HIGH SCHOOL EDITION
Journal of Student Research
Volume 12 Issue 1 (2023)

Finding the Most Effective Data Augmentation Techniques on Brain MRI Data Using Deep Networks

Nishank Raisinghani[1] and Mason McGill[#]

[!]Dougherty Valley High School
[#]Advisor

ABSTRACT

In 2020, over 250,000 people died from brain and Central Nervous System (CNS) tumors. Brain tumors account for around 85% or more of these. As of 2021, over 16 million people in the United States have been diagnosed with some type of cognitive impairment. The goal of this paper is to find the most effective set of data augmentations to correctly classify cognitive diseases with deep networks, using structural Magnetic Resonance Imaging (MRI) data. This paper demonstrates a Greedy optimization technique to find the most effective sequence of data augmentations out of blurring, distortion, position, and red noise (an overlay augmentation displaying random clouds of noise on the images). We sought to classify 3 tumors: glioma tumors, pituitary tumors, and meningioma tumors, as well as detect if there was no tumor at all. We also classified 3 stages of Alzheimer's disease: not demented, very mildly demented, and mildly demented, to further demonstrate the effectiveness of the data augmentation sequence.

Introduction

Tumors

A brain tumor is an abnormal growth of cells that can appear in any part of the brain or skull. Brain tumors are a large problem in the world, being responsible for around 250,000 deaths worldwide as of 2020.[1] In this paper, we are only looking at classifying 3 distinct types of brain tumors: glioma, pituitary, and meningioma. Brain tumors are very dangerous as they can spread to healthy parts of the brain and can also directly interfere with other parts of the body, for example blocking spinal fluid from traveling. Brain tumors can be either benign or malignant, both being very dangerous. Benign tumors grow slowly, but still cause negative effects on brain function because they compress sections of the brain. Though it is rare, it is still very possible for a benign tumor to become malignant. Malignant tumors are cancerous, grow rapidly, and invade tissue much faster than benign tumors. Malignant tumors are often life-threatening as they affect critical parts of the brain while attacking it faster than benign tumors.[2]

Meningioma tumors are formed in the meninges, which is the lining of the brain. Pituitary tumors are formed in the pituitary gland of the brain. Glioma tumors develop in the glial cells of the brain and are more likely to be malignant than any other type of tumor. It is extremely important to classify these diseases with as much accuracy as possible due to them being life-threatening in many cases. Early detection of tumors makes treatment much easier and increases survival rates.

Alzheimer’s

Alzheimer's is a neurodegenerative disease that negatively affects a patient's memory and thinking skills. It progresses slowly, and it affects the patient's brain more over time. It is extremely prevalent in people over 60 years of age, while being extremely rare in people between their 30s and 60s. As of 2021, it was estimated over 6 million Americans over

the age of 65 had this disease. At time of writing, Alzheimer's is the 7th leading cause of death in the United States.[3] Though there is no cure for Alzheimer's there are various treatments to suppress symptoms and make the patient's quality of life much better. Alzheimer's is also incredibly hard to diagnose quickly. It is not diagnosed through any sort of brain imaging, but through a medical professional observing the actions of a patient. Being able to diagnose Alzheimer's accurately using Machine Learning can increase confidence in a doctor's diagnosis. Diagnosing Alzheimer's as early as possible in the progression would allow patients to start treating their symptoms much sooner than they usually would.

Magnetic Resonance Imaging

A Magnetic Resonance Image (MRI) is one of the most common forms of medical imaging. An MRI uses extremely powerful magnets which make protons align with these magnetic fields. It employs a radio-frequency current to stimulate the protons. When this current stops the MRI is able to create an image based on the relative strength of energy released from the protons and how quickly they release energy. Taking an MRI requires the patient to be inside of this magnetic field and stay still.[4] MRIs are much better to image soft tissues and organs, rather than bone. They also give much clearer pictures of tissues and ligaments than computed tomography (CT) scans and X-rays, two other popular brain imaging techniques. MRIs have become a staple for brain tissue imaging because of this. MRIs are usually 3-dimensional images, but MRIs can also be represented as multiple 2-dimensional slices of the 3D image. For our research, we will be using 2D MRI slices.

Neural Networks

Neural networks are a subset of machine learning algorithms. They connect multiple layers of 'neurons' through a network of parameters (weights). Neural networks are built to change their weights so that they can get the most accurate outputs possible. They do this through backpropagation and stochastic gradient descent (SGD).[5] The goal of this method is to find the minimum loss of the neural network, by using partial derivatives of the network's weights. The loss is computed through a logarithmic error function which is very closely correlated to the accuracy of the network. A neural network is built to simulate a human brain and learn just like we do. By adapting its weights through the constant processing of new data, it learns just like a human brain when we process new information.

Related Works

Our research is closely related to a 2016 study conducted by Justin Paul et. al., 2016.[6] They used a dataset of MRI images with 4 classifications of brain tumors-- glioma, meningioma, pituitary, and no tumor-- similar to our research. The study trained these images using different variations of neural networks and augmentations. It used different combinations of convolutional neural networks (CNN), fully connected neural networks (FCNN), and concatenated CNNs and FCNNs (ConcatNN) paired with preprocessing methods of none, tumor zoomed, cropped, counteracting overfitting, and tumor locating. It used these combinations strictly for classifying the images with tumors. It then measured the effectiveness of these methods with Average 5-Fold Cross Validation and got the best results by using no preprocessing and using a CNN. It also compared a CNN and FCNN without any preprocessing to classify all 4 of the classes, not just the images with brain tumors. In this experiment, the best result was achieved by using the CNN architecture with an accuracy of 89.13. This study has similarities to ours through the type of data being classified and using neural networks to classify them.

Another study conducted by Muhammed Farhan Safdar et. al., 2020 also closely follows the research we are doing.[7] This research uses a You Only Look Once (YOLO) model to detect tumor locations in brain MRI images. After finding the tumor the model classified the tumor as well. The goal of their study was to see which data augmentation was the best out of a 180-degree rotation, 90-degree rotation, cropping/scaling, horizontal flipping, vertical

flipping, shearing, Gaussian blurring, and noise. The results of their study showed that the best augmentation on this data was the 180-degree rotation with an accuracy of 96%, with the 90-degree rotation coming in as second with an accuracy of 92%. Their study looks into the effect of data augmentation on brain MRI detection and classification, similar to ours.

Methods

Data

There are two datasets being used in this research. The first is a brain tumor classification dataset,[8] labeling brain images between no tumor, glioma tumor, pituitary tumor, and meningioma tumor (as shown in Figure 1). This dataset is split into a training set and a holdout set, with 5712 and 1311 images in each set, respectively. Part of the labeled data in this set was confirmed by Dr. Shashikant N. Ube of Chandrabhaga Clinic and Nursing Home.

