notebook6666fab6fa (1)

December 18, 2024

[1]:
```
!pip install nibabel
!pip install einops
```

```
Requirement already satisfied: nibabel in /opt/conda/lib/python3.10/site-
packages (5.3.2)
Requirement already satisfied: importlib-resources>=5.12 in
/opt/conda/lib/python3.10/site-packages (from nibabel) (6.4.0)
Requirement already satisfied: numpy>=1.22 in /opt/conda/lib/python3.10/site-
packages (from nibabel) (1.26.4)
Requirement already satisfied: packaging>=20 in /opt/conda/lib/python3.10/site-
packages (from nibabel) (21.3)
Requirement already satisfied: typing-extensions>=4.6 in
/opt/conda/lib/python3.10/site-packages (from nibabel) (4.12.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.10/site-packages (from packaging>=20->nibabel) (3.1.2)
Collecting einops
  Downloading einops-0.8.0-py3-none-any.whl.metadata (12 kB)
Downloading einops-0.8.0-py3-none-any.whl (43 kB)
                         43.2/43.2 kB
2.5 MB/s eta 0:00:00
Installing collected packages: einops
Successfully installed einops-0.8.0
```

[2]:
```
import nibabel as nib
from glob import glob
import torch
import torch.nn as nn
import torch.nn.functional as F
from einops import rearrange #pip install einops
from typing import List
import random
import math
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from timm.utils import ModelEmaV3 #pip install timm
from tqdm import tqdm #pip install tqdm
import matplotlib.pyplot as plt #pip install matplotlib
import torch.optim as optim
```

```
import numpy as np
```

[3]:
```
import re

def atoi(text):
    return int(text) if text.isdigit() else text

def natural_keys(text):
    '''
    alist.sort(key=natural_keys) sorts in human order
    http://nedbatchelder.com/blog/200712/human_sorting.html
    (See Toothy's implementation in the comments)
    '''
    return [ atoi(c) for c in re.split(r'(\d+)', text) ]
```

[4]:
```
img_path = glob("/kaggle/input/brats18/dataset/Train/BraTS18_Training_*/
 ↪BraTS18_Training_*_flair.nii")
mask_path = glob("/kaggle/input/brats18/dataset/Train/BraTS18_Training_*/
 ↪BraTS18_Training_*_seg.nii")
# mask_path = glob("/kaggle/input/liver-tumor-segmentation/segmentations/
 ↪segmentation-*.nii")

# mask_path = mask_path[:11]
print("Number of images :", len(mask_path))
```

```
Number of images : 285
```

[5]:
```
img_path.sort(key=natural_keys)
mask_path.sort(key=natural_keys)
```

[ ]:
```

```

[ ]:
```

```

[6]:
```
18*8
```

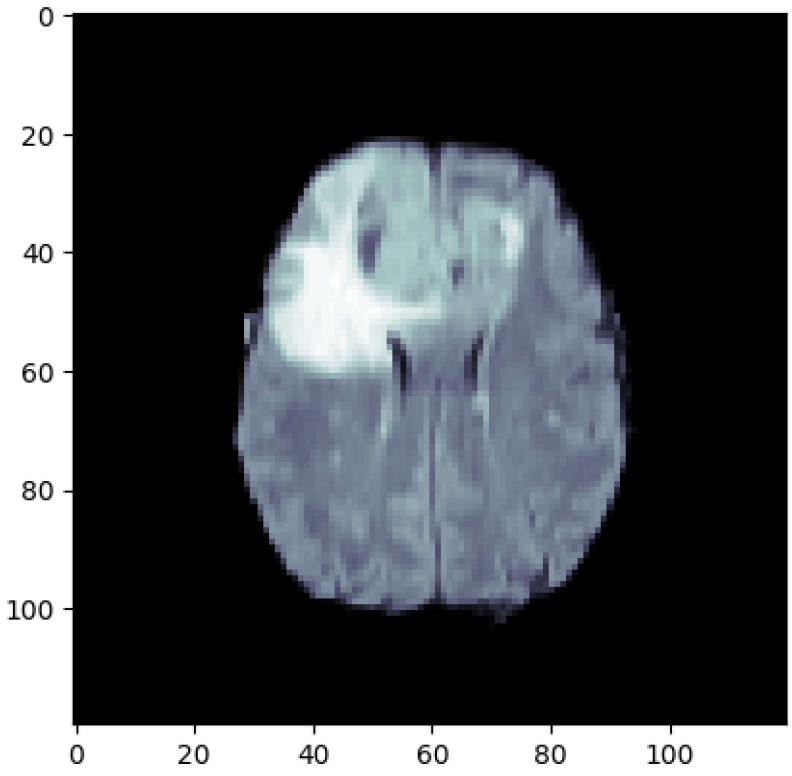
[6]:
```
144
```

[7]:
```
import numpy as np
def downsample_volume(volume, factor=2):
    # Check if the volume dimensions are divisible by the factor
    volume = volume[:,:,4:148]
    assert all(dim % factor == 0 for dim in volume.shape), "Volume dimensions␣
 ↪must be divisible by the factor."

    # Reshape and downsample
    downsampled = volume.reshape(
        volume.shape[0] // factor, factor,
```

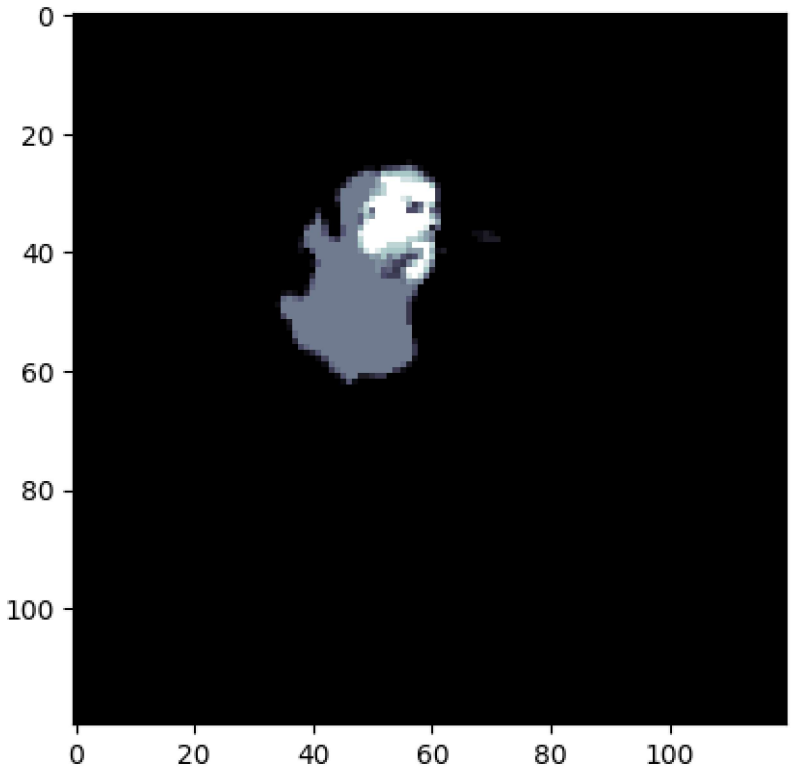
```
        volume.shape[1] // factor, factor,
        volume.shape[2] // factor ,factor
    ).mean(axis=(1, 3, 5))

    downsampled = downsampled[:,:,:]
    return downsampled
```

[ ]:

[8]:
```
img_ex = nib.load(img_path[0]).get_fdata()
img_ex = downsample_volume(img_ex)
# print(img_ex.shape)
plt.imshow(np.rot90(img_ex[:,:,45]/np.max(img_ex[:,:,45]), 3), cmap = 'bone')
plt.show()
print(img_ex.shape)

img_ex = nib.load(mask_path[0]).get_fdata()
img_ex = downsample_volume(img_ex)
# print(img_ex.shape)
plt.imshow(np.rot90(img_ex[:,:,50]/np.max(img_ex[:,:,50]), 3), cmap = 'bone')
plt.show()
print(np.sum(img_ex[:,:,50]/np.max(img_ex[:,:,50])))
```



```
(120, 120, 72)
```




```
379.71875
```

[9]:
```
def slice_to_patch(slice, patch_ratio):

  slice[slice == 1] = 0
  slice[slice == 2] = 1

  patch_list = []

  for x_bin in range(2, len(patch_ratio)):
    for y_bin in range(2, len(patch_ratio)):
      patch = slice[patch_ratio[x_bin-2] : patch_ratio[x_bin],␣
↪patch_ratio[y_bin - 2] : patch_ratio[y_bin]]
      patch = patch.reshape(patch.shape + (1,))
      patch_list.append(patch)

  return np.array(patch_list)
```

[10]:
```
def patch_to_slice(patch, patch_ratio, input_shape, conf_threshold):
```

```
  slice = np.zeros((512, 512, 1))
  row_idx = 0
  col_idx = 0

  for i in range(len(patch)):

    slice[patch_ratio[row_idx]:patch_ratio[row_idx + 2], patch_ratio[col_idx]:
↪patch_ratio[col_idx + 2]][patch[i] > conf_threshold] = 1

    col_idx += 1

    if i != 0 and (i+1) % 15 == 0:
      row_idx += 1
      col_idx = 0

  return slice
```

