SSL-MRI

Learning Structures from the Essence

Bereketoğlu, Burkan

Middle East Technical University (METU)
Athinoula Martinos Center for Biomedical Imaging (HMS-MIT)

April 7, 2023

Outline

Outline

The main structure of this presentation are:

- Introduction
- Methodology
  - Simple CNN
  - Custom Loss Function
  - Gradient Problem- Epsilon and Threshold
- Results
  - Evaluation Scores
  - Loss
  - True and Predicted Mask Images
  - References

Outline

Introduction

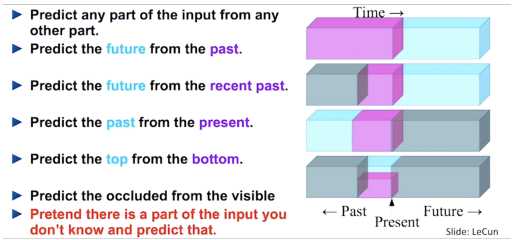

Figure: Self-Supervised Learning [1]

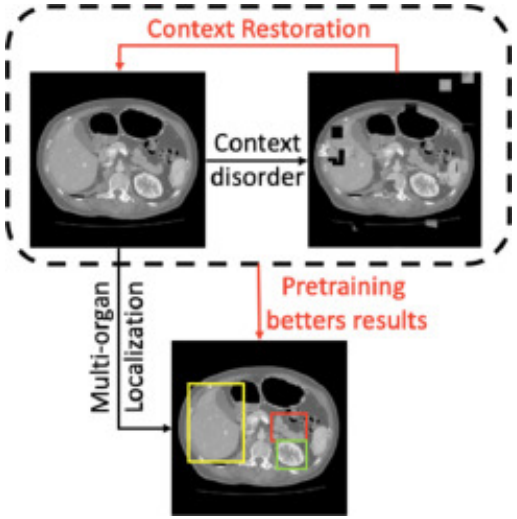

Figure: SSL in Medical Imaging [2]

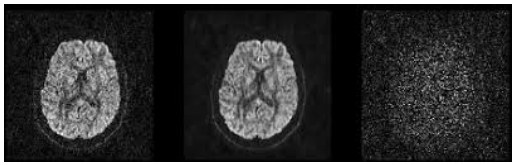

Figure: Noisy Brain Image [3]

- Read the literature for Shihan Qiu from Cedars Sinai (realized that what she does is)
- Rather similar to supervised learning compared to self-supervised.
- Also I realized she does find only T1 and T2. Doesn't deal with PD.

Outline

Simple Convolutional Neural Network(CNN)

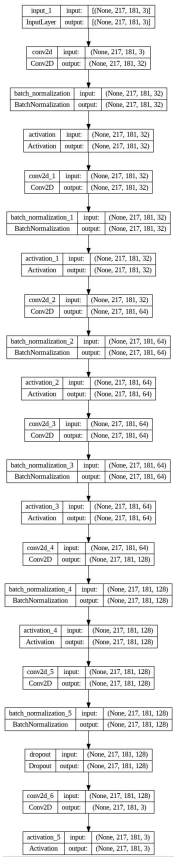

Figure: CNN Model Architecture

- A simple CNN with Size Preservation (actually a big cross-correlation system)
- Batch Normalization $\rightarrow$ prevents gradient explosion/vanishing.
- Softmax $\rightarrow$ multiclass probabilities
- filters(kernel) of 32, 64, and 128 used creates n featured maps (n is the filter number) with input and weights $X_i \cdot W$

Simple Encoder/Decoder CNN

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 217, 181, 3)]     0

 sequential (Sequential)     (None, 217, 181, 512)     4728756

 sequential_1 (Sequential)   (None, 217, 181, 3)       7087959

=================================================================
Total params: 11,816,715
Trainable params: 11,808,555
Non-trainable params: 8,160
_________________________________________________________________
```

Figure: AE CNN architecture

- has filters from (16,32,64,128,256,512).
- Made tests on (32,64,128) standard then (4,8,16,32,64,128,256,512) also as (32,64,128,256)
- Gave results same as our simple sequential CNN with (32,64,128,256). Second one is useless.
- Currently unused. Thought would be great to try another model just in case.

U-net CNN

Figure: U-net similar to AE

- Strides more than 1 so low resolution image with segmentation.
- Again various filtering levels tried. Since maxpooling and upsampling used it is impossible to achieve (217,181) our image size with this method without cropping. Which makes us lose knowledge
- Not that useful for our problem, just helped me

Loss Function

```python
def custom_loss(y_true, y_pred, l1_weight=0.01, l2_weight=0.01, Tsat=1950, TI=145, TR=1200, TE=50, epsilon=1e-8):
    # Extract T1, T2, and PD maps from y_pred
    T1_pred, T2_pred, PD_pred = y_pred[:, :, :, 0], y_pred[:, :, :, 1], y_pred[:, :, :, 2]

    # Define the three Bloch equations with epsilon added to avoid zero division errors
    Img1 = PD_pred * (1 - K.exp(-TR / (T1_pred + epsilon)))
    Img2 = PD_pred * (1 - K.exp(-TR / (T1_pred + epsilon)) * K.exp(-TE / (T2_pred + epsilon)))
    Img3 = PD_pred * K.exp(-Tsat / (T1_pred + epsilon)) * (1 - 2 * K.exp(-TI / (T1_pred + epsilon))) * K.exp(-TE / (T2_pred + epsilon))

    # Compute the mean squared error between the predicted and true images
    loss = K.mean(K.square(Img1 - y_true[:, :, :, 0]) + K.square(Img2 - y_true[:, :, :, 1]) + K.square(Img3 - y_true[:, :, :, 2]))

    # Compute L1 and L2 regularization penalties
    l1_penalty = K.sum(K.abs(loss))
    l2_penalty = K.sum(K.square(loss))

    # Add the regularization penalties to the loss
    loss += l1_weight * l1_penalty + l2_weight * l2_penalty

    return loss
```

Figure: Custom Loss Function

Gradient Problem- Epsilon and Threshold

$$\frac{1}{0} = \text{Undefined}$$

- At values which we have pixel color as black (value is 0, weight 0.), Gradient for the Bloch Equation vanishes, therefore training stops.
- Gradient vanishing issue is fixed by a non-stabilizing minimum ϵ.
- ϵ addition to the equation causes, 0's to become values other than 0 which makes alterations in the images, which necessitates the cancellation of the value after training, at the prediction.

Outline

Changes in CNN

- Added more filters with more sizes (256, 512) different times experimented on them.
- 256 changed the validation loss from 0.4-0.5's to 0.2's. But 512 only makes the overfit faster.
- Dropout as usual. The parameter amount is increased due to filtering. (so slower)
- Found out why U-net encoder/decoder architecture was faster even with higher parameters.
- Reason is strides, due to upsampling and max-pooling I didn't change its strides however we need higher resolution images rather than upsampling. so U-net architecture is useless here.