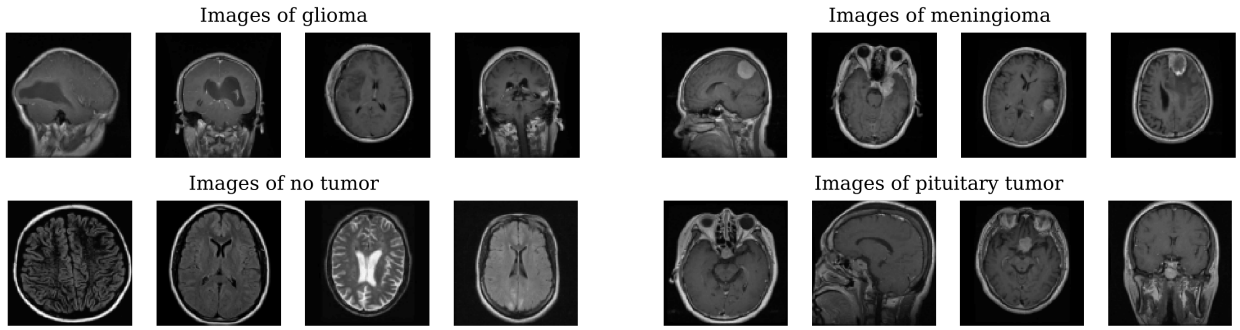

Figure 1. Example images for the tumor dataset by the class label. This dataset uses various MRI slices for the data.

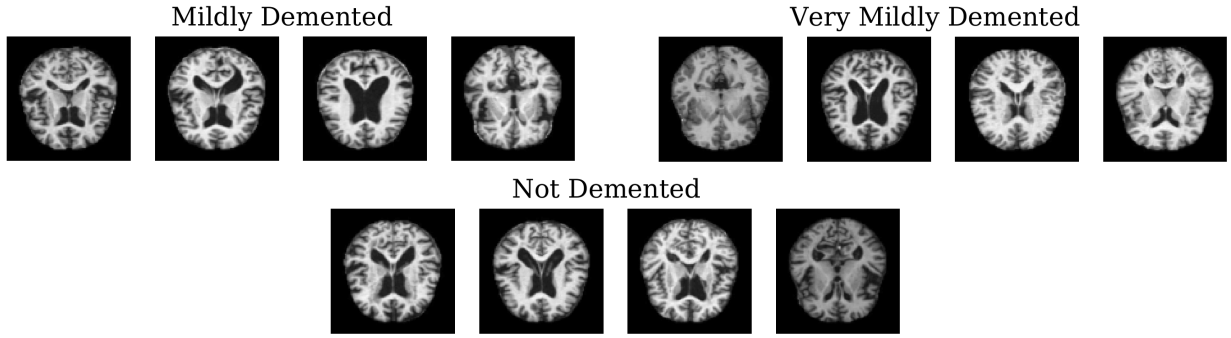

Figure 2. Example images for the Alzheimer's dataset by the class label.

The second dataset is an Alzheimer's classification dataset.[9] This dataset is labeled and split into 4 stages of Alzheimer's--mild demented, very mild demented, not demented, and moderate demented. Due to severe class imbalance, we decided not to use the moderate demented class with this dataset. Examples of this data are shown in Figure 2. We randomly divided this data into training and holdout, with 5068 training points and 1268 holdout points. Both datasets are labeled 2D Magnetic Resonance Imaging (MRI) slices, one of the most common methods of medical imaging. All of this data is being loaded in as a 224 x 224 x 3 image to normalize the size of all the data. For both these datasets, we are encoding the labels numerically to make it easy for the neural network to process.

We also used a subsampled tumor dataset, randomly taking 20% of the tumor dataset's training examples, while maintaining the same holdout set. We used this data to run our greedy optimizer and gauge our results. Before

every run, we randomly split our data into a validation set and a training set, with 30% and 70% in each set, respectively.

EfficientNet

Every experiment in this paper is trained using the same Deep Network (DN) architecture. We are using the EfficientNetB3 architecture with network weights pretrained with ImageNet.[10, 11] We are training our model to change these pretrained weights and build a more accurate network. EffecientNet is a network that scaled up the ResNet architecture by using compound scaling.[12] Compound scaling simultaneously scales the width, depth, and resolution of a convolutional neural network through set coefficients. This makes the network a lot more complex without just making the network deeper, leading to the network being more efficient. Research done by Mingxing Tan et. al. shows that EffecientNet is more accurate than other commonly used transfer learning architectures while taking up less computational power and having less trainable parameters.[10]

At the end of the EfficientNet model, we added a global max pooling layer and two Dense Layers to make the model more complex and improve its accuracy. The first Dense layer has 64 inputs and uses a GlorotNormal activation function, while the second has an input size of the number of classes being predicted, and it outputs the model's logits. The last layer is a softmax layer that takes the logits as the input and returns a vector of the normalized logits. Before each Dense layer, a Dropout layer was included to prevent overfitting of the model, with a dropout rate of 0.5 and 0.3, respectively. This network did not vary at all within any of our experiments other than the number of output units changing between datasets. The input of the network is a NumPy array of all our training images. All our networks are trained on a training and validation set.

ADAM Optimizer

For all the models we trained, we used an ADAM optimizer.[14] ADAM is an abbreviation for an Adaptive Moment Estimation Function. It utilizes a stochastic gradient descent (SGD) method that uses two distinct moments: The mean of the model's parameters (gradients), and the uncentered variance of these parameters.[5] Both of these moments are estimated by utilizing the exponential weighted averages of the model's gradients. The optimizer computes bias-corrected estimates using the moving averages of these moments.

Due to the moving averages starting as vectors of 0, it is necessary to correct for its bias towards 0. The optimizer takes 4 hyperparameters: learning rate (LR), beta 1, beta 2, and epsilon. The goal of the optimizer is to tune the model's parameters so that it minimizes the model's loss. We are using categorical cross entropy (CE) as our model's loss, which is given by this formula:

$$CE = -\sum_{i}^{N} t_i log(s_i)$$

With N being the total number of outputs, t_i being the the true y value (0 or 1), and s_i being the predicted y value (0 or 1).

We are using a learning rate scheduler for all the networks. After the first 10 epochs, the learning rate of the model exponentially changes by an exponent of -0.1 before every epoch. This allows the optimizer to take bigger learning steps at the beginning of training, then narrow down its search as it continues, giving the model a wider range of exploration, and making it more accurate and efficient.

Data Augmentation

Data augmentation is extremely important to increase a model's accuracy. It not only helps increase the amount of data the model gets to train with, but it also exposes the model to possible real-world differences in data that it could see when making predictions.

