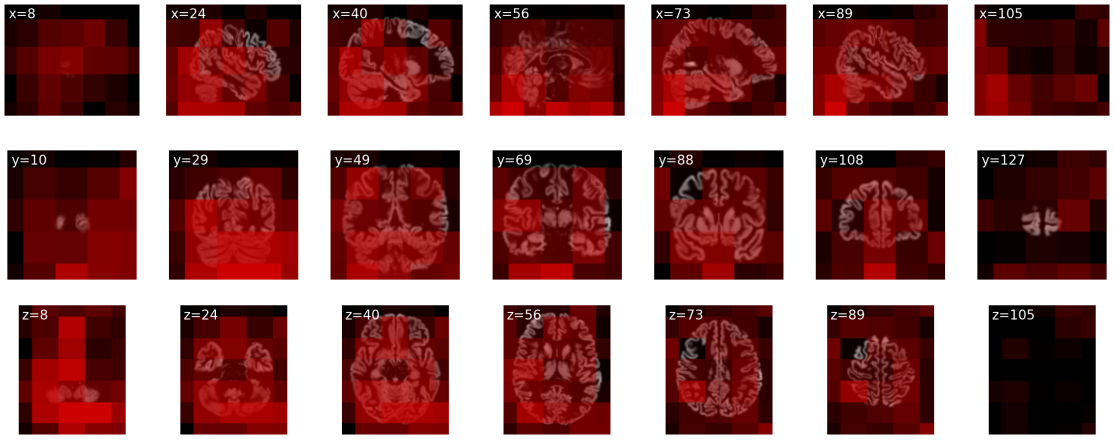
[29]:
```
# TODO: Something is apparently wrong with gradcam, try to fix it. Could also␣
↪simply be because feature maps of last layer are just 4x5x4, so very small.
```

[30]:
```
relevance_map_grad_cam = interpretation.grad_cam(net, image_tensor, cuda=True,␣
↪resize=True, interpolation=0, verbose=True)
```

```
Found last conv layer: Conv3d(32, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1))
Stored feature maps
Image was classified as 0 with probability 0.8310659
tensor([[1., 0.]], device='cuda:0')
Stored gradient
-0.00024387334 2.0185616e-06 0.00018876795
(64, 1, 1, 1)
(64, 5, 7, 5)
(5, 7, 5)
-0.00057951704 0.00060575595 0.0017724105
0.0 0.00063122803 0.0017724105

c:\Users\amana\anaconda3\envs\pyt\Lib\site-
packages\torch\nn\modules\module.py:1640: FutureWarning: Using a non-full
backward hook when the forward contains multiple autograd Nodes is deprecated
and will be removed in future versions. This hook will be missing some
grad_input. Please use register_full_backward_hook to get the documented
behavior.
  self._maybe_warn_non_full_backward_hook(args, result, grad_fn)
c:\AD_Detection_v2\interpretation.py:389: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  output = F.softmax(output)
```

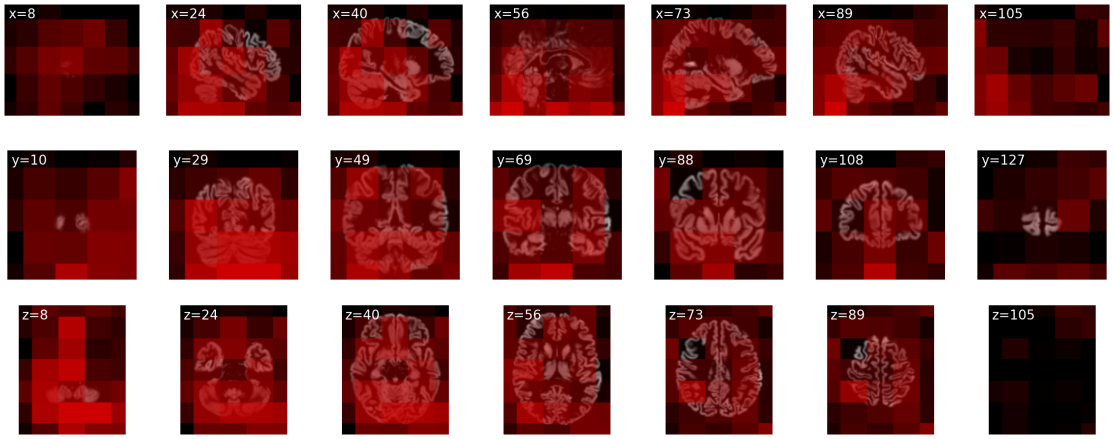
[31]:
```
utils.plot_slices(raw_image, overlay=relevance_map_grad_cam * test_dataset.std␣
↪+ test_dataset.mean)
```

```
0.0 1.2856388 0.0 0.0017724105
```