[11]:
```
total_data = 0
total_patch = []
# img_path = img_path[:125]
for i in range(len(img_path)):

  img_3D = nib.load(img_path[i]).get_fdata()
  patch = torch.tensor(downsample_volume(img_3D), dtype=torch.float32)
  img_3D_mask = nib.load(mask_path[i]).get_fdata()
  patch_mask = torch.tensor(downsample_volume(img_3D_mask), dtype=torch.float32)
  # patch = patch_sampling(img_3D,patch_ratio)
  # print(patch.shape)
  for k in range(patch.shape[2]):
      m = patch[:,:,k]/(torch.max(patch[:,:,k]+0.000001))
      m_mask = patch_mask[:,:,k]/(torch.max(patch_mask[:,:,k]+0.000001))

      # print(m)
      if torch.sum(m_mask)<=0:
          total_patch.append(m)
      # print(m.shape,torch.max(m),torch.min(m))
      # break


  print("======= Step [{0} / {1}] : # of patches = {2} | # of total training␣
↪images = {3} =======".
        format(format(i+1, '>2'), len(img_path),format(len(patch)),␣
↪format(len(total_patch), '>5')))
```

```
======= Step [ 1 / 285] : # of patches = 120 | # of total training images =
31 =======
======= Step [ 2 / 285] : # of patches = 120 | # of total training images =
78 =======
```

```
======= Step [ 3 / 285] : # of patches = 120 | # of total training images =
124 =======
======= Step [ 4 / 285] : # of patches = 120 | # of total training images =
150 =======
======= Step [ 5 / 285] : # of patches = 120 | # of total training images =
204 =======
======= Step [ 6 / 285] : # of patches = 120 | # of total training images =
239 =======
======= Step [ 7 / 285] : # of patches = 120 | # of total training images =
271 =======
======= Step [ 8 / 285] : # of patches = 120 | # of total training images =
312 =======
======= Step [ 9 / 285] : # of patches = 120 | # of total training images =
355 =======
======= Step [10 / 285] : # of patches = 120 | # of total training images =
400 =======
======= Step [11 / 285] : # of patches = 120 | # of total training images =
438 =======
======= Step [12 / 285] : # of patches = 120 | # of total training images =
472 =======
======= Step [13 / 285] : # of patches = 120 | # of total training images =
502 =======
======= Step [14 / 285] : # of patches = 120 | # of total training images =
535 =======
======= Step [15 / 285] : # of patches = 120 | # of total training images =
576 =======
======= Step [16 / 285] : # of patches = 120 | # of total training images =
611 =======
======= Step [17 / 285] : # of patches = 120 | # of total training images =
658 =======
======= Step [18 / 285] : # of patches = 120 | # of total training images =
701 =======
======= Step [19 / 285] : # of patches = 120 | # of total training images =
734 =======
======= Step [20 / 285] : # of patches = 120 | # of total training images =
766 =======
======= Step [21 / 285] : # of patches = 120 | # of total training images =
816 =======
======= Step [22 / 285] : # of patches = 120 | # of total training images =
853 =======
======= Step [23 / 285] : # of patches = 120 | # of total training images =
901 =======
======= Step [24 / 285] : # of patches = 120 | # of total training images =
943 =======
======= Step [25 / 285] : # of patches = 120 | # of total training images =
974 =======
======= Step [26 / 285] : # of patches = 120 | # of total training images =
1019 =======
```

```
======= Step [27 / 285] : # of patches = 120 | # of total training images =
1049 =======
======= Step [28 / 285] : # of patches = 120 | # of total training images =
1101 =======
======= Step [29 / 285] : # of patches = 120 | # of total training images =
1145 =======
======= Step [30 / 285] : # of patches = 120 | # of total training images =
1180 =======
======= Step [31 / 285] : # of patches = 120 | # of total training images =
1228 =======
======= Step [32 / 285] : # of patches = 120 | # of total training images =
1260 =======
======= Step [33 / 285] : # of patches = 120 | # of total training images =
1293 =======
======= Step [34 / 285] : # of patches = 120 | # of total training images =
1345 =======
======= Step [35 / 285] : # of patches = 120 | # of total training images =
1384 =======
======= Step [36 / 285] : # of patches = 120 | # of total training images =
1416 =======
======= Step [37 / 285] : # of patches = 120 | # of total training images =
1459 =======
======= Step [38 / 285] : # of patches = 120 | # of total training images =
1489 =======
======= Step [39 / 285] : # of patches = 120 | # of total training images =
1540 =======
======= Step [40 / 285] : # of patches = 120 | # of total training images =
1582 =======
======= Step [41 / 285] : # of patches = 120 | # of total training images =
1626 =======
======= Step [42 / 285] : # of patches = 120 | # of total training images =
1678 =======
======= Step [43 / 285] : # of patches = 120 | # of total training images =
1729 =======
======= Step [44 / 285] : # of patches = 120 | # of total training images =
1776 =======
======= Step [45 / 285] : # of patches = 120 | # of total training images =
1811 =======
======= Step [46 / 285] : # of patches = 120 | # of total training images =
1853 =======
======= Step [47 / 285] : # of patches = 120 | # of total training images =
1882 =======
======= Step [48 / 285] : # of patches = 120 | # of total training images =
1933 =======
======= Step [49 / 285] : # of patches = 120 | # of total training images =
1965 =======
======= Step [50 / 285] : # of patches = 120 | # of total training images =
2000 =======
```

```
======= Step [51 / 285] : # of patches = 120 | # of total training images =
2032 =======
======= Step [52 / 285] : # of patches = 120 | # of total training images =
2074 =======
======= Step [53 / 285] : # of patches = 120 | # of total training images =
2112 =======
======= Step [54 / 285] : # of patches = 120 | # of total training images =
2156 =======
======= Step [55 / 285] : # of patches = 120 | # of total training images =
2213 =======
======= Step [56 / 285] : # of patches = 120 | # of total training images =
2255 =======
======= Step [57 / 285] : # of patches = 120 | # of total training images =
2283 =======
======= Step [58 / 285] : # of patches = 120 | # of total training images =
2326 =======
======= Step [59 / 285] : # of patches = 120 | # of total training images =
2363 =======
======= Step [60 / 285] : # of patches = 120 | # of total training images =
2401 =======
======= Step [61 / 285] : # of patches = 120 | # of total training images =
2422 =======
======= Step [62 / 285] : # of patches = 120 | # of total training images =
2458 =======
======= Step [63 / 285] : # of patches = 120 | # of total training images =
2486 =======
======= Step [64 / 285] : # of patches = 120 | # of total training images =
2518 =======
======= Step [65 / 285] : # of patches = 120 | # of total training images =
2563 =======
======= Step [66 / 285] : # of patches = 120 | # of total training images =
2595 =======
======= Step [67 / 285] : # of patches = 120 | # of total training images =
2631 =======
======= Step [68 / 285] : # of patches = 120 | # of total training images =
2680 =======
======= Step [69 / 285] : # of patches = 120 | # of total training images =
2724 =======
======= Step [70 / 285] : # of patches = 120 | # of total training images =
2757 =======
======= Step [71 / 285] : # of patches = 120 | # of total training images =
2789 =======
======= Step [72 / 285] : # of patches = 120 | # of total training images =
2836 =======
======= Step [73 / 285] : # of patches = 120 | # of total training images =
2873 =======
======= Step [74 / 285] : # of patches = 120 | # of total training images =
2915 =======
```

```
======= Step [75 / 285] : # of patches = 120 | # of total training images =
2950 =======
======= Step [76 / 285] : # of patches = 120 | # of total training images =
3000 =======
======= Step [77 / 285] : # of patches = 120 | # of total training images =
3045 =======
======= Step [78 / 285] : # of patches = 120 | # of total training images =
3090 =======
======= Step [79 / 285] : # of patches = 120 | # of total training images =
3119 =======
======= Step [80 / 285] : # of patches = 120 | # of total training images =
3161 =======
======= Step [81 / 285] : # of patches = 120 | # of total training images =
3202 =======
======= Step [82 / 285] : # of patches = 120 | # of total training images =
3248 =======
======= Step [83 / 285] : # of patches = 120 | # of total training images =
3279 =======
======= Step [84 / 285] : # of patches = 120 | # of total training images =
3326 =======
======= Step [85 / 285] : # of patches = 120 | # of total training images =
3365 =======
======= Step [86 / 285] : # of patches = 120 | # of total training images =
3406 =======
======= Step [87 / 285] : # of patches = 120 | # of total training images =
3441 =======
======= Step [88 / 285] : # of patches = 120 | # of total training images =
3474 =======
======= Step [89 / 285] : # of patches = 120 | # of total training images =
3498 =======
======= Step [90 / 285] : # of patches = 120 | # of total training images =
3543 =======
======= Step [91 / 285] : # of patches = 120 | # of total training images =
3584 =======
======= Step [92 / 285] : # of patches = 120 | # of total training images =
3618 =======
======= Step [93 / 285] : # of patches = 120 | # of total training images =
3662 =======
======= Step [94 / 285] : # of patches = 120 | # of total training images =
3714 =======
======= Step [95 / 285] : # of patches = 120 | # of total training images =
3751 =======
======= Step [96 / 285] : # of patches = 120 | # of total training images =
3775 =======
======= Step [97 / 285] : # of patches = 120 | # of total training images =
3808 =======
======= Step [98 / 285] : # of patches = 120 | # of total training images =
3834 =======
```

```
======= Step [99 / 285] : # of patches = 120 | # of total training images =
3884 =======
======= Step [100 / 285] : # of patches = 120 | # of total training images =
3919 =======
======= Step [101 / 285] : # of patches = 120 | # of total training images =
3953 =======
======= Step [102 / 285] : # of patches = 120 | # of total training images =
3975 =======
======= Step [103 / 285] : # of patches = 120 | # of total training images =
4018 =======
======= Step [104 / 285] : # of patches = 120 | # of total training images =
4050 =======
======= Step [105 / 285] : # of patches = 120 | # of total training images =
4096 =======
======= Step [106 / 285] : # of patches = 120 | # of total training images =
4131 =======
======= Step [107 / 285] : # of patches = 120 | # of total training images =
4174 =======
======= Step [108 / 285] : # of patches = 120 | # of total training images =
4206 =======
======= Step [109 / 285] : # of patches = 120 | # of total training images =
4237 =======
======= Step [110 / 285] : # of patches = 120 | # of total training images =
4267 =======
======= Step [111 / 285] : # of patches = 120 | # of total training images =
4308 =======
======= Step [112 / 285] : # of patches = 120 | # of total training images =
4355 =======
======= Step [113 / 285] : # of patches = 120 | # of total training images =
4389 =======
======= Step [114 / 285] : # of patches = 120 | # of total training images =
4428 =======
======= Step [115 / 285] : # of patches = 120 | # of total training images =
4462 =======
======= Step [116 / 285] : # of patches = 120 | # of total training images =
4506 =======
======= Step [117 / 285] : # of patches = 120 | # of total training images =
4552 =======
======= Step [118 / 285] : # of patches = 120 | # of total training images =
4596 =======
======= Step [119 / 285] : # of patches = 120 | # of total training images =
4627 =======
======= Step [120 / 285] : # of patches = 120 | # of total training images =
4667 =======
======= Step [121 / 285] : # of patches = 120 | # of total training images =
4704 =======
======= Step [122 / 285] : # of patches = 120 | # of total training images =
4745 =======
```

```
======= Step [123 / 285] : # of patches = 120 | # of total training images =
4786 =======
======= Step [124 / 285] : # of patches = 120 | # of total training images =
4824 =======
======= Step [125 / 285] : # of patches = 120 | # of total training images =
4867 =======
======= Step [126 / 285] : # of patches = 120 | # of total training images =
4894 =======
======= Step [127 / 285] : # of patches = 120 | # of total training images =
4925 =======
======= Step [128 / 285] : # of patches = 120 | # of total training images =
4957 =======
======= Step [129 / 285] : # of patches = 120 | # of total training images =
4995 =======
======= Step [130 / 285] : # of patches = 120 | # of total training images =
5034 =======
======= Step [131 / 285] : # of patches = 120 | # of total training images =
5073 =======
======= Step [132 / 285] : # of patches = 120 | # of total training images =
5116 =======
======= Step [133 / 285] : # of patches = 120 | # of total training images =
5149 =======
======= Step [134 / 285] : # of patches = 120 | # of total training images =
5179 =======
======= Step [135 / 285] : # of patches = 120 | # of total training images =
5219 =======
======= Step [136 / 285] : # of patches = 120 | # of total training images =
5252 =======
======= Step [137 / 285] : # of patches = 120 | # of total training images =
5278 =======
======= Step [138 / 285] : # of patches = 120 | # of total training images =
5323 =======
======= Step [139 / 285] : # of patches = 120 | # of total training images =
5354 =======
======= Step [140 / 285] : # of patches = 120 | # of total training images =
5379 =======
======= Step [141 / 285] : # of patches = 120 | # of total training images =
5415 =======
======= Step [142 / 285] : # of patches = 120 | # of total training images =
5461 =======
======= Step [143 / 285] : # of patches = 120 | # of total training images =
5506 =======
======= Step [144 / 285] : # of patches = 120 | # of total training images =
5549 =======
======= Step [145 / 285] : # of patches = 120 | # of total training images =
5576 =======
======= Step [146 / 285] : # of patches = 120 | # of total training images =
5606 =======
```

```
======= Step [147 / 285] : # of patches = 120 | # of total training images =
5643 =======
======= Step [148 / 285] : # of patches = 120 | # of total training images =
5678 =======
======= Step [149 / 285] : # of patches = 120 | # of total training images =
5710 =======
======= Step [150 / 285] : # of patches = 120 | # of total training images =
5746 =======
======= Step [151 / 285] : # of patches = 120 | # of total training images =
5778 =======
======= Step [152 / 285] : # of patches = 120 | # of total training images =
5816 =======
======= Step [153 / 285] : # of patches = 120 | # of total training images =
5847 =======
======= Step [154 / 285] : # of patches = 120 | # of total training images =
5882 =======
======= Step [155 / 285] : # of patches = 120 | # of total training images =
5935 =======
======= Step [156 / 285] : # of patches = 120 | # of total training images =
5970 =======
======= Step [157 / 285] : # of patches = 120 | # of total training images =
6017 =======
======= Step [158 / 285] : # of patches = 120 | # of total training images =
6043 =======
======= Step [159 / 285] : # of patches = 120 | # of total training images =
6089 =======
======= Step [160 / 285] : # of patches = 120 | # of total training images =
6117 =======
======= Step [161 / 285] : # of patches = 120 | # of total training images =
6159 =======
======= Step [162 / 285] : # of patches = 120 | # of total training images =
6196 =======
======= Step [163 / 285] : # of patches = 120 | # of total training images =
6234 =======
======= Step [164 / 285] : # of patches = 120 | # of total training images =
6254 =======
======= Step [165 / 285] : # of patches = 120 | # of total training images =
6304 =======
======= Step [166 / 285] : # of patches = 120 | # of total training images =
6350 =======
======= Step [167 / 285] : # of patches = 120 | # of total training images =
6380 =======
======= Step [168 / 285] : # of patches = 120 | # of total training images =
6413 =======
======= Step [169 / 285] : # of patches = 120 | # of total training images =
6457 =======
======= Step [170 / 285] : # of patches = 120 | # of total training images =
6503 =======
```

```
======= Step [171 / 285] : # of patches = 120 | # of total training images =
6548 =======
======= Step [172 / 285] : # of patches = 120 | # of total training images =
6582 =======
======= Step [173 / 285] : # of patches = 120 | # of total training images =
6621 =======
======= Step [174 / 285] : # of patches = 120 | # of total training images =
6662 =======
======= Step [175 / 285] : # of patches = 120 | # of total training images =
6695 =======
======= Step [176 / 285] : # of patches = 120 | # of total training images =
6725 =======
======= Step [177 / 285] : # of patches = 120 | # of total training images =
6754 =======
======= Step [178 / 285] : # of patches = 120 | # of total training images =
6789 =======
======= Step [179 / 285] : # of patches = 120 | # of total training images =
6832 =======
======= Step [180 / 285] : # of patches = 120 | # of total training images =
6875 =======
======= Step [181 / 285] : # of patches = 120 | # of total training images =
6929 =======
======= Step [182 / 285] : # of patches = 120 | # of total training images =
6972 =======
======= Step [183 / 285] : # of patches = 120 | # of total training images =
7002 =======
======= Step [184 / 285] : # of patches = 120 | # of total training images =
7035 =======
======= Step [185 / 285] : # of patches = 120 | # of total training images =
7058 =======
======= Step [186 / 285] : # of patches = 120 | # of total training images =
7098 =======
======= Step [187 / 285] : # of patches = 120 | # of total training images =
7139 =======
======= Step [188 / 285] : # of patches = 120 | # of total training images =
7190 =======
======= Step [189 / 285] : # of patches = 120 | # of total training images =
7230 =======
======= Step [190 / 285] : # of patches = 120 | # of total training images =
7269 =======
======= Step [191 / 285] : # of patches = 120 | # of total training images =
7301 =======
======= Step [192 / 285] : # of patches = 120 | # of total training images =
7356 =======
======= Step [193 / 285] : # of patches = 120 | # of total training images =
7392 =======
======= Step [194 / 285] : # of patches = 120 | # of total training images =
7438 =======
```

```
======= Step [195 / 285] : # of patches = 120 | # of total training images =
7473 =======
======= Step [196 / 285] : # of patches = 120 | # of total training images =
7504 =======
======= Step [197 / 285] : # of patches = 120 | # of total training images =
7538 =======
======= Step [198 / 285] : # of patches = 120 | # of total training images =
7562 =======
======= Step [199 / 285] : # of patches = 120 | # of total training images =
7608 =======
======= Step [200 / 285] : # of patches = 120 | # of total training images =
7640 =======
======= Step [201 / 285] : # of patches = 120 | # of total training images =
7682 =======
======= Step [202 / 285] : # of patches = 120 | # of total training images =
7715 =======
======= Step [203 / 285] : # of patches = 120 | # of total training images =
7753 =======
======= Step [204 / 285] : # of patches = 120 | # of total training images =
7782 =======
======= Step [205 / 285] : # of patches = 120 | # of total training images =
7811 =======
======= Step [206 / 285] : # of patches = 120 | # of total training images =
7858 =======
======= Step [207 / 285] : # of patches = 120 | # of total training images =
7901 =======
======= Step [208 / 285] : # of patches = 120 | # of total training images =
7959 =======
======= Step [209 / 285] : # of patches = 120 | # of total training images =
7999 =======
======= Step [210 / 285] : # of patches = 120 | # of total training images =
8034 =======
======= Step [211 / 285] : # of patches = 120 | # of total training images =
8068 =======
======= Step [212 / 285] : # of patches = 120 | # of total training images =
8095 =======
======= Step [213 / 285] : # of patches = 120 | # of total training images =
8124 =======
======= Step [214 / 285] : # of patches = 120 | # of total training images =
8174 =======
======= Step [215 / 285] : # of patches = 120 | # of total training images =
8220 =======
======= Step [216 / 285] : # of patches = 120 | # of total training images =
8268 =======
======= Step [217 / 285] : # of patches = 120 | # of total training images =
8304 =======
======= Step [218 / 285] : # of patches = 120 | # of total training images =
8338 =======
```

```
======= Step [219 / 285] : # of patches = 120 | # of total training images =
8368 =======
======= Step [220 / 285] : # of patches = 120 | # of total training images =
8403 =======
======= Step [221 / 285] : # of patches = 120 | # of total training images =
8443 =======
======= Step [222 / 285] : # of patches = 120 | # of total training images =
8487 =======
======= Step [223 / 285] : # of patches = 120 | # of total training images =
8518 =======
======= Step [224 / 285] : # of patches = 120 | # of total training images =
8549 =======
======= Step [225 / 285] : # of patches = 120 | # of total training images =
8579 =======
======= Step [226 / 285] : # of patches = 120 | # of total training images =
8612 =======
======= Step [227 / 285] : # of patches = 120 | # of total training images =
8659 =======
======= Step [228 / 285] : # of patches = 120 | # of total training images =
8698 =======
======= Step [229 / 285] : # of patches = 120 | # of total training images =
8735 =======
======= Step [230 / 285] : # of patches = 120 | # of total training images =
8778 =======
======= Step [231 / 285] : # of patches = 120 | # of total training images =
8802 =======
======= Step [232 / 285] : # of patches = 120 | # of total training images =
8836 =======
======= Step [233 / 285] : # of patches = 120 | # of total training images =
8880 =======
======= Step [234 / 285] : # of patches = 120 | # of total training images =
8913 =======
======= Step [235 / 285] : # of patches = 120 | # of total training images =
8940 =======
======= Step [236 / 285] : # of patches = 120 | # of total training images =
8974 =======
======= Step [237 / 285] : # of patches = 120 | # of total training images =
9007 =======
======= Step [238 / 285] : # of patches = 120 | # of total training images =
9053 =======
======= Step [239 / 285] : # of patches = 120 | # of total training images =
9090 =======
======= Step [240 / 285] : # of patches = 120 | # of total training images =
9129 =======
======= Step [241 / 285] : # of patches = 120 | # of total training images =
9170 =======
======= Step [242 / 285] : # of patches = 120 | # of total training images =
9209 =======
```

```
======= Step [243 / 285] : # of patches = 120 | # of total training images =
9246 =======
======= Step [244 / 285] : # of patches = 120 | # of total training images =
9295 =======
======= Step [245 / 285] : # of patches = 120 | # of total training images =
9327 =======
======= Step [246 / 285] : # of patches = 120 | # of total training images =
9370 =======
======= Step [247 / 285] : # of patches = 120 | # of total training images =
9423 =======
======= Step [248 / 285] : # of patches = 120 | # of total training images =
9467 =======
======= Step [249 / 285] : # of patches = 120 | # of total training images =
9498 =======
======= Step [250 / 285] : # of patches = 120 | # of total training images =
9542 =======
======= Step [251 / 285] : # of patches = 120 | # of total training images =
9591 =======
======= Step [252 / 285] : # of patches = 120 | # of total training images =
9622 =======
======= Step [253 / 285] : # of patches = 120 | # of total training images =
9653 =======
======= Step [254 / 285] : # of patches = 120 | # of total training images =
9682 =======
======= Step [255 / 285] : # of patches = 120 | # of total training images =
9708 =======
======= Step [256 / 285] : # of patches = 120 | # of total training images =
9752 =======
======= Step [257 / 285] : # of patches = 120 | # of total training images =
9797 =======
======= Step [258 / 285] : # of patches = 120 | # of total training images =
9833 =======
======= Step [259 / 285] : # of patches = 120 | # of total training images =
9883 =======
======= Step [260 / 285] : # of patches = 120 | # of total training images =
9920 =======
======= Step [261 / 285] : # of patches = 120 | # of total training images =
9955 =======
======= Step [262 / 285] : # of patches = 120 | # of total training images =
10002 =======
======= Step [263 / 285] : # of patches = 120 | # of total training images =
10043 =======
======= Step [264 / 285] : # of patches = 120 | # of total training images =
10079 =======
======= Step [265 / 285] : # of patches = 120 | # of total training images =
10110 =======
======= Step [266 / 285] : # of patches = 120 | # of total training images =
10146 =======
```

```
======= Step [267 / 285] : # of patches = 120 | # of total training images =
10179 =======
======= Step [268 / 285] : # of patches = 120 | # of total training images =
10209 =======
======= Step [269 / 285] : # of patches = 120 | # of total training images =
10258 =======
======= Step [270 / 285] : # of patches = 120 | # of total training images =
10299 =======
======= Step [271 / 285] : # of patches = 120 | # of total training images =
10340 =======
======= Step [272 / 285] : # of patches = 120 | # of total training images =
10387 =======
======= Step [273 / 285] : # of patches = 120 | # of total training images =
10426 =======
======= Step [274 / 285] : # of patches = 120 | # of total training images =
10455 =======
======= Step [275 / 285] : # of patches = 120 | # of total training images =
10488 =======
======= Step [276 / 285] : # of patches = 120 | # of total training images =
10544 =======
======= Step [277 / 285] : # of patches = 120 | # of total training images =
10586 =======
======= Step [278 / 285] : # of patches = 120 | # of total training images =
10632 =======
======= Step [279 / 285] : # of patches = 120 | # of total training images =
10660 =======
======= Step [280 / 285] : # of patches = 120 | # of total training images =
10709 =======
======= Step [281 / 285] : # of patches = 120 | # of total training images =
10758 =======
======= Step [282 / 285] : # of patches = 120 | # of total training images =
10795 =======
======= Step [283 / 285] : # of patches = 120 | # of total training images =
10839 =======
======= Step [284 / 285] : # of patches = 120 | # of total training images =
10868 =======
======= Step [285 / 285] : # of patches = 120 | # of total training images =
10905 =======
```

[12]:
```
20520//64*64
```

[12]: 20480

[13]:
```
total_patch =total_patch[:5000]
```

[14]:
```
len(total_patch)
```

[14]: 5000

[15]:
```
12825//32
```

[15]: 400

[16]:
```
# Imports


class SinusoidalEmbeddings(nn.Module):
    def __init__(self, time_steps:int, embed_dim: int):
        super().__init__()
        position = torch.arange(time_steps).unsqueeze(1).float()  #creat␣
 ↪position value
        div = torch.exp(torch.arange(0, embed_dim, 2).float() * -(math.
 ↪log(10000.0) / embed_dim))
        embeddings = torch.zeros(time_steps, embed_dim, requires_grad=False)
        embeddings[:, 0::2] = torch.sin(position * div)
        embeddings[:, 1::2] = torch.cos(position * div)
        self.embeddings = embeddings

    def forward(self, x, t):
        embeds = self.embeddings[t].to(x.device)

        return embeds[:, :, None, None]
```

[17]:
```
class ResBlock(nn.Module):
    def __init__(self, C: int, num_groups: int, dropout_prob: float):
        super().__init__()
        self.relu = nn.ReLU(inplace=True)
        self.gnorm1 = nn.GroupNorm(num_groups=num_groups, num_channels=C)
        self.gnorm2 = nn.GroupNorm(num_groups=num_groups, num_channels=C)
        self.conv1 = nn.Conv2d(C, C, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(C, C, kernel_size=3, padding=1)
        self.dropout = nn.Dropout(p=dropout_prob, inplace=True)

    def forward(self, x, embeddings):
        x = x + embeddings[:, :x.shape[1], :, :]
        r = self.conv1(self.relu(self.gnorm1(x)))
        r = self.dropout(r)
        r = self.conv2(self.relu(self.gnorm2(r)))
        return r + x
```

