ESTIMATING PREDICTIVE UNCERTAINTY IN DEEP LEARNING SEGMENTATION FOR DIFFUSION MRI

ESTIMATION OF PREDICTIVE UNCERTAINTY IN THE SUPERVISED SEGMENTATION OF MAGNETIC RESONANCE IMAGING (MRI) DIFFUSION IMAGES USING DEEP ENSEMBLE LEARNING

BY

BRIAN MCCRINDLE, B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2021) McMaster University

(Electrical and Computer Engineering) Hamilton, Ontario, Canada

TITLE: ESTIMATION OF PREDICTIVE UNCERTAINTY IN THE SUPERVISED SEGMENTATION OF MAGNETIC RESONANCE IMAGING (MRI) DIFFUSION IMAGES USING DEEP ENSEMBLE LEARNING

AUTHOR: Brian McCrindle
B.Eng. (Engineering Physics),
McMaster University, Hamilton, Canada

SUPERVISOR: Dr. Michael Noseworthy

NUMBER OF PAGES: xv, 134

Abstract

With the desired deployment of Artificial Intelligence (AI), concerns over whether AI can "communicate" why it has made its decisions is of particular importance. In this thesis, we utilize predictive entropy (PE) as an surrogate for predictive uncertainty and report it for various test-time conditions that alter the testing distribution. This is done to evaluate the potential for PE to indicate when users should trust or distrust model predictions under dataset shift or out-of-distribution (OOD) conditions, two scenarios that are prevalent in real-world settings. Specifically, we trained an ensemble of three 2D-UNet architectures to segment synthetically damaged regions in fractional anisotropy scalar maps, a widely used diffusion metric to indicate microstructural white-matter damage. Baseline ensemble statistics report that the true positive rate, false negative rate, false positive rate, true negative rate, Dice score, and precision are 0.91, 0.091, 0.23, 0.77, 0.85, and 0.80, respectively. Test-time PE was reported before and after the ensemble was exposed to increasing geometric distortions (OOD), adversarial examples (OOD), and decreasing signal-to-noise ratios (dataset shift). We observed that even though PE shows a strong negative correlation with model performance for increasing adversarial severity ($\rho_{AE} = -1$), this correlation is not seen under distortion or SNR conditions ($\rho_D = -0.26$, $\rho_{SNR} = -0.30$). However, the PE variability (PE-Std) between individual model predictions was shown to be a better indicator of uncertainty as strong negative correlations between model performance and PE-Std were seen during geometric distortions and adversarial examples ($\rho_D = -0.83$, $\rho_{AE} = -1$). Unfortunately, PE fails to report large *absolute* uncertainties during these conditions, thus restricting the analysis to correlative relationships. Finally, determining an uncertainty threshold between “certain” and “uncertain” model predictions was seen to be heavily dependant on model calibration. For augmentation conditions close to the training distribution, a single threshold could be hypothesized. However, caution must be taken if such a technique is clinically applied, as model miscalibration could nullify such a threshold for samples far from the distribution. To ensure that PE or PE-Std could be used more broadly for uncertainty estimation, further work must be completed.

Dedicated to The Homies.

Acknowledgments

Taking on the challenge of graduate studies can be a daunting task, but I am eternally grateful for the support I had along the way. In particular, I would like to extent many thanks to my supervisor Dr. Michael Noseworthy who provided invaluable knowledge regarding medicine, medical imaging, machine learning, and continued to motivate me in the right directions. The ability to explore research in the ways I did was only possible with the unwavering support I had from Mike.

Additionally, I owe great thanks to Dr. Katherine Zukotynski, who consistently went out of her way to check up on me during this process. Our discussions about life, academia, and medicine will continue to motivate me for many years to come. Further, I would like to extend my gratitude to Dr. Thomas Doyle, who supported my research, gave critical feedback, and gave unique perspectives that shaped the progression of this project.

I would also like to thank a dear friend of mine, Ian Waudby-Smith, for putting up with my ridiculously simple "how do you not know this already"-type statistics questions. If I learned anything from this process, it's that I'm not a statistician! Thankfully, we have people like Ian for that.

Lastly, I would like to thank all my friends, family, and lab mates for making this fun! In particular, I would like to give a shout-out to Nick Simard for his support on data processing and data curation. Without his help, this project would have been in a very different place. His willingness to get involved and support my MASc was both unexpected and warming. I gained both a colleague and a friend. This has been a blast, and I'm excited to see what comes next!

Contents

List of Figures

List of Tables

Notation, Definitions, and Abbreviations

Notation

$A \leq B$	A is less than or equal to B

Abbreviations (Alphabetical Order)

AI	Artificial intelligence
AD	Axial Diffusivity
CNN	Convolutional Neural Network
DL	Deep Learning
DTI	Diffusion Tensor Imaging
XAI	Explainable Artificial Intelligence
FA	Fractional Anisotropy
IID	Independent and Identically Distributed
ML	Machine Learning
MRI	Magnetic Resonance Imaging

MD	Mean Diffusivity
MC	Monte-Carlo
NN	Neural Network
OOD	Out-of-Distribution
PCA	Principal Component Analysis
PE	Predictive Entropy
PMDSC	Peak Mean Dice Score
RD	Radial Diffusivity
SNR	Signal-to-Noise Ratio
TTA	Test-Time-Augmentation
TC	Transposed Convolution
VI	Variational Inference
WM	White Matter

Chapter 1

Introduction

1.1 Background

Since the inception of statistical learning, otherwise known as machine learning (ML), its use has become increasingly ubiquitous. The ability to derive important conclusions from multivariate datasets through robust statistical methods has unequivocally improved our quality of life [4]. With information becoming more accessible, we have entered into the age of "Big Data," and as such, many are in the pursuit to leverage large datasets for scientific discovery and industrial purposes. An application of particular interest is medical imaging, where ML research has led to improved image reconstruction qualities, faster acquisition speeds, and the automated detection and diagnoses of abnormalities and disease. However, there can be an absence of true model understanding once models become increasingly complex. This becomes of particular importance with the advent of deep learning (DL), a method for learning representations within data with multiple levels of abstraction [5]. DL has revolutionized the fields of computer vision [6], natural language processing [7], and biochemistry [8], to name a few. Along with their unprecedented value, ML models can drive increases in inequality [9], produce spurious outputs [10], and result in unfavourable medical diagnoses for the most vulnerable [11]. As such, it has lead to professionals, users, and the general public to acquire a source of reasonable doubt that machine learning models do not always have their best interests at heart. Therefore, a small group of researchers have decided to dedicate their careers to *Explainable Artificial Intelligence (XAI)* [12], developing new models, metrics, and frameworks to derive a deeper understanding of these inherently complicated, non-linear, self-optimizing models.

For the purposes of the following thesis, we narrow our focus of XAI to the application of supervised segmentation and uncertainty estimation with magnetic resonance imaging (MRI) data. A completely non-invasive, non-ionizing imaging modality, MRI can be used to acquire spatial and geometric information of the human body with excellent soft-tissue contrast. This makes MRI a technology of choice for radiologists who want to image even some of the smallest structures with no risk to the patient. Radiologists have an invested interest in wanting to accelerate their workflow in order to provide streamlined healthcare pipelines for their patients. In time-sensitive scenarios, radiologists can be asked to interpret an MRI scan with only a few minutes to make a decision. This can lead to errors which manifest in poorer clinical outcomes. Therefore, there is a clear need to implement ML techniques into the clinical space to augment the radiologist's workflow in order to improve both the speed and quality of medical image analysis.

The proper clinical implementation of ML systems comes with an important assumption that these models will provide accurate, reliable, and trustworthy predictions. A key component of a medical professional's workflow is the documentation of what they are able to interpret within the image. If an implemented model is unable to communicate *why* it arrived at a particular predictive outcome, it provides no use to the user when they must use them in high-risk scenarios. Furthermore, a model that produces an output that opposes the radiologist leads to a crossroads - should the clinician trust their initial opinion, default to the algorithm, or bring in additional human supervision? The action made by the clinician will ultimately rest on the trust they have in the algorithm and their experience within the field [13].

We attempt to build upon the current body of work focused on utilizing interpretability methods for DL segmentation tasks in the context of neuroimaging. Interpretable predictive models are models in which their reasoning processes are understandable by humans [14]. In this thesis, we utilize the work done by Lakshminarayanan *et al.* [15] to derive predictive uncertainty estimates through an ensemble of deep neural networks to foster a notion of trust in the ML model. We apply this technique to detect and segment microstructural white-matter (WM) brain damage in patients who experience mild traumatic brain injury (mTBI). We employ an MR imaging technique known as diffusion tensor imaging (DTI) to measure changes in the patient's water diffusion along anatomically determined WM tracts. Models were trained using synthetic mTBI patient data, whereby local reductions in a patient's fractional anisotropy (FA) scalar map was applied within small WM regions, a topic further discussed in chapter 5. The goal was to train models to segment difficult to find damaged regions in FA maps, a task that cannot currently be done by radiologists. Finally, we report the predictive entropy (PE) associated to each prediction to give an overall indication of the model's predictive confidence and showcase how PE changes under various test-time augmentation conditions. Test-time refers to experiments conducted after training is complete.

1.2 Motivation

The motivation for developing models that aggregate sources of error lies in the purpose of such a system. These systems can be used in a wide variety of areas, one of which includes medical imaging. The hypothesis is that aggregated predictions will improve medical AI models and reduce medical errors when predictions are coupled with estimates of uncertainty. For the purposes of this thesis, we attempt to understand how an uncertainty metric such as PE can give the user an indication of model confidence under test-time conditions that alter the testing (clinical) distribution, a scenario that is likely to occur in the real-world. Though the applications in which this methodology could be used are variable, we attempt to answer this hypothesis in the context of mTBI.

In North America alone, over 1.7 million people are affected by mTBI each year [16]. Typically, victims are left with a vague diagnosis of their condition since there is no quantifiable way to understand the patient's injury clinically. In this work, we explore an uncommon application of deep learning in neuroimaging with the intent to arrive at reliable predictive estimates of WM microstructural damage. Currently, microstructural WM abnormalities are completely undetected by current radiology workflows. With this information, patients can potentially undergo targeted rehabilitation plans to accelerate their recovery process. This thesis provides a stepping stone in the direction for improved clinical outcomes for mTBI patients.

1.3 Contributions

The author has contributed to the field of medical image analysis and XAI in the following ways:

1. Provided a review of the available predictive uncertainty methods that can be applied to deep learning and medical imaging for segmentation tasks through the publication: **B. McCrindle**, K. Zukotynski, T.E. Doyle, M.D. Noseworthy. *A Radiology Focused Review of Predictive Uncertainty for AI Interpretability in Computer-Assisted Segmentation.* Radiology: Artificial Intelligence.**Accepted August 25th, 2021**.

2. Actively engaged with the research community through the following accepted conference submissions:

   (a) **B. McCrindle**, N. Simard, T.E. Doyle, M.D. Noseworthy. *Applying Deep Ensemble Learning to Quantify Model Uncertainty for mTBI Brain Segmentation.* St. Joseph's Healthcare, **Celebrate Research Day**, 2020. Hamilton, Ontario [**Accepted Abstract, Poster, Oral Presentation**].

   (b) **B. McCrindle**, N. Simard, T.E. Doyle, M.D. Noseworthy. *A Deep 2D-UNet Ensemble for the Segmentation of Microstructural White Matter Damage in mTBI Patients using Diffusion Tensor Imaging.* **Imaging Network of Ontario (ImNO)**, 2021. [**Accepted Abstract, Pitch Presentation**].

   (c) **B. McCrindle**, N. Simard, E. Samson, E. Danielli, T.E. Doyle, M.D. Noseworthy. *Microstructural White Matter Segmentation in Mild Traumatic Brain Injury Patients using DTI and a Deep 2D-UNet Ensemble.* **International Society for Magnetic Resoance in Medicine (ISMRM)**, 2021. [**Poster Presentation**].

1.4 Outline of Thesis

This thesis presents the work done to apply ensemble networks and predictive uncertainty for improved predictive performance and model trust in the context of medical image analysis.

Chapter 2 discusses the technical aspects of magnetic resonance imaging and focuses on introducing diffusion tensor imaging. Specifically, we discuss how information is moved from voxel-space to pixel-space and the necessary steps taken for proper data preparation.

Chapter 3 touches briefly on the history of statistical learning to the development of deep learning. We provide the reader with the technical aspects of convolutional neural networks (CNNs), the backbone of all modern computer vision systems. Further, we dive into various CNN architectures that can be used for biomedical segmentation tasks.

Chapter 4 discusses the difference between interpretability and explainability and how this distinction is not widely used within the current literature. Post-hoc interpretability is introduced, leading to a literature review focused on current methods to estimate uncertainty from neural network predictions in the context of segmentation. Lastly, an uncertainty metric known as predictive entropy (PE) is discussed.

Chapter 5 describes the pre-processing methodologies designed to generate synthetic mTBI data and prepare such data for model training, validation, and testing.

Chapter 6 outlines the experiments completed for in-domain dataset shift and out-of-distribution testing. Results for each experiment are communicated through the correlations between ensemble performance, PE, and PE-Std.

Chapter 7 provides a discussion to interpret the results reported in chapter 6. A summary indicating the main takeaways from the thesis are included.

Chapter 8 concludes the thesis with a discussion for future work.

Chapter 2

Magnetic Resonance Imaging

2.1 Background

Magnetic Resonance Imaging (MRI) is a non-invasive, non-ionizing radiological imaging modality that leverages the quantum mechanical property of spin angular momentum of an atomic nucleus. The term “magnetic” refers to the use of external and internal magnetic fields, where “resonance” indicates the requirement to match radiofrequency pulses to the precessional frequency of the spin of the atom of interest in order to obtain a signal. This signal is spatially encoded into three orthogonal magnetic field gradients which are then used to reconstruct the measured physiological environment.

The phenomenon that takes place during acquisition can be interpreted using two popular descriptions: quantum mechanics or classical Newtonian physics. The former can describe the entirety of the measurement process, while the latter abstracts and simplifies the phenomena in a way that relates to the understandable physical world around us. The majority of MRI can be understood using the Newtonian perspective.

In the absence of an external magnetic field ($\vec{B_o}$), the spin (J) of an atomic nucleus will be in a random orientation relative to all other nuclei in the system. With $\vec{B_o}$ present, nuclei will tend to align with this external field to minimize its potential energy. The interaction of a nuclei's spin with $\vec{B_o}$ produces a torque causing the it to precess about the direction of $\vec{B_o}$. This produces a magnetic moment vector ($\vec{\mu_o}$) proportional to the spin. Quantum mechanically, $\vec{B_o}$ will cause two discrete spin-states corresponding to parallel ($+\frac{1}{2}$) and anti-parallel ($-\frac{1}{2}$) directions relative to $\vec{B_o}$ for dipolar nuclei, such as an ^{1}H (proton) atom. This splitting is known as the *Zeeman Effect* and derives the energy difference between these two states as

$$\Delta E = \hbar\gamma\vec{B_o} = \hbar\omega_o \tag{2.1.1}$$

where $\hbar$ is the reduced Plank's constant ($h/2\pi$) $[J \cdot s]$, γ is the gyromagnetic ratio $[\frac{Rad}{s \cdot T}]$, and ω_o is the precessional angular frequency $[\frac{Rad}{s}]$. The number of spins that are in the parallel or anti-parallel state is governed by the Boltzmann probability distribution that relates state population to the temperature of the system.

$$\frac{N_+}{N_-} = e^{-\hbar\gamma\vec{B_o}/kT} \tag{2.1.2}$$

N_- and N_+ are the number of spins in the anti-parallel and parallel states, respectively, k is the Boltzmann constant $[J \cdot K^{-1}]$, and T is the temperature $[K]$. With these relationships, we can determine that there is a net magnetization ($\vec{M_o}$) in the direction of $\vec{B_o}$

$$\vec{M_o} = \frac{\rho_o\gamma^2\hbar^2}{4kT}B_o \tag{2.1.3}$$

where ρ_o is the density of protons per unit volume, otherwise known as the spin density. The derivation of this net magnetization is left out for brevity. The perturbation of the net magnetization vector through an external electro-magnetic radiofrequency (RF) pulse is what generates the measured signal to be reconstructed.

2.2 Radiofrequency Pulses

RF pulses are generated using the RF body coil, which is built in to the MR system. The frequency generated by the coil coincides with the precessional (Larmor) frequency of the desired nuclei, thus satisfying a resonance condition. $\vec{M_o}$ is thus tipped *away* from its alignment with $\vec{B_o}$ (also known as the z-axis or longitudinal direction) by the second external magnetic field known as the $\vec{B_1^+}$ field. Typically, $\vec{M_o}$ is pulsed 90° onto the transverse plane where the relaxation (return to the z-axis) of $\vec{M_o}$ is described by the Bloch equation.

$$\frac{d\vec{M}}{dt} = \gamma \vec{M} \times \vec{B}_{ext} + \frac{1}{T_1}(M_o - M_z)\hat{z} - \frac{1}{T_2}\vec{M}_\perp \tag{2.2.1}$$

The solutions for the x, y, and z components of the magnetization are as follows,

$$M_x(t) = e^{-t/T_2}\left(M_x(0)\cos\omega_o t + M_y(0)\sin\omega_o t\right) \tag{2.2.2}$$

$$M_y(t) = e^{-t/T_2}\left(M_y(0)\cos\omega_o t - M_x(0)\sin\omega_o t\right) \tag{2.2.3}$$

$$M_z(t) = M_z(0)e^{-t/T_1} + M_o(1 - e^{t/T_1}) \tag{2.2.4}$$

where T_1 and T_2 are tissue-specific constants known as relaxation times that derive from the interaction of the spins with their surroundings. T_1, also known as the longitudinal relaxation time, describes the regrowth of $\vec{M_o}$ to the low energy state parallel to $\vec{B_o}$. This is governed by spin-lattice interactions. T_2 describes the loss of phase coherence between the various spins in the system during relaxation. The dephasing of spins represents a spin-spin decay of the transverse magnetization. As such, the relaxation terms describe the magnetization vector's return to the low energy equilibrium state. The longitudinal and transverse magnetization as a function of time are shown in figure 2.1 below.

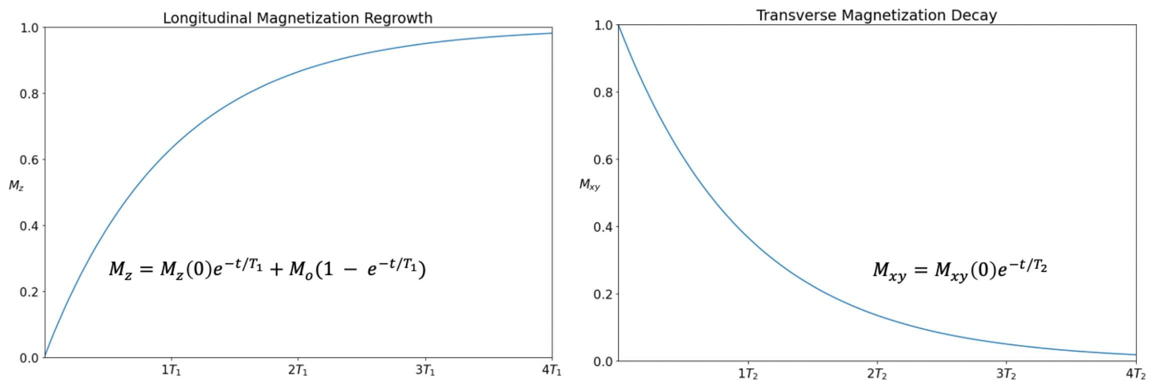

Figure. 2.1. Regrowth and decay of M_z and M_{xy} quantities, respectively. M_o is the initial magnetization before tipping. This is set to 1 for illustration purposes.

2.3 Image Reconstruction

Since the magnetization of aggregate nuclei spins has an associated EM-field, temporal or spatial changes to the $\vec{B}$ or $\vec{E}$ fields, respectively, causes an electromotive force (ϵ) in the receive coil. This phenomenon is governed by Faraday's Law of Induction (2.3.1) and will cause the receive coil to generate a current that *opposes* the change in magnetic flux according to Lenz's Law (2.3.2),

$$\nabla \times E = -\frac{\partial \vec{B}}{\partial t} \tag{2.3.1}$$

$$\epsilon = -N\frac{\vec{B}A}{\Delta t} \tag{2.3.2}$$

where A is the cross-sectional area of the coil. Unfortunately, measuring a single signal is not enough to generate an image.

Therefore, the reconstruction of a 3D volume measured in MR scanner requires the spatial localization of the set of discretized signals in all three dimensions. In order to do so, three orthogonal spatially varying magnetic field gradients (G_x, G_y, G_z) are applied to encode frequency and phase deviations in the excited nuclei.

$$G_x = \frac{\delta \vec{B}_z}{\delta x}, G_y = \frac{\delta \vec{B}_z}{\delta y}, G_z = \frac{\delta \vec{B}_z}{\delta z} \tag{2.3.3}$$

$$\omega_{larmor} = \gamma(\vec{B}_o + \Delta i G_i) \tag{2.3.4}$$

where i can represent either the x, y, or z directions for a discrete step Δi. The discretization of signals into voxels encoded by frequency and phase enables the use of reconstruction algorithms to produce an image. During acquisition, the digitized signals are measured in *k-space*, a formalism that describes the range of spatial frequencies encoded by the scanning procedure. The final image is constructed by taking the inverse 2D or 3D Fourier-transform of k-space.

Clinically, there are always trade-offs when tuning the imaging parameters for a particular scan. The relationships between the voxel size, echo time (TE), repetition time (TR), flip angle, field-of-view (FOV), signal-to-noise ratio (SNR), contrast-to-noise ratio (CNR), and receiver bandwidth (to name a few) ultimately governs the properties of the final image. Practically, noise corrupting the final image can never be fully removed, but its minimization is crucial for high SNR and CNR. Furthermore, minimizing artifacts such as ghosting (aliasing) by sampling much higher than Nyquist during acquisition is also of high priority to avoid wrap-around effects due to the improper spatial encoding of voxel signals.

2.4 Diffusion Tensor Imaging

Diffusion MRI (dMRI) is an MR imaging technique used to measure and characterize the diffusion of water within tissues. The diffusion of water can exhibit either an isotropic or restricted behaviour determined by the orientations of cell membranes and other cellular structures. The diffusion profiles of water in each of these cases is

Gaussian and non-Gaussian, respectively.

In diffusion-weighted imaging (DWI), the measured loss in signal intensity due to diffusion is controlled through bi-polar dephasing and rephasing gradients, as shown in figure 2.2.

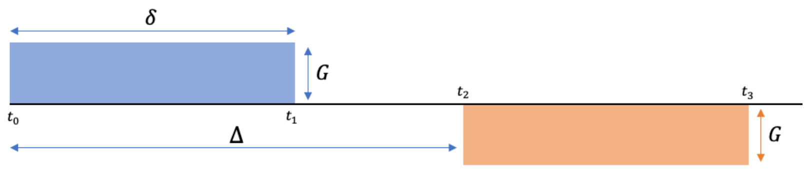

Figure. 2.2. DWI encoding gradients. Dephasing and rephasing gradients are blue and orange, respectively.

The parameters that govern the signal loss are the duration of the gradient (δ), the time between the start of each gradient (Δ), and the amplitude of the gradients (G). By knowing both the phase (2.4.1) and proton (2.4.2) distributions, the resulting signal (2.4.3) can be determined.

$$\phi(x) = e^{i\gamma\delta Gx} \tag{2.4.1}$$

$$p(x,t) = \frac{e^{-x^2/4Dt}}{\sqrt{4\pi Dt}} \tag{2.4.2}$$

$$\begin{aligned} S &= \int\int p(x,t)\phi(t)dxdt \\ &= S_o e^{-\gamma^2 G^2 \delta^2 (\Delta - \delta/3)D} \\ &= S_o e^{-bD} \\ ln(S) &= ln(S_o) - bD \end{aligned} \tag{2.4.3}$$

Where the intrinsic diffusion constant (D) is determined through the fit. Therefore, the amount of diffusion weighting is controlled through the b-value. A minimum of two b-values must be used to determine the slope, D of the curve, where b-values can range from $0s/mm^2$ to upwards of $3000s/mm^2$. The initial signal intensity S_o is the signal with a b-value equal to $0s/mm^2$.

In diffusion tensor imaging (DTI), water diffusion is encoded along a minimum of 6 spatial directions and is highly sensitive to minor changes in microsctructural WM integrity. The shape of water diffusion can be described as a rank-2 tensor (3x3 positive-definite matrix, 2.4.4) with three mutually orthogonal eigenvectors with corresponding eigenvalues shown in figure 2.3 .

$$\bar{\bar{D}} = \begin{bmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{bmatrix} \tag{2.4.4}$$

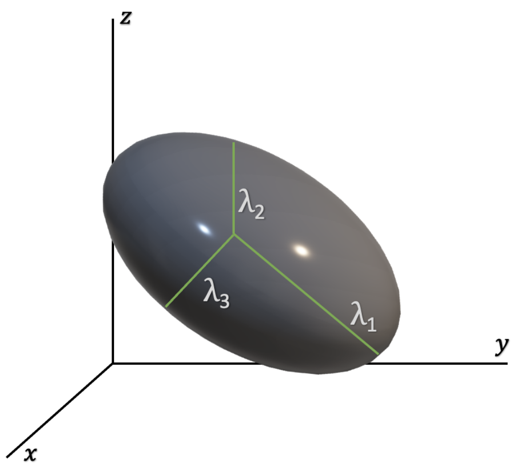

Figure. 2.3. Anisotropic diffusion profile with the three eigenvectors coloured in green and corresponding eigenvalues, λ_1, λ_2, λ_3.

The major eigenvector signifies the principle direction of diffusion and the measured macroscopic diffusion anisotropy is due to microstructural tissue heterogeneity. Combinations of these eigenvalues derive the four spatial mappings of diffusion: axial diffusivity (AD), radial diffusivity (RD), mean diffusivity (MD), and fractional anisotropy (FA),

$$AD = \lambda_1 \tag{2.4.5}$$

$$RD = \frac{\lambda_2 + \lambda_3}{2} \tag{2.4.6}$$

$$MD = \frac{\lambda_1 + \lambda_2 + \lambda_3}{3} \tag{2.4.7}$$

$$FA = \sqrt{\frac{1}{2}} \frac{\sqrt{(\lambda_1 - \lambda_2)^2 + (\lambda_1 - \lambda_3)^2 + (\lambda_2 - \lambda_3)^2}}{\sqrt{(\lambda_1^2 + \lambda_2^2 + \lambda_3^2)}} \tag{2.4.8}$$

For patients who have been subjected to a form of brain injury, there tends to be a measurable change in the individual's WM diffusion. This damage can occur in brain myelin, the primary structure of WM tracts. Therefore, the characteristic anisotropic diffusion of fluid along a WM tract can suddenly become more isotropic due to the microstructural damage. Damaged regions correlate with decreased FA and increased RD (relative to AD), and increased MD [17]. FA is bounded between 0-1 due to the $\sqrt{(\lambda_1^2 + \lambda_2^2 + \lambda_3^2)}$ normalization factor, where realistic diffusion values are between 0.30 to 0.70. FA is also the most characteristic parameter for quantifying diffusion, thus making it the most common mapping for statistical analyses.

Like all MR imaging modalities, dMRI is susceptible to partial volume effects, whereby the diffusion model assumes that there is only a single tissue type in a voxel, where in reality, voxels are likely to hold a mixture of tissues. The proportionality of each tissue within the voxel alters the final encoded signal and results in systematic error. Therefore, there is reduced sensitivity within the measurement that cannot be easily accounted for in a traditional 6-direction scan. This leads researchers to apply as many as 180 scanning directions to improve diffusion modelling results.

Detecting and quantifying microstructural damage has the potential to be an incredibly useful clinical tool. However, these changes cannot be visually identified by a human interpreter alone. Therefore, there is a necessity to implement sophisticated statistical tools such as Artificial Intelligence to uncover this unseen information.

Chapter 3

Artificial Intelligence

3.1 Machine Learning to Deep Learning

In the field of machine learning, mathematical algorithms are developed to make data-driven predictions. Algorithms are given a set of data and are optimized to learn the important features that characterize a particular dataset through a defined loss function. The generalizability of a machine learning model depends not only on the amount of data the algorithm has been trained on, but the diversity of samples within the dataset. A dataset is randomly decomposed into training, validation, and testing sets, where the performance of the model is evaluated on the unseen data within the test set and reported through a variety of metrics.

Machine learning models can be broken down into three subsets known as *Supervised*, *Unsupervised*, and *Semi-Supervised* learning. Most of the established machine learning frameworks today are supervised, meaning that models are trained with a dataset where each data point has an associated label (binary, multi-label, and/or multi-class). Supervised techniques are largely used for tasks such as classification and segmentation, where the model is trained to classify the data into groups or segment regions of interest within an image, for example. Popular algorithms include linear regression, neural networks, and random forests. Unsupervised learning approaches are situations where the model is trained with unlabelled data, forcing the model to learn associations within the data that it deems suitable based on the loss function. In the absence of labels, unsupervised models are typically employed for clustering, dimentionality reduction, and anomaly detection [18]. Examples include k-nearest-neighbours, Gaussian mixture models, and variational autoencoders. Finally, there are situations where the class labels are incomplete. While wanting to use this information, techniques of semi-supervised learning have been developed to make use of the available labels to increase potential task accuracy. Compared to traditional supervised and unsupervised frameworks, semi-supervised methods are much less common in practice.

Neural networks, the stepping stone to deep learning, are frameworks that were developed in this pursuit for scalable models. Composed of layers of neurons that are connected in ways to learn abstract representations of the input data, the empirically shown capabilities of neural networks to learn important features and perform exceedingly well in its prediction task has been the main reason for its growth in the machine learning community. Figure 3.1 illustrates the relationship between the inputs and output of a single neuron in a neural network.

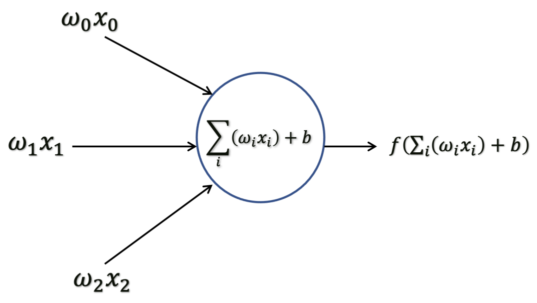

Figure. 3.1. Representation of a computational neuron used in deep learning.

Three weighted inputs $\omega_0 x_0$, $\omega_1 x_1$, and $\omega_2 x_2$ are fed into the neuron. Each x is the input to the current neuron from a neuron in the previous layer and each weight

ω associated to each input will be updated through an optimization training process known as stochastic gradient decent [19]. The inputs are summed together along with an optional bias term b. The summation is transformed using a non-linear activation function, f, traditionally a sigmoid, rectified linear-unit (ReLU), or hyperbolic tangent (tanh) function [19]. Introducing such non-linearities allow for the network to increase its ability to learn complex functional mappings within the data. If, for example, this non-linear step were to be omitted, multi-layered neural networks could be mathematically reduced to the simplest case of a single-layer perceptron[1], thus dramatically reducing the ability of the model to generalize over the non-linear data distribution [19].

The way in which neurons are connected between/within layers and how the layers are structured ultimately determines network properties such as the number of parameters and the types of tasks optimal for the network. As the number of network layers increase, we arrive at the concept of *deep learning*: a neural network architecture with an arbitrary number of hidden layers[2] with the purpose of creating a model with state-of-the-art performance on complicated tasks. Deep networks are empirically shown to have improved predictive performance compared to traditional statistical learning approaches because the self-optimization process removes the need for explicit feature engineering [19]. These hand-made features limit the flexibility of a particular model to concentrate onto other minute features that could potentially have more discriminative power within the dataset.

In the field of computer vision where imagery comes with significant variation, it becomes difficult to pinpoint and mathematically define a discrete number of features that will generalize to the entire distribution. Features could include estimates of

[1]Perceptron: An algorithm used for binary classification.

