```
pip install voxelmorph
```

```
Collecting voxelmorph
  Downloading voxelmorph-0.2-py3-none-any.whl (54 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.2/54.2 kB 1.5 MB/s eta 0:00:0
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: nibabel in /usr/local/lib/python3.10/dist-packa
Collecting neurite>=0.2 (from voxelmorph)
  Downloading neurite-0.2-py3-none-any.whl (108 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 108.9/108.9 kB 5.9 MB/s eta 0:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
Collecting pystrum>=0.2 (from neurite>=0.2->voxelmorph)
  Downloading pystrum-0.4.tar.gz (17 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.1
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Building wheels for collected packages: pystrum
  Building wheel for pystrum (setup.py) ... done
  Created wheel for pystrum: filename=pystrum-0.4-py3-none-any.whl size=19532
  Stored in directory: /root/.cache/pip/wheels/aa/08/d0/914025beb5a12a855b8aa
Successfully built pystrum
Installing collected packages: pystrum, neurite, voxelmorph
Successfully installed neurite-0.2 pystrum-0.4 voxelmorph-0.2
```

```
1 !nvidia-smi
```

```
Fri Dec 15 17:58:58 2023
+---------------------------------------------------------------------------------
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version:
|-----------------------------------------+----------------------+----------------
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Un
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  C
|                                         |                      |
|=========================================+======================+================
|   0  Tesla V100-SXM2-16GB           Off | 00000000:00:04.0 Off |
| N/A   35C    P0              40W / 300W |  15124MiB / 16384MiB |      0%
|                                         |                      |
+-----------------------------------------+----------------------+----------------

+---------------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                             G
|        ID   ID                                                              U
|=================================================================================
+---------------------------------------------------------------------------------
```

```
1 !pip install numba
```

```
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/py
Requirement already satisfied: numpy<1.27,>=1.22 in /usr/local/lib/python3.10/
```

```
1 from numba import cuda
2 device = cuda.get_current_device()
3 device.reset()
```

```
1 !nvidia-smi
```

```
Fri Dec 15 18:53:41 2023
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version:
|-----------------------------------------+----------------------+----------------------
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Un
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  C
|                                         |                      |
|=========================================+======================+======================
|   0  Tesla V100-SXM2-16GB           Off | 00000000:00:04.0 Off |
| N/A   44C    P0              58W / 300W |      2MiB / 16384MiB |    100%
|                                         |                      |
+-----------------------------------------+----------------------+----------------------

+---------------------------------------------------------------------------------------+
| Processes:
|  GPU   GI   CI        PID   Type   Process name                                 G
|        ID   ID                                                                  U
|=======================================================================================
|  No running processes found
+---------------------------------------------------------------------------------------+
```

```
1 from google.colab import auth
2 auth.authenticate_user()
```

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
1 import os
2 import nibabel as nib
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import tensorflow as tf
6 import voxelmorph as vxm
7 import neurite as ne
```

```
1 dir_folder = os.path.join('/content/drive','My Drive','hw4','Dataset_HW4')
```

```
train_folder = os.path.join(dir_folder,'train')
test_folder = os.path.join(dir_folder,'test')
```

```
def load_img(files_path):

    batch_size = len(files_path)
    temp_file = nib.load(files_path[0])
    img_shape = temp_file.shape
    img_array = np.zeros([batch_size,*img_shape])

    for i,files in enumerate(files_path):

        img = nib.load(files)
        img_array[i] = img.get_fdata()

    return np.expand_dims(img_array,axis = -1)
```

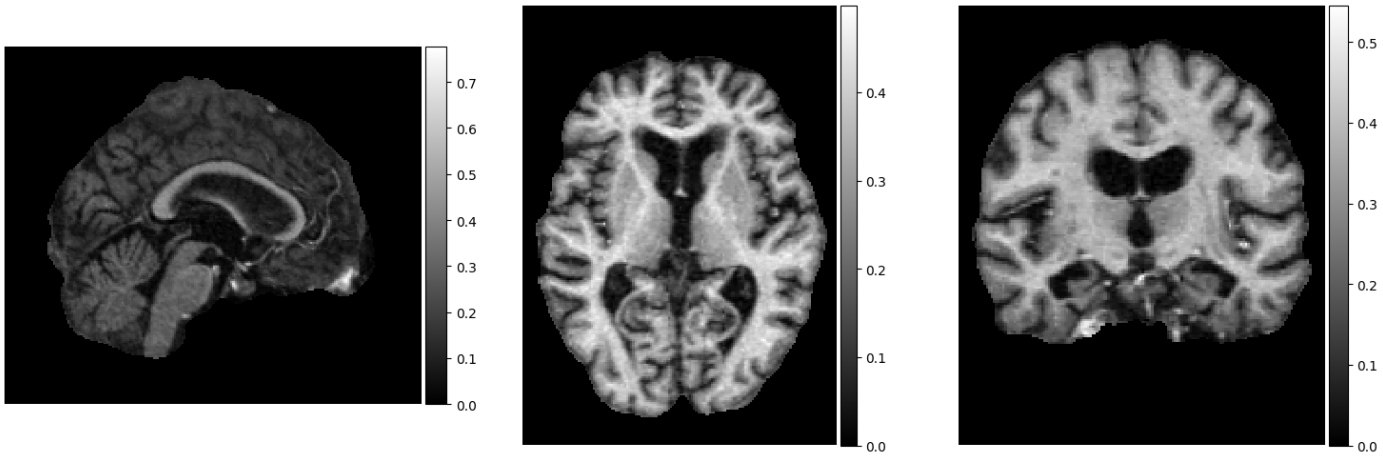
```
def data_generator(dir_folder,batch_size = 32):

    while True:

        files = os.listdir(dir_folder)
        #print(files)
        num_files = len(files)
        idx1 = np.random.randint(0,num_files,size = batch_size)
        files_1 = [os.path.join(dir_folder,files[i]) for i in idx1]
        moving_images = load_img(files_1)

        idx2 = np.random.randint(0,num_files,size = batch_size)
        files_2 = [os.path.join(dir_folder,files[i]) for i in idx2]
        fixed_images = load_img(files_2)

        inputs = [moving_images,fixed_images]

        img_shape = moving_images.shape[1:]
        img_shape = img_shape[:-1]

        ndims = len(img_shape)
        zero_phi = np.zeros([batch_size,*img_shape,ndims])
        outputs = [fixed_images,zero_phi]

        yield (inputs,outputs)

    # while True:

    #     idx1 = np.random.randint(0,num_files,size = batch_size)

    #     moving_images = load_img(
```

```
#compile model

vol_shape = (160, 192, 224)
nb_features = [
    [16, 32, 32, 32],
    [32, 32, 32, 32, 32, 16, 16]
]
vxm_model = vxm.networks.VxmDense(vol_shape, nb_features, int_steps=0)

losses = ['mse',vxm.losses.Grad('l2').loss]
loss_weights = [1,0.01]

vxm_model.compile(optimizer = tf.keras.optimizers.legacy.Adam(learning_rate =
```

```
train_generator = data_generator(train_folder,batch_size = 1)
```

```
ip_sample,op_sample = next(train_generator)
print(ip_sample[0].shape,ip_sample[1].shape,op_sample[0].shape,op_sample[1].sh

## show data across all three axis

val_volume_2 = ip_sample[0][0,:,:,:,0]

#print(val_volume_2.shape)

mid_slices_fixed = [np.take(val_volume_2, vol_shape[d]//2, axis=d) for d in ra
mid_slices_fixed[1] = np.rot90(mid_slices_fixed[1], 1)
mid_slices_fixed[2] = np.rot90(mid_slices_fixed[2], -1)

ne.plot.slices(mid_slices_fixed, cmaps=['gray'], do_colorbars=True, grid=[1,3]
```

```
(1, 160, 192, 224, 1) (1, 160, 192, 224, 1) (1, 160, 192, 224, 1) (1, 160, 192
```



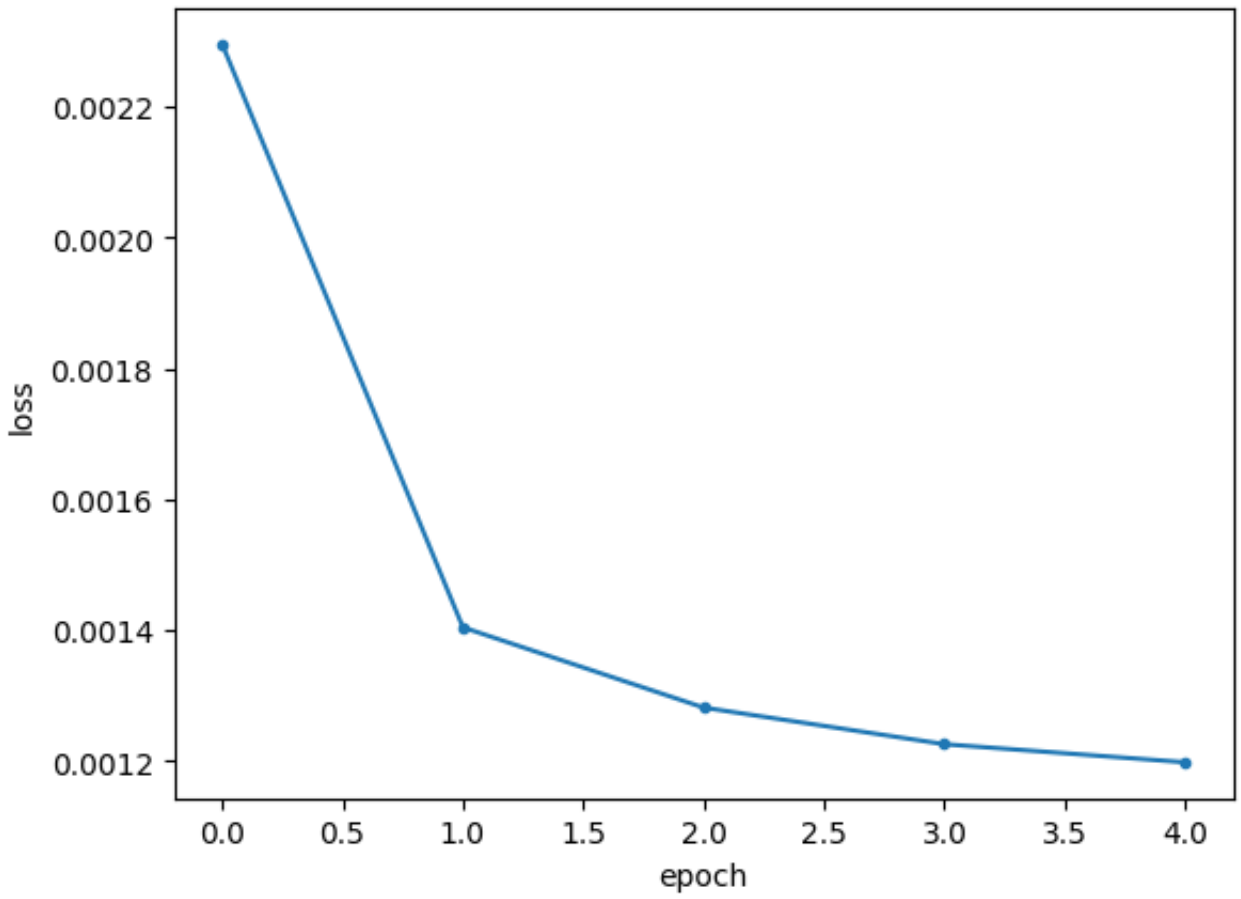
```
(<Figure size 1500x500 with 6 Axes>,
 array([<Axes: >, <Axes: >, <Axes: >], dtype=object))
```

```
# training the model
hist = vxm_model.fit(train_generator,epochs = 5, steps_per_epoch = 800, verbos
```

```
Epoch 1/5
800/800 - 706s - loss: 0.0023 - vxm_dense_transformer_loss: 0.0021 - vxm_dense
Epoch 2/5
800/800 - 651s - loss: 0.0014 - vxm_dense_transformer_loss: 0.0011 - vxm_dense
Epoch 3/5
800/800 - 651s - loss: 0.0013 - vxm_dense_transformer_loss: 9.4760e-04 - vxm_d
Epoch 4/5
800/800 - 651s - loss: 0.0012 - vxm_dense_transformer_loss: 8.9079e-04 - vxm_d
Epoch 5/5
800/800 - 651s - loss: 0.0012 - vxm_dense_transformer_loss: 8.5948e-04 - vxm_d
```

```
import matplotlib.pyplot as plt

def plot_history(hist, loss_name='loss'):
    # Simple function to plot training history.
    plt.figure()
    plt.plot(hist.epoch, hist.history[loss_name], '.-')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.show()
```

```
1 plot_history(hist)
```

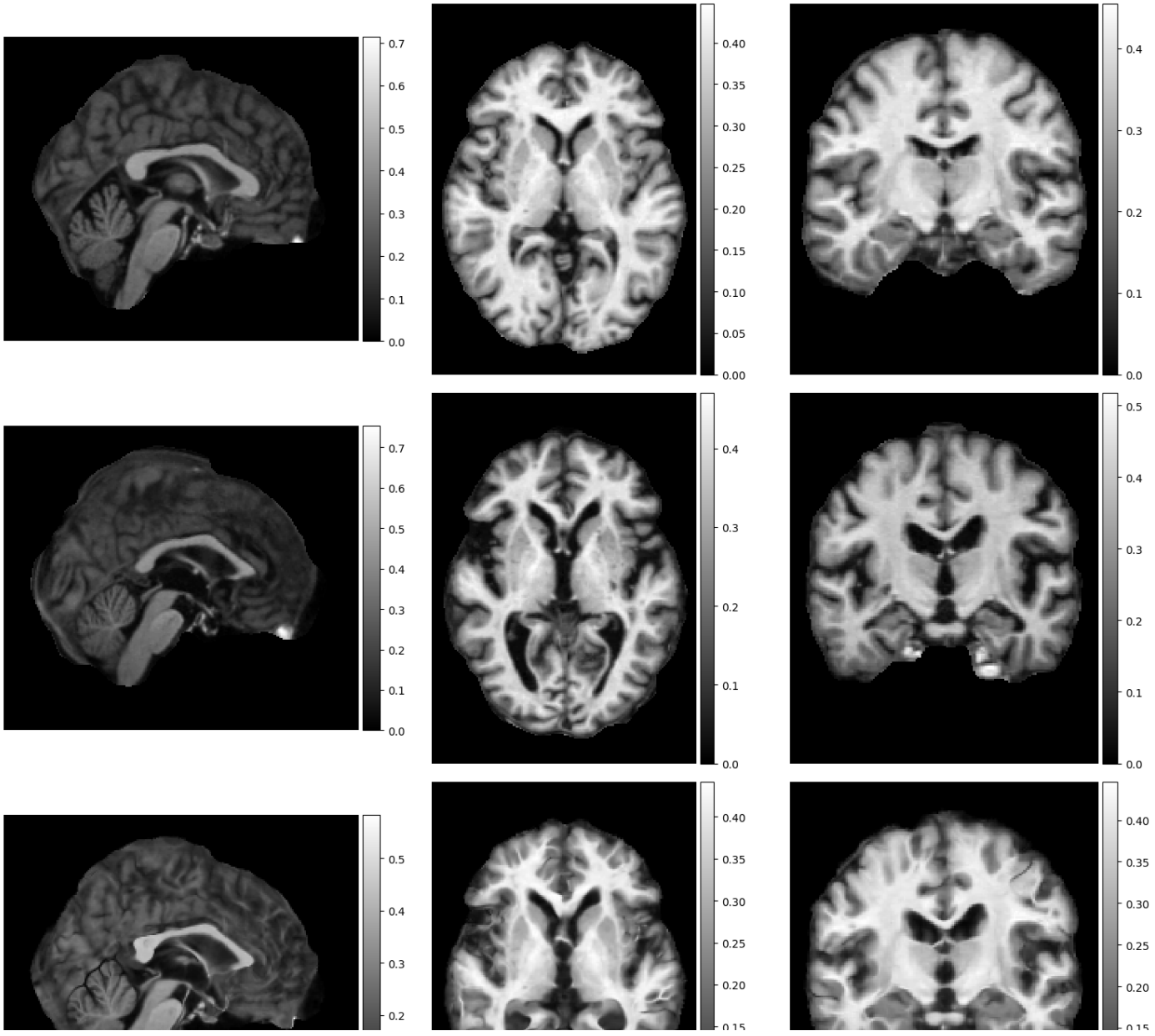


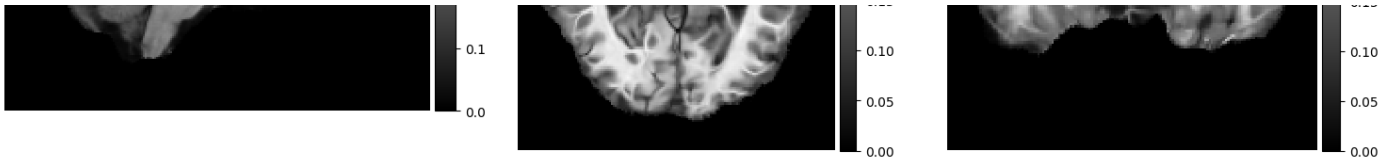
```
def data_generator_test(dir_folder,img_type,batch_size = 32):

    while True:

        files = [file for file in os.listdir(dir_folder) if file[0:3] == img_t
        #print(files)
        num_files = len(files)
        idx1 = np.random.randint(0,num_files,size = batch_size)
        files_1 = [os.path.join(dir_folder,files[i]) for i in idx1]
        moving_images = load_img(files_1)

        idx2 = np.random.randint(0,num_files,size = batch_size)
        files_2 = [os.path.join(dir_folder,files[i]) for i in idx2]
        fixed_images = load_img(files_2)

        inputs = [moving_images,fixed_images]

        img_shape = moving_images.shape[1:]
        img_shape = img_shape[:-1]

        ndims = len(img_shape)
        zero_phi = np.zeros([batch_size,*img_shape,ndims])
        outputs = [fixed_images,zero_phi]

        yield (inputs,outputs)

```

```
# showing the visulaization result for the given trained model. Task 2 for the

num_pairs = 5
test_generator = data_generator_test(test_folder,'img',batch_size = 1)

for i in range(num_pairs):

    inputs,outputs = next(test_generator)
    val_pred = vxm_model.predict(inputs)
    val_volume_1 = inputs[0][0,:,:,:,0]
    val_volume_2 = inputs[1][0,:,:,:,0]
    moved_pred = val_pred[0][0,:,:,:,0]
    print('1st moving image')
    mid_slices_moving = [np.take(val_volume_1, vol_shape[d]//2, axis=d) for d
    mid_slices_moving[1] = np.rot90(mid_slices_moving[1], 1)
    mid_slices_moving[2] = np.rot90(mid_slices_moving[2], -1)
```

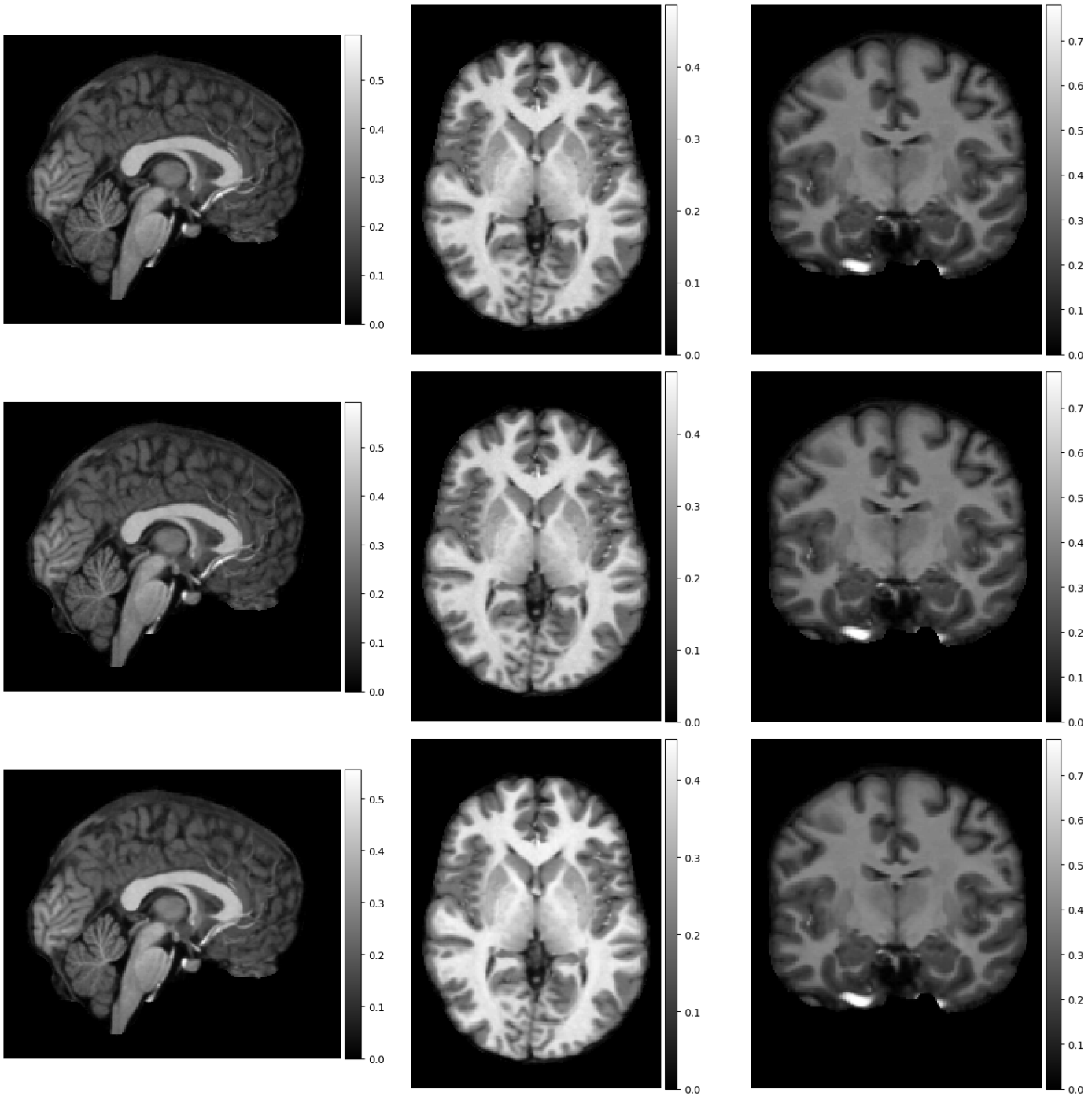
```
    print('2nd fixed image')
    mid_slices_fixed = [np.take(val_volume_2, vol_shape[d]//2, axis=d) for d i
    mid_slices_fixed[1] = np.rot90(mid_slices_fixed[1], 1)
    mid_slices_fixed[2] = np.rot90(mid_slices_fixed[2], -1)
    print('3rd predicted image')
    mid_slices_pred = [np.take(moved_pred, vol_shape[d]//2, axis=d) for d in r
    mid_slices_pred[1] = np.rot90(mid_slices_pred[1], 1)
    mid_slices_pred[2] = np.rot90(mid_slices_pred[2], -1)
    ne.plot.slices(mid_slices_moving + mid_slices_fixed + mid_slices_pred, cma

```