[18]:
```
class Attention(nn.Module):
    def __init__(self, C: int, num_heads:int , dropout_prob: float):
        super().__init__()
        self.proj1 = nn.Linear(C, C*3)
        self.proj2 = nn.Linear(C, C)
        self.num_heads = num_heads
```

```
        self.dropout_prob = dropout_prob

    def forward(self, x):
        h, w = x.shape[2:]
        x = rearrange(x, 'b c h w -> b (h w) c')
        x = self.proj1(x)
        x = rearrange(x, 'b L (C H K) -> K b H L C', K=3, H=self.num_heads)
        q,k,v = x[0], x[1], x[2]
        x = F.scaled_dot_product_attention(q,k,v, is_causal=False,␣
↪dropout_p=self.dropout_prob)
        x = rearrange(x, 'b H (h w) C -> b h w (C H)', h=h, w=w)
        x = self.proj2(x)
        return rearrange(x, 'b h w C -> b C h w')
```

[19]:
```
class UnetLayer(nn.Module):
    def __init__(self,
                 upscale: bool,
                 attention: bool,
                 num_groups: int,
                 dropout_prob: float,
                 num_heads: int,
                 C: int):
        super().__init__()
        self.ResBlock1 = ResBlock(C=C, num_groups=num_groups,␣
↪dropout_prob=dropout_prob)
        self.ResBlock2 = ResBlock(C=C, num_groups=num_groups,␣
↪dropout_prob=dropout_prob)
        if upscale:
            self.conv = nn.ConvTranspose2d(C, C//2, kernel_size=4, stride=2,␣
↪padding=1)
        else:
            self.conv = nn.Conv2d(C, C*2, kernel_size=3, stride=2, padding=1)
        if attention:
            self.attention_layer = Attention(C, num_heads=num_heads,␣
↪dropout_prob=dropout_prob)

    def forward(self, x, embeddings):
        x = self.ResBlock1(x, embeddings)
        if hasattr(self, 'attention_layer'):
            x = self.attention_layer(x)
        x = self.ResBlock2(x, embeddings)
        return self.conv(x), x
```

[20]:
```
class UNET(nn.Module):
    def __init__(self,
                 Channels: List = [64, 128, 256,512,512, 384],
                 # Channels: List = [64,128,256,512,1024,1024,768, 512],
```

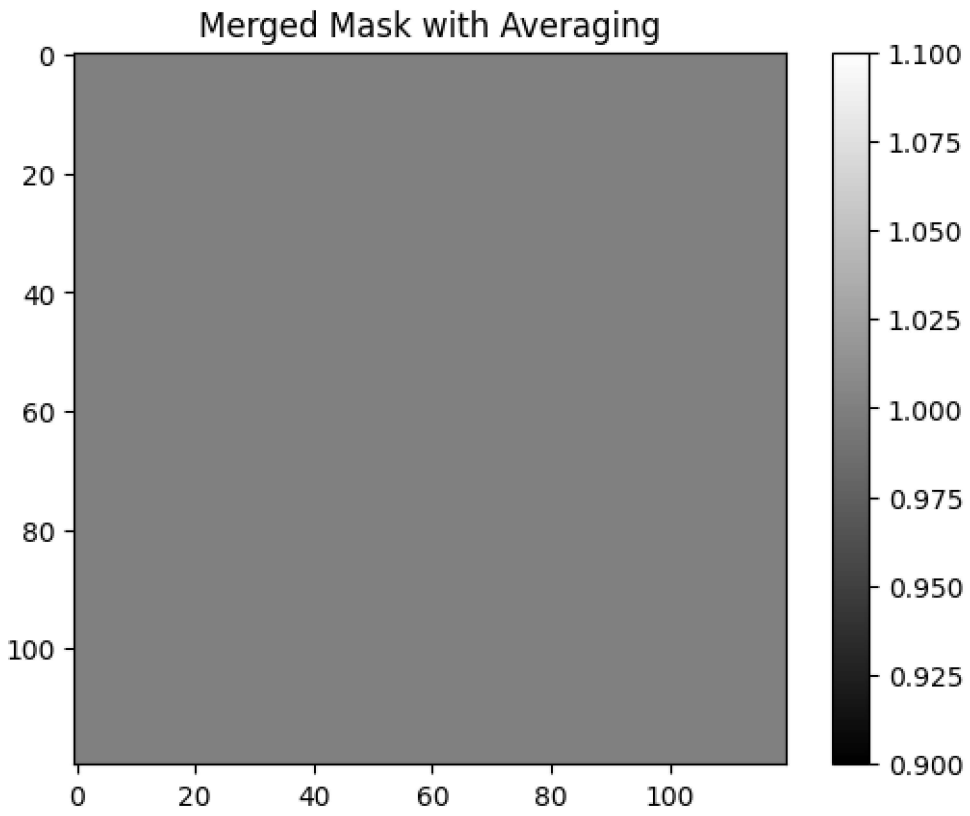
```
            Attentions: List = [False, True, False, False, False, True],
            Upscales: List = [False, False, False, True, True, True],
            num_groups: int = 32,
            dropout_prob: float = 0.1,
            num_heads: int = 8,
            input_channels: int = 1,
            output_channels: int = 1,
            time_steps: int = 1000):
        super().__init__()
        self.num_layers = len(Channels)
        self.shallow_conv = nn.Conv2d(input_channels, Channels[0],␣
↪kernel_size=3, padding=1)
        out_channels = (Channels[-1]//2)+Channels[0]
        self.late_conv = nn.Conv2d(out_channels, out_channels//2,␣
↪kernel_size=3, padding=1)
        self.output_conv = nn.Conv2d(out_channels//2, output_channels,␣
↪kernel_size=1)
        self.relu = nn.ReLU(inplace=True)
        self.embeddings = SinusoidalEmbeddings(time_steps=time_steps,␣
↪embed_dim=max(Channels))
        for i in range(self.num_layers):
            layer = UnetLayer(
                upscale=Upscales[i],
                attention=Attentions[i],
                num_groups=num_groups,
                dropout_prob=dropout_prob,
                C=Channels[i],
                num_heads=num_heads
            )
            setattr(self, f'Layer{i+1}', layer)

    def forward(self, x, t):

        x = self.shallow_conv(x)
        residuals = []
        for i in range(self.num_layers//2):
            layer = getattr(self, f'Layer{i+1}')
            embeddings = self.embeddings(x, t)
            x, r = layer(x, embeddings)
            residuals.append(r)
        for i in range(self.num_layers//2, self.num_layers):
            layer = getattr(self, f'Layer{i+1}')
            x = torch.concat((layer(x, embeddings)[0], residuals[self.
↪num_layers-i-1]), dim=1)
        return self.output_conv(self.relu(self.late_conv(x)))
```

[21]:
```
class DDPM_Scheduler(nn.Module):
    def __init__(self, num_time_steps: int=1000):
        super().__init__()
        self.beta = torch.linspace(1e-4, 0.02, num_time_steps,␣
 ↪requires_grad=False)
        alpha = 1 - self.beta
        self.alpha = torch.cumprod(alpha, dim=0).requires_grad_(False)

    def forward(self, t):
        return self.beta[t], self.alpha[t]
```

[22]:
```
def set_seed(seed: int = 42):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    np.random.seed(seed)
    random.seed(seed)
```

[23]:
```
base_mask = torch.zeros((2, 1, 120, 120), requires_grad=False)
device = 'cuda'
# Clone the base mask to create four different masks
mask1 = base_mask.clone().to(device)
# mask2 = base_mask.clone().to(device)
# mask3 = base_mask.clone().to(device)
# mask4 = base_mask.clone().to(device)

# Modify specific regions in each mask
# mask1[:,:,0:60, 0:60] = 1       # Top-left region
# mask2[:,:,0:60, 60:120] = 1     # Top-right region
# mask3[:,:,60:120, 0:60] = 1     # Bottom-left region
# mask4[:,:,60:120, 60:120] = 1   # Bottom-right region
# plt.imshow(mask2.squeeze(0).squeeze(0),cmap='gray')
mask = []
patch = [0,30,60,90,120]
for i in range(len(patch)-2):
    for j in range(len(patch)-2):
        mask1 = base_mask.clone().to(device)
        mask1[:,:,patch[i]:patch[i+2], patch[j]:patch[j+2]] = 1
        new = mask1
        mask.append(new)

import torch
import matplotlib.pyplot as plt

# Initialize base mask and device
base_mask = torch.zeros((1, 1, 120, 120), requires_grad=False)
```

```
device = 'cuda'

# Patch sizes and positions
patch = [0, 30, 60, 90, 120]
mask = []

# Generate masks for patches
for i in range(len(patch) - 2):
    for j in range(len(patch) - 2):
        mask1 = base_mask.clone().to(device)
        mask1[:, :, patch[i]:patch[i+2], patch[j]:patch[j+2]] = 1
        mask.append(mask1)

# Initialize accumulators for summing masks and counting overlaps
merged_mask_sum = torch.zeros_like(base_mask).to(device)  # Sum of all masks
merged_mask_count = torch.zeros_like(base_mask).to(device)  # Count␣
 ↪contributions

# Add each mask to the accumulators
for m in mask:
    merged_mask_sum += m
    merged_mask_count += (m > 0).float()  # Increment count where mask > 0

# Avoid division by zero
merged_mask_count = torch.where(merged_mask_count == 0, torch.tensor(1.0).
 ↪to(device), merged_mask_count)

# Compute the averaged mask
merged_mask_avg = merged_mask_sum / merged_mask_count

# Visualize the averaged mask (for first channel, first batch)
plt.imshow(merged_mask_avg[0, 0].cpu().detach().numpy(), cmap='gray')
plt.title("Merged Mask with Averaging")
plt.colorbar()
plt.show()
```



[24]:
```
def train(batch_size: int=2,
          num_time_steps: int=1000,
          num_epochs: int=15,
          seed: int=-1,
          ema_decay: float=0.9999,
          lr=1e-5,
          checkpoint_path: str=None):
    set_seed(random.randint(0, 2**32-1)) if seed == -1 else set_seed(seed)

    # train_dataset = datasets.MNIST(root='./data', train=True,␣
↪download=True,transform=transforms.ToTensor())
    # train_dataset =TensorDataset(torch.tensor(total_patch).float())
    # train_loader = DataLoader(train_dataset, batch_size=batch_size,␣
↪shuffle=True, drop_last=True, num_workers=4)
    # train_dataset = datasets.MNIST(root='./data', train=True,␣
↪download=True,transform=transforms.ToTensor())
    train_loader = DataLoader(total_patch, batch_size=2, shuffle=True,␣
↪drop_last=True, num_workers=4)
    scheduler = DDPM_Scheduler(num_time_steps=num_time_steps)
    model = UNET().cuda()
```

```
    # print(model)
    optimizer = optim.Adam(model.parameters(), lr=lr)
    ema = ModelEmaV3(model, decay=ema_decay)
    if checkpoint_path is not None:
        checkpoint = torch.load(checkpoint_path)
        model.load_state_dict(checkpoint['weights'])
        ema.load_state_dict(checkpoint['ema'])
        optimizer.load_state_dict(checkpoint['optimizer'])
    criterion = nn.MSELoss(reduction='mean')

    for i in range(num_epochs):
        total_loss = 0
        for index,x in enumerate(tqdm( train_loader, desc=f"Epoch {i+1}/
↪{num_epochs}")):


            x = x.unsqueeze(1)
            # print(x.shape)
            x = x.cuda()
            # print(x.shape,torch.max(x),torch.min(x))
            # break
            # x = F.pad(x, (1,1,1,1))
            # print(x.shape,torch.max(x),torch.min(x))
            # break
            t = torch.randint(0,num_time_steps,(batch_size,))
            e = torch.randn_like(x, requires_grad=False)
            a = scheduler.alpha[t].view(batch_size,1,1,1).cuda()
            x_out = (torch.sqrt(a)*x) + (torch.sqrt(1-a)*e)
            for k in range(len(mask)):
                x_out_k = x*(1-mask[k]) + x_out*mask[k]
                # print(x.shape,mask[i].shape,x_out.shape)
                output = model(x_out_k, t)
                optimizer.zero_grad()
                loss = criterion(output*mask[k], x*mask[k])
                total_loss += loss.item()
                loss.backward()
                optimizer.step()
                ema.update(model)
        print(f'Epoch {i+1} | Loss {total_loss/batch_size/4:.5f}')

        checkpoint = {
            'weights': model.state_dict(),
            'optimizer': optimizer.state_dict(),
            'ema': ema.state_dict()
        }
        torch.save(checkpoint, '/kaggle/working/ddpm')
```

[25]:
```
# import torch
# import numpy as np
# from einops import rearrange
# from typing import List
# import matplotlib.pyplot as plt

# # Display function remains unchanged
# def display_reverse(images: List):
#     fig, axes = plt.subplots(1, 10, figsize=(10, 1))
#     for i, ax in enumerate(axes.flat):
#         x = images[i].squeeze(0)
#         x = rearrange(x, 'c h w -> h w c')
#         x = x.numpy()
#         ax.imshow(x)
#         ax.axis('off')
#     plt.show()

# # Patch extraction function
# def extract_patches(image, patch_size):
#     patches = []
#     h, w = image.shape[-2:]
#     for i in range(0, h, patch_size[0]):
#         for j in range(0, w, patch_size[1]):
#             patch = image[:, :, i:i+patch_size[0], j:j+patch_size[1]]
#             patches.append(patch)
#     return patches

# # Patch reconstruction function
# def reconstruct_image(patches, image_shape, patch_size):
#     reconstructed = torch.zeros(image_shape)
#     h, w = image_shape[-2:]
#     idx = 0
#     for i in range(0, h, patch_size[0]):
#         for j in range(0, w, patch_size[1]):
#             reconstructed[:, :, i:i+patch_size[0], j:j+patch_size[1]] =␣
 ↪patches[idx]
#             idx += 1
#     return reconstructed

# # Updated inference function
# def inference_whole_image(checkpoint_path: str = None,
#                           num_time_steps: int = 1000,
#                           ema_decay: float = 0.9999,
#                           image_size=(240, 240),
#                           patch_size=(24, 24)):
#     checkpoint = torch.load(checkpoint_path)
#     model = UNET().cuda()
```

```
#     model.load_state_dict(checkpoint['weights'])
#     ema = ModelEmaV3(model, decay=ema_decay)
#     ema.load_state_dict(checkpoint['ema'])
#     scheduler = DDPM_Scheduler(num_time_steps=num_time_steps)
#     times = [0, 15, 50, 100, 200, 300, 400, 550, 700, 999]
#     final_images = []

#     with torch.no_grad():
#         model = ema.module.eval()

#         # Generate noise for the entire image
#         z_whole = torch.randn(1, 1, *image_size)
#         patches = extract_patches(z_whole, patch_size)
#         denoised_patches = []

#         for patch in patches:
#             images = []
#             z = patch.clone()
#             for t in reversed(range(1, num_time_steps)):
#                 t = [t]
#                 temp = (scheduler.beta[t] / ((torch.sqrt(1 - scheduler.
 ↪alpha[t])) * (torch.sqrt(1 - scheduler.beta[t]))))
#                 z = (1 / (torch.sqrt(1 - scheduler.beta[t]))) * z - (temp *␣
 ↪model(z.cuda(), t).cpu())
#                 if t[0] in times:
#                     images.append(z)
#                 e = torch.randn_like(z)
#                 z = z + (e * torch.sqrt(scheduler.beta[t]))

#             temp = scheduler.beta[0] / ((torch.sqrt(1 - scheduler.alpha[0]))␣
 ↪* (torch.sqrt(1 - scheduler.beta[0])))
#             x = (1 / (torch.sqrt(1 - scheduler.beta[0]))) * z - (temp *␣
 ↪model(z.cuda(), [0]).cpu())
#             denoised_patches.append(x)

#         # Reconstruct the whole image from patches
#         final_image = reconstruct_image(denoised_patches, (1, 1,␣
 ↪*image_size), patch_size)

#         # Visualize and return
#         x = rearrange(final_image.squeeze(0), 'c h w -> h w c').detach()
#         x = x.numpy()
#         plt.imshow(x)
#         plt.show()

#         return final_image
```

[ ]:

[26]:
```
train(checkpoint_path=None, lr=1e-5, num_epochs=6)
```

```
Epoch 1/6: 100%|      | 2500/2500 [1:20:42<00:00,  1.94s/it]
Epoch 1 | Loss 22.14016
Epoch 2/6: 100%|      | 2500/2500 [1:20:52<00:00,  1.94s/it]
Epoch 2 | Loss 10.50447
Epoch 3/6: 100%|      | 2500/2500 [1:21:11<00:00,  1.95s/it]
Epoch 3 | Loss 7.65819
Epoch 4/6: 100%|      | 2500/2500 [1:21:29<00:00,  1.96s/it]
Epoch 4 | Loss 6.55113
Epoch 5/6: 100%|      | 2500/2500 [1:21:40<00:00,  1.96s/it]
Epoch 5 | Loss 6.06865
Epoch 6/6: 100%|      | 2500/2500 [1:21:49<00:00,  1.96s/it]
Epoch 6 | Loss 5.81496
```

[27]:
```
base_mask = torch.zeros((1, 1, 120, 120), requires_grad=False)
device = 'cpu'

# Patch sizes and positions
patch = [0, 30, 60, 90, 120]
mask = []

# Generate masks for patches
for i in range(len(patch) - 2):
    for j in range(len(patch) - 2):
        mask1 = base_mask.clone().to(device)
        mask1[:, :, patch[i]:patch[i+2], patch[j]:patch[j+2]] = 1
        mask.append(mask1)

# Initialize accumulators for summing masks and counting overlaps
def maskk(base_mask,mask):
    device = 'cpu'
    merged_mask_sum = torch.zeros_like(base_mask).to(device)  # Sum of all masks
    merged_mask_count = torch.zeros_like(base_mask).to(device)  # Count␣
 ↪contributions

    # Add each mask to the accumulators
    for m in mask:
        merged_mask_sum += m
        merged_mask_count += (m > 0).float()  # Increment count where mask > 0
```