[2]Strictly speaking, the output of the input layer is not directly observed, but this is not considered a hidden layer

the image's fractal dimension, histogram distribution, or the size of circular blobs, for example, but rarely does feature engineering result in predictive performances comparable to deep learning on the same multi-class dataset. Most notably, this was demonstrated during the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012, where a deep convolutional neural network architecture halved the second place error rate and lead to the "deep learning revolution" we have seen today [20].

3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are the backbone of all modern computer vision systems. Spanning areas of study from autonomous driving to medical imaging, the millions of convolutional kernels that are generated and optimized through the training procedure is what makes this type of framework so powerful. The in the following discussion, we outline the necessary components for CNNs and the various ways they are constructed together to produce classification and segmentation models.

Before the advent of CNNs, if a typical feed-forward neural network architecture was applied for the task of *image* classification, each pixel would represent a single input to only *one* neuron in the input layer. For a 60x60x3 (60px wide, 60px tall, 3 colour channels) image, this would lead to an input layer with 10800 neurons, where the network has an N number of hidden layers, leading to $10800N$ parameters (excluding bias terms). With even larger, more respectable images (256x256x3), the number of parameters quickly blows up and would lead to *huge* computation times and overfitting [19]. The way CNNs have circumvented this problem is by defining a set of layers that "compress" a local receptive field within the image into a single number representing the bulk information within the field. CNNs are built upon these most basic layers and are outlined in the following sections.

3.2.1 Convolutional Layers

The convolutional layer within the network is what does most of the heavy computational work. These layers are composed of a **set** of trainable filters (or kernels) that operate on only a small portion of the input image. For example, a single filter could be of size $6 \times 6 \times 3$ acting on an image of size $256 \times 256 \times 3$. The *depth* of the filter will always match the channel depth of the input image. The filter will *convolve* along the height and width of the image, computing the dot product of the filter values and the image input values within the filter's *receptive field*, F. This convolutional procedure thus produces a single 2D activation map for a single filter. With a set of K filters, the convolutional layer thus creates a stack of K activation maps. This processes is shown in figure 3.2.

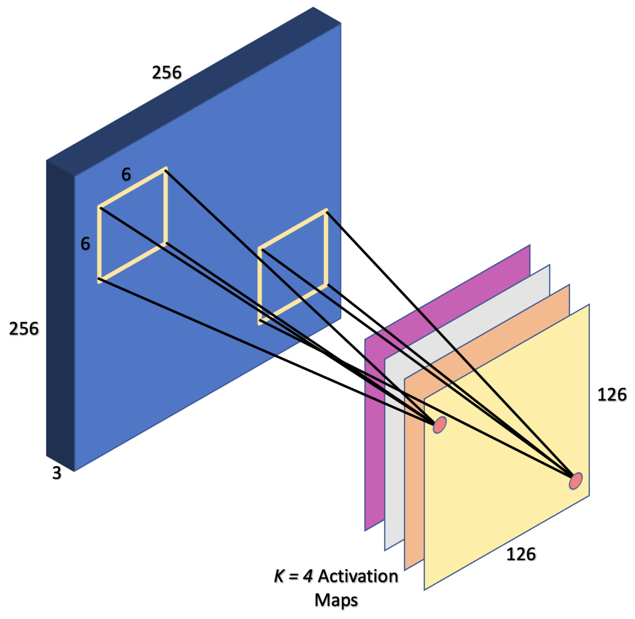

Figure. 3.2. A set of convolutional filters applied to an input image of size 256x256x3 with receptive fields of 6x6x3, generating 4 activation maps.

In the situation illustrated above, there are four independently trained convolutional filters that are applied to the input image. The receptive field of each filter is 6x6x3 and computes a single number on the corresponding output activation map for a single filter location. Ideally, each activation map is an $n \times n$ square matrix but this does not always have to be the case. Square matrices are easier to work with due to their shape, thus making it desirable to ensure equal dimensions along the height and width of the activation map. Formally, a pixel value, y_{ij}, is produced from a convolutional operator through the following,

$$y_{ij} = \sum_{k=0}^{K} \left(\sum_{i=0,j=0} (x_{ij}\omega_{ij}^{k}) \right) + b \tag{3.2.1}$$

where i, j are row and column indices within the receptive field, x_{ij} is the pixel value, ω_{ij}^{k} is the filter weight for at index i, j for the k^{th} filter, and b is the optional bias term.

To ensure the convolutional operation can occur without error, other hyperparameters known as the convolutional stride (S) and zero-padding (P) must be used. S is defined as the number of pixels the filter moves after each computation, where $S = S_{height} = S_{width}$. P adds a border of zeros to the input image to ensure square output activation. To calculate the activation output size, the following relationships are used

$$W_o = \frac{W_i - F + 2P}{S} + 1 \tag{3.2.2}$$

$$H_o = \frac{H_i - F + 2P}{S} + 1 \tag{3.2.3}$$

$$D_o = K \tag{3.2.4}$$

where W_o, W_i, H_o, H_i, D_o are the output and input dimensions of the width, height, and depth, respectively [21]. It should be explicitly noted that this operation *reduces* the size of the input image and thus, cannot be applied indefinitely. Interestingly, once the training procedure is completed, filters earlier in the model have been shown to pinpoint “lower-level features”, such as edges, blobs, and contrast, where deeper filters continually become more abstract and look for “higher-level features”, such as facial features or shapes. Due to the convolutional action the filters undergo, they possess a property known as *translation invariance*. As such, if a filter has been optimized to detect a specific orientation of an edge, the filter will have a maximum output when it encounters that edge regardless of the feature’s location within the image.

3.2.2 Transposed Convolution

An image can be up-sampled using a method known as *transposed convolution (TC)*. Unlike a traditional interpolation method, TC has tunable parameters that govern the up-sampling. Mathematically, TC is not the same as deconvolution, but the results are identical. To understand TC, consider how a convolutional operator can be performed using matrix multiplication. The input image used during the convolutional process is turned into an $(n \times n) \times 1$ column vector, where the filter is subsequently transformed into an $(n \times n^2)$ convolutional matrix with added zeros to account for the receptive field. n represents the number of rows (or columns) for a square input image.

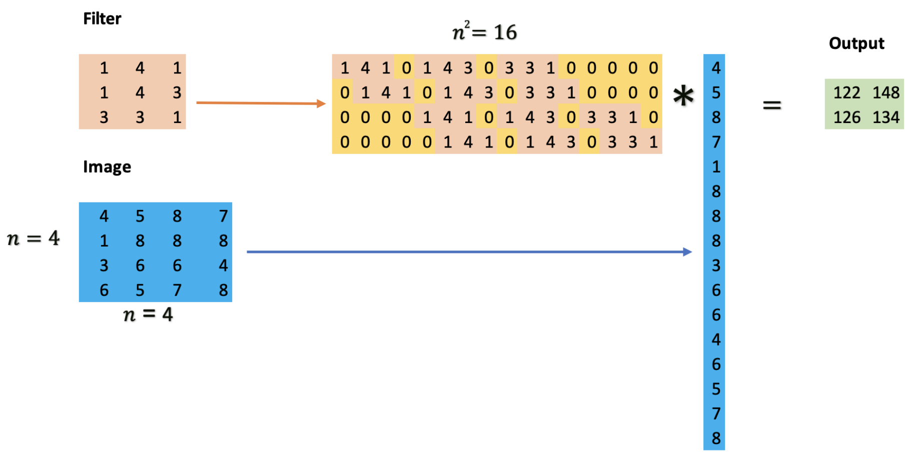

Figure. 3.3. Convolution operator using matrix mathematics. Filter is zero-padded to account for striding during normal convolution.

As seen in figure 3.3, the input image has been reduced from a size of (4x4) to (2x2).

In the reverse case where we desire a (4x4) output from a lower resolution (2x2) input, the *transpose* of the proper convolutional matrix will result in the desired output. Using the transpose of the matrix shown above, we show the following transpose convolution in figure 3.4,

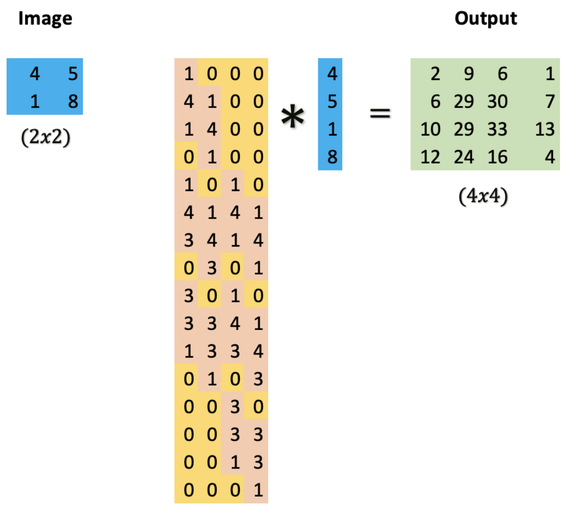

Figure. 3.4. Transpose of a convolutional matrix is able to produce the same effect as deconvolution by up-sampling a (2x2) image into a (4x4) output.

3.2.3 Pooling

Pooling layers are typically placed periodically after convolutional layers with the intention of further reducing the spatial representation of the image, thus reducing the number of overall parameters controlling overfitting. Similar to a convolutional filter, a pooling operator has a square receptive field on the order of single pixels and strides along the image. It applies either a MAX, MIN, or AVERAGE operation to the values in the receptive field and places this into the correct pixel location within the compressed image.

Pooling layers were a previously successful operation that improved CNN performance but has been slowly phased out since many researchers dislike the explicit removal of information. Newer models that include attention and transformers have continued to improve in the absence of pooling layers [22, 23].

3.2.4 Full-Connected Layers

Once the image has been reduced to an abstract representation after many convolutional layers, the representation is known to be in a "latent space". The image is typically reformatted into a $(N\text{x}N\text{x}K)\text{x}1$ column vector and fed through a set of fully-connected layers, as shown in figure 3.5.

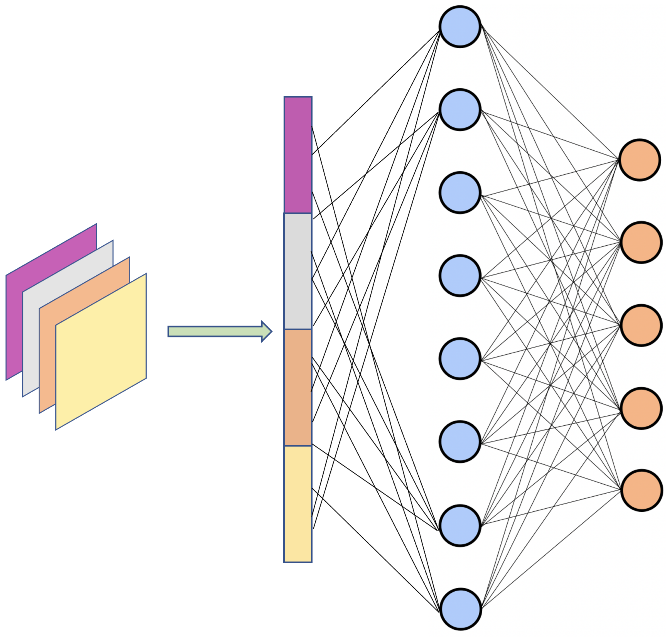

Figure. 3.5. Fully-connected layer at the end of a convolutional neural network framework. All final activation maps are re-represented as a single 1D vector and is the input to the fully-connected layer. Not all arrows are drawn to reduce image clutter. In this case, there is only one hidden layer prior to the final classification layer (5-class output).

At the end of the fully-connected layer, the output can be represented as a column vector with a length equal to the number of classes it has been trained to classify. The vector is typically passed through a sigmoid activation function to compress each probability between 0 and 1.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.2.5}$$

3.3 Transfer Learning

It is important to recognize all CNN architectures have the commonality of reducing the complexity of the input image and learning convolutional filters that end up representing characteristic features that represent the dataset. Furthermore, the filter weights are tuned specifically for the task it has been trained to do. But, *transfer learning* approaches may be implemented to borrow weights from previously trained models to accelerate the training process for a new task on a different dataset. As previously eluded, the earlier filters tended to learn features that may be ubiquitous to all imagery. Therefore, transfer learning techniques have shown to provide significant improvements to model training time, depending on the task [24]. With that said, there is an ongoing debate as to whether or not ImageNet-trained weights are a valuable tool for transfer learning in medical imaging domains. This is likely due to the domain mismatch between the naturally occurring images seen in ImageNet compared to the vastly different images that can be obtained through medical imaging modalities. New work has proposed that ImageNet transfer learning can indeed improve DL models made for biomedical imaging tasks, but the performance boosts depend on both the *scale* of the model's capacity (very large networks) and the size of the pre-training dataset [25]. Typically, early model weights are initialized using transfer learning, later weights are randomly initialized, and the whole model is fine-tuned to acquire optimal task performance.

3.4 Make Deep Learning Work!

Along side the various network layers that compose CNNs, there are a set of considerations and tools that are needed in order to actually train a model. These considerations are needed *regardless* of the model's architecture or the task type.

3.4.1 Loss Functions

The loss function, $E(x, y)$, determines the discrepancy between the prediction of a model and the desired output. Based on the value computed by the loss function, the network weights are updated through a processes known as *backpropagation* and *stochastic gradient descent*. In the simplest form, a learning problem adjusts the weights within the model to minimize $E(x, y)$.

Popular loss functions include the **Mean-Squared Error (MSE)**

$$E(x, y) = \frac{1}{n}\sum_i (x_i - y_i)^2 \tag{3.4.1}$$

where x_i is either a single or a set of predictions, y_i are the corresponding class labels in a supervised learning problem.

Binary Cross-Entropy measures the performance of a classification model when outputs are between 0 and 1.

$$E(x, y) = -\sum_i \left(y_i \log(\frac{1}{1 - e^{-x_i}}) + (1 - y_i)\log(1 - \frac{1}{1 - e^{-x_i}}) \right) \tag{3.4.2}$$

The loss function can be adapted to account for class imbalanced datasets by weighting the observation by its proportion within the dataset.

$$E(x, y) = -\sum_i \left(\omega_p y_i \log(\frac{1}{1 - e^{-x_i}}) + \omega_n(1 - y_i)\log(1 - \frac{1}{1 - e^{-x_i}}) \right) \tag{3.4.3}$$

Where ω_p and ω_n are the positive and negative weighted proportions, respectively. For a 2D or 3D image, this would correspond to weightings of $\omega_p = \frac{NumZeros}{Resolution}$ and $\omega_n = 1 - \omega_p$.

Dice Loss optimizes the weights based on the overlap between two classes A and

B.

$$E(A,B) = 2 * \frac{|A \cap B|}{|A| + |B|} \tag{3.4.4}$$

When not using it as an explicit loss function, $E(A,B)$ is known as the Dice Similarity Coefficient, or Dice Score, (DSC). An evaluation metric used to measure the performance of a segmentation model, it can also be calculated through the true positive (TP), false positive (FP), and false negative (FN) rates reported through a confusion matrix,

$$DSC = \frac{2TP}{2TP + FP + FN} \tag{3.4.5}$$

Mathematically, this formulation is identical to the well-known F1-score in binary classification problems.

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2}{\frac{TP+FP}{TP} + \frac{TP+FN}{TP}} = \frac{2TP}{2TP + FP + FN} \tag{3.4.6}$$

3.4.2 Gradient Descent

After determining the loss associated to the model's prediction, there needs to be a way to update the weights of the network to *minimize* the loss of future predictions. This is done through a method known as backpropogation. Loss function minimization must also work in conjunction with the network's ability to *generalize* to new data. The model's performance on the training and validation sets are ultimately unimportant and the model must not overfit to the training data. An overfit model would show low loss during training but poor performance during test-time.

Propagating the loss through the network to update the model weights and move through the loss landscape is most successfully done through gradient-based methods. Backpropogation is typically very slow with large models that tend to have highly non-convex, high dimensional loss spaces with many local maxima/minima and/or flat regions [26]. As such, there is no guarantee that the model will converge to the global minimum, that this convergence would be quick, or if convergence would even occur. However, it has been shown empirically that gradient descent methods can often find good solutions within the loss landscape when using the gradients calculated through backpropagation.

Stochastic gradient descent refers to a descent process that uses the loss of a single random observation, selected with replacement, to update the network parameters, θ.

$$\theta(t+1) = \theta(t) - \eta \frac{\partial E^t}{\partial \theta} \tag{3.4.7}$$

where η is the learning rate for gradient descent. This rate is typically dynamic and changes depending on how the loss stagnates during training. Large η promote faster movements and can improve convergence speed early in the training process, but can cause overshooting and prevent convergence as training progresses. Similarly, small η can prevent convergence within a reasonable computation time. Typically, η starts around 10^{-4} and is reduced by an order of magnitude as the loss stagnates.

Batch stochastic gradient descent is the same update procedure but instead of a single observation, the *entire* dataset is used to acquire a direction of average loss. These steps will often be much larger than that of stochastic gradient descent, which can reduce computation times, but can lead to worse results [26].

$$\theta(t+1) = \theta(t) - \eta \frac{\partial E^s}{\partial \theta} \tag{3.4.8}$$

Where E^s is the loss on the entire set.

Mini-batch stochastic gradient descent is a hybrid method between the two discussed previously. The dataset for DL problems is often very large and it is impractical to determine the loss of the entire training set in a single computation. Furthermore, computing the loss on single observations is often prohibitively slow. Therefore, mini-batch stochastic gradient descent takes *mini-batches* of the training set, computes the loss, and updates the weights. The amount of data that is placed in the mini-batch is limited by the memory capabilities of the computer facilitating model training and is typically equal to a power of 2^n.

3.4.3 Regularization

DL models have the capacity to "memorize" (overfit) data if improperly trained. Regularization methods have been developed to prevent the model from overfitting to the training dataset. Popular regularization methods are:

L1 and L2 Regularization: Includes a second factor to the loss function.

$$L_1 = E(x, y) + \lambda \sum_i |\theta_i| \tag{3.4.9}$$

$$L_2 = E(x, y) + \lambda \sum_i \theta_i^2 \tag{3.4.10}$$

where λ is the regularization parameter. L_1 and L_2 regularization methods can be applied in tandem with the weighted loss functions described previously.

Data Shuffling and Augmentation: Networks learn the quickest when exposed to an unexpected sample. That is, each subsequent observation should be of a different class, if possible [26]. Therefore, to maximize the chance of this occurring, data shuffling is highly advisable. Furthermore, each observation can be subjected to a set of random affine transformations such as rotations, flipping, and scaling to promote diversity within the dataset.

Dropout: Dropout refers to a training procedure that randomly suppresses a subset of nodes and their corresponding connections within the network to reduce the chance of co-adaptation between layers [27]. Effectively, dropout forces the network to learn more relevant features, reduces overfitting, and thus improves generalization ability. Most implementations of DL use dropout for performative improvements. The figure below illustrates dropout in a simple network.

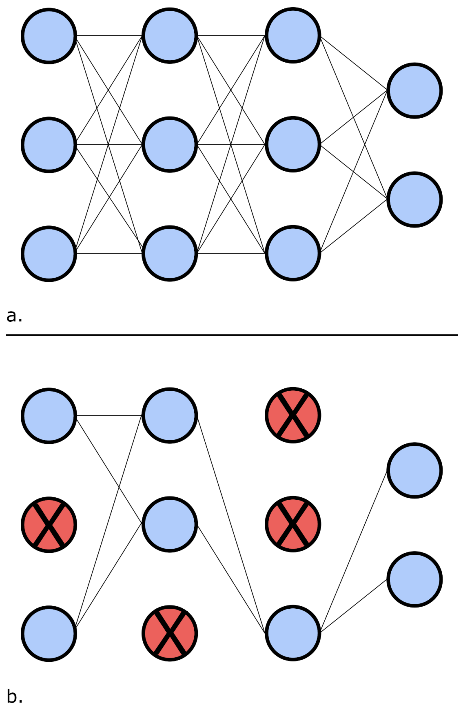

Figure. 3.6. a: Simple fully-connected neural network. **b:** Neural network with dropout. All nodes and corresponding neural connections have a dropout probability of $0 < p < 1$

3.4.4 Normalization

Normalization is a crucial pre-processing step that has shown to improve the stability of neural networks, promoting faster convergence [26]. With MR images, there is no universal intensity scale that is invariant of the scanner or the image (unlike CT). Therefore, normalization is *mandatory* if one is looking to have directly comparable histogram distributions.

Different from regularization, normalization scales the data so each observation has comparable statistics. In a computer vision problem, the set of data can be globally normalized or have an intra-image normalization using a variety of methods, but simple options include:

Standardization (Z-score Normalization):

$$y_i = \frac{x_i - \mu}{\sigma} \tag{3.4.11}$$

where μ and σ are the mean and standard deviation of the image x, respectively, resulting in pixel value y_i.

Min-Max Normalization:

$$y_i = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{3.4.12}$$

where every observation x_i is shifted by the minimum value x_{min} and normalized by the range of observation values $x_{max} - x_{min}$. This will fix the range of observations in a particular feature between 0 and 1 exactly, while standardization has no guarantee that such a condition will occur.

3.5 CNN Architectures

Applications of DL for computer vision problems have been widely successful with the use of CNNs. While the property of translation invariance on convolutional kernels is advantageous for these tasks, the ways in which various computation layers are connected also significantly influence the model's ability to learn complex representations. As such, researchers have developed various models which has lead to a huge variety of CNN-based architectures for visual learning tasks. Models such as Vgg [28], ResNet [29], and DenseNet [30] are popular options for classification and have easy-to-use implementations in both PyTorch and Tensorflow. Reconstruction approaches for segmentation include unsupervised autoencoders [31], fully-convolutional networks (FCNs) [32], and U-Nets [33]. These are employed in situations where feature detection and localization are both relevant. Here, we discuss the differences between all of these networks, leading to a discussion on the 2D U-Net.

3.5.1 Vgg

Introduced by Simonyan et al., very deep convolutional networks (Vgg) demonstrated the empirical benefits of using a larger quantity of smaller convolutional filters (3x3) to improve computational performance and learning ability for large-scale recognition tasks [28]. They further showed that adding many more deep layers was only feasible if the receptive fields of the convolutional kernels were small. The number of layers within a model architecture is typically denoted by the number following the model name, such as Vgg16 or Vgg19, for example. Max-pooling layers placed after convolutional layers are interspersed within the model architecture, eventually leading to a fully-connected (FC) sigmoid layer for multi-class classification.

3.5.2 ResNet

Introduced by He et al., the residual network (ResNet) featured the addition of skip-connections between layers to improve learning performance for deeper models [29]. The skip connections explicitly reformulate traditional learning layers as learning residual functions with reference to the layer input(s). The non-linear mapping at the output of each residual block is defined through equation 3.5.1.

$$y = \mathcal{H}(x) = \mathcal{F}(x) + x \tag{3.5.1}$$

where $\mathcal{F}(x)$ is the residual function and (x,y) are the input and output, respectively. Training the block to learn a residual function provides extra flexibility when the desired functional relationship needs to approximate an identity function [29].

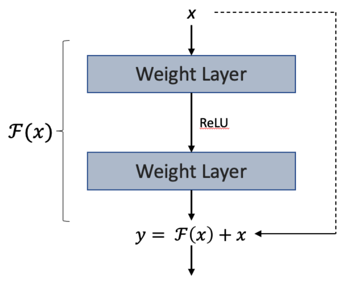

Figure. 3.7. Residual learning block [29].

The skip connections were empirically shown to improve the learning capacity of deep networks and provide a solution to previously impossible optimization on such large networks. ResNet52, 101, and 152 are popular variants of the architecture. ResNet architectures are built with classification in mind and terminate with a FC sigmoid layer.

3.5.3 DenseNet

Introduced by Huang et al., DenseNets were developed using the methodology of residual networks in mind [30]. Rather than summing the input and output of a single functional block repeatedly over many blocks, DenseNets both *concatenate* the input to the output of a block and propagate the output through skip-connections to *every subsequent layer*. The architecture was designed to remove the vanishing-gradient dilemma that occurs with large networks. Furthermore, the architecture shows that it requires *fewer* parameters to achieve impressive task performance compared to other larger networks. A schematic of a DenseNet is shown in figure 3.8.

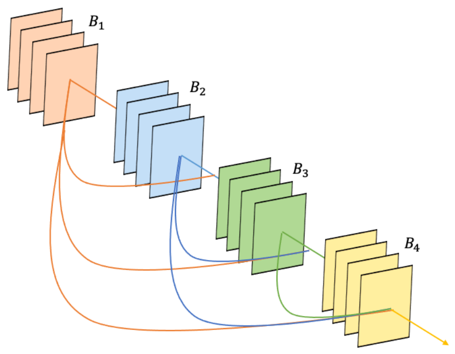

Figure. 3.8. DenseNet Architecture [30]. B_n represents a block in the network.

3.5.4 Unsupervised Autoencoders

Introduced by Ballard et al. in 1987, autoencoders are unsupervised models that were designed for the purposes of image reconstruction by compressing the input into a latent representation and reconstructing the image using the latent representations of the input [31]. Through a desired loss function, the difference between the input image and output reconstruction is sought to be minimized. The latent space can be interpreted as a dimensionality reduction scheme similar to principal component analysis (PCA) but in a non-linear regime. The initial autoencoder networks were simple fully-connected networks with 1-2 hidden layers, similar to the network illustrated in figure 3.6. Autoencoders have provided the foundation of all modern DL segmentation models.

3.5.5 Fully-Convolutional Networks

Introduced by Long et al., Fully-Convolutional Networks (FCNs) transform traditional fully-connected layers in classification networks to convolutional layers for semantic segmentation [32]. The model is able to output a classification heatmap (activation map) on an image by implementing a spatial loss between the output and corresponding label map. At its incecption, FCNs were the first end-to-end model for pixelwise prediction.

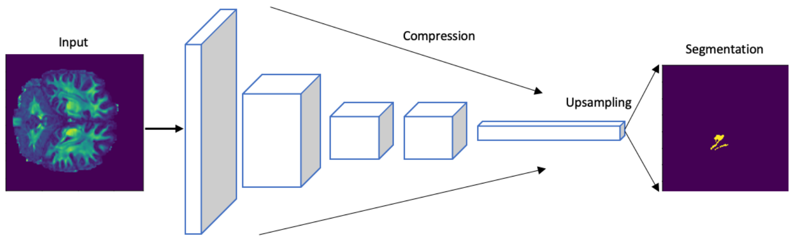

Figure. 3.9. Fully-Convolutional Network for semantic segmentation. Input image is reduced to a smaller spatial representation through convolutional operations and up-sampled to the output activation map [32].

FCNs utilize batch-normalization, dropout, and pooling layers for computational considerations and overfitting. The up-sampling step is done through a bi-linear interpolation and thus, only up-samples *once*. Therefore, the process of up-sampling cannot be optimized explicitly through model training.

3.5.6 U-Net

Introduced by Ronneberger et al., the U-Net has become the most popular option for semantic segmentation tasks, particularly in biomedical imaging [33]. The network is composed of three main branches: the encoder, latent space, and decoder. The formation of the three of these building blocks illustrate a U-like structure, as seen in figure 3.10.

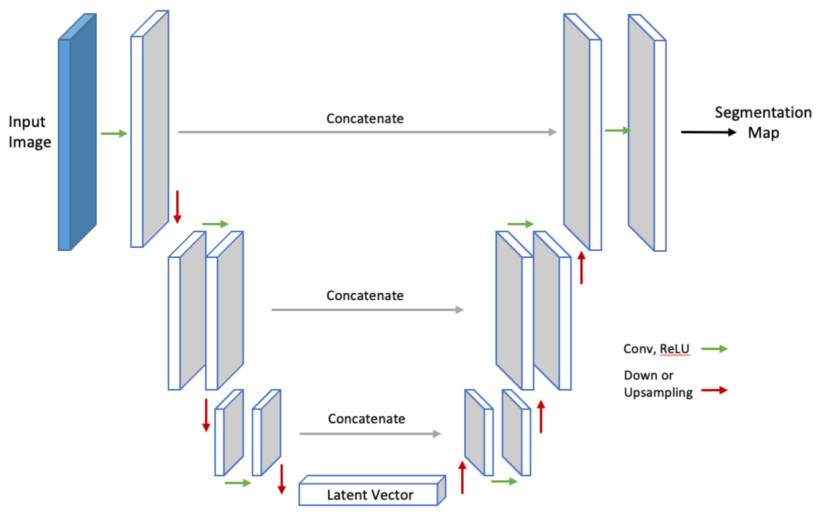

Figure. 3.10. Illustration of a simple U-Net architecture [33]. Each block is a set of activation maps from filters trained through backprop.

The U-Net architecture heavily borrows the properties of the unsupervised autoencoder by taking its input and encoding it into a latent feature representation. The decoding pathway applies a set of successive trainable transpose convolutional operations (otherwise known as deconvolution) to reconstruct a pixel-wise activation map. In binary segmentation, a test-time threshold is applied to the activation map to obtain an estimate of the test-time performance. The maps that can be obtained from a U-Net architecture have been shown to be superior to FCNs due to the longer reconstruction decoding pathway.

The architecture takes advantage of dropout and max-pooling layers to optimize performance. Max-pooling layers are replaced with transpose convolutional operations during decoding. For segmentation of images with arbitrary dimensions, the architecture employs an overlap-tile strategy that enables the efficient prediction of pixels at the border region [33]. Further, the model has an additional concatenation step that bridges corresponding encoding/decoding blocks. This reinforces the model to take advantage of nuanced information learned earlier in the pipeline that could have been lost during compression. The large number of decoding feature maps, along with the representative encoding feature map, enables localization of high resolution features for improved segmentation performance.

In its initial implementation, the architecture was optimized for 2D spatial learning (hxwxc), where c are the number of colour channels. Therefore, models of this type are typically denoted as a **2D U-Net**. The architecture has since been extended and applied for 3D learning to take advantage of the extra depth component that is present in many biomedical imaging datasets, thus making the **3D U-Net** [34]. This architecture has the same base layer layout as a 2D U-Net with 3D convolutional operators. With the additional depth information, 3D U-Nets can be seen to have either comparable or greater volumetric segmentation performance [34, 35, 36, 37, 38]. Within medical imaging, scanning procedures that are able to acquire sequential slices of patient anatomy, such as MRI and CT, are posed well to take advantage of the depth dimension during 3D processing. The performance improvements unavoidably come with increased computational demands due to the dramatic increase in the total number of model parameters.

The encoder of the 2D/3D U-Net is often referred to as the *backbone*. Various encoding backbones can be chosen depending on the target task. The optimal encoding backbone must be empirically determined by observing the test-time performance.

Structurally, DL models are build upon thousands to millions of parameters, which is why they are often considered to be *black-boxes*. In the following sections, the discussion surrounding *Explainable Artificial Intelligence* is covered with a particular focus on its application in clinical settings.

Chapter 4

Explainable Artificial Intelligence

[1] As machine learning (ML) permeates through many facets of society, it is not surprising that there has been interest from users and stakeholders to make these algorithms more understandable [39]. While there have been many attempts to do so, there is no consistent distinction between interpretability and explainability, two key themes for understandable ML. Depending on the literature, these terms are either used synonymously or with explicit distinction [39, 40, 41, 42, 43, 44]. An explainable model would be one that describes, in detail, how it arrived at a prediction, outlining the relationship between the feature space and data domain. With this perspective, no black-box model can be *explainable*, and we should be in the pursuit to implement either:

1. inherently interpretable decision platforms, or
2. to apply post-hoc methods to promote case-based reasoning within complicated domains [43].

The former points to implementations of simpler, potentially non-ML models, where the latter acknowledges the predictive power of ML/DL and emphasizes the need to implement tools for interpretable DL. It goes without saying that computer

[1]Chapter 4 includes many components that are in the revision process for publication.

vision applications have benefited greatly from neural networks, but a universal definition of "interpretability" in the computer vision domain does not exist [43]. In the following sections, we focus on explaining post-hoc methods that provide interpretability to black-box DL models, leading to a metric known as predictive entropy (PE) that is heavily used in chapter 6.

