BMI 8400 - Final Project Report
David Ellis

Problem Statement
When training a convolutional neural network (CNN) on a medical imaging dataset, it is often desirable to augment the images to expand the dataset. Common modifications are translation, rotation, flipping, scaling, and shearing. When training a deep learning model, it is simple to think of these augmentations as separate steps. However, implementing them as separate steps requires resampling after each step. Performing multiple resampling steps leads to a loss of information and adds a substantial amount of computing time. By using linear algebra, we can compute each augmentation step as an affine transformation matrix, multiply the matrices, and then perform the combined augmentations in a single sampling step. In this project, I wrote code to compute the affine transformation matrices for translation, rotation, flipping, scaling, and shearing transformations. I then validated that my code generates identical transformations as MONAI (https://monai.io/), a popular open-source deep learning tool for medical imaging. Finally, I trained a model using my code for augmentation and compared it to an identical model trained using the MONAI code for augmentation. The results of my project showed that performing a single resampling using my code was faster, retained more of the original signal, and produced a better-performing model than using the MONAI code for augmentation.

Experimental Design
I wrote Python code to perform five different augmentations using affine matrices using PyTorch: translation, rotation, flipping, shearing, and scaling. The transformation matrices were calculated according to the voxel space rather than real-world space. The transformations were applied to the sampling grid of the image. After the sampling grid of the image had been augmented in the voxel space, the transformation of the grid from voxel space to real-world space was applied. The resulting sampling grid was then used to resample the original image according to the augmented sampling grid. All transformations, excluding translation and flipping, were applied to the center of the image rather than the image origin. This was accomplished by creating a translation matrix that translated the sampling grid to the center of the image. This translation matrix was multiplied by the desired transformation matrix (rotation, shear, or scale) and then multiplied by the inverse of the centering translation.

Centering the transformations helped keep most of the sampling grid overlapping with the original image while also simulating deviations that are potentially more realistic. For example, if a patient's head is rotated, their head is likely to be still centered in the scanner rather than rotated according to the corner voxel of the image.

Using unit tests, I verified that each of the Python functions that I wrote produced equivalent transformations to functions provided by MONAI for data augmentation. I also verified that when combining augmentations, the resulting image of multiple augmentations had an equivalent resulting affine matrix that defined the transformation from voxel to real-world space.

I also tested the impacts of performing multiple resampling augmentations as compared to performing a single resampling augmentation. I compared the two approaches in terms of computation time, signal loss, and effects on deep learning model training. To test for differences in processing time and signal loss, I generated 250 random sets of augmentation parameters, varying in the number of augmentations from 1 to 5, with 50 augmentations each. The augmentations were performed on a single T2w brain image from the Human Connectome Project Yong Adults dataset[1]. For each augmentation, I recorded the time to perform the augmentation. As a measure of signal loss, I computed the mean squared error compared to the original image after the augmented image was resampled back into the sampling grid of the original image.

To test the effect of the stepwise resampling augmentations compared to the single resampling on model training, I trained identical models using each method. The models were trained to predict T1-weighted (T1w) brain MR images from the aligned T2-weighted (T2w) images from the same subject. T2-weighted and T2-weighted MR images are often acquired in tandem as each provides complementary information about the brain's anatomy and pathology. Most brain MRI processing tools, such as FreeSurfer[2], require a T1w image to perform analysis, cortical surface generation, and spatial normalization. However, if the T1w image is corrupted (by excessive motion, for example), then such analysis cannot be performed. In such cases, it would be beneficial to be able to predict the T1w image from the T2w well enough to allow tools like FreeSurfer to produce reasonably accurate results.

The data used were from the Human Connectome Project Young Adults dataset. Eighty percent of the subjects were used for training (892 subjects) and 20 percent for validation (224 subjects). Both the T1w and T2w images were cropped to a size of 192x224x160.

During training, the parameters for each augmentation were randomly generated during image retrieval. Were randomly generated from a normal distribution of mean 0 voxels and standard deviation of 10 voxels. Rotation parameters were generated from a normal distribution of mean 0 radians and standard deviation of 0.5 radians. The flipping parameters were randomly set to True or False with an equal probability for each. The shear parameters were generated from a normal distribution with a mean of 0 and a standard of 0.1. The scale parameters were generated from a normal distribution of mean one and a standard deviation of 0.1. In addition, each of the five possible augmentations had a 50% probability of occurring at each image retrieval. These augmentation parameters were set to be conservative to avoid overly augmenting the data. The random number generator seed for PyTorch was set to be the same for both model training runs to ensure that the random permutation parameters were the same for each run.