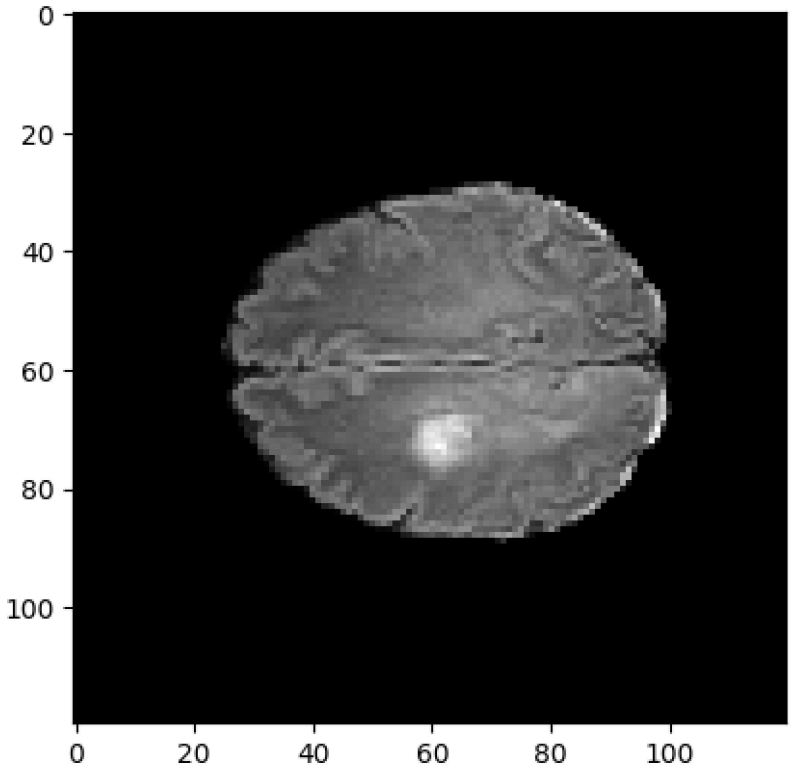
```
    # Avoid division by zero
    merged_mask_count = torch.where(merged_mask_count == 0, torch.tensor(1.0).
 ↪to(device), merged_mask_count)

    # Compute the averaged mask
    merged_mask_avg = merged_mask_sum / merged_mask_count

    # Visualize the averaged mask (for first channel, first batch)
    return merged_mask_avg.cpu()[0,0].detach().numpy()
```

[28]:
```
# def display_reverse(images):
    # fig, axes = plt.subplots(1, 1, figsize=(10,10))
    # for i, ax in enumerate(axes.flat):
    #     x = images[i].squeeze(0)
    #     print(x.shape)
    #     x = rearrange(x, 'c h w -> h w c')
    #     x = x.numpy()
    #     ax.imshow(x,cmap= 'gray')
    #     ax.axis('off')
    # plt.show()

def inference(checkpoint_path: str=None,
              num_time_steps: int=1000,
              ema_decay: float=0.9999, ):
    checkpoint = torch.load(checkpoint_path)
    model = UNET().to(device)
    model.load_state_dict(checkpoint['weights'])
    ema = ModelEmaV3(model, decay=ema_decay)
    ema.load_state_dict(checkpoint['ema'])
    scheduler = DDPM_Scheduler(num_time_steps=num_time_steps)
    times = [0,15,50,200,500,700,999]
    images = []
    img_path = glob("/kaggle/input/brats18/dataset/Validation/
 ↪BraTS18_Validation_00*/BraTS18_Validation_00*_flair.nii")
    img_path.sort(key=natural_keys)
    for n in range(1,6):
        img_3D = nib.load(img_path[n]).get_fdata()
        patch = torch.tensor(downsample_volume(img_3D), dtype=torch.float32)
        patch = (patch[:,:,50]/torch.max(patch[:,:,50]+0.000001)).unsqueeze(0).
 ↪unsqueeze(0)
        plt.imshow(patch.squeeze(0).squeeze(0),cmap='gray')
        plt.show()
        print(patch.shape)

        with torch.no_grad():
            model = ema.module.eval()
```

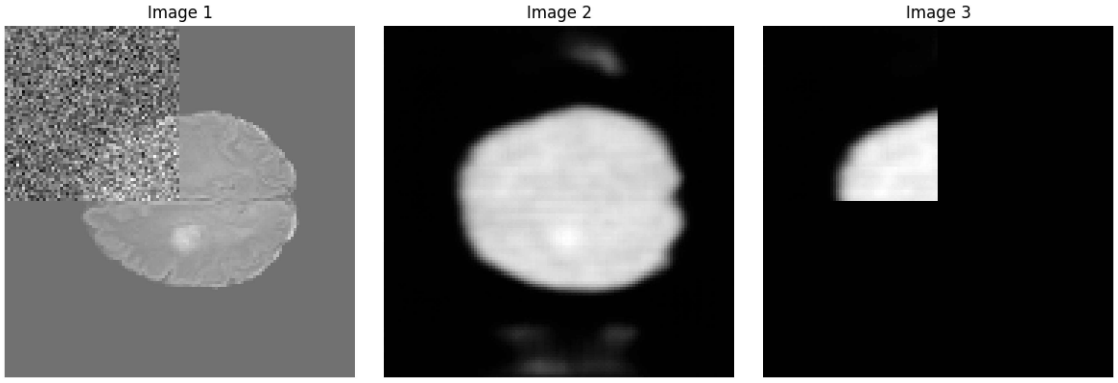
```
            time_ex=[100,200,300,500,999]
            for t in time_ex:
                x_final = []
                # print(x_final.shape)
                for i in range(len(mask)):

                    e = torch.randn_like( patch, requires_grad=False)
                    a = scheduler.alpha[t].view(1,1,1,1)
                    x_out = (torch.sqrt(a)* patch) + (torch.sqrt(1-a)*e)
                    z = x_out*mask[i] +(1-mask[i])*patch
                    x_l = model(z.to(device),[500]).cpu()
                    x = x_l*mask[i]
                    # for t in reversed(range(1, num_time_steps)):
                    #     t = [t]
                    #     temp = (scheduler.beta[t]/( (torch.sqrt(1-scheduler.
↪alpha[t]))*(torch.sqrt(1-scheduler.beta[t])) ))
                    #     z = (1/(torch.sqrt(1-scheduler.beta[t])))*z -␣
↪(temp*model(z.to(device),t).cpu())
                    #     print(t)
                    #     print(model(z.to(device),t).shape)
                    #     break
                    #     if t[0] in times:
                    #         images.append(z)
                    #     e = torch.randn(1, 1, 240, 240)
                    #     z = z + (e*torch.sqrt(scheduler.beta[t]))
                    # temp = scheduler.beta[0]/( (torch.sqrt(1-scheduler.
↪alpha[0]))*(torch.sqrt(1-scheduler.beta[0])) )
                    # x = (1/(torch.sqrt(1-scheduler.beta[0])))*z -␣
↪(temp*model(z.to(device),[0]).cpu())
                    x_final.append(x)
                    images.append(x)
                    z = rearrange(z.squeeze(0), 'c h w -> h w c').detach().cpu()
                    z = z.numpy()
                    x_l = rearrange(x_l.squeeze(0), 'c h w -> h w c').detach().
↪cpu()
                    x_l = x_l.numpy()
                    x = rearrange(x.squeeze(0), 'c h w -> h w c').detach().cpu()
                    x = x.numpy()
                    fig, axes = plt.subplots(1, 3, figsize=(12, 4))  # 1 row, 3␣
↪columns
                    axes[0].imshow(z, cmap='gray')
                    axes[0].set_title('Image 1')  # Optional: Add title
                    axes[0].axis('off')           # Optional: Remove axes

                    axes[1].imshow(x_l, cmap='gray')
```

```
                    axes[1].set_title('Image 2')  # Optional
                    axes[1].axis('off')

                    axes[2].imshow(x, cmap='gray')
                    axes[2].set_title('Image 3')  # Optional
                    axes[2].axis('off')

                    plt.tight_layout()  # Adjust spacing between subplots
                    plt.show()
                    print("next")
                final = maskk(patch,x_final)
                plt.imshow(final,cmap='gray')
                plt.show()
                # print(x_final.shape,patch.shape)
                diff = patch.squeeze(0).squeeze(0)-final
                # diff[diff<0]=0
                plt.imshow(diff,cmap='gray')
                plt.show()
                # plt.imshow(z-x,cmap='gray')
            # plt.show()
            # display_reverse(images)
device='cuda'
inference('/kaggle/working/ddpm')
# inference("/kaggle/input/brain-dff-2/ddpm")
```

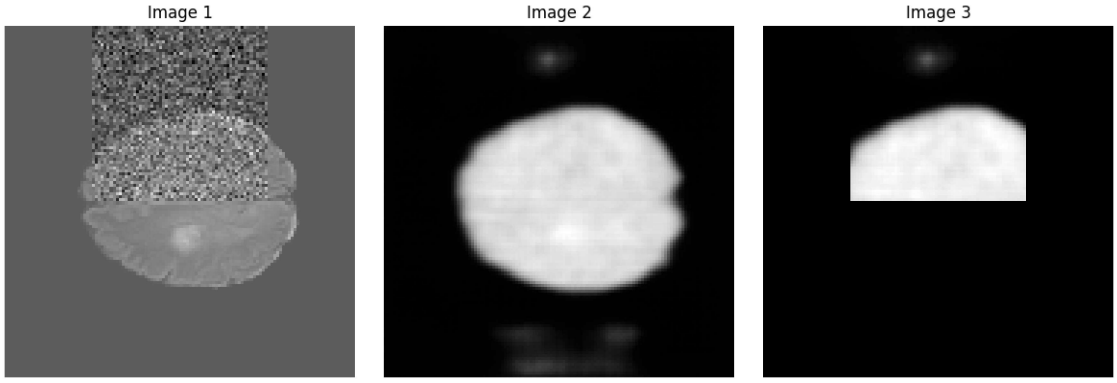
```
/tmp/ipykernel_23/2550445856.py:15: FutureWarning: You are using `torch.load`
with `weights_only=False` (the current default value), which uses the default
pickle module implicitly. It is possible to construct malicious pickle data
which will execute arbitrary code during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models 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.
  checkpoint = torch.load(checkpoint_path)
```