4.1 Post-Hoc Interpretability

As Lipton [45] states in *The Mythos of Model Interpretability*, the preceding literature addressing interpretability typically have diverse and conflicting objectives. While there is a general notion that model "transparency" confers interpretability, post-hoc methods have been derived as to align better with human expectations [45]. Post-hoc methods for interpretability do not explicitly state how the model works, but rather provide the end user with additional information that elucidates reasoning. One major advantage of post-hoc methods is that models can be interpreted after the prediction without sacrificing predictive performance. Post-hoc methods include text explanations to explain a model's state, explanations through example, and the popular method of local explanations through gradient-based saliency/attention maps [45]. An additional metric that could further assist model interpretability is uncertainty. Providing users with uncertainty estimates and corresponding visualizations to quantify the degree of belief the model has in its prediction could be a useful addition to current post-hoc interpretability tools. For the purposes of this thesis, we employ predictive uncertainty in efforts to increase user confidence during test-time and improve trust in the ML model.

4.2 Trust and Interpretability

While acquiring interpretable predictions from ML models is a desired attribute for their successful deployment in high-risk applications (medicine, finance, criminal justice), *trust* also plays a critical role. As trust is a subjective concept, its definition is malleable depending on the context of the situation. Some argue that model interpretability is a precursor to trust [46]. The cultivation of trust in ML models could benefit from an externalist epistemological perspective, where trust is rationally justified through proven, repetitive, and reproducible experiences [47]. This is in contrast to the internalist epistemology that defines the justification of trust through social determinants or goodwill.

Generally, in order to develop trust in a relationship, there must be an accepted amount of risk. This risk manifests through an expectation that the model will complete the desired request, but we must acknowledge that there is no comprehensive solution for the model to be competent in every way [47]. Trust cannot be built if the trustor cannot will themselves to accept this risk. In these situations, hesitation to accept AI technology is expected, but experiential interaction is paramount. In the absence of this exploratory experience, clinical implementations would likely be costly, cumbersome, and ultimately ineffective.

The 2018 Radiological Society of North America (RSNA) summit on AI makes it apparent that building trust is a key component for the practical implementation of AI [48]. The RSNA indicates that initiatives such as AI education and data curation are of top priority in order to build this trust. With an emphasis on gaining personal experiences with AI, providing clinicians with post-hoc interpretability tools, such as predictive uncertainty estimation, could assist the integration of this technology into the clinical workflow.

4.3 Post-Hoc Predictive Uncertainty

As of today, popular DL algorithms do not provide native uncertainty estimates in regard to output predictions. This seems counter-intuitive since traditionally, classification or segmentation (i.e. pixel-wise classification) tasks output a probability that a particular object or pixel corresponds to 1 of the K number of classes that the network has been trained to identify. This probability is often erroneously interpreted as model confidence which leads to confusion for the clinician [49]. To understand this concretely, consider an extreme case where a sophisticated DL model has randomly initialized weights and has not been trained. In a tumour segmentation task, for example, the sum of the probabilities for a pixel belonging to either of the two possible classes (tumour, not tumour) must sum to 1. Therefore, there can be instances where an untrained network can output high class-wise probability where there is no basis to do so. Figure 4.1 provides an illustration of such a segmentation output. In these cases, predictive uncertainty methods can provide a way to evaluate model confidence at the output.

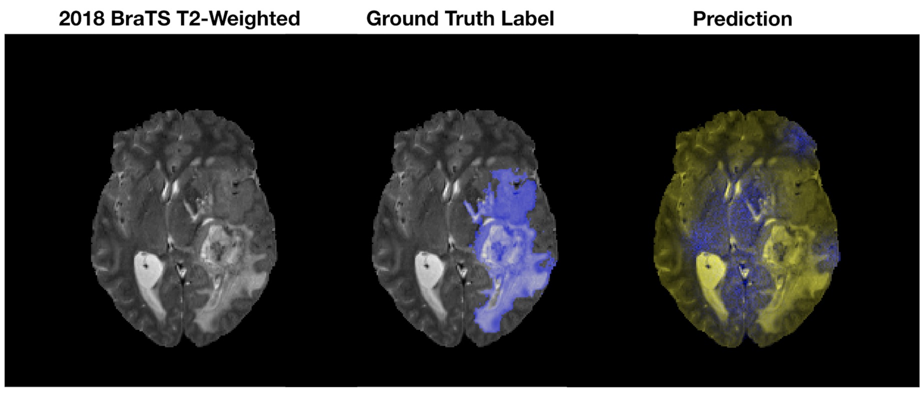

Figure. 4.1. An example of a pixel-wise classification output fused to a sample T2-weighted MR axial slice from the 2018 MICCAI BraTS dataset [50, 51]. The ground truth abnormality is shown in blue. The prediction from the untrained network classifies the abnormal and healthy brain tissues as blue and yellow, respectively.

To provide the end user with additional information that could alleviate the discrepancy between the prediction probability and model confidence, figure 4.1 should be coupled with an uncertainty estimate. The uncertainty should be high in a scenario such as this.

The description of uncertainty becomes clear with an illustrative example. Consider a hypothetical model that has been trained on a large dataset of routine brain MR scans with the goal of being able to determine the volume's scanning sequence (T1-weighted, T2-weighted, T2-FLAIR, etc.). Presumably, if the model has been trained well, it will correctly distinguish the sequence with high confidence. What would happen if the model were to be exposed to a modality it was not trained with? As Gal et al. [52] describes, this is an example of *out-of-distribution* (OOD) data. The desired behaviour of the model in this case would be to try and provide a reasonable prediction, and also report the lack of confidence the model has on its output.

The uncertainty associated with a model's prediction can be broken down into three main factors:

- Sources of random noise within the data (otherwise known as *aleatoric uncertainty*)
- Parameter uncertainty (uncertainty in the model's weights)
- Structure uncertainty (what is the best model for the job)

The combination of the latter two points is defined as epistemic uncertainty, where predictive uncertainty is derived from the addition of the aleatoric and epistemic uncertainties. Epistemic uncertainty can be minimized by training the model with a diverse set of data that covers the range of possibilities the model could be exposed to. Alternatively, aleatoric uncertainty has a fundamental limit to which it can be reduced, since noise can never be fully characterized. With these two quantities, the user can identify if the data quality or the model itself is causing shortcomings in performance [53]. In situations where uncertainty is provided, indications of over or under confidence can directly benefit from human intervention while still providing value to the clinician. These uncertainties become particularly useful when visually displayed, as shown in figure 4.2.

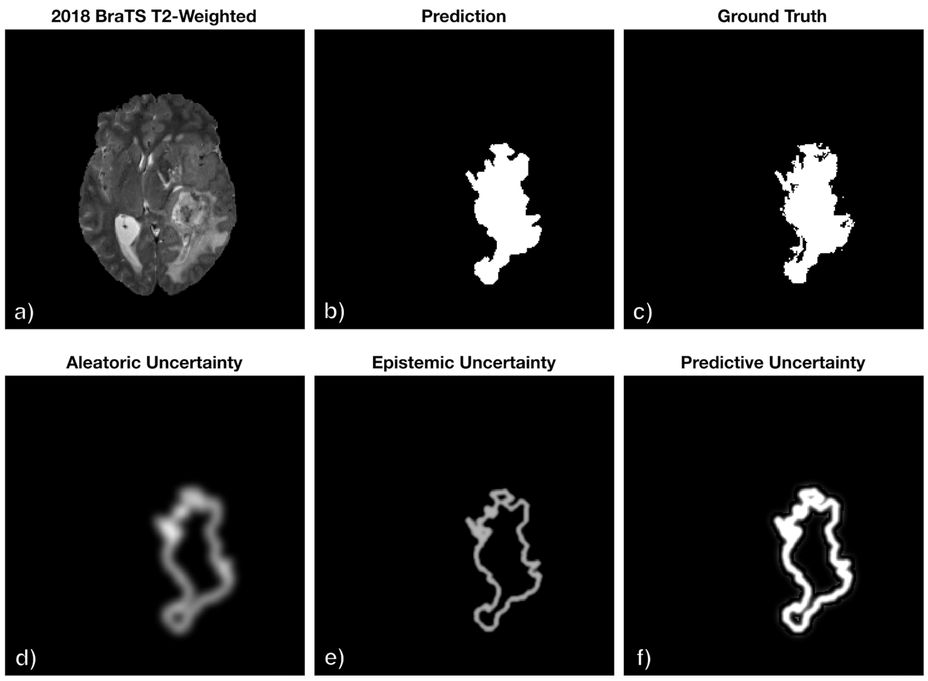

Figure. 4.2. A simulated example of the aleatoric, epistemic, and predictive uncertainties for a pixel-wise classification task. Brighter pixels indicate larger uncertainty. (a) A T2-weighted axial MR slice from the 2018 MICCAI BraTS dataset [50, 51]. (b) The hypothetical segmentation output. (c) The ground truth segmentation. (d) Aleatoric uncertainty localized to the boundaries of the segmentation. (e) Epistemic uncertainty localized to the boundary of the segmentation with less ambiguity compared to the aleatoric uncertainty. (f) Predictive uncertainty, which is the addition of (d) and (e). We notice that the model is confident within the interior of the segmented lesion and less so at the boundary.

Uncertainty can be represented in a variety of different ways depending on the application. In a segmentation task, visualizations can quickly communicate areas in which the model has low or high uncertainty and act as a proxy for the quality of the segmentation [54]. If desired, these values could be aggregated together to report a single numeric uncertainty estimate to enable clinicians to directly compare predictions. Whether a reported value, or set of values, is given within a numeric range, as a confidence interval, or a standard deviation, the representation of uncertainty should be malleable depending on the task and the individuals interpreting the model's prediction. As such, uncertainty estimates can be standardized in relation to the task or discipline as clinicians see fit.

Estimating predictive uncertainty is indispensable in the case of DL and there has been significant work to integrate these methods into prediction pipelines. Some models incorporate prior information (what is known about the data) and likelihood probabilities (how likely this data is to occur) to obtain predictions, other models act in the absence of explicit distributional information. These methods are known as Bayesian and Frequentist statistics, respectively, and both have been used in development of DL predictive uncertainty tools.

4.3.1 Bayesian Neural Networks

Since the inception of ML models for inference tasks, the increase in data complexity has led to the development of more complex models to obtain state-of-the-art predictive performance. As a result, the number of total model parameters has generally increased along with this complexity. In Bayesian inference, the model parameters are seen as a set of random variables, each possessing an intrinsic probability distribution around its mean.

In this formulation, we are ultimately looking to compute a Bayesian model average to determine the probability of the outcome, y, given the data D. This is defined as the predictive probability distribution and is shown below.

$$p(y|D) = \int p(y|\omega)p(\omega|D)d\omega \tag{4.3.1}$$

This probability is unconditional of the network weights, ω, since the integral is marginalizing over ω. Therefore, rather than using a single setting of model parameters, Bayesian inference uses all settings of parameters weighted by their prior probabilities $p(\omega|D)$. The likelihood of the outcome, y, given a specific setting of weights, ω, is denoted as $p(y|\omega)$.

This integral is intractable with a large number of parameters and impractical for neural networks (NNs) [52]. As such, methods such as variational inference (VI) have been developed to approximate the optimal predictive distribution through optimization rather than marginalization.

In order to estimate $p(y|D)$, VI postulates an approximate distribution $q(\omega|D)$ that should be distributionally similar to $p(\omega|D)$. To ensure that the approximate distribution is optimal, the difference between $q(\omega|D)$ and $p(\omega|D)$ is measured and iteratively minimized during network training through a metric known as Kullback-Leibler (KL) Divergence.

$$D_{KL}(P \,||\, Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \tag{4.3.2}$$

Once optimized, the epistemic and aleatoric uncertainties are derived through the variance of the estimated predictive distribution, $p(y|D)$ [55].

A noteworthy limitation of Bayesian modelling is the requirement to inject the necessary prior and likelihood information into the model when determining $q(\omega|D)$. The designer of the model needs to make assumptions about the characteristics of the output distribution, where a Gaussian approximation is typically used [56]. These assumptions typically hinder the optimization process which often results in underestimating the predictive uncertainty [57].

VI has been used for regression, classification, and segmentation tasks with varying degrees of success. Kwon et al. [55] demonstrated this technique by applying it to two multi-sequence MR datasets from the 2015 Ischemic Stroke Lesion Segmentation challenge. With VI, Kwon et al. were able to build on the initial work completed by Kendall et al. [58] and proposed a new way to obtain and decompose predictive uncertainty without incorporating additional parameters into the model. The method is able to provide voxel-wise estimations of the aleatoric and epistemic uncertainties which can then be formed into corresponding visualizations.

VI is a promising Bayesian technique, but it is noted that applications of VI are not frequently used in the area of medical image interpretation (there are few publications of such an application). With the associated complexity of medical image data, optimizing the parameters of the network can be difficult using VI [59]. To obtain similar results without estimating a posterior distribution, methods such as Monte-Carlo (MC) dropout have been formulated to provide a simple way to obtain Bayesian-like uncertainty estimates.

4.3.2 Monte-Carlo Dropout

Initially proposed by Gal et al. [49], MC-dropout utilizes this concept during both the training and testing procedures of the network. The primary goal of MC-dropout is to generate random predictions and interpret the results as samples acquired from the predictive distribution. The randomness in the prediction is a direct result of the dropout process. In order to acquire estimates of the epistemic and aleatoric uncertainties, each input sample is put through an N number of stochastic forward passes. The mean and variance of the set of N predictions is used to determine the uncertainties [52].

Conversely to VI, MC-dropout requires no prior information to be injected into the model and can obtain an approximation of the output distribution without additional bias. As such, MC-dropout can be seen as a Frequentist type of solution to estimating predictive uncertainty. With the comparative ease of implementation with Bayesian-like outputs, MC-dropout has gained more traction within the medical image analysis community compared to its Bayesian counterparts. Nair et al. [60] presented the first exploration of multiple uncertainty estimates using MC-dropout with a 3D multiple-sclerosis (MS) lesion segmentation CNN. The network was trained with a proprietary, large-scale, multi-site, multi-scanner, MS dataset where voxel-wise uncertainty measures were reported. Nair et al. showed that by filtering predictions based on its uncertainty, the model's detection accuracy was greatly improved, particularly in the case of small lesions.

Roy et al. [61] utilized MC-dropout for full brain segmentation with structure-wise uncertainty. Wang et al. [62] derived aleatoric uncertainties with MC-dropout and test-time augmentation for fetal brain tumour segmentation. In both cases, the uncertainty estimates resulted in improved predictive performance by accounting for predictions with low confidence. Although MC-dropout provides a simplistic method for obtaining uncertainty, deep ensembles attempt to obtain even better estimates by aggregating models that have been optimized in different areas in the loss landscape.

4.3.3 Ensemble Methods

The popular method of ensemble-based DL has shown great success for a variety of prediction tasks. By training numerous models, the implementation allows for robust prediction ability in tandem with out-of-distribution stability [63].

A DL ensemble aggregates the results from multiple deterministic NNs trained on different parameter initializations. The framework of the NNs can be identical or different as long as the predictive output is of the same form, such as object-wise or pixel-wise classification. Due to the random initialization, the model starts at a different point within the parameter space and can potentially optimize to different locations within the loss landscape. This is highly advantageous if the purpose is model aggregation over a non-convex loss space [56]. Following Bayesian language, each model should ideally concentrate to different basins of attraction which fundamentally supports diversity within the end prediction. The mean and variance of the prediction can be calculated if the ensemble is treated as a uniformly-weighted Gaussian mixture model.

$$p(y|x) = \frac{1}{M}\sum_{m=1}^{M} p_{\theta_m}(y|x,\theta_m) \tag{4.3.3}$$

Where M represents the number of models used and p_{θ_m} is the probability of prediction y given the input x and the model parameters θ_m. For classification, this corresponds to averaging the predictive probabilities. In regression, the ensemble is further approximated as Gaussian with mean and variance equal to the ensemble defined below [15].

$$\mu_*(x) = \frac{1}{M}\sum_{m} \mu_{\theta_m}(x) \tag{4.3.4}$$

$$\sigma_*^2(x) = \frac{1}{M}\sum_{m}(\sigma_{\theta_m}^2(x) - \mu_{\theta_m}^2(x)) - \mu_*(x) \tag{4.3.5}$$

The generalization ability of an ensemble is often stronger than any of the individual models that compose the framework [15]. Therefore, ensembles derive improvements in performance and natively include uncertainty measures at the cost of training multiple large networks. Compared to its Bayesian counterparts, the lack of prior assumptions allows for more flexible parameter optimization enabling models to optimize into different areas of the loss landscape.

Ensembling DL predictions was popularized by Lakshminarayanan et al. [15] and was evaluated through a series of non-medical regression and classification benchmarks. Since its inception, the method has been extended for other DL applications. De Fauw et al. [16] presented the first clinical application of this method on a large set of optical coherence tomography scans, showing an ensemble was able to achieve comparable expert level segmentation performance. Mehrash et al. [64] applied this process to 2D brain, heart, and prostate segmentation tasks for confidence calibration, showing that ensembles performed better in whole volume and sub-volume cases compared to a non-ensemble framework. In both publications, the uncertainty estimates were used to improve segmentation in ambiguous regions.

4.4 Reliability of Uncertainty Methods

When applying uncertainty metrics to a DL model, how can we evaluate which method is superior? In real-world applications, well calibrated uncertainty estimates are crucial in order to determine if a model's output should be trusted. In this case, proper calibration means that the model should output inference probabilities representative of the true likelihood of occurrence [65]. Further, *good* uncertainty estimates quantify when we can trust a model's predictions [66].

It is critically important to know which methods work most reliably under *dataset shift*, a common problem in medical data. Dataset shift means that something has changed between the training, testing, and clinical distributions, where these shifts are normally attributed to changes in population type, acquisition protocols, and/or annotation inconsistencies. Formally, this means that $p_{test}(y|x) \neq p_{train}(y|x)$, as opposed to the independent and identically distributed (IID) case where $p_{test}(y|x) = p_{train}(y|x)$ holds. As with any machine, imaging systems need regular quality assurance and upgrades to ensure that the image quality is consistent and reliable over the system's lifespan. In effect, this attempts to reduce systematic drifts in the clinical distribution. Therefore, the evaluation of predictive uncertainty is most meaningful during dataset shift and when models are exposed to OOD samples. These situations should arrive at increasing test-time uncertainty.

There has been specific work done in attempt to determine how different methods behave with in-domain and OOD data. Ovadia et al. [67] presented a large-scale benchmark for a variety of classification problems to investigate dataset shift and OOD samples on accuracy and uncertainty calibration for VI, MC-dropout, and ensemble networks. Trained with non-medical data, the quality of the uncertainty degraded with increasing dataset shift independent of the model used. Overall, ensembles were consistently seen to perform the best across all of the tested datasets while being the most robust to shifting, even when using a small number of classification models.

Jungo et al. [68] suggests that uncertainty estimates should be coupled with an evaluation of model calibration to ensure that the estimates are sensitive to dataset shift. In a recent study evaluating ML accuracy on ImageNet, Shankar et al. [69] stated that robustness to small, naturally occurring dataset shifts is a performance dimension that is not addressed by current benchmarks but is easily handled by humans. In order to move towards clinical translation, considerable work must be done to ensure that interpretability metrics, such as predictive uncertainty, can capture these shifts.

4.5 Communicating Predictive Uncertainty

Figure 4.2 illustrates a scenario where the user is provided with a representation of a **voxel-wise** predictive uncertainty measure. Uncertainty on a per-voxel basis can be communicated through the variance of model predictions (obtained through Ensemble methods, for example) or through a more generalized metric known as

predictive entropy (PE). Inspired from information theory, PE attempts to capture the uncertainty of the overall prediction by measuring how close a prediction is to the mid-point of the activation range (0-1). Entropy measures the expected amount of information from an event that has an associated probability of occurrence, p, and mathematically, PE is defined through equation 4.5.1.

$$PE = \sum_{pixels} \sum_{c} (-p(y^* = c|x) \log_2(p(y^* = c|x))) \tag{4.5.1}$$

y^*, c, and x are the class activation, the binary class 0 or 1, and the input, respectively. In computer vision, this corresponds to summing the PE for each pixel (event) in the activation map and reporting a single aggregate PE value. Based on the relationship in equation 4.5.1, the PE is a symmetric function centered at an activation probability of 0.50, as shown in figure 4.3.

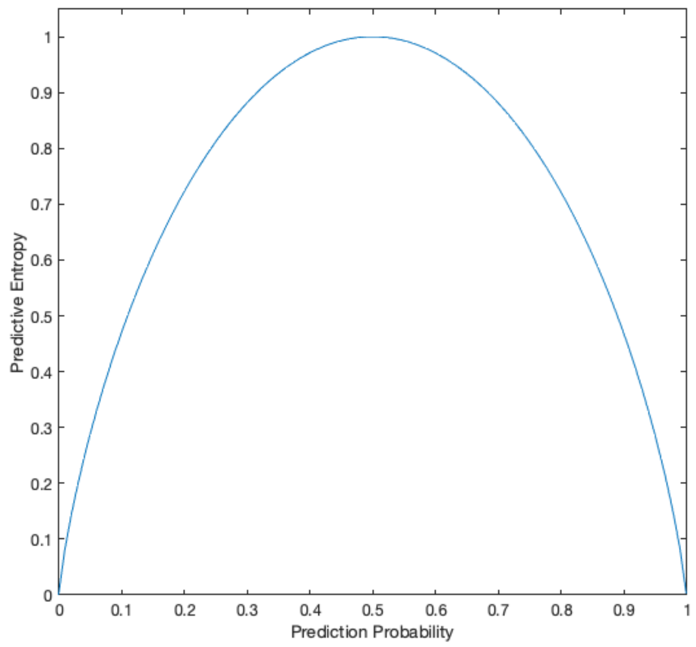

Figure. 4.3. Predictive entropy vs prediction probability for a binary classification problem.

Therefore, the largest predictive entropy for an image is where all pixels are given an activation probability of 0.50, where the output would say "I don't know what to predict", as this produces the largest entropy. After pixel-wise summation, the PE can be normalized by the size of the image to report a value bounded between 0 and 1, indicating certain and uncertain predictions, respectively. As such, PE is an estimate of the image-level uncertainty [54], rather than a voxel-wise level of uncertainty, as illustrated in figure 4.2. In combination with an accuracy metric, such as Dice score, we can observe instances where the structural form of the activation at a particular test-time threshold produces a representative segmentation, but also estimate how confident the model is in that prediction. The combination of structural accuracy and image-wise uncertainty will be the basis of evaluating model performance under a variety of conditions discussed in chapter 6.

Chapter 5

Pre-Processing Methodology

In this chapter, the data pre-processing pipelines and chosen model architectures are discussed. The goal was to implement a pipeline that is able to generate synthetic damage data from normal patient brains and generate voxel-wise labels for the supervised learning task from a large normative dataset for each synthetically damaged brain.

5.1 Synthetic Data Generation

To produce well performing DL models for both in-domain and out-of-domain tasks, a significant amount of data and clever data augmentation techniques during training are needed. Although there are many open-source healthy brain data repositories that one can access, there is a lack of accessible scans of mTBI patients. Furthermore, even if the data are available, the quantity and quality of the data is lacking. Therefore, we developed a pipeline to generate fractional anisotropy volumes from DTI scans and apply synthetic damage comparable to true damaged brains only within WM-regions.

For the purposes of data generation, we make the important assumption that **a patient experiencing mTBI will showcase a localized, radiative reduction in small sub-volumes within the original FA scalar volume**. Though real damage may deviate away from such an assumption, this will provide a foundation for the following work. The pipeline below outlines the steps taken to generate such data. The information specific to MR standardized spaces used for the pipeline are presented afterwards.

1. Determine a global anatomical WM map for all brains in the dataset. To do so, all data must be taken into a standard space.

2. From open-source data repositories [1, 2], collect a large set of *normal*[1] brain MRI DTI data from both male and female populations, with varied ages. Table 5.1 reports the descriptive statistics for the cohort collected. After collection, process all data to determine FA maps.

Sex	**Age Range**	**Total Scans**
Female	26-30	198
Male	26-30	198

Table 5.1: Set of descriptive statistics describing the collected data used for synthetic damage generation and modelling. All volumes collected were age-anonymized prior to access and were placed in the 26-30 age bin on the repositories [1, 2] itself.

3. Remove and set aside a random subset of brains from this normal dataset to be used for creation of the damaged dataset. The subset was assumed to come from the same distribution as the normal data. The subset was removed to ensure that the damaged data is independent of the generated statistical volume (point 4).

4. Determine the voxel-wise FA mean and standard deviation for both male and female normal data by observing the voxel value at the same (x,y,z) spatial location for all sex-relevant brains. Each voxel-wise distribution was assumed

[1]What does “normal” mean? Natural human variability is large and knowing when a sample is “abnormal” can be difficult. We assume that non-damaged brains are normal, but this might not always be the case.

to be Gaussian. 198 female and 198 male brains were used for the generation of each male/female statistical volume.

5. Randomly select a number of square sub-volumes (bounded between 1-5) and randomly determine the dimension for each direction (bounded between 10-30 voxels). "Bounded" indicates that the value must be equal to or between the range specified. Upper and lower bound values are chosen arbitrarily but are restricted to be smaller than the patient volume and small enough to indicate localized damage.

6. For each sub-volume, apply an inverted 3D-Gaussian probability density function with random standard deviations in the x, y, and z-directions. The amplitude of the distribution is modulated through the standard deviation, where the applied damage is bounded such that we apply a 5% to 20% reduction of the initial FA value. This is a liberal damage estimate but consistent with the literature [70]. The multivariate normal density distribution used for damage generation is reported in equation 5.1.1,

$$f_X(x_1, ..., x_k) = 1 - \frac{1}{\sqrt{2\pi^k \det(\Sigma)}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{5.1.1}$$

where Σ is a positive-definite covariance matrix, x is a real k-dimensional column vector, and μ is a k-dimensional mean vector. The amplitude of the distribution can be modulated through Σ because the integral within an n-sigma interval is constant and thus, the distribution must sharpen or widen depending on Σ.

7. The set of damage sub-volumes is applied to a **single** brain from the excluded normal dataset. Each sub-volume index V_{ijk} is multiplied with the corresponding FA index FA_{ijk}, such that

$$FA^{'}_{ijk} = FA_{ijk} V_{ijk} \tag{5.1.2}$$

where $FA'_{ijk} < FA_{ijk}$ and for integers $0 \leq i, j, k \leq 192$.

The damage is applied *only to corresponding* i, j, k *indices of WM* in the normal brain, thus creating a new *damaged* brain. To ensure that all other tissue is ignored within the sub-volumes, the WM mask determined in step 1 is utilized. Sub-volumes are permitted to overlap.

8. Using the mean and standard deviation statistical FA volumes produced in step 4, each voxel within the augmented brain is checked to determine if its value is greater or less than 2σ from the sex-relevant normal mean. If so, this voxel is labelled as damaged. By using this construction, not all voxels that experience a reduction in FA are considered to be statistically significant. After checking all voxels in the FA volume, a single binary labelled volume is created for the supervised learning task.

9. Repeat steps 5-8 ten times for each brain in the excluded dataset to increase the size of the damaged dataset by 10x. During model training, samples from the same patient will either lie in the training, validation, or testing sets, thus removing the chance of memorization and artificially boosting test-time performance.

The generation of the global WM mask used in the aforementioned pipeline was constructed using 18 different regions-of-interests (ROIs) related to the global structure of WM tracts throughout the brain. The ROIs are as follows:

Brain Regions	**Associated Symptoms**
Left Acoustic Radiation	Deficits in speech related comprehension, along with environmental and verbal auditory agnosia.
Right Acoustic Radiation	
Callosal Body	Deficits in language, visual, and tactile information processing. Behavioural changes.
Left Cingulum	Deficits in executive control and episodic memory. Sensations of pain and depression.
Right Cingulum	
Left Corticospinal Tract	Reduced control of voluntary movements. Bilateral pain. Ipsilateral paralysis, paresis, hypertonia.
Right Corticospinal Tract	
Fornix	Deficits in verbal, declarative, and episodic memory.
Left Inferior Occipito-Frontal Fascicle	Deficits in social cognition, decision-making, empathy, attention, multi-tasking, episodic memory, and executive function.
Right Inferior Occipito-Frontal Fascicle	
Left Optic Radiation	Reduced visual field, sensitivity to light, dizziness, anopsias, scotomas
Right Optic Radiation	
Left Superior Longitudinal Fascicle	Deficits in attention, working memory, theory of mind. Visuospatial disfunction.
Right Superior Longitudinal Fascicle	
Left Superior Occipito-Frontal Fascicle	Deficits in visual and cognitive processing, reduced peripheral visual field, difficult grasping objects.
Right Superior Occipito-Frontal Fascicle	
Left Uncinate Fascicle	Behavioural changes, social deprivation. Alterations in risk taking, social conduct, and substance abuse.
Right Uncinate Fascicle	

Table 5.2: List of Structural Brain Regions, based on the Juelich Histological probabilistic brain atlas. [3]

The anatomic ROIs were determined using the Juelich Histological probabilistic brain atlas based on cyto- and myelo-architectonic segmentations [3]. The microscopic and quantitative histological examination of ten human post-mortem brains were used in the construction of the atlas. The atlas is transformed into the Montreal Neurological Institute (MNI)-space for co-registration of new data [1]. Due to the probabilistic nature of the mapping (i.e. every brain is different), each brain voxel is given a probability that the voxel is included within a particular ROI. As such, an inclusion criterion of > 0.9 (90%) is used to classify a voxel into various ROIs. The combination of all of the ROIs listed creates the desired WM mask. Figure 5.1 shows the WM mask used for damage sub-volumes where brighter regions indicate ROI overlap due to the inclusion criterion.

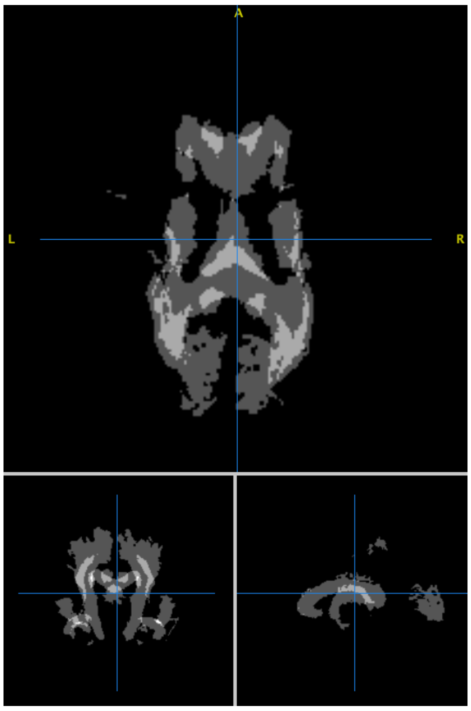

Figure. 5.1. Axial, sagittal, and coronal slices of the WM mask generated through the Juelich Histological Atlas. All WM ROIs indicated through table 5.2 are shown. Brighter regions indicate ROI overlap based on the 90% inclusion criterion.

Normal, healthy control subjects (ages 26-30, 198 female and 198 male) were sourced from the following open-source repositories [1, 2].

- The Human Connectome Project (HCP)
- International Consortium for Brain Mapping (ICBM)
- Parkinson's Progression Markers Initiative (PPMI)

Inclusion criteria required data to be from either GE or Siemens 3T machines with b-values = $1000 s/mm^2$. All DWI volumes underwent the following pre-processing,

1. Convert DICOM images to NiFTi files using dcm2niix [71]. This is a lossless conversion and no (additional) artifacts should be present after conversion.

2. Eddy-current correction using FSL command *eddy_correct* [72].

3. Skull-stripping using the Brain Extraction Tool (FSL-BET2) [72].

4. Co-registration into the MNI152-space. The reference volume is a 1mm isotropic T1-weighted scan. All DWI volumes were registered into 182x182x218 volumes using 12 degrees-of-freedom and trilinear interpolation.