```
1/1 [==============================] - 0s 328ms/step
1st moving image
2nd fixed image
3rd predicted image
```



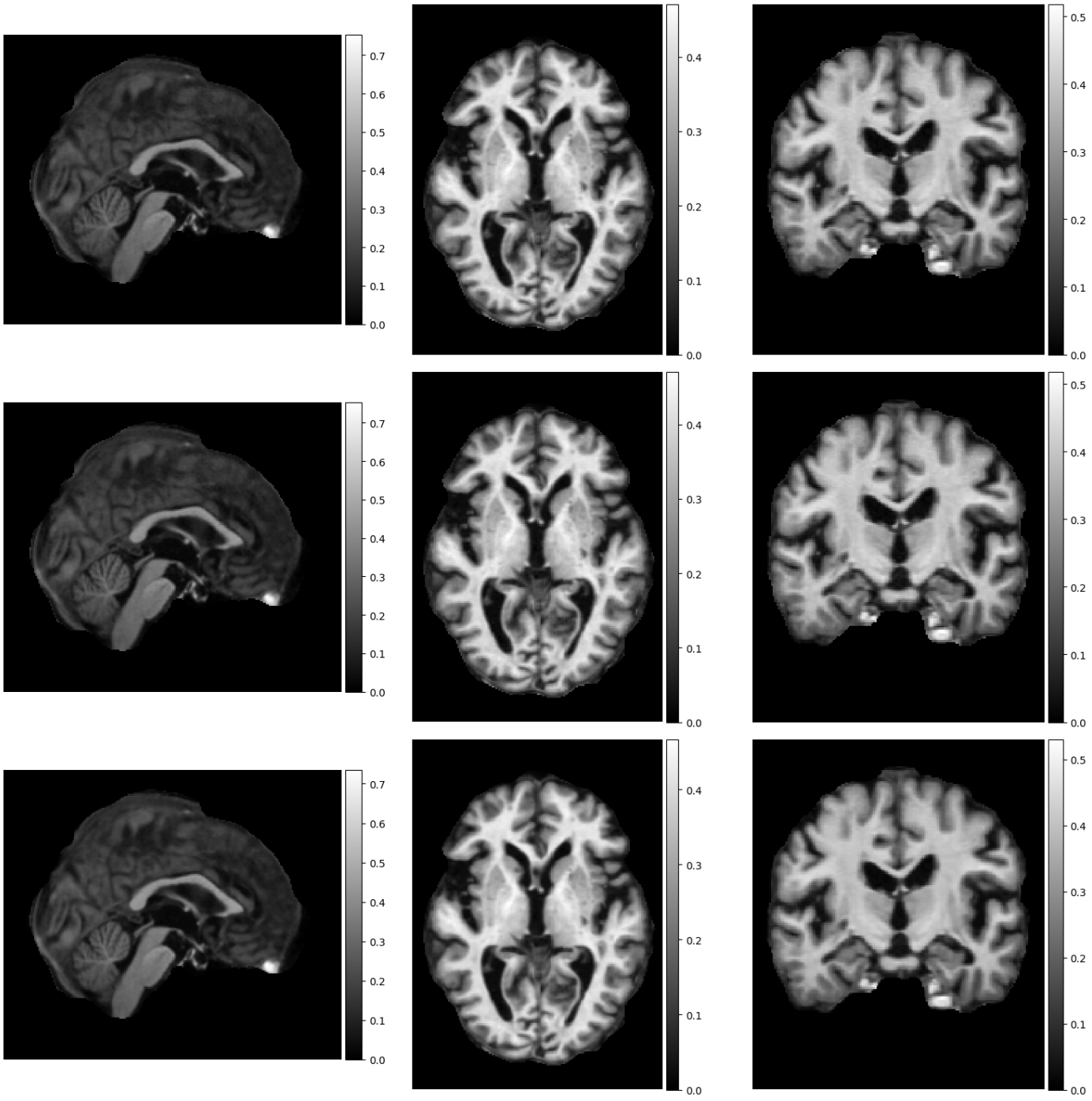


```
1/1 [==============================] - 0s 310ms/step
1st moving image
2nd fixed image
3rd predicted image
```

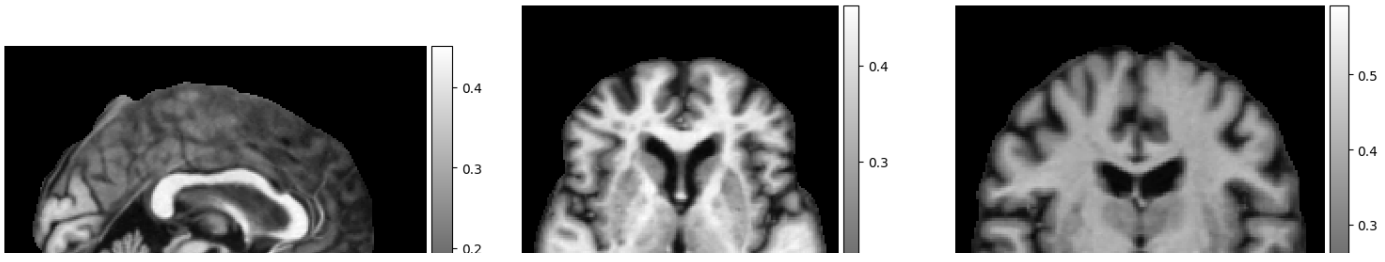


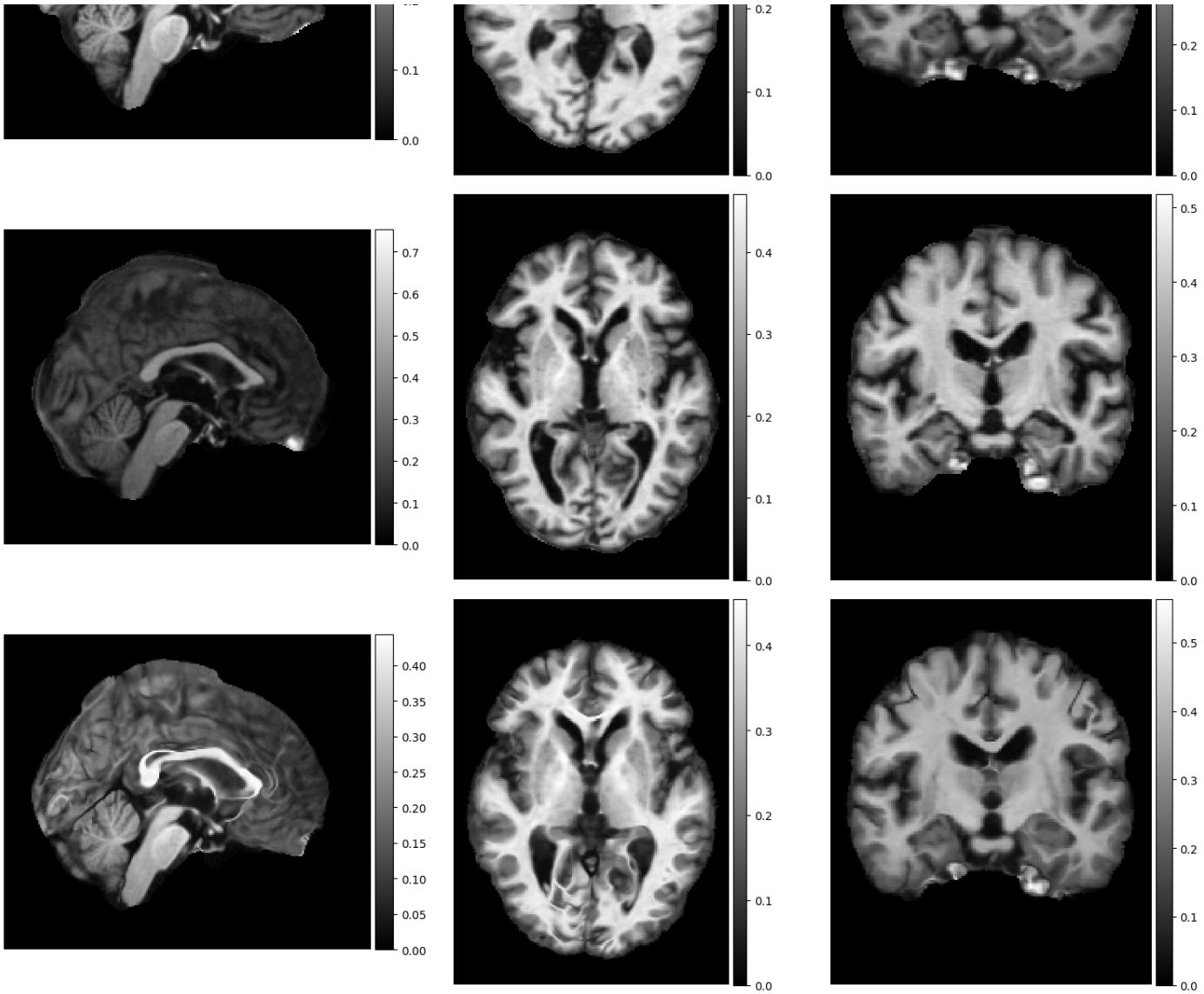
```
1/1 [==============================] - 0s 302ms/step
1st moving image
2nd fixed image
```

```
3rd predicted image
```

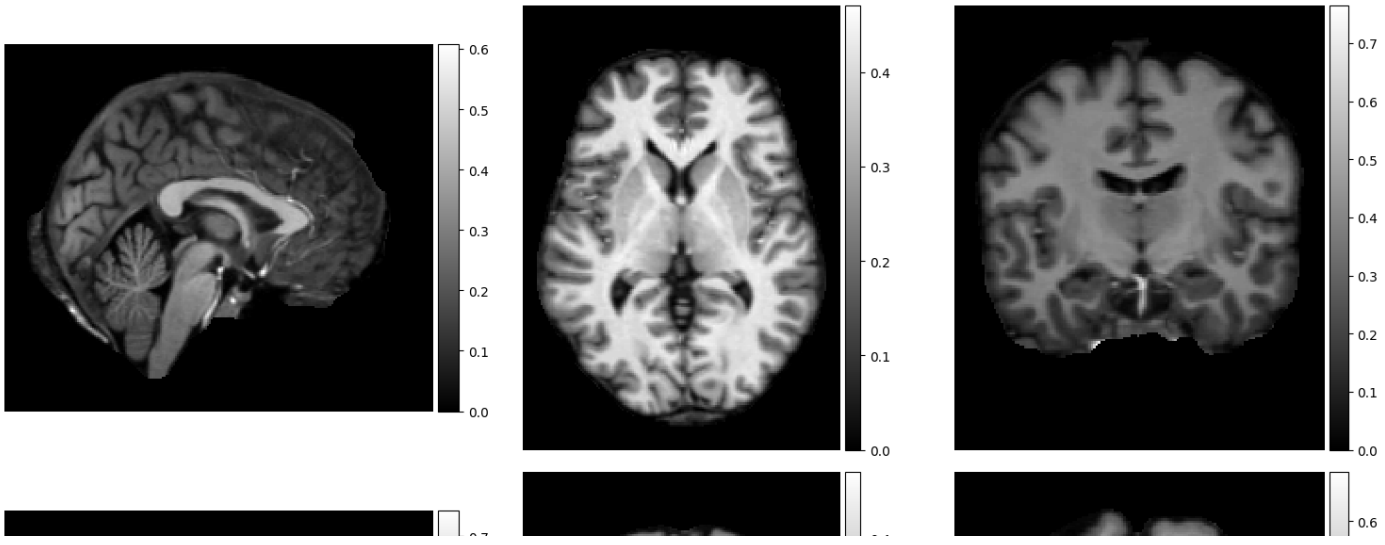


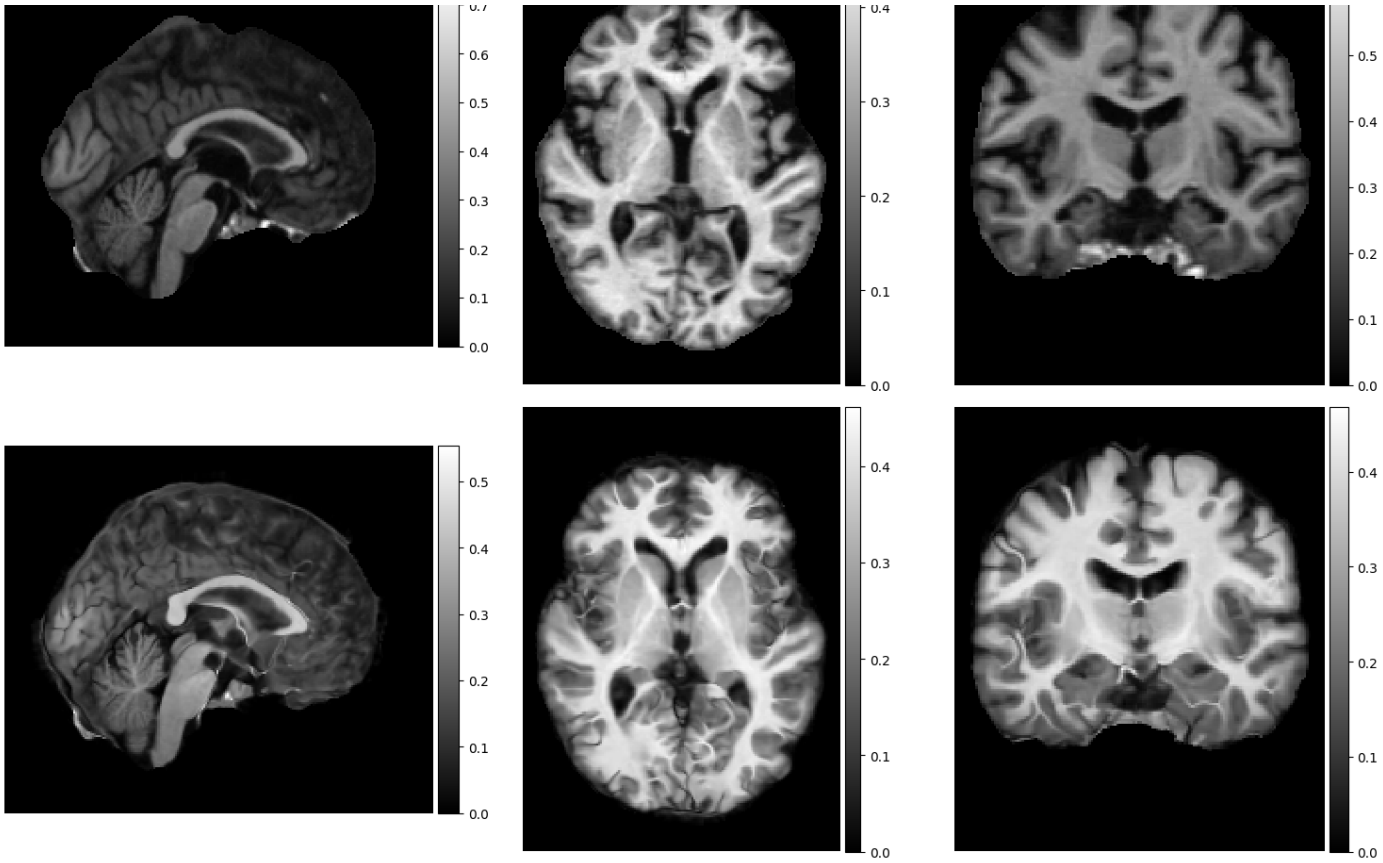
```
1/1 [==============================] - 0s 314ms/step
1st moving image
2nd fixed image
3rd predicted image
```





```
1/1 [==============================] - 0s 302ms/step
1st moving image
2nd fixed image
3rd predicted image
```




0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.4
0.3
0.2
0.1
0.0
0.5
0.4
0.3
0.2
0.1
0.0
0.5
0.4
0.3
0.2
0.1
0.0
0.4
0.3
0.2
0.1
0.0
0.4
0.3
0.2
0.1
0.0

```
import os,shutil
```

```
vxm_model.save_weights('model_mse_hw4_weights.h5')
```

```
#restarting witht he notebook. Loading the weights of the NN.
weight_path = os.path.join(dir_folder,'model_mse_hw4_weights.h5')
vxm_model.load_weights(weight_path)
```

```
# to get both img and segments , data generator
def data_generator_test_seg(dir_folder,img_type,batch_size = 1):

    files = [file for file in os.listdir(dir_folder) if file[0:3] == img_type]
    segs = ['seg'+file[3:] for file in files]

    while True:

        #return files,segs

        #print(files)
        num_files = len(files)
        idx1 = np.random.randint(0,num_files,size = batch_size)
        files_1 = [os.path.join(dir_folder,files[i]) for i in idx1]
        segs_1 = [os.path.join(dir_folder,segs[i]) for i in idx1]

        #print(len(files_1))
        #print(len(segs_1))

        moving_images = load_img(files_1)
        moving_images_seg = load_img(segs_1)

        idx2 = np.random.randint(0,num_files,size = batch_size)
        files_2 = [os.path.join(dir_folder,files[i]) for i in idx2]
        segs_2 = [os.path.join(dir_folder,segs[i]) for i in idx2]

        fixed_images = load_img(files_2)
        fixed_images_seg = load_img(segs_2)
        # print(len(files_1))
        # print(len(segs_1))

        inputs = [moving_images,fixed_images]

        img_shape = moving_images.shape[1:]
```

```
        img_shape = img_shape[:-1]

        ndims = len(img_shape)
        zero_phi = np.zeros([batch_size,*img_shape,ndims])
        outputs = [fixed_images,zero_phi]

        both_segs = [moving_images_seg,fixed_images_seg]

        return inputs,outputs,both_segs


```

```
in_sample , out_sample, sample_seg = data_generator_test_seg(test_folder,'img'
```