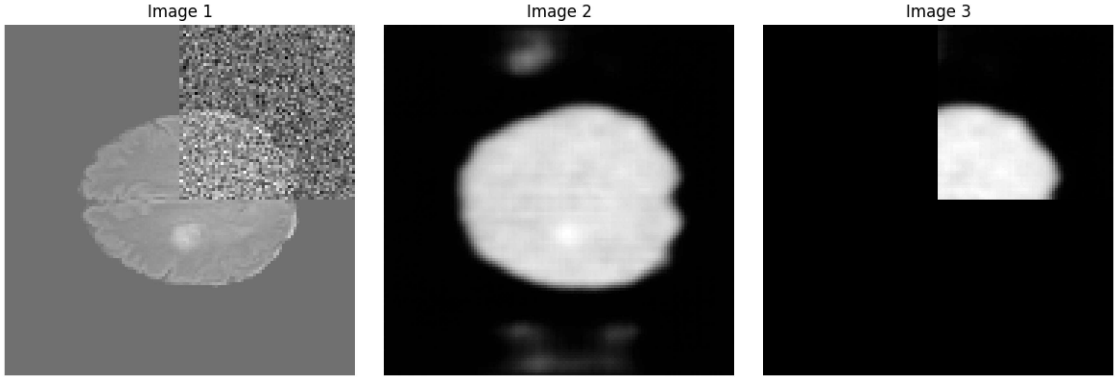

```
torch.Size([1, 1, 120, 120])
```



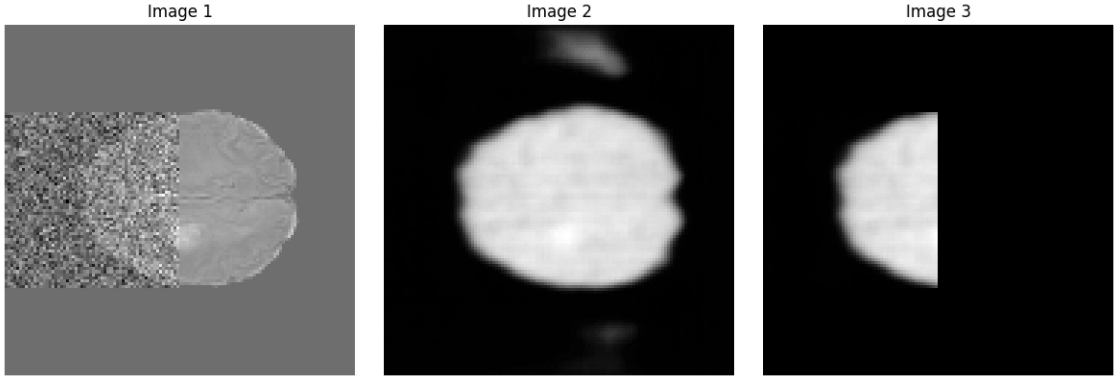

```
next
```



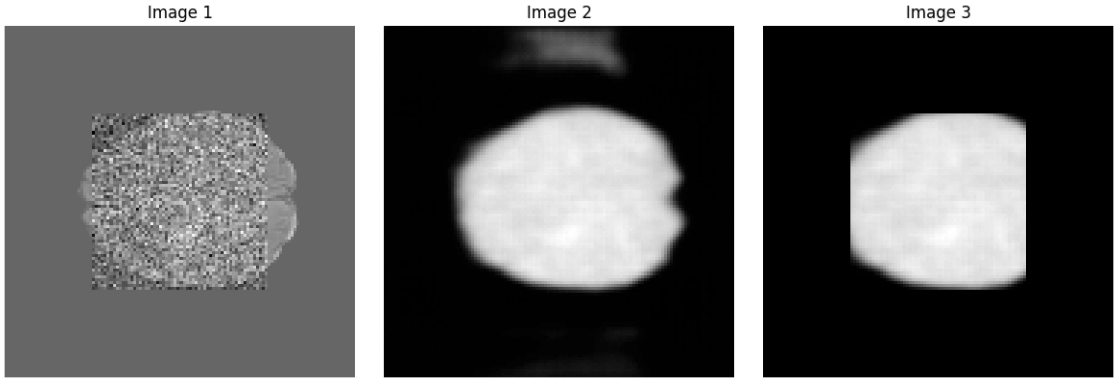
next



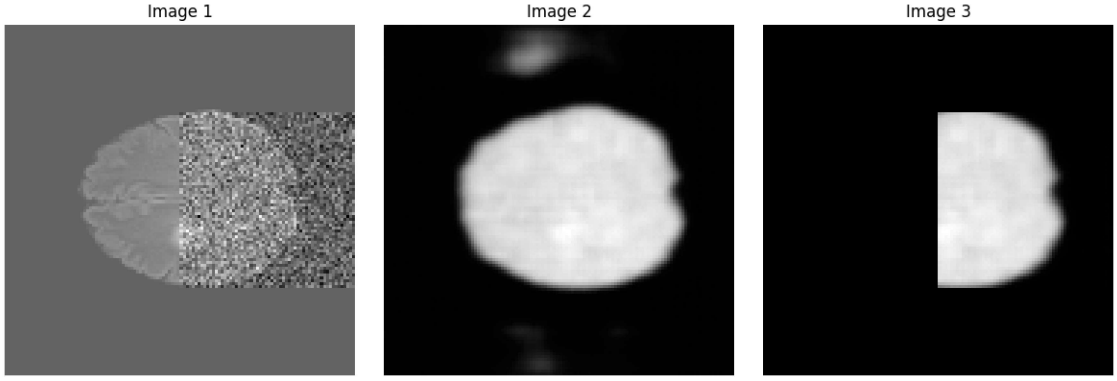
next



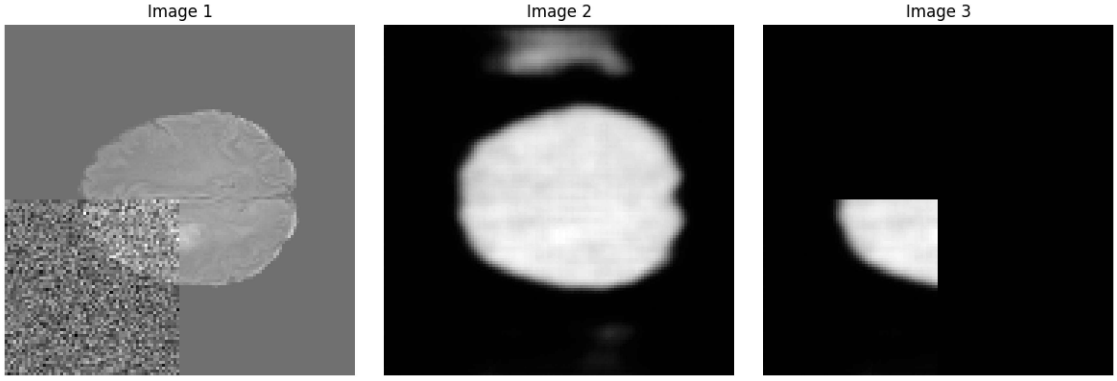
next



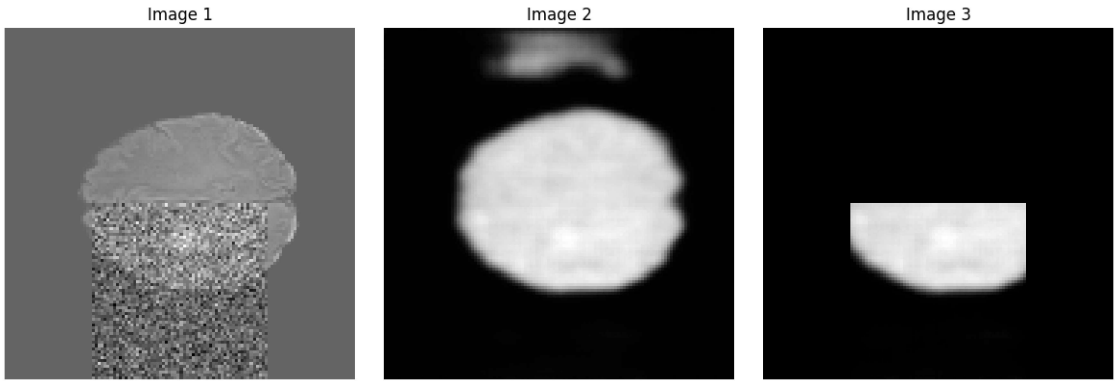
next



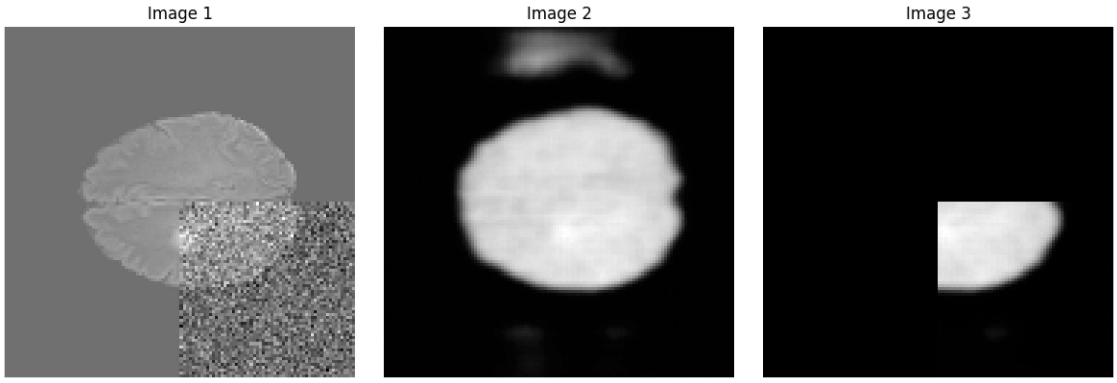
next



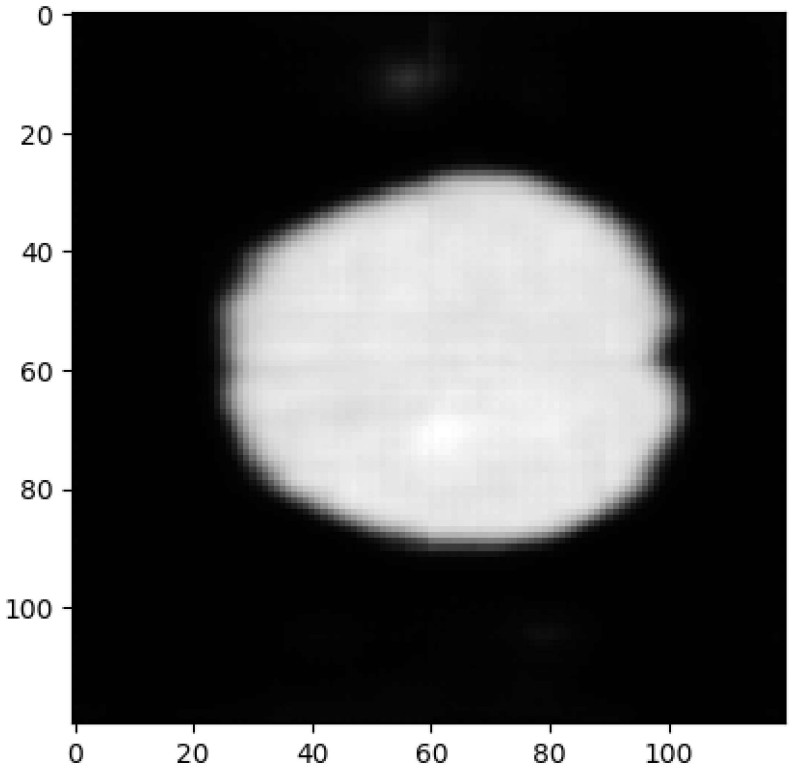
next

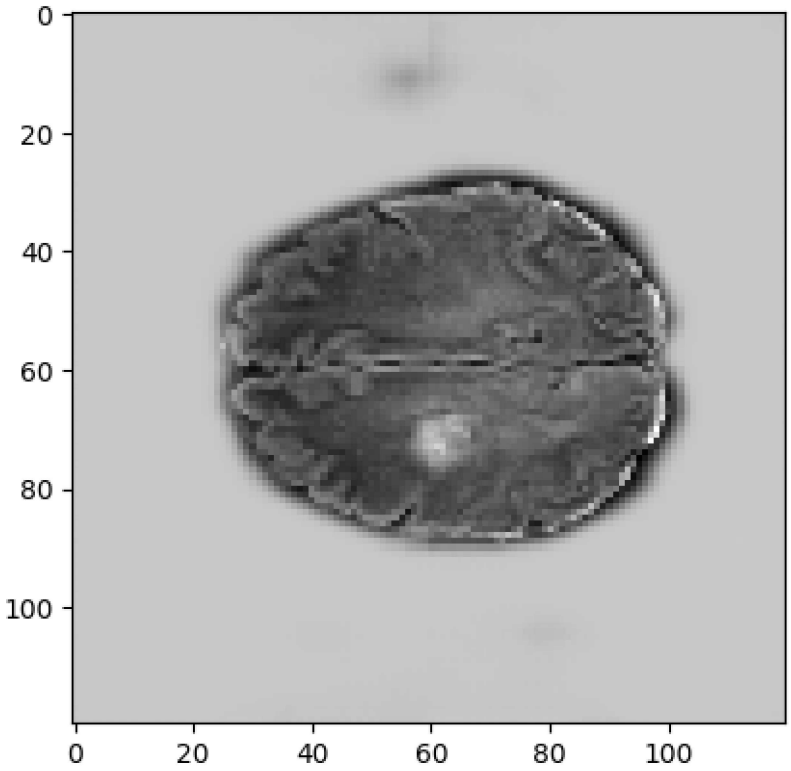


next



next


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100

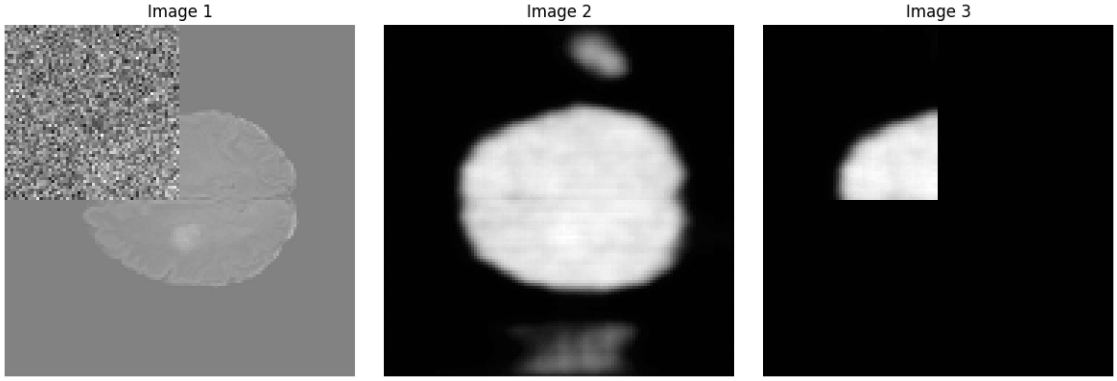


Image 1
Image 2
Image 3

next

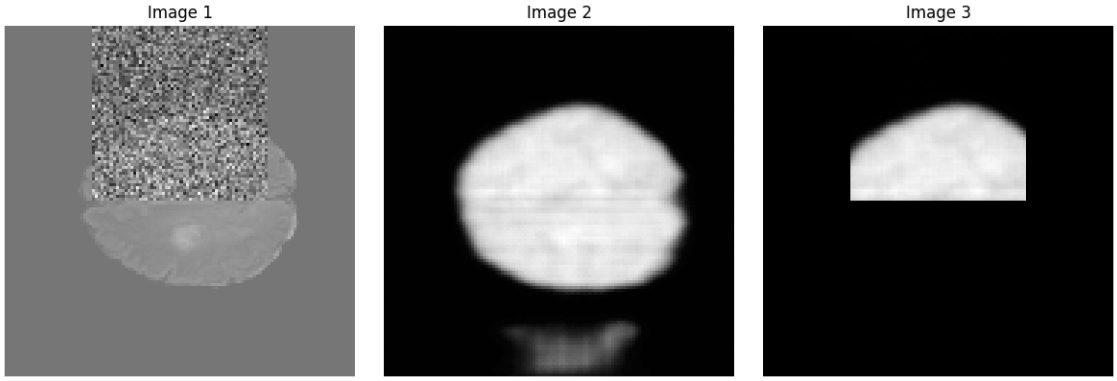


Image 1
Image 2
Image 3

next

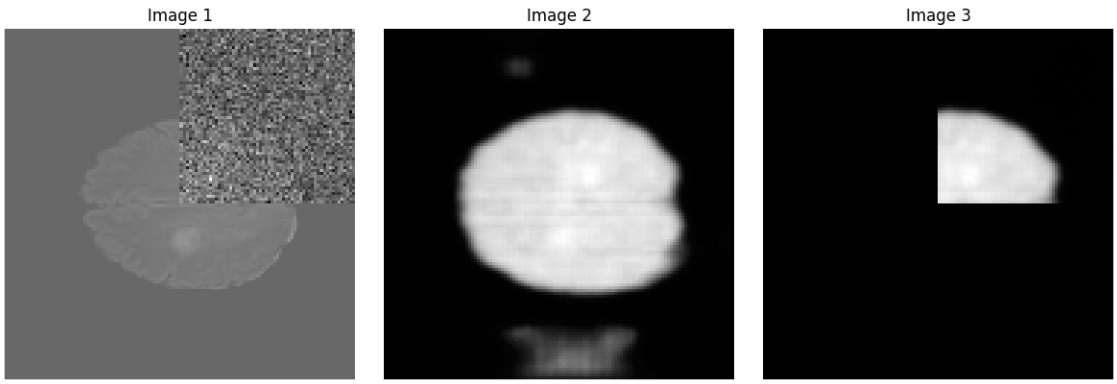
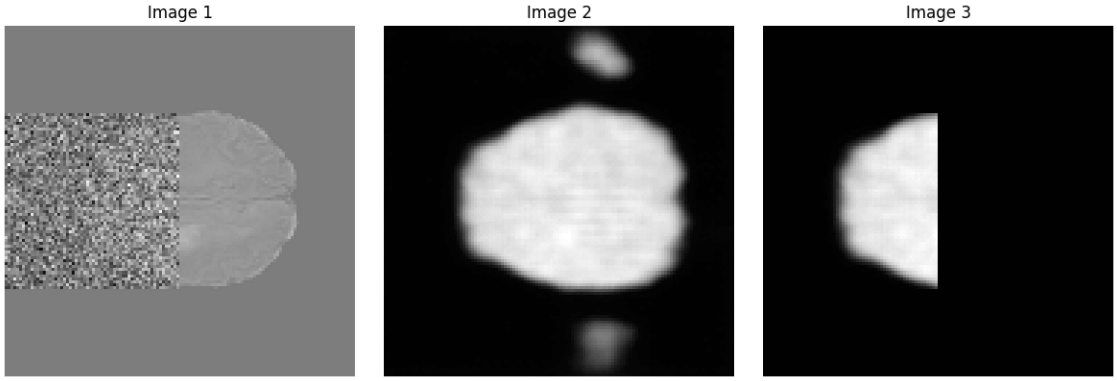
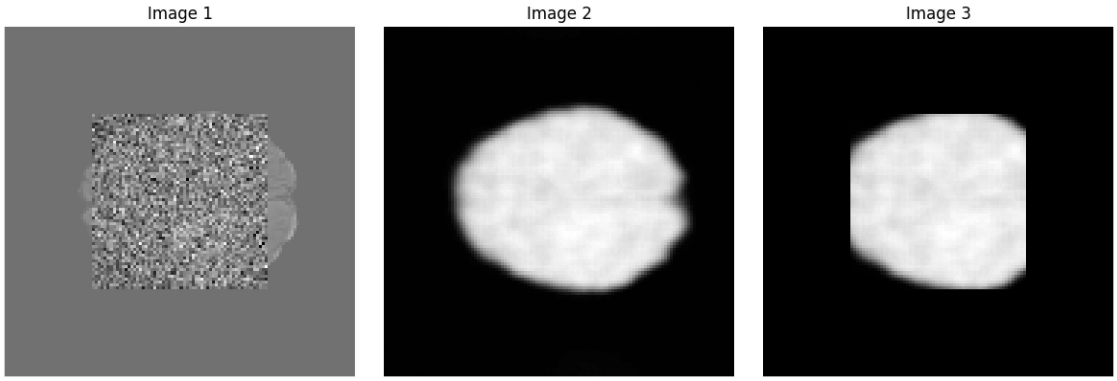