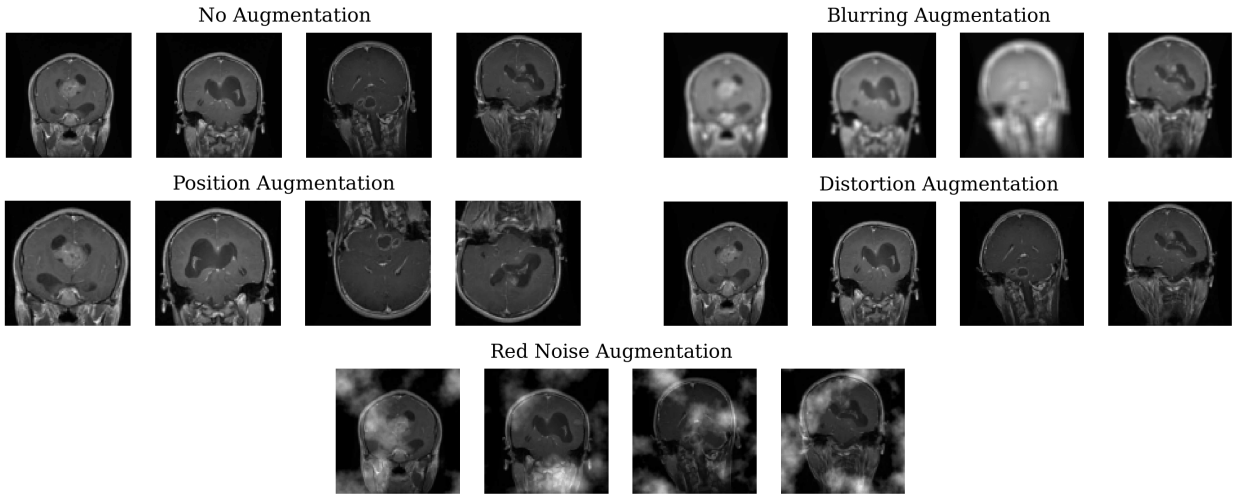

Figure 3. The red noise augmentation example in this figure is not the same strength as the one used in the greedy optimizer, in order to make the augmentation more visible in this figure (see the red noise augmentation section for more explanation).

Blurring Augmentation

For this augmentation we are doubling the amount of training data by repeating all the images in the training set again and applying the blurring augmentation to 75% of the repeated data. We did this to make sure our model's learning wasn't dependent on the blurring. For this blurring augmentation, we used the blur function from the Albumentations augmentation package (version 1.2.1).[15] There is 1 parameter for this function: blur limit (BL), which we have set at 15. For each pixel, this function takes a 15x15 matrix with a selected pixel as the middle of this kernel. The function finds the average of all the values in the kernel and makes this average value as the new value of the selected pixel. It repeats this process for every pixel in the image to create a blurred image (which can be seen in Figure 3).

Position Augmentation

The position augmentations being used include flipping, zooming, and rotating the images. We are sampling 400 images from each class, applying horizontal flipping to 50%, vertical flipping to 50%, rotations to 50%, and zooming onto 50% of these images. We are then adding these samples to the original data, creating 400 new data points for every class in the dataset. For these position augmentations, we used transformations from Augmentor's augmentation package (version 0.2.10).[16] Research done by Marcus D. Bloise et. al. shows that Augmentor is a viable data augmentation library for biomedical images .[17]

The horizontal flip function subtracts the x-coordinate of each pixel from the total width of the image as the new x-coordinate of the pixel, keeping the y-coordinate the same. The vertical flip subtracts the y-coordinate of each pixel from the total height of the image as the new y-coordinate of each pixel, keeping the x-coordinate the same.

The zoom function takes a random percentage area parameter, which we have set as 0.8 (80%). It chooses a random float value between 0.1 and 0.8 (RP). It then creates a new height (NH) and new width (NW) value for the image using the original height (h) and width (w):

$$NW = floor(w * RP)$$
$$NH = floor(h * RP)$$

It then finds a height crop value (HC) by finding a random integer between 0 and $h - NH$, and a width crop value (WC) by finding a random integer between 0 and $w - NW$. It crops off WC columns of pixels from both the left of the image and the right of the image and crops off WH rows of pixels from both the top and bottom of the image. It then resizes this cropped image to the original width and height to create the final zoomed image.

For the rotation function, we have a maximum left rotation angle and a maximum right rotation angle, which we are both setting at 5. The function finds a random value between -5 and 5 and sets it as our value of θ. We start off the rotation with a rotation matrix M:

$$\begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

We also generate a pixel coordinate matrix, I:

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

We then multiply both matrices by $I * M$, which generates the new coordinates of the pixel. This process is then repeated for every pixel, and the rotated image is generated by using these new pixel coordinates. Examples of these augmentations are shown in Figure 3.

Distortion Augmentation

The distortion augmentations are being used in a similar fashion to the position augmentations, in the sense that it uses the same sampling technique to apply the augmentation. With the distortion augmentation, we are randomly distorting 50% of the 400 samples from each class and then adding them to the original data. We are using the Augmentor Random Distortion method (version 0.2.10).[16] The augmentation first turns the image into a 4x4 grid of pixels. Each grid will be treated as a polygon. The function then selects random corners of the polygon. It moves these selected corners by a random distance in a random direction, given by a value between 0 and a given magnitude (in our case the magnitude is 7). Examples of this augmentation can be shown in Figure 3.

Red Noise Augmentation

For each instance of the red noise augmentation function we are randomly taking half of the dataset, applying red noise augmentations to that data, and again adding them to the original data to create more data points. This augmentation is produced through an overlay of filtered white noise. Before we apply this augmentation it is necessary to make sure our images are a 2D matrix of values. The first step to creating this augmentation is to create a random 2D array of values in the shape of the images, and this becomes the white noise. Then apply a Fast Fourier transform (FFT) algorithm to this white noise array.[18] All images are made up of multiple different frequencies. Through the FFT algorithm, we are able to extract the white noise's vertical frequencies (VF) and horizontal frequencies (HF). We calculate the white noise's final frequencies (FF) with the following formula:

$$FF = 2\pi\sqrt{HF^2 + VF^2}$$

We then create an array of weights to apply to these frequencies. There is an inverse relationship between the value of the frequency and the value of the weight (i.e. the higher the frequency the lower the weight, the lower the frequency the higher the weight). The formula to generate a matrix of weights (w) is as follows:

$$w = \frac{(highest\ frequency)^2}{FF^2}$$

These weights are then multiplied by the white noise frequencies, and inverse FFT transformed, to result in a pink noise augmented image. We only keep the real values of the inversely transformed frequencies, as FFT algorithms generate both real and complex outputs. To generate the red noise, we then only take the values in the top 50th percentile, making the rest of the values in the array zero. We multiply this array by 255 to make it scaled the same as our data. This red noise is then added to our original image (OI) by multiplying the red noise array (RN) by a blending coefficient (BC) or overlayed by adding it with a gain coefficient (GC).

The blending formula is as follows:

$$FI = BC * RN + (1 - BC) * OI$$

with FI being the final image.

The additive formula is as follows:

$$FI = BC * RN + OI$$

with FI being the final image.