```
1
1
```

```
seg_sample[0].shape
```

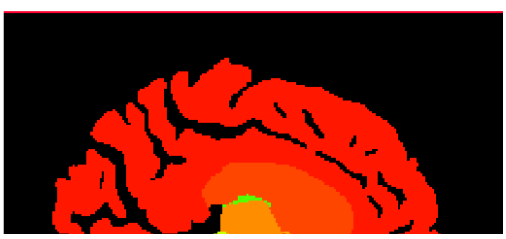
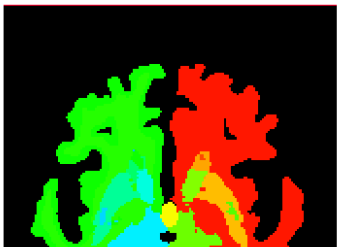
```
(1, 160, 192, 224, 1)
```

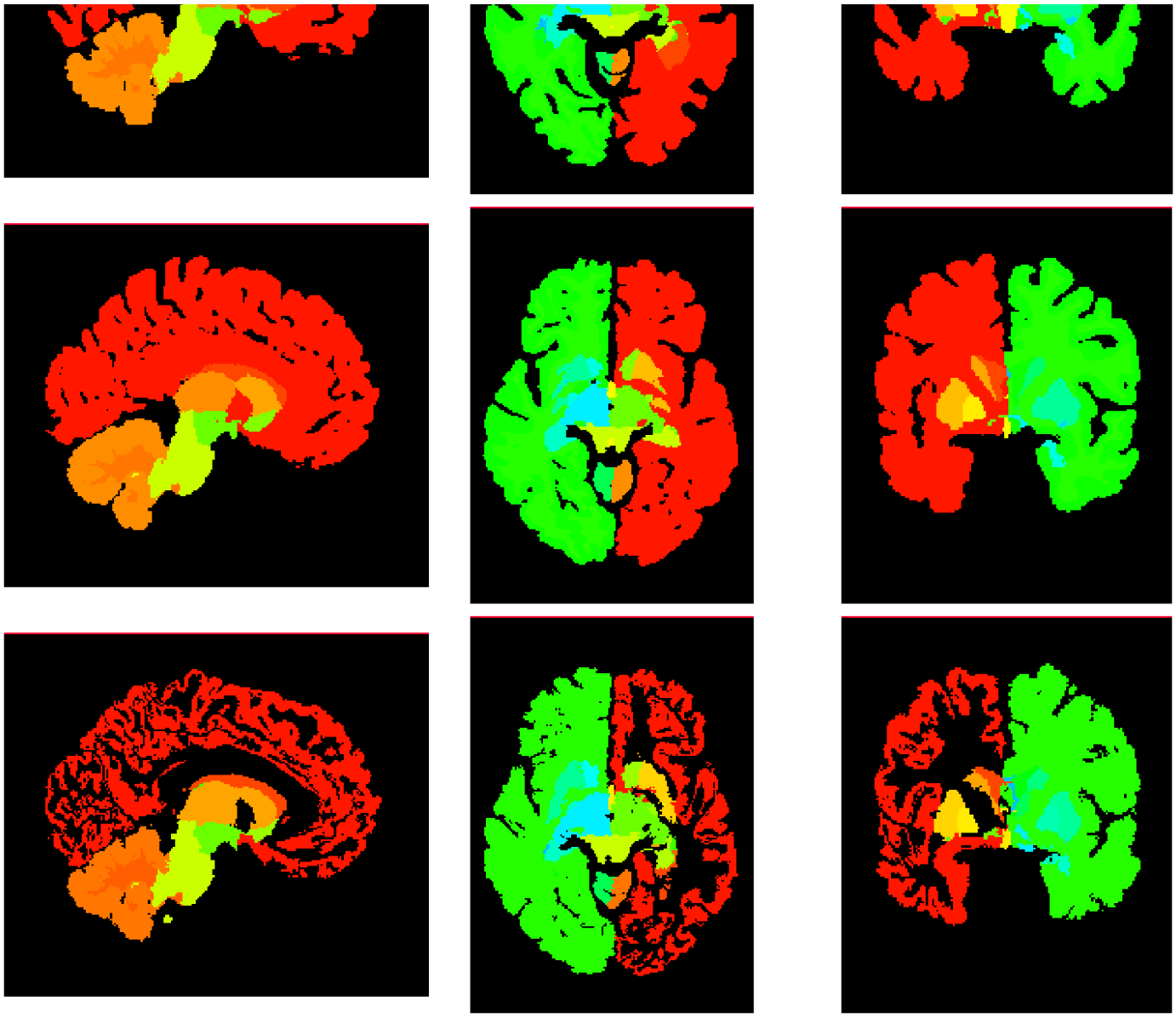
```
from pystrum.pytools.plot import jitter
import matplotlib

[ccmap, scrambled_cmap] = jitter(255, nargout=2)
scrambled_cmap[0, :] = np.array([0, 0, 0, 1])
ccmap = matplotlib.colors.ListedColormap(scrambled_cmap)
```

```
# showing the visulaization result for the given trained model. Task 2 for the

num_pairs = 5
test_generator = data_generator_test_seg(test_folder,'img',batch_size = 1)

for i in range(num_pairs):

    inputs,outputs,segs = data_generator_test_seg(test_folder,'img',batch_size
    val_pred = vxm_model.predict(inputs)
    moved_pred = val_pred[0][0,:,:,:,0]
    pred_warp  = val_pred[1]


    warp_model = vxm.networks.Transform(vol_shape,interp_method = 'nearest')
    warped_seg = warp_model.predict([segs[0],pred_warp])
```

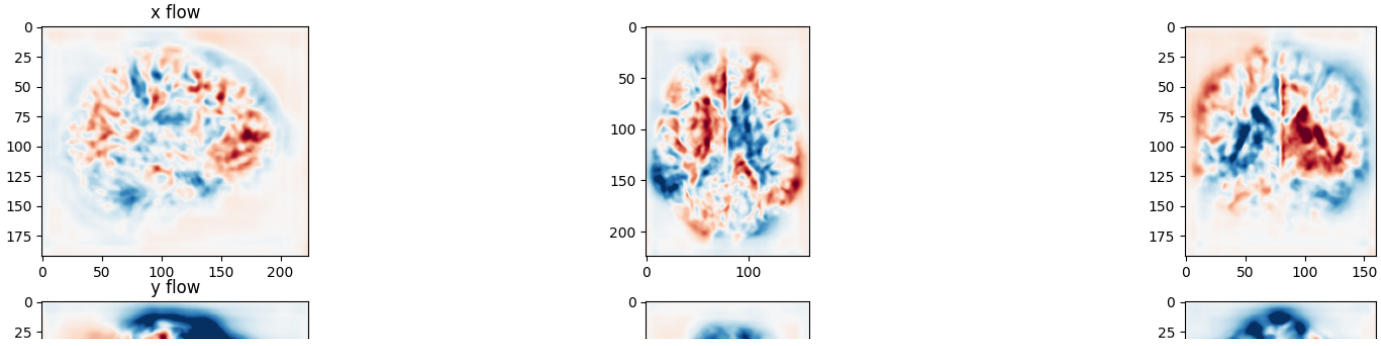
```
  print('warped segmentation')

  print('1st moving image')
  mid_slices_moving = [np.take(segs[0].squeeze(), vol_shape[d]//1.8, axis=d)
  mid_slices_moving[1] = np.rot90(mid_slices_moving[1], 1)
  mid_slices_moving[2] = np.rot90(mid_slices_moving[2], -1)
  print('2nd fixed image')
  mid_slices_fixed = [np.take(segs[1].squeeze(), vol_shape[d]//1.8, axis=d)
  mid_slices_fixed[1] = np.rot90(mid_slices_fixed[1], 1)
  mid_slices_fixed[2] = np.rot90(mid_slices_fixed[2], -1)
  print('3rd predicted image')
  mid_slices_pred = [np.take(warped_seg.squeeze(), vol_shape[d]//1.8, axis=d
  mid_slices_pred[1] = np.rot90(mid_slices_pred[1], 1)
  mid_slices_pred[2] = np.rot90(mid_slices_pred[2], -1)
  slices = mid_slices_moving + mid_slices_fixed + mid_slices_pred
  for si, slc  in enumerate(slices):
    slices[si][0] = 255
  ne.plot.slices(slices, cmaps = [ccmap], grid=[3,3]);

  flow = val_pred[1][0, :, :, :, :]
  flow_sd = np.std(flow)
  v_args = dict(cmap = 'RdBu', vmin = -flow_sd, vmax = +flow_sd)
  fig, m_axs = plt.subplots(3, 3, figsize = (20, 10))
  for i, (ax1, ax2, ax3) in enumerate(m_axs):
      ax1.imshow(np.mean(flow[:, :, :, i], 0), **v_args)
      ax1.set_title('xyz'[i]+' flow')
      ax2.imshow(np.rot90(np.mean(flow[:, :, :, i], 1),1), **v_args)
      ax3.imshow(np.rot90(np.mean(flow[:, :, :, i], 2),-1), **v_args)

```

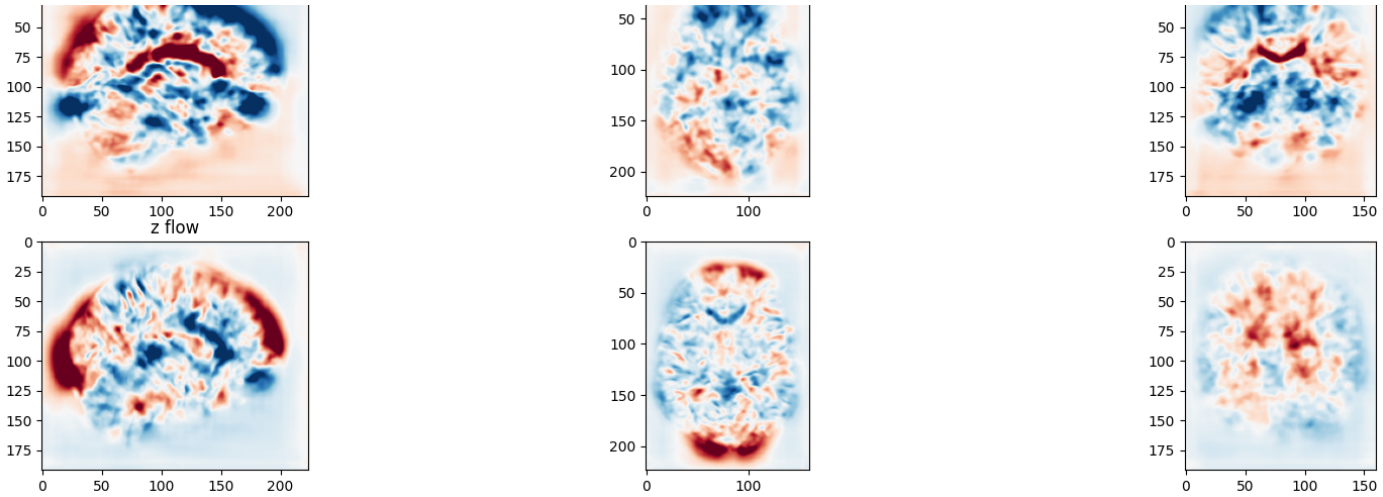
```
1
1
1
1
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 241ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```

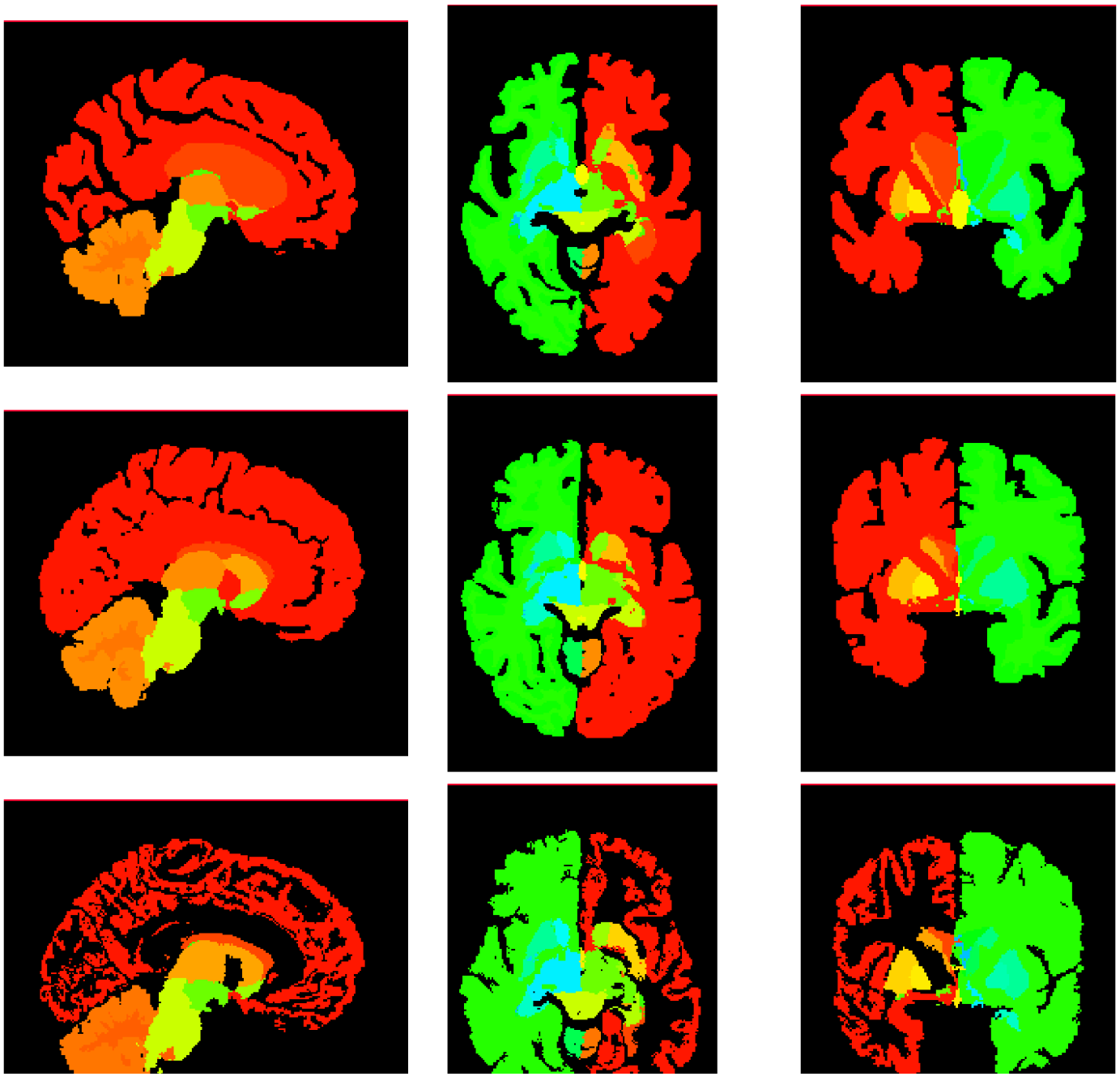



```
1
1
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 224ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```

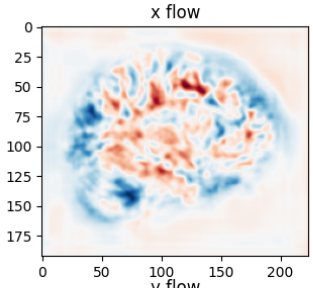


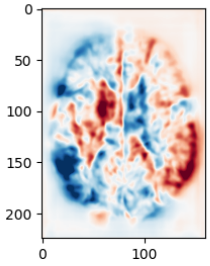
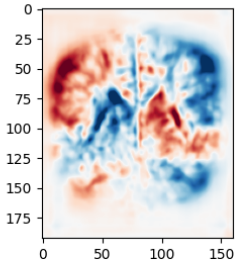
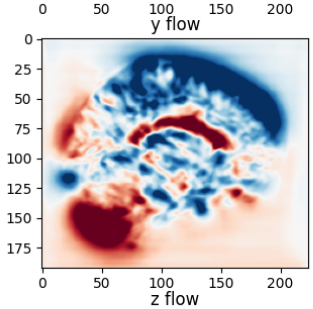

z flow

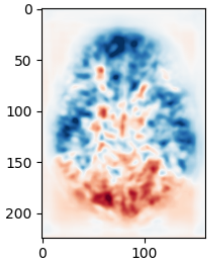
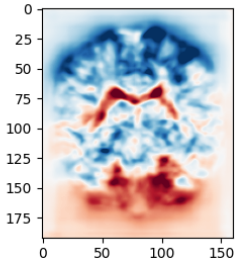
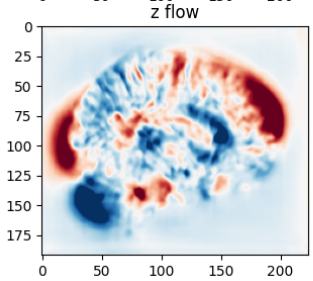


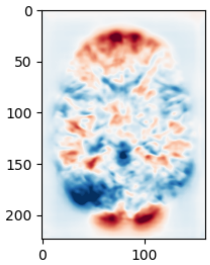
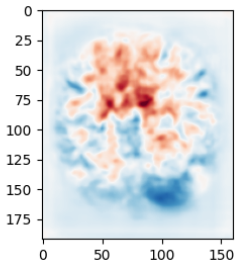
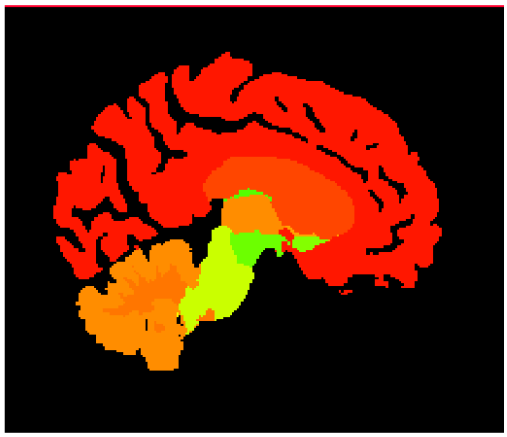


```
1
1
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 225ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```