```python
def custom_loss(y_true, y_pred, l1_weight=0.01, l2_weight=0.00, TI1=2208, TI2 = 545, Tsat=1400, TI=2075, TR=6670, TE=50, epsilon=1e-8):
    # Extract T1, T2, and PD maps from y_pred
    T1_pred, T2_pred, PD_pred = y_pred[:, :, :, 0], y_pred[:, :, :, 1], y_pred[:, :, :, 2] #  DIR [0],FLAIR[1],  FSE [2]  29/3/2023

    """
    https://brainweb.bic.mni.mcgill.ca/brainweb/tissue_mr_parameters.txt
    the The applied multiplications are based on the brainweb txt In a generalized manner
    one can consider those values as the maximum of each. So actually should be as;
    T1_pred = T1_pred*np.max(T1_pred)
    T2_pred = T2_pred*np.max(T2_pred)
    This is done due to softmax activation makes weights in between 0 and 1.
    Didn't apply it on to Proton Density map because it is already in between 0 and 1.
    y_true[:,:,:,0] is fse, y_true[:,:,:,2] is flair

    loss = K.mean(K.square(Img1 - y_true[:, :, :, 0]) + K.square(Img2 - y_true[:, :, :, 1]) + K.square(Img3 - y_true[:, :, :, 2]))
    if it is this type then channels were not appropriate. Refer to load data.

    We apply T2_w FSE and Flair because we only put images of those, however in a generalized model, we will apply all types of image
    and we will make a system for that.

    DIR resembles T1, FLAIR resembles T2 and FSE for PD structural similarities
    """
    T1_pred = T1_pred*2569 # to make them in the same region of Ground truth
    T2_pred = T2_pred*329 # to make them in the same region of Ground truth

    # Define the three Bloch equations with epsilon added to avoid zero division errors
    Img1 = PD_pred * (1- 2*K.exp(-TI2 / (T1_pred + epsilon))+ (2*K.exp(-(TI1 + TI2)/(T1_pred + epsilon))) - K.exp(-TR/(T1_pred+epsilon)))*K.exp(-TE/(T2_pred+epsilon)) # DIR
    Img2 = PD_pred * K.exp(-Tsat / (T1_pred + epsilon)) * (1 - 2 * K.exp(-TI / (T1_pred + epsilon))) * K.exp(-TE / (T2_pred + epsilon)) # FLAIR
    Img3 = PD_pred * (1 - K.exp(-TR / (T1_pred + epsilon)) * K.exp(-TE / (T2_pred + epsilon))) # FSE

    # Compute the mean squared error between the predicted and true images
    loss = K.mean(K.square(Img1 -  y_true[:, :, :, 0]) + K.square(Img2 - y_true[:, :, :, 1]) + K.square(Img3 - y_true[:, :, :, 2]))

    # Compute L1 and L2 regularization penalties
    l1_penalty = K.sum(K.abs(loss))
    l2_penalty = K.sum(K.square(loss))

    # Add the regularization penalties to the loss
    loss += l1_weight * l1_penalty + l2_weight * l2_penalty

    return loss
```

Figure: Custom Loss Function for Flair,FSE,DIR (more complex formulation)

```python
def custom_loss_t1(y_pred, y_true, l1_weight=0.01, l2_weight=0.01, TI1=2208, TI2 = 545, Tsat=1400, TI=2075, TR=6670, TE=50, epsilon=1e-8):
    # Extract T1, T2, and PD maps from y_pred
    T1_pred, T2_pred, PD_pred = y_pred[:, :, :, 0], y_pred[:, :, :, 1], y_pred[:, :, :, 2] #  T1-w [0],T2-w [1],  FLAIR [2]  29/3/2023

    """
    https://brainweb.bic.mni.mcgill.ca/brainweb/tissue_mr_parameters.txt
    the The applied multiplications are based on the brainweb txt In a generalized manner
    one can consider those values as the maximum of each. So actually should be as;
    T1_pred = T1_pred*np.max(T1_pred)
    T2_pred = T2_pred*np.max(T2_pred)
    This is done due to softmax activation makes weights in between 0 and 1.
    Didn't apply it on to Proton Density map because it is already in between 0 and 1.
    y_true[:,:,:,0] is fse, y_true[:,:,:,2] is flair

    loss = K.mean(K.square(Img1 - y_true[:, :, :, 0]) + K.square(Img2 - y_true[:, :, :, 1]) + K.square(Img3 - y_true[:, :, :, 2]))
    if it is this type then channels were not appropriate. Refer to load data.

    We apply T2_w FSE and Flair because we only put images of those, however in a generalized model, we will apply all types of image
    and we will make a system for that.

    DIR resembles T1, FLAIR resembles T2 and FSE for PD structural similarities
    """
    T1_pred = T1_pred*2569 # to make them in the same region of Ground truth
    T2_pred = T2_pred*329 # to make them in the same region of Ground truth

    # Define the three Bloch equations with epsilon added to avoid zero division errors
    Img1 = PD_pred * (1-K.exp(-TR/(T1_pred + epsilon)))# T1-w according to nishimura book.
    Img2 = PD_pred * (K.exp(-TE/(T2_pred+epsilon))) # T2-w according to nishimura book.
    Img3 = PD_pred * K.exp(-Tsat / (T1_pred + epsilon)) * (1 - 2 * K.exp(-TI / (T1_pred + epsilon))) * K.exp(-TE / (T2_pred + epsilon)) # FLAIR

    # Compute the mean squared error between the predicted and true images
    loss = K.mean(K.square(Img1 -  y_true[:, :, :, 0]) + K.square(Img2 - y_true[:, :, :, 1]) + K.square(Img3 - y_true[:, :, :, 2]))