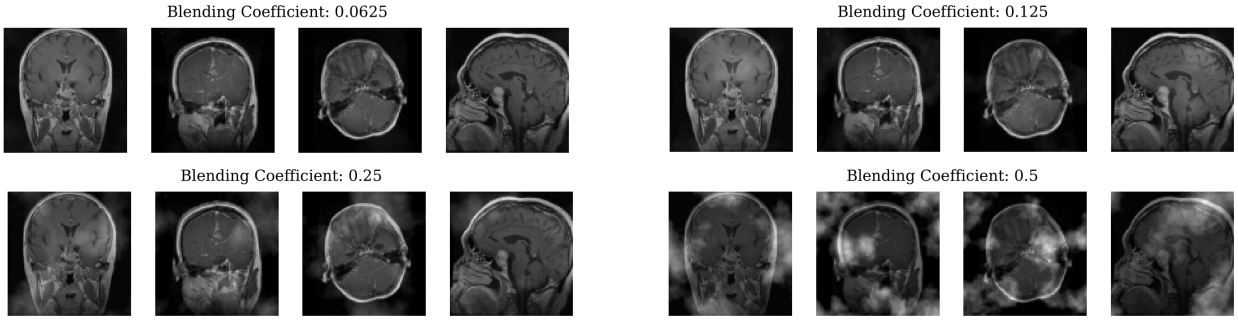

Figure 4. Example images for the blending red noise augmentation by blending coefficient**.** The blending coefficient dictates the strength of red noise shown in the image.

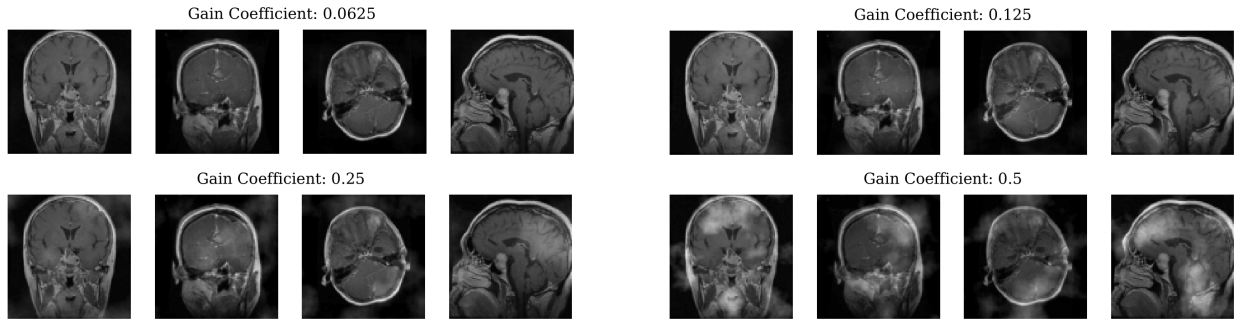

Figure 5. Example images for the additive red noise augmentation by gain coefficient. The adding coefficient dictates the strength of red noise shown in the image.

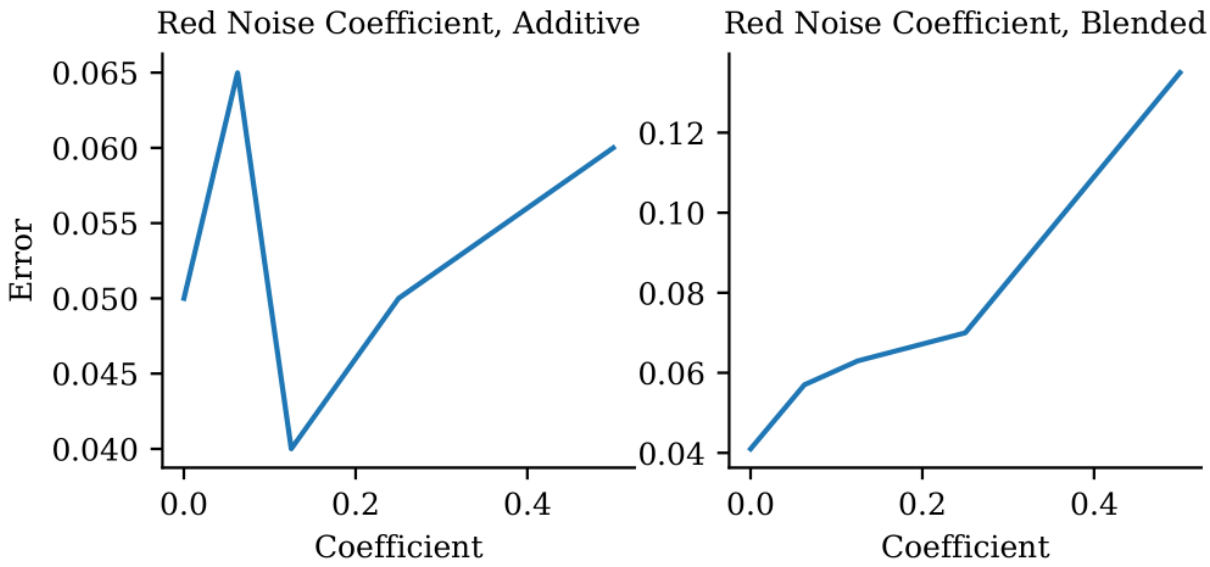

Figure 6. Red noise experiments by coefficient. The left graph is the adding coefficient vs. the model error. The right graph is the blending coefficient vs. the model error. The model error is derived from the brain tumor holdout set.

We trained five different networks using the red noise blending augmentation. For this experiment, we used a subsample of the original tumor dataset. We created 5 new datasets using the red noise augmentation function detailed above with the blending formula starting with a BC of 0.5 then 0.25, 0.125, 0.0625, and 0 (no augmentation). Examples of the augmented images of each blending coefficient are shown in Figure 4. The results of this experiment are shown in Figure 6.

Given these results, we can come to the conclusion that the blended red noise augmentation does not have much use. As the strength of the red noise decreases so does the error, meaning that the less red noise blended, the more effective the augmentation is. We then ran this same experiment with the additive formula rather than the blending formula. We used GCs of 0.5, 0.25, 0.125, 0.0625, and 0. Examples of each of these gain coefficients can be seen in Figure 5. The results of this experiment can be shown in Figure 6. For our blending coefficient of 0, we are still randomly re-sampling images to the dataset like aforementioned, just without any sort of augmentation to keep a uniform amount of data for every set. There is a slight discrepancy between the 0 coefficients for each dataset, but this can likely be due to different images from the dataset being re-sampled. We can see that the error is the lowest out of both experiments with a GC of 0.125. Because of these results, all the following red noise augmentations will be using the additive formula with a GC of 0.125.

Greedy Optimizer

A greedy search technique is much more efficient than an exhaustive search. An exhaustive search is built to try every single combination possible and select the best one out of them all. A greedy search on the other hand takes much less time. As shown in Figure 7, a greedy search selects the best possible trial at each section, builds off that, and doesn't try any other combinations. In our greedy search, we included the 4 augmentations explained: blurring, position, distortion, and red noise. Similar to Figure 7, each of the different colored boxes represents a different augmentation. We begin by training our network on 4 datasets separately, each one having one of the aforementioned augmentations applied to it. From there we choose the most accurate network (by checking the accuracy on the holdout set) to be the first 'layer' of our stack. Once we find the best first layer we run 3 trials, applying the first augmentation and then combining it with one of each of the remaining augmentations. We then find the second augmentation that has the best accuracy by checking the accuracy of the network on the holdout set and making that our second layer. We

continue this process with the remaining 'unstacked' augmentations until all the augmentations have been used simultaneously in the same dataset or until the accuracy does not improve in a succeeding layer.