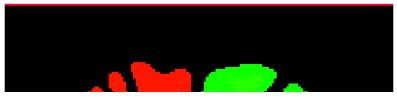


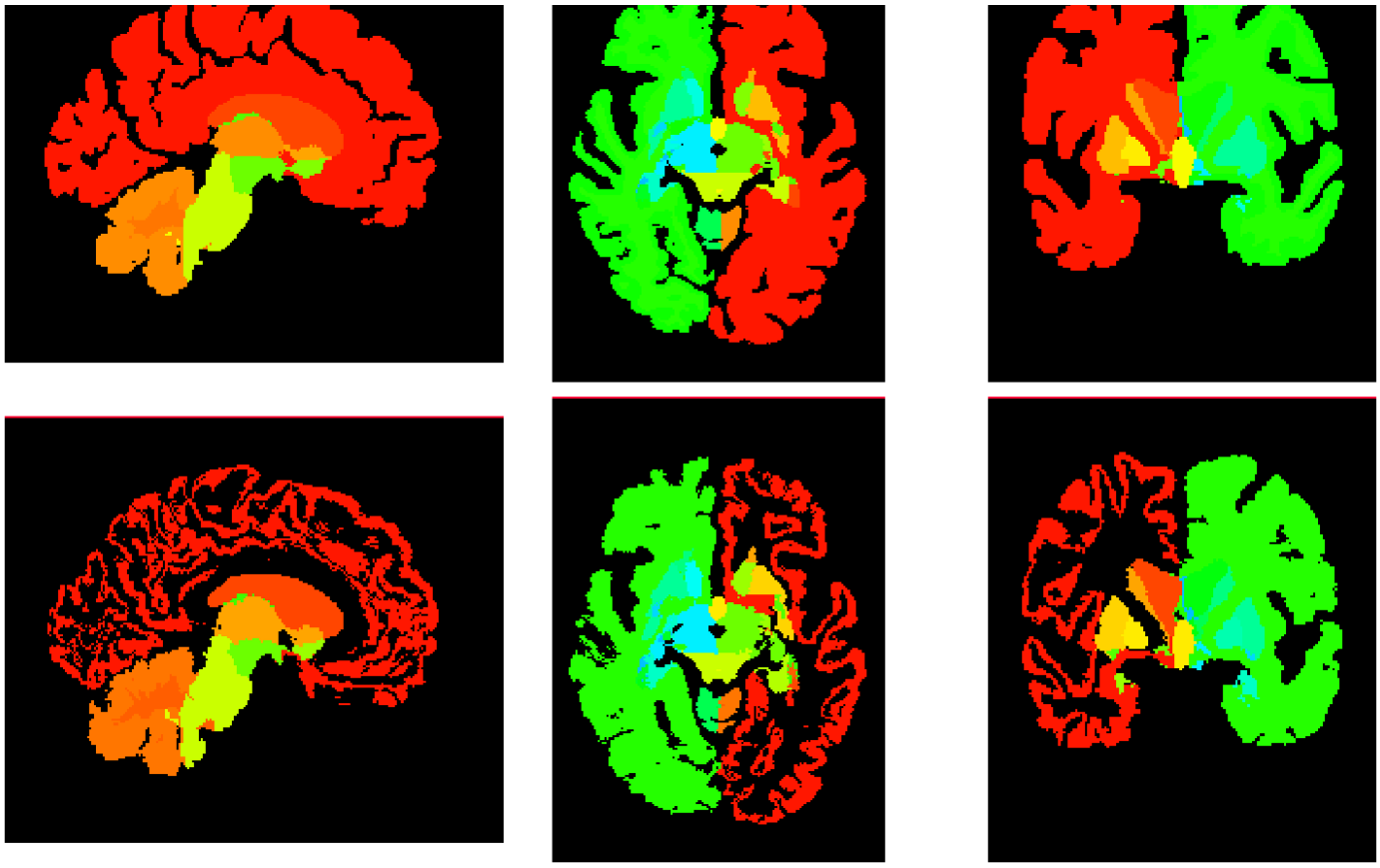




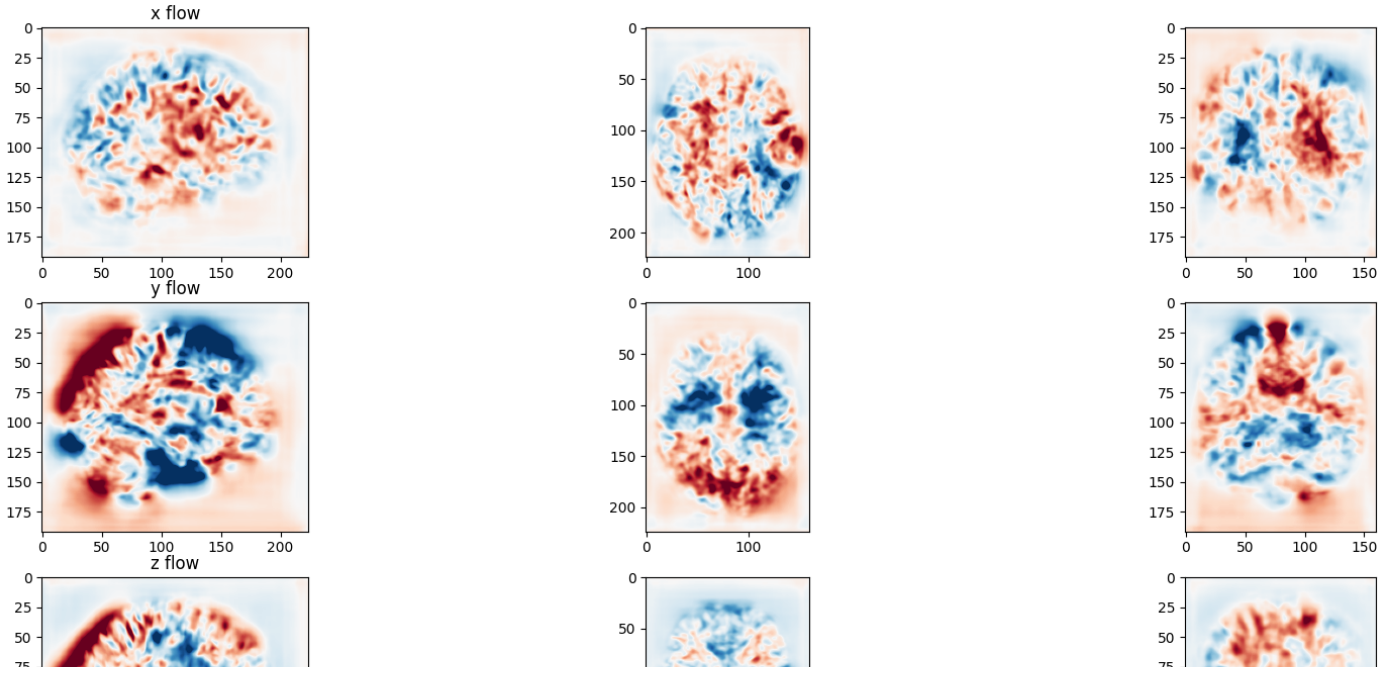




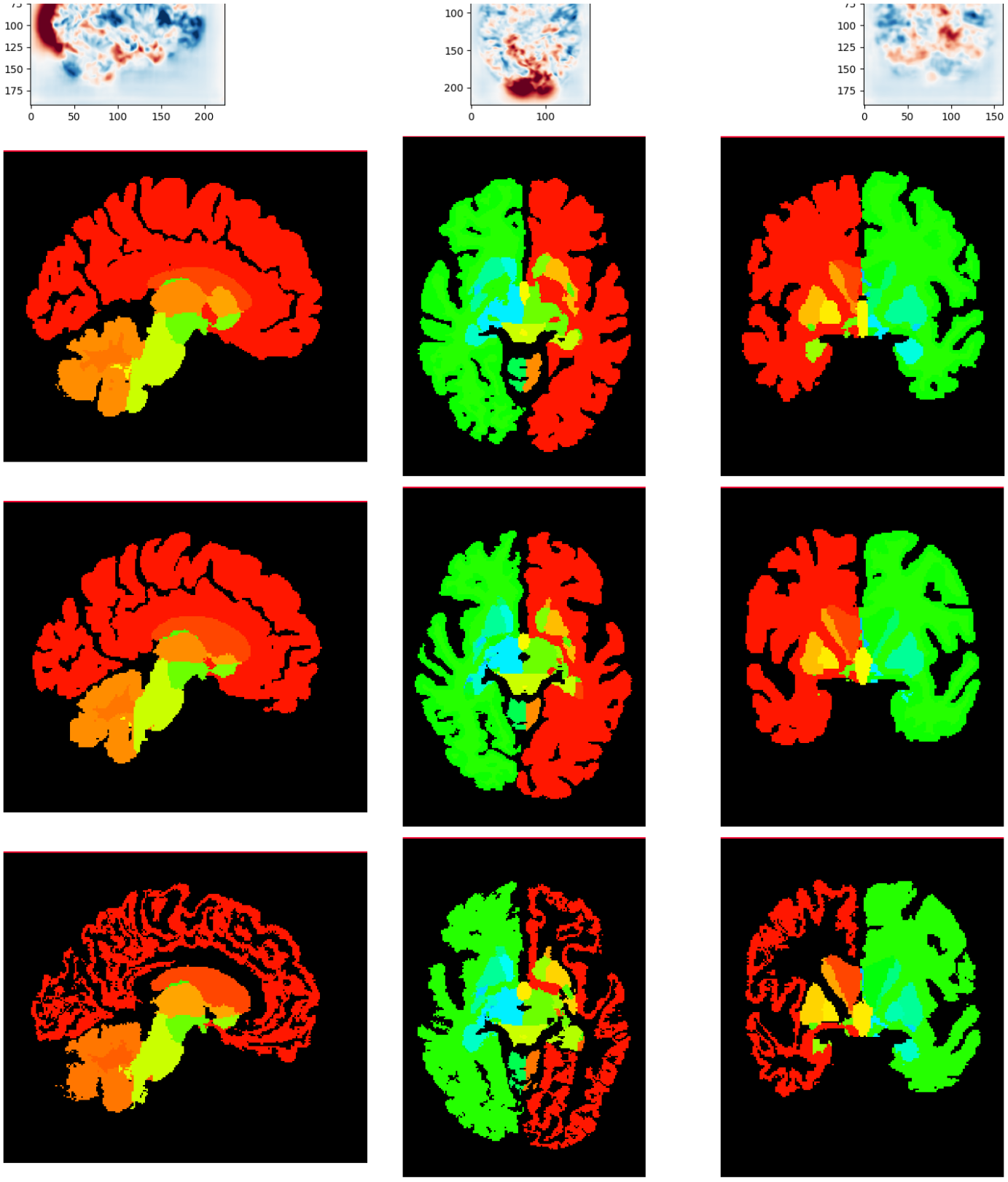




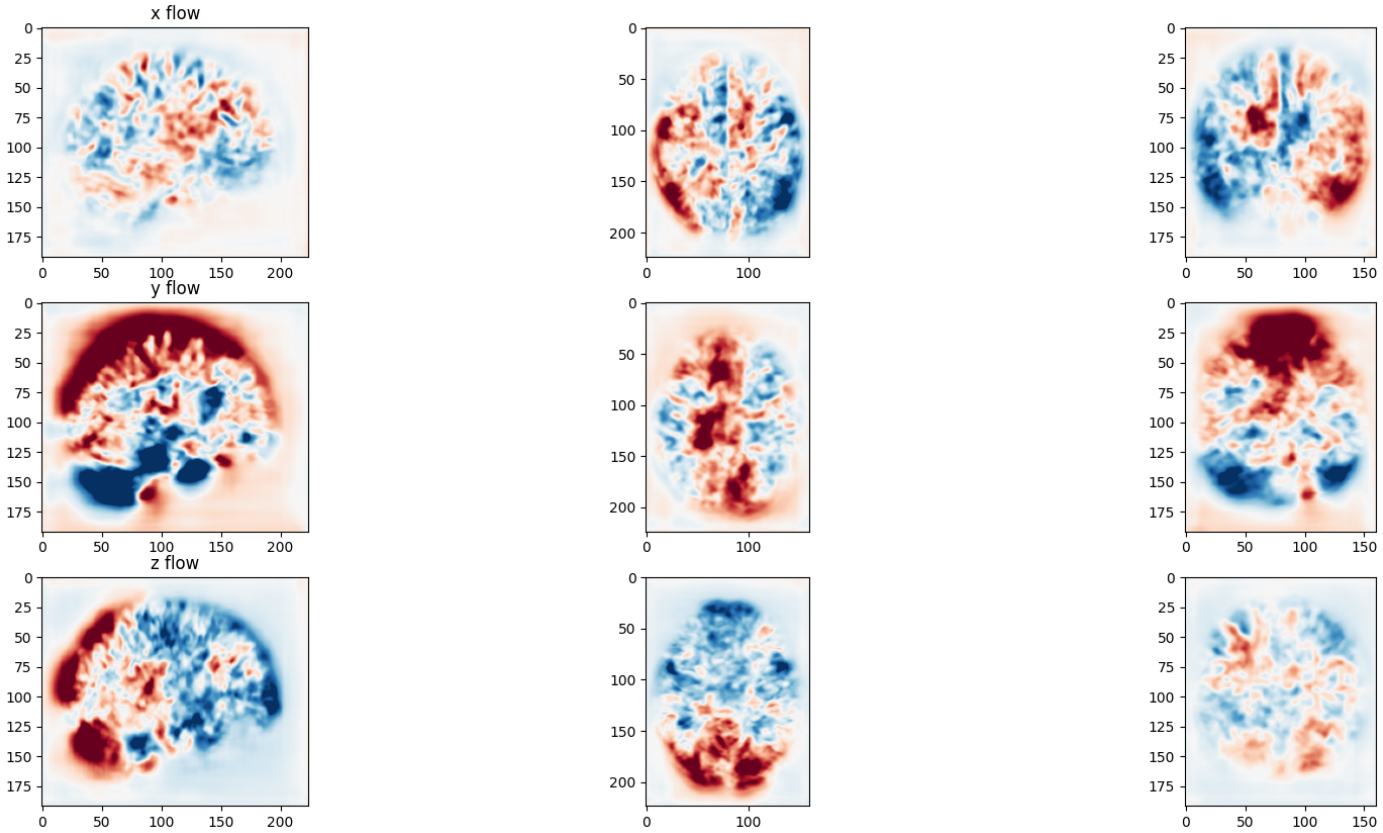
```
1
1
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 233ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```

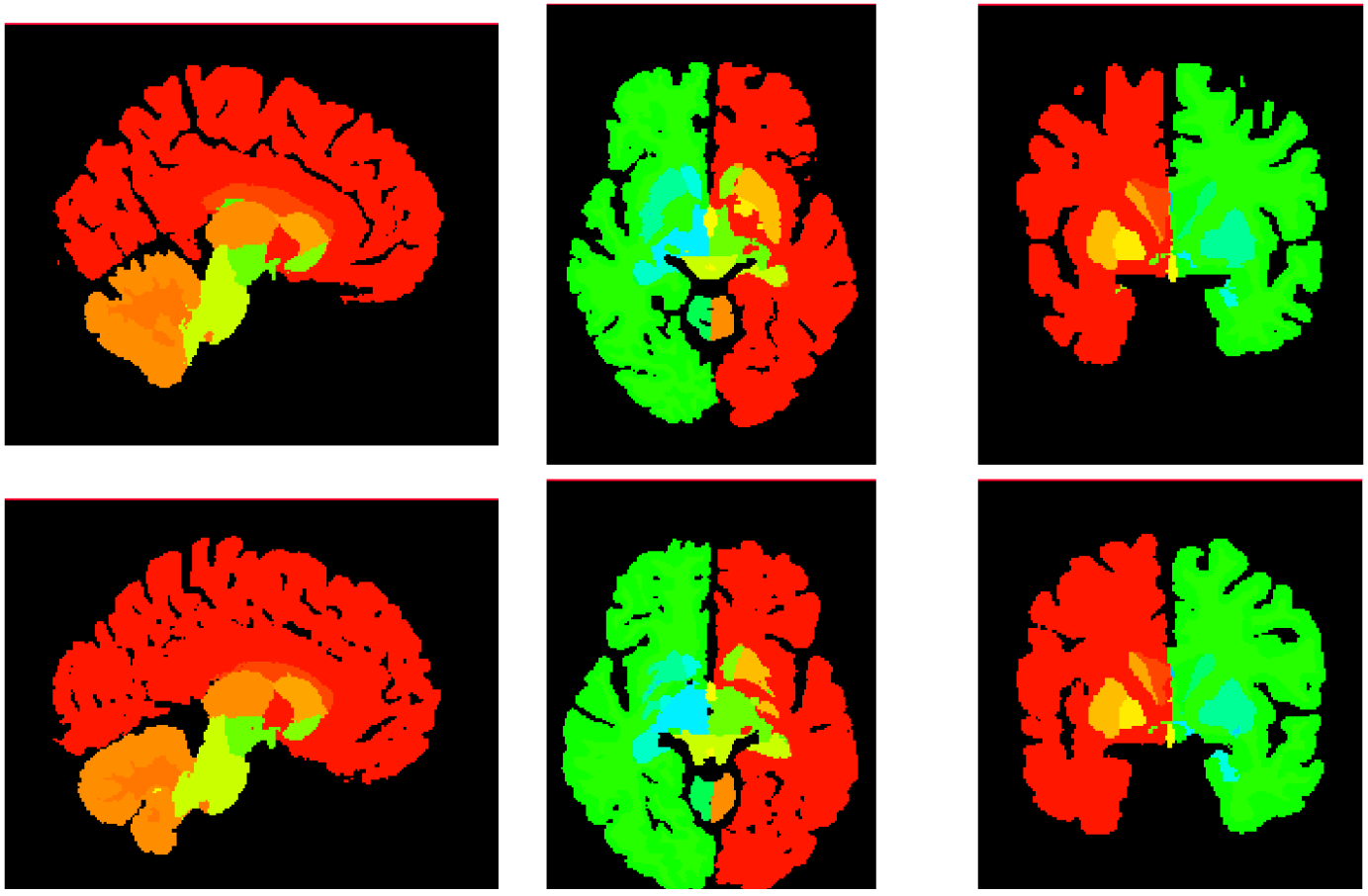


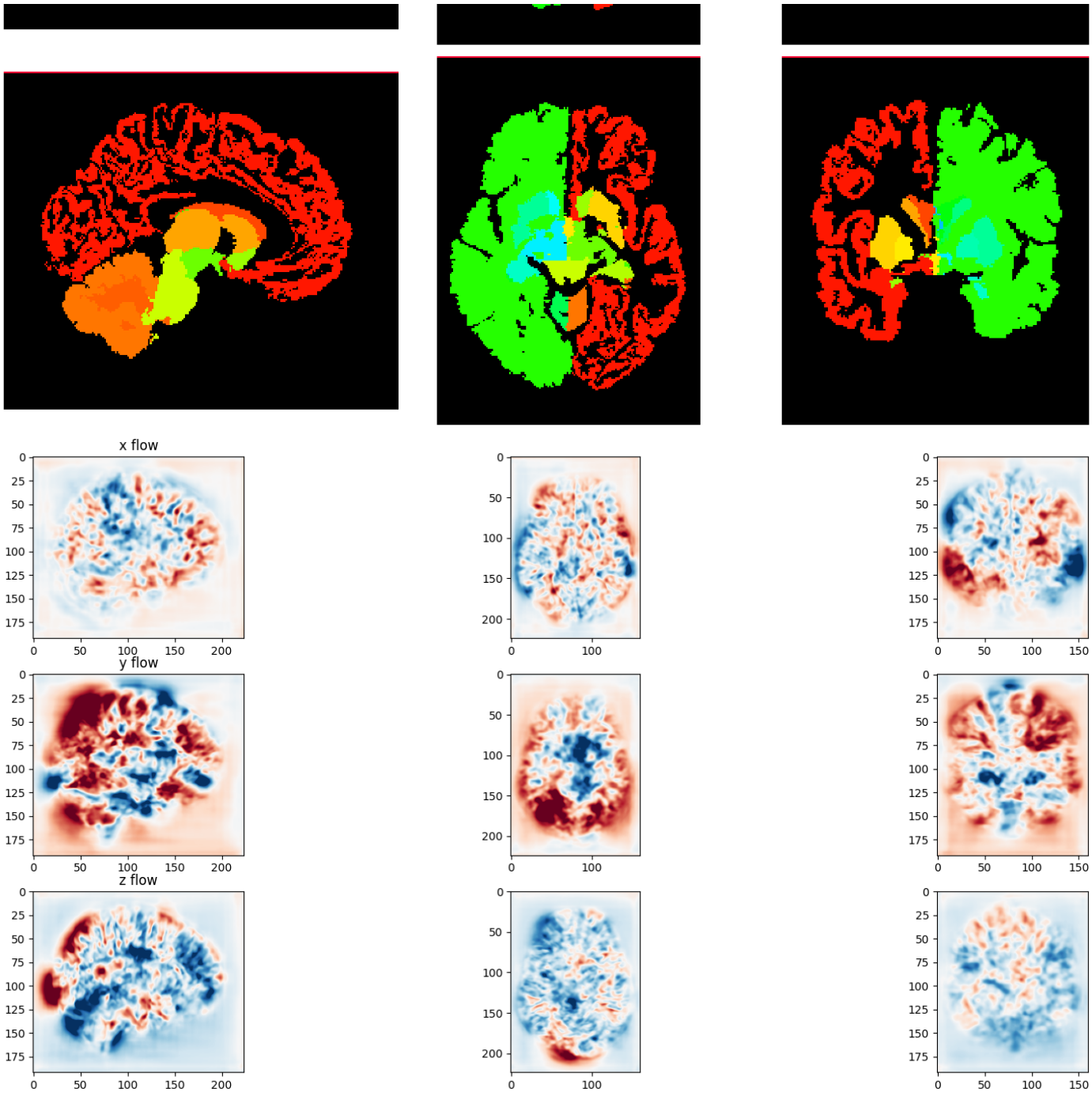


```
1
1
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 223ms/step
warped segmentation
1st moving image
```

2nd fixed image
3rd predicted image






x flow
y flow
z flow

```
def dice_coef(segmentation1, segmentation2):

    unique_labels1 = np.unique(segmentation1)
    unique_labels2 = np.unique(segmentation2)

    # Exclude background label (usually 0) from calculations
    unique_labels1 = unique_labels1[unique_labels1 != 0]
    unique_labels2 = unique_labels2[unique_labels2 != 0]

    dice_scores = []

    for label1 in unique_labels1:
            # Create binary masks for each pair of labels
          mask1 = (segmentation1 == label1).astype(int)
          mask2 = (segmentation2 == label1).astype(int)

            # Calculate Dice coefficient for the pair
          intersection = np.sum(mask1 * mask2)
          total_voxels = np.sum(mask1) + np.sum(mask2)
          dice_coefficient = 2.0 * intersection / total_voxels if total_voxels

          dice_scores.append(dice_coefficient)

    # Calculate average Dice score
    average_dice_score = np.mean(dice_scores) if len(dice_scores) > 0 else 0.0

    return average_dice_score
```

```
from scipy.spatial.distance import directed_hausdorff

def hausdorff_distance_fast(segmentation1, segmentation2):

    unique_labels1 = np.unique(segmentation1)
    # Exclude background label (usually 0) from calculations
    unique_labels1 = unique_labels1[unique_labels1 != 0]

    all_distances = []

    for label1 in unique_labels1:

        # Create binary masks for each pair of labels
        mask1 = (segmentation1 == label1).astype(int)
        mask2 = (segmentation2 == label1).astype(int)

        # Find the directed Hausdorff distance for each pair
        distance1 = directed_hausdorff(mask1, mask2)[0]
        distance2 = directed_hausdorff(mask2, mask1)[0]

        # Take the maximum distance for the pair
        max_distance = max(distance1, distance2)
        all_distances.append(max_distance)

    # Calculate the overall Hausdorff distance
    hausdorff_distance = np.mean(all_distances) if len(all_distances) > 0 else

    return hausdorff_distance

```



```
num_pairs = 10
dice_score = []
hausdroff_distance = []
for i in range(num_pairs):

    inputs,outputs,segs = data_generator_test_seg(test_folder,'img',batch_size
    val_pred = vxm_model.predict(inputs)
    moved_pred = val_pred[0][0,:,:,:,0]
    pred_warp  = val_pred[1]


```

```
    warp_model = vxm.networks.Transform(vol_shape,interp_method = 'nearest')
    warped_seg = warp_model.predict([segs[0],pred_warp])

    dice_score.append(dice_coef(segs[1].squeeze(),warped_seg.squeeze()))
    print(dice_score[i])
    #hausdroff_distance.append(hausdorff_distance_fast(segs[1].squeeze(),warpe
    #print(hausdroff_distance[i])

dice_score = np.array(dice_score)
mean_dice_score = np.mean(dice_score)
#hausdroff_distance = np.array(hausdroff_distance)
#mean_hausdroff_distance = np.mean(hausdroff_distance)

print(f'The mean dice score is {mean_dice_score}')
#print(f'The mean hausdroff distance is {mean_hausdroff_distance}')
```

```
1
1
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 251ms/step
0.7504003748490484
1
1
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 233ms/step
0.7706752502161418
1
1
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 260ms/step
0.7922613666186606
1
1
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 226ms/step
0.6354750623428762
1
1
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 223ms/step
0.7473721509572837
1
1
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 270ms/step
0.6559308146449379
1
1
```

```
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 229ms/step
0.7712127664471303
1
1
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 226ms/step
0.7431253584921385
1
1
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 252ms/step
0.9948590825822994
1
1
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 222ms/step
0.7618312123709834
The mean dice score is 0.76231434395215
```

```
pip install voxelmorph
```

```
Collecting voxelmorph
  Downloading voxelmorph-0.2-py3-none-any.whl (54 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.2/54.2 kB 1.5 MB/s eta 0:00:0
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: nibabel in /usr/local/lib/python3.10/dist-packa
Collecting neurite>=0.2 (from voxelmorph)
  Downloading neurite-0.2-py3-none-any.whl (108 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 108.9/108.9 kB 5.9 MB/s eta 0:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
Collecting pystrum>=0.2 (from neurite>=0.2->voxelmorph)
  Downloading pystrum-0.4.tar.gz (17 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.1
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Building wheels for collected packages: pystrum
  Building wheel for pystrum (setup.py) ... done
  Created wheel for pystrum: filename=pystrum-0.4-py3-none-any.whl size=19532
  Stored in directory: /root/.cache/pip/wheels/aa/08/d0/914025beb5a12a855b8aa
Successfully built pystrum
Installing collected packages: pystrum, neurite, voxelmorph
Successfully installed neurite-0.2 pystrum-0.4 voxelmorph-0.2
```

This one is for NCC LOSS