The model I used for this comparison was MONAI's DynUNet which is based on the popular nnU-Net[3]. The model follows a standard U-Net design with an encoder that downsamples that progressively downsamples the image, an encoder that progressively upsamples the image, and skip connections between each layer in the encoder and decoder. The U-Net I used had six layers, with five downsampling convolutions and five upsampling convolutions. The number of filters for each layer was progressively increased with $2^{(i+1)}$ filters for layers i=1,2,3,4,5 and 2^8

filters for layer 6. The input and output images were normalized to a mean of zero and a standard deviation of one for each image.

Adam optimizer was used for training along with mean squared error (MSE) loss and a learning rate of 0.001. The models were trained for 250 epochs. Training, validation loss, and combined training and validation time were recorded for each epoch. The validation loss was assessed on the original images that had not been augmented.

Results

As expected, the time to perform the augmentations increased with each additional augmentation step when using the MONAI code for stepwise augmentations but remained the same when using my code to combine the augmentations and perform a single resampling step (Figure 1).

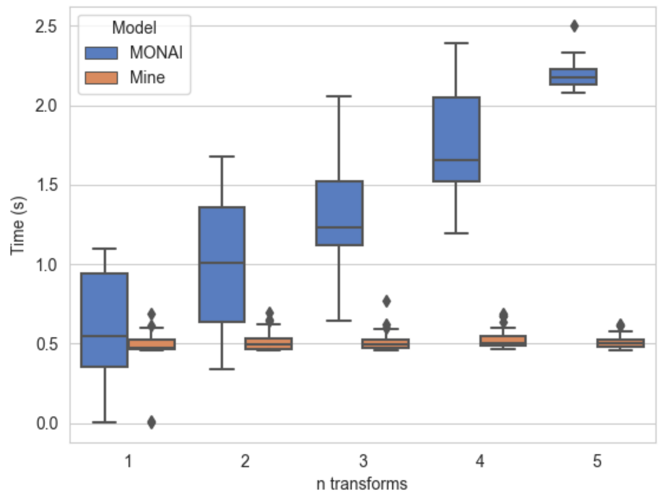

Figure 1 - Boxplot comparing the number of transformations performed and the computation time required. The blue boxplots show the augmentations using MONAI and performing a resampling after each augmentation step while the orange boxplots used my code that had a single resampling step. The results show that performing multiple resampling steps can greatly increase the computation time.

Also, as expected, the amount of signal loss for each transformation increased with each resampling step, while combining the transformations into a single resampling step reduced the amount of signal loss when two or more transformations were applied (Figure 2).

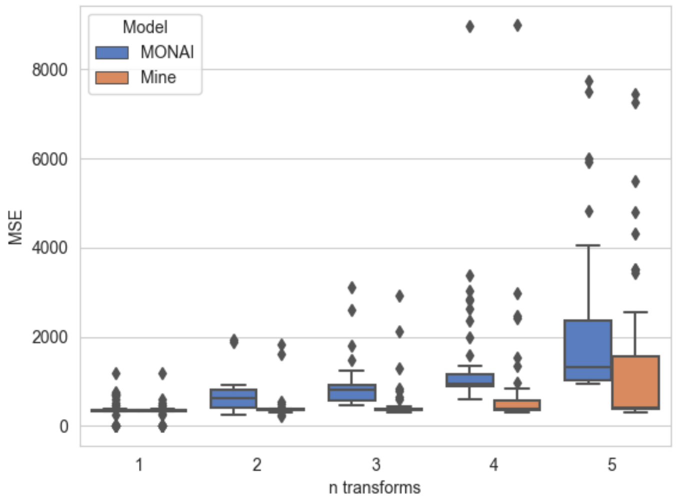

Figure 2 - Boxplot comparing the number of transformations performed and the mean squared error between the augmented and resampled image. The blue boxplots show the augmentations using MONAI and performing a resampling after each augmentation step while the orange boxplots used my code that had a single resampling step. The results show that performing multiple resampling steps can increase the mean squared error between the augmented and original images.

When comparing the MSE loss, the model trained using my code that combined augmentations into a single resampling step produced better predictions on the validation set than the model trained using MONAI's transformations (Figure 3). Surprisingly, the model trained on my code for augmenting the data also overfits less to the training data than the model trained using MONAI's code (Figure 3).