    # Compute L1 and L2 regularization penalties
    l1_penalty = K.sum(K.abs(loss))
    l2_penalty = K.sum(K.square(loss))

    # Add the regularization penalties to the loss
    loss += l1_weight * l1_penalty + l2_weight * l2_penalty

    return loss
```

Figure: Custom Loss Function (less complex formulation)

Various Loss Functions

- My goal was to understand since Shihan Qiu used T1w, T2w, and Flair
- To see whether a more complex loss function with much more complex weighted images improves the feature extraction, or has negative effects. (In results we see that simpler images make better results.)
- Currently finishing trials for all models. For better understanding (but have results for many)

Optimizers

- Since Adam is a combination of ADAgrad and RMSProp, and acts with the first and second moment of the gradient. I currently experiment on Adam and RMSProp.
- To see if they change the validation loss, basically, converge near the true features.
- Currently did one experiment on this and I can only state that RMSProp gets to overfit faster. and not better (validation loss for Adam was similar for the same model.). I will also plan on trying RAdam which is rectified by Adam a novel optimizer.

Random Seeds

- In theory as we know sigmoid and other squashing methods are prone to get affected by initialization, so setting that is intuitively important. Even though recent methods such as more convex leakyReLu and so are not that prone still I tried 2 different seeds.
- I believe it didn't change the loss since when I thought that the loss got really high, it was not because of the initialization but rather due to not setting weighted images to their correct spots. for each Bloch equation, their weighted image should be the corresponding image to that bloch, but if you do not fix it then they can get mixed which results in different losses. I fix them for my experiments.

Regularization

- Since lasso and ridge regularization may change the learning curve and how the model learns I try to make 4 different models for each based on that. One for l1l2 using with penalty 0.01 each one converges itself without penalty, one that used l1, and one that uses l2.

5 Results

- Complex Loss Results - U-net
- Complex Loss Results - Simple-CNN
- Simple Loss Results - Simple-CNN
- Simple Loss Results - U-net
- Complex Loss Results - SimpleCNN
- Complex Loss Results - SimpleCNN
- Complex Loss Results - SimpleCNN
- Complex Loss Results - SimpleCNN
- Complex Loss Results - Unet
- Complex Loss Results - Unet
- Complex Loss Results - Unet
- Complex Loss Results - Unet

Peak SNR, RMSE and more...

For U-NET

1. w/ lasso ve ridge penalty -- loss: 0.3088 - val_loss: 0.3268 , Reconstruction error: 0.226, Evaluation loss: 0.4421, RMSE: 0.4926, PSNR = 54.21, ssim_score: 0.2021 +++
2. w/o lasso and ridge regularization -- val_loss : 0.32 loss :0.32 (keeps decreasing overfit) - Reconstruction error: 0.251 (to prevent overfit 10 epochs sufficient.) SSIM == 0.34 RMSE == 0.52 PSNR = 53.8 eval_loss = 0.41 ++