Image 1
Image 2
Image 3

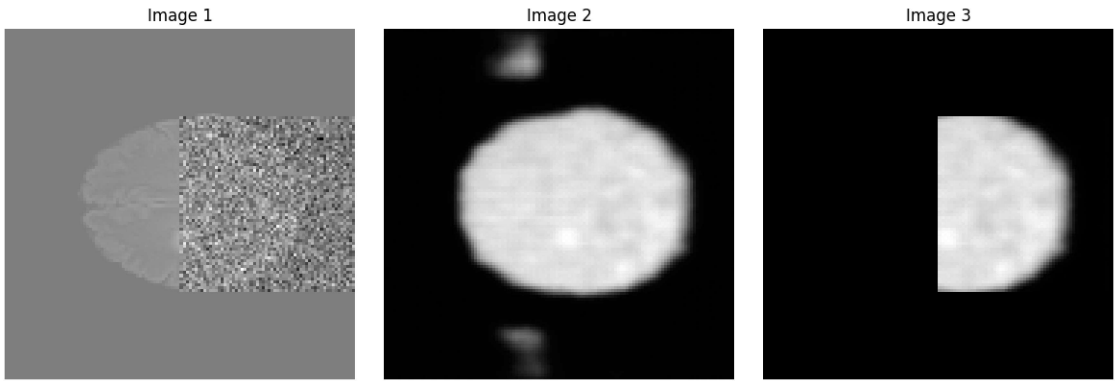
next



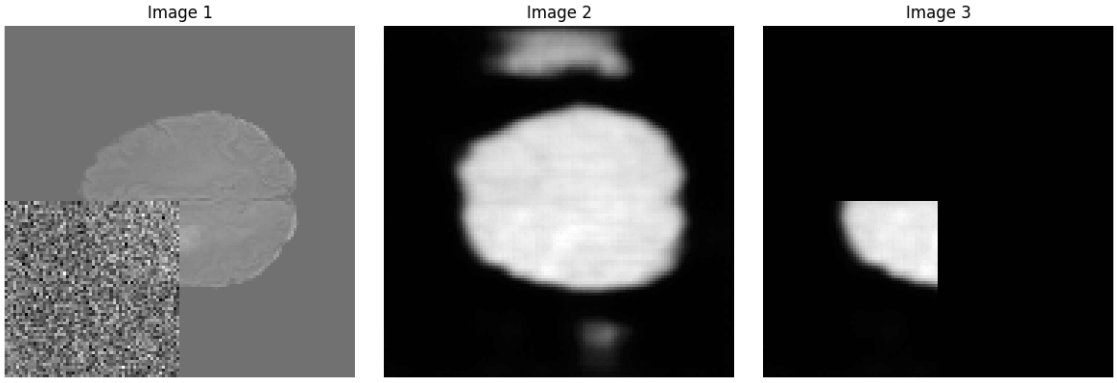
next



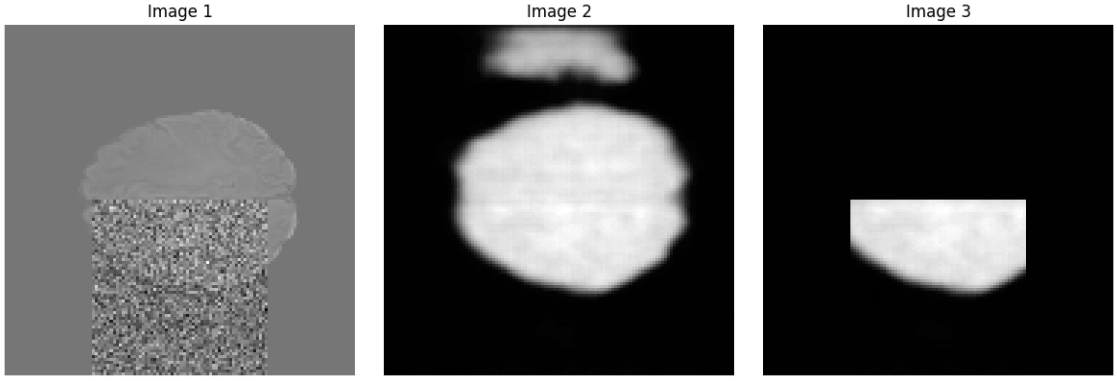
next



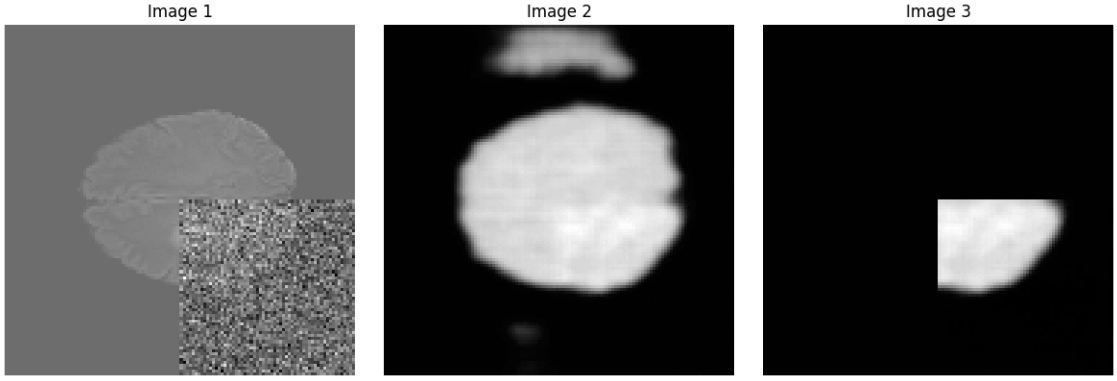
next



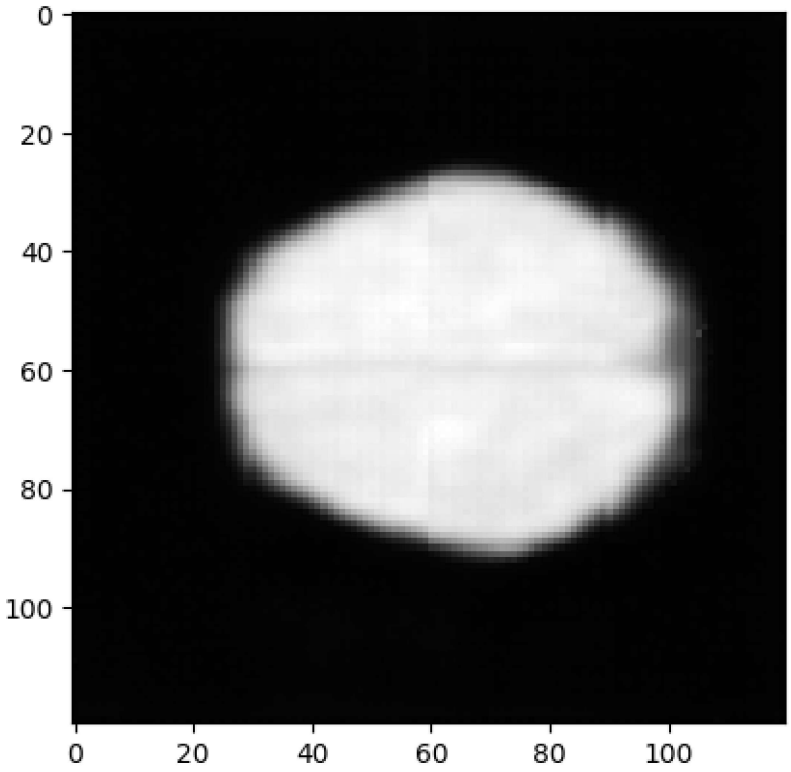
next

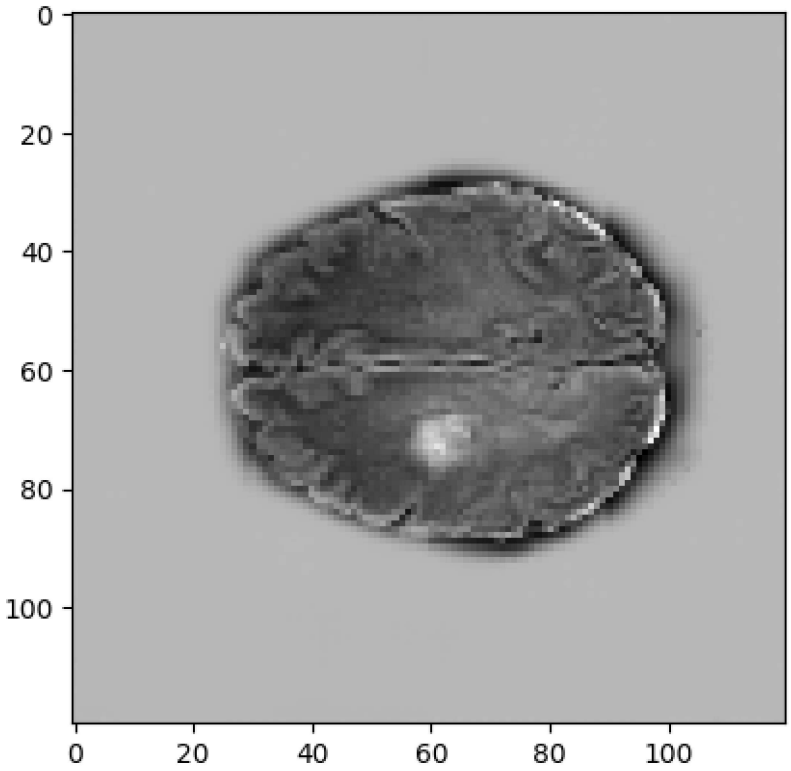


next



next


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100

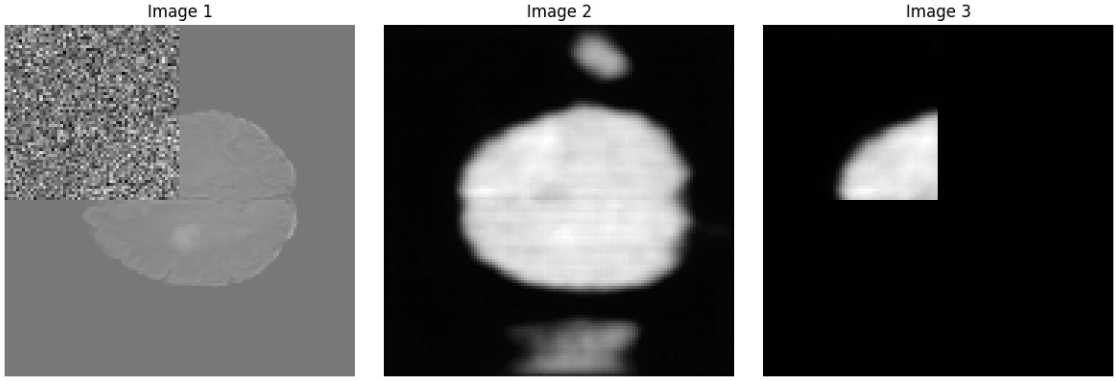


Image 1
Image 2
Image 3

next

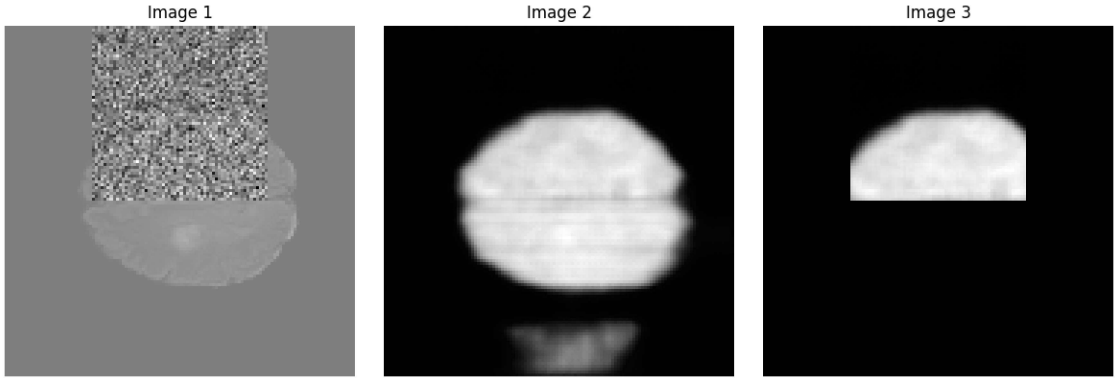


Image 1
Image 2
Image 3

next

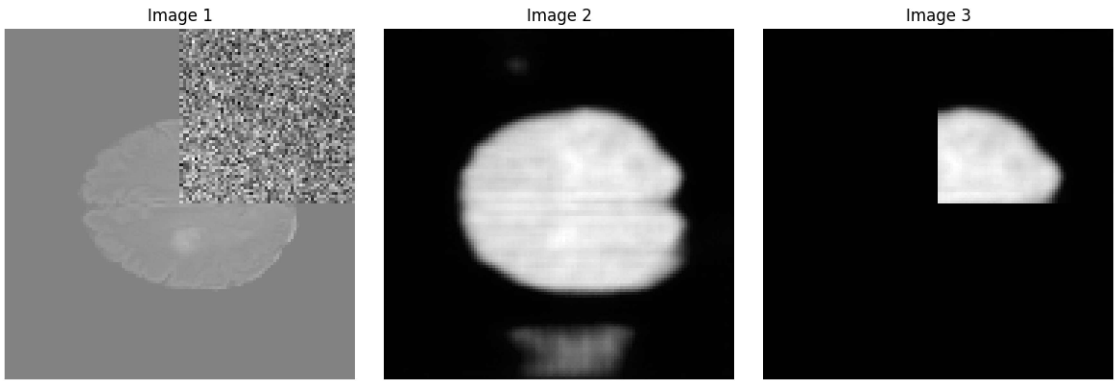
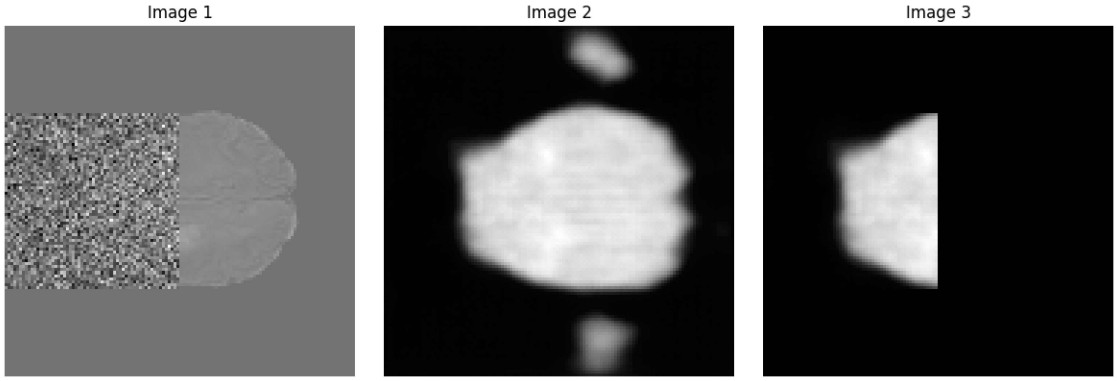
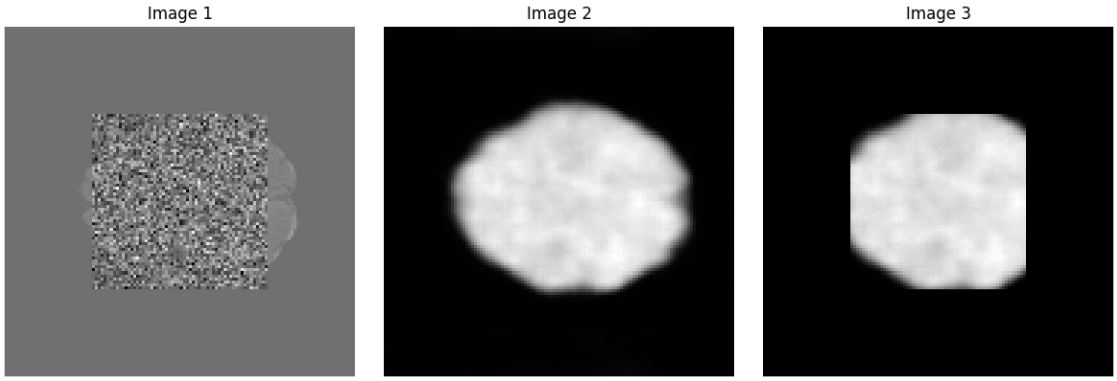