5. FSL DTIFit to create the corresponding FA, MD, RD, AD tensor maps [72].

At the end of this pipeline, DTI scans have been used to calculate FA scalar volumes and are ready for data manipulation and statistical analyses. DTI processing and damage pre-processing was done using FSL and MATLAB. Nicholas Simard, a PhD student in Electrical and Computer Engineering supervised by Dr. Michael Noseworthy, kindly developed the entirety of the aforementioned pipeline.

An example image of a FA volume with synthetic damage and its corresponding label are shown in figure 5.2.

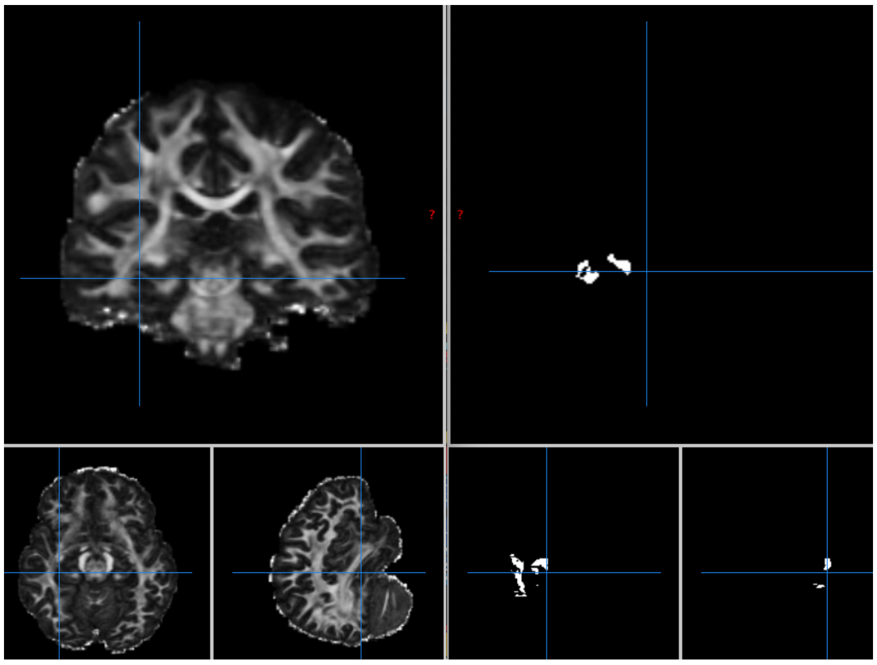

Figure. 5.2. Example of a FA map from a healthy brain with the addition of the corresponding label map. Average FA reduction in labelled voxels is 7.2% ± 0.6%.

During data generation, 13 female and 13 male brains were set aside from the large normal dataset for damage augmentation, where the normative dataset was *only* used for the construction of voxel-wise labels per brain. Since an assumption was made that the FA value at an (x,y,z) spatial location across all sex-relevant patients in the normal dataset was Gaussian, the normal dataset should be as large as possible. By generating 10x the data from a single brain, 13 brains/sex was chosen as to maintain a balance between having a large normative dataset and creating a dataset of reasonable size for training NNs.

With volume dimensions of 182x182x218, there are 218 slices per brain when sliced in the axial direction. However, a typical MR scan will have empty space above and below the brain, resulting in images that have no signal. For the purposes of the analysis, we remove 20% of the slices above and below to avoid having a significant amount of empty slices within the datasets that would redundantly increase computation times. When removing 40% of each volume (86 slices), on average 10.2±1.7

slices contained non-zero values with non-zero proportions of 3.1%±0.7% over all removed slices. The variation in the number of slices with non-zero values is due to varying morphology over the patient population. With a nominal computation time across models of 3.43ms per image while on the GPU, the total computation time for models trained with and without the removed empty slices are $0.054n$ and $0.033n$, respectively, where n is the number of EPOCHS. As such, removing 40% of the data results in complete model training 1.6x faster[2]. With this rational, the middle 60% of the brain is saved, resulting in 131 slices per registered brain. By increasing the amount of initial data by 10x through augmentation, the total amount of damaged brain data available for training, validation, and testing are 17030 images for both female and male brains (34060 total).

5.2 Model Training, Validation, and Testing

An open-source Github repository called *Segmentation Models PyTorch* for instantiating many 2D-UNet architectures with various encoding backbones was utilized [73]. As such, all model training, validation, and testing was completed using PyTorch 1.4.0, CUDA 10.1, and a GPU computing cluster with an Intel Core i9-9820X 3.30GHz CPU, 64GB RAM, and an NVIDIA GeForce RTX 2080 Ti with 12GB of memory. We instantiate three 2D-UNet models with Resnet101 (total params: 51,506,961), Vgg19 (total params: 29,056,785), and Densenet121 (total params: 6,320,721) encoding backbones. This was done to observe the differences in performance depending on how the model encoded information into the latent space. Model results were ensembled and test-time performance was evaluated for each model and ensemble.

With a 2D-UNet architecture, the model is only capable of processing 2D spatial images with an arbitrary number of colour channels. Furthermore, feeding data into

[2] $time_{without} = \frac{0.00343s}{3600s}(34060)n = 0.033n$. $time_{with} = \frac{0.00343s}{3600s}(56680)n = 0.054n$

the model becomes difficult if the data are stored in a non-standard format. As such, instead of creating custom PyTorch data loaders that naively take NiFTi volumes, all NIfTis were converted into tiff slices using Python and Pillow prior to data ingestion. This conversion is lossless[3]. Each image has a unique identifier that relates the image back to a particular patient (such as 10-img1.tiff, 10-img2.tiff, 20-img1.tiff, 20-img2.tiff, to indicate patients 10 and 20, respectively, where img1 and img2 refer to two sequential slices within the MR volume). Slices from the same patient were not used across the training/validation/testing splits and we ensure an equal male/female balance for each split.

Synthetic damaged patient data was split into training, validation, and testing sets based a user defined fraction. We utilized a 60/15/25 train/val/test split (20960, 5240, 7860 images), where the data in each set was unique to a subset of patients. We trained each model with 80 EPOCHS. An initial learning rate of $\alpha = 10^{-4}$ was used and reduced by an order of magnitude every 10 EPOCHS if the average training loss stagnated with a difference of 0.5% compared to the loss at the beginning of the 10 EPOCHS. Adam optimization was used during stochastic gradient descent with weighted binary cross-entropy loss [75]. Dropout probability of 0.40 was used since this showed the best test-time results. We chose a batch size of 16.

During training and validation procedures, data were exposed to a set of random affine transformations including rotation between [-45^o, 45^o], scaling from [1, 1.3], and horizontal flipping with probability 0.50. All images were zero-padded to 192x192 to meet geometric image constrains (i.e. need to be integer sizes of 64*n) for proper CNN processing [73]. To ensure that models were comparable, the training/validation/test sets contained proper subject partitions and these sets were the same across models, but data augmentation during training was kept random to reinforce diversity. A

[3]Loading a NiFTi image using *nibabel* [74], extracting the data using *nib.get_fdata()*, and comparing the stack of tiffs to the NiFTi image using *numpy.allclose(nifti, tiffs)* produces equality.

global seed of 1234 was set to ensure reproducibility. Data augmentation was applied during test-time under specific conditions and discussed in section 5.3. Each of the training, validation, and testing pipelines can be seen in *smp_main.py*, Appendix A. Test-time performance was evaluated through the Dice score with various test-time threshold conditions ranging from 0-1 with increments of 0.1.

5.3 Baseline Model Performance

The training / validation BCE loss as a function of the epoch for the three models used for the ensemble is shown in figure 5.3. Validation loss is expected to be noisy compared to training loss as we are optimizing on the training set explicitly. Temporary spikes in the validation loss, as seen in Densenet121 at epoch 63, can occur if the local minima at the current epoch does not generalize as well as the minima at the previous epoch. If the validation loss continues to follow the training loss after such a spike, the model has deviated away from this sub-optimal minima.

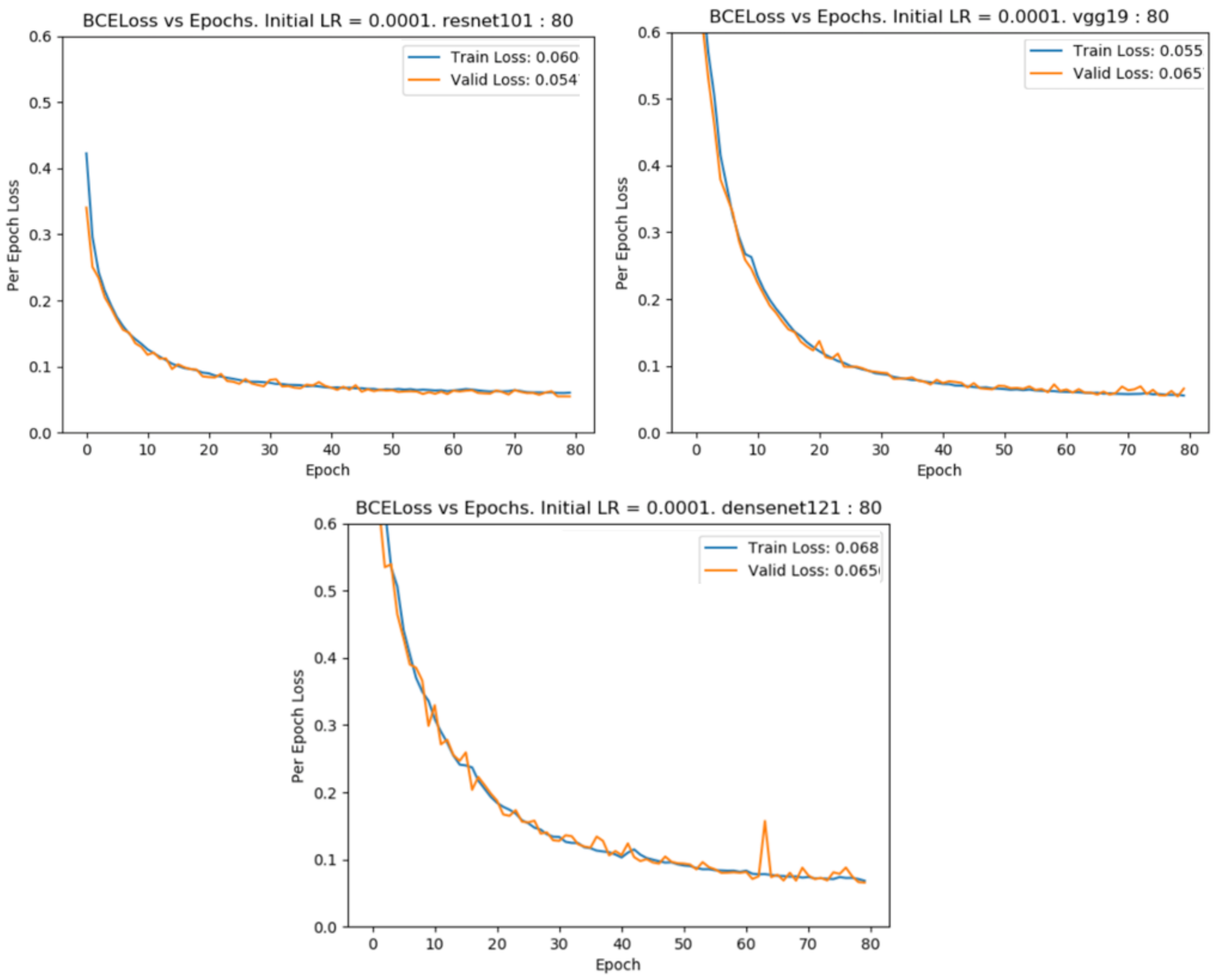

Figure. 5.3. BCE Loss vs Epoch for ResNet101, Vgg19, and Densenet121 models for 80 EPOCHS. All models are shown to have trained well as training and validation loss agree over all EPOCHS.

To succinctly communicate Dice score performance of each model and the ensemble during test-time, the peak mean Dice score (PMDSC) for each distortion value can be reported. The PMDSC is the largest average Dice score evaluated over the entire test-set after checking the [0-1] probability threshold range. The threshold is applied to convert the output activation maps to a binary prediction and thus, a single threshold must be chosen in order to compare the model prediction to its corresponding label. The PMDSC provides a simple way to communicate model performance independent of threshold. This rationale is based on the ideal that if a model were to be clinically deployed, we would either fix the test-time threshold value where we obtain optimal segmentation accuracy during testing, or implement a threshold slider for the user during inference. Further discussion regarding the derivation of the PMDSC is reported in Appendix B.

The DSC of each model and the ensemble as a function of the test-time threshold prior to augmentation is shown in figure 5.4. Model DSC performance at a particular threshold is calculated by taking the mean DSC over the entire test-set. Normalized baseline true positive rate (TPR), false negative rate (FNR), false positive rate (FPR), true negative rate (TNR), Dice Score, and precision results for all models are reported in table 5.3. Numbers in the table that have been bolded indicate the best value within the column. The DSC reported in table 5.3 is the peak DSC for each model illustrated in figure 5.4. Absolute pixel quantities are provided in table 5.4, where it should be noted that the number of FPs greatly exceeds the number of TP for all models. This is hypothesized to be attributed to the models over-predicting lesion size for small lesions, but further work should investigate this result.

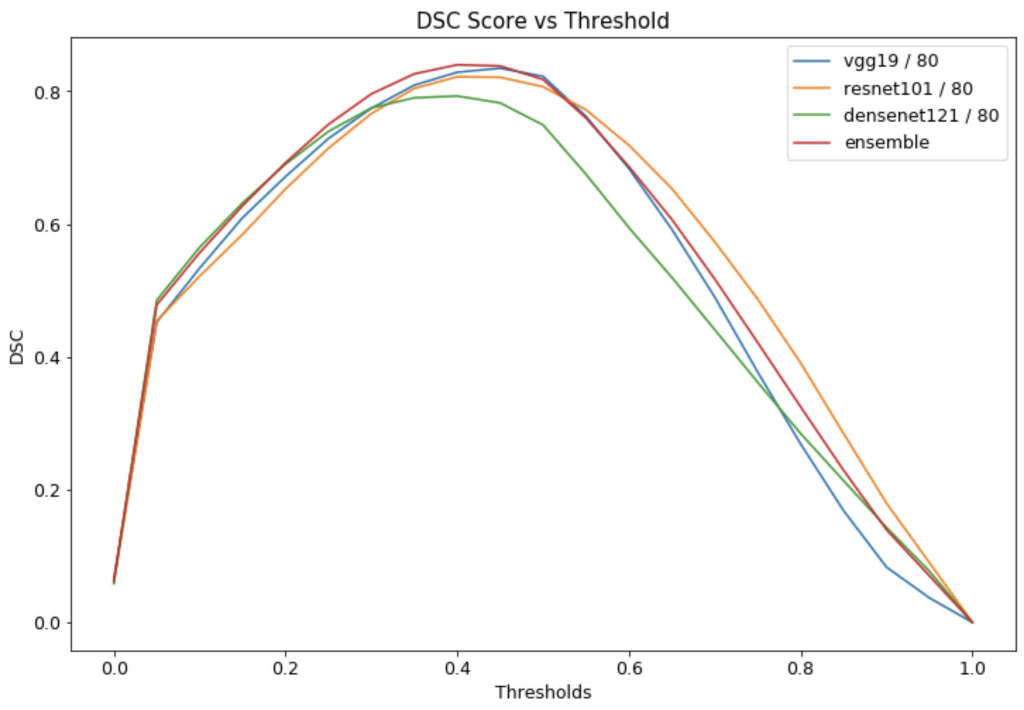

Figure. 5.4. The DSC score for each model and the ensemble as a function of test-time threshold prior to augmentation. The threshold chosen is the one that provides the highest performance. Thresholds for Vgg, Resnet, Densenet, and Ensemble models are 0.45, 0.40, 0.40, 0.40, respectively. Peak DSC is reported in table 5.3.

Models	**TPR (Recall)**	**FNR**	**FPR**	**TNR**	**PMDSC**	**Precision**
Vgg19	0.87	0.13	0.22	**0.78**	0.83	0.79
Resnet101	0.81	0.19	0.15	0.85	0.81	**0.84**
Densenet121	0.84	0.16	0.25	0.75	0.80	0.77
Ensemble	**0.91**	**0.091**	**0.23**	0.77	**0.85**	0.80

Table 5.3: Normalized confusion matricies for each model. Columns report pre-augmentation (baseline) true positive rate (TPR), false negative rate (FNR), false positive rate (FPR), true negative rate (TNR), Peak Mean Dice Score (PMDSC), and precision results. Bolded values indicate the best scoring model in the column. Baseline ensemble results indicate that it obtains the best performance for TPR, FNR, FPR, and PMDSC.

Models	**TP**	**FN**	**FP**	**TN**
Vgg19	11513771	1720447	60833700	215683120
Resnet101	10719718	2514501	41477523	235039296
Densenet121	11116744	2117475	69129204	207387615
Ensemble	12043138	1191080	63598868	212917952

Table 5.4: Baseline confusion matrices with absolute pixel values for each model. Columns report the TP, FN, FP, and TN values. It should be noted that the number of FPs is much larger than the number of TPs.

Chapter 6

Experiments and Results

In this chapter, the experiments conducted to illustrate the effect of in-domain dataset shift and OOD samples are described. In particular, we are interested in the following questions:

1. Does predictive uncertainty increase with dataset shift and/or OOD samples?
2. Can predictive uncertainty indicate model prediction confidence for a new test-time sample?

In the pursuit to answer these questions, we evaluate an ensemble though conditions that alter the testing distribution, a situation that is likely to occur during model deployment. This is done through various levels of *geometric distortions*, *adversarial perturbations*, and *decreasing signal-to-noise ratios*. The results are communicated through the overall accuracy of the segmentation, the PE, and the variation in PE across models for the same test-time image. The correlations between each of these metrics are reported through the Spearman rank coefficient, ρ, to account for potential non-linear relationships.

Varying the SNR is understood as an in-domain dataset shift, where adversarial perturbations and image distortion are true OOD augmentations applied to in-domain data. It should be noted again that these augmentations were not applied during the training / validation procedures of the network. **It is hypothesized that decreased Dice scores, increased PE, and increased PE variability with more extreme test-time augmentations should be observed**.

To illustrate the output from the set of models used in the ensemble, figure 6.1 reports the pixel-wise activation maps for each model, the PE of the activation map, and the binarized image after thresholding. All models were given the same FA slice.

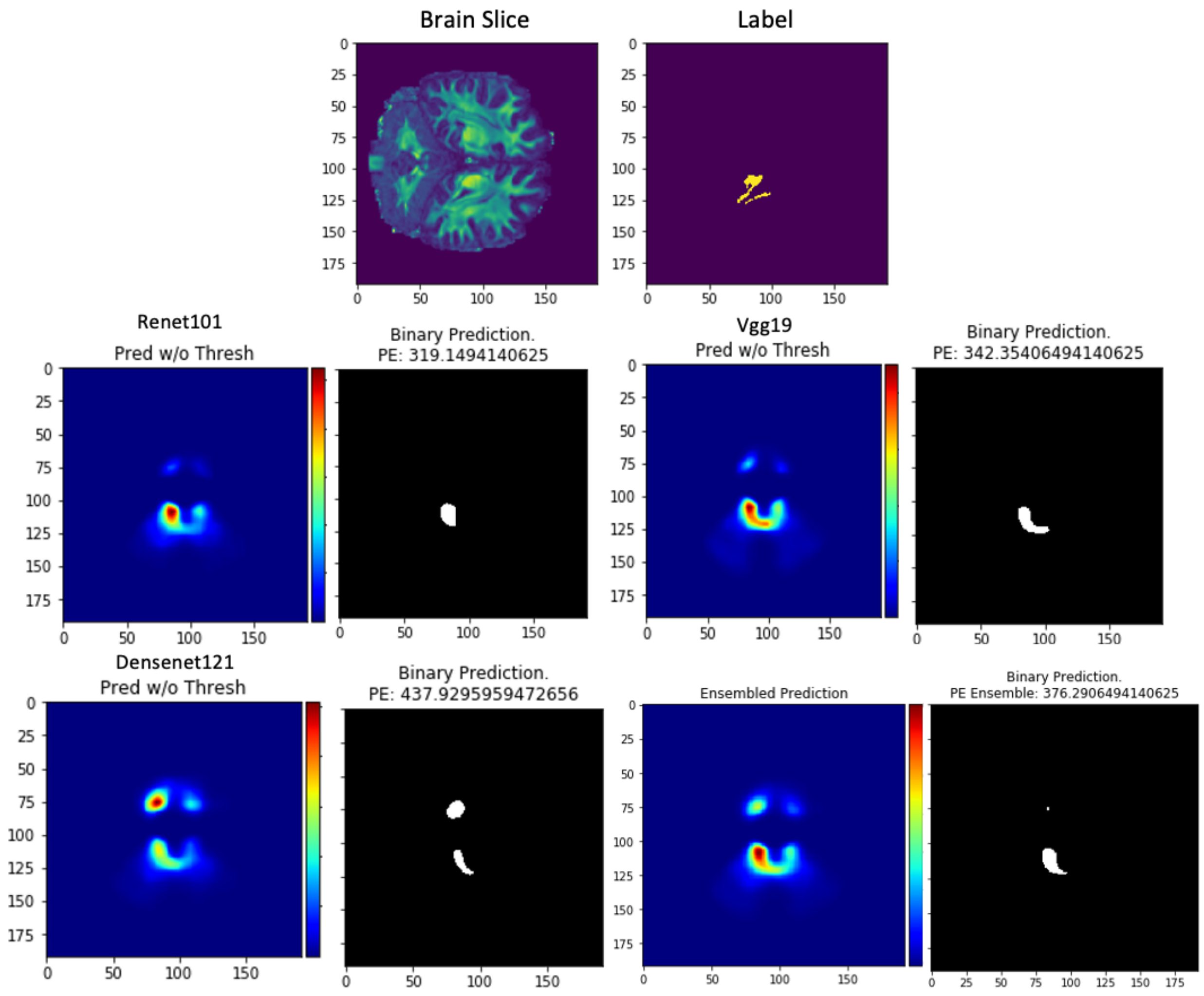

Figure. 6.1. Model predictions for a single FA slice to illustrate the variability that can occur between Resnet101, Vgg19, Densenet121, and ensemble models. PE for the prediction is reported in the title of the binary outputs. Results reported in table 6.1.

Model	Dice Score	PE (Non-Norm)	PE-Std (Non-Norm)
Resnet101	0.54	319.15	N/A
Vgg19	0.54	342.35	N/A
Densenet121	0.32	437.93	N/A
Ensemble	0.48	376.29	51.41

Table 6.1: Dice Score, PE, and PE-Std for all models shown in figure 6.1. The ensemble reports a mean Dice Score over individual model predictions and a PE value slightly higher than the $PE_{mean} = 366.48 < 376.29$.

6.1 Distortion

Distortion was applied by using the *torchvision.transforms.RandomPerspective* function with various degrees of distortion scaling set to 0.10, 0.30, 0.50, 0.70, or 1. The function performs a random perspective transformation with probability p, where we choose $p = 1$. This is a linear projection in 3D space and any interpolation needed is done using bi-linear interpolation. The deformation of brain slices with these varying degrees of distortion is shown in figure 6.2.

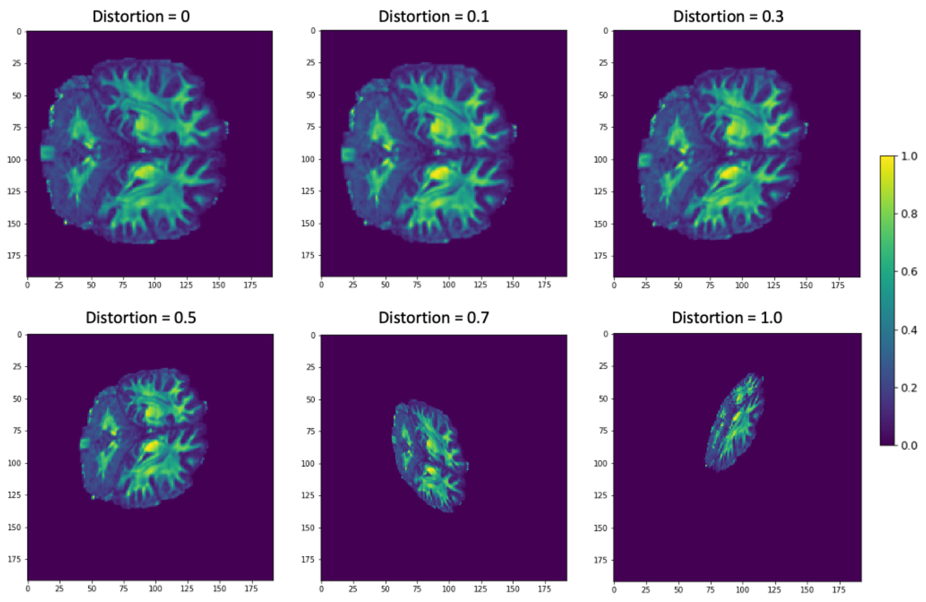

Figure. 6.2. A single coronal slice showing FA with increasing distortion scaling.

As the scaling becomes large, the distortion observed becomes characteristically unrealistic in the context of imaging. However, we are interested in observing the results in situations where the distortion is exacerbated even if this is outside of a physical regime. The PMDSC for all models and PE histograms for the ensemble for the chosen distortion conditions are shown in figures 6.3 and 6.4 below.

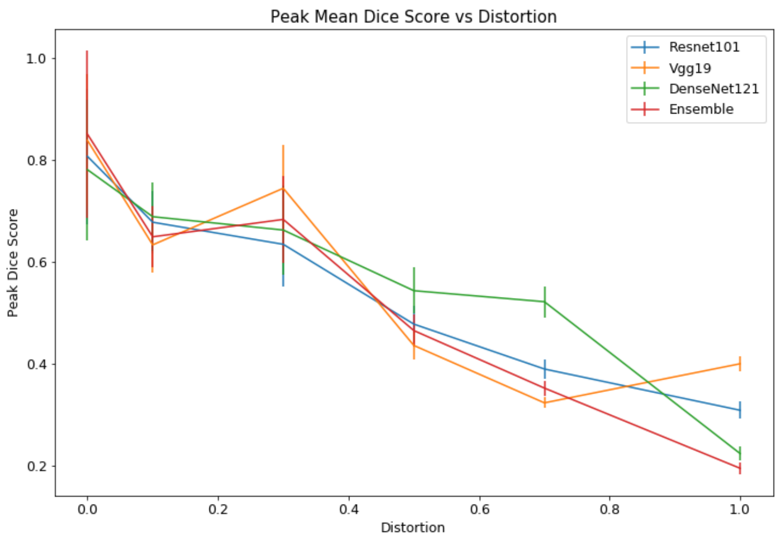

Figure. 6.3. Peak Mean Dice Score (PMDSC) for increasing levels of test-time distortion. $\rho_{ensemble} = -0.94$. A linear trend towards decreasing PMDSC with increasing distortion is observed. 2σ confidence intervals indicate the variability in Dice score over all samples in the test set at the threshold that obtains the highest mean segmentation accuracy.

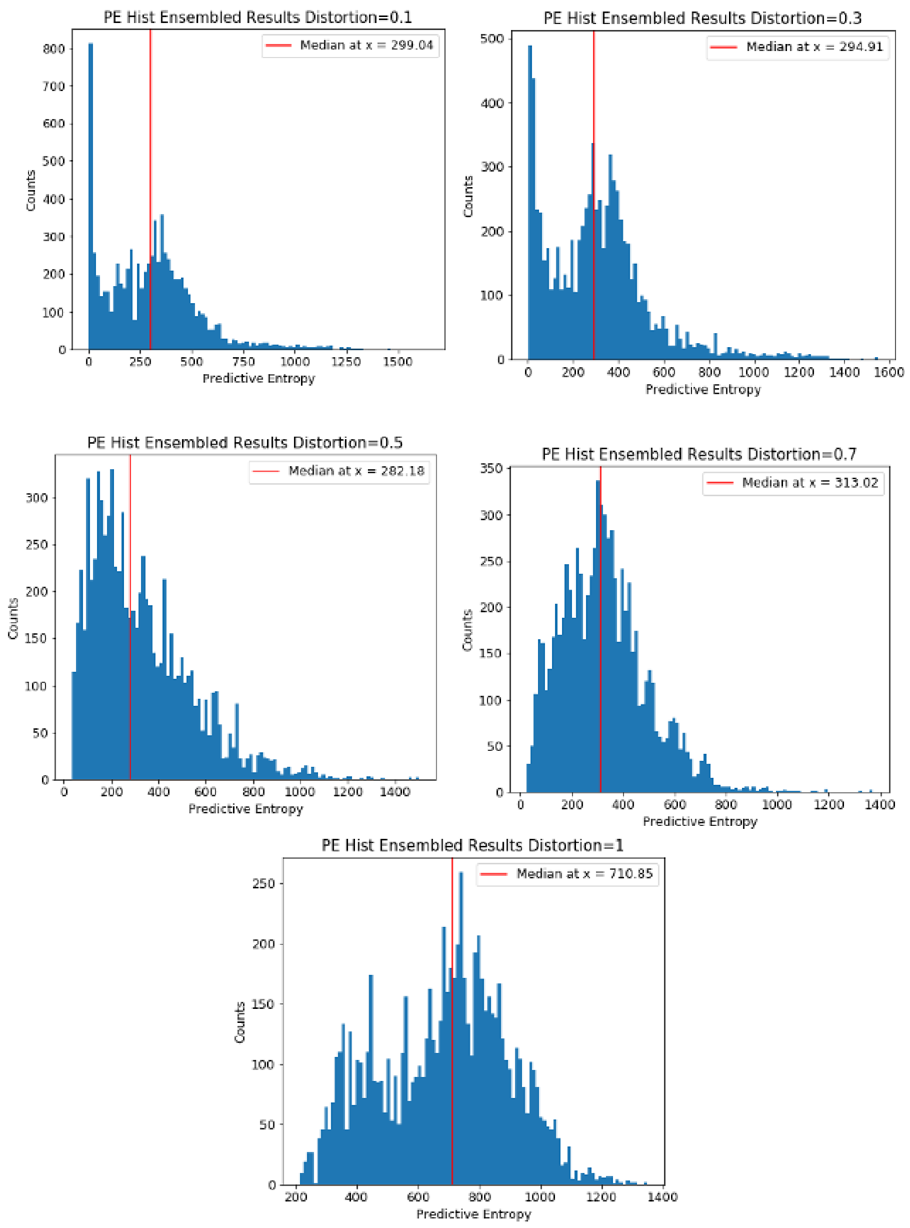

Figure. 6.4. Histograms of the non-normalized predictive entropy (PE) with increasing distortion for the ensembled model. x and y axes are the PE and number of counts, respectively. As the distributions are broad, the median is reported as it is more robust to outliers.