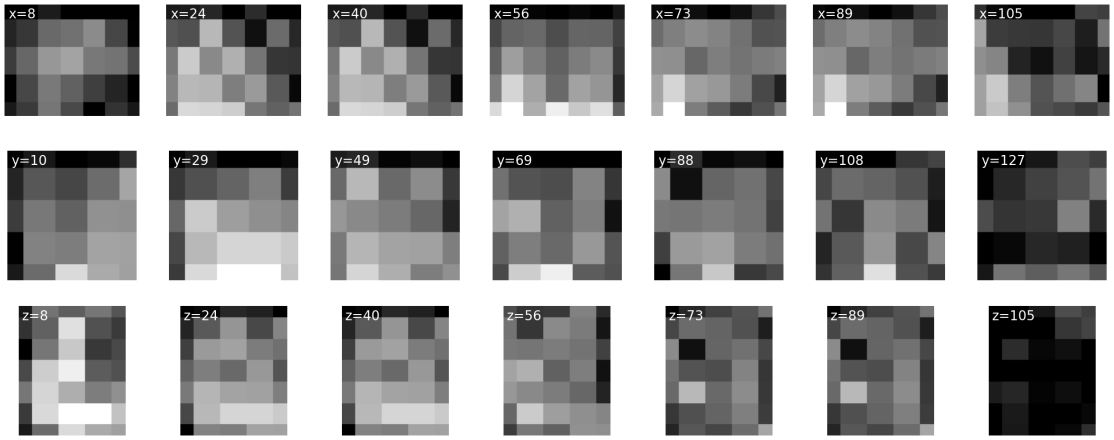
[32]:
```
utils.plot_slices(raw_image, overlay=relevance_map_grad_cam)
```

```
0.0 1.2856388 0.0 0.0017724105
```



[33]:
```
utils.plot_slices(relevance_map_grad_cam)
```

```
0.0 0.0017724105 None None
```



[34]:
```
get_relevance_per_area(relevance_map_grad_cam)[:10]
```

[34]:
```
[('Temporal_Mid', 0.049766948032382255),
 ('Temporal_Inf', 0.040614745219818234),
 ('Cerebelum_Crus1', 0.03916771474785748),
 ('Postcentral', 0.03888811028895307),
 ('Frontal_Mid', 0.03567589211051895),
 ('Occipital_Mid', 0.035644128556878596),
```

```
('Frontal_Sup', 0.03318833404528547),
('Cerebelum_Crus2', 0.031220567389770377),
('Fusiform', 0.031193906472430075),
('Precuneus', 0.031156061205216416)]
```

3 Relevance heatmaps averaged over the dataset

Plot an average relevance map of all Alzheimer and all control patients (in the test set). These are the kind of plots that are included in the paper.

[35]:
```
# If None, average over the entire dataset, otherwise pick a number of samples.
num_samples=None

# The background over which to plot the heatmaps.
bg = test_dataset.mean
```

3.1 Backpropagation

[36]:
```
avg_relevance_map_AD_backprop, avg_relevance_map_NC_backprop,␣
↪avg_relevance_map_all_backprop = interpretation.
↪average_over_dataset(interpretation.sensitivity_analysis, net, test_dataset,␣
↪num_samples=num_samples, seed=0, show_progress=True, cuda=True)
```

```
  0%|          | 0/103 [00:00<?, ?it/s]
```

[37]:
```
utils.plot_slices(bg, overlay=avg_relevance_map_AD_backprop[0], overlay_vmax=np.
↪percentile(avg_relevance_map_AD_backprop, 99.9))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[37], line 1
----> 1␣
↪utils.plot_slices(bg, overlay=avg_relevance_map_AD_backprop[0], overlay_vmax=np.percentile

File c:\AD_Detection_v2\utils.py:110, in plot_slices(struct_arr, num_slices,␣
↪cmap, vmin, vmax, overlay, overlay_cmap, overlay_vmin, overlay_vmax)
     95 """
     96 Plot equally spaced slices of a 3D image (and an overlay) along every␣
↪axis
     97
   (…)
    107     overlay_vmax (float): Same as in matplotlib.imshow. If `None`, take
↪the global maximum of `overlay`.
    108 """
    109 if vmin is None:
--> 110     vmin = struct_arr.min()
```

```
    111 if vmax is None:
    112     vmax = struct_arr.max()

AttributeError: 'int' object has no attribute 'min'
```