Image 1
Image 2
Image 3

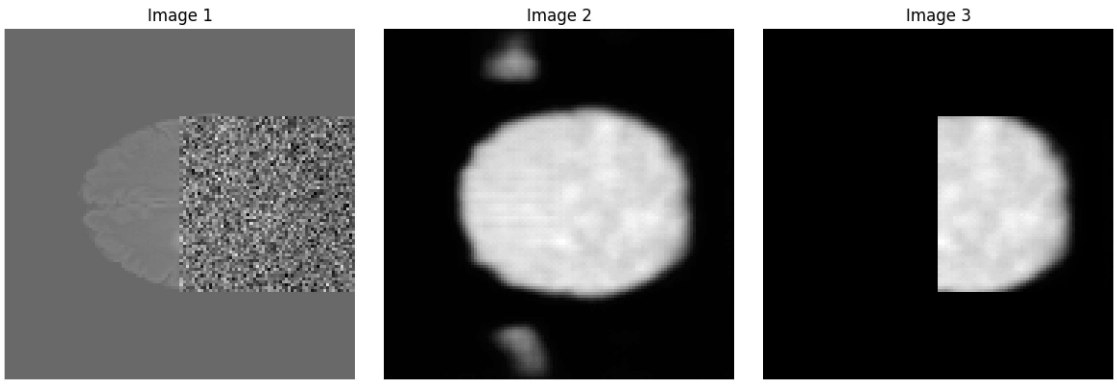
next



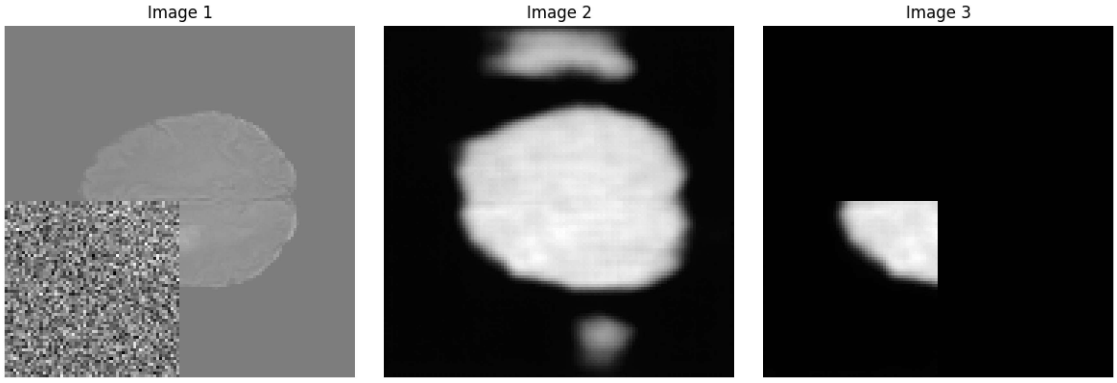
next



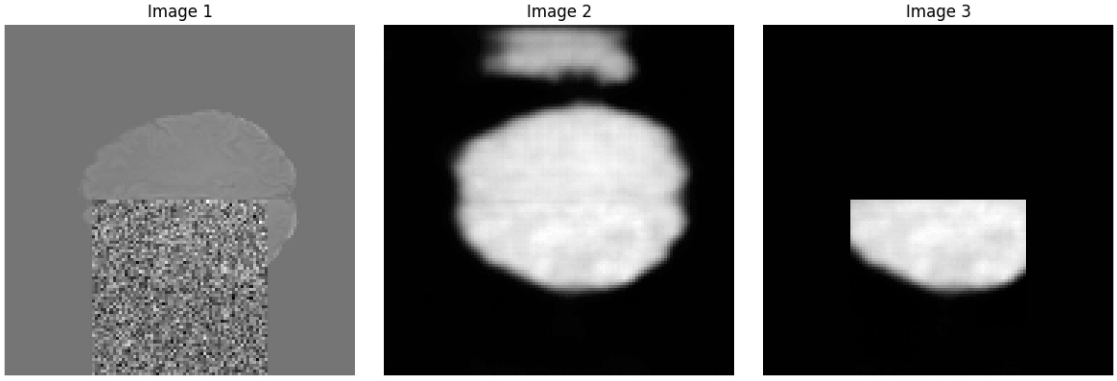
next



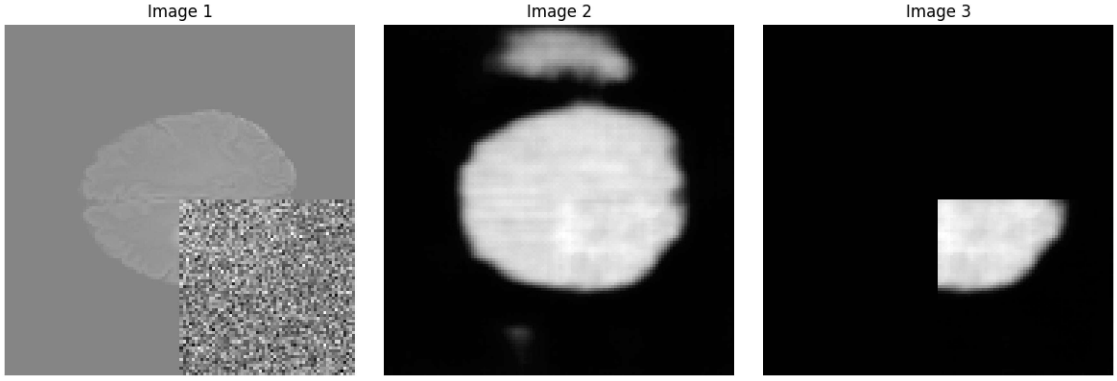
next



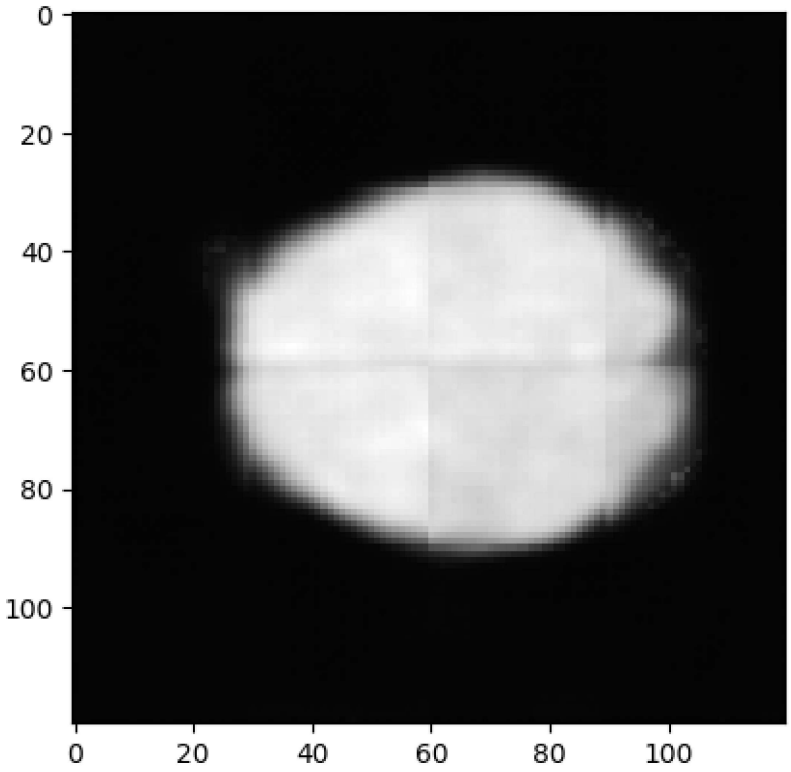
next

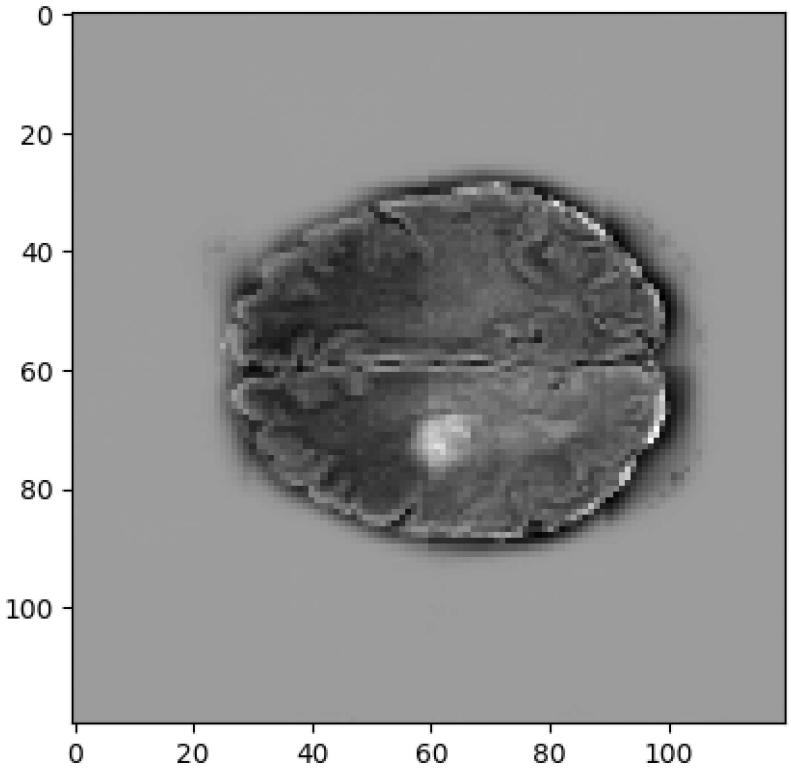


next



next


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100

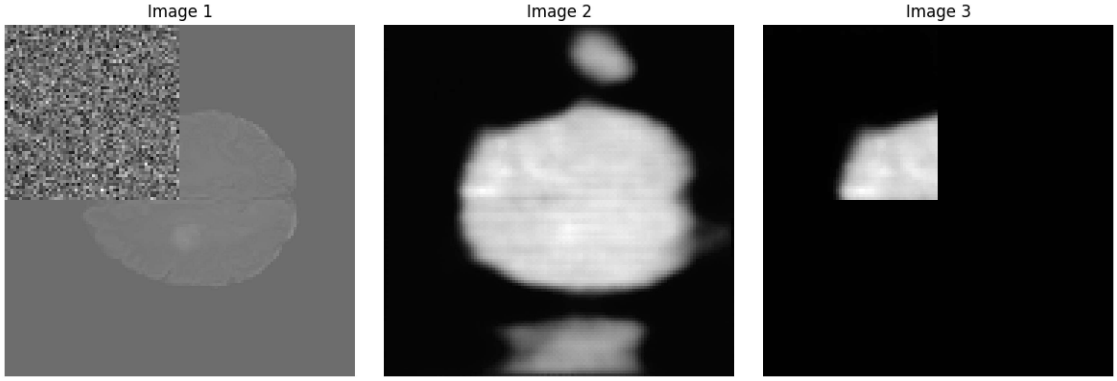


Image 1
Image 2
Image 3

next

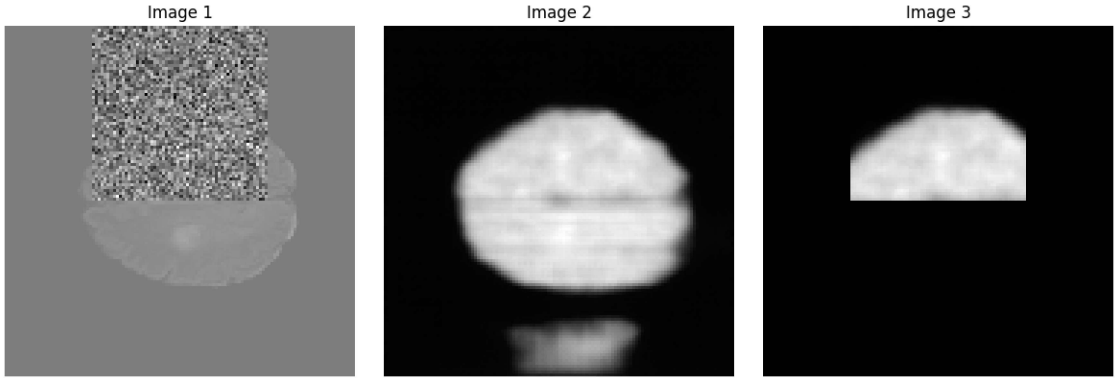


Image 1
Image 2
Image 3

next

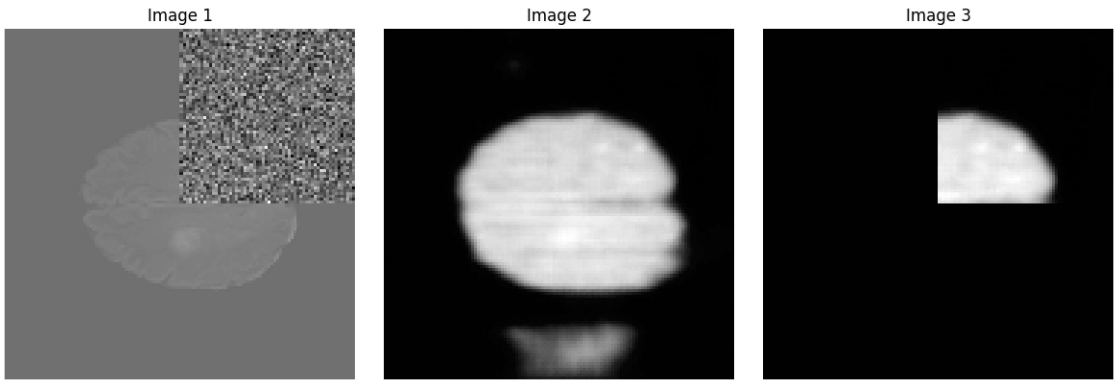
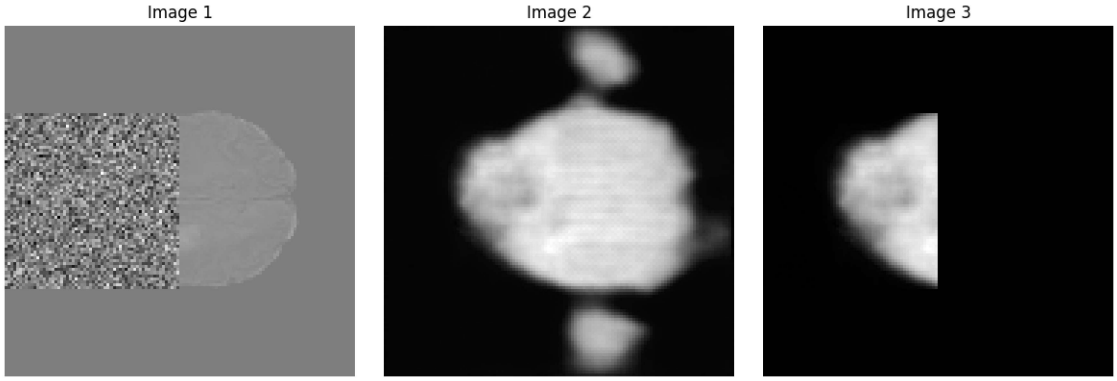
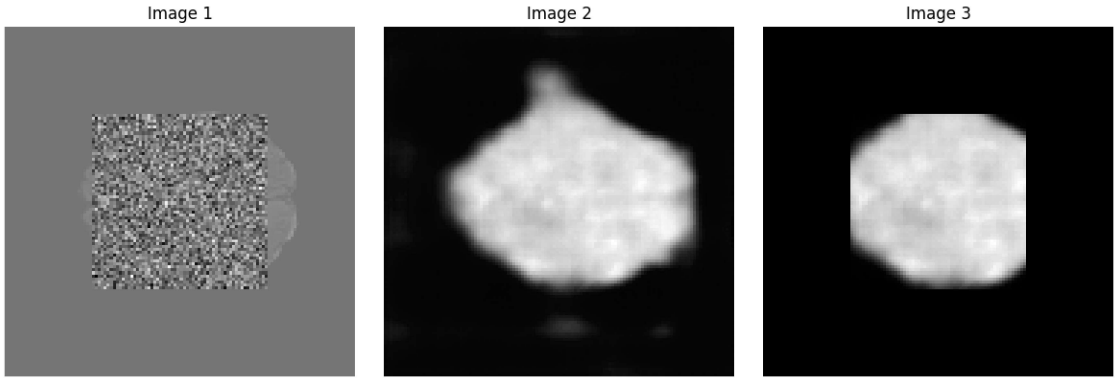