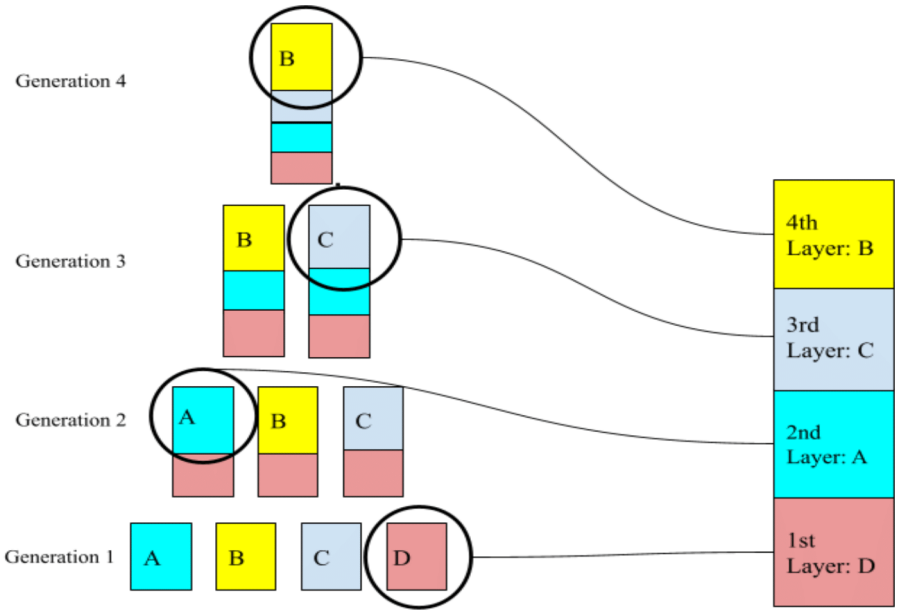

Figure 7. Visual Representation of a Greedy Optimization technique. The blocks in each generation represent a separate trial with a certain augmentation. These are not the results of the greedy optimization, just a general representation of a greedy optimizer. Each circle represents the trial with the highest accuracy, making it the best augmentation of that generation, adding it to the stack and establishing it as a 'layer'. Every trial in the succeeding generations will implement the established layers from the previous generations, in the same order, they were added to the stack.

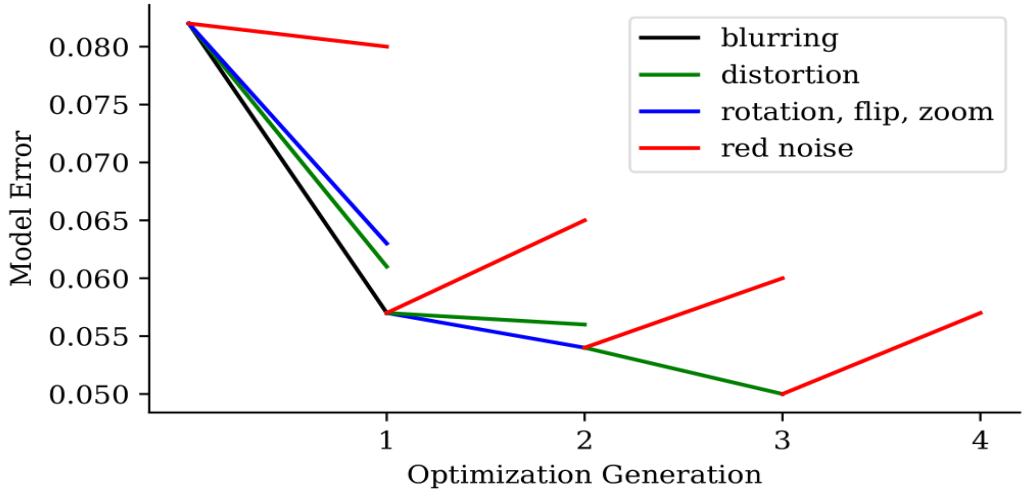

Figure 8. Greedy Optimizer Results. The points at generation 1 illustrate the error rates of each model trained on the corresponding augmentations. The line with the lowest error is the first augmentation in the stack (blurring). It continues to do this for the next 3 generations, the augmentation with the lowest error rate at each generation being added to the augmentation stack. The red noise is not included in the stack due to the error at generation 4 not improving. The lower the error (1-accuracy) the more beneficial the augmentation is to the data. The subsampled tumor dataset was used to generate these results.

Results and Discussion

All of the results in Figures 9, 10, 11, 12, and 13 are derived from model predictions on the holdout sets previously mentioned. For these figures, we are also using 3 different datasets: subsampled tumor dataset (SSTD) [a random subsample of the full tumor dataset], full tumor dataset (FTD), and the full Alzheimer's dataset (FAD). For every dataset without any augmentation, we trained the network with 100 epochs (double the amount compared to networks trained on augmented data) and halved the rate of the learning rate scheduler compared to the one explained in the ADAM Optimizer subsection. This is due to there being less data in the nonaugmented dataset compared to any of the augmented ones. For the rest of the networks, we trained them using 50 epochs and the learning rate described previously, exponentially changing every epoch after the first 10.

Augmentation Stack

We used the subsampled tumor dataset as our training data in the greedy optimizer to find the final augmentation stack. This dataset did not change at all before we applied the greedy optimizer. Figure 8 is a visual representation of the results of this greedy stack. We ran each model in the stack for 50 epochs and got the ideal augmentation stack of blurring, position, and distortion augmentations, in that respective order. The best layers were determined by the accuracy of the model on the corresponding holdout set. As shown in Figure 8, the red noise augmentation didn’t have much of an effect as an augmentation and proved to not be a useful component in the final augmentation stack. Our final augmentation stack was blurring, position, and distortion augmentations in that respective order of application.

Accuracy and Loss

A great baseline to measure the general success of a model is by measuring its accuracy and loss. We are using categorical cross entropy as our loss function. The main goal of a model is to minimize the loss, hence, the lower the loss, the better. As we can see in Figure 9, the augmentation stack does cause a significant decrease in the loss compared to without any augmentation, displaying the benefits of our augmentation stack. We can also see in Figure 10 that the accuracy of the model after the augmentation stack is applied is also higher compared to nonaugmented data in every dataset. As shown in Figure 10, our network does get more accurate with the augmentation stack compared to a nonaugmented dataset. We can also see the progression after every augmentation in the greedy stack is applied to each dataset.

The full tumor dataset does seem to dip in accuracy a bit after the third augmentation is applied, but due to the dataset already being so large, it could be a variance in the training of the model. It could also mean that the dataset is so large that the final augmentation really isn't necessary at all. Due to the size of the holdout set, this difference in accuracy could be the misclassification of 1 or 2 images, which is likely just variance in the model's learning.

Regardless, we can see a decrease in the error of each model by at least 50% in each of the datasets comparing the full augmentation stack to no augmentation at all.