```
from numba import cuda
device = cuda.get_current_device()
device.reset()
```

```
!nvidia-smi
```

```
Sat Dec 16 03:56:11 2023
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version:
|-----------------------------------------+----------------------+----------------------
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Un
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  C
|                                         |                      |
|=========================================+======================+======================
|   0  NVIDIA A100-SXM4-40GB          Off | 00000000:00:04.0 Off |
| N/A   30C    P0              44W / 400W |      5MiB / 40960MiB |      0%
|                                         |                      |
+-----------------------------------------+----------------------+----------------------

+---------------------------------------------------------------------------------------+
| Processes:
|  GPU   GI   CI        PID   Type   Process name                             G
|        ID   ID                                                              U
|=======================================================================================
|  No running processes found
+---------------------------------------------------------------------------------------+
```

```
import os
import nibabel as nib
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import voxelmorph as vxm
import neurite as ne
```

```
dir_folder = os.path.join('/content/drive','My Drive','hw4','Dataset_HW4')
```

```
train_folder = os.path.join(dir_folder,'train')
test_folder = os.path.join(dir_folder,'test')
```

```
def load_img(files_path):

    batch_size = len(files_path)
    temp_file = nib.load(files_path[0])
    img_shape = temp_file.shape
    img_array = np.zeros([batch_size,*img_shape])

    for i,files in enumerate(files_path):

        img = nib.load(files)
        img_array[i] = img.get_fdata()

    return np.expand_dims(img_array,axis = -1)
```

```
def data_generator(dir_folder,batch_size = 32):

    while True:

        files = os.listdir(dir_folder)
        #print(files)
        num_files = len(files)
        idx1 = np.random.randint(0,num_files,size = batch_size)
        files_1 = [os.path.join(dir_folder,files[i]) for i in idx1]
        moving_images = load_img(files_1)

        idx2 = np.random.randint(0,num_files,size = batch_size)
        files_2 = [os.path.join(dir_folder,files[i]) for i in idx2]
        fixed_images = load_img(files_2)

        inputs = [moving_images,fixed_images]

        img_shape = moving_images.shape[1:]
        img_shape = img_shape[:-1]

        ndims = len(img_shape)
        zero_phi = np.zeros([batch_size,*img_shape,ndims])
        outputs = [fixed_images,zero_phi]

        yield (inputs,outputs)

    # while True:

    #     idx1 = np.random.randint(0,num_files,size = batch_size)

    #     moving_images = load_img(
```

```
#compile model

vol_shape = (160, 192, 224)
nb_features = [
    [16, 32, 32, 32],
    [32, 32, 32, 32, 32, 16, 16]
]
vxm_model = vxm.networks.VxmDense(vol_shape, nb_features, int_steps=0)
ncc_loss = vxm.tf.losses.NCC(win = 9)
losses = [ncc_loss.loss,vxm.losses.Grad('l2').loss]
loss_weights = [1,0.01]

vxm_model.compile(optimizer = tf.keras.optimizers.legacy.Adam(learning_rate =
```

```
1
```

```
train_generator = data_generator(train_folder,batch_size = 1)
```

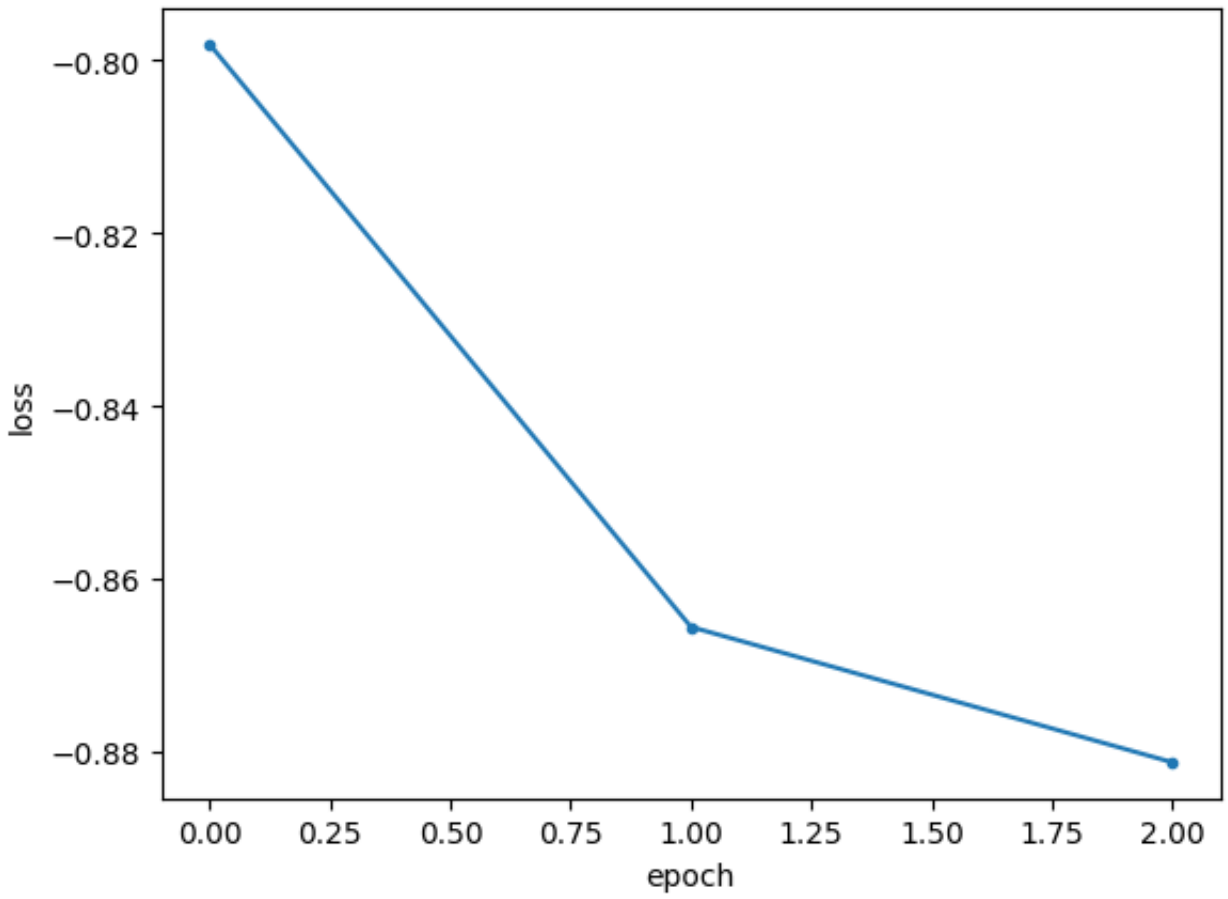
```
# training the model
hist = vxm_model.fit(train_generator,epochs = 3, steps_per_epoch = 800, verbos
```

```
Epoch 1/3
800/800 - 515s - loss: -7.9815e-01 - vxm_dense_transformer_loss: -7.9920e-01 -
Epoch 2/3
800/800 - 409s - loss: -8.6557e-01 - vxm_dense_transformer_loss: -8.6713e-01 -
Epoch 3/3
800/800 - 409s - loss: -8.8124e-01 - vxm_dense_transformer_loss: -8.8296e-01 -
```

```
1
```

```
import matplotlib.pyplot as plt

def plot_history(hist, loss_name='loss'):
    # Simple function to plot training history.
    plt.figure()
    plt.plot(hist.epoch, hist.history[loss_name], '.-')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.show()
```

```
1 plot_history(hist)
```

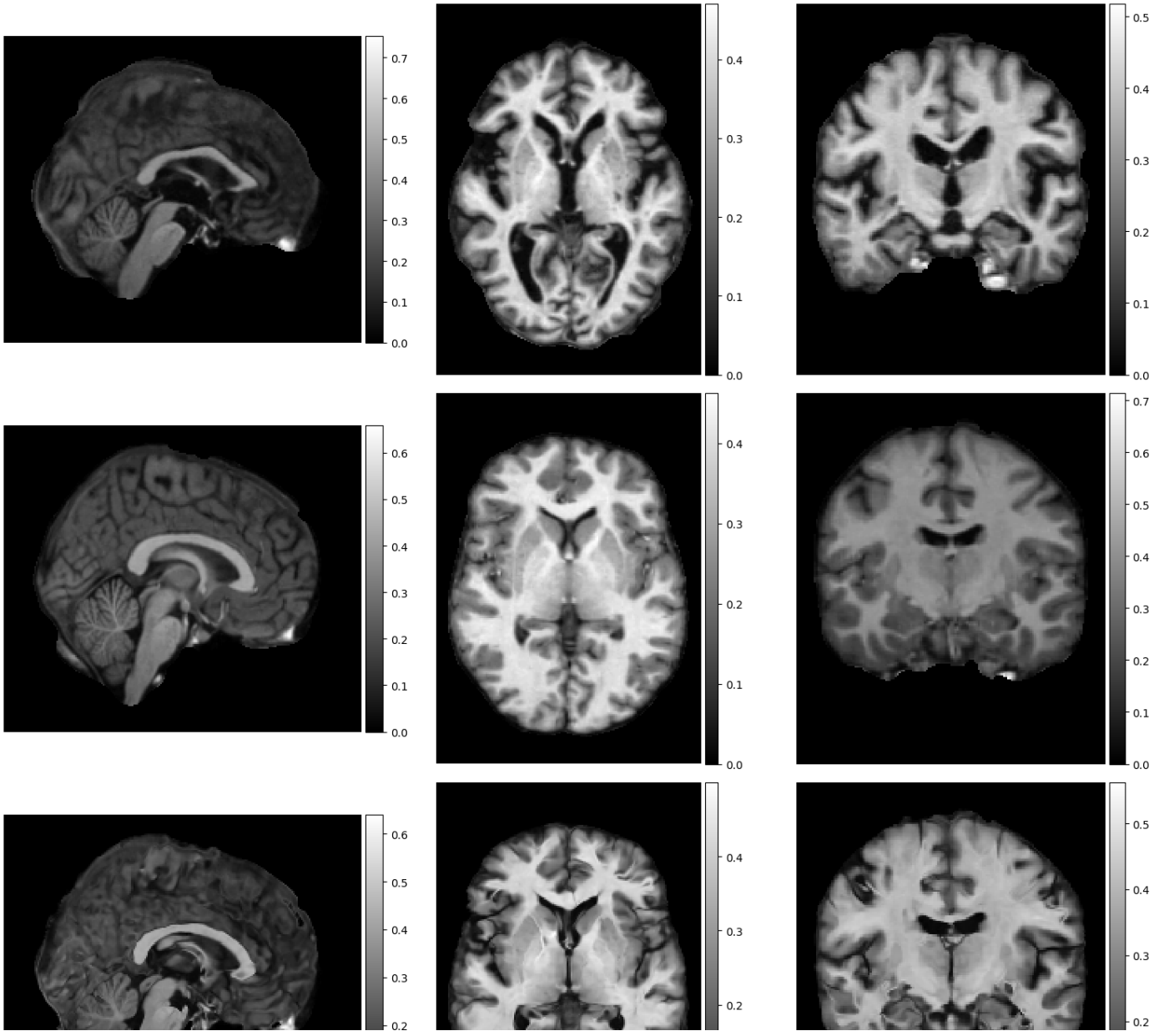


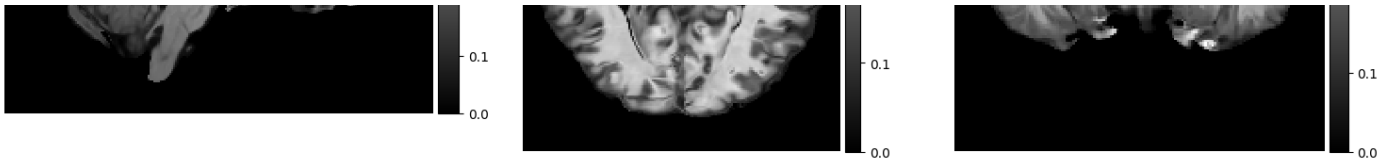
```
def data_generator_test(dir_folder,img_type,batch_size = 32):

    while True:

        files = [file for file in os.listdir(dir_folder) if file[0:3] == img_t
        #print(files)
        num_files = len(files)
        idx1 = np.random.randint(0,num_files,size = batch_size)
        files_1 = [os.path.join(dir_folder,files[i]) for i in idx1]
        moving_images = load_img(files_1)

        idx2 = np.random.randint(0,num_files,size = batch_size)
        files_2 = [os.path.join(dir_folder,files[i]) for i in idx2]
        fixed_images = load_img(files_2)

        inputs = [moving_images,fixed_images]

        img_shape = moving_images.shape[1:]
        img_shape = img_shape[:-1]

        ndims = len(img_shape)
        zero_phi = np.zeros([batch_size,*img_shape,ndims])
        outputs = [fixed_images,zero_phi]

        yield (inputs,outputs)

```

```
# showing the visulaization result for the given trained model. Task 2 for the

num_pairs = 5
test_generator = data_generator_test(test_folder,'img',batch_size = 1)

for i in range(num_pairs):

    inputs,outputs = next(test_generator)
    val_pred = vxm_model.predict(inputs)
    val_volume_1 = inputs[0][0,:,:,:,0]
    val_volume_2 = inputs[1][0,:,:,:,0]
    moved_pred = val_pred[0][0,:,:,:,0]
    print('1st moving image')
    mid_slices_moving = [np.take(val_volume_1, vol_shape[d]//2, axis=d) for d
    mid_slices_moving[1] = np.rot90(mid_slices_moving[1], 1)
    mid_slices_moving[2] = np.rot90(mid_slices_moving[2], -1)
```

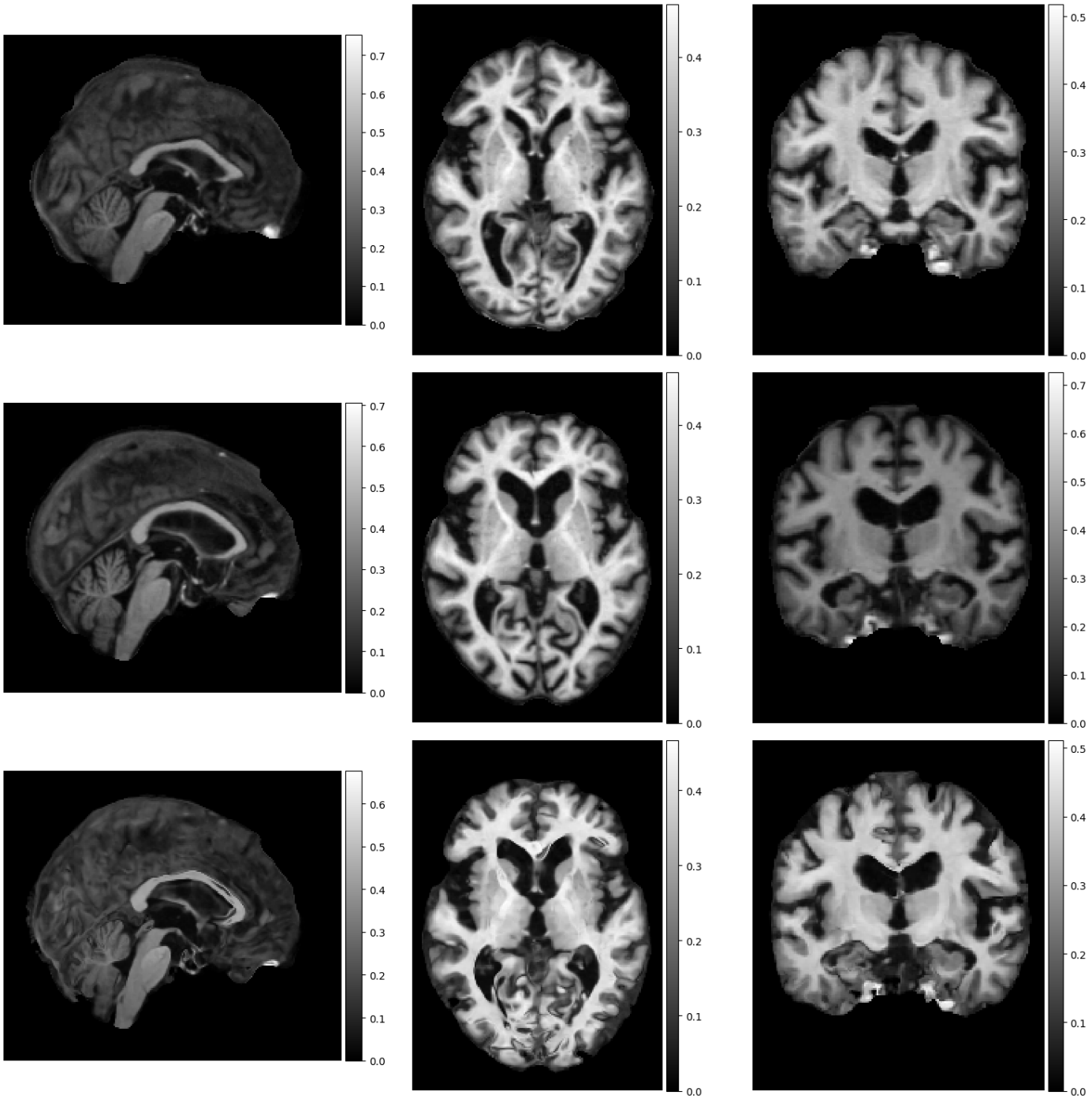
```
  print('2nd fixed image')
  mid_slices_fixed = [np.take(val_volume_2, vol_shape[d]//2, axis=d) for d i
  mid_slices_fixed[1] = np.rot90(mid_slices_fixed[1], 1)
  mid_slices_fixed[2] = np.rot90(mid_slices_fixed[2], -1)
  print('3rd predicted image')
  mid_slices_pred = [np.take(moved_pred, vol_shape[d]//2, axis=d) for d in r
  mid_slices_pred[1] = np.rot90(mid_slices_pred[1], 1)
  mid_slices_pred[2] = np.rot90(mid_slices_pred[2], -1)
  ne.plot.slices(mid_slices_moving + mid_slices_fixed + mid_slices_pred, cma

```

```
1/1 [==============================] - 6s 6s/step
1st moving image
2nd fixed image
3rd predicted image
```



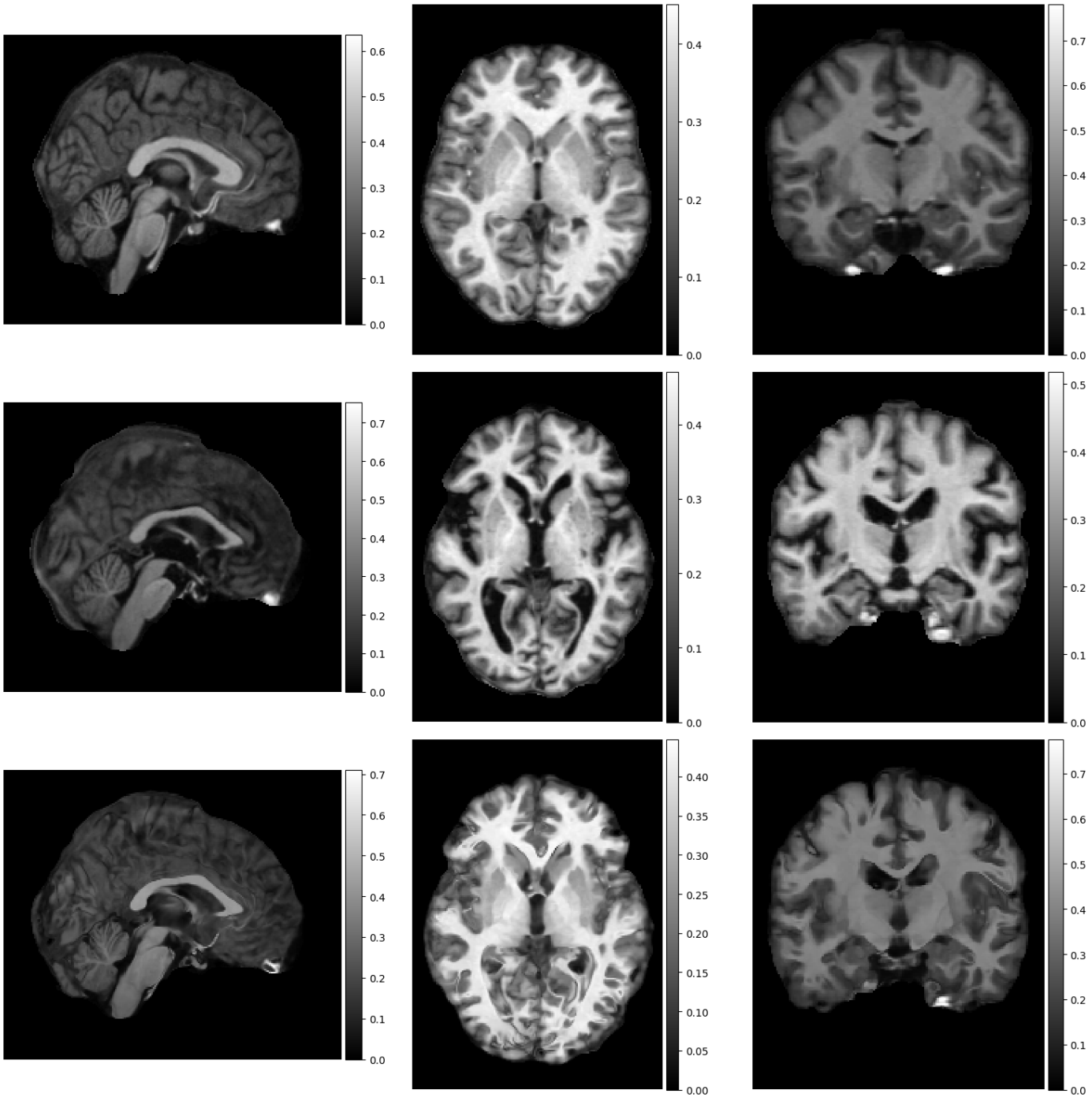