3. w/ lasso(l1) only --loss: 0.3112 - val_loss: 0.3155, Reconstruction error: 0.2367, Evaluation loss: 0.4368, RMSE: 0.5055, PSNR = 54.055, ssim_score: 0.2796
4. w/ ridge(l2) only-- loss: 3.1400 - val_loss: 0.7991, eval_loss = 0.7991, RSME = 0.6558, ssim_score: 0.2584 ,Reconstruction error: 0.4423, RMSE: 0.6558, PSNR = 51.7953 (only using Ridge makes our model learn nothing. So we should not use l2.)

l1 is the best among all.

Figure: Results U-net for Flair,FSE,DIR (more complex formulation)

Peak SNR, RMSE and more...

For Simple CNN

1. w/ lasso ve ridge regularization -- loss: 3.4419 - val_loss: 0.8531,eval_loss: 0.8855, RMSE: 0.6558,Reconstruction error: 0.2792,ssim_score: 0.2464, PSNR = 51.8638 ++
2. (for 42) w/o lasso and ridge regularization -- loss: 0.0673 - val_loss: 0.2756, PSNR =53.4969, ssim_score: 0.2369, Reconstruction error = 0.2459, RMSE: 0.5391, for FSE,FLAIR DIR it was loss = 0.3, val_loss 0.55 ++ (overfitted so less epoch.)

-- random_seed also changes the loss and overfit. So I tried 42 and 14 for 2. w/o Simple CNN will also try for others hera re the results for 14 (all the other results are taken for 14 not 42, but will make them 42 too.)

2. w/o lasso and ridge regularization -- loss: 0.2691 - val_loss: 0.3340,reconstruction_error = 0.2639, Evaluation loss: 0.4382, RMSE: 0.5394, PSNR= 53.4918, ssim_score: 0.2791.
3. w/ lasso(l1) only --loss: 0.2717 - val_loss: 0.3381, Reconstruction error: 0.2646, Evaluation loss: 0.4387,
4. w/ ridge(l2) only--loss: 0.2717 - val_loss: 0.3201, Evaluation loss: 0.4292, Reconstruction error: 0.2152, RMSE: 0.5394, PSNR = 54.3704, ssim_score: 0.1155