[ ]:
```
get_relevance_per_area(avg_relevance_map_AD_backprop)[:10]
```

[ ]:
```
utils.plot_slices(bg, overlay=avg_relevance_map_NC_backprop[0], overlay_vmax=np.
↪percentile(avg_relevance_map_NC_backprop, 99.9))
```

[ ]:
```
get_relevance_per_area(avg_relevance_map_NC_backprop)[:10]
```

3.2 Guided Backpropagation

[ ]:
```
avg_relevance_map_AD_guided, avg_relevance_map_NC_guided,␣
↪avg_relevance_map_all_guided = interpretation.
↪average_over_dataset(interpretation.guided_backprop, net, test_dataset,␣
↪num_samples=num_samples, seed=0, show_progress=True, cuda=True)
```

[ ]:
```
utils.plot_slices(bg, overlay=avg_relevance_map_AD_guided[0], overlay_vmax=np.
↪percentile(avg_relevance_map_AD_guided, 99.9))
```

[ ]:
```
get_relevance_per_area(avg_relevance_map_AD_guided)[:10]
```

[ ]:
```
utils.plot_slices(bg, overlay=avg_relevance_map_NC_guided[0], overlay_vmax=np.
↪percentile(avg_relevance_map_NC_guided, 99.9))
```

[ ]:
```
get_relevance_per_area(avg_relevance_map_NC_guided)[:10]
```

3.3 Occlusion

[ ]:
```
avg_relevance_map_AD_occlusion, avg_relevance_map_NC_occlusion,␣
↪avg_relevance_map_all_occlusion = interpretation.
↪average_over_dataset(interpretation.occlusion, net, test_dataset,␣
↪num_samples=num_samples, show_progress=True, size=40, stride=40, cuda=True)
```

[ ]:
```
utils.plot_slices(bg, overlay=avg_relevance_map_AD_occlusion,␣
↪overlay_cmap=utils.alpha_to_red_cmap)
```

[ ]:
```
get_relevance_per_area(avg_relevance_map_AD_occlusion)[:10]
```

[ ]:
```
utils.plot_slices(bg, overlay=avg_relevance_map_NC_occlusion,␣
↪overlay_cmap=utils.alpha_to_red_cmap)
```

[ ]:
```
get_relevance_per_area(avg_relevance_map_NC_occlusion)[:10]
```

3.4 Area Occlusion

```
[ ]: avg_relevance_map_AD_area_occlusion, avg_relevance_map_NC_area_occlusion,␣
      ↪avg_relevance_map_all_area_occlusion = interpretation.
      ↪average_over_dataset(interpretation.area_occlusion, net, test_dataset,␣
      ↪num_samples=num_samples, seed=0, show_progress=True, cuda=True,␣
      ↪area_masks=area_masks)
```

```
[ ]: utils.plot_slices(bg, overlay=avg_relevance_map_AD_area_occlusion,␣
      ↪overlay_cmap=utils.alpha_to_red_cmap)
```

```
[ ]: get_relevance_per_area(avg_relevance_map_AD_area_occlusion)[:10]
```

```
[ ]: utils.plot_slices(bg, overlay=avg_relevance_map_NC_area_occlusion)
```

```
[ ]: get_relevance_per_area(avg_relevance_map_NC_area_occlusion)[:10]
```

3.5 Grad-CAM

```
[ ]: avg_relevance_map_AD_grad_cam, avg_relevance_map_NC_grad_cam,␣
      ↪avg_relevance_map_all_grad_cam = average_over_dataset(grad_cam, net,␣
      ↪val_dataset, num_samples=10, show_progress=True, resize=True,␣
      ↪interpolation=0, cuda=True)
```

```
[ ]: utils.plot_slices(mask, overlay=avg_relevance_map_AD_grad_cam)
```

```
[ ]: get_relevance_per_area(avg_relevance_map_AD_grad_cam)[:10]
```

```
[ ]: utils.plot_slices(mask, overlay=avg_relevance_map_NC_grad_cam)
```

```
[ ]: get_relevance_per_area(avg_relevance_map_NC_grad_cam)[:10]
```