```
1/1 [==============================] - 0s 135ms/step
1st moving image
2nd fixed image
3rd predicted image
```

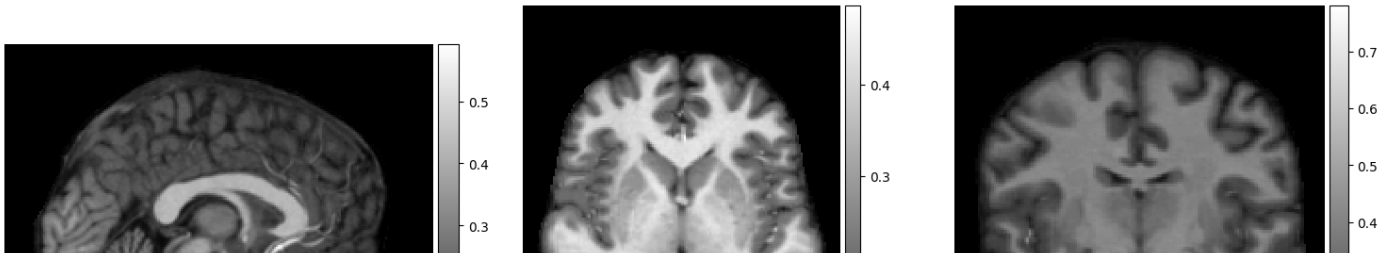


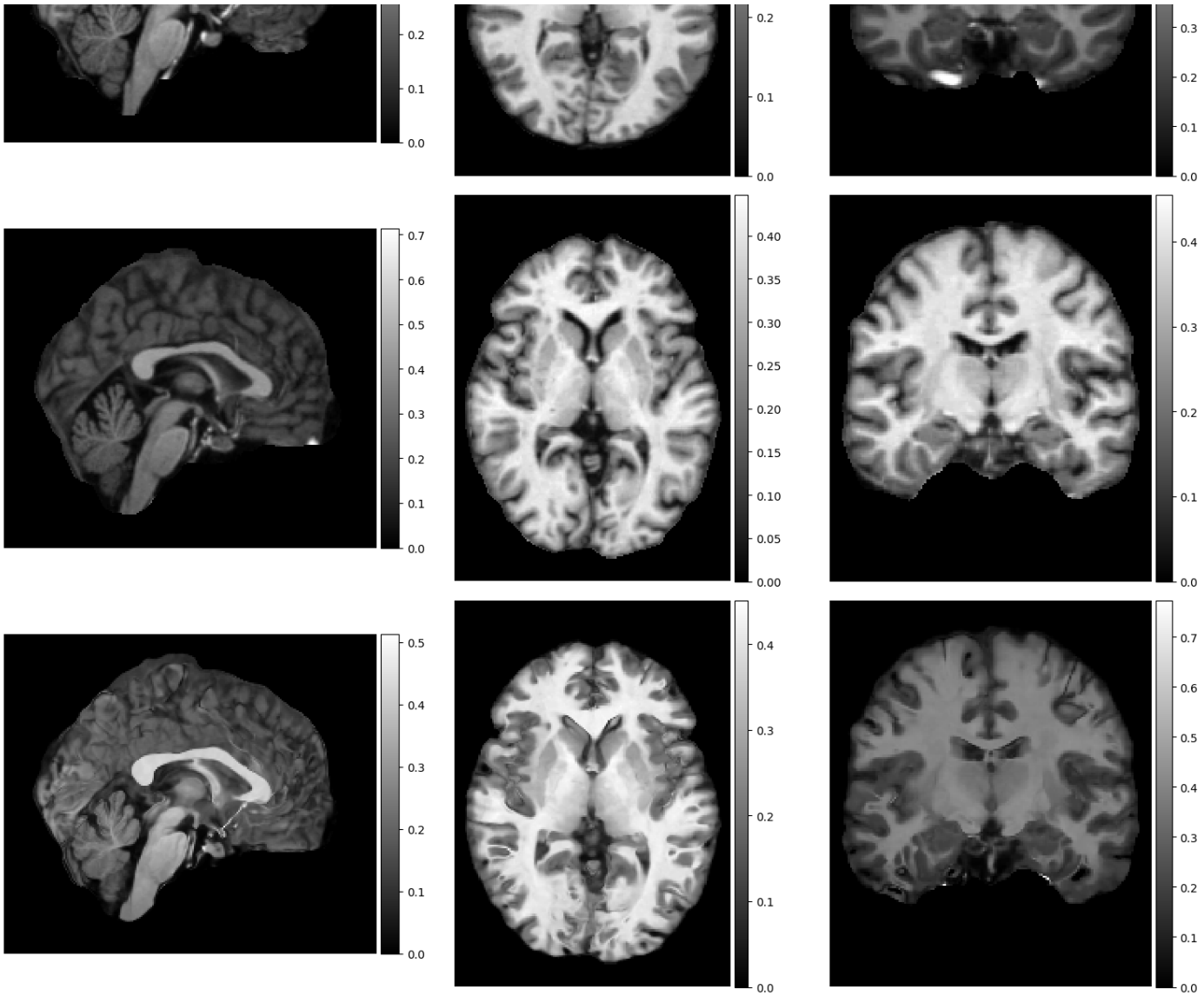
```
1/1 [==============================] - 0s 144ms/step
1st moving image
2nd fixed image
```

```
3rd predicted image
```

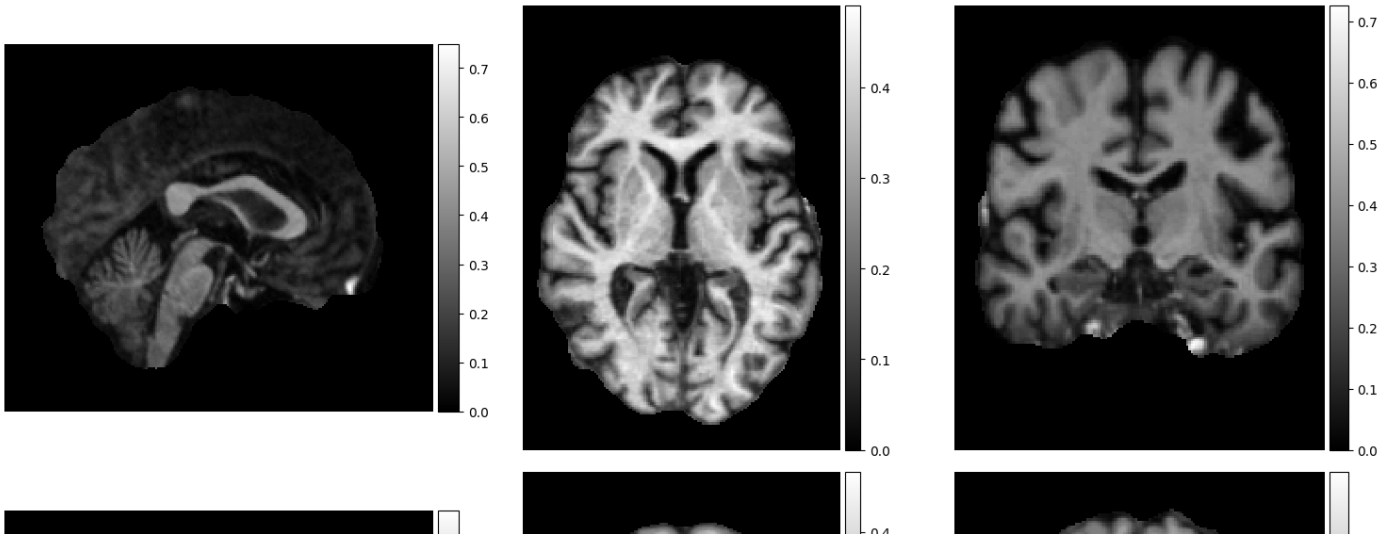


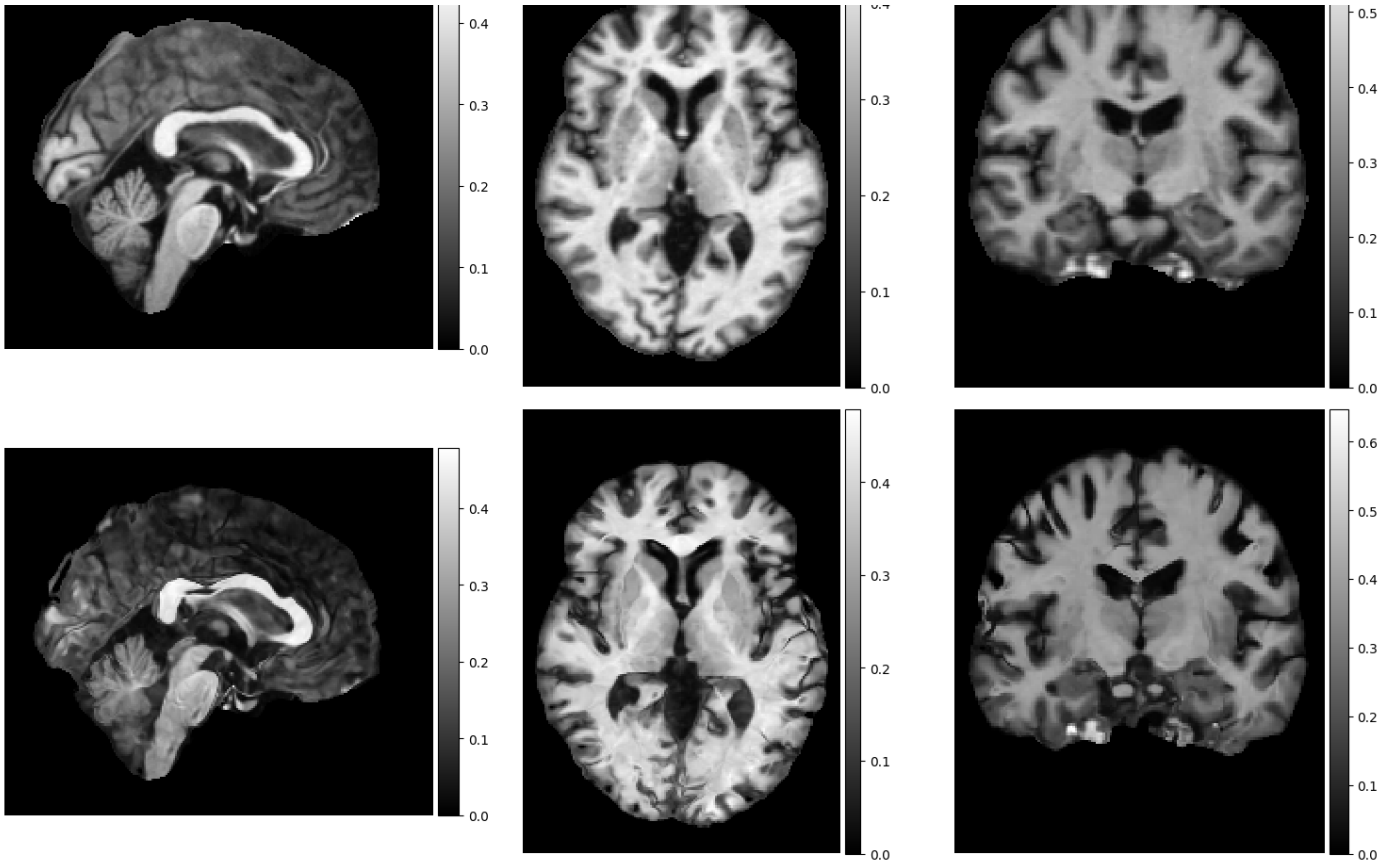
```
1/1 [==============================] - 0s 138ms/step
1st moving image
2nd fixed image
3rd predicted image
```





```
1/1 [==============================] - 0s 134ms/step
1st moving image
2nd fixed image
3rd predicted image
```




0.4
0.3
0.2
0.1
0.0
0.4
0.3
0.2
0.1
0.0
0.5
0.4
0.3
0.2
0.1
0.0
0.4
0.3
0.2
0.1
0.0
0.4
0.3
0.2
0.1
0.0
0.6
0.5
0.4
0.3
0.2
0.1
0.0

```
vxm_model.save_weights('model_ncc_hw4_weights.h5')
```

```
#restarting witht he notebook. Loading the weights of the NN.
weight_path = os.path.join(dir_folder,'model_ncc_hw4_weights.h5')
vxm_model.load_weights(weight_path)
```

```
# to get both img and segments , data generator
def data_generator_test_seg(dir_folder,img_type,batch_size = 1):

    files = [file for file in os.listdir(dir_folder) if file[0:3] == img_type]
    segs = ['seg'+file[3:] for file in files]

    while True:

        #return files,segs

        #print(files)
        num_files = len(files)
        idx1 = np.random.randint(0,num_files,size = batch_size)
        files_1 = [os.path.join(dir_folder,files[i]) for i in idx1]
        segs_1 = [os.path.join(dir_folder,segs[i]) for i in idx1]

        #print(len(files_1))
        #print(len(segs_1))

        moving_images = load_img(files_1)
        moving_images_seg = load_img(segs_1)

        idx2 = np.random.randint(0,num_files,size = batch_size)
        files_2 = [os.path.join(dir_folder,files[i]) for i in idx2]
        segs_2 = [os.path.join(dir_folder,segs[i]) for i in idx2]

        fixed_images = load_img(files_2)
        fixed_images_seg = load_img(segs_2)
        # print(len(files_1))
        # print(len(segs_1))

        inputs = [moving_images,fixed_images]

        img_shape = moving_images.shape[1:]
        img_shape = img_shape[:-1]

        ndims = len(img_shape)
```

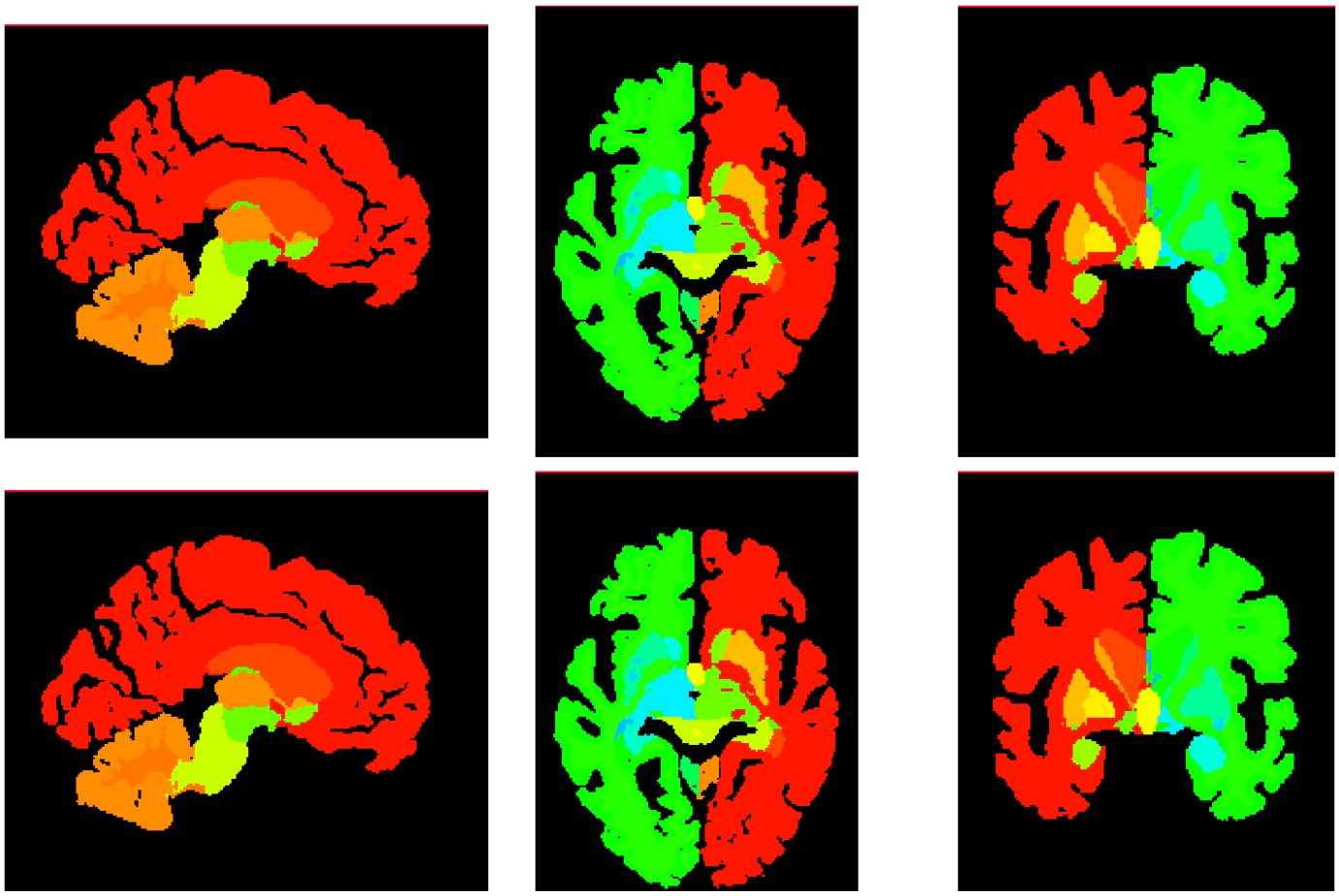
```
        zero_phi = np.zeros([batch_size,*img_shape,ndims])
        outputs = [fixed_images,zero_phi]

        both_segs = [moving_images_seg,fixed_images_seg]

        return inputs,outputs,both_segs


```

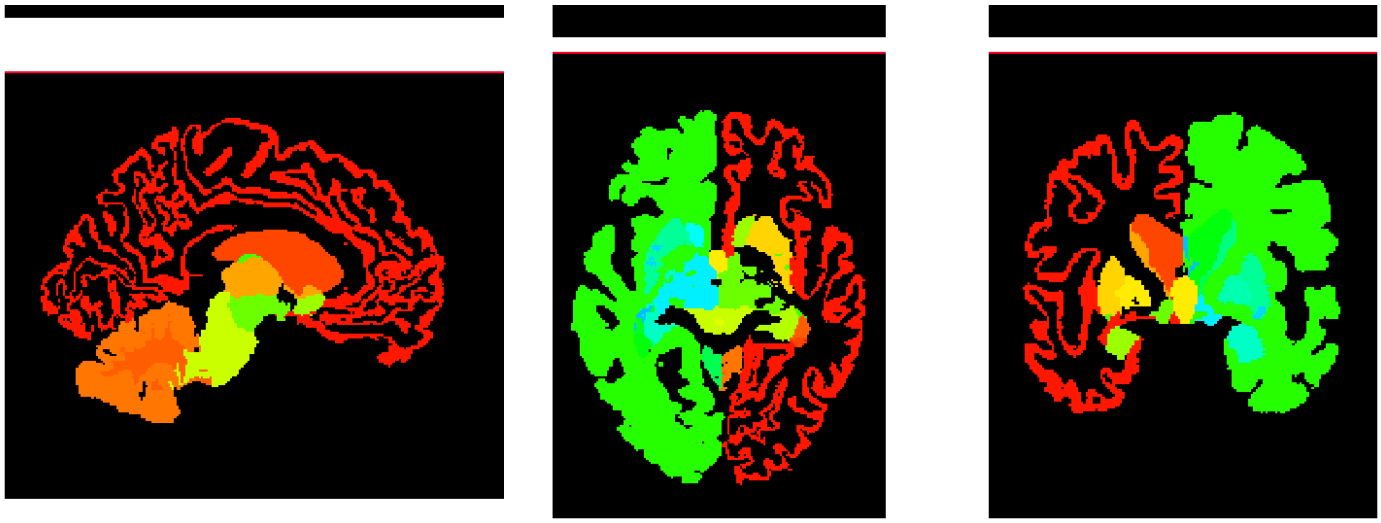
```
from pystrum.pytools.plot import jitter
import matplotlib

[ccmap, scrambled_cmap] = jitter(255, nargout=2)
scrambled_cmap[0, :] = np.array([0, 0, 0, 1])
ccmap = matplotlib.colors.ListedColormap(scrambled_cmap)
```