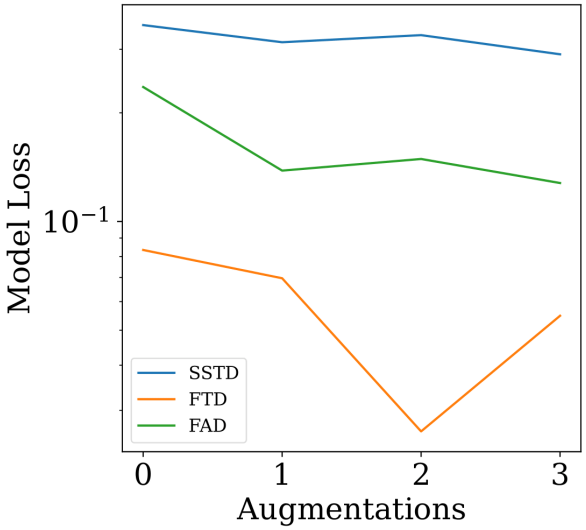

Figure 9. Loss Progression for 3 datasets. SSTD: Subsampled Tumor Dataset; FTD: Full Tumor Dataset; FAD: Full Alzheimer's Dataset. Augmentations follow the same order as how they were added to the augmentation stack. The figure shows the loss at every augmentation applied.

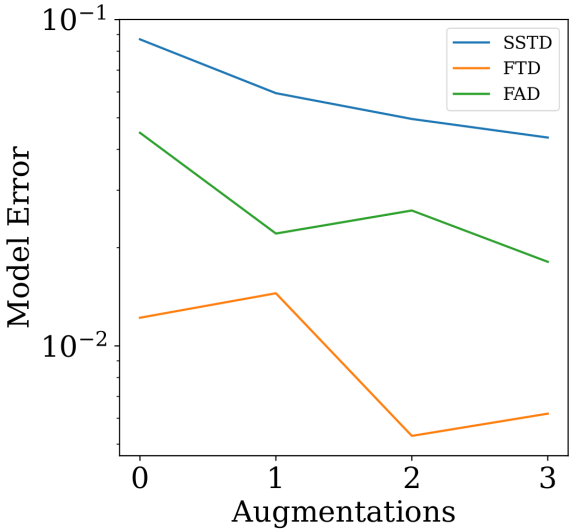

Figure 10. Accuracy Progression for 3 datasets. SSTD: Subsampled Tumor Dataset; FTD: Full Tumor Dataset; FAD: Full Alzheimer's Dataset. Augmentations follow the same order as how they were added to the augmentation stack. The figure shows the accuracy at every augmentation applied.

Confusion Matrices

A confusion matrix is a visual representation of the accuracy of a model by class. In Figure 11, we can see how the accuracy for most of the classes increases after the augmentation stack is applied, compared to without any augmentation. We can also see the progression of these accuracies after each layer of the stack is applied. Confusion matrices are extremely important because they show whether one class, in particular, is affecting the accuracy of the model. If this was the case, it could mean that the model is not very good at classifying certain classes while being extremely

good at classifying others, leading to a lot of wrong predictions and would be very detrimental to the patients being diagnosed. But our confusion matrices show that as the augmentation stack grows, all the classes increase in accuracy, meaning that the augmentations are beneficial.

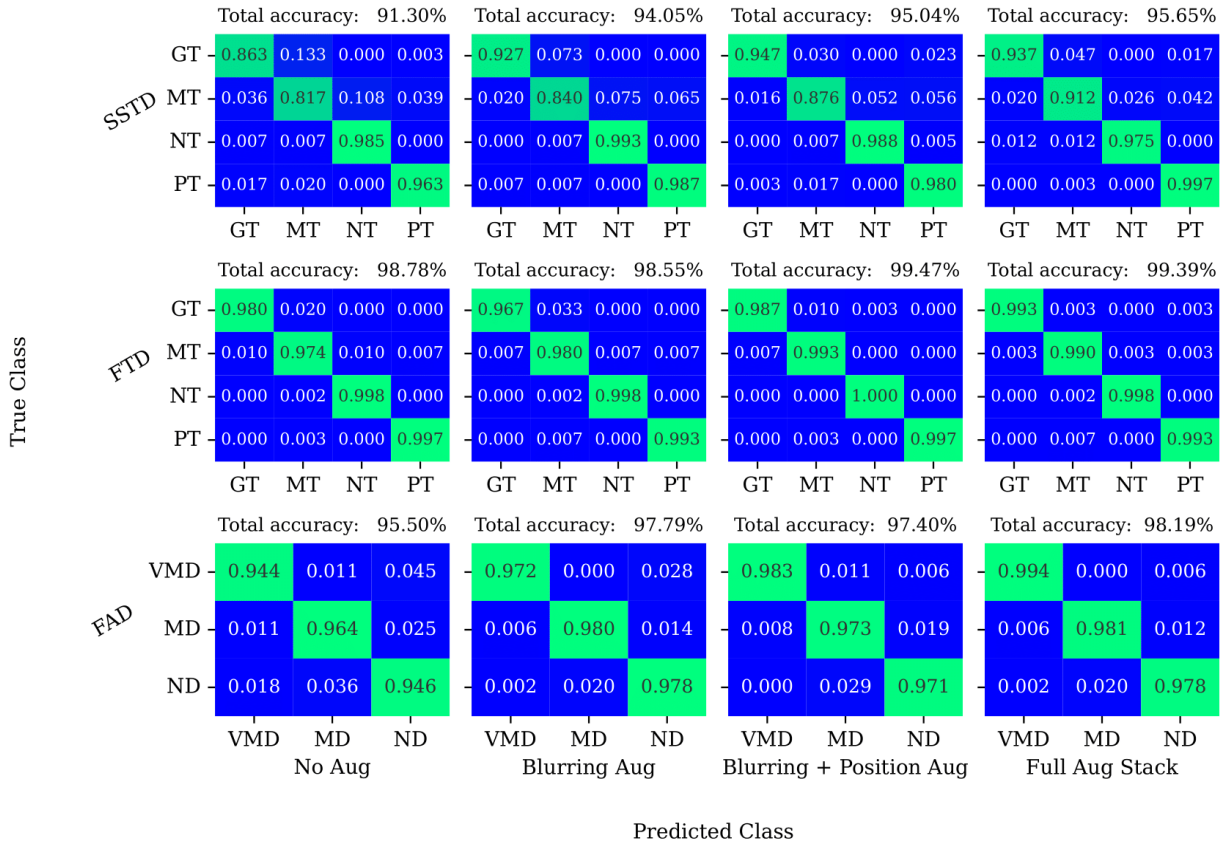

Figure 11. Confusion Matrices for 3 Datasets. Top: Subsampled Tumor Dataset; Middle: Full Tumor Dataset; Bottom: Full Alzheimer's Dataset. All results are from predictions on corresponding holdout sets. All models are trained on the corresponding training sets. GT: Glioma Tumor; MT: Meningioma tumor; PT: Pituitary Tumor; ND: Not Demented; VMD: Very Mild Demented; MD: Mild Demented