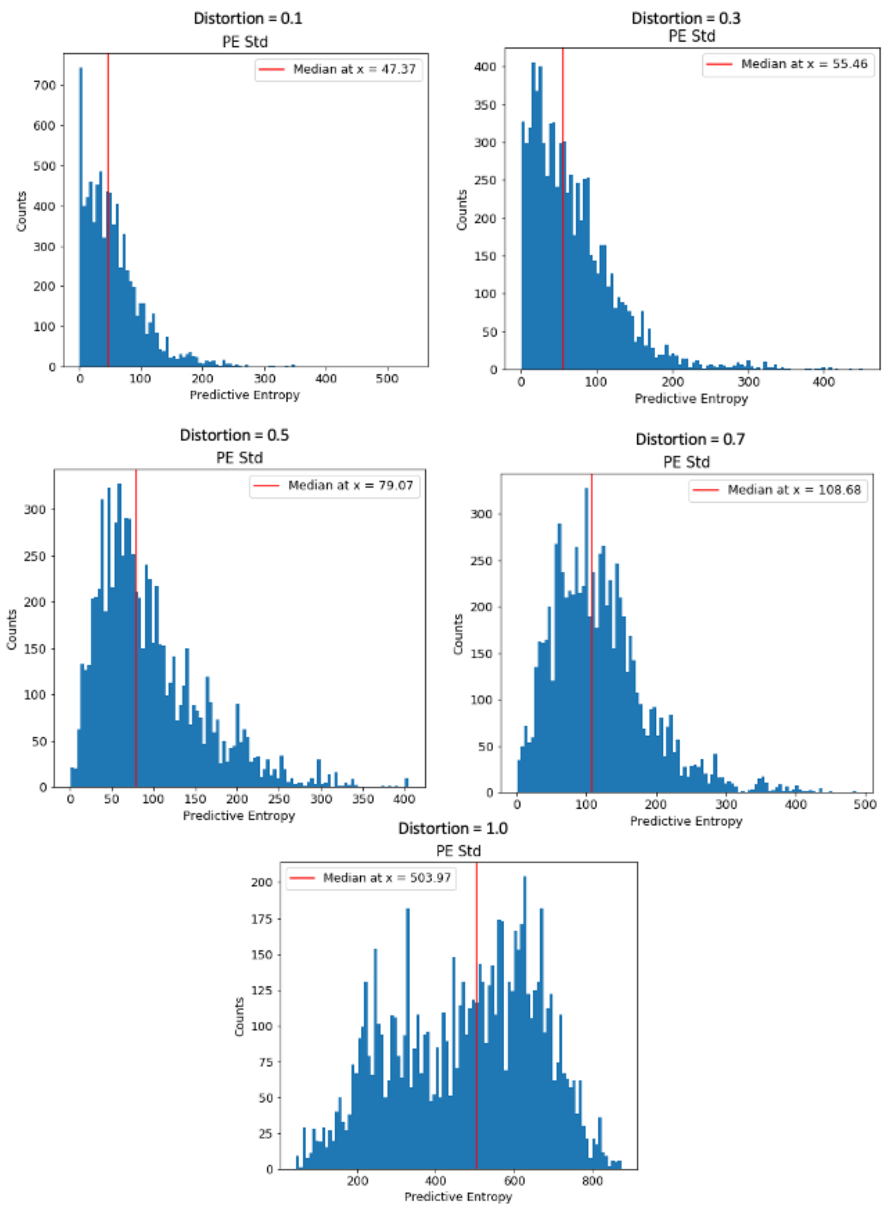

Figure. 6.5. Histograms of the non-normalized predictive entropy variability (PE-Std) with increasing distortion for the ensembled model. x and y axes are the PE and number of counts, respectively. PE Std values are determined by taking standard deviation of the PE values produced from the Resnet, Vgg, and Densenet predictions for each image in the test set. As the distributions are broad, the median is reported as it is more robust to outliers.

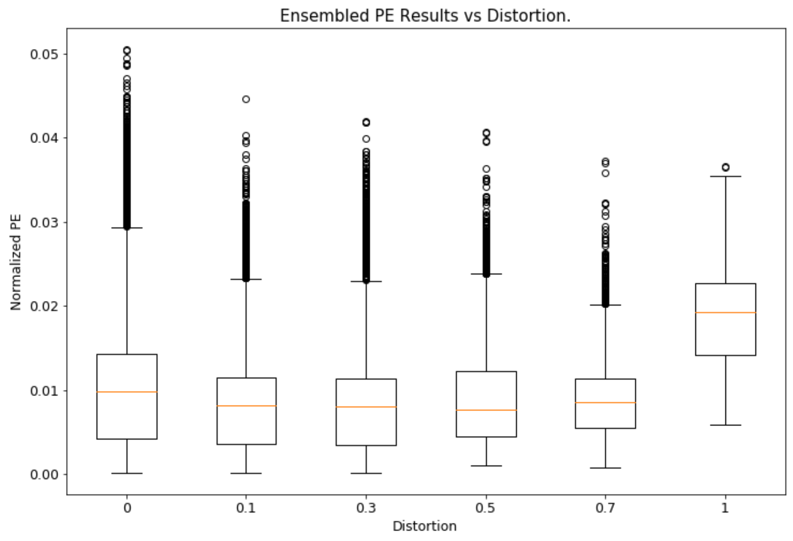

Figure. 6.6. Normalized PE-distortion whisker plot summarizing median values with boxes extending to the upper and lower quartiles of the data. Whiskers extend to show the data range. Outliers are shown as points. $\rho = 0.20$

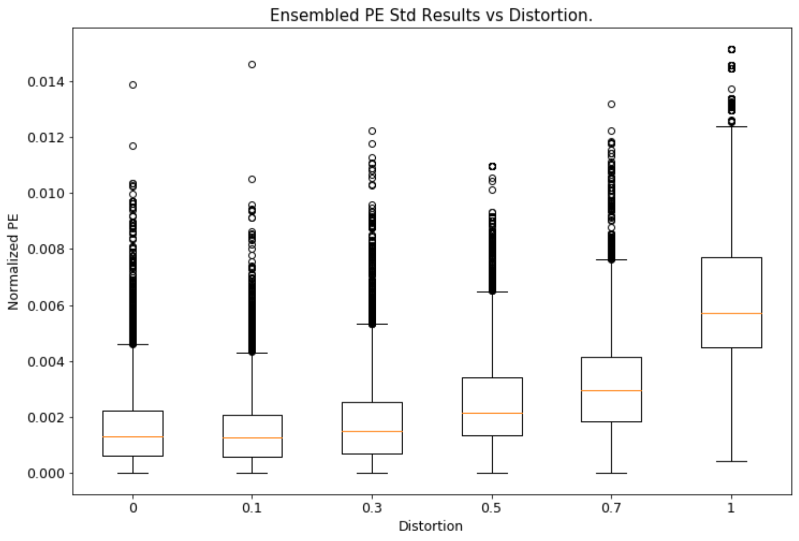

Figure. 6.7. Normalized PE-distortion Std whisker plots summarizing median values with boxes extending to the upper and lower quartiles of the data. Whiskers extend to show the data range. Outliers are shown as points. $\rho = 0.94$

Distortion	Peak Mean DSC (Std)	Median PE [Norm, Non-Norm]	Normalized Median PE [Lower, Upper Quartile]	Median PE Std [Norm, Non-Norm]	Normalized Median PE Std [Lower, Upper Quartile]
0	0.851(0.164)	0.00982, 362.0	0.00415, 0.0143	0.00131, 48.29	0.000635, 0.00222
0.10	0.649(0.0604)*	0.00811, 299.04*	0.00361, 0.0114	0.00128, 47.37*	0.000579, 0.00207
0.30	0.683(0.0844)*	0.00799, 294.91*	0.00350, 0.0113	0.00150, 55.46*	0.000702, 0.00255
0.50	0.465(0.0309)*	0.00765, 282.18*	0.00448, 0.0122	0.00214, 79.07*	0.00134, 0.00304
0.70	0.352(0.0148)*	0.00849, 313.02*	0.00549, 0.0113	0.00265, 108.68*	0.00183, 0.00415
1	0.195(0.0122)*	0.0192, 710.85*	0.0141, 0.0227	0.00570, 210.19*	0.00455, 0.00755

Table 6.2: Ensemble segmentation performance and predictive entropy with increasing distortion. *: difference relative to baseline is statistically different (one-sample t-test, $p < 0.05/N$, $N = 7860$). Image normalization [192x192].

To determine if the ensemble results for each distortion are statistically significant, we take the difference between each distortion dataset and the baseline to ensure independence. Using a one sample t-test with the Bonferroni correction of α/N for each of the PMDSC and PE difference, distortion values that result in a rejection of the null hypothesis is indicated with an asterisk (*), as seen in table 6.2.

As seen in figure 6.4, we observe an strong linear trend towards poorer PMDSC with increasing distortion ($\rho_{ensemble}$ = -0.94). Interestingly, there is a flat relationship between the ensemble PE and increasing distortion (until a distortion value of 1) which was not expected. We can qualitatively observe that the PE distributions become broadened with increasing distortion, but this is not captured by the median value alone. Therefore, a PE value should be coupled with an estimation of the variation between all model predictions prior to aggregation. When reporting the PE-Std between Resnet101, Vgg19, and Densenet121 U-Net models for each image during test-time, as seen in Figure 6.7, we begin to observe that the set of models show increasing prediction variability with increasing distortion. This follows the expectation that each model will have optimized to learn particular features within the dataset which manifest in performance variability during these test-time augmentations. With increasing distortion, models are expected to perform inconsistently with each other on the same test-time image.

It is key to note that the absolute value of the PE is *low* regardless of the distortion, indicating that voxel-wise activation maps for all model predictions lie close to either 0 or 1, as shown in figure 4.3. With image sizes of 192×192, the maximum non-normalized PE would be 36864. Therefore, in the most extreme distortion case, we only observe an absolute PE of 1.9% of the total range. When reporting the median PE and PE-Std percentage changes relative to their respective baseline values, we can arrive at important trends independent of the absolute PE values, as reported in table 6.3.

Distortion	**Relative PE % Change**	**Relative PE-Std % Change**
0	N/A	N/A
0.10	-17.4%	-1.8%
0.30	-18.8%	14.8%
0.50	-22.0%	63.7%
0.70	-13.5%	125.2%
1	96.4%	335.3%

Table 6.3: Percentage change of the median PE and PE-Std relative to baseline values for increasing distortion.

These results suggest that even though we have a weak relationship of increasing PE, the relative PE-Std percentage indicates that we can observe significant changes in the median value once the distortion becomes geometrically noticeable (0.30), as seen in Figure 6.2. Therefore, a single PE value alone cannot indicate whether or not the ensemble is uncertain or not, but rather, the variability between model predictions provides valuable insight as to how we should interpret the ensembled prediction for OOD data.

6.2 Adversarial Perturbations

ML models have been shown to perform particularly poor when exposed to adversarial examples – in-domain data that has been corrupted by some perturbation that explicitly *increases* the loss function. The magnitude of the adversarial perturbation is bounded such that the L_∞-norm is less than some user-defined hyperparameter ϵ. Mathematically, $||X - \hat{X}||_\infty < \epsilon$. Epsilon should be chosen to cause an imperceptible perturbation of the input image X. During test-time, an in-domain image is propagated through the network and the gradient of the loss function with respect to the input image is determined through back-propagation. Here the Fast-Gradient Sign Method (FGSM) [76] is employed to generate an adversarial example, $\hat{X}$, through

$$\hat{X} = X + \epsilon\,\mathrm{sign}(\nabla_X E(X, Y_{true})) \tag{6.2.1}$$

An example $\hat{X}$ with $\epsilon = 0.01$ along with the perturbation map is shown in figure 6.8.

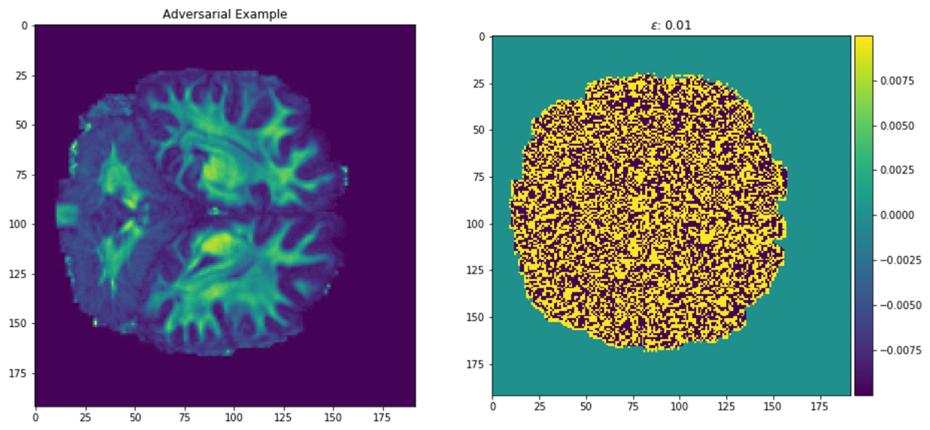

Figure. 6.8. Coronal slice of FA with adversarial perturbation $\epsilon = 0.01$. The adversarial perturbation map adds or subtracts the value of ϵ on a per-voxel basis from the original FA map based on the gradients calculated through equation 6.2.1 to create $\hat{X}$.

Perfect adversarial perturbations are augmentation conditions that one would not expect to find in "in the wild". However, machine learning systems are vulnerable to such conditions and their performance in these situations is of interest. We apply

$\epsilon = 0.01$, 0.03, and 0.05 and evaluate the performance of the ensemble under these conditions. The PMDSC and PE histograms for these conditions are shown below.

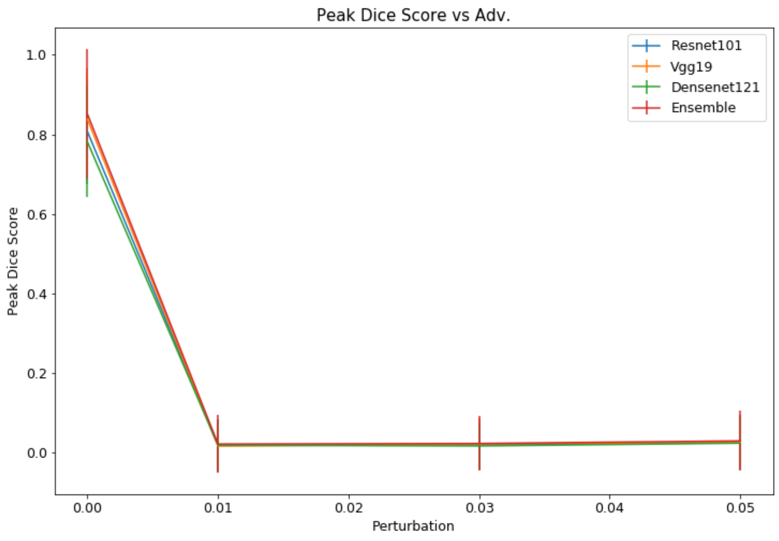

Figure. 6.9. Peak Mean Dice Score (PMDSC) vs increasing test-time ϵ via FGSM. $\rho = -0.20$. 2σ confidence intervals indicate the variability in Dice score over all samples in the test set at the threshold that obtains the highest mean segmentation accuracy.

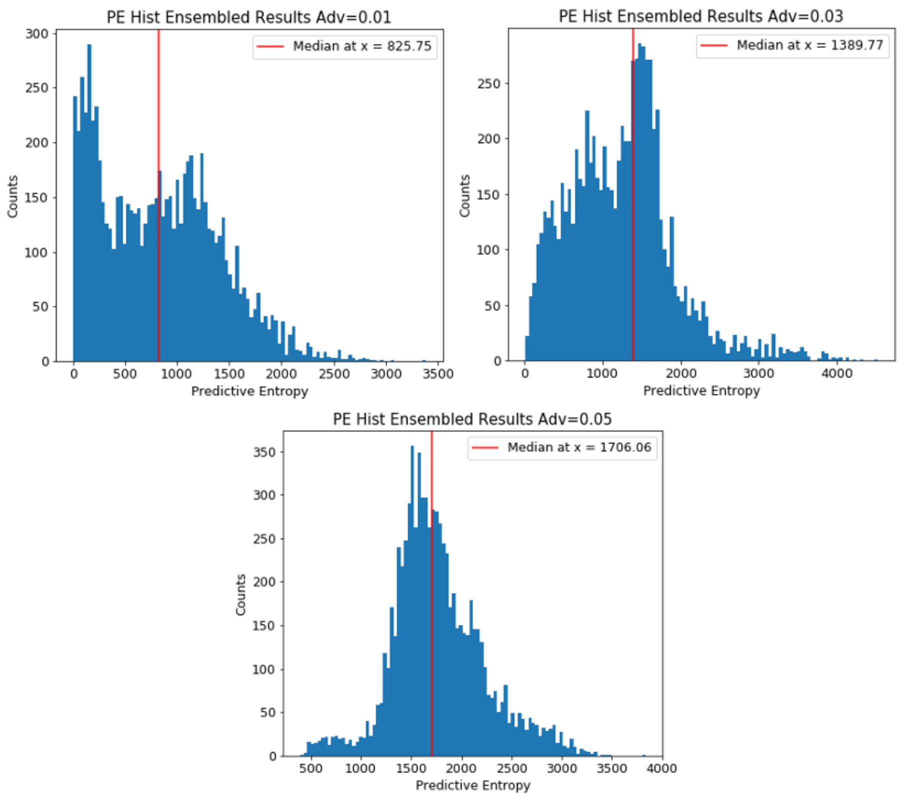

Figure. 6.10. Histograms of the non-normalized predictive entropy (PE) with increasing ϵ for the ensembled model. x and y axes are the PE and number of counts, respectively. As the distributions are broad, the median is reported as it is more robust to outliers.

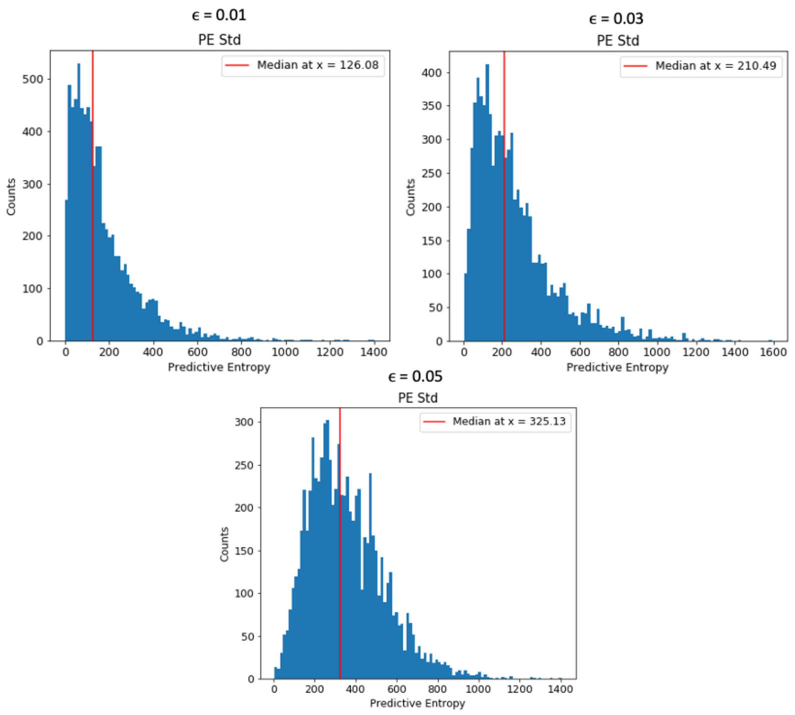

Figure. 6.11. Histograms of the non-normalized predictive entropy variability (PE-Std) with increasing ϵ for the ensembled model. x and y axes are the PE and number of counts, respectively. PE Std values are determined by taking standard deviation of the PE values produced from the Resnet, Vgg, and Densenet predictions for each image in the test set. As the distributions are broad, the median is reported as it is more robust to outliers.

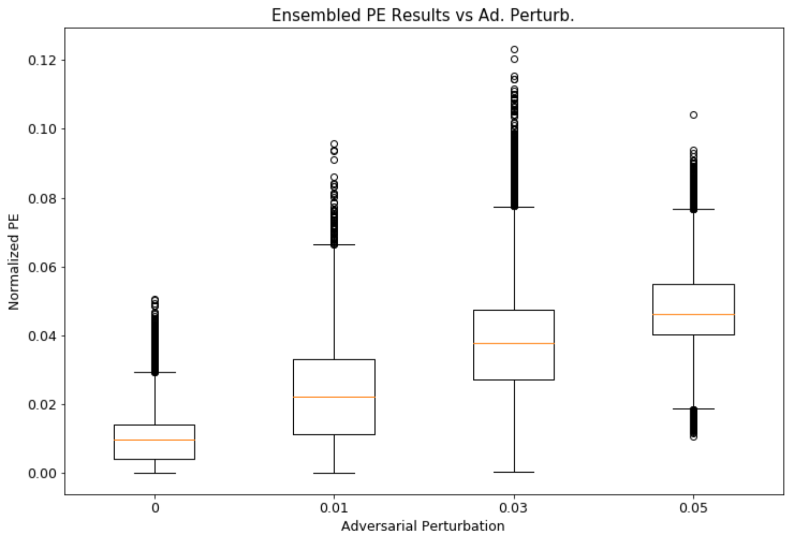

Figure. 6.12. Normalized PE-ϵ whisker plot summarizing median values with boxes extending to the upper and lower quartiles of the data. Whiskers extend to show the data range. Outliers are shown as points. $\rho = 1.0$

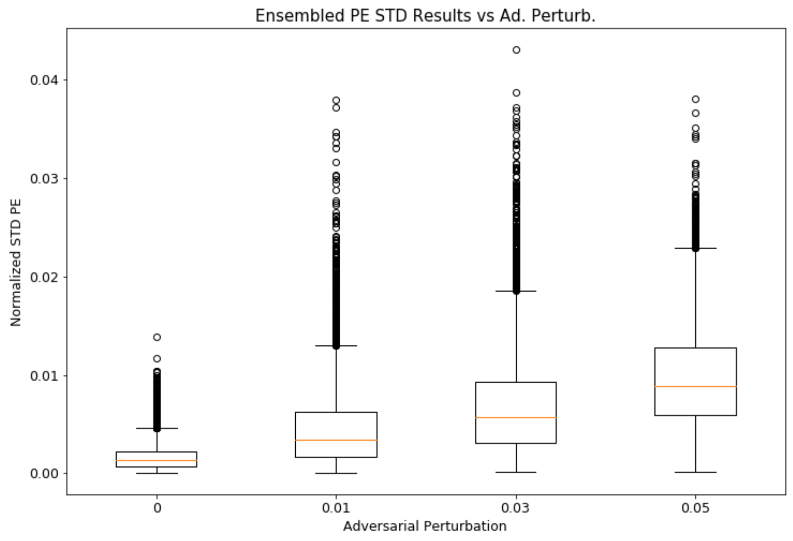

Figure. 6.13. Normalized PE-ϵ Std whisker plot summarizing median values with boxes extending to the upper and lower quartiles of the data. Whiskers extend to show the data range. Outliers are shown as points. $\rho = 1.0$

ϵ	Peak Mean DSC (Std)	Median PE [Norm, Non-Norm]	Normalized Median PE [Lower, Upper Quartile]	Median PE Std [Norm, Non-Norm]	Normalized Median PE Std [Lower, Upper Quartile]
0.01	0.0216(0.0730)*	0.0224, 825.75*	0.0112, 0.0333	0.0032, 126.08*	0.00168, 0.00620
0.03	0.0230(0.0695)*	0.0377, 1389.77*	0.0273, 0.0475	0.00571, 210.49*	0.00304, 0.00927
0.05	0.0295(0.0761)*	0.0463, 1706.80*	0.0404, 0.0549	0.00822, 325.14*	0.00594, 0.0127

Table 6.4: Ensemble segmentation performance and predictive entropy with increasing ϵ. *: difference relative to baseline is statistically different (one-sample t-test, $p < 0.05/N$, $N = 7860$). Image normalization [192x192].

As seen in figure 6.9, the PMDSC drastically decreases with small ϵ, leading to a correlation of $\rho = -0.20$. Since adversarial examples were not used during training, it comes of no surprise that the models are not robust to these during test-time. While the slight increase in PMDSC with increasing ϵ was not expected, obtaining 2σ confidence intervals at each ϵ shows that we include zero, leading to the assumption that the model performs equally poor at all ϵ. Interestingly, even with similar test-time performance, both the PE and PE-Std increase along with increasing ϵ, each with a perfect Spearman rank correlation, $\rho = 1$. This is consistent with our hypothesis that the model should become uncertain with OOD samples, but it is important to see increasing PE and PE-Std in the absence of continually decreasing PMDSC. This observation is in contrast to the test-time PE distortion trend shown in figure 6.7, where PE does not clearly increase with increasing distortion.

With these observations, inferring the behaviour of one metric based on the other is challenging. However, as seen in figures 6.7 and 6.13, the variability in model PE is apparent regardless of the test-time augmentation that occurs.

The table below reports the percentage change of the median PE and PE-Std for increasing ϵ.

ϵ	**Relative PE % Change**	**Relative PE-Std % Change**
0	N/A	N/A
0.01	128.1%	161.1%
0.03	283.9%	335.9%
0.05	371.5%	573.3%

Table 6.5: Percentage change of the median PE and PE-Std relative to baseline values for increasing adversarial perturbations. There is both a strong increase in PE and PE-Std relative to the baseline value.

With greater than 100% change relative to baseline values for all ϵ, it is clear that the ensemble is indicating representative uncertainties where there is a drastic decrease in model performance.

6.3 Signal-to-Noise Ratios

The noise associated with any image is an important factor determining if objects or features can be discerned from the background. For the following analysis, we evaluate the ensemble's performance under decreasing SNR by corrupting the signal with random Gaussian distributed noise — a form of in-domain dataset shift. SNR degradation was applied to each image independently and examples of these SNR image slices are shown in figure 6.14.

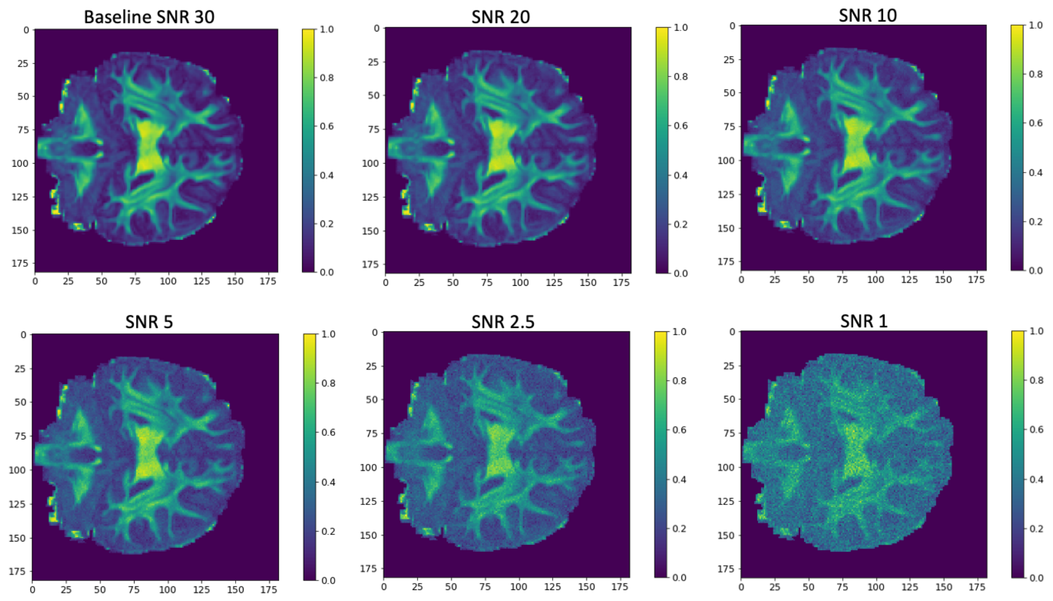

Figure. 6.14. A coronal brain image showing FA corrupted with various levels of random Gaussian distributed noise. For simplicity it was assumed that noise in parametric diffusion images is Gaussian, when it may very well not be.

The estimation of the applied SNR degradation is calculated by first determining the baseline SNR of a T2-weighted image acquired during diffusion imaging (i.e. $b = 0s/mm^2$), since this is the best SNR we can expect to see. Since we are most interested with the signal of WM, we measure the average signal of an ROI within the corpus-callosum, a deep WM structure. We divide this signal by the standard deviation of the noise from an ROI far away from the brain. This is to avoid correlated noise artifacts present in the background as a result of shifting each line in k-space from the center since the gradients rasterize the data collection. 2D slices with ROIs and noise correlation are shown in figure 6.15.

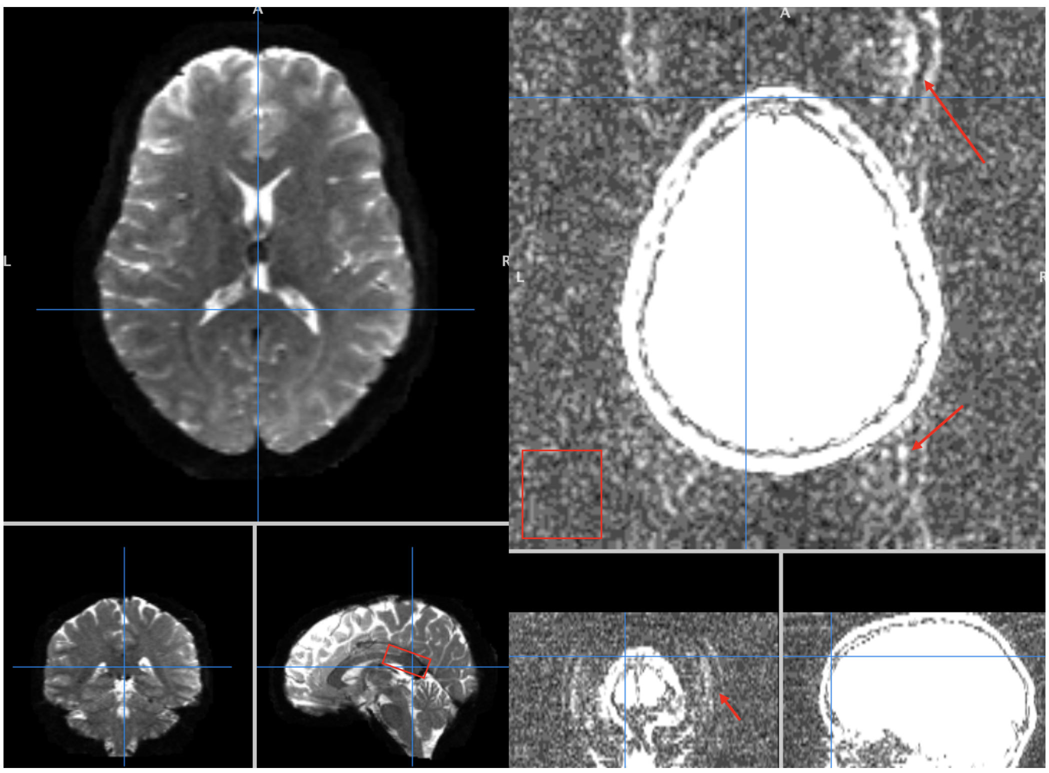

Figure. 6.15. T2-weighted diffusion scan (b-value = $0s/mm^2$) with signal and noise ROIs shown. Brain ROI is the corpus-callosum, a deep WM structure. Regions of the background noise are shown to be correlated with the brain due to an shifted k-space, indicated with red arrows.

When measuring the baseline SNR over all 26 volumes, an SNR variability of 30.4 ± 1.3 was seen. During degradation, we apply noise such that the standard deviation of the additional noise distribution is equal to $\frac{\mu_{BS}}{n}$[1], where μ_{BS} is the mean brain signal within an ROI, and n is the degradation value. Therefore, the SNR with increasing degradation is calculated using the equation below.

$$SNR = \frac{\mu_{BS}}{\frac{\mu_{BS}}{n}} = n \tag{6.3.1}$$

It is important to reiterate that every image is going to have a different mean signal within the chosen WM ROI. Furthermore, since SNR degredation is applied to each slice independently during testing, not all slices through the MR volume are going to contain the corpus-callosum, the ROI used for measuring the baseline SNR.

[1]BS = Brain Signal

As such, the calculation of μ_{BS} for each slice is done by taking the mean signal of all WM in the slice. Therefore, the degradation values reported are only estimates of the true SNR for each image during test-time. An important assumption made is that all slice used during the following analysis have a similar baseline SNR, but there could be some unaccounted for SNR variability acting as a confounding variable during the following analysis.

We choose n = 20, 10, 5, 2.5, and 1 for the analysis. The figures below show the PMDSC and PE histograms for the ensemble while exposed to these SNR conditions.