3.6 Save average heatmaps

```
[ ]: # Save heatmaps to file.
     #np.savez_compressed('output/relevance_maps_final_compressed.npz', **{
     #    'avg_relevance_map_AD_backprop': avg_relevance_map_AD_backprop,
     #    'avg_relevance_map_AD_guided': avg_relevance_map_AD_guided,
     #    'avg_relevance_map_AD_occlusion': avg_relevance_map_AD_occlusion,
     #    'avg_relevance_map_AD_area_occlusion': avg_relevance_map_AD_area_occlusion,
     #    'avg_relevance_map_AD_grad_cam': avg_relevance_map_AD_grad_cam,

     #    'avg_relevance_map_NC_backprop': avg_relevance_map_NC_backprop,
     #    'avg_relevance_map_NC_guided': avg_relevance_map_NC_guided,
     #    'avg_relevance_map_NC_occlusion': avg_relevance_map_NC_occlusion,
     #    'avg_relevance_map_NC_area_occlusion': avg_relevance_map_NC_area_occlusion,
     #    'avg_relevance_map_NC_grad_cam': avg_relevance_map_NC_grad_cam
```

```
#})
```

4 Figures and tables for paper

[ ]:
```
# Load heatmaps from file.
loaded_heatmaps = np.load('output/relevance_maps_final_compressed.npz')
avg_relevance_map_AD_backprop = loaded_heatmaps['avg_relevance_map_AD_backprop']
avg_relevance_map_AD_guided = loaded_heatmaps['avg_relevance_map_AD_guided']
avg_relevance_map_AD_occlusion =␣
 ↪loaded_heatmaps['avg_relevance_map_AD_occlusion']
avg_relevance_map_AD_area_occlusion =␣
 ↪loaded_heatmaps['avg_relevance_map_AD_area_occlusion']
avg_relevance_map_AD_grad_cam = loaded_heatmaps['avg_relevance_map_AD_grad_cam']
avg_relevance_map_NC_backprop = loaded_heatmaps['avg_relevance_map_NC_backprop']
avg_relevance_map_NC_guided = loaded_heatmaps['avg_relevance_map_NC_guided']
avg_relevance_map_NC_occlusion =␣
 ↪loaded_heatmaps['avg_relevance_map_NC_occlusion']
avg_relevance_map_NC_area_occlusion =␣
 ↪loaded_heatmaps['avg_relevance_map_NC_area_occlusion']
avg_relevance_map_NC_grad_cam = loaded_heatmaps['avg_relevance_map_NC_grad_cam']
```

4.1 Figure 1 (Average heatmaps)

[ ]:
```
def plot_column(struct_arr, axes, title, cmap='gray', vmin=None, vmax=None,␣
 ↪overlay=None, overlay_cmap=utils.alpha_to_red_cmap, overlay_vmin=None,␣
 ↪overlay_vmax=None):
    if vmin is None:
        vmin = struct_arr.min()
    if vmax is None:
        vmax = struct_arr.max()
    if overlay_vmin is None and overlay is not None:
        overlay_vmin = overlay.min()
    if overlay_vmax is None and overlay is not None:
        overlay_vmax = overlay.max()
    print(vmin, vmax, overlay_vmin, overlay_vmax)

    offset = 80
    num_slices = len(axes)

    intervals = (np.asarray(struct_arr.shape) - 2 * offset) / (num_slices - 1)

    axis = 1
    axis_label = 'y'

    #for axes_column in zip(titles):
```

```
    for i, ax in enumerate(axes):
        #print(axis_label, 'plotting slice', i_slice)
        i_slice = int(np.round(offset + i * intervals[axis]))

        plt.sca(ax)
        ax.get_xaxis().set_ticks([])
        ax.get_yaxis().set_ticks([])
        #plt.axis('off')
        plt.imshow(sp.ndimage.rotate(np.take(struct_arr, i_slice, axis=axis),␣
↪90), vmin=vmin, vmax=vmax,
                   cmap=cmap, interpolation=None)
        plt.text(0.03, 0.97, '{}'.format(i_slice), color='white',
                 horizontalalignment='left', verticalalignment='top',␣
↪transform=ax.transAxes)

        if overlay is not None:
            plt.imshow(sp.ndimage.rotate(np.take(overlay, i_slice, axis=axis),␣
↪90), cmap=overlay_cmap,
                       vmin=overlay_vmin, vmax=overlay_vmax, interpolation=None)

        if i == 0:
            plt.title(title + '\n')
            #plt.ylabel('Backpropagation', rotation=0, size='large')
            #ax.get_yaxis().set_label_coords(-0.5, 0.5)
```