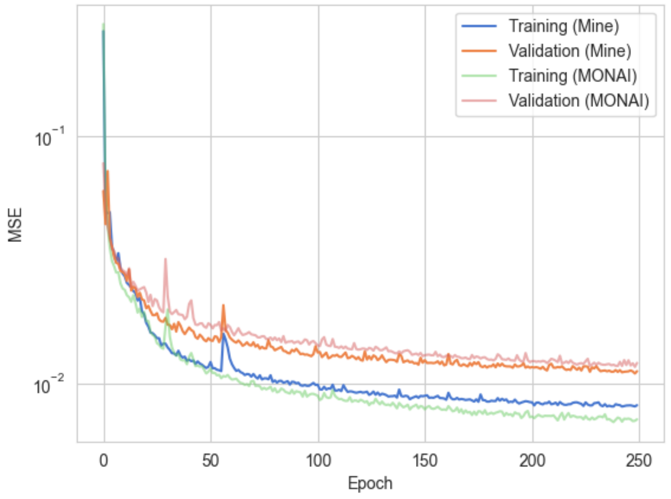

Figure 3 - Training and validation loss for the U-Net models trained to predict the T1w images from the T2w images. The model overfits less and achieves better validation loss when the resampling is done in a single step.

In addition to performing better on the validation set, the model trained using my code also had a faster training time per epoch by about 5 to 7 seconds (Figure 4).

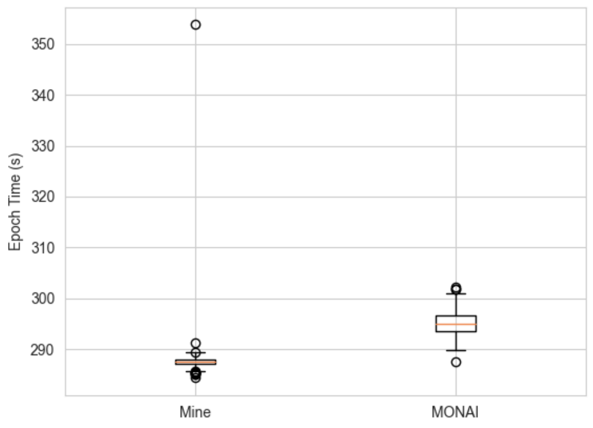

Figure 4 – Training times per epoch between the model trained using my code to perform a single resampling step and the model using MONAI code that performed multiple resampling steps. Performing multiple resampling steps added about 5-7 seconds per epoch.

Discussion and Conclusion

As expected, my project showed that performing multiple resampling steps increased the computation time, contributed to signal loss, and worsened a deep learning model's prediction for predicting T1w images from T2w images. When I started this project, I did not think that MONAI had the ability to combine augmentation transformations, but as I was looking through the documentation, I found that their transformation class called "Affine" does allow for these augmentations to be combined. This makes my work on this project somewhat redundant. However, I still learned a lot from this project. I learned that the method of translating the sampling grid to the center of the image to perform augmentations is easy and intuitive when you consider it from an affine transformation perspective. I also learned that when performing some transformations, such as flipping, it is important to consider that the origin location is the center of the corner voxel and not at the corner of the corner voxel.

Surprisingly, the hardest part of this project was working with MONAI's resampling function. MONAI is constantly changing, and the resampling method that I wanted to use for my augmentation method was transposing two of the image dimensions when I was testing it. I'm not sure what was causing this issue, but switching the resampling function fixed the issue.

I found it interesting that the model trained using a single resampling step overfit less to the training data. I would have expected the opposite, given that performing multiple resampling steps would be similar to blurring the image. I would think that a model trained on blurry images would perform would pick up on fewer details and would be hindered from overfitting.

I think it would be interesting to see if affine augmentations actually help CNN models learn. MR techs do a pretty good job of minimizing variation from things like translation and rotation within a scanner. Preprocessing can also minimize these variations in datasets so that models don't have to deal with them. To me, shearing and flipping don't seem like realistic augmentations that would be similar to the variation seen in non-augmented datasets. Scaling might be the most realistic augmentation and has been shown to increase model performance in a previous study that I found[4]. My hypothesis would be that for small datasets of less than 100 subjects, a U-Net would benefit from all affine augmentations to some degree, but that once datasets get larger than 100-500 subjects, there would be little to no benefit from doing affine augmentations beyond scaling and maybe left/right flipping. Once datasets get sufficiently large, maybe 1000-2000 subjects, I wouldn't be surprised if doing affine augmentations hurt the performance.

References

1. Van Essen DC, Ugurbil K, Auerbach E, et al. The Human Connectome Project: a data acquisition perspective. *Neuroimage*. Oct 1 2012;62(4):2222-31. doi:10.1016/j.neuroimage.2012.02.018
2. Fischl B. FreeSurfer. *Neuroimage*. 2012;62(2):774-781.
3. Isensee F, Jaeger PF, Kohl SA, Petersen J, Maier-Hein KH. nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature methods*. 2021;18(2):203-211.
4. Zhang L, Wang X, Yang D, et al. Generalizing deep learning for medical image segmentation to unseen domains via deep stacked transformation. *IEEE Trans Med Imaging*. 2020;39(7):2531-2540.