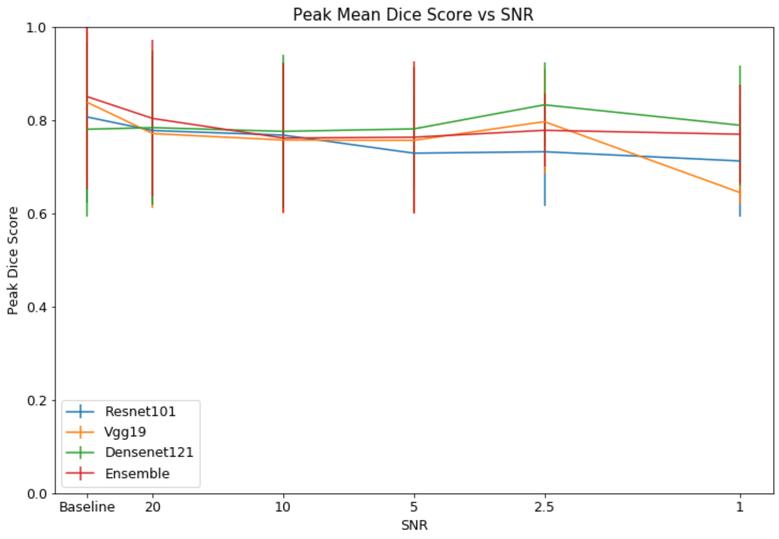

Figure. 6.16. Peak Mean Dice Score (PMDSC) vs decreasing test-time SNR via random Gaussian distributed noise. $\rho_{ensemble} = -0.10$. 2σ confidence intervals indicate the variability in Dice score over all samples in the test set at the threshold that obtains the highest mean segmentation accuracy. As the distributions are broad, the median is reported as it is more robust to outliers.

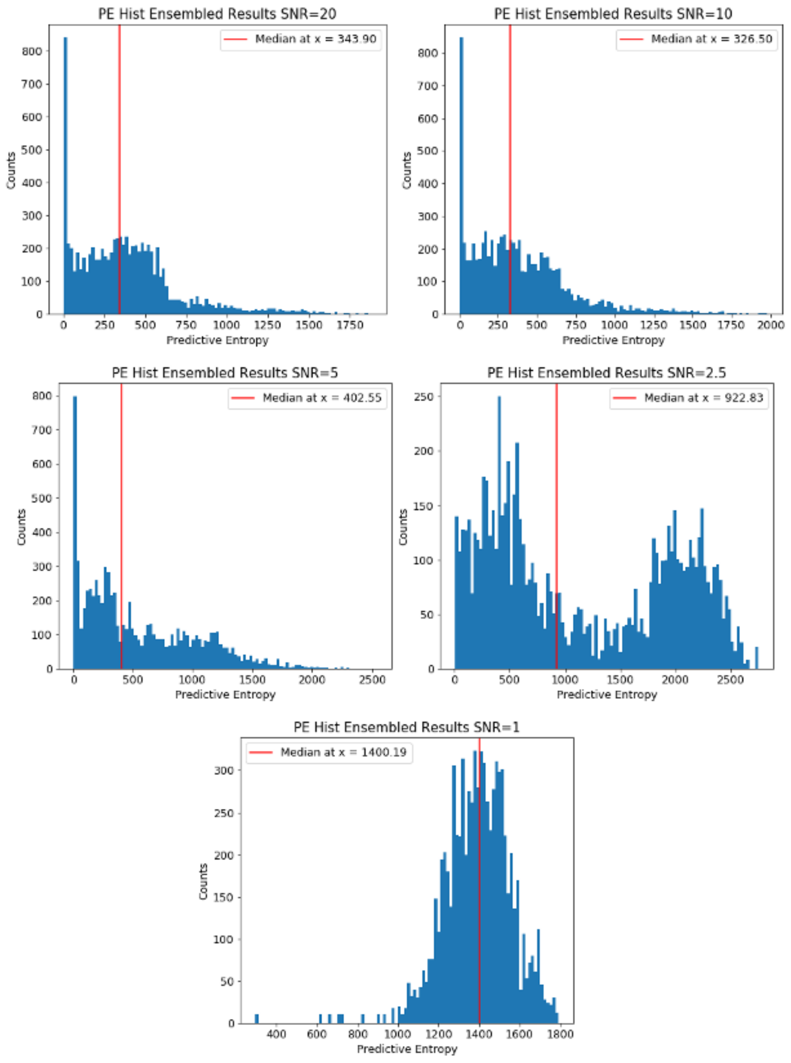

Figure. 6.17. Histograms of the non-normalized predictive entropy (PE) with decreasing SNR for the ensembled model. x and y axes are the PE and number of counts, respectively.

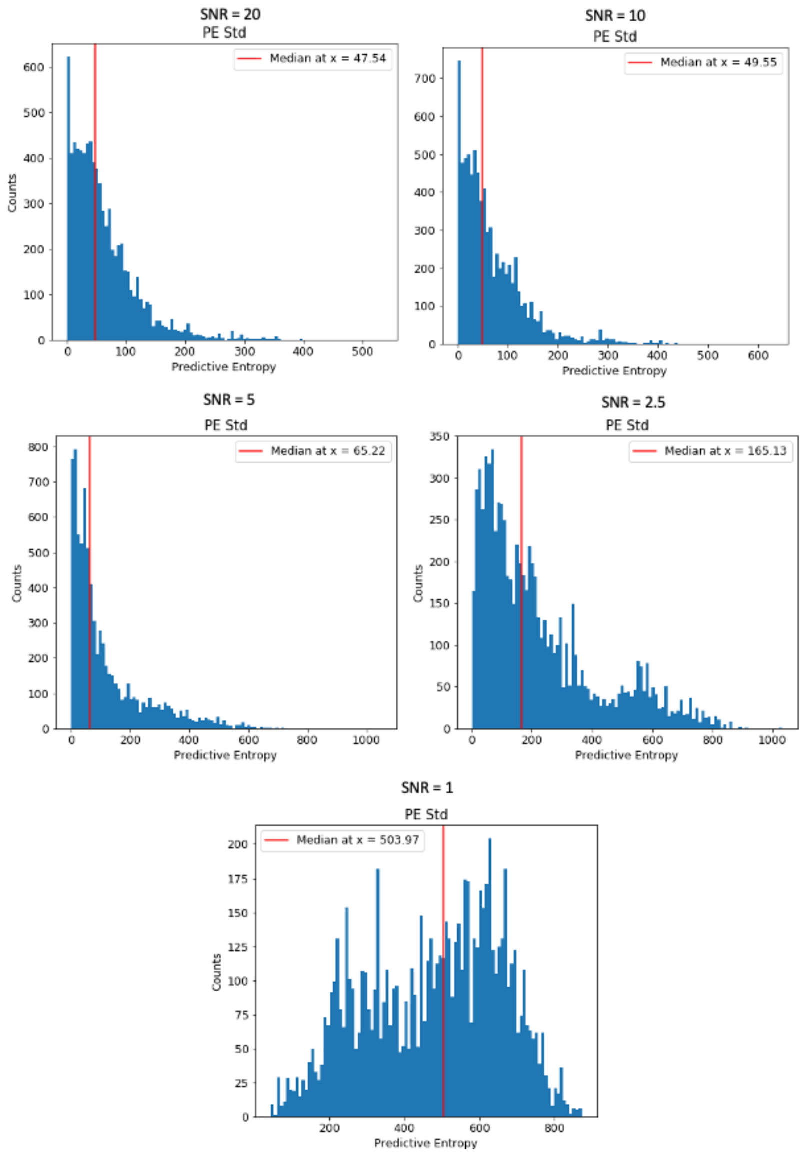

Figure. 6.18. Histograms of the non-normalized predictive entropy variability (PE-Std) with decreasing SNR for the ensembled model. x and y axes are the PE and number of counts, respectively. PE Std values are determined by taking standard deviation of the PE values produced from the Resnet, Vgg, and Densenet predictions for each image in the test set.

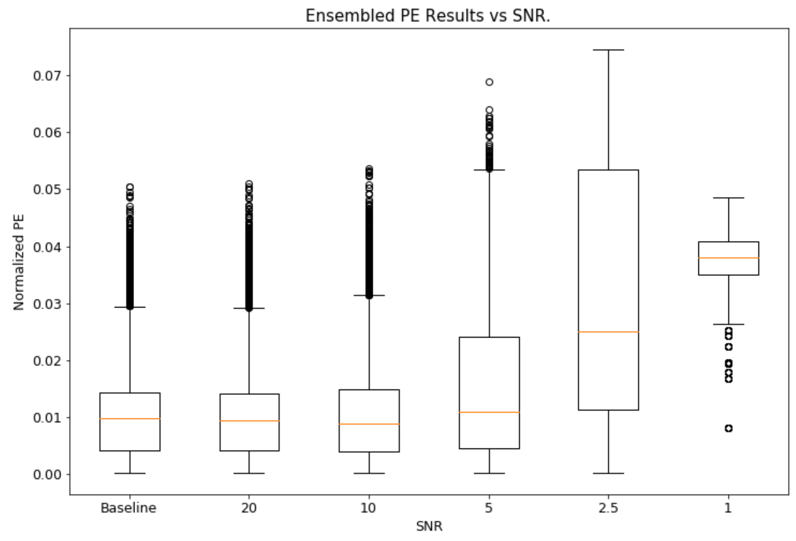

Figure. 6.19. Normalized PE-SNR whisker plot summarizing median values with boxes extending to the upper and lower quartiles of the data. Whiskers extend to show the data range. Outliers are shown as points. $\rho = 0.9$

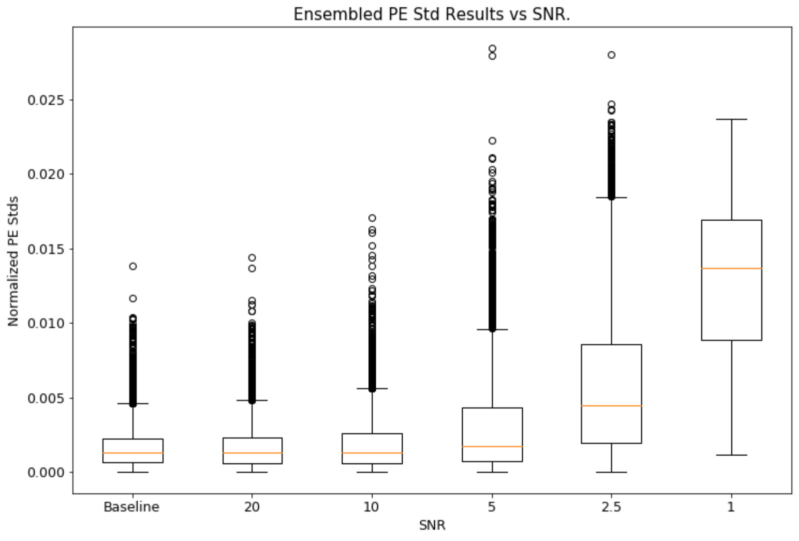

Figure. 6.20. Normalized PE-SNR Std whisker plot summarizing median values with boxes extending to the upper and lower quartiles of the data. Whiskers extend to show the data range. Outliers are shown as points. $\rho = 1.0$

SNR	Peak Mean DSC (Std)	Median PE [Norm, Non-Norm]	Normalized Median PE [Lower, Upper Quartile]	Median PE Std [Norm, Non-Norm]	Normalized Median PE Std [Lower, Upper Quartile]
20	0.804(0.138)*	0.00933, 343.57*	0.00401, 0.0141	0.001289, 47.54*	0.00169, 0.00621
10	0.762(0.113)*	0.008857, 326.50*	0.00388, 0.0149	0.001344, 49.55*	0.00304, 0.00927
5	0.764(0.136)*	0.0109, 401.82*	0.00452, 0.0241	0.001769, 65.22*	0.00594, 0.0127
2.5	0.779(0.0526)*	0.0250, 921.60*	0.0112, 0.0535	0.00447, 165.13*	0.00197, 0.00858
1	0.770(0.0898)*	0.0380, 1400.09*	0.0350, 0.0408	0.0137, 503.97*	0.00890, 0.0169

Table 6.6: Ensemble segmentation performance and predictive entropy with decreasing SNR. *: difference relative to baseline is statistically different (one-sample t-test, $p < 0.05/N$, $N = 7860$). Image normalization [192x192].

As seen in Figure 6.16, the PMDSC shows a weak relationship with decreasing SNR ($\rho^2_{ensemble}$ = -0.10), which was not expected. At an SNR of 1, we only observe a decrease in ensemble performance of 9.4% relative to the baseline PMDSC.

Seen in Figure 6.17, the PE distribution is relatively unaffected until an SNR of 2.5 where we observe a bi-modal behaviour into a complete distribution shift at an SNR of 1. As such, the median PE value does not clearly increase with decreasing SNR until an SNR of 2.5, which was not expected. We observe a similar behaviour in the PE-Std distribution, as seen in Figure 6.20, where the median PE-Std value does not clearly increase until an SNR of 2.5. Both the PE and PE-Std show strong Spearman rank correlations of 0.9 and 1, respectively, with respect to SNR. The table below reports the percentage change of the median PE and PE-Std for decreasing SNR.

SNR	Relative PE % Change	Relative PE-Std % Change
30	N/A	N/A
20	-5.1%	-1.5%
10	-9.8%	2.6%
5	11.0%	35.1%
2.5	154.6%	242.3%
1	286.7%	943.6%

Table 6.7: Percentage change of the median PE and PE-Std relative to baseline values for decreasing SNR.

Similar to previous results, we can obtain useful information regarding the uncertainty of an ensemble prediction irrespective of the PMDSC. In this case, we observe almost uniform test-time performance over the entire SNR range, but continue to discover significant inter-model variability at lower SNR values. These results are promising in that the ensemble could arrive at excellent test-time performance with large SNR degradation while indicating representative uncertainties when exposed to in-domain dataset shift.

Chapter 7

Discussion

In this work, we reported how an uncertainty metric such as PE changes under various test-time conditions. Here, we summarize the key findings and indicate relationships that are not initially clear.

Ensembles should be able to derive improvements in test-time performance compared to a single model and report low or high uncertainty for in-domain and out-of-distribution samples, respectively. As such, the relationships between the PMDSC and the PE should follow the following logic,

PMDSC	**PE, PE-Std**	**Relationship**
High to Low	Low to High	In-domain dataset shift.
High	Low	In-domain samples. Model performing well.
Low	High	OOD samples.
Low	Low	In-distribution samples. Model performing poorly.

Table 7.1: Expected relationships between model performance and PE, PE-Std.

For cases where we know the data are from the training distribution, the model should perform well and the uncertainty reported by the PE or through the PE variability should be low. This is an example of a well trained model. This is most easily seen at the $\epsilon = 0$ adversarial perturbation case, where ensemble performance is high at 0.851, median PE is 0.00982, and median PE-Std is 0.00131, as reported in tables 6.2 and 6.4. If the model is reporting low uncertainty where we observe poor test-time performance, this is an example of a poorly trained model. Where these relationships become complicated are when the model is exposed to either OOD data or data that is similar to the training distribution but has shifted in some way that hinders performance (otherwise known as in-domain dataset shift). In the former case, the model should show both a drastic decrease in performance along with indications of high uncertainty. In the latter, the test-time performance can range from high to low depending on how much the distribution has shifted and the uncertainty should increase with this shift.

While the absolute values of PMDSC, PE, and PE-Std are important for determining an uncertainty threshold, the correlations between these data can also reveal interesting relationships. The empirically determined correlations between the reported uncertainty metrics and test-time performance for each of the augmentation conditions are reported in table 7.2.

Augmentation	**PMDSC–Aug. (ρ)**	**PMDSC–PE (ρ)**	**PMDSC–PE-Std (ρ)**	**PE–PE-Std (ρ)**
Distortion	Strong (-0.94)	Weak (-0.26)	Strong (-0.83)	Weak (0.314)
Adversarial Examples	Weak (-0.20)	Strong (-1)	Strong (-1)	Strong (1)
SNR	Weak (-0.10)	Weak (-0.30)	Weak (-0.10)	Strong (0.90)

Table 7.2: Empirically determined correlations between model performance (PMDSC), the severity of the augmentation condition (Aug.), PE, and PE-Std.

With increasing distortion, the PE and PE-Std distributions the low correlation ($\rho = 0.314$) indicates that these two relationships are weakly related to each other. With the relative changes reported in table 6.3, we observe that the inter-model variability is a much stronger uncertainty metric when compared to PE alone since a larger relative change is seen at smaller distortion values. From this observation, we calculate that the **the PMDSC–PE correlation is much weaker than the PMDSC–PE-Std correlation ($\rho = -0.26$ and $\rho = -0.83$, respectively)**. Furthermore, distortion was the only test-time augmentation condition where a progressive decrease in the PMDSC was observed, but this behaviour was only expected from samples generated from a shifted distribution. As we have interpreted the distortion data as OOD, we hypothesize that this trend arises due to potentially similar augmentations during training-time. With the inclusion of random affine scaling and rotations during training, these operations could potentially mimic the random perspective transformation obtained through less-severe distortions (0.10, 0.30). As distortion becomes prominent (> 0.50), the test-time data becomes characteristically OOD, but smaller distortions seem to consist of shifted in-domain data, which would explain the linear decrease in PMDSC and slow increase in PE-Std.

With adversarial examples, the **PMDSC–PE and PMDSC–PE-Std correlations are strong, arriving at a $\rho = -1$ for both cases**. This was expected as the PE–PE-Std correlation shows a strong positive relationship of $\rho = 1$. Adversarial examples showcase the expected relationship indicated in table 7.1, where the PMDSC is low for all OOD samples and the associated uncertainty shows a strong correlation with increasing perturbations. Furthermore, adversarial examples arrive at the largest relative percentage changes for the PE or PE-Std distributions along with the largest *absolute* uncertainties (table 6.4, Median PE column) for all augmentation conditions considered, even for small ϵ. As such, the ensemble is clearly sensitive to OOD samples / adversarial attacks, and reports a representative uncertainty trend for these cases. However, it would be desirable for the model to output larger absolute uncertainties (closer to PE = 1) in the OOD cases rather than relying on the relative trend.

Decreasing SNR shows the most interesting relationships out of all of the test-time augmentation conditions. While the underlying signal is characteristically in-domain, the explicit dataset shift does not result in significant reductions in performance. Since the PMDSC over the SNR range is highly uncorrelated with the severity of degradation ($\rho_{ensemble} = -0.10$), **the PE and PE-Std also fail to show any significant correlation with the PMDSC** ($\rho = -0.30$ and $\rho = -0.10$, respectively). The PE–PE-Std correlation is strong ($\rho = 0.90$) even with close to uniform test-time performance over the SNR range. With these results, we hypothesize that the ensemble is fairly robust to in-domain shifts that preserve the geometric integrity of the underlying signal (i.e, the ensemble seems to be robust to noise). Similar to the results discussed previously, the uncertainty clearly increases with increasing shift, which is a positive result for clinical deployment as many MR scans can be corrupted with various degrees of systematic or random noise as figure 6.15 clearly illustrates.

Finally, it is important to clearly illustrate the relationship between model performance, a chosen uncertainty metric, and a chosen test-time augmentation scheme, as this could derive uncertainty threshold(s) and observations not initial clear. As the augmentation condition is easy to quantify, figure 7.1 showcases the PE-SNR relationship with DSC colour coating for the ensemble. Each point is a single image in the test set.

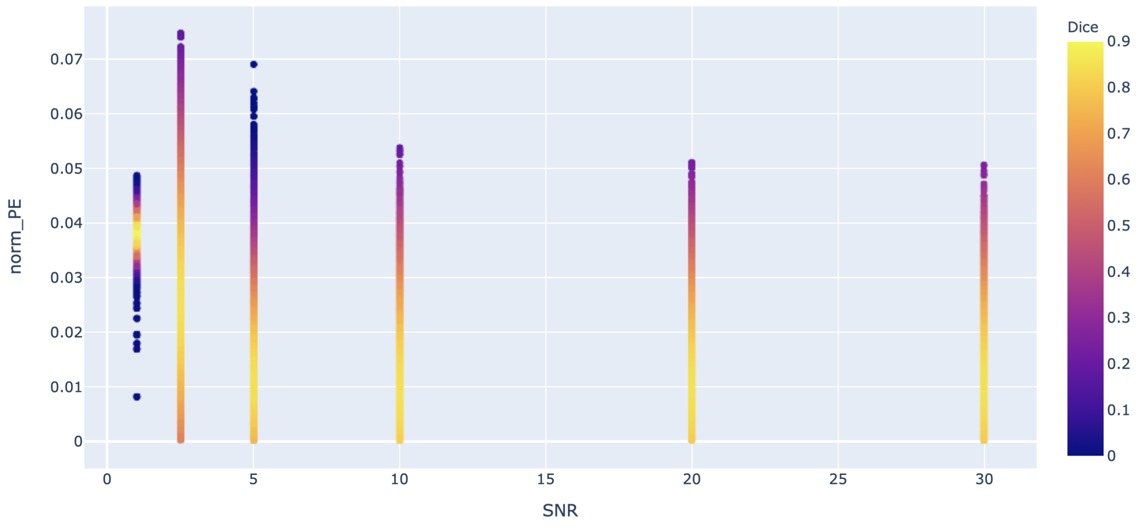

Figure. 7.1. PE vs increasing SNR for each image in the test set with colour coating corresponding to model performance (DSC).

Under SNR 30, 20, and 10, the model performance is high with low PE, indicative of a well calibrated model. Samples with larger normalized PE start to perform poorly. For SNR 5 and 2.5, a similar result is seen but samples with low normalized PE start to showcase poorer performance (moving from yellow to orange). Lastly, an SNR condition of 1 results in a concentration of well performing samples with poor performing samples with lower and higher PE around it. This is indicative of poor model calibration at large augmentation conditions. With these results, an uncertainty threshold could be determined to indicate when predictions should not be trusted for conditions close to the training distribution (SNR 30, 20, 10). As model miscalibration becomes evident during SNR 5, 2.5, and 1 conditions, determining a singular threshold becomes more difficult. However, this might not be the case for all augmentation conditions considered and further work must be done to see if this observation is consistent.

7.1 Summary

The goal of the conducted research was to answer the following questions:

1. Does predictive uncertainty increase with dataset shift and/or OOD samples?
2. Can predictive uncertainty indicate model prediction confidence for a new test-time sample?

PE alone does not seem to indicate strong correlative relationships between model performance and uncertainty for all of the test-time conditions presented (as reported in table 7.2). When compared to the PE-Std, the variability between models often leads to stronger correlations with respect to model performance, indicating that PE-Std could potentially be used as a more reliable estimate for uncertainty. **Though PE *can* increase with increasing dataset shift and/or OOD samples, using an ensemble of well-trained DL models to report test-time PE variability**

seems like a viable method for the detection of OOD or drastically shifted data.

However, the surprisingly stable test-time performance of degrading SNR data brings to question the replicability of such an analysis, as this was a majorly unexpected result. Furthermore, all of the distributions that describe the PE with increasing augmentation are heavily right-skewed, which is clearly evident in all of the whisker plots reported in 6. As we used the median values alone to acquire correlation estimates, the correlations could change in favour of PMDSC–PE if the distributions were to heavily concentrate around the mean. This would be particularly useful to see in distortion cases, where the little to no change in PE seen in figure 6.6 with increasing distortion could be confounded in these broad distributions.

Finally, as previously mentioned, the absolute values of the PE itself are close to zero for all of the test-time augmentation conditions that we considered. This further leads us to the hypothesis that inter-model variability is a more representative metric for uncertainty as it would be sensitive in the low-PE regime, but further testing must be done to confirm this hypothesis. However, even within this low-PE range, an uncertainty decision boundary could be formulated depending on model calibration, as seen in figure 7.1. As this analysis was only completed for decreasing SNR conditions, further work must be done to see if this observation is consistent under other augmentation settings.

Chapter 8

Conclusion

In this thesis, the application of deep ensembles for uncertainty quantification in MR diffusion imaging was explored. We hope this thesis provides a stepping stone and the motivation needed for further work to investigate the use of such a tool for a wider range of clinical imaging tasks. With robust ML models providing uncertainties associated to test-time predictions, we are one step closer to the clinical adoption of the most advanced research in pursuit of improving the health outcomes of those we love.

Future Work

As with any project, the decisions that are made throughout might not always be optimal. For future work that focuses on the detection and segmentation of microstructural WM damage in mTBI patients, the inclusion of multi-channel information to utilize the FA, MD, AD, RD scalar maps together might reveal important relationships that were not accounted for in this thesis. Additionally, 3D model architectures that enforce the localization of signal within space would be highly advantageous to preserve relationships between brain slices.

Of course, the generation of the synthetic mTBI data is biased, where we assumed a localized radiative damage profile within WM regions. Though this was done to reduce the complexity of the data generation process, a deeper literature review focused on how brain injuries manifest in WM changes would be necessary if synthetic data generation were to be applied in the future. Furthermore, since the distribution was modulated through the covariance matrix Σ, pixel variances much larger than the bounds of damage sub-volume could arise in artifacts at the boundary of the box, as there would be non-zero reductions at these boundaries. Ensuring that the distribution has a smooth transition to zero at the boundaries of the sub-volume is a note for future work.

Throughout this thesis, we observed the advantages of utilizing and contrasting an ensemble of deep NNs for prediction. To potentially boost test-time performance, weighting ensemble predictions by the relative model performances could have proven to arrive at more stable test-time predictions. Further, more models could have been used for the ensemble to observe both potential performance improvements and the threshold of diminishing returns.

Finally, we acknowledge that there is no one model that is universally better than another. One interesting observation that is seen within this thesis is that UNet models with different encoding / decoding backbones arrive at similar test-time performances even with a significantly varying number of parameters. Inline with a growing body of literature [77], the hyperparameters that define the training process of the network seem to influence the final result much more significantly than the architecture itself. As significant time was placed into tuning hyperparameters for optimal performance behind the scenes, the work completed here continues to emphasize this point. As such, automating the search for ideal hyperparameters is a clear direction of future work.

Appendix A

Code

A.1 Damage Pre-Processing Code

All Python code is available at mccrinbc's GitHub Repository [78].

A.1.1 main.m

```
close all;
clear all;
%script to determine voxel-wise statistical information for normal dataset.
%Dataset includes Female/Male from 26-30 (267, 222)

%Brains are already normalized at the end of the pipeline because DTI FA
%values only range from 0-1. Any values > 1 are attributed to either noise
%or suboptimal skull stripping, thus they are truncated to 1.

load_brains = true;

nifti_dir = "/Users/brianmccrindle/Documents/Research/Normal_Dataset_FA_26-30";
subdir = ["Female", "Male"];

%load in a WM mask that includes 18 structural regions from MNI-152 space
WM_mask = niftiread("/Users/brianmccrindle/Documents/Research/Normal_Dataset_FA_26-30/show_all_ROIs.nii.gz");
WM_mask = WM_mask > 0; %Make all values 1.

rotateDims = [1,3,2];
WM_mask = permute(WM_mask,rotateDims); %permute to have slices of 182x182x218

if load_brains == false
    %Calculate Voxel-Wise Z-Scoring
    for ii = 1:length(subdir)
        [mean, variance, std] = calculateStats(nifti_dir, subdir(ii), WM_mask, rotateDims); %Call external function
        if subdir(ii) == "Female"
            meanF = mean;
            varianceF = variance;
            stdF = std;
        else
            meanM = mean;
            varianceM = variance;
            stdM = std;
        end
    end
    save('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/meanF.mat','meanF');
    save('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/varianceF.mat','varianceF');
    save('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/stdF.mat','stdF');
    save('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/meanM.mat','meanM');
    save('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/varianceM.mat','varianceM');
    save('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/stdM.mat','stdM');
else

    load('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/meanF.mat','meanF');
    load('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/varianceF.mat','varianceF');
    load('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/stdF.mat','stdF');
    load('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/meanM.mat','meanM');
    load('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/varianceM.mat','varianceM');
    load('/Users/brianmccrindle/Documents/MATLAB/TBI_Research/Z-Score/stdM.mat','stdM');
end

%Here we want to create a system where we apply a set of damage profiles to
%a single brain and have the correpsonding label.
%We have 25 brains per set, so we're applying 40 different damage profiles
%to each brain to have a total of 1000 brain volumes per set:
%    1000 Male
%    1000 Female
brain_locs = ["/Users/brianmccrindle/Documents/Research/Normal_Dataset_FA_26-30/Female_beforeDamage",...
              "/Users/brianmccrindle/Documents/Research/Normal_Dataset_FA_26-30/Male_beforeDamage"];

save_locs = ["/Users/brianmccrindle/Documents/Research/Normal_Dataset_FA_26-30/Female_afterDamage",...
              "/Users/brianmccrindle/Documents/Research/Normal_Dataset_FA_26-30/Male_afterDamage"];

label_locs = ["/Users/brianmccrindle/Documents/Research/Normal_Dataset_FA_26-30/Female_labels",...
              "/Users/brianmccrindle/Documents/Research/Normal_Dataset_FA_26-30/Male_labels"];

%for ii = 1:length(brain_locs) %Loop the 2 datasets
for ii = 1:2

    dirlist = dir(brain_locs(ii));
    brains = dirlist(~ismember({dirlist.name}, {'.','..','.DS_Store'}));

    for j = 1:length(brains) %Loop the data within the folder
        brain = niftiread(fullfile(brain_locs(ii),brains(j).name));
        brain = permute(brain,rotateDims); %182x182x218

        %%%%%%%%%%%%%%%%%%%%%%%%
        %There are values that are > 1. Supress and make them 1.
        [brain_row, brain_col, brain_depth] = ind2sub(size(brain),find(brain > 1));
        for index = 1:length(brain_row) %The same for all row,col,depth.
            brain(brain_row(index), brain_col(index), brain_depth(index)) = 1; %Make these indicies 1.
        end
        %%%%%%%%%%%%%%%%%%%%%%%%
        for k = 1:10 %Loop 10 times to generate 40 damage profiles. Save each volume
            num_blobs = randi([1 5]); %Random number of blobs
            %disp(num_blobs) %Report number of blobs in terminal
            [blob, blob_volume] = createDamageVolume(WM_mask, num_blobs); %return the last blob shape and the set.

            blob_volume_onlyWM = blob_volume.*WM_mask; %Remove values that are NOT on WM.

            brain_onlyBlobs = blob_volume_onlyWM.*brain; %Apply Damage to WM blob locations
            [row, col, depth] = ind2sub(size(brain_onlyBlobs),find(brain_onlyBlobs > 0));

            %order of operations matters! process female dataset first
            if ii == 1
                label = createLabel(row, col, depth, meanF, stdF, brain_onlyBlobs);
                label = filterSmallBlobs(label, 30); %You only need to alter the label where the damage
            elseif ii == 2
                label = createLabel(row, col, depth, meanM, stdM, brain_onlyBlobs);
                label = filterSmallBlobs(label, 30);
            end


            brain_onlyBlobs_filtered = label.*brain_onlyBlobs;
            blob_mask = ~(brain_onlyBlobs_filtered); %mask of all 1s excluding blob locations
            brain_noDamage_missingBlobs = brain.*blob_mask; %brain information, damaged areas are ZERO

            brain_withDamage = brain_noDamage_missingBlobs + brain_onlyBlobs_filtered;

            brain_name = append(brains(j).name(13:regexp(brains(j).name, ['nii'])-2), '_', num2str(k), '.nii');
            niftiwrite(brain_withDamage, fullfile(save_locs(ii), brain_name)); %write brain data

            label_name = append(brains(j).name(13:regexp(brains(j).name, ['nii'])-2), '_', ...
                num2str(k), '_', 'label', '.nii');
            niftiwrite(label, fullfile(label_locs(ii), label_name)); %write label
            disp(append("k-value: ", num2str(k)));

%               figure(1); sliceViewer(brain_onlyBlobs_filtered); title('Brain Only Blobs');
%               figure(2); sliceViewer(brain_noDamage_missingBlobs); title('Brain NO Damange');
%               figure(3); sliceViewer(brain_withDamage); title('Brain with Damage');
%               figure(4); sliceViewer(brain); title('Brain');
%               figure(5); sliceViewer(blob_volume); title('blob volume');
%               figure(7); sliceViewer(label); title('Filtered Label')
        end
    end
end
```