-- extra note: different weighted images have different structural similarities to the t1,t2,pd so even changing what they represent change the loss. For example if we wish fse to represent t1 then it is whole another topic, if we want flair to represent t1 it is something else.

-- 25 or 15 or 10 epochs are too much, lowered to 5. (they all overfit.) -- I will make another ipynb for this one and report all in the slide

-- Change of formulation after 4 of each. Now I will use T1w -T2w and FLAIR (flair is the best one to show pd T1w is best for T1 and T2w is best for T2.)

Figure: Results Simple CNN for Flair,FSE,DIR (more complex formulation)

Peak SNR, RMSE and more...

For Simple CNN (seed 42)

1. w/ lasso ve ridge regularization -- loss: 0.3376 - val_loss: 0.5903, Reconstruction error: 0.2497, RMSE: 0.4988, PSNR = 54.1716 , ssim_score: 0.4296;
2. Also tried with more filters (8 filters with [32,32,64,64,128,128,256,256]), Evaluation loss: 0.2243, loss: 0.2087 - val_loss: 0.2346, RMSE: 0.4681, PSNR: 54.7225,ssim_score: 0.1007
3. Now trying with AutoEncoder CNN model with 8 filters, tons of parameters that we are trying to prevent. (looks like more complex autoencoder does not change much, but we'll see the result) PSNR: 55.0022, RMSE = 0.45, loss: 0.2068 - val_loss: 0.2351, eval_loss = 0.21, Reconstruction error: 0.24859, more than 256 filter didn't decrease loss however made easier overfit.