[ ]:
```
fig, axes = plt.subplots(3, 4, figsize=(11, 7.5))
axes = axes.T

#bg = mask
bg = test_dataset.mean
vmax = 550  # make the background brain a bit lighter

plot_column(bg, axes[0], 'Sensitivity Analysis\n(Backpropagation)',␣
↪overlay=avg_relevance_map_NC_backprop[0], overlay_vmax=np.
↪percentile(avg_relevance_map_NC_backprop, 99.9), vmax=vmax)
plot_column(bg, axes[1], 'Guided\nBackpropagation',␣
↪overlay=avg_relevance_map_NC_guided[0], overlay_vmax=np.
↪percentile(avg_relevance_map_NC_guided, 99.9), vmax=vmax)
plot_column(bg, axes[2], 'Occlusion\n', overlay=avg_relevance_map_NC_occlusion,␣
↪vmax=vmax)
plot_column(bg, axes[3], 'Brain Area\nOcclusion',␣
↪overlay=avg_relevance_map_NC_area_occlusion, vmax=vmax)
#plot_column(bg, axes[4], 'Grad-CAM\n', overlay=avg_relevance_map_AD_grad_cam,␣
↪vmax=vmax)
plt.tight_layout()
```

```
#plt.savefig('heatmaps-nc.png', dpi=300)
```

4.2 Table 1 (Most relevant brain areas)

[ ]:
```
most_relevant_areas_per_method = [
    get_relevance_per_area(avg_relevance_map_AD_backprop)[:4],
    get_relevance_per_area(avg_relevance_map_AD_guided)[:4],
    get_relevance_per_area(avg_relevance_map_AD_occlusion)[:4],
    get_relevance_per_area(avg_relevance_map_AD_area_occlusion)[:4]#,
    #get_relevance_per_area(avg_relevance_map_AD_grad_cam)[:4]
]
```

[ ]:
```
lines = []
for i in range(len(most_relevant_areas_per_method[0])):
    line = ' & '.join(['{} ({:.1f} \\%)'.format(areas[i][0].replace('_', ''),␣
 ↪areas[i][1]*100) for areas in most_relevant_areas_per_method])
    lines.append(line)
print(' \\\\ \n'.join(lines))  # output in latex format
```

4.3 Table 2 (Euclidean distances between heatmaps)

[ ]:
```
relevance_maps_AD = [avg_relevance_map_AD_backprop,␣
 ↪avg_relevance_map_AD_guided, avg_relevance_map_AD_occlusion,␣
 ↪avg_relevance_map_AD_area_occlusion]
relevance_maps_NC = [avg_relevance_map_NC_backprop,␣
 ↪avg_relevance_map_NC_guided, avg_relevance_map_NC_occlusion,␣
 ↪avg_relevance_map_NC_area_occlusion]
names = ['backprop ', 'guided   ', 'occlusion', 'area occl']
```

[ ]:
```
scale_factor = 1e-4

print('Euclidean distance between average heatmaps for AD / NC samples in {}'.
 ↪format(scale_factor))
print()

print('\t\t' + '\t'.join(names))

for a_AD, a_NC, a_name in zip(relevance_maps_AD, relevance_maps_NC, names):
    print(a_name, end='\t')
    for b_AD, b_NC, b_name in zip(relevance_maps_AD, relevance_maps_NC, names):
        print('{:.2f} / {:.2f}'.format(interpretation.heatmap_distance(a_AD,␣
 ↪b_AD) / scale_factor, interpretation.heatmap_distance(a_NC, b_NC) /␣
 ↪scale_factor), end='\t')
    print()
```

[ ]:
```
print('Euclidean distance between AD and NC heatmaps for each method in {}'.
 ↪format(scale_factor))
print()

print('\t'.join(names))
for a, b in zip(relevance_maps_AD, relevance_maps_NC):
    print('{:.2f}'.format(interpretation.heatmap_distance(a, b) /␣
 ↪scale_factor), end='\t\t')
```

5 Export to Animation and MRIcron

[ ]:
```
anim = utils.animate_slices(test_dataset.mean,␣
 ↪overlay=avg_relevance_map_AD_guided[0], overlay_vmax=np.
 ↪percentile(avg_relevance_map_AD_guided, 99.9), axis=2,␣
 ↪reverse_direction=True, interval=70)
plt.close()  # suppress plot output
```

[ ]:
```
plt.rcParams['animation.ffmpeg_path'] = u'/home/johannesr/ffmpeg-3.4.
 ↪1-64bit-static/ffmpeg'
```

[ ]:
```
# Display the animation inline.
from IPython.display import HTML
HTML(anim.to_html5_video())
```

[ ]:
```
anim.save('data/anim-guided.gif')
```

[ ]:
```
# You can also save the relevance map in NIFTI format and load it as an overlay␣
 ↪in MRIcron (https://www.nitrc.org/projects/mricron) later.
#utils.save_nifti('guided_backprop.nii', relevance_map_guided[0])
```