A.1.2 calculateStats.m

```
function [mean, variance, std] = calculateStats(nifti_dir, subdir, WM_mask, rotateDims)
%Function to create the voxel-wise statistical mapping from normal dataset.
%nifti_dir: directory of nifti folders
%subdir: sub directory
%WM_mask: Anatomical binary mask of WM locations in brain
%rotateDims = [1,2,3] dimentions to permute the volume

    folder = fullfile(nifti_dir, subdir);
    dirlist = dir(folder);
    dirlist = dirlist(~ismember({dirlist.name}, {'.','..','.DS_Store'})); %remove the useless hidden folders

    %Create the statistical voxel-wise map that the damaged brain is going to
    %be compared to.
    brain_volume_mean = zeros(size(WM_mask)); %All volumes are of the same size, so using the WM_mask is fine.
    brain_volume_var = zeros(size(WM_mask));

    disp("Calculating Statistics")

    for ii = 1:length(dirlist) %loop through all of the nifti file names
        %Calculating the variance and mean in a single pass is adventageous for
        %computation. We use Bessel's Correction for an unbiased estimate of
        %the population variance from a finite sample of n-observations.

        brain = niftiread(fullfile(folder,dirlist(ii).name)); %Load in the brain

        brain = permute(brain,rotateDims);

        brain_volume_mean = brain + brain_volume_mean;
        brain_volume_var = brain.^2 + brain_volume_var;
    end

    %Normal Dataset Z-Scoring Values!
    mean = brain_volume_mean ./ ii;
    variance = (brain_volume_var ./ ii - (brain_volume_mean./ii).^2) ./(ii / (ii - 1)); %Bessel's Correction
    std = sqrt(variance);

end
```

A.1.3 createDamageVolume.m

```
function [blob, blob_volume] = createDamageVolume(WM_mask, num_blobs)
    %Function to create a set of 3D Gaussian damage profiles to be applied to a
    %single brain FA map.
    % WM_mask: Anatomical WM Mask from MNI
    % A: Amplitude of the damage
    % std: standard deviation of the Gaussian
    % num_blobs: number of Gaussian profiles within 3D volume

    %Steps:
    % 1. Select, at random, a single pixel value that is 1 in the mask.
    % 2. Make this pixel the center of the Gaussian
    % 3. Apply 3D Gaussian to mask of 1s (decreasing pixel values)

    %Side note:
    %The amount of damage that would be considered to be OOD is 2*sigma. We
    %know that sigma values tend to be on the order of only ~10% of the true FA
    %value, so anything significantly larger than this could be severe injury
    %or non-realistic for mTBI cases. Therefore, we can apply a conservative
    %damage dynamic range and apply amplitudes [0.2:0.05:0.5] (0.5 being more
    %severe)

    %rng(0,'twister');

    volume_size_min = 20;
```

```
volume_size_max = 30;
std_min = 500; %This would be in pixel values (spatial std)
std_max = 1000;

[WM_row, WM_col, WM_depth] = ind2sub(size(WM_mask),find(WM_mask == 1));
blob_volume = zeros(size(WM_mask)); %hold all of the damage blobs

for ii = 1:num_blobs
    %A = amp_min+(amp_max-amp_min)*rand(1,1); %Random amplitude
    A = 0.70; %Mike suggests just applying a single aplitude, but we can change the std to modify the variation

    std_x = std_min+(std_max-std_min)*rand(1,1);
    std_y = std_min+(std_max-std_min)*rand(1,1);
    std_z = std_min+(std_max-std_min)*rand(1,1);

    %Pick a random index within the WM mask to center the damage blob
    random_index = randi(length(WM_row)); %length(row) == length(col) == ...
    volume_size = ceil(volume_size_min+(volume_size_max-volume_size_min)*rand(1,1)); %same dimention in all directic

    %under the assumption that these conditions do not overlap with
    %each other.

    volume_size_x = 0;
    volume_size_y = 0;
    volume_size_z = 0;
    %We need this just in case that the volume extends out of the true
    %brain volume
    if (WM_row(random_index) - volume_size) < 1
        volume_size_x = WM_row(random_index) - 1; %The dims of the volume is equal to the pixel value

    elseif (WM_row(random_index) + volume_size) > 182
        volume_size_x = 182 - WM_row(random_index);

    elseif (WM_col(random_index) - volume_size) < 1
        volume_size_y = WM_col(random_index) - 1; %The dims of the volume is equal to the pixel value

    elseif (WM_col(random_index) + volume_size) > 182
        volume_size_y = 182 - WM_col(random_index);

    elseif (WM_depth(random_index) - volume_size) < 1
        volume_size_z = WM_depth(random_index) - 2; %The dims of the volume is equal to the pixel value

    elseif (WM_depth(random_index) + volume_size) > 218
        volume_size_z = 218 - WM_depth(random_index);
    end

    %disp(append('volume size: ', num2str(volume_size)))
    %disp([volume_size_x, volume_size_y, volume_size_z])

    %If any of these are not zero, then we have a volume that extends
    %outside of the WM_mask. Make the size of the volume to the size of
    %the minimum dimention.
    %We use MAX here because usually [0, 0, num > 0]. Therefore, this
    %can totally break, but using this for now.
    if volume_size_x | volume_size_y | volume_size_z
        volume_size = max([volume_size_x, volume_size_y, volume_size_z]);
    end
```

```
        %Apply the blob into the volume
        disp([WM_row(random_index) - volume_size, WM_row(random_index) + volume_size])
        disp([WM_col(random_index) - volume_size, WM_col(random_index) + volume_size])
        disp([WM_depth(random_index) - volume_size, WM_depth(random_index) + volume_size])


        %Creating the 3D Gaussian distribution.
        %Large std values are used to not have the damage concentrate to single
        %pixels.
        mu = [0, 0, 0];
        Sigma = [std_x 1 1; 1 std_y 1; 1 1 std_z];

        %Create damage subvolume
        x1 = -volume_size:1:volume_size;
        x2 = -volume_size:1:volume_size;
        x3 = -volume_size:1:volume_size;

        [X1,X2,X3] = meshgrid(x1, x2, x3);
        X = [X1(:), X2(:), X3(:)];

        blob = mvnpdf(X,mu,Sigma); %Create multivariate probability dist.
        blob = reshape(blob,length(x1),length(x2),length(x3))./max(blob);
        blob = 1 - blob + A; %Invert the values (lower values mean damage), add scalar to modulate damage.

        blob_volume(WM_row(random_index) - volume_size : WM_row(random_index) + volume_size,...
            WM_col(random_index) - volume_size : WM_col(random_index) + volume_size, ...
            WM_depth(random_index) - volume_size: WM_depth(random_index) + volume_size) = blob;

    end

    %any values that would be increasing the damage, make 1.
    [blob_row, blob_col, blob_depth] = ind2sub(size(blob_volume),find(blob_volume > 1));
    for j = 1:length(blob_row)
        blob_volume(blob_row(j), blob_col(j), blob_depth(j)) = 1; %Make these indicies 1.
    end

end
```

A.1.4 createLabel.m

```
function label = createLabel(row, col, depth, mean, std, brain)
%Function to create the OOD labels based on mean, std, and brain given.
%row, col, and depth indicate the voxels that have undergone damage from
%the synthetic process.

%mean, std are volume sof equal size of brain.

label = zeros(size(brain));
for ii = 1:length(row) %This will be the same for row, col, and depth

    [x,y,z] = deal(row(ii), col(ii), depth(ii)); %make things easier to see. declare multiple variables

    if or(brain(x,y,z) > mean(x,y,z) + 2*std(x,y,x), brain(x,y,z) < mean(x,y,z) - 2*std(x,y,x))
        label(x,y,z) = 1;
    end
end
```

A.1.5 filterSmallBlobs.m

```
function label = filterSmallBlobs(label, filterSize)
%filter the blobs that are too small for the neural network on a per slice
%basis.
%filterSize = integer value

    [~,~,depth] = size(label);
    for jj = 1:depth
        slice = label(:,:,jj);

        %looking for all of the connected componants, get centroids
        CC = bwconncomp(slice); %using the image processing toolbox

        if ~isempty(CC.PixelIdxList) %if there is nothing in the image slice, skip
            for z = 1:length(CC.PixelIdxList)
                if length(CC.PixelIdxList{z}) < filterSize
                    slice(CC.PixelIdxList{z}) = 0;
                    label(:,:,jj) = slice; %replace the label slice with the filtered slice
                end
            end
        end

    end

end
```

A.1.6 niirearrangeandTif.py

```
import nibabel as nib
import numpy as np

import argparse
import os
from glob import glob as glob
from PIL import Image
import sys

'''
This is a utilitiy script to convert your nii(.gz) files into tif.
We specify a rotational aspect of the script and defaults to switching the volume into 1,0,2.
Our data is always received in [182,218,182], so we rearrange the slices. This does not matter
for processing purposes, but changes the axial slices to be the correct orientation.

'''

def arg_parser():
    parser = argparse.ArgumentParser(description='Re-Arrange_Dimentions_of_3D')
    parser.add_argument('img_dir', type=str,
                        help='path_to_nifti_image_directory')
    parser.add_argument('out_dir', type=str,
                        help='path_to_output')
    parser.add_argument('-dims', '--dims', type=str, default='1,0,2', #currently dims does nothing
                        help='dimentions_to_change_into')
    parser.add_argument('-split', type = int, default = 1)
    parser.add_argument('-a', '--axis', type=int, default=0, #This is the argument to us. --axis 1
                        help='axis_of_the_3d_image_array_on_which_to_sample_the_slices')
    parser.add_argument('-p', '--pct-range', nargs=2, type=float, default=(0.2,0.8),
                        help=('range_of_indices,_as_a_percentage,_from_which_to_sample_'
                              'in_each_3d_image_volume._used_to_avoid_creating_blank_tif_'
                              'images_if_there_is_substantial_empty_space_along_the_ends_'
                              'of_the_chosen_axis'))
    return parser

def split_filename(filepath):
    path = os.path.dirname(filepath)
    filename = os.path.basename(filepath)
    base, ext = os.path.splitext(filename)
    if ext == '.gz':
        base, ext2 = os.path.splitext(base)
        ext = ext2 + ext
    return path, base, ext

def nii_to_tif(img,base,ext,args):
    if img.ndim != 3:
        print(f'Only_3D_data_supported._File_{base}{ext}_has_dimension_{img.ndim}._Skipping.')
    start = int(args.pct_range[0] * img.shape[args.axis])
    end = int(args.pct_range[1] * img.shape[args.axis])
    for i in range(start, end):
        I = Image.fromarray(img[i,:,:], mode='F') if args.axis == 0 else \
            Image.fromarray(img[:,i,:], mode='F') if args.axis == 1 else \
            Image.fromarray(img[:,:,i], mode='F')
        I.save(os.path.join(args.out_dir, f'{base}_{i:04}.tif'))
    return 0

def main():
    try:
        args = arg_parser().parse_args()
        filenames = glob(os.path.join(args.img_dir,'*.nii*'))

        if args.dims != None:
            dims = list(args.dims.split(','))
        else:
            dims = ['0','1','2'] #If we don't want to rotate, then do nothing

        for f in filenames:
            _, base, ext = split_filename(f)
            img = nib.load(f).get_fdata().astype(np.float32).squeeze()
            #img = np.moveaxis(img,[0,1,2],[int(dims[0]),int(dims[1]),int(dims[2])])

            if args.split == 1:
                nii_to_tif(img,base,ext,args)

    except Exception as e:
        print(e)
        return 1
```

```
if __name__ == "__main__":
    sys.exit(main())
```

A.1.7 smp_main.py

```
import os
import random
import numpy as np
from PIL import Image

from datetime import datetime
import pickle
from tqdm import tqdm #loading bar

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data import Dataset as ParentDataset

import torchvision
from torchvision import transforms
from torchvision.transforms import Compose

import sklearn.metrics

import segmentation_models_pytorch as smp #model we're using for now.
import evaluateModel

from matplotlib import pyplot as plt

import batch_testing_script #user defined.
import image_indicies

#Errors associated to potantial randomness / non-deterministic behaviour is a VERY common issue in PT.
#Look at the following github discussion for more information:
#https://github.com/pytorch/pytorch/issues/7068
#       sbelharbi commented on Apr 19, 2019

seed = 1010
#torch.manual_seed(seed)
#torch.cuda.manual_seed(seed)
#torch.cuda.manual_seed_all(seed)  # if you are using multi-GPU.
np.random.seed(seed)  # Numpy module.
random.seed(seed)  # Python random module.
#torch.backends.cudnn.benchmark = False
#torch.backends.cudnn.deterministic = True



class TBI_dataset(ParentDataset): #Obtain the attributes of ParentDataset from torch.utils.data
#Finds Image and Label locations, creates random list of indicies for training / val / testing sets to be called
    def __init__(
        self,
        images_dir,
        labels_dir,
        subset="train",
        transform = None, #base transformation is into Tensor.
        mapping = None #there is no mapping initially. model should stop if no mapping provided.
        #seed=seed, #We'll get the same thing everytime if we keep using the same seed.
    ):

        #filter and sort the list
        self.ImageIds = sorted(list(filter(('.DS_Store').__ne__,os.listdir(images_dir))))
        self.LabelIds = sorted(list(filter(('.DS_Store').__ne__,os.listdir(labels_dir))))

        self.images_fps = [os.path.join(images_dir, image_id) for image_id in self.ImageIds] #full_paths to slices
        self.labels_fps = [os.path.join(labels_dir, image_id) for image_id in self.LabelIds] #full_paths to labels

        if mapping == None:
            print("Mapping Required for Model to Run!")

        #We define a mapping to use when calling the Dataset loader based on the parameter "mapping"
        if subset == "train":
            self.mapping = mapping['train']['set']
            print(self.mapping)
        elif subset == "val":
            self.mapping = mapping['val']['set']
            print(self.mapping)
        elif subset == "test":
            self.mapping = mapping['test']['set']
            print(self.mapping)
        else:
            print("subset parameter requires train, val, or test exactly.")

        self.transform = transform #trasform given by transform_function

    def __getitem__(self, ii): #ii is the index

        #Current implementations of transforms only use PIL images.
        #Apparently we can use np.array(Image.open(...)) to remove the error that happens each epoch
        image = Image.open(self.images_fps[self.mapping[ii]]) #open as PIL image.
        label = Image.open(self.labels_fps[self.mapping[ii]])

        image = self.transform(image)
        label = self.transform(label)

        return image, label #, self.images_fps[self.mapping[ii]], self.labels_fps[self.mapping[ii]]

    def __len__(self):
        return len(self.mapping)


def datasets(images_dir, labels_dir, train_size, aug_angle, aug_scale, flip_prob, mapping):

    #mapping = return_image_indicies(images_dir,labels_dir, train_size, random_sampling = True)

    train = TBI_dataset(
        images_dir = images_dir,
        labels_dir = labels_dir,
        subset = "train",
        transform = transform_function(degrees=aug_angle, scale=aug_scale, flip_prob=flip_prob),
        mapping = mapping
    )
    valid = TBI_dataset(
        images_dir = images_dir,
        labels_dir = labels_dir,
        subset = ”val”,
        transform = transform_function(degrees=aug_angle, scale=aug_scale, flip_prob=flip_prob),
        mapping = mapping
    )

    test = TBI_dataset(
        images_dir = images_dir,
        labels_dir = labels_dir,
        subset= ”test”,
        transform = transform_function(degrees=0, scale = [1,1], flip_prob = 0), #make sure nothing changes.
        mapping = mapping
    )

    return train, valid, test

def transform_function(degrees,scale,flip_prob):
    transform_list = []

    transform_list.append(transforms.RandomAffine(degrees, scale = scale))
    transform_list.append(transforms.RandomHorizontalFlip(p=flip_prob))
    transform_list.append(transforms.Pad(5)) #all images should be 182x182 before padding.
    transform_list.append(transforms.ToTensor())

    return Compose(transform_list)

def Weights(labels, device):
    #expects an [batch_size,c,n,n] input

    weights = torch.rand(labels.shape) #create a random tensor of weight values.
    weights = weights.to(device) #put everything onto the GPU.

    for batch_num in range(0,labels.shape[0]):
        num_ones = torch.sum(labels[batch_num,0,:,:]);
        resolution = labels.shape[2] * labels.shape[3]
        num_zeros = resolution - num_ones

        #https://discuss.pytorch.org/t/how-to-apply-a-weighted-bce-loss-to-an-imbalanced-dataset-what-will-the-weight-te
        #Weight for the positive class
        pos_weight = num_zeros / resolution #should be close to 1.
        neg_weight = 1 - pos_weight

        #create 1s tensor, put to GPU.
        ones = torch.ones(labels.shape[2],labels.shape[3])
        ones = ones.to(device)

        weights[batch_num,0,:,:] = ones*neg_weight + labels[batch_num,0,:,:]*pos_weight

    #this keeps the clas imbalance in check
    return weights,pos_weight,neg_weight #should be a tensor.

class DiceLoss(nn.Module):
    def __init__(self):
        super(DiceLoss, self).__init__()
        self.smooth = 1.0

    def forward(self, y_pred, y_true):
        assert y_pred.size() == y_true.size()
        y_pred = y_pred[:, 0].contiguous().view(-1)
        y_true = y_true[:, 0].contiguous().view(-1)
        intersection = (y_pred * y_true).sum()
        dsc = (2. * intersection + self.smooth) / (
            y_pred.sum() + y_true.sum() + self.smooth
        )
        return 1. - dsc

def train_validate(train_dataset, valid_dataset, lr):

    earlystop = False

    if torch.cuda.is_available():
        dev ="cuda:2"
    else:
        dev = "cpu"

    dev = torch.device(dev)

    #this might break, remove worker_init_fn = _init_fn(num_workers)) if so
    train_loader = DataLoader(train_dataset, batch_size, shuffle = True, num_workers = num_workers)
    valid_loader = DataLoader(valid_dataset, batch_size, shuffle = True, num_workers = num_workers)

    model.to(dev) #cast the model onto the device
    optimizer = optim.Adam(model.parameters(), lr = lr) #learning rate should change

    loss_function = torch.nn.BCELoss() #this takes in a weighted input
    #loss_function = smp.utils.losses.DiceLoss()
    #loss_function = DiceLoss()
    #metrics = [smp.utils.metrics.IoU(threshold=0.5)]

    loss_train = []
    loss_valid = []

    loss_train_batchset = []
    loss_valid_batchset = []

    epochLoss_train = []
    epochLoss_valid = []

    for epoch in range(EPOCHS):
        image_count = 0
        for phase in ["train","val"]:
            print(lr)
            #This determines which portions of the model will have gradients turned off or on.
            if phase == "train":
                model.train() #put into training mode
                loader = train_loader
            else:
                model.eval() #evaluation mode.
                loader = valid_loader

            lr_flag = True #flag is set to False if LR has changed and reset once we go back into training.
            for ii, data in enumerate(loader):

                brains = data[0] #[batch_size, channels, height, width]
                labels = data[1]

                image_count += len(brains)
                print(epoch, phase, ii, image_count)

                brains,labels = brains.to(dev), labels.to(dev) #put the data onto the device
                predictions, single_class = model(brains) #single class is not a useful output.

                predictions = torch.sigmoid(predictions) #using this so that the output is bounded [0,1]
                single_class = torch.sigmoid(single_class)

                weights = Weights(labels,dev) #generate the weights for each slice in the batch
                loss_function.pos_weight = weights

                #Implementing BCE Loss
                loss = loss_function(predictions, labels) #loss changes here.

                if phase == "train":
                    #employ this so we don't get multiples in the same list.
                    if (loss_valid and ii == 0):
                    #if loss_valid is NOT empty AND it's the first time we see this in the loop
                        #epochLoss_valid.append(loss_valid[-1]) #append the last value in the
                        epochLoss_valid.append(np.mean(loss_valid_batchset))
                        loss_valid_batchset = []


                    model.zero_grad() #recommended way to perform validation
                    #for p in model.parameters(): p.grad = None #This also sets the gradients to zero. better?

                    loss_train.append(loss.item())
                    loss_train_batchset.append(loss.item()) #append to list of losses in the batch

                    loss.backward()
                    optimizer.step()

                    print(f"Phase: {phase}. Epoch: {epoch}. Loss: {loss.item()}")

                else:
                    if (loss_train and ii == 0):#if loss_valid is NOT empty AND it's the first time we see this in the
                        #epochLoss_train.append(loss_train[-1]) #append the last value in the loss_train list.
                        epochLoss_train.append(np.mean(loss_train_batchset))
                        loss_train_batchset = []

                    loss_valid.append(loss.item())
                    loss_valid_batchset.append(loss.item())

                    print(f"Phase: {phase}. Epoch: {epoch}. Loss: {loss.item()}")

                    #learning rate changes and early stopping
                    #This only occurs during validation.
                    if epoch > 0:
```

```
                    if (epoch % 10) == 0: #if the epoch is divisable by 10
                        meanVal = np.mean(loss_valid[epoch - 5 : epoch])
                        print(meanVal, np.abs((meanVal - loss.item()) / meanVal) <= 0.4)
                        if (np.abs((meanVal - loss.item()) / meanVal) <= 0.4 and lr_flag):
                        #if the % difference is small
                        #if epoch == 40 and lr_flag: #doing this for now.
                            lr = lr * 0.1
                            lr_flag = False
                            for param_group in optimizer.param_groups:
                                #print(optimizer)
                                print('Reducing_the_Learning_Rate:_', lr )
                                param_group['lr'] = lr
                                #print(optimizer)

                    if (epoch % 100) == 0:
                        meanVal = np.mean(loss_valid[epoch - 50 : epoch])
                        if np.abs((meanVal - loss.item()) / meanVal) <= 0.05:
                            earlystop = True

                #Implementation of early stopping
                if earlystop == True:
                    date = datetime.now()
                    torch.save(model.state_dict(), os.path.join(os.getcwd(), 'results_TBI_model-' + str(date.date())
                    + '-' + str(date.hour) + '-' + str(date.minute) + '-EARLYSTOP.pt'))
                    #save the model
                    break
            else:
                continue
            break
        else:
            #save the model at the end of this epoch.
            #date = datetime.now()
            #torch.save(model.state_dict(), os.path.join(os.getcwd(), "Registered_Brains_FA/models_saved",
            "TBI_model-epoch"
            + str(epoch) + '-' + str(date.date()) + '-' + str(date.hour) + '-' + str(date.minute) + ".pt"))
            continue
        break

    #Need to add the last element from loss_valid to epochLoss_valid to equal the number of epochs.
    #We could potentially make this value an average of the relevant loss_valids
    #epochLoss_valid.append(loss_valid[-1])
    epochLoss_valid.append(np.mean(loss_valid_batchset))

    print('Final_Learning_Rate:_', lr)
    return brains, labels, predictions, single_class, loss_train,
    loss_valid, epochLoss_train, epochLoss_valid, model.state_dict(), lr

#This function expects singular images.
#Also biased from high class imbalance. function currently not in use.
def IoU(prediction, label):
    #Prediction IoU
    intersection = int(torch.sum(torch.mul(prediction,label)))
    union = int(torch.sum(prediction) + torch.sum(label)) - intersection
    IOU_predicted = intersection / (union + 0.0001) #for stability
    mean_IoU = IOU_predicted

    #Not including background IoU
    #Background IoU
    #all_zeros = (prediction + label) > 0 #before the inversion
    #intersection = int(torch.sum(~all_zeros))
    #union = int(torch.sum(~ (prediction > 0)) + torch.sum(~ (label > 0)) - intersection)
    #IOU_background = intersection / (union + 0.0001)

    #mean_IOU = (IOU_background + IOU_predicted)/2
    return mean_IoU

def testModel(test_dataset, modelPath, threshold): #model = the model class = smp.UNet()

    total_images = 0
    mean_IoUs = []
    loss_set_batch = []
    loss_set = []
    CM_values = [0,0,0,0] #tp, fn, fp, tn
    model.load_state_dict(torch.load(modelPath))

    loss_function = torch.nn.BCELoss() #implementing the loss function to show loss for each threshold.

    if torch.cuda.is_available():
        dev ="cuda:2"
    else:
        dev = "cpu"

    dev = torch.device(dev)
    model.to(dev)
    model.eval() #evaluation mode to turn off the gradients / training.

    #turn shuffle off
    loader = DataLoader(test_dataset, batch_size = 12, shuffle = False, num_workers = num_workers)
    for ii, data in tqdm(enumerate(loader)):

        brains = data[0]
        labels = data[1]

        #move the data to the GPU
        brains = brains.to(dev)
        labels = labels.to(dev)

        total_images += brains.shape[0] #this would be the same if we used labels or predictions.
        #print(total_images)

        predictions, _ = model(brains)
        predictions = torch.sigmoid(predictions)

        weights = Weights(labels, dev)
        loss_function.pos_weight = weights

        #Implementing BCE Loss
        preds = predictions > threshold
        preds = preds.float() #cast the bool tensor into float32 for loss function
        loss = loss_function(preds, labels) #loss changes here.
        loss_set_batch.append(loss.item()) #append the loss of the batch

        predictions_numpy = predictions.cpu().detach().numpy()
        labels_numpy = labels.cpu().detach().numpy()
        for j in range(predictions.shape[0]):
            CM = sklearn.metrics.confusion_matrix(labels_numpy[j,0,:,:].ravel(), predictions_numpy[j,0,:,:].ravel()
            > threshold, labels = [True,False])
            try:
                CM_values[0] = CM_values[0] + CM[0][0]
                CM_values[1] = CM_values[1] + CM[0][1]
                CM_values[2] = CM_values[2] + CM[1][0]
                CM_values[3] = CM_values[3] + CM[1][1]
            except:
                print("Error␣in␣Appending")
                return CM, CM_values

        loss_set.append(np.mean(loss_set_batch)) #append the mean loss of the entire set

    del loader #delete loader, might be wrong to do this
    return np.divide(CM_values , (total_images*(192*192))), np.mean(loss_set)


## Read the tests from batch_testing_script ##
tests = batch_testing_script.report_tests()
batch_results = []

#images_dir = "/home/mccrinbc/Data_Removed_Useless_Slices/normalized_slices"
#labels_dir = "/home/mccrinbc/Data_Removed_Useless_Slices/slice_labels"

images_dir = "/home/mccrinbc/Female_afterDamage_tiffs"
labels_dir = "/home/mccrinbc/Female_labels_tiffs"

#Train size is always the same (for now). Implement a better solution later.
train_size = 0.75
dataset_id = 'Female'

#We need to run the train/val/test indicies split before we go into the for loop.
#For this reason, we've developed a small script to do this sampling for us, and to confirm that it's consistent.
mapping = image_indicies.return_image_indicies(images_dir, labels_dir, train_size, dataset_id, seed,
random_sampling = True)

for ii in tests:
        batch_size = tests[ii]['batch_size']
        EPOCHS = tests[ii]['EPOCHS']
        lr = tests[ii]['lr']
        aug_scale = tests[ii]['aug_scale']
        aug_angle = tests[ii]['aug_angle']
        flip_prob = tests[ii]['flip_prob']
        num_workers = tests[ii]['num_workers']

        #images_dir = "/Users/brianmccrindle/Documents/Research/TBIFinder_Final
        /Registered_Brains_FA/test_slices"
␣␣␣␣␣␣␣␣#labels_dir␣=␣"/Users/brianmccrindle/Documents/Research/TBIFinder_Final
        /Registered_Brains_FA/test_labels"

␣␣␣␣␣␣␣␣#smp␣specific␣variables
␣␣␣␣␣␣␣␣ENCODER␣=␣tests[ii]['ENCODER']
␣␣␣␣␣␣␣␣aux_params=dict(
␣␣␣␣␣␣␣␣␣␣␣␣pooling='avg',
␣␣␣␣␣␣␣␣␣␣␣␣dropout=␣tests[ii]['dropout'],
␣␣␣␣␣␣␣␣␣␣␣␣#activation='softmax2d',
            classes=1,
        )

        #classes = 2 for the softmax transformation.
#       model = getattr(smp, model_arch)
#       setattr(model, 'encoder_name', ENCODER)
#       setattr(model, 'in_channels' , 1)
#       setattr(model, 'classes'     , 1)
#       setattr(model, 'aux_params', aux_params)
        model = smp.Unet(encoder_name = ENCODER, in_channels=1, classes = 1, aux_params = aux_params)

        train_dataset, valid_dataset, test_dataset = datasets(images_dir, labels_dir, train_size, aug_angle, aug_scale,

        #Training Cell
        brains, labels, predictions, single_class, loss_train, loss_valid, epochLoss_train, epochLoss_valid, model_state
        lr_final,  = train_validate(train_dataset, valid_dataset, lr)
        
        date = datetime.now()
        base = "results_TBI_model-End-" + str(date.date()) + '-' + str(date.hour) + '-' + str(date.minute)
        folder_path = r'/home/mccrinbc/' + base
        pkl_name = base + '.pkl'
        model_name = base + '.pt'

        pkl_location = os.path.join(folder_path, pkl_name)

        if not os.path.exists(folder_path):
            os.makedirs(folder_path)

        torch.save(model_state, os.path.join(folder_path, base +".pt"))
        
        # Saving the objects:
        with open(pkl_location, 'wb') as f:  # Python 3: open(..., 'wb')
            pickle.dump([brains, labels, predictions, single_class,
        
            loss_train, loss_valid, epochLoss_train,
            epochLoss_valid, test_dataset], f)

#Look at the train / validation loss
#Possible variables that might cause the validation loss to jump are:
    # - Learning rate is too high in later epochs
    # - Model could be too big?
    # - Batch size could be too small causing loss in generality between epochs.

        plt.figure()
        plt.plot(np.arange(0, EPOCHS, 1), epochLoss_train)
        plt.plot(np.arange(0, EPOCHS, 1), epochLoss_valid)
        plt.ylabel('Per Epoch Loss')
        plt.xlabel('Epoch')
        plt.title('BCELoss vs Epochs. Initial LR = '
        + str(lr) + '. ' + ENCODER + ' : ' + str(EPOCHS))
        plt.legend(['Train Loss: ' + str(epochLoss_train[-1]),
        'Valid Loss: ' + str(epochLoss_valid[-1])],
        loc = "upper right")
        plt.ylim([0,0.6])

        plt.savefig(os.path.join(folder_path, 'Train-Val_Loss.png'))
```

```
␣␣␣␣␣␣␣␣#Create␣names␣for␣data␣to␣be␣stored.
␣␣␣␣␣␣␣␣model_name␣=␣base␣+␣'.pt'
␣␣␣␣␣␣␣␣modelPath␣=␣os.path.join(folder_path,␣model_name)


␣␣␣␣␣␣␣␣#Testing␣Loop.
␣␣␣␣␣␣␣␣del␣model␣#This␣removes␣any␣confliction␣with␣an␣existing␣model␣running␣on␣the␣GPU.
#␣␣␣␣␣␣␣model␣=␣getattr(smp,␣model_arch)
#␣␣␣␣␣␣␣setattr(model,␣'encoder_name',␣ENCODER)
#␣␣␣␣␣␣␣setattr(model,␣'in_channels'␣,␣1)
#␣␣␣␣␣␣␣setattr(model,␣'classes'␣␣␣␣␣,␣1)
#␣␣␣␣␣␣␣setattr(model,␣'aux_params',␣aux_params)
␣␣␣␣␣␣␣␣model␣=␣smp.Unet(encoder_name␣=␣ENCODER,␣in_channels=1,␣classes␣=␣1,␣aux_params␣=␣aux_params)
␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣#read␣pickle
␣␣␣␣␣␣␣␣if␣'pkl_location'␣not␣in␣locals():
␣␣␣␣␣␣␣␣␣␣␣␣folder_path␣=␣'/home/mccrinbc/results_TBI_model-End-2020-10-05-11'
␣␣␣␣␣␣␣␣␣␣␣␣pkl_location␣=␣"/home/mccrinbc/results_TBI_model-End-2020-10-05-11/results_TBI_model-End-2020-10-05-11.pkl"
␣␣␣␣␣␣␣␣print(pkl_location)

␣␣␣␣␣␣␣␣with␣open(pkl_location,'rb')␣as␣f:
␣␣␣␣␣␣␣␣␣␣␣␣brains,␣labels,␣predictions,␣single_class,␣loss_train,␣loss_valid,
␣␣␣␣␣␣␣␣␣␣␣␣epochLoss_train,␣epochLoss_valid,␣test_dataset␣=␣pickle.load(f)

␣␣␣␣␣␣␣␣if␣'modelPath'␣not␣in␣locals():
␣␣␣␣␣␣␣␣␣␣␣␣modelPath␣=␣'input␣something'

␣␣␣␣␣␣␣␣#model␣=␣model.load_state_dict(torch.load(modelPath))␣#see␣if␣this␣works.
␣␣␣␣␣␣␣␣#modelPath␣=␣"/Users/brianmccrindle/Documents/Research
          /TBIFinder_Final/Registered_Brains_FA/models_saved
          /TBI_model-epoch2-2020-08-27-9-55.pt"
␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣thresholds␣=␣np.arange(0,1.05,0.05)␣#skipping␣every␣other␣element,␣[[0.␣␣␣0.05␣0.1␣...␣1]
␣␣␣␣␣␣␣␣#thresholds␣=␣np.arange(0,0.15,0.05)␣#skipping␣every␣other␣element,␣[[0.␣␣␣0.05␣0.1␣...␣1]
␣␣␣␣␣␣␣␣#thresholds␣=␣[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
␣␣␣␣␣␣␣␣TPR_list␣=␣[]␣#This␣is␣also␣known␣as␣RECALL.
␣␣␣␣␣␣␣␣FPR_list␣=␣[]
␣␣␣␣␣␣␣␣IoUs␣=␣[]
␣␣␣␣␣␣␣␣Dice␣=␣[]
␣␣␣␣␣␣␣␣total_error␣=␣[]
␣␣␣␣␣␣␣␣precision␣=␣[]
␣␣␣␣␣␣␣␣BCE_loss_thresh=␣[]

␣␣␣␣␣␣␣␣for␣threshold␣in␣thresholds:
␣␣␣␣␣␣␣␣␣␣␣␣print(threshold)
␣␣␣␣␣␣␣␣␣␣␣␣#test␣the␣model␣to␣capture␣performance.␣Reported␣in␣the
␣␣␣␣␣␣␣␣␣␣␣␣Confusion␣Matrix␣values
␣␣␣␣␣␣␣␣␣␣␣␣CM_values,␣BCE_loss␣=␣testModel(test_dataset,␣modelPath,
␣␣␣␣␣␣␣␣␣␣␣␣threshold)␣#tp,␣fn,␣fp,␣tn,␣[mean_IoUs]
␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣BCE_loss_thresh.append(BCE_loss)
␣␣␣␣␣␣␣␣␣␣␣␣TPR␣=␣CM_values[0]␣/␣(CM_values[0]␣+␣CM_values[1])
␣␣␣␣␣␣␣␣␣␣␣␣FPR␣=␣CM_values[2]␣/␣(CM_values[2]␣+␣CM_values[3])
␣␣␣␣␣␣␣␣␣␣␣␣TPR_list.append(TPR)
␣␣␣␣␣␣␣␣␣␣␣␣FPR_list.append(FPR)
␣␣␣␣␣␣␣␣
```

```
            #IoUs, Dice, Total Error, and Precision.
            IoUs.append(CM_values[0] / (CM_values[0] + CM_values[1] + CM_values[2]))  #IoU = TP / (TP + FN + FP)
            Dice.append(2 * CM_values[0] / (2 * CM_values[0] + CM_values[1] + CM_values[2])) #Dice = 2TP / (2TP + FN + 
            total_error.append(CM_values[1] + CM_values[2]) #Error = FP + FN. Weighted equally for now.
            precision.append(CM_values[0] / (CM_values[0] + CM_values[2]))
            #Precision = TP / (TP + FP)

        TPR_list = np.nan_to_num(TPR_list, nan = 0) #Replace any nans with 0.
        FPR_list = np.nan_to_num(FPR_list, nan = 0) #Replace any nans with 0.
        IoUs = np.nan_to_num(IoUs, nan = 0) #Replace any nans with 0.
        Dice = np.nan_to_num(Dice, nan = 0) #Replace any nans with 0.
        precision = np.nan_to_num(precision, nan = 0) #Replace any nans with 0.



        #Within the same cell. Save the information from testing.
        difference_array = np.array(TPR_list) - (1-np.array(FPR_list))
        best_acc_thresh = thresholds[abs(difference_array).argmin()] #Thresholds is already defined.

        best_IoU_thresh = thresholds[np.where(IoUs == np.max(IoUs))][0]
        best_Dice_thresh = thresholds[np.where(Dice == np.max(Dice))][0]

        results_name = 'test_results.pkl'
        results_location = os.path.join(folder_path, results_name)

        #This is saving the test results into a pkl file
        with open(results_location, 'wb') as f:  # Python 3: open(..., 'wb')
            pickle.dump([ENCODER, EPOCHS, lr_final, TPR_list, FPR_list, precision, thresholds, best_acc_thresh, IoUs,
            Dice, BCE_loss_thresh], f)
        #pickle.dump([TPR_list, FPR_list,
        precision, thresholds, best_acc_thresh, IoUs, Dice], f)
        


        ## Looking at how well the data is doing ##
        difference_array = np.array(TPR_list) - (1-np.array(FPR_list))
        #Utility to Plot Relavent Information
        print(EPOCHS)
        #print('Best IoU Threshold',best_IoU_thresh)
        print('Best Accuracy Threshold:', best_acc_thresh)

        sens = TPR_list[abs(difference_array).argmin()]
        spec = 1 - FPR_list[abs(difference_array).argmin()]

        print('Optimal Accuracy Sens:', sens)
        print('Optimal Accuracy Spec:', spec)
        print('')

        prec_recall_thresh = abs(precision - TPR_list)[0:-2] #remove the last element
        optimal_index = np.where(prec_recall_thresh ==
        np.min(prec_recall_thresh))
        optimal_prec_rec_thresh = thresholds[optimal_index][0]
        prec_thresh = precision[optimal_index][0]
        recall_thresh = TPR_list[optimal_index][0]

        #Here is where we're going to put all of the data into a txt file.
        #base = base name of the folder, pkl, and pt files.
        batch_results.append([base, tests[ii]['ENCODER'],
        EPOCHS, epochLoss_train[-1], epochLoss_valid[-1],
        best_IoU_thresh, np.max(IoUs), best_Dice_thresh, np.max(Dice), optimal_prec_rec_thresh, prec_thresh,
        recall_thresh, best_acc_thresh, sens, spec])


#Outside of the for-loop now.
with open('batch_testing_results.txt', 'w') as f:
    for item in batch_results:
        f.write("%s\n" % item)
```