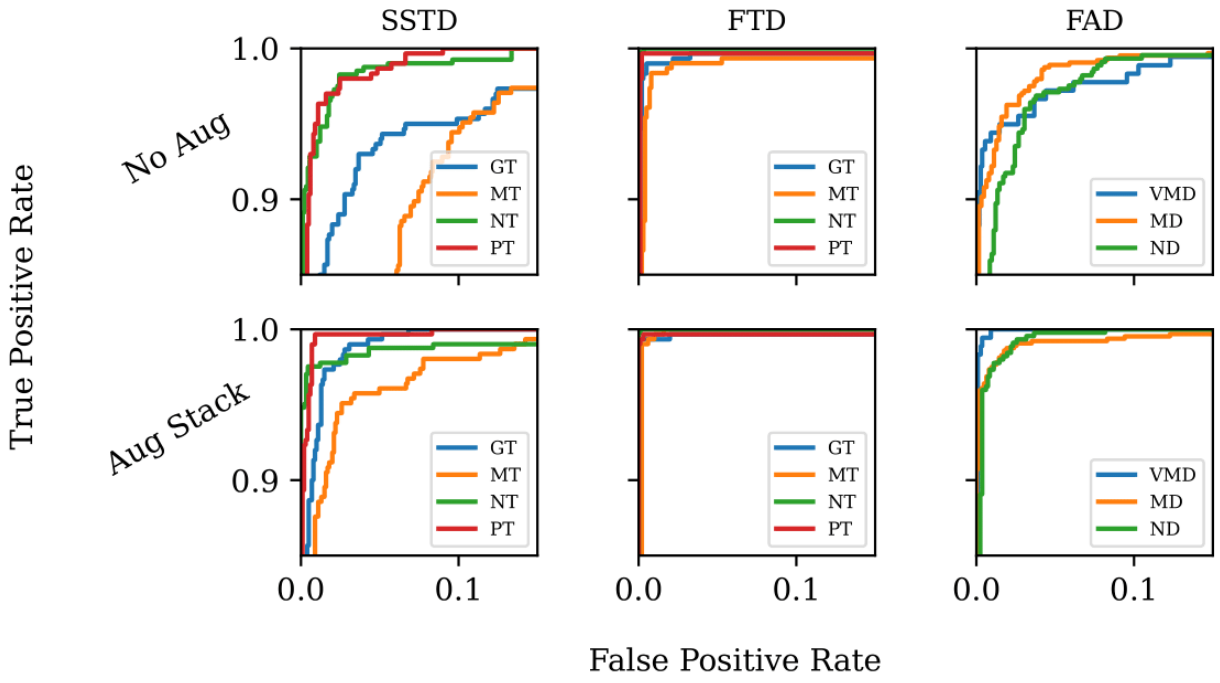

Figure 12. ROC Curves for 3 datasets; Top: Subsampled tumor dataset; Middle: Full Tumor Dataset; Bottom: Full Alzheimers Dataset. All results are from predictions on corresponding holdout sets. All models are trained on the corresponding training sets

ROC Curves

A receiver operating characteristic curve (ROC curve) compares a false positive rate (FPR) vs. a true positive rate (TPR), or sensitivity. The FPR is calculated using the false positives (FP) and true negatives (TN) using this formula:

$$FPR = \frac{FP}{FP + TN}$$

A false positive is when a model incorrectly predicts the presence of a certain condition. A true negative is when a model correctly predicts the absence of a certain condition. The TPR is calculated using the true positives (TP) and false negatives (FN) using this formula:

$$TPR = \frac{TP}{TP+FN}$$

A true positive is when a model correctly predicts the presence of a certain condition and a false negative is when a model incorrectly predicts the absence of the condition. The goal of any model is to have a high sensitivity compared to a low FPR. Higher sensitivity and lower FPR makes the ROC curve closer to the coordinates (0,1), or the top left edge of the graph. Therefore, making it the most important to take a look at the top left corner of the graph, which is shown in Figure 12. We can see that after the augmentation stack the ROC curves for all the datasets have a higher sensitivity and lower FPR compared to those without it. We can also see this in a numerical form below with the average ROC area under the curve (ROC AUC) for every class without any augmentation compared to the augmentation stack. A higher ROC AUC means that your model has a higher sensitivity and lower FPR. As we can see in Table 1, the ROC AUC increases for every dataset from no augmentation to the applied augmentation stack.

Table 1. Average ROC area under the curve values for each dataset, with and without augmentation

Dataset	ROC AUC No Aug	ROC AUC Aug Stack
SSTD	0.985	0.996
FTD	0.998	0.999
FAD	0.994	0.998