4. AE CNN with 16-16 encoder-decoder filters from (4,8,16,32,64,128,256,512) with high resolution preserved with stride = 1. however only changed the spot where we overfit. (Easier to overfit, too much parameters, computationally costly and inefficient.) Evaluation loss: 0.2136, minimum (loss: 0.2105 - val_loss: 0.2557), last (loss: 0.2074 - val_loss: 0.3548), RMSE: 0.5433, PSNR= 53.4285, ssim_score: 0.0542 (similarity decreased harshly.).
5. Different Optimizer with Also tried with more filters (8 filters with [32,32,64,64,128,128,256,256]) sequential. loss: 0.2504 - val_loss: 0.2710, Evaluation loss: 0.2819, PSNR: 54.6716, RMSE= 0.4709, ssim_score: 0.1276

Figure: Results Simple CNN for T1W,T2W,FLAIR (simpler formulation)

--- FLAIR T1 T2 (experimenting with different filter sizes, if more complex filtering works will try to the up one too.) For U-NET (seed 42)

1. w/ lasso ve ridge penalty -- (the results are for this one.) loss: 0.62 - val_loss: 0.6734, Evaluation loss: 0.6733, RMSE: 0.6534, ssim_score: 0.2812, PSNR 51.8269

Figure: Currently On Work (simple loss U-net)

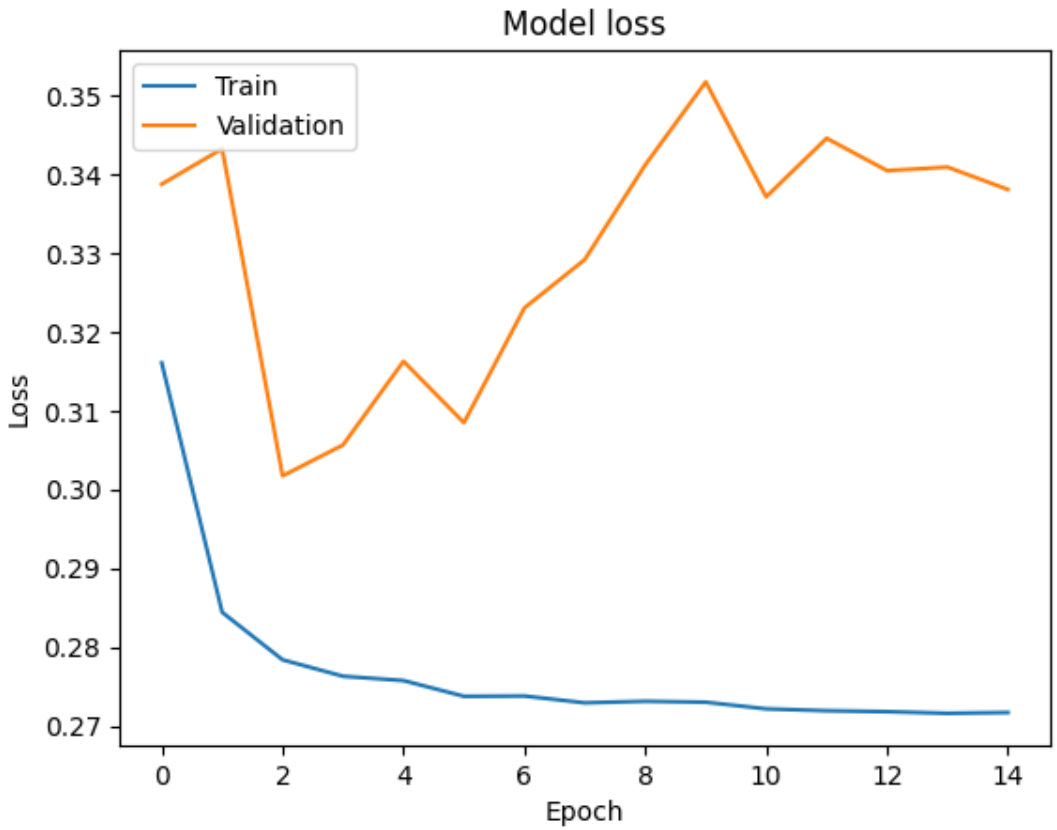

Figure: loss(l1)

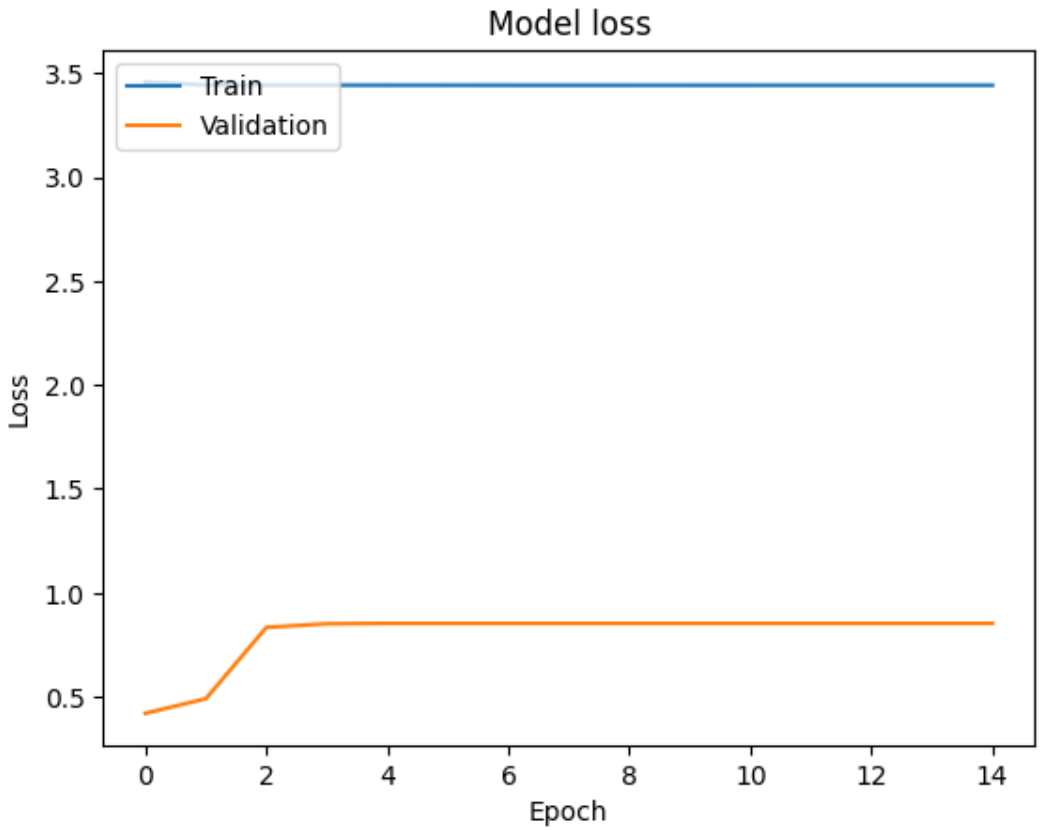

Figure: loss(l1l2)

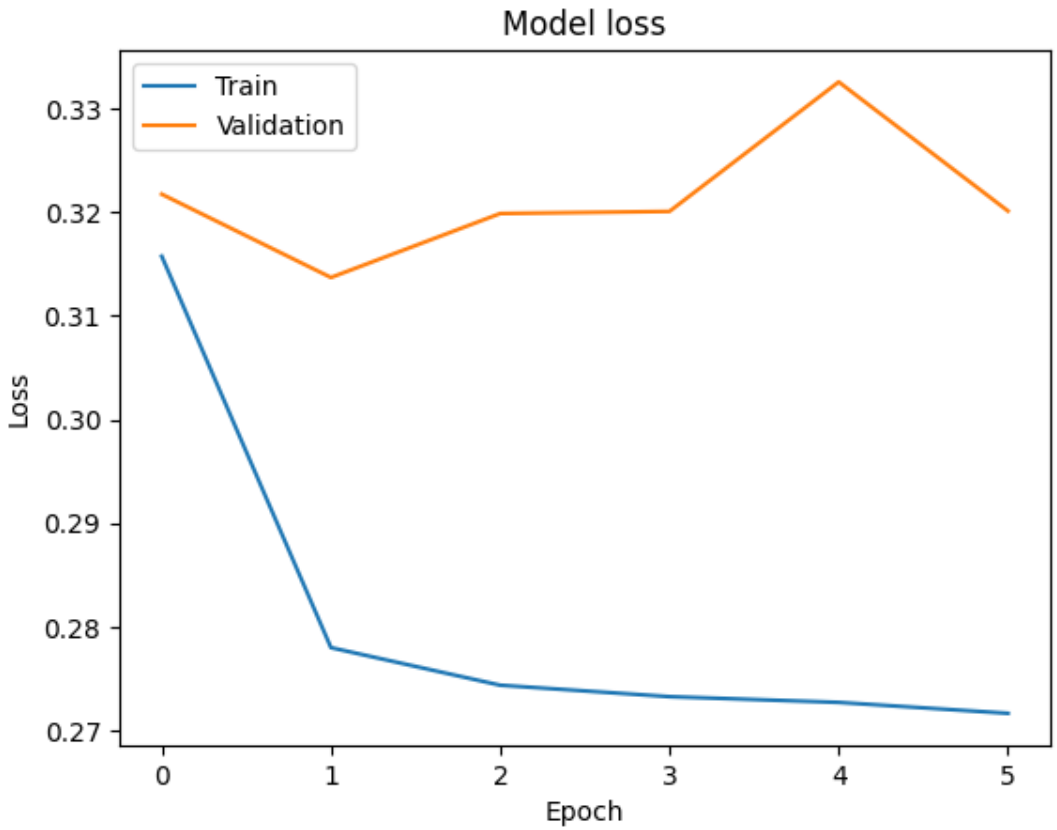

Figure: loss(l2)

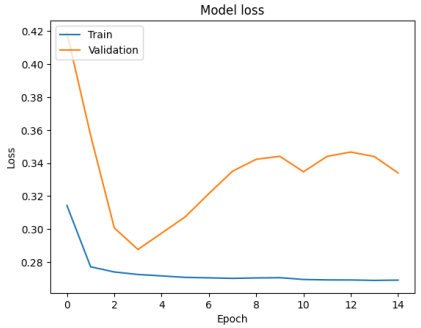

Figure: loss (without$_{/}$1/2*seed*14)

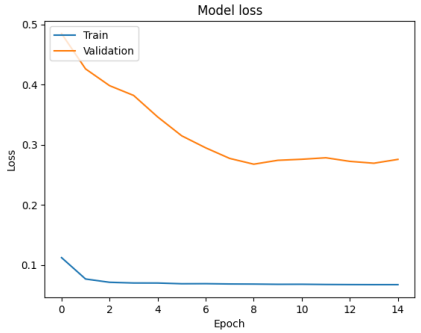

Figure: loss (without$_{/}$1/2*seed*42)

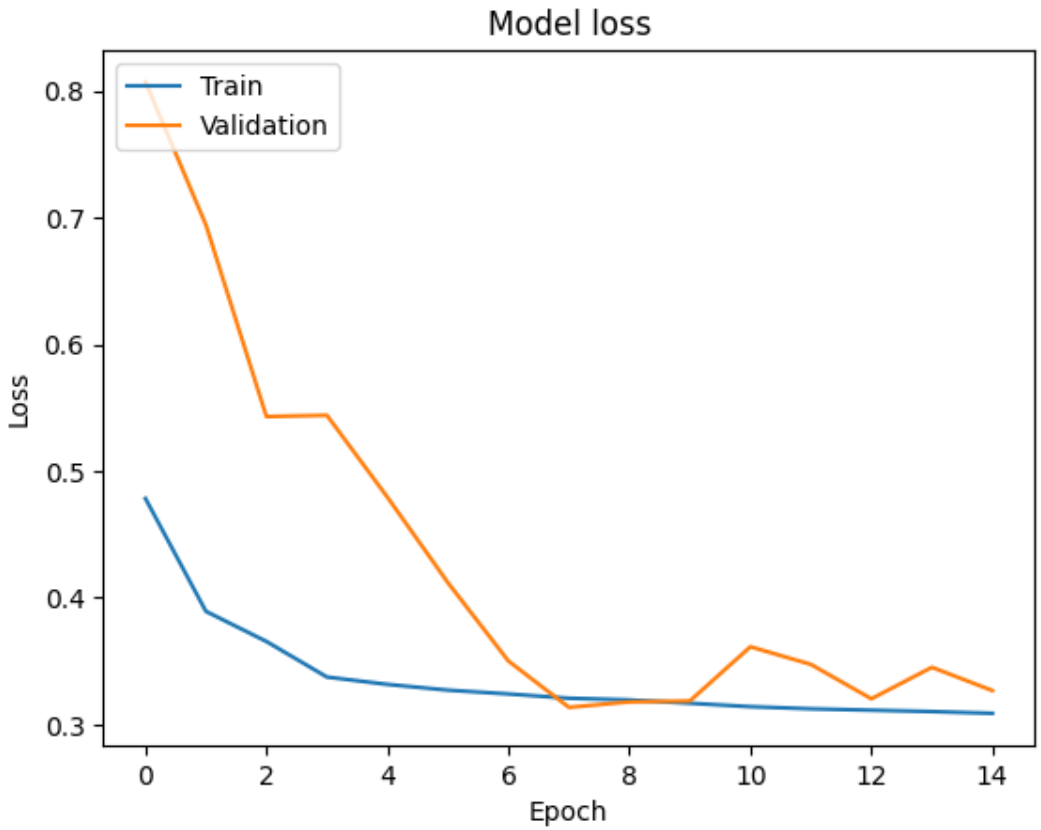

Figure: loss (with l1l2)

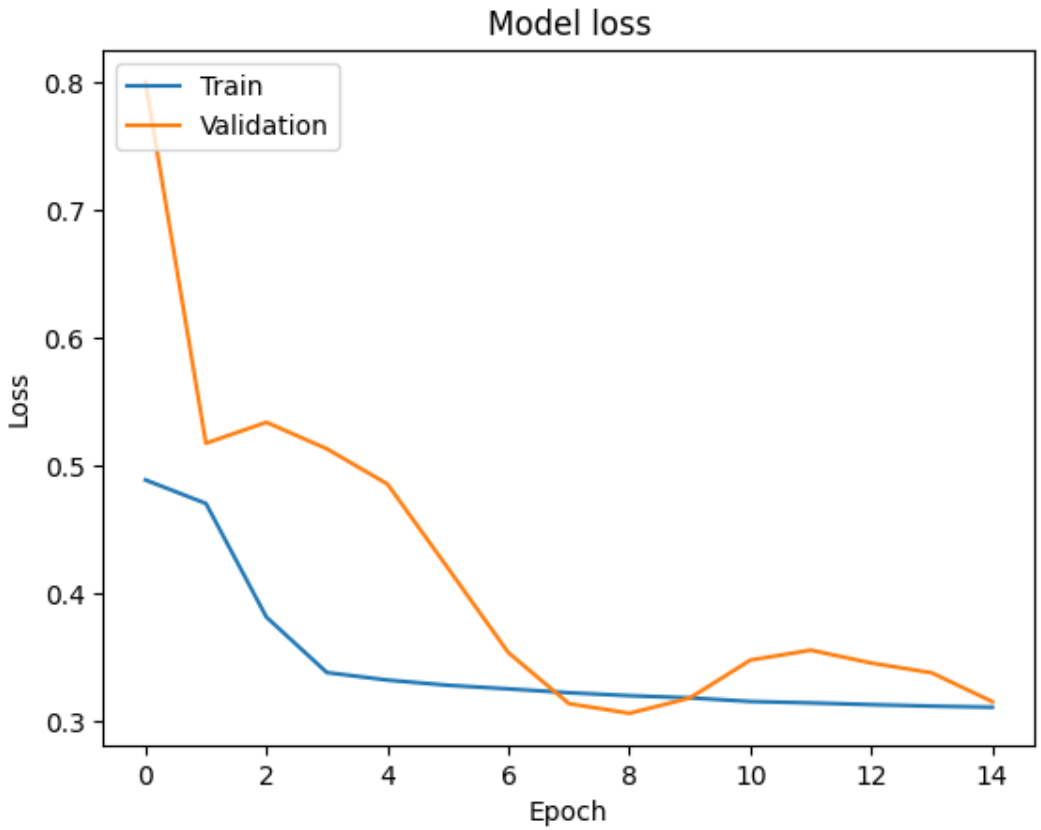

Figure: loss (l1)