```
# showing the visulaization result for the given trained model. Task 2 for the

num_pairs = 5
test_generator = data_generator_test_seg(test_folder,'img',batch_size = 1)

for i in range(num_pairs):

    inputs,outputs,segs = data_generator_test_seg(test_folder,'img',batch_size
    val_pred = vxm_model.predict(inputs)
    moved_pred = val_pred[0][0,:,:,:,0]
    pred_warp  = val_pred[1]


    warp_model = vxm.networks.Transform(vol_shape,interp_method = 'nearest')
    warped_seg = warp_model.predict([segs[0],pred_warp])
    print('warped segmentation')

    print('1st moving image')
    mid_slices_moving = [np.take(segs[0].squeeze(), vol_shape[d]//1.8, axis=d)
    mid_slices_moving[1] = np.rot90(mid_slices_moving[1], 1)
    mid_slices_moving[2] = np.rot90(mid_slices_moving[2], -1)
    print('2nd fixed image')
    mid_slices_fixed = [np.take(segs[1].squeeze(), vol_shape[d]//1.8, axis=d)
    mid_slices_fixed[1] = np.rot90(mid_slices_fixed[1], 1)
    mid_slices_fixed[2] = np.rot90(mid_slices_fixed[2], -1)
    print('3rd predicted image')
    mid_slices_pred = [np.take(warped_seg.squeeze(), vol_shape[d]//1.8, axis=d
    mid_slices_pred[1] = np.rot90(mid_slices_pred[1], 1)
```

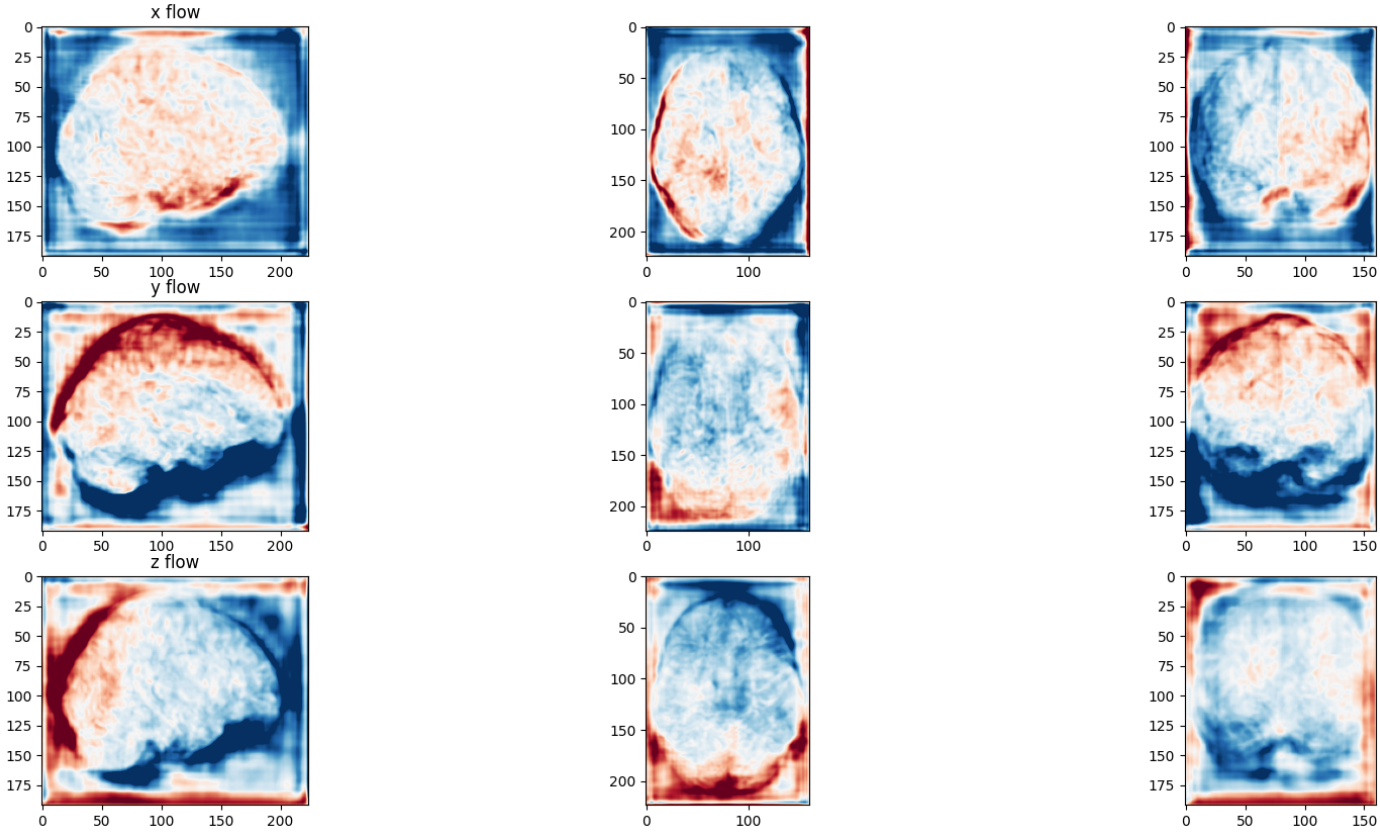
```
    mid_slices_pred[2] = np.rot90(mid_slices_pred[2], -1)
    slices = mid_slices_moving + mid_slices_fixed + mid_slices_pred
    for si, slc  in enumerate(slices):
      slices[si][0] = 255
    ne.plot.slices(slices, cmaps = [ccmap], grid=[3,3]);

    flow = val_pred[1][0, :, :, :, :]
    flow_sd = np.std(flow)
    v_args = dict(cmap = 'RdBu', vmin = -flow_sd, vmax = +flow_sd)
    fig, m_axs = plt.subplots(3, 3, figsize = (20, 10))
    for i, (ax1, ax2, ax3) in enumerate(m_axs):
        ax1.imshow(np.mean(flow[:, :, :, i], 0), **v_args)
        ax1.set_title('xyz'[i]+' flow')
        ax2.imshow(np.rot90(np.mean(flow[:, :, :, i], 1),1), **v_args)
        ax3.imshow(np.rot90(np.mean(flow[:, :, :, i], 2),-1), **v_args)

```

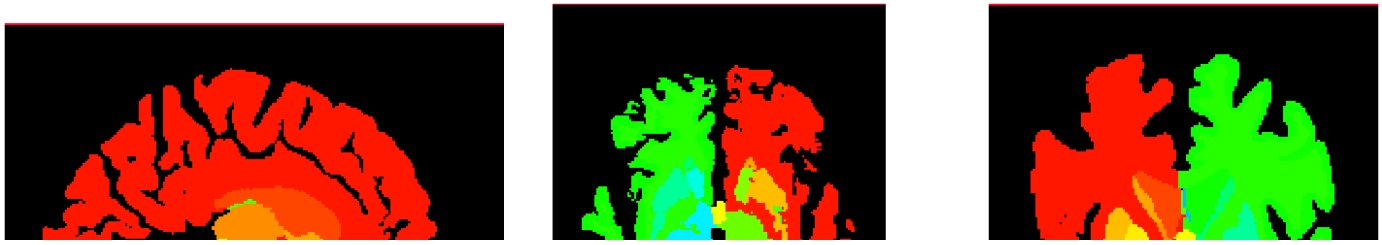
```
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 336ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```

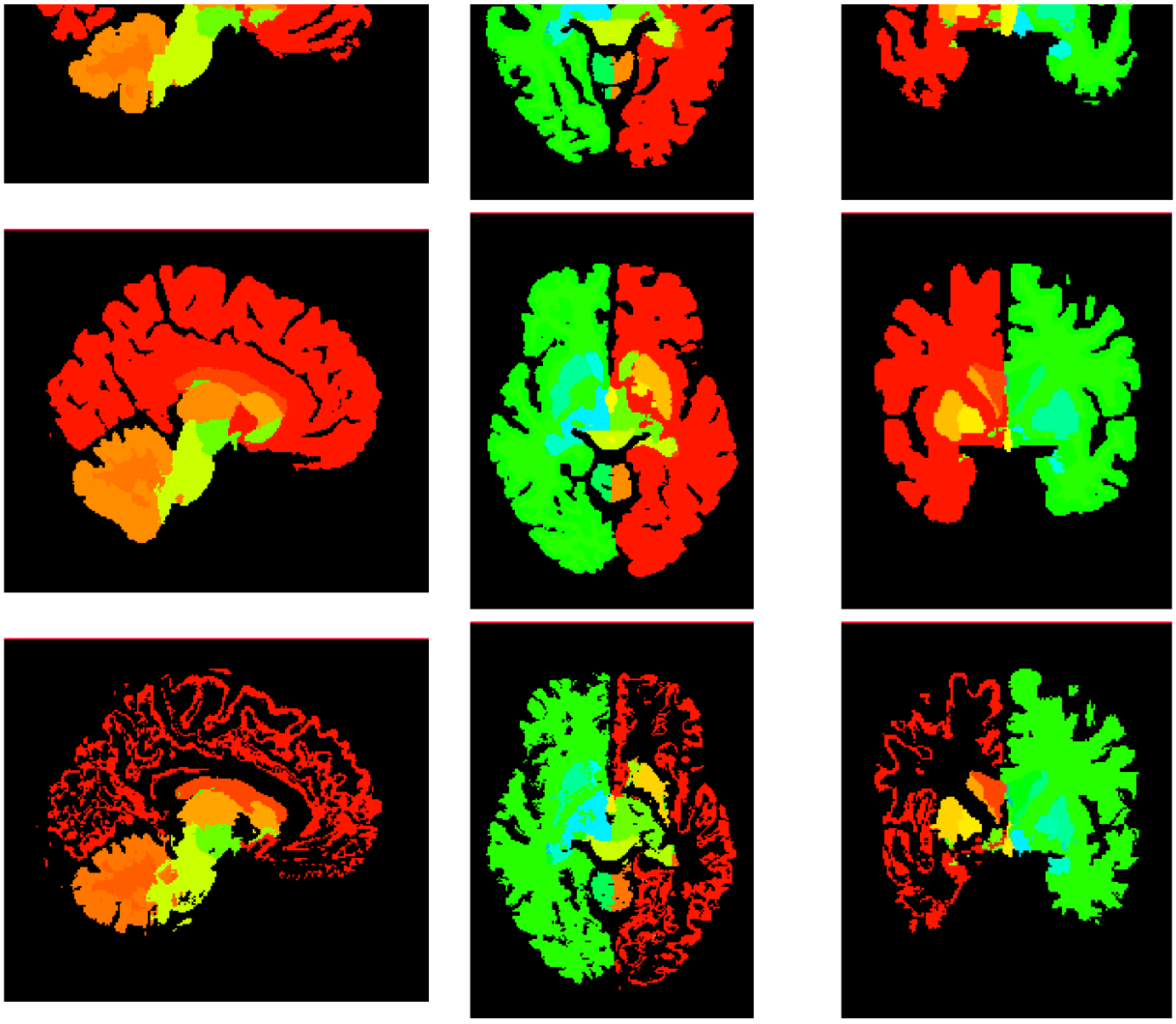




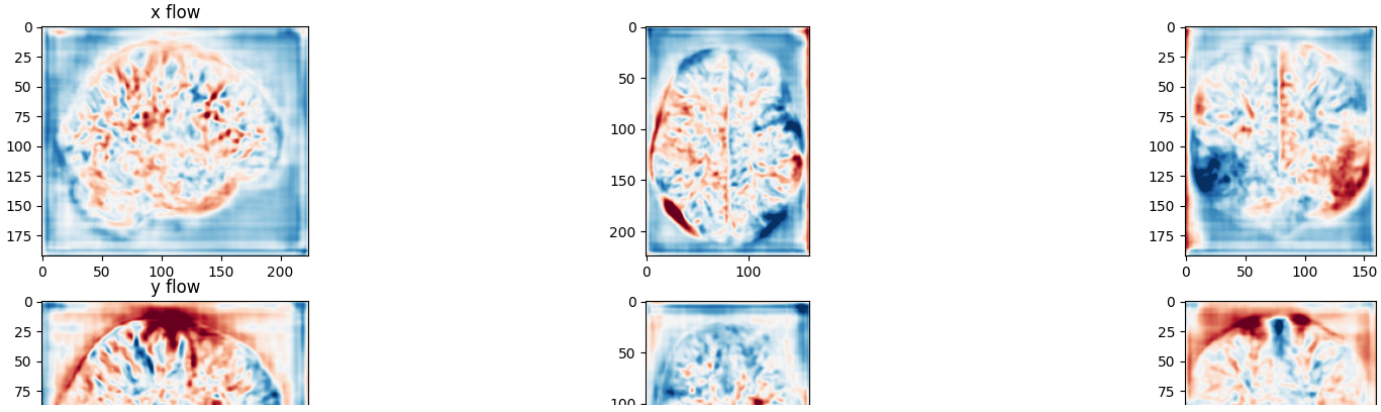
```
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 230ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```

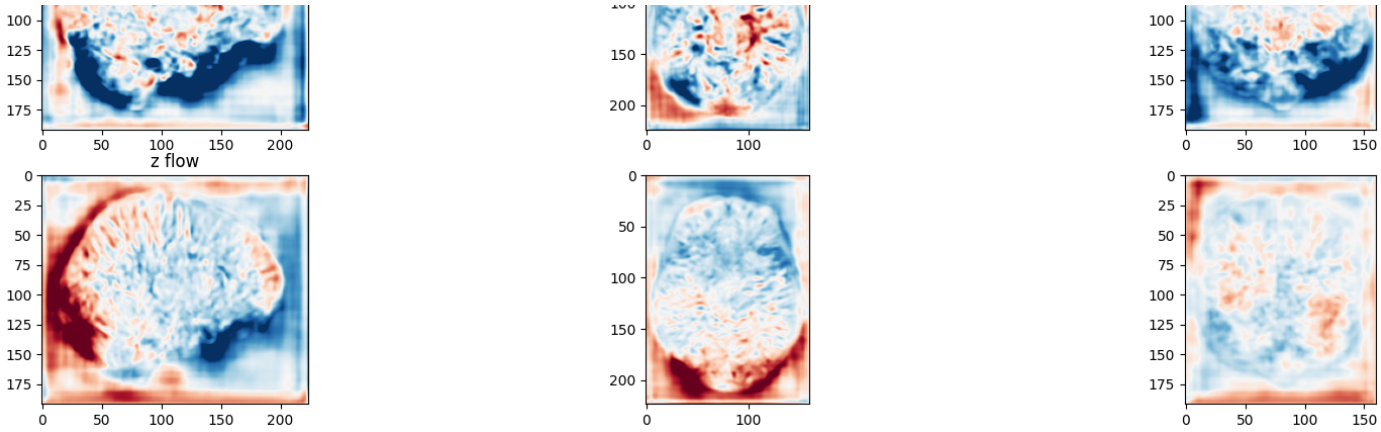


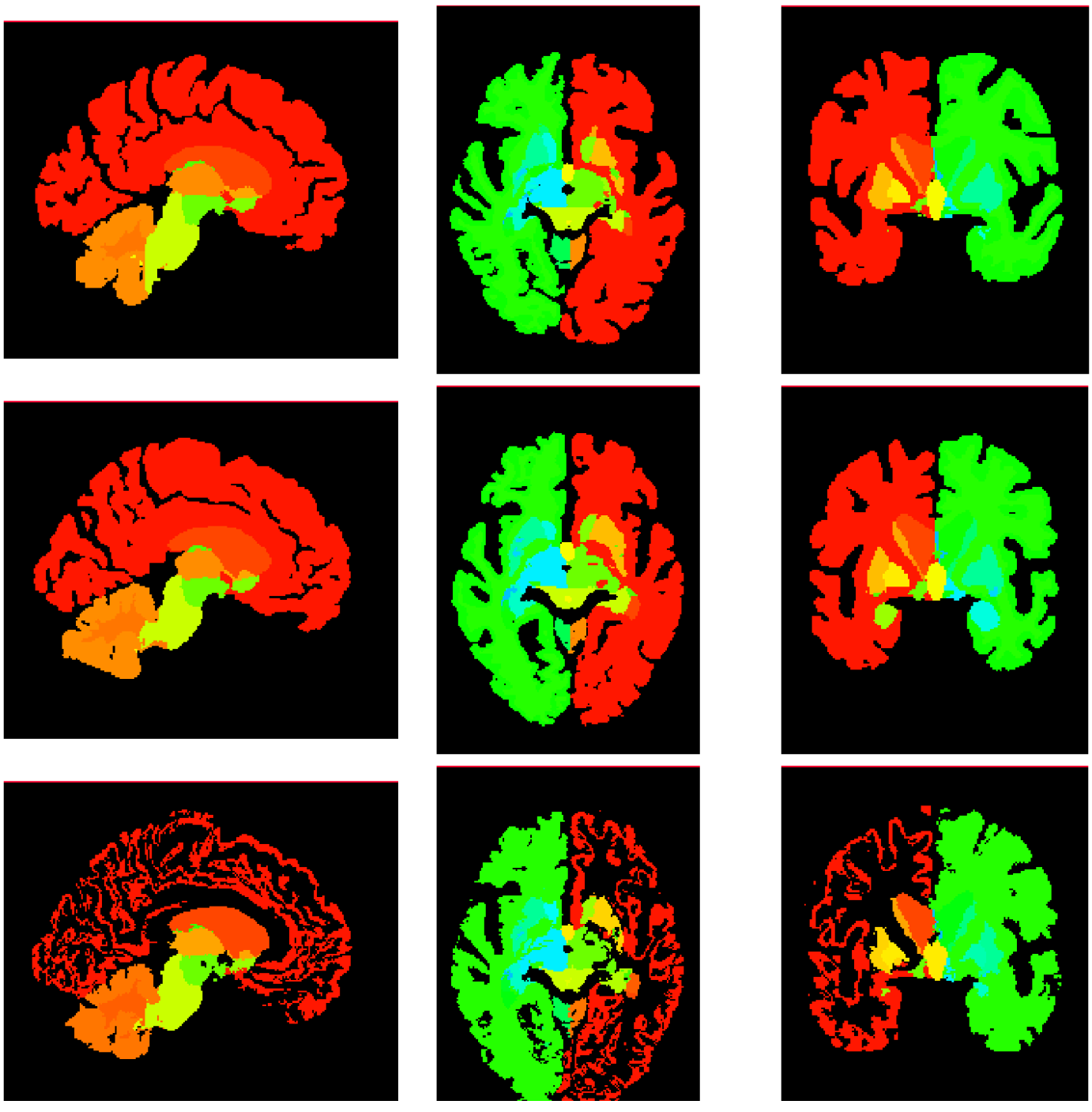




```
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 275ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```