Calibration Curves

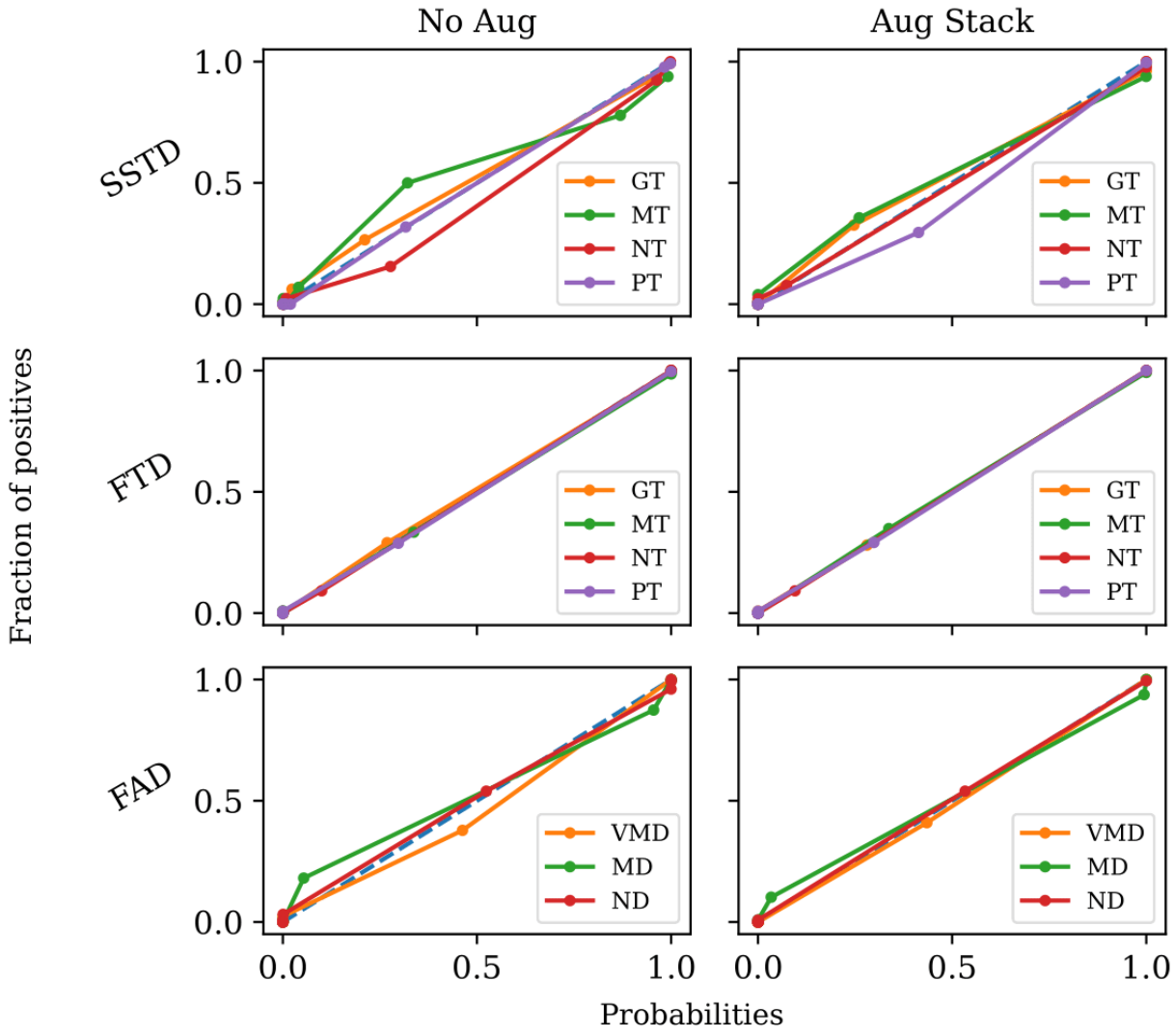

Figure 13. Calibration Curves for 3 Datasets. Top: Subsampled Tumor Dataset; Middle: Full Tumor Dataset; Bottom: Full Alzheimer's Dataset. All results are from predictions on corresponding holdout sets. All models are trained on the corresponding training sets. Every graph shown are results after the model was calibrated.

A model's calibration shows how accurately a model's probability distribution represents the results. Most deep learning classification models output a distribution of the probability of each class being positive. We extract this matrix of probability distributions from the logits layer near the end of our network. Figure 13 shows the calibration curves for all the classes in the 3 datasets post model calibration, with the augmentation stack applied, and without any augmentation at all. All of these results are calibrated using temperature scaling. By training a logistic regression with this extracted logits matrix (LM) and true values, it produces a temperature (T) coefficient value. The logits matrix is then divided by the T value and computes a calibrated probability distribution (CPD) for every prediction.

$$CPD = \frac{e^{LM}}{LM_1 + LM_2 + \ldots + LM_n}$$

The predicted probabilities should be representative of the actual results. For example, of all datapoints that a perfectly calibrated model predicts a 77% probability of being positive, 77% of those datapoints should be positive. Therefore, a perfectly calibrated model's probability distribution would show how certain the model truly is when making a prediction. The certainty of the model is especially important in the field of medical diagnosis so patients can know what the exact chance is that they are positive for a certain condition.

The ideal calibration curve should be an x = y line, meaning that every prediction probability is a true representative of the chance of a datapoint being positive. These curves are all split into 15 bins, with each of these bins being split in quantiles. The results of Figure 13 show that the network trained on the augmentation stack is calibrated better than without any augmentation, as the curves fit the x = y line much closer. This increase in calibration is also shown numerically, through the models' average expected calibration error (ECE) scores, shown in Table 2.

Table 2. Average Expected Calibration Error Scores for each dataset, with and without augmentation

Dataset	ECE Score No Aug	ECE Score Aug Stack
SSTD	0.0066	0.0029
FTD	0.0005	0.0002
FAD	0.0011	0.0005

The table shows a definite decrease in the ECE score after augmentation is applied compared to no augmentation. This affirms the results shown in the graph that the model calibrates much better when trained on data with the augmentation stack applied.

Conclusion and Future Work

Our augmentation stack clearly improved the accuracy of our networks' predictions, having an immensely positive effect on the certainty and sensitivity of the model as well. We can conclude that a data augmentation stack of blurring, position, and distortion augmentation is beneficial to the precision and accuracy of a deep network when predicting cognitive diseases.

In the future, this augmentation stack can be used for all types of classifications using brain MRIs. This greedy optimizer can also be used to find the most ideal augmentation stack for a specific dataset. Augmentations can be added to the optimizer as well, for a more complex stack. We also came to the conclusion that the red noise doesn't have much of a benefit as an augmentation. The red noise can still be expanded upon or have other use cases in the field. Research can also be done into these augmentations on different types of brain imaging (i.e., CT scans, fMRI).

Acknowledgments

Thanks to my advisor Mason McGill for providing me with extremely insightful mentorship. Also, thanks to my father, Ravi Raisinghani, for his endless motivation and guidance.

References

1. *Brain Tumor - Statistics*. (2018, March 20). Cancer.net. https://www.cancer.net/cancer-types/brain-tumor/statistics
2. *Brain Tumor*. (2022). Www.hopkinsmedicine.org. https://www.hopkinsmedicine.org/health/conditions-and-diseases/brain-tumor
3. Centers for Disease Control and Prevention. (2022). *Leading causes of death*. Centers for Disease Control and Prevention. https://www.cdc.gov/nchs/fastats/leading-causes-of-death.htm
4. NIBIB. (2018, July 17). *Magnetic Resonance Imaging (MRI)*. National Institute of Biomedical Imaging and Bioengineering. https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri
5. Amari, S. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, *5*(4-5), 185–196. https://doi.org/10.1016/0925-2312(93)90006-o
6. Paul, J. S., Plassard, A. J., Landman, B. A., & Fabbri, D. (2017). Deep learning for brain tumor classification. *Medical Imaging 2017: Biomedical Applications in Molecular, Structural, and Functional Imaging*. https://doi.org/10.1117/12.2254195
7. Safdar, M., Kobaisi, S., & Zahra, F. (2020). A Comparative Analysis of Data Augmentation Approaches for Magnetic Resonance Imaging (MRI) Scan Images of Brain Tumor. *Acta Informatica Medica*, *28*(1), 29. https://doi.org/10.5455/aim.2020.28.29-36
8. Nickpravar, M. (n.d.). *Brain Tumor MRI Dataset*. Www.kaggle.com. https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset
9. Pinamonti, M. (n.d.). *Alzheimer MRI 4 classes dataset*. Www.kaggle.com. Retrieved October 9, 2022, from https://www.kaggle.com/datasets/marcopinamonti/alzheimer-mri-4-classes-dataset
10. Tan, M. & Le, Q.. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Proceedings of the 36th International Conference on Machine Learning, in Proceedings of Machine Learning Research 97:6105-6114 Available from https://proceedings.mlr.press/v97/tan19a.html.
11. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y
12. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
13. *tf.Module | TensorFlow v2.10.0*. (2022). TensorFlow. https://www.tensorflow.org/api_docs/python/tf/Module
14. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. https://doi.org/10.48550/arXiv.1412.6980
15. Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., & Kalinin, A. A. (2020). Albumentations: Fast and Flexible Image Augmentations. *Information*, *11*(2), 125. https://doi.org/10.3390/info11020125
16. D Bloice, M., Stocker, C., & Holzinger, A. (2017). Augmentor: An Image Augmentation Library for Machine Learning. *The Journal of Open Source Software*, *2*(19), 432. https://doi.org/10.21105/joss.00432
17. Bloice, M. D., Roth, P. M., & Holzinger, A. (2019). Biomedical image augmentation using Augmentor. *Bioinformatics*, *35*(21), 4522–4524. https://doi.org/10.1093/bioinformatics/btz259
18. Nussbaumer, H. J. (1981). The Fast Fourier Transform. *Fast Fourier Transform and Convolution Algorithms*, 80–111. https://doi.org/10.1007/978-3-662-00551-4_4