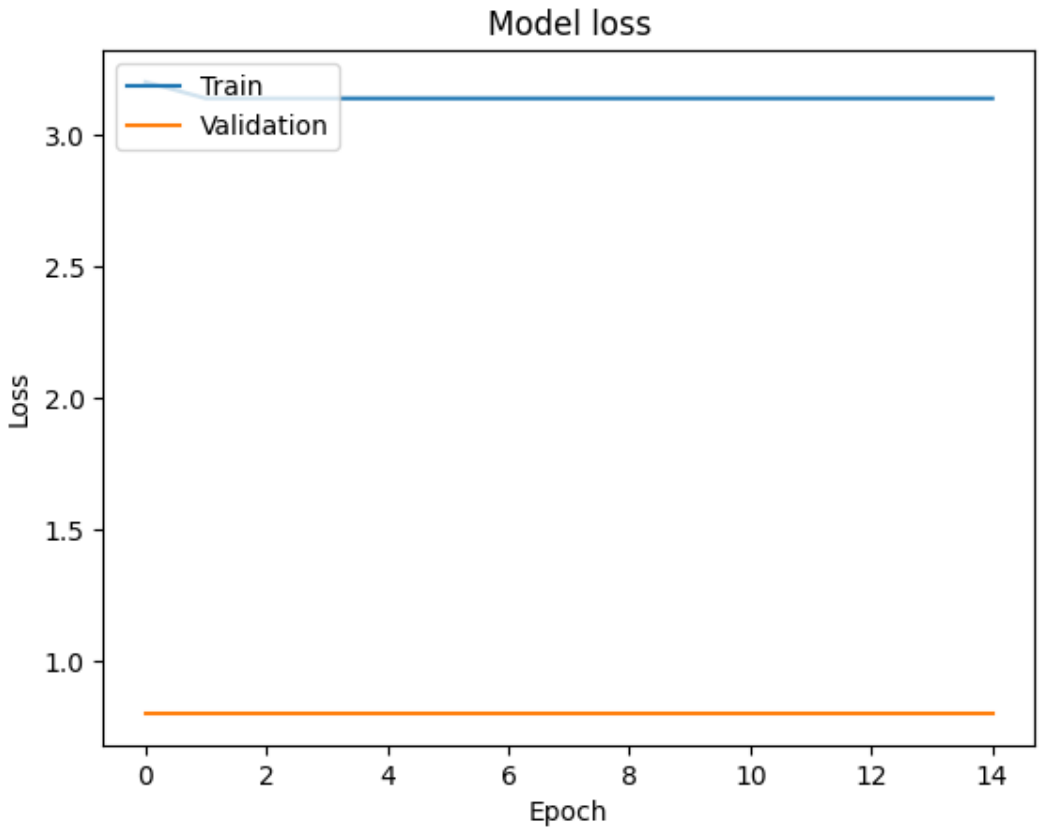

Figure: loss (l2)

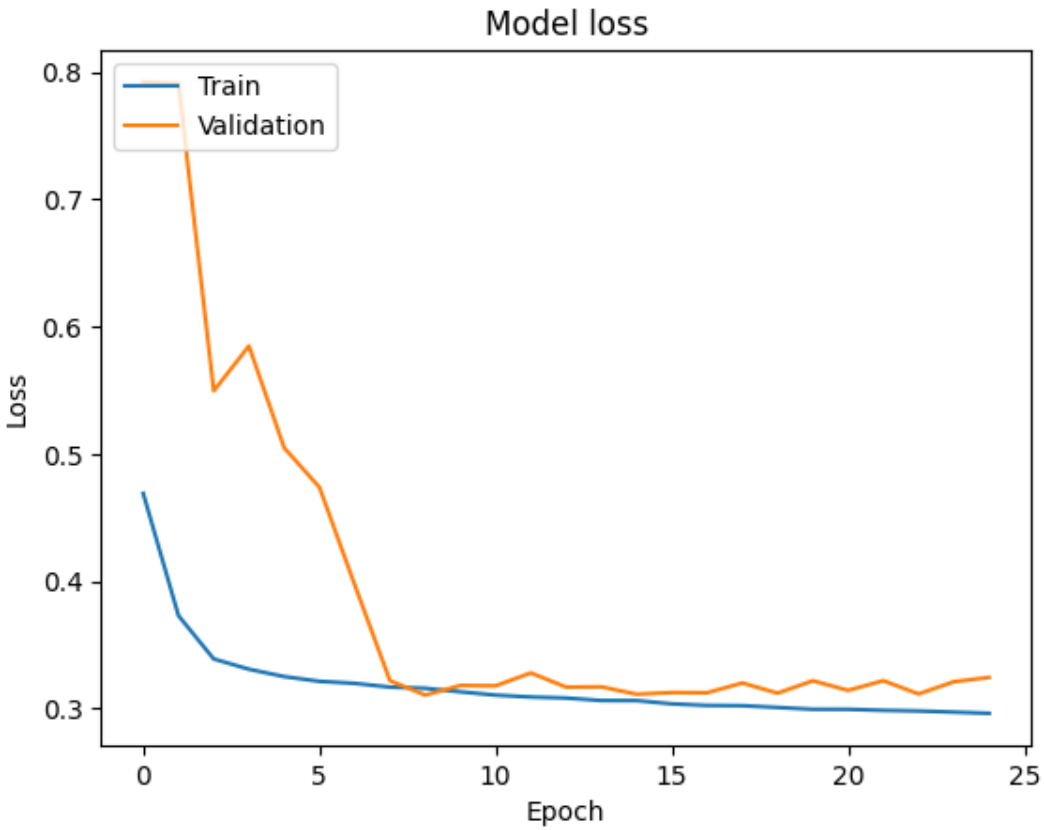

Figure: loss(without l1l2)

Image Comparisons - simpleCNN complex

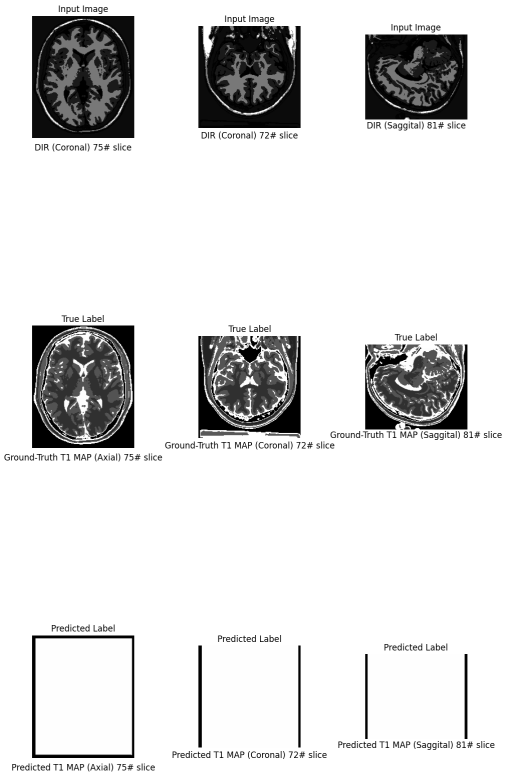

Figure: T1 Mask Map

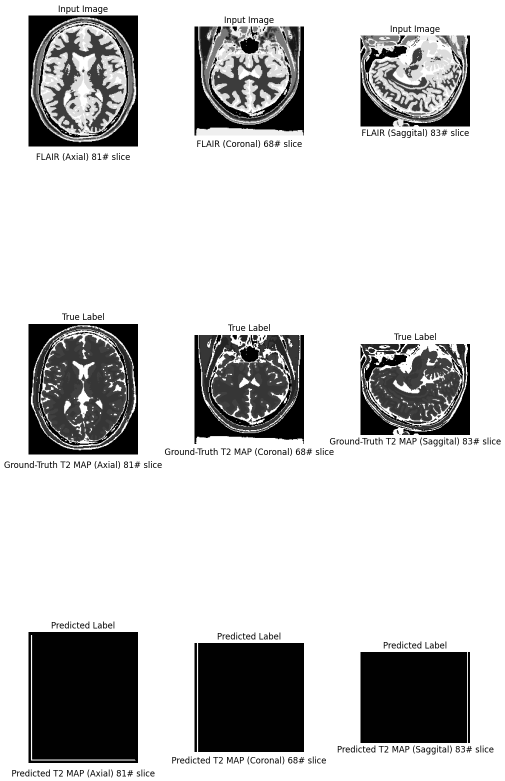

Figure: T2 Mask Map

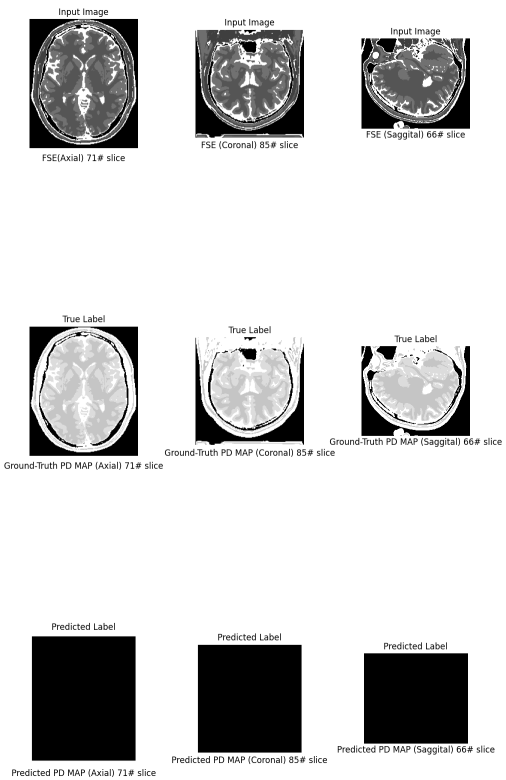

Figure: Proton Density Mask Map

Image Comparisons - simpleCNN simple

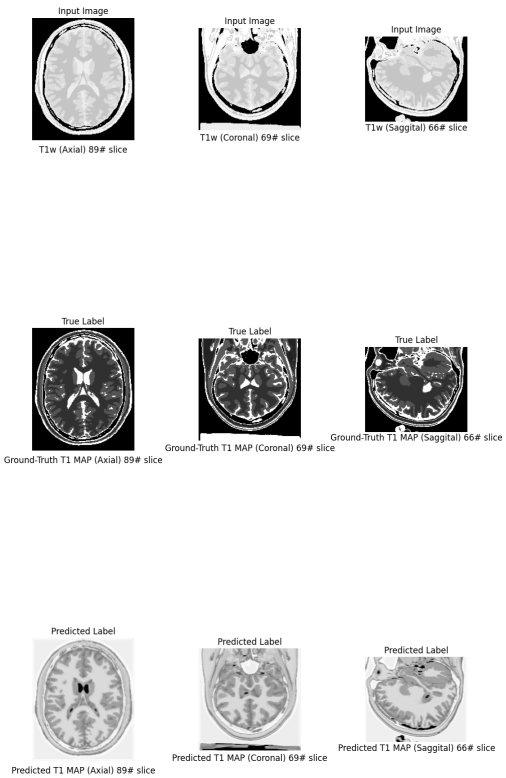

Figure: T1 Mask Map

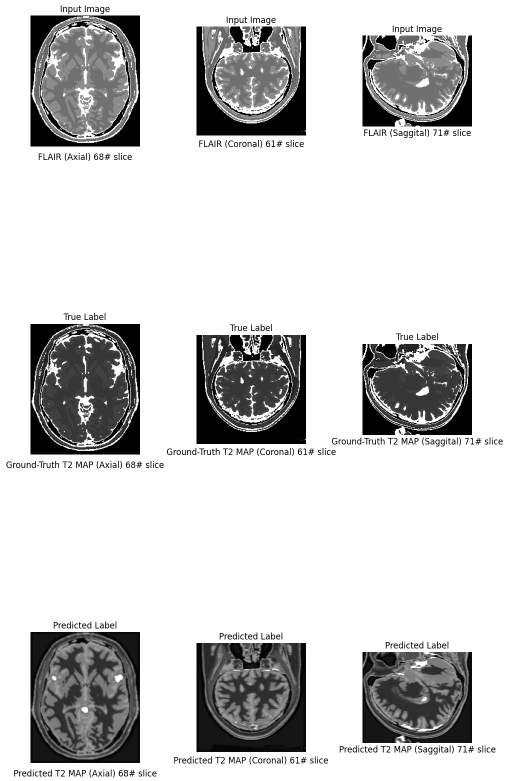

Figure: T2 Mask Map

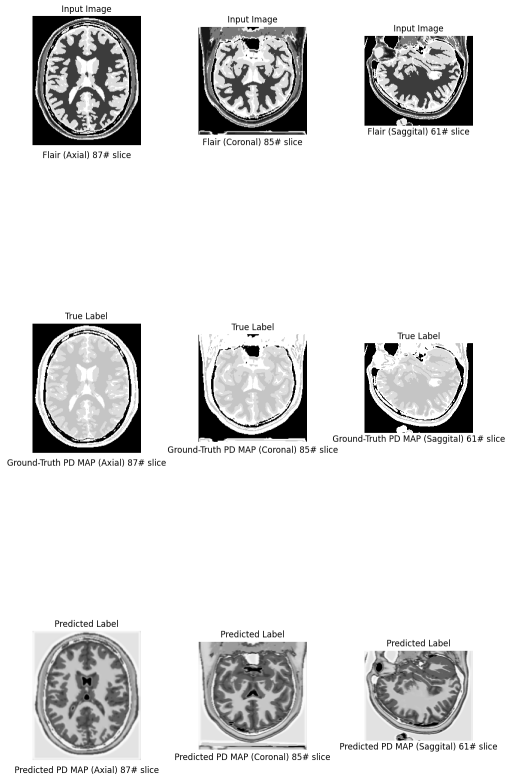

Figure: Proton Density Mask Map

Image Comparisons - Unet

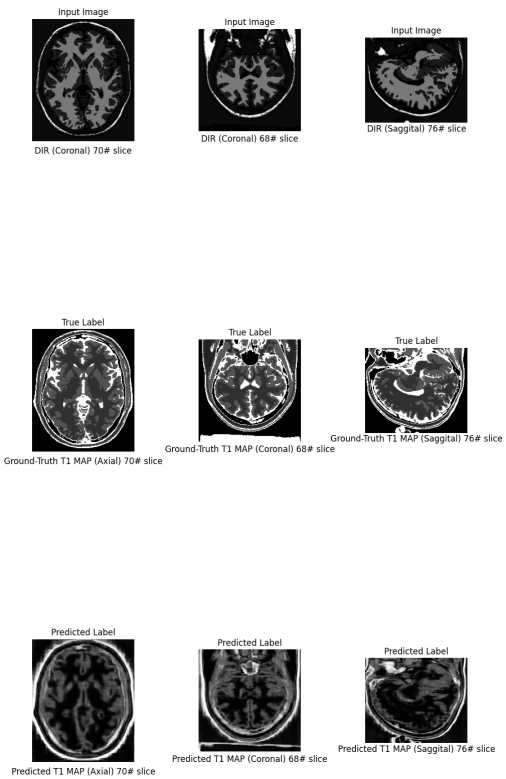

Figure: T1 Mask Map

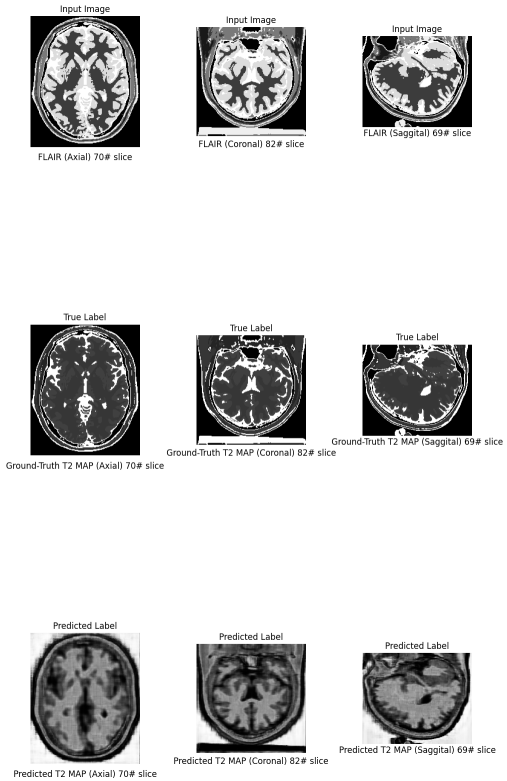

Figure: T2 Mask Map

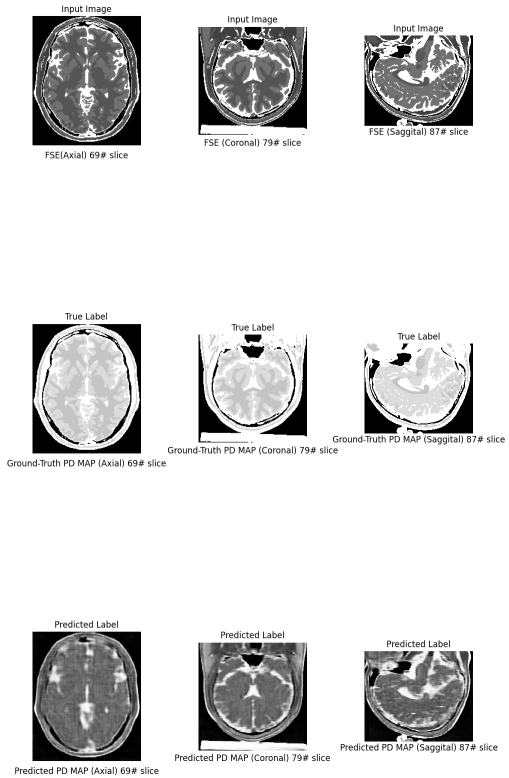

Figure: Proton Density Mask Map

- Complex Loss Results - U-net
- Complex Loss Results - Simple-CNN
- Simple Loss Results - Simple-CNN
- Simple Loss Results - U-net
- Complex Loss Results - SimpleCNN
- Complex Loss Results - SimpleCNN
- Complex Loss Results - SimpleCNN
- Complex Loss Results - SimpleCNN
- Complex Loss Results - Unet
- Complex Loss Results - Unet
- Complex Loss Results - Unet
- Complex Loss Results - Unet

6 References

References I

- Y. LeCun, "Self-supervised learning," 2018.
- B. P. M. K. M. K. F. M. R. D. Chen, L., "Self-supervised learning for medical image analysis using image context restoration," 2019.
- B. J. G. E. Fadnavis, S., "Patch2self: Denoising diffusion mri with self-supervised learning," 2020.