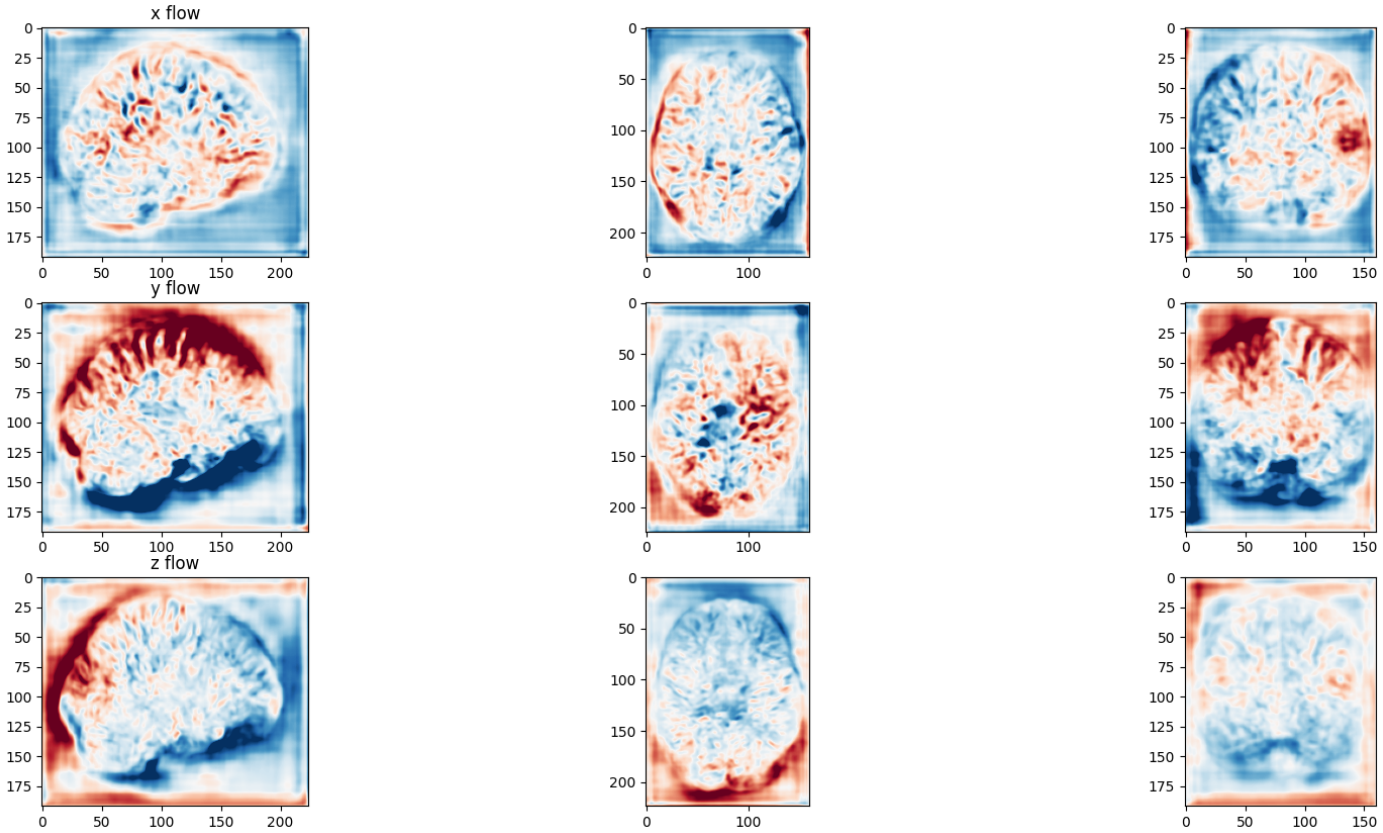


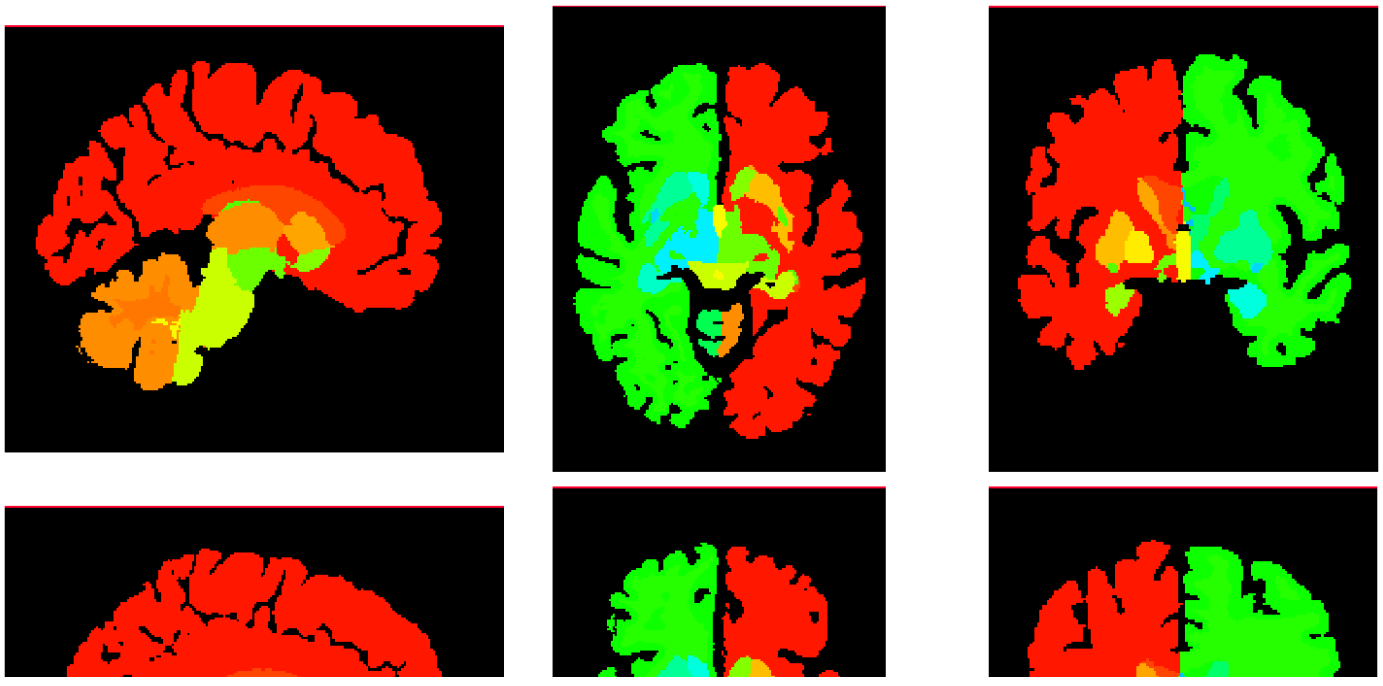

z flow

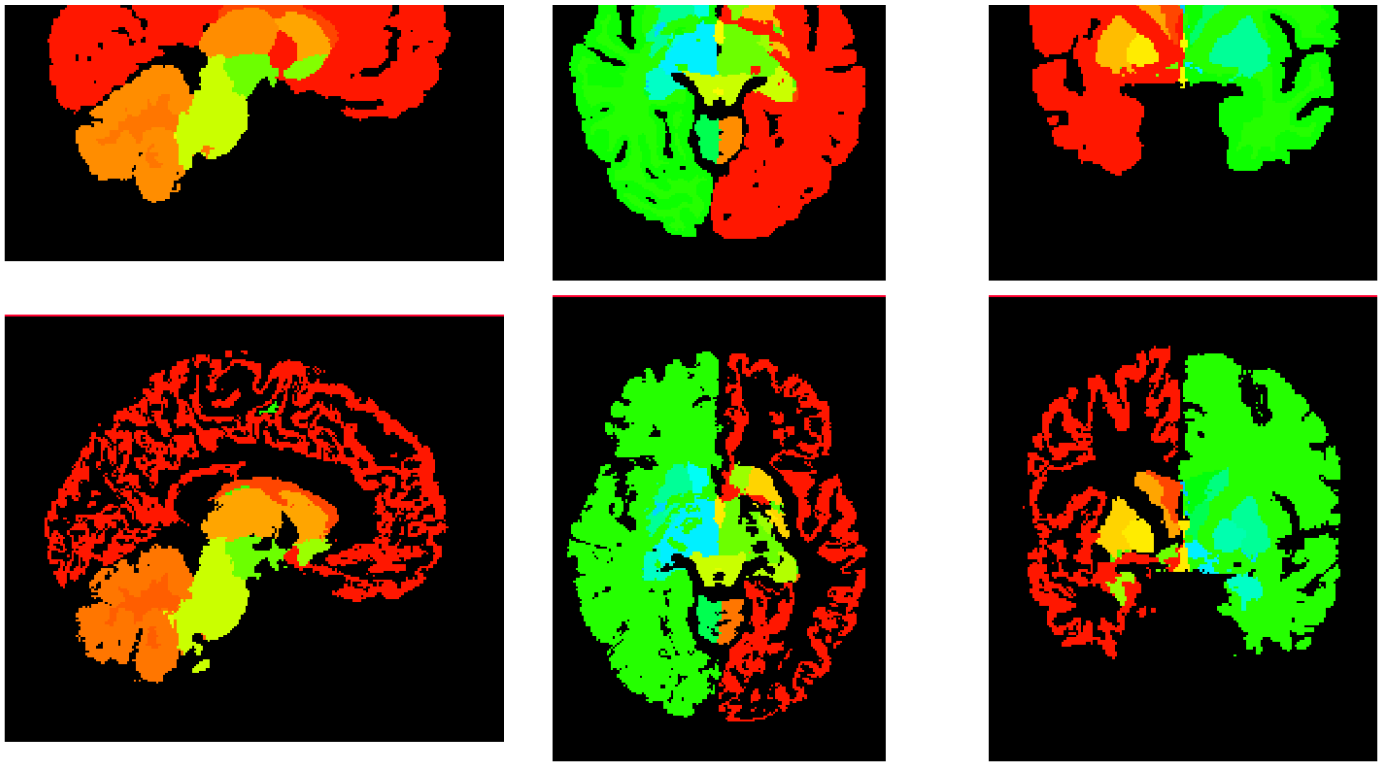




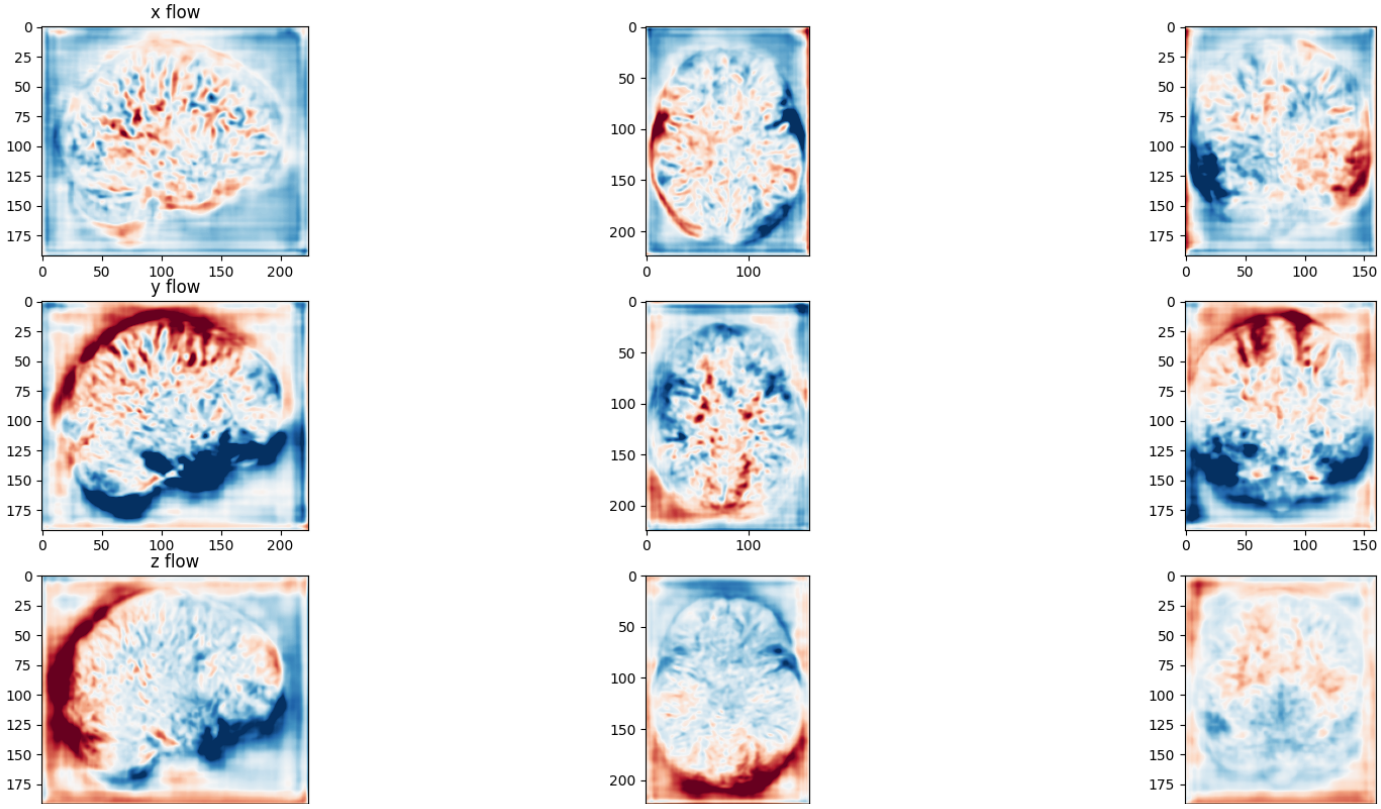
```
1/1 [==============================] - 0s 132ms/step
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_
1/1 [==============================] - 0s 237ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```

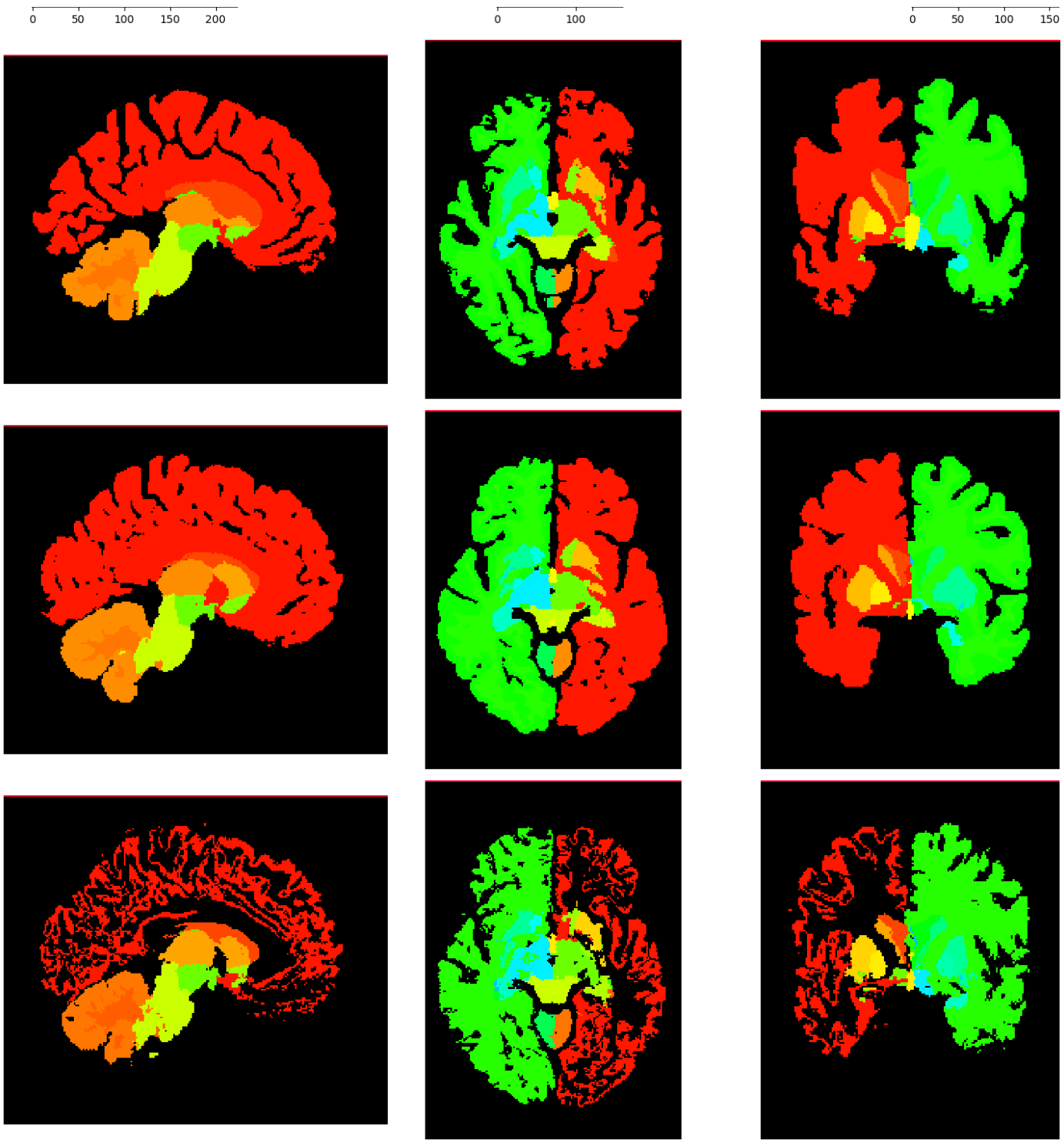


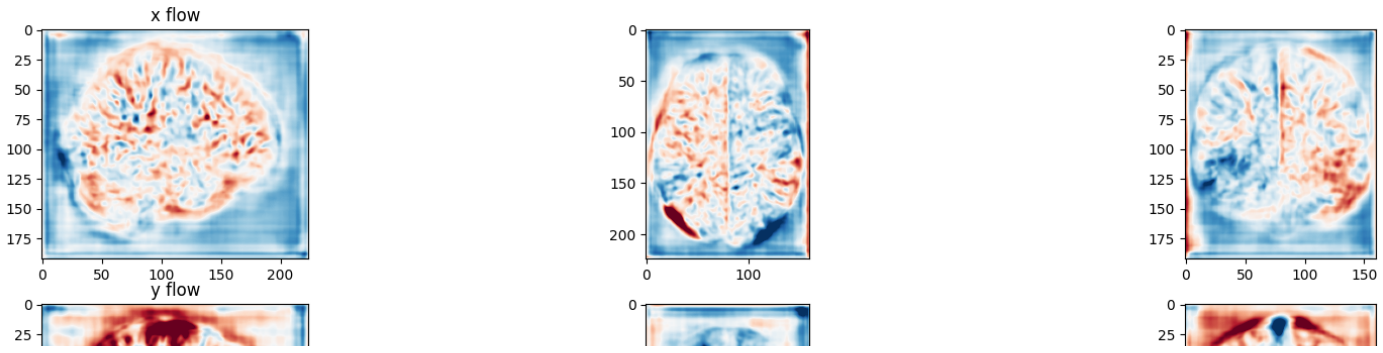


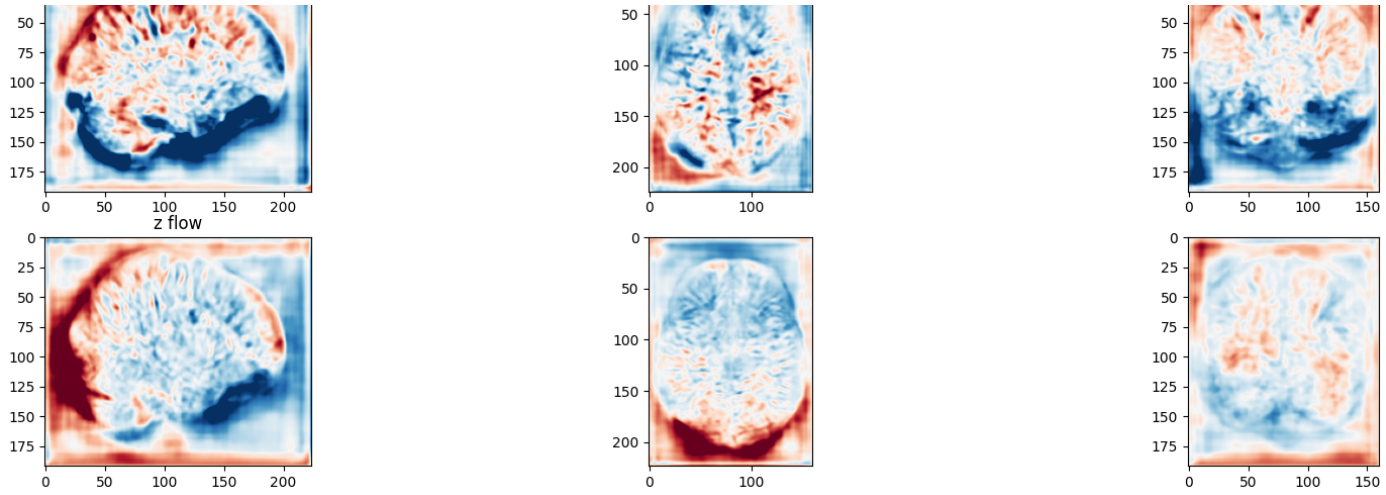


```
1/1 [==============================] - 0s 135ms/step
WARNING:tensorflow:6 out of the last 15 calls to <function Model.make_predict_
1/1 [==============================] - 0s 245ms/step
warped segmentation
1st moving image
2nd fixed image
3rd predicted image
```






x flow
y flow


z flow

```
def dice_coef(segmentation1, segmentation2):

    unique_labels1 = np.unique(segmentation1)
    unique_labels2 = np.unique(segmentation2)

    # Exclude background label (usually 0) from calculations
    unique_labels1 = unique_labels1[unique_labels1 != 0]
    unique_labels2 = unique_labels2[unique_labels2 != 0]

    dice_scores = []

    for label1 in unique_labels1:
            # Create binary masks for each pair of labels
          mask1 = (segmentation1 == label1).astype(int)
          mask2 = (segmentation2 == label1).astype(int)

            # Calculate Dice coefficient for the pair
          intersection = np.sum(mask1 * mask2)
          total_voxels = np.sum(mask1) + np.sum(mask2)
          dice_coefficient = 2.0 * intersection / total_voxels if total_voxels

          dice_scores.append(dice_coefficient)

    # Calculate average Dice score
    average_dice_score = np.mean(dice_scores) if len(dice_scores) > 0 else 0.0

    return average_dice_score
```

```
num_pairs = 10
```

```
dice_score = []
hausdroff_distance = []
for i in range(num_pairs):

    inputs,outputs,segs = data_generator_test_seg(test_folder,'img',batch_size
    val_pred = vxm_model.predict(inputs)
    moved_pred = val_pred[0][0,:,:,:,0]
    pred_warp  = val_pred[1]


    warp_model = vxm.networks.Transform(vol_shape,interp_method = 'nearest')
    warped_seg = warp_model.predict([segs[0],pred_warp])

    dice_score.append(dice_coef(segs[1].squeeze(),warped_seg.squeeze()))
    print(dice_score[i])
    #hausdroff_distance.append(hausdorff_distance_fast(segs[1].squeeze(),warpe
    #print(hausdroff_distance[i])

dice_score = np.array(dice_score)
mean_dice_score = np.mean(dice_score)
#hausdroff_distance = np.array(hausdroff_distance)
#mean_hausdroff_distance = np.mean(hausdroff_distance)

print(f'The mean dice score is {mean_dice_score}')
#print(f'The mean hausdroff distance is {mean_hausdroff_distance}')
```

```
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 231ms/step
0.65775952029042
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 230ms/step
0.7286383673509078
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 257ms/step
0.6214670864289982
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 259ms/step
0.7732751153526977
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 228ms/step
0.667312628331267
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 271ms/step
0.9338102400750758
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 271ms/step
0.9338102400750758
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 226ms/step
0.722431829509364
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 244ms/step
0.7425417922281696
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 249ms/step
0.697536493904502
The mean dice score is 0.7478583313546477
```

```
1
```

Double-click (or enter) to edit