A.1.8 batch testing script.py

```
def report_tests():

    tests = {}

    tests[1] = {'ENCODER'     : 'resnet101' ,
                'train_size' : 0.75      ,
                'batch_size' : 16        ,
                'EPOCHS'      : 20       ,
                'lr'          : 0.0001   ,
                'aug_angle'   : 20       ,
                'aug_scale'   : [1,1.3]  ,
                'flip_prob'   : 0.3      ,
                'num_workers': 1         ,
                'dropout'     : 0.5
                }

    tests[2] = {'ENCODER'     : 'densenet121' ,
                'train_size' : 0.75      ,
                'batch_size' : 16        ,
                'EPOCHS'      : 30       ,
                'lr'          : 0.0001   ,
                'aug_angle'   : 20       ,
                'aug_scale'   : [1,1.3]  ,
                'flip_prob'   : 0.3      ,
                'num_workers': 1         ,
                'dropout'     : 0.5
                }

    tests[3] = {'ENCODER'     : 'vgg19' ,
                'train_size' : 0.75      ,
                'batch_size' : 16        ,
                'EPOCHS'      : 30       ,
                'lr'          : 0.0001   ,
                'aug_angle'   : 20       ,
                'aug_scale'   : [1,1.3]  ,
                'flip_prob'   : 0.3      ,
                'num_workers': 1         ,
                'dropout'     : 0.5
                }

    return tests
```

Appendix B

Derivations

B.1 Peak Mean Dice Score (PMDSC)

The test-time evaluation of a model is a function of the binary threshold the user decides to employ. As a result, the applied threshold directly effects the trade-off between the precision and recall of the model(s). The ML engineer has an option to either 1) choose a threshold that optimizes the model performance, or 2) choose a threshold that derives the desired location on the precision - recall curve. For the purposes of this thesis, we apply the former. In order to do this, we must evaluate the model performance for each test-time image over a discrete number of thresholds. The performance of the model at any singular threshold is determined by taking the mean dice performance over the entire test set. As such, this derives the *peak mean* of the test-time performance curve. Mathematically,

$$\mathrm{DSC}_\tau = \frac{1}{N}\sum_{i=0}^{N}(\mathrm{DSC}_i|\tau) \tag{B.1.1}$$

where τ is the test-time threshold applied. We apply this for $\tau = 0$ to 1 with 0.05 intervals for the thesis. Throughout the thesis, we assumed that the data for all thresholds are normally distributed.

Figure B.1 illustrates each model's precision - recall curves before and after test-time augmentation of distortion = 0.50. Figure B.2 shows the corresponding model performances as a function of threshold at the applied distortion.

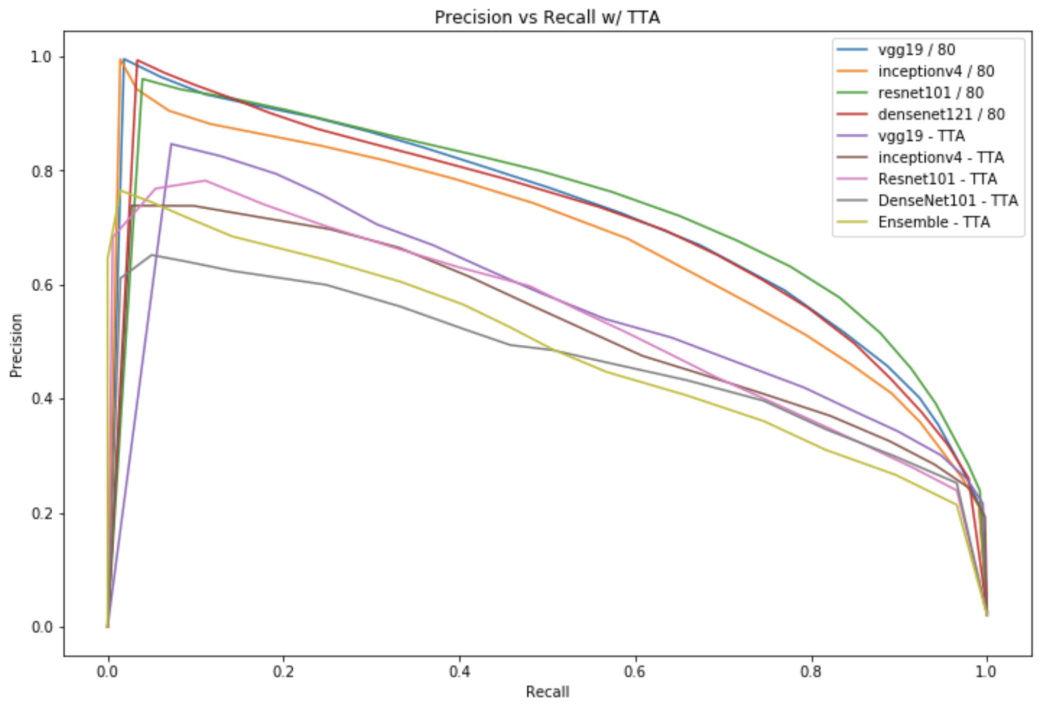

Figure. B.1. Precision vs recall for models used in the thesis for both test-time augmentation (TTA) and non-augmentation. Augmentation applied is distortion at 0.50. Inception model was not used for data analysis. There is a notable reduction in precision-recall curves for the augmentation condition, which is expected.

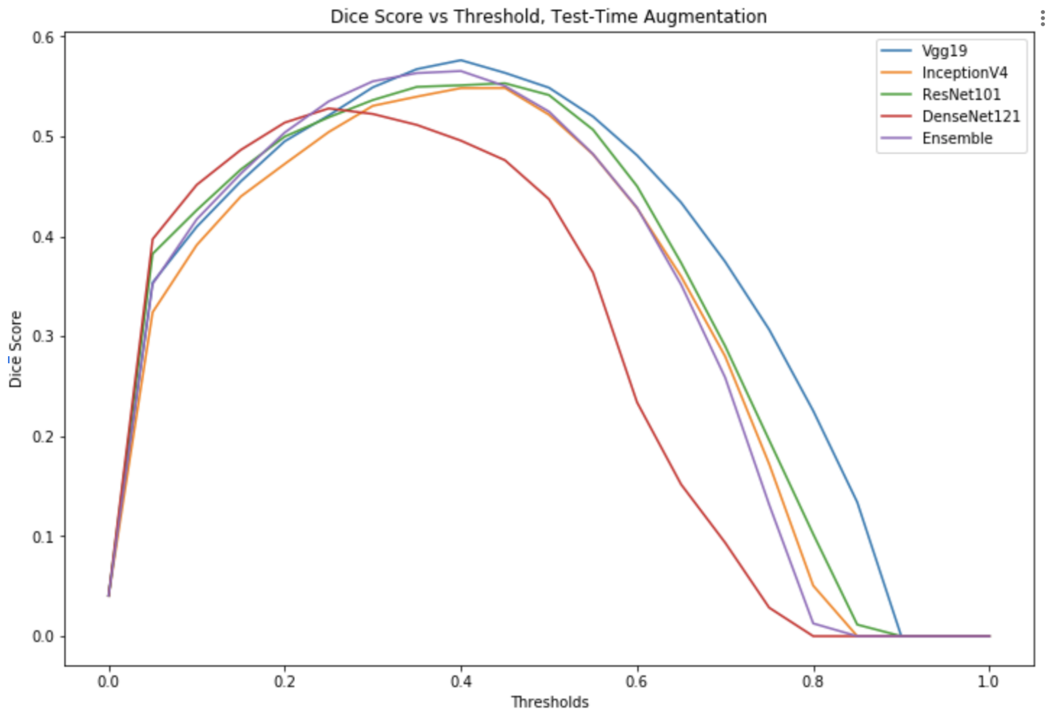

Figure. B.2. Model Dice Score as a function of threshold for distortion condition 0.50. The threshold picked is the threshold that optimizes the performance of *each model individually*. As a result, each model will lie on different locations on the precision-recall curve. Vgg19, ResNet101, DenseNet121, and Ensemble thresholds are 0.40, 0.50, 0.25, 0.40, respectively. InceptionV4 model was not used.

Bibliography

[1] John Pickett. International consortium for brain mapping, 2011. URL `https://ida.loni.usc.edu/`.

[2] D.M. Barch T.E.J. Behrens E. Yacoub K. Ugurbil D.C. Van Essen, S.M. Smith. The wu-minn human connectome project: An overview. *Neuroimage*, 2, 2013. doi: https://doi.org/10.1016/j.neuroimage.2013.05.041.

[3] Mohlberg H Grefkes C Fink GR Amunts K Zilles K Eickhoff SB, Stephan KE. A new spm toolbox for combining probabilistic cytoarchitectonic maps and functional imaging data. *Neuroimage*, pages 1325–35, 2005. doi: 10.1016/j.neuroimage.2004.12.034.

[4] David M. Blei and Padhraic Smyth. Science and data science. *Proceedings of the National Academy of Sciences*, 114(33):8689–8692, 2017. ISSN 0027-8424. doi: 10.1073/pnas.1702076114. URL `https://www.pnas.org/content/114/33/8689`.

[5] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015. ISSN 14764687. doi: 10.1038/nature14539.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances*

in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[8] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander W R Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020. ISSN 1476-4687. doi: 10.1038/s41586-019-1923-7. URL `https://doi.org/10.1038/s41586-019-1923-7`.

[9] Cathy O'Neil. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy.* Crown Publishers, 2016.

[10] A. L'Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz. Machine learning with big data: Challenges and approaches. *IEEE Access*, 5:7776–7797, 2017. doi: 10.1109/ACCESS.2017.2696365.

[11] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019. ISSN 0036-8075. doi: 10.1126/science.aax2342. URL `https://science.sciencemag.org/content/366/6464/447`.

[12] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel

Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2019.12.012. URL `https://www.sciencedirect.com/science/article/pii/S1566253519308103`.

[13] Thomas Grote and Philipp Berens. On the ethics of algorithmic decision-making in healthcare. *Journal of Medical Ethics*, 46(3):205–211, 2020. ISSN 14734257. doi: 10.1136/medethics-2019-105586.

[14] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges, 2021.

[15] Balaji. Lakshminarayanan, Alexander. Pritzel, and Charles. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Neural Information Processing Systems (NIPS 2017)*, 2017. ISSN 03400131. doi: 10.1007/BF00378152.

[16] Jeffrey De Fauw, Joseph R. Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, George van den Driessche, Balaji Lakshminarayanan, Clemens Meyer, Faith Mackinder, Simon Bouton, Kareem Ayoub, Reena Chopra, Dominic King, Alan Karthikesalingam, Cían O. Hughes, Rosalind Raine, Julian Hughes, Dawn A. Sim, Catherine Egan, Adnan Tufail, Hugh Montgomery, Demis Hassabis, Geraint Rees, Trevor Back, Peng T. Khaw, Mustafa Suleyman, Julien Cornebise, Pearse A. Keane, and Olaf Ronneberger. *Clinically applicable deep learning for diagnosis and referral in retinal disease*, volume 24. 2018. ISBN 4159101801. doi: 10.1038/s41591-018-0107-6.

[17] Ethan Danielli, Carol DeMatteo, Geoffrey B. Hall, and Michael D. Noseworthy.

A review of mri and exercise treatment for improved concussion diagnosis and recovery. *Critical Reviewstrade; in Biomedical Engineering*, 48(5):261–283, 2020. ISSN 0278-940X.

[18] Iqbal H Sarker. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science*, 2(3):160, 2021. ISSN 2661-8907. doi: 10.1007/s42979-021-00592-x. URL `https://doi.org/10.1007/s42979-021-00592-x`.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[21] Fei-Fei Li. Cs231n: Convolutional neural networks for visual recognition, 2020. URL `http://cs231n.stanford.edu/`.

[22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[24] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu

Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

[25] Basil Mustafa, Aaron Loh, Jan Freyberg, Patricia MacWilliams, Alan Karthikesalingam, Neil Houlsby, and Vivek Natarajan. Supervised transfer learning at scale for medical imaging, 2021.

[26] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_3. URL https://doi.org/10.1007/978-3-642-35289-8_3.

[27] A. Krizhevsky I. Sutskever R. Salakhutdinov N. Srivastava, G. Hinton. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. ISSN 03702693. doi: 10.1016/0370-2693(93)90272-J.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[30] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[31] D. Ballard. Modular learning in neural networks. In *AAAI*, 1987.

[32] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[34] ozgun cicek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation, 2016.

[35] Wei Chen, Boqiang Liu, Suting Peng, Jiawei Sun, editor="Crimi Alessandro Qiao, Xu", Spyridon Bakas, Hugo Kuijf, Farahani Keyvan, Mauricio Reyes, and Theo van Walsum. S3d-unet: Separable 3d u-net for brain tumor segmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 358–368, Cham, 2019. Springer International Publishing.

[36] Guodong Zeng, Xin Yang, Jing Li, Lequan Yu, Pheng-Ann Heng, and Guoyan Zheng. 3d u-net with multi-level deep supervision: Fully automatic segmentation of proximal femur in 3d mr images. In Qian Wang, Yinghuan Shi, Heung-Il Suk, and Kenji Suzuki, editors, *Machine Learning in Medical Imaging*, pages 274–282, Cham, 2017. Springer International Publishing.

[37] Raghav Mehta and Tal Arbel. 3d u-net for brain tumour segmentation. In Alessandro Crimi, Spyridon Bakas, Hugo Kuijf, Farahani Keyvan, Mauricio Reyes, and Theo van Walsum, editors, *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 254–266, Cham, 2019. Springer International Publishing. doi: 10.1007/978-3-030-11726-9_23.

[38] Qianqian Tong, Munan Ning, Weixin Si, Xiangyun Liao, and Jing Qin. 3d deeply-supervised u-net based whole heart segmentation. In Mihaela Pop, Maxime Sermesant, Pierre-Marc Jodoin, Alain Lalande, Xiahai Zhuang, Guang Yang, Alistair Young, and Olivier Bernard, editors, *Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges*, pages 224–232, Cham, 2018. Springer International Publishing. doi: 10.1007/978-3-319-75541-0_24.

[39] Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss,

Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, and Yunfeng Zhang. One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques. 2019.

[40] Mauricio Reyes, Raphael Meier, Sérgio Pereira, Carlos A. Silva, Fried-Michael Dahlweid, von Hendrik Tengg-Kobligk, Ronald M. Summers, and Roland Wiest. On the interpretability of artificial intelligence in radiology: Challenges and opportunities. *Radiology: Artificial Intelligence*, 2(3):e190043, 2020. doi: 10.1148/ryai.2020190043.

[41] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José M.F. Moura, and Peter Eckersley. Explainable machine learning in deployment. *FAT* 2020 - Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 648–657, 2020. doi: 10.1145/3351095.3375624.

[42] Himabindu Lakkaraju, Nino Arsov, and Osbert Bastani. Robust and stable black box explanations. *Icml*, 2020.

[43] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019. ISSN 25225839. doi: 10.1038/s42256-019-0048-x.

[44] Andrew D. Selbst and Solon Barocas. The intuitive appeal of explainable machines. *Fordham Law Review*, 87(3):1085–1139, 2018. ISSN 0015704X. doi: 10.2139/ssrn.3126971.

[45] Zachary C. Lipton. The mythos of model interpretability. *Communications of the ACM*, 61(10):35–43, 2018. ISSN 15577317. doi: 10.1145/3233231.

[46] Been Kim, Elena Glassman, Brittney Johnson, and Julie Shah. ibcm : Interactive bayesian case model empowering humans via intuitive interaction. *CSAIL-technical report*, pages 1–12, 2015.

[47] Carolyn McLeod and Edward Zalta (Ed.). Trust, 2020. URL `https://plato.stanford.edu/archives/fall2020/entries/trust/`. [Online; accessed 2020-09-14].

[48] Falgun H. Chokshi, Adam E. Flanders, Luciano M. Prevedello, and Curtis P. Langlotz. Fostering a healthy ai ecosystem for radiology: Conclusions of the 2018 rsna summit on ai in radiology. *Radiology: Artificial Intelligence*, 1(2): 190021, 2019. doi: 10.1148/ryai.2019190021.

[49] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *33rd International Conference on Machine Learning, ICML 2016*, 3:1651–1660, 2016.

[50] Bjoern H Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, Levente Lanczi, Elizabeth Gerstner, Marc-andré Weber, Tal Arbel, Brian B Avants, Nicholas Ayache, Patricia Buendia, D Louis Collins, Nicolas Cordier, Jason J Corso, Antonio Criminisi, Tilak Das, Hervé Delingette, Çağatay Demiralp, Christopher R Durst, Michel Dojat, Senan Doyle, Joana Festa, Florence Forbes, Ezequiel Geremia, Ben Glocker, Polina Golland, Xiaotao Guo, Andac Hamamci, Khan M Iftekharuddin, Raj Jena, Nigel M John, Ender Konukoglu, Danial Lashkari, José António Mariz, Raphael Meier, Sérgio Pereira, Doina Precup, Stephen J Price, Tammy Riklin Raviv, Syed M S Reza, Michael Ryan, Duygu Sarikaya, Lawrence Schwartz, Hoo-chang Shin, Jamie Shotton, Carlos A Silva, Nuno Sousa, Nagesh K Subbanna, Gabor Szekely, Thomas J Taylor, Owen M Thomas, Nicholas J Tustison, Gozde Unal, Flor Vasseur, Max Wintermark, Dong Hye Ye, Liang Zhao, Binsheng Zhao, Darko Zikic, and Marcel

Prastawa. The multimodal brain tumor image segmentation benchmark (brats). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, 2015. ISSN 1558254X. doi: 10.1109/TMI.2014.2377694.

[51] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin S. Kirby, John B. Freymann, Keyvan Farahani, and Christos Davatzikos. Advancing the cancer genome atlas glioma mri collections with expert segmentation labels and radiomic features. *Scientific Data*, 4(July):1–13, 2017. ISSN 20524463. doi: 10.1038/sdata.2017.117.

[52] Yarin Gal. *Uncertainty in Deep Learning (PhD Thesis) | Yarin Gal - Blog | Cambridge Machine Learning Group*. PhD thesis, 2016.

[53] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. Uncertainty estimations by softplus normalization in bayesian convolutional neural networks with variational inference. 2018.

[54] Katharina Hoebel, Vincent Andrearczyk, Andrew Beers, Jay Patel, Ken Chang, Adrien Depeursinge, Henning Müller, and Jayashree Kalpathy-Cramer. An exploration of uncertainty information for segmentation quality assessment. In Ivana Išgum and Bennett A Landman, editors, *Medical Imaging 2020: Image Processing*, volume 11313, pages 381–390. International Society for Optics and Photonics, SPIE, 2020. URL `https://doi.org/10.1117/12.2548722`.

[55] Yongchan Kwon, Joong Ho Won, Beom Joon Kim, and Myunghee Cho Paik. Uncertainty quantification using bayesian neural networks in classification: Application to biomedical image segmentation. *Computational Statistics and Data Analysis*, 142(Midl), 2020. ISSN 01679473. doi: 10.1016/j.csda.2019.106816.

[56] Andrew Gordon Wilson. Bayesian deep learning and a probabilistic perspective of model construction, 2020. URL `https://cims.nyu.edu/{~}andrewgw/bayesdlicml2020.pdf`. [Online; accessed 2020-09-14].

[57] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112 (518):859–877, 2017. ISSN 1537274X. doi: 10.1080/01621459.2017.1285773.

[58] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):5575–5585, 2017. ISSN 10495258.

[59] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference. pages 1–38, 2019.

[60] Tanya Nair, Doina Precup, Douglas L. Arnold, and Tal Arbel. Exploring uncertainty measures in deep networks for multiple sclerosis lesion detection and segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11070 LNCS (Dl):655–663, 2018. ISSN 16113349. doi: 10.1007/978-3-030-00928-1_74.

[61] Abhijit Guha Roy, Sailesh Conjeti, Nassir Navab, and Christian Wachinger. Inherent brain segmentation quality control from fully convnet monte carlo sampling. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11070 LNCS:664–672, 2018. ISSN 16113349. doi: 10.1007/978-3-030-00928-1_75.

[62] Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. *Neurocomputing*, 338:34–45, 2019. ISSN 18728286. doi: 10.1016/j.neucom.2019.01.103.

[63] Tim Pearce, Mohamed Zaki, Alexandra Brintrup, Nicolas Anastassacos, and Andy Neely. Uncertainty in neural networks: Approximately bayesian ensembling. 2018.

[64] Alireza Mehrtash, William M. Wells, Clare M. Tempany, Purang Abolmaesumi, and Tina Kapur. Confidence calibration and predictive uncertainty estimation for deep medical image segmentation. pages 1–11, 2019.

[65] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. volume 3, pages 1321–1330. JMLR.org, 2017. ISBN 9781510855144.

[66] Justin Tran, Balaji Lakshminarayanan, and Jasper Snoek. Practical uncertainty estimation and out-of-distribution robustness in deep learning, 2020. URL `https://nips.cc/Conferences/2020/ScheduleMultitrack?event=16649`.

[67] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, V. Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. 2019.

[68] Alain Jungo and Mauricio Reyes. Assessing reliability and challenges of uncertainty estimations for medical image segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11765 LNCS:48–56, 2019. ISSN 16113349. doi: 10.1007/978-3-030-32245-8_6.

[69] Vaishaal Shankar, Rebecca Roelofs, Horia Mania, Alex Fang, Benjamin Recht, and Ludwig Schmidt. Evaluating machine accuracy on imagenet. *International Conference on Machine Learning (ICML)*, PMLR 119, 2020.

[70] P. Shahim, L. Holleran, J. H. Kim, and D. L. Brody. Test-retest reliability of high spatial resolution diffusion tensor and diffusion kurtosis imaging. *Sci Rep*, 7(1):11141, 09 2017.

[71] Ashburner J Smith J Rorden C Li X, Morgan PS. The first step for neuroimaging

data analysis: Dicom to nifti conversion. *Journal of Neuroscience Methods*, 2: 264:47–56, 2016. doi: 10.1016/j.jneumeth.2016.03.001.

[72] Behrens TE Woolrich MW Smith SM Jenkinson M, Beckmann CF. Fsl. *Neuroimage*, pages 62(2):782–90., 2012. doi: 10.1016/j.neuroimage.2011.09.015.

[73] Pavel Yakubovskiy. Segmentation models pytorch. `https://github.com/qubvel/segmentation_models.pytorch`, 2020.

[74] Matthew Brett, Christopher J. Markiewicz, Michael Hanke, Marc-Alexandre Côté, Ben Cipollini, Paul McCarthy, Dorota Jarecka, Christopher P. Cheng, Yaroslav O. Halchenko, Michiel Cottaar, Eric Larson, Satrajit Ghosh, Demian Wassermann, Stephan Gerhard, Gregory R. Lee, Hao-Ting Wang, Erik Kastman, Jakub Kaczmarzyk, Roberto Guidotti, Or Duek, Jonathan Daniel, Ariel Rokem, Cindee Madison, Brendan Moloney, Félix C. Morency, Mathias Goncalves, Ross Markello, Cameron Riddell, Christopher Burns, Jarrod Millman, Alexandre Gramfort, Jaakko Leppäkangas, Anibal Sólon, Jasper J.F. van den Bosch, Robert D. Vincent, Henry Braun, Krish Subramaniam, Krzysztof J. Gorgolewski, Pradeep Reddy Raamana, Julian Klug, B. Nolan Nichols, Eric M. Baker, Soichi Hayashi, Basile Pinsard, Christian Haselgrove, Mark Hymers, Oscar Esteban, Serge Koudoro, Fernando Pérez-García, Nikolaas N. Oosterhof, Bago Amirbekian, Ian Nimmo-Smith, Ly Nguyen, Samir Reddigari, Samuel St-Jean, Egor Panfilov, Eleftherios Garyfallidis, Gael Varoquaux, Jon Haitz Legarreta, Kevin S. Hahn, Oliver P. Hinds, Bennet Fauber, Jean-Baptiste Poline, Jon Stutters, Kesshi Jordan, Matthew Cieslak, Miguel Estevan Moreno, Valentin Haenel, Yannick Schwartz, Zvi Baratz, Benjamin C Darwin, Bertrand Thirion, Carl Gauthier, Dimitri Papadopoulos Orfanos, Igor Solovey, Ivan Gonzalez, Jath Palasubramaniam, Justin Lecher, Katrin Leinweber, Konstantinos Raktivan, Markéta Calábková, Peter Fischer, Philippe Gervais, Syam Gadde, Thomas Ballinger,

Thomas Roos, Venkateswara Reddy Reddam, and freec84. nipy/nibabel: 3.2.1, November 2020. URL https://doi.org/10.5281/zenodo.4295521.

[75] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[76] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[77] nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature Methods*, 18(2):203–211, 2021. ISSN 1548-7105. doi: 10.1038/s41592-020-01008-z. URL https://doi.org/10.1038/s41592-020-01008-z.

[78] Brian McCrindle. Tbi segmentation, 2021. URL https://github.com/mccrinbc/TBI_Segmentation.

[79] RW Cox. Afni: Software for analysis and visualization of functional magnetic resonance neuroimages. *Computers and Biomedical Research*, pages 162–173, 1996. doi: 10.1016/j.neuroimage.2011.09.015.