Image 1
Image 2
Image 3

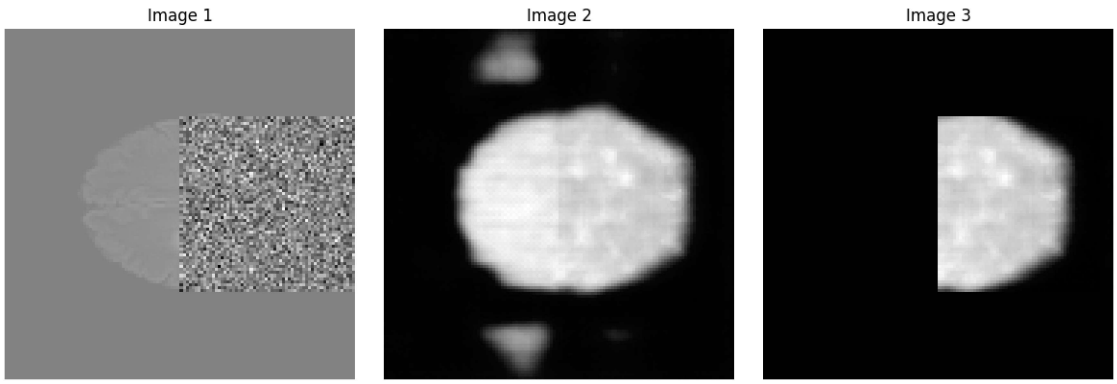
next



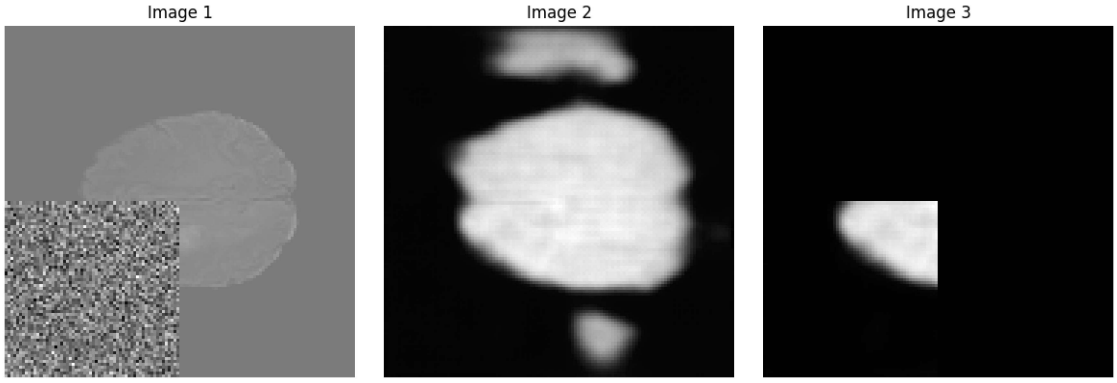
next



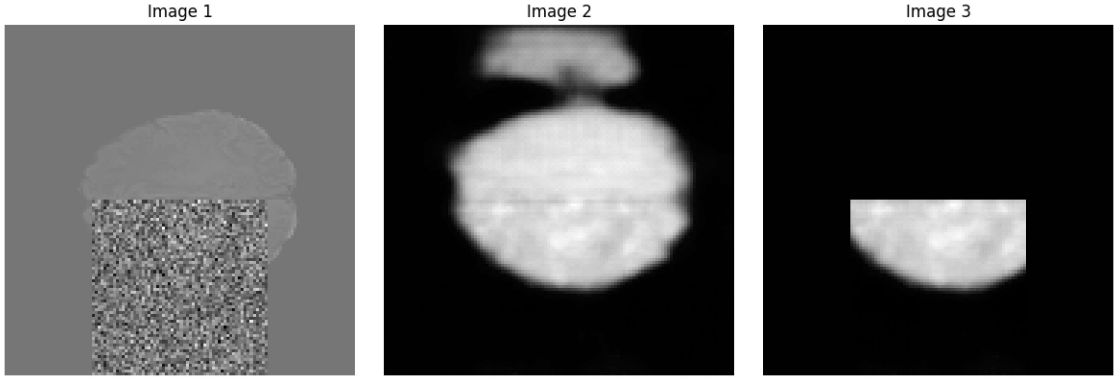
next



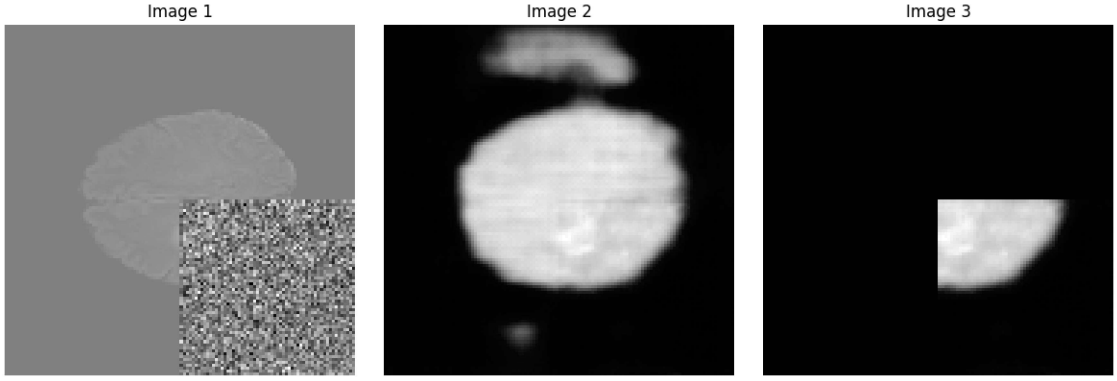
next



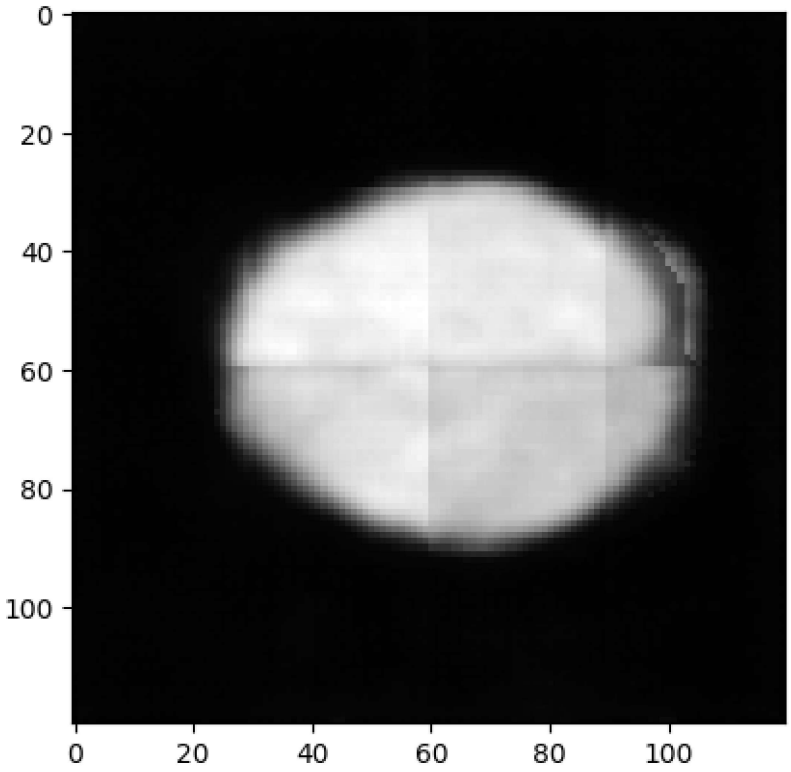
next

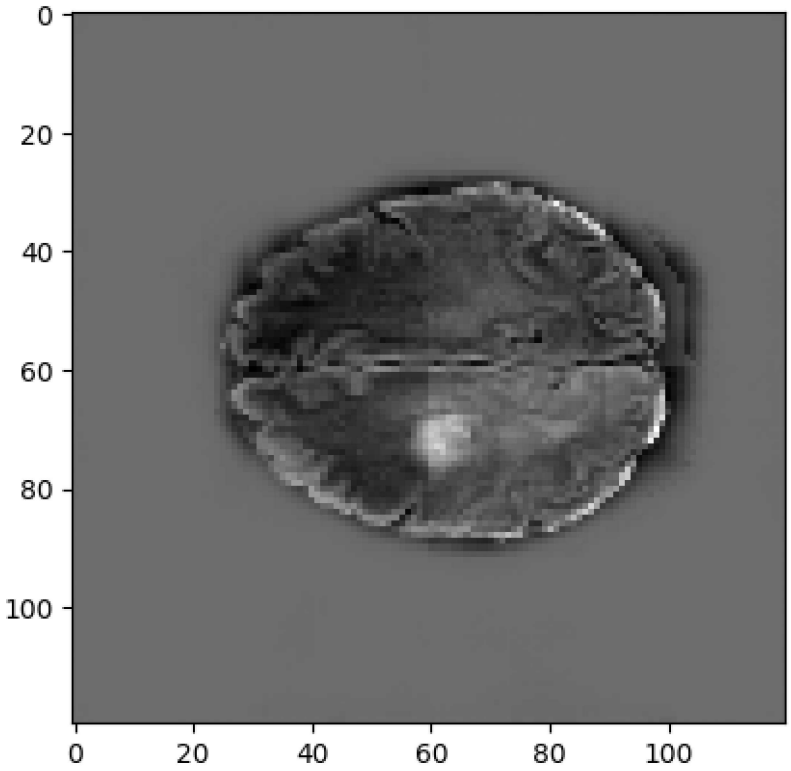


next



next


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100

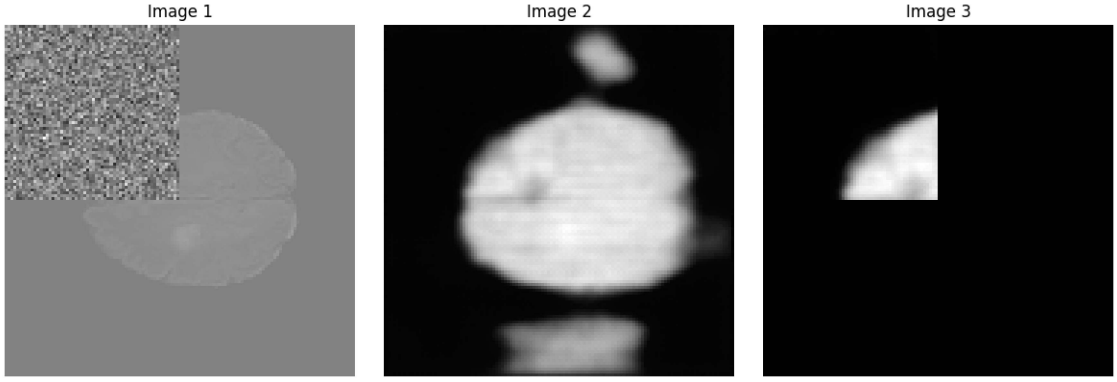


Image 1
Image 2
Image 3

next

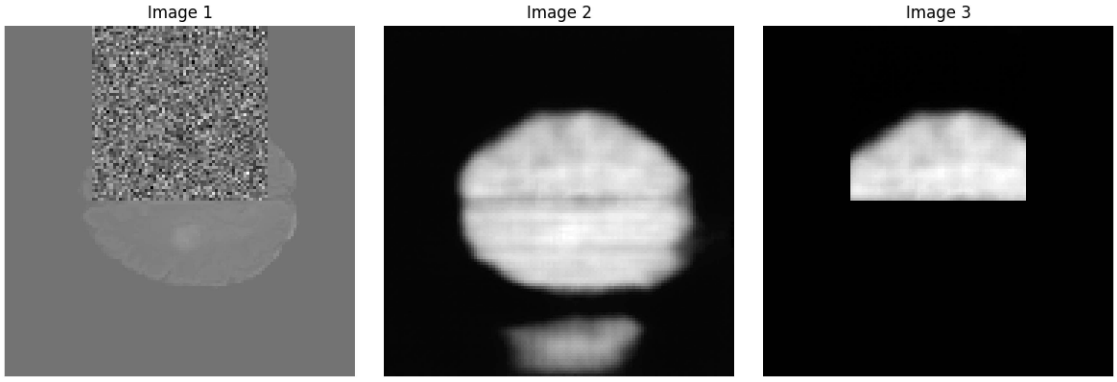


Image 1
Image 2
Image 3

next

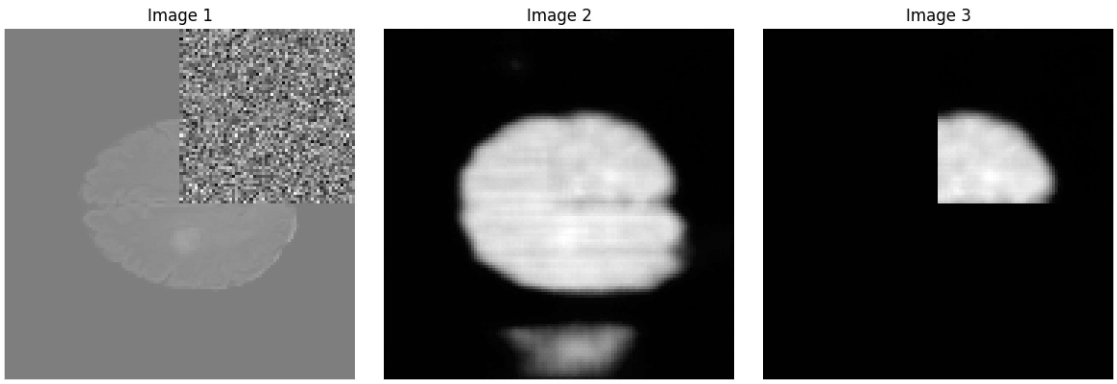
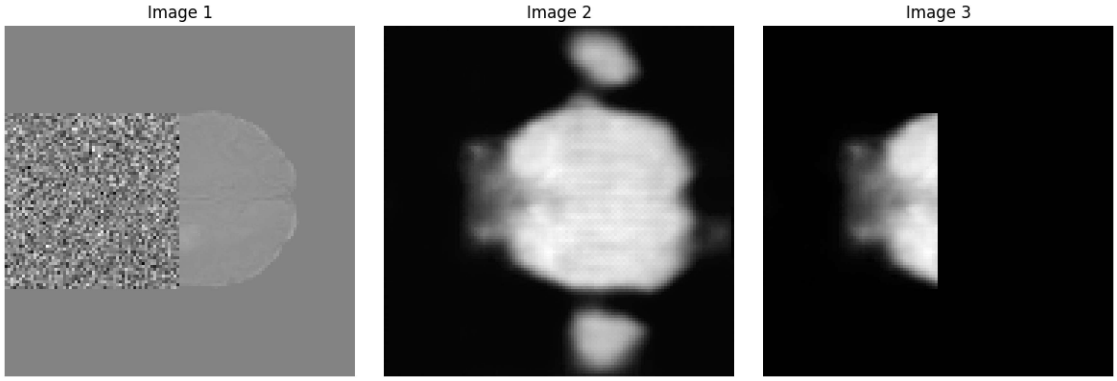
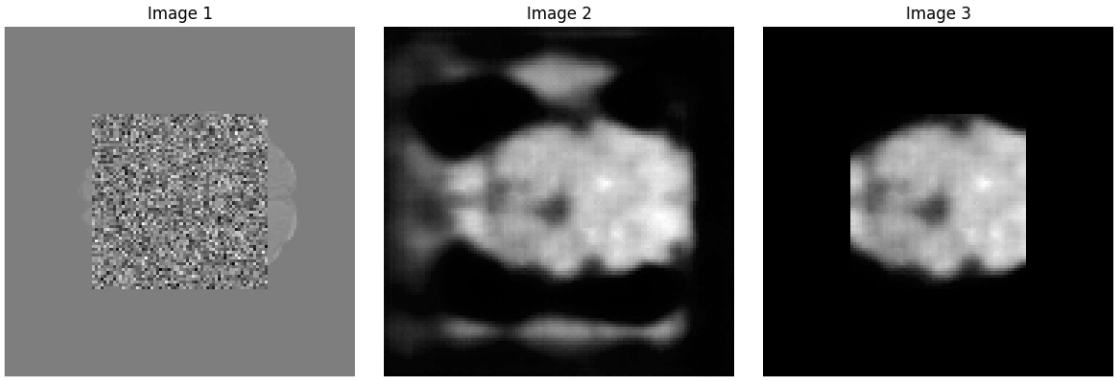


Image 1
Image 2
Image 3

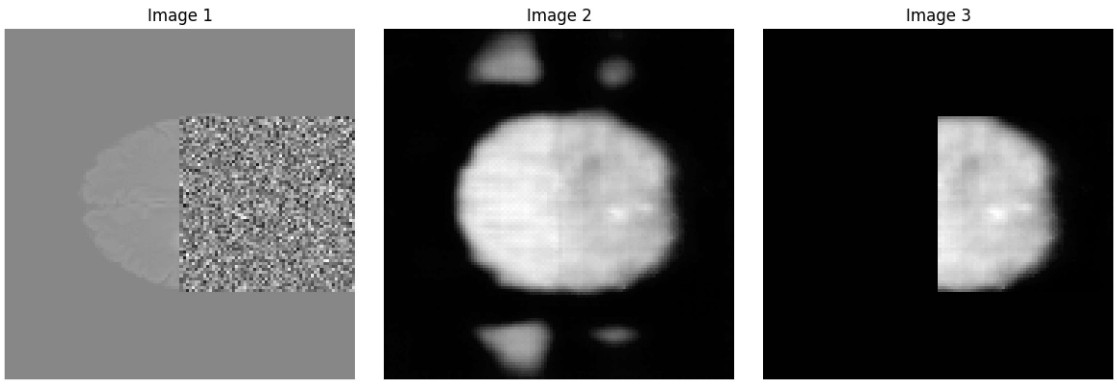
next



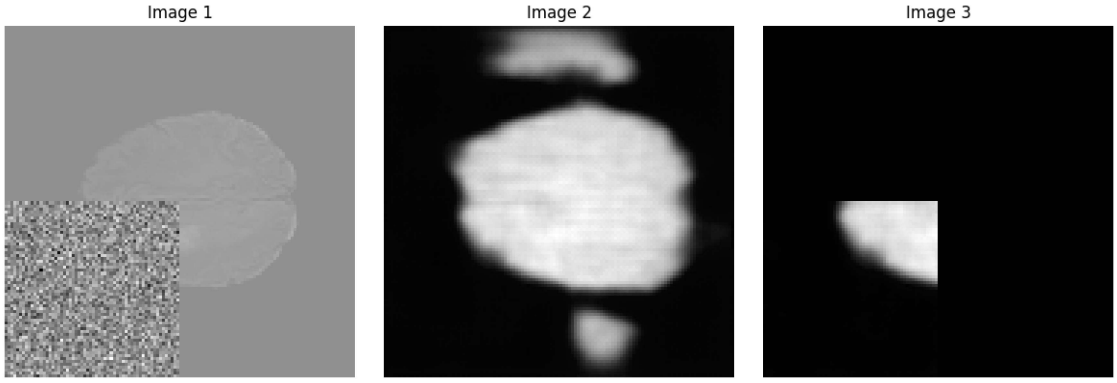
next



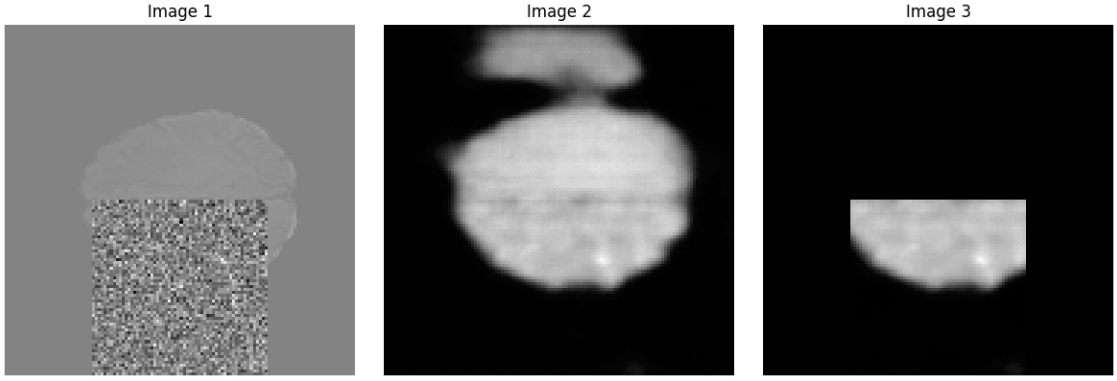
next



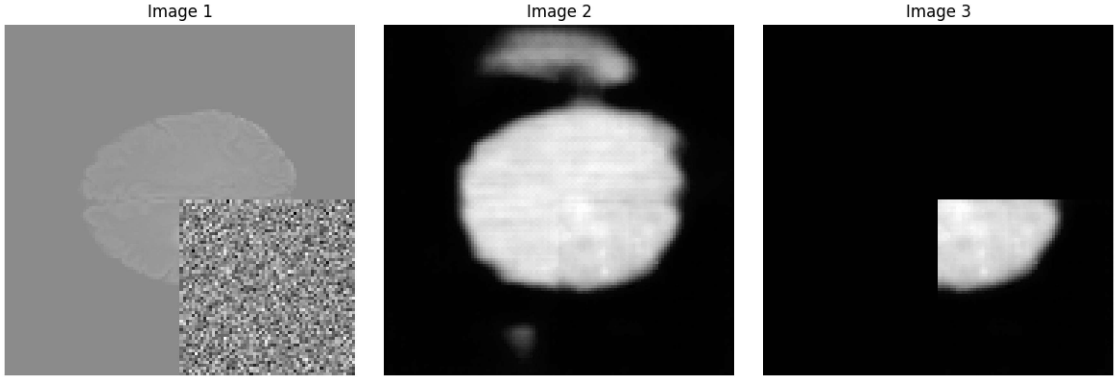
next



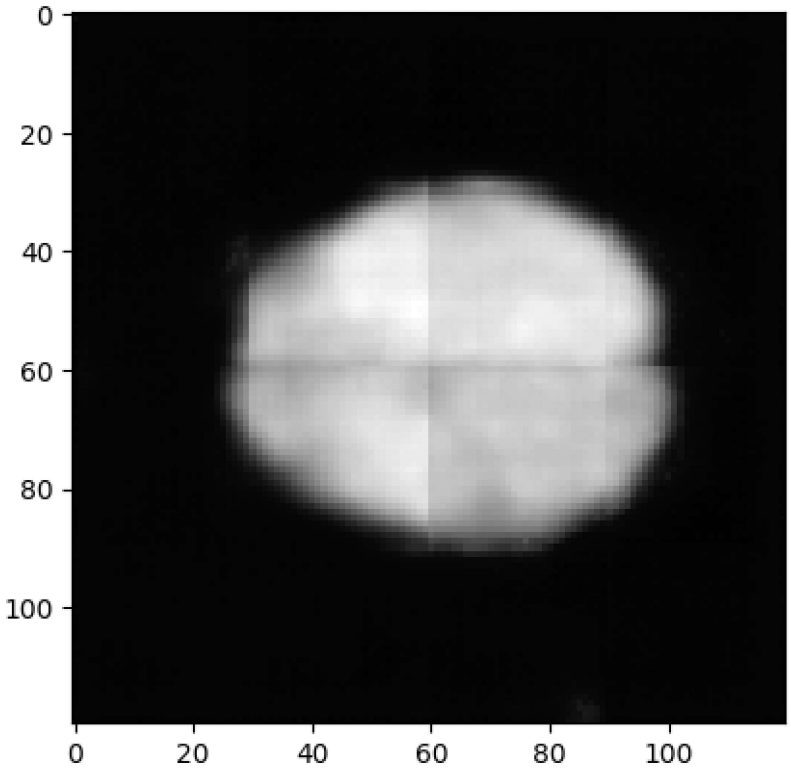
next

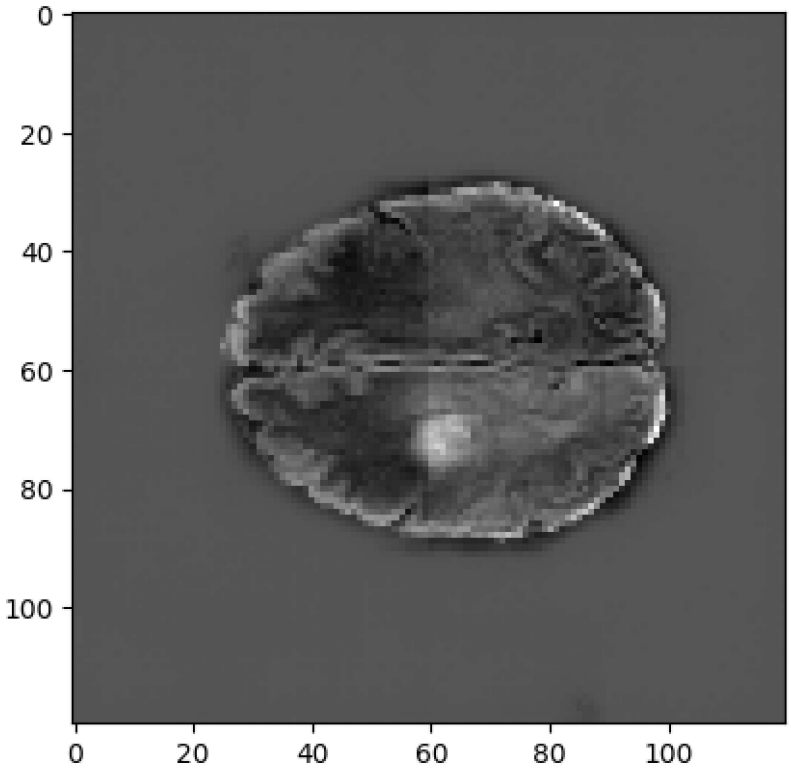


next



next


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100