Machine learning methods for brain lesion delineation

Kevin Raina

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science Mathematics and Statistics[1]

Department of Mathematics and Statistics
Faculty of Science
University of Ottawa

[1]The M.Sc. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

Brain lesions are regions of abnormal or damaged tissue in the brain, commonly due to stroke, cancer or other disease. They are diagnosed primarily using neuroimaging, the most common modalities being Magnetic Resonance Imaging (MRI) or Computed Tomography (CT). Brain lesions have a high degree of variability in terms of location, size, intensity and form, which makes diagnosis challenging. Traditionally, radiologists diagnose lesions by inspecting neuroimages directly by eye; however, this is time-consuming and subjective. For these reasons, many automated methods have been developed for lesion delineation (segmentation), lesion identification and diagnosis. The goal of this thesis is to improve and develop automated methods for delineating brain lesions from multimodal MRI scans. First, we propose an improvement to existing segmentation methods by exploiting the bilateral quasi-symmetry of healthy brains, which breaks down when lesions are present. We augment our data using nonlinear registration of a neuroimage to a reflected version of itself, leading to an improvement in Dice coefficient of 13 percent. Second, we model lesion volume in brain image patches with a modified Poisson regression method. The model accurately identified the lesion image with the larger lesion volume for 86 percent of paired sample patches. Both of these projects were published in the proceedings of the BIOSTEC 2020 conference. In the last two chapters, we propose a confidence-based approach to measure segmentation uncertainty, and apply an unsupervised segmentation method based on mutual information.

Dedications

To my parents, Pritam and Sandeep Raina, for teaching me the value of passion, perserverance, persistence, people, and positivity.

Acknowledgement

I would like to thank firsly and foremost my supervisor Dr.Tanya Schmah for giving me the opportunity to pursue my Master's degree in her lab. Her dedication, expertise, support and pleasant personality have all made my graduate experience outstanding. I cannot express how thankful I am for her encouraging words, stories, and intellect that have uplifted me in times of difficulty. I am grateful for all the opportunities she has given me during my undergraduate and graduate studies; since my Honor's project in fourth year to Masters and hopefully onward. Her ongoing passion and enthusiasm for Mathematics and Statistics has pushed me to my full potential. I am very thankful for her guidance on my thesis, honor's project, collaborative, and independent projects. These past years have been incredibily memorable and I will never forget this wonderful experience. During my graduate studies, I have been fortunate to have worked alongside some extremely intelligent people. I would like to extend a special thanks to Dr.Uladzimir Yahorau and Dr.Dongyang Kuag for being amazing teachers, friends, and guides. I would like to especially acknowledge my family for their continuous support. To my mom, who has always been my hero, inspiration, and an extremely bright light in my life. To my brother, Vik, who has taught me to always set a higher bar for myself. Finally, this is for my father, who had battled ALS for three long years, yet always had a smile. He has taught me the value of life, perserverance and courage. Being a mathematician himself, he always prioritized my education, and taught me everything I know. This is for you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The problem: automatic delineation of brain lesions

In the broadest sense, a brain lesion is any abnormal or damaged tissue within the confines of the human brain [2]. They are most commonly the result of strokes, tumors or infections, as discussed in the following section. Lesions are diagnosed primarily using noninvasive neuroimaging, the most common modalities being Magnetic Resonance Imaging (MRI) or Computed Tomography (CT). As experienced as the radiologist may be, there are significant challenges in making a diagnosis. Physically, brain lesions have a high degree of variability in terms of location, size, intensity and form, which makes diagnosis challenging. They can also often appear similar to healthy brain tissue.

In addition, diagnosis is often time-critical. For instance, ischemic stroke causes increasing tissue death within hours of onset, requiring reperfusion therapies around this time [77]. A prompt and accurate diagnosis based on reliable information is crucial, as the stroke progresses through acute, sub-acute and chronic stages within days. The longer the lesion is left untreated, the higher the chance of the treatment

resulting in bleeding. Currently, only a qualitative analysis of the lesion is undertaken in a clinical setting, as a manual voxel by voxel delineation is time-consuming, and has low inter-rater agreement. Gliomas, the most common type of malignant brain tumor, grow at a rate of increasing severity depending on the grade [22]. As the tumor gets larger, symptoms often worsen, reinforcing the need to monitor lesions. High-grade, aggressive variants require immediate treatment, yet still only a qualitative assessment or simple quantitative measures such as axial diameter are used in practice. Infections of the brain are also fast acting, but are frequently overlooked in diagnosis due to its rarity. The prognosis is worse for patients who carry the infection for a longer time, and so the physician needs to exhibit a high degree of suspicion early to identify an abscess.

Due to the difficulty of the task, its time-critical nature, and the need for accurate, consistent and quantitative results, there is a need for automatic and semi-automatic lesion delineation and identification methods. Thus, a worthwhile goal is to improve and develop machine learning methods for automatic interpretation of brain scans such as MRI and CT, which can provide plentiful and detailed information in a short time-frame. This could benefit many patients whose health would otherwise be jeopardized by an inaccurate or untimely diagnosis. Another important application of automated lesion delineation is in large multi-subject statistical studies, where the speed and replicability of an automatic method can be valuable. The automated methods and contributions presented in this thesis will acknowledge the challenges associated with diagnosing lesions by improving key factors such as computational time, and accuracy.

1.2 Major types of lesions

Brain lesions are primarily classified by their location. For instance, lesions can be located within the skull, in which case they are termed intracranial, or in proximity

of the skull base, in which case they are termed extracranial. Acknowledging that extracranial masses are complex in their own right and warrant a deeper understanding, hereinafter, we restrict our attention and methods to arbitrary intracranial lesions.

Intracranial lesions can be further sub-divided by location into intra-axial, extra-axial and intracentricular [54]. Intra-axial lesions occur within the brain parenchyma (the functional tissue of the brain), whereas extra-axial lesions are naturally located outside it. Intraventricular refers to arbitrary processes that occur within the ventricles of the brain or heart, and thus intraventricular lesions are located in the center of the brain, within its fluid-filled cavities.

All intracranial lesions have mass, but the term mass lesion refers specifically to an abnormal growth of mass in the brain, independent of its source [5]. Mass lesions are routinely confused with ordinary lesions in a clinical setting [82], but mass lesions are distinct from ordinary lesions in that they lead to an increase in overall brain mass as opposed to the damaging of existing tissue. Furthermore, mass lesions can be classified as neoplasms or non-neoplastic mass lesions. Neoplasms, also popularly referred to as brain tumors, occur when the source of mass increase is via brain tissue. Conversely, non-neoplastic mass lesions collectively obtain their mass from any non-tissue source. A neoplasm may be malignant, where it invades the surrounding tissue or spreads across the body. When a neoplasm doesn't exhibit these cancerous characteristics, it is termed benign.

There are many types of lesions, each with their own individual causes, symptoms, and prognosis. Despite the immense variation, stroke and, with an accelerating rate, tumors are leading causes of death worldwide [8, 4]. Together with infection, or abscess, these three types of lesions are well-known for being fast acting and difficult to diagnose [98, 85]. For these reasons, stroke, tumors, and infection will be at the forefront of consideration in this thesis. Furthermore, the methods and analysis presented will largely be focused on the challenges associated with diagnosing these lesions.

1.2.1 Stroke

Stroke occurs when a blood vessel in the brain is blocked by a blood clot or ruptures [19]. Blood vessels, or arteries, are vital for brain functioning as they supply blood, oxygen, and essential nutrients to the brain, making stroke a fatal disease [20]. Regardless of the cause, and whether the artery is blocked or ruptures, a stroke always leads to a restricted supply of blood in part of the brain. Over the next hours,

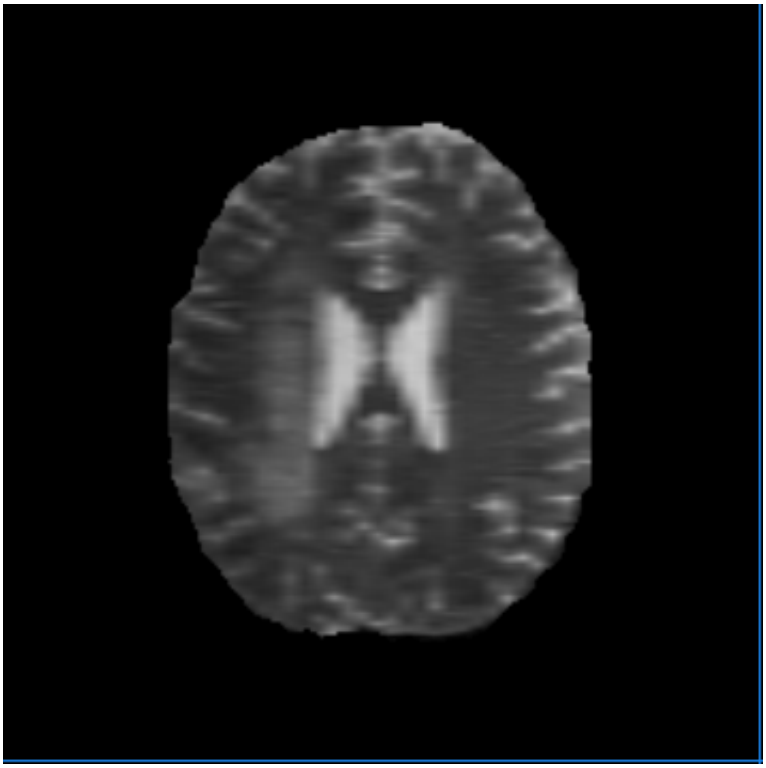

Figure 1.1: T2-weighted MR sequence of a stroke lesion [77].

the starved area of the brain begins to die. Figure 1.1 shows a medical image of a stroke.

Since there are many blood vessels in the brain, there is a high variation in the symptoms. For instance, if a stroke occurs on the left side of the brain, it will affect the right side of the body and is associated with speech, reading, writing, learning, calculation, and logic. Furthermore, when a stroke occurs on the right side, it will

affect the corresponding left side of the body, and attributes such as depth perception, directions, creativity, art, music, and emotion. Moreover, depending on the specific brain area and the extent of damage, direct symptoms include problems with vision, sleeping, moving, fatigue, and depression.

The most prominent risk factor of stroke is high blood pressure [18]. If left untreated over longer periods of time, high blood pressure continuously damages arteries, creating an environment where they can clog or burst easily. The second risk factor is smoking, but is actually closely related to high blood pressure. The chemicals nicotine and carbon monoxide have detrimental effects on the heart and blood vessels. In particular, nicotine constricts blood vessels making them stiff, more prone to damage and decreasing the amount of oxygen and nutrients they can provide to the brain. Similarly, carbon monoxide binds to the molecule hemoglobin in blood, and simultaneously prevents oxygen binding. The combination of stiffer blood vessels and less oxygen forces the heart to pump harder, thereby increasing blood pressure. There are also diet and exercise related risk factors such as diabetes, cholesterol, obesity, excessive consumption of alcohol, and physical inactivity. Heart diseases and blood disorders, with their own unique symptoms and causes can also increase the risk of stroke.

The most common type of stroke is an ischemic stroke [120], which occurs due to the blockage of a blood vessel (occlusion). When the blockage forms in the arteries that directly supply blood to the brain, it is termed thrombotic. Rather than initiating in these arteries, if the blockage originates from the heart or other main artery, but moves toward the brain arteries, it is termed embolic. If a blockage occurs, but for whatever reason the blood flow is restored, the stroke is classified as transient, and is a precursor for something more serious. The second type of stroke is hemorrhagic stroke, which results from a spontaneous blood vessel rupture. The blood then spreads into the brain, making this a mass lesion. Intracerebral hemorrhage results from the rupture bleeding into brain tissue as opposed to subarachnoid hemorrhage, which

bleeds into the space between the brain and skull.

1.2.2 Tumors

A brain tumor is an abnormal growth of tissue (mass lesion), in the brain or spinal cord [89]. As aforementioned, one of the major concerns of brain tumors is that they may invade and destroy the surrounding tissue. Moreover, whether malignant or benign, a brain tumor is a growing mass inside the fixed amount of space inherent to the brain. The resulting compression and displacement of the brain can be fatal or cause significant damage. Figure 1.2 shows a medical image of a brain tumor.

Figure 1.2: Flair MR sequence of a brain tumor [6].

There is ongoing research into the matter of what causes tumors, as they are still not fully understood [9]. Despite this, radiation exposure is one of the well-known risk

factors. In particular, exposure to ionizing radiation is believed to induce cancer, and is associated with environmental factors. Another well-established but continuously studied factor is genetics, as brain tumors are largely present in those who have a family history. Specific genes in DNA, that may be expressed later in life, are under investigation. Other risk factors include a weakened immune system, and chemicals in the environment.

Similar to a stroke, depending on where the tumor grows, it will have different effects on the body [17]. Since tumors are tissue-based, their general symptoms can be divided by the lobes in the brain. For instance, a tumor located in the frontal lobe can impair thought, reasoning, and memory. The temporal lobe controls hearing, vision, and emotions. Furthermore, the parietal lobe controls sensory perception, whereas the occipital lobe is associated with speech, motion, and abstract concepts. In terms of daily life, a brain tumor can cause major changes in personality and emotions, and may lead to depression. Difficulty concentrating, memory loss, and impaired speaking can also be accompanied by a tumor. Noticeably strength, appearance, and mobility can be compromised if the tumor affects physical compartments of the brain.

There are over 120 brain tumor classifications [3], making the diagnosis complex. In addition to malignant or benign classification, tumors are classified by whether they originate in the brain, or originate elsewhere but spread to the brain. The latter is defined as a secondary tumor or metastases, and the former is defined to be a primary tumor. Deeper classification of tumors comprises of their cell origin and a number ranging from 1-4. For instance, in terms of cell origin, gliomas originate from glial cells located in the brain's supportive tissue. Other examples based on the originating tissue are meningiomas, schwannomas, pituitary tumors, and lymphomas. The number associated with a tumor, often called the grade, represents the severity of the disease. Typically low grade tumors are those that have a grade of 1 or 2, are associated with long term survival, and are not aggressive. High grade tumors, on the other hand, have a grade of 3 or 4, grow quickly, are cancerous, and lead to shorter

survival times.

1.2.3 Infection

When infectious bacteria makes its way into the brain, it fixates around a particular area and begins damaging tissue [85]. Immediately, the body's defense mechanism is triggered and white blood cells are sent to fight the infection. This reaction results in inflammation at the area of infection and the creation of pus: a mixture of dead tissue, white blood cells, and bacteria. Eventually, the pus and bacteria becomes surrounded and protected by a strong outer layer (capsule), making the bacteria more resistant. The well-defined, barriered area forms a brain abscess lesion that can be life threatening. Figure 1.3 shows a medical image of a brain infection.

Figure 1.3: CT image of a brain abscess [7].

Bacteria can enter the brain from an infection occuring in the ears, nose, and

teeth. In addition, though less common, a penetrating head injury could also inject bacteria into the brain. The source of bacteria, bacteria strand, and location of infection have a high degree of variability, but a brain abscess is mainly classified and treated according to the point of entry. Similar to stroke and tumor, the symptoms will depend on the location of the lesion, but symptoms strongly associated with an abscess are: headaches, fever, stiff neck, and sickness [1].

1.3 Diagnosing lesions

When presented with a lesion, a neuroradiologist attempts to make an accurate and timely diagnosis. The process is an intellectual one comprising of an initial eye scan of the MR images or CT images [102]. At this stage, some lesions are instantly diagnosed as they exhibit typical visual characteristics. More difficult lesions are diagnosed by excluding unlikely types, and carefully comparing with images in the radiologists' personal database, which is obtained from training, research, and practice. Some lesions can be more complex and require a detailed analysis of its visual characteristics. Images are the most straightforward way to diagnose a lesion, but factors such as the patients' age are also significant covariates, as some lesions can be excluded solely based on age.

One factor that can aid the radiologist is the relative incidence or estimated proportion of a lesion's occurrence amongst patients. For instance, it is estimated that half of intra-axial tumors are metastases [102]. Moreover, amongst metastases, approximately half of them are solitary. Since intrinsic tumors and metastases are exhaustive and disjoint events, it follows that approximately half of intra-axial tumors are intrinsic. In addition, it is estimated that half of all intrinsic brain tumors are gliomas, and roughly 75% of all gliomas are malignant. Applying Bayes' theorem twice to these estimated probabilities, it can be shown that approximately half of all intra-axial tumors are either a malignant glioma or solitary metastasis. Schulthess

et al. [102] claims that diagnosing these lesions "might be good as guessing, without even looking at the images."

1.4 Medical image data

Noninvasive imaging modalities such as Computed Tomography (CT) or Magnetic Resonance Imaging (MRI) scans give radiologists three-dimensional representations of internal brain structures to diagnose and treat lesions [12]. This 3D representation can be viewed from any desired angle, which can provide vital information for locating lesions, surgery planning, and other diagnostics. Medical image data are represented as non-negative real numbers, or intensities, organized into a three-dimensional grid comprising of identically volumed mini-cubes, called voxels (they are the 3D analogue of pixels). Thus, the medical image has three hyper-parameters indicating its dimensions: the number of voxels along each axis. The signal intensity depends on the imaging instrument, but usually is based on many factors such as water, proton density, chemical binding, blood, cerebrospinal fluid, calcification, fat, and melanin of the tissue it represents [102]. For the statistician the collection of intensities form the data for which pattern recognition is desired.

X-Ray CT is a popular modality which is used routinely in clinical practice. CT scanners employ x-ray radiation to generate the data needed to reconstruct internal structures. Traditionally, x-ray based imaging systems would pass x-ray radiation linearly through the patient and record the reduction in magnitude of the radiation as it passes through the tissue. The density of different tissues will affect the magnitude of the radiation differently. By passing the beams linearly without moving the radiation source, these traditional systems would produce 2D projections of a 3D object. In other words, the patient would be place between a fixed x-ray source and x-ray detector. The X-ray CT modality is built upon these same physical principles, but is able to collect complete 3D images, one 2D slice at a time. In order to acquire slice

data, the x-ray beam enters the patient at the edge of the desired slice and the tissue densities are recorded in computer memory via the x-ray detector, however, the fixed angular displacement of the gantry (the x-ray source and detector) provides limited information. To provide the complete slice data, the gantry must be rotated 180° around the patient. This process continues for all slices, where each slice is obtained by adjusting the patient into the correct position via a movable couch, and repeating the gantry rotation. Figure 1.4 shows a gantry and Figure 1.5 shows the complete scanning methodology. Finally, projection algorithms are applied to all the collected data for a full three-dimensional reconstruction. As mentioned, CT scanners generally involve motion of either the x-ray source, detector, or both. Due to this framework, CT works best on anatomical areas that don't involve a lot of motion. Consequently, the head region and in particular, the brain, are prime candidates. Areas such as the heart can't be examined effectively because of the constant movement.

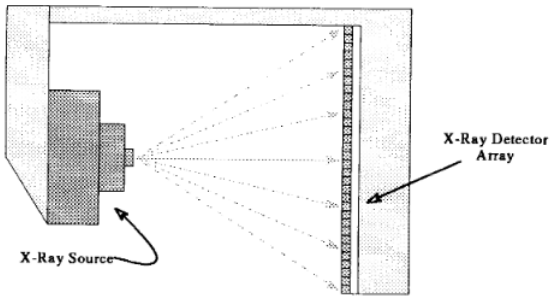

Fig. 2. CT Scanner gentry.

Figure 1.4: Gantry used in CT scans [12].

Although both CT and MRI scanners are able to generate 3D data sets of the desired anatomy, they differ in how the images are acquired. MRI uses safer, non-ionizing radiofrequency waves coupled with magnetic fields to obtain image data. In particular, powerful magnets are set up to produce an external magnetic field, which causes the nuclei of tissue atoms to be aligned. Then, radiofrequency energy

is pulsed through the patient causing these nuclei to be stimulated and spin out of the alignment. Removing the energy will force the nuclei to their aligned state, and the energy released in the process is referred to as the MR signal. This signal, which comprises of the energy released and time for the nuclei to return to the aligned state, is dependent on the specific molecular environment and chemical nature of the particular nucleus. These signals collectively display the image. Patients are placed inside an enclosed magnet, and are reminded to not make any exaggerated movements. In practice, a certain set of parameters such as the flip angle, time to echo, time to repetition and diffusion weighting define a MRI sequence. Each MRI sequence has its own unique features and characteristics which can be advantageous in different scenarios. For instance, a T2-weighted sequence depicts fluid with higher intensities whereas a T1-weighted sequence depicts fluid with lower intensities. Other sequences include Fluid attenuated (FLAIR), proton density weighted (PD), contrast enhanced (CE), and diffusion weighted (DWI). MRIs are especially effective in contrasting soft tissue, white matter, and gray matter. One caveat, however, is that MRI data is generally more expensive to produce in comparison to CT and x-ray imaging.

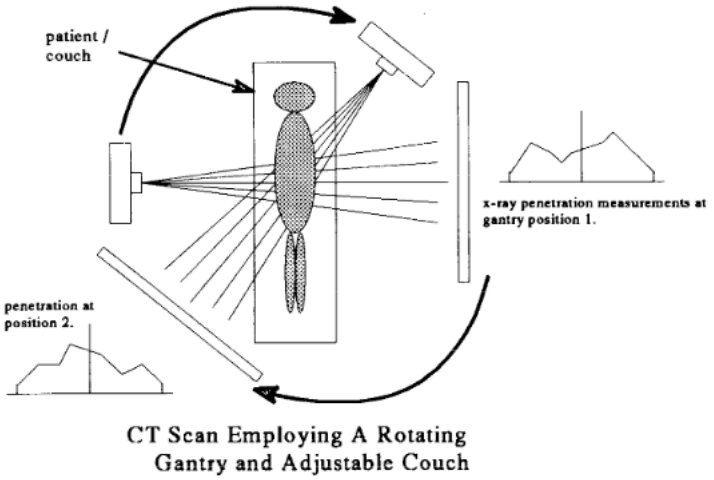

Figure 1.5: CT scanner methodology [12].

1.5 Outline of thesis

The rest of this thesis is organized as follows. Chapter 2 reviews the literature of automatic brain lesion delineation. Chapter 3 introduces Convolutional Neural Networks (CNNs), which are used in all of the following chapters. Chapters 4 and 5 consist of two articles that we published in the proceedings of the BIOSTEC 2020 - BIOIMAGING conference. Though they are separate works, they in fact have relationship that can be understood by referencing section 3.4 and the discussion up to that point. In Chapters 6 and 7, we propose a confidence-based approach to measure segmentation uncertainty, and apply an unsupervised segmentation method based on mutual information. In all of these works, quality of segmentation is evaluated using the Dice metric, which is defined in Chapter 6.

In our first article titled "Exploiting bilateral symmetry in brain lesion segmentation with reflective registration", the response outcome is a Bernoulli random variable which models the presence of a lesion at a particular voxel in a medical image. Its covariates are the MRI signal intensities, from four different scanners, of a 2D patch surrounding the voxel. The contribution is then an increase in model performance by adding additional symmetry based covariates. Our second article titled "Modelling brain lesion volume in patches with CNN-based Poisson Regression" proposed to model a Poisson response variable which measures the number of lesion voxels in a given 3D patch. This response equivalently models the lesion volume, since each voxel captures $1mm^3$ of space. Thus, the covariates are the MRI signal intensities from four different scanners, of a 3D patch. This contribution goes on to evaluate model performance.

The implementations for all projects were done in the Python programming language using the Tensorflow [10] package. All code was programmed from scratch by referencing details from the architectures and methodologies presented in published papers. The Python scripts for each project can be found in the Appendix.

Chapter 2

Literature review

The number of works pertaining to automated methods for brain lesion delineation is growing at a fast rate, and capturing all methods in one comprehensive literature search is a challenging task. Contributions are becoming increasingly diversified in their methodology and application. Nonetheless, the body of literature can be organized into major groupings of works, each with a common theme, but there has been variation in the way they have been grouped over time [24, 74, 75, 55, 113]. Medical image segmentation and lesion volume estimation are two closely related sub-branches, with the former being the most detailed, and informative delineation task. Consequently, it comprises a major part of the delineation literature.

2.1 Segmentation

Recent brain tumor segmentation surveys such as the ones by Ghaffari et al.[55] and Liu et al.[75] are focused around deep learning methods and are exclusively machine learning-based, while older surveys encompass conventional methods as well [24, 74]. The survey of Tiwari et al.[113] is an exception to this pattern and attempts to combine the two categories. Although their work suggests a hierarchy of unsupervised

and supervised methods, a list of methods ungrouped in these branches is presented, demonstrating the difficulty of labeling methods as such. The review presented will attempt to combine the categories, while refraining from insisting on dividing every method according to the unsupervised and supervised categories. Segmentation methods will be organized into conventional and machine learning-based.

2.1.1 Conventional

The performance of conventional segmentation methods did not meet the requirements of clinicians, and are unable to yield the same brain lesion segmentation results in recent challenges [74]. Although they were not particularly effective in stand alone segmentation algorithms, they have become standard in image pre-processing. These methods included threshold-based methods and region-based methods, and are often employed in two-dimensional segmentation [117].

2.1.1.1 Thresholding

Threshold-based methods cast label predictions by comparing the pixel-wise MRI intensities with thresholds, which are either manually tuned [58], or estimated [109]. Stadlbauer et al. [107] recognized that healthy pixels follow a Gaussian distribution and modelled the MRI intensities accordingly. By estimating the mean and standard deviation from the healthy voxels, they were then able to identify pathological tumor pixels by thresholding intensities 3 standard deviations above the mean, despite the non-normality of the combined pathological and healthy voxel intensities. Harris et al. [58] used thresholding for the segmentation of brains into cerebrospinal fluid (CSF), gray and white matter. They noted that CSF pixels appear brighter in T2 images, and in proton-weighted images they appear darker. Thus, by subtracting proton-weighted images from T2 images and adding a constant to ensure positive pixel values, the CSF is highlighted and thresholding can be used to segment CSF from background. In

gray and white matter segmentation, the thresholding parameter was manually tuned by visual correspondence of a rater. As expected, the effectiveness of thresholding as a segmentation algorithm depends on image contrast, and can generally be used to find the approximate location of a tumor.

2.1.1.2 Region-based

Region-based methods assume that neighboring pixels will have similar labels, and compare them through a homogeneity criterion. If a neighboring pixel satisfies the criterion, it will share the same label. In this way, region-based methods attempt to partition the pixels of an image into a mutually exclusive and exhaustive set [121]. In particular, region growing algorithms are the simplest and most common region-based algorithms. They work by initializing isolated pixel locations in the image, and then incorporating (growing) neighboring pixels via a similarity criterion. This results in connected regions with similar pixels. For instance, Adams and Bischof [13] used a seeded region growing algorithm to segment CT images into white, light-gray and dark-gray. In their work starting pixels or a collection of connected pixels (seeds) are either manually chosen by a user or automatically chosen in conjunction with domain-specific prior information, with the seeds forming mutually independent sets. The algorithm then proposes to address all unallocated pixels that are 8-connected to the sets. If there is only one set the pixel connects to, it will be included in the set. However, if the pixel is near a boundary, then it may connect to multiple sets, in which case the absolute deviance between the pixel intensity and the mean intensities (homogeneity criterion) of each set is computed. The set with the minimum absolute deviance will encompass the pixel. Pixel seeds are chosen if the neighboring intensities have less noise, indicating that the pixel is representative of its region. If a region is noisy, then a single pixel may be an outlier. In this case a neighborhood of connected pixels is chosen as the starting seed, and the mean over all pixels is used in the

algorithm. One option is to scale the absolute deviance by the standard deviation of the set, but they note that this should be avoided as it is much more computationally expensive. There have been many variants of the region growing algorithm with an emphasis on improving boundary delineation [80, 65, 39, 101], and using different homogeneity criterion [33, 34, 86].

Another class of region-based methods is the watershed algorithm. The watershed algorithm segments images by identifying boundaries. As per its name, the watershed algorithm visualizes a rainfall on a landscape generated by plotting the image intensities over space as shown in Figure 2.1. In this landscape, higher intensities form higher terrain, and changes across intensities delineate changes in terrain. When the rain hits the landscape, it will naturally flow from the higher to the lower points, and fill basins. The boundaries between the basins are watersheds and form the boundaries of the segmentation. For a more accurate segmentation, the gradient of the image intensities over space is used instead as watersheds will be built on the changes in the rate of change of intensities. Variants of the watershed algorithm have been proposed in brain tumor segmentation including multi-scale [71, 44] multi-parameter [32, 91] and marker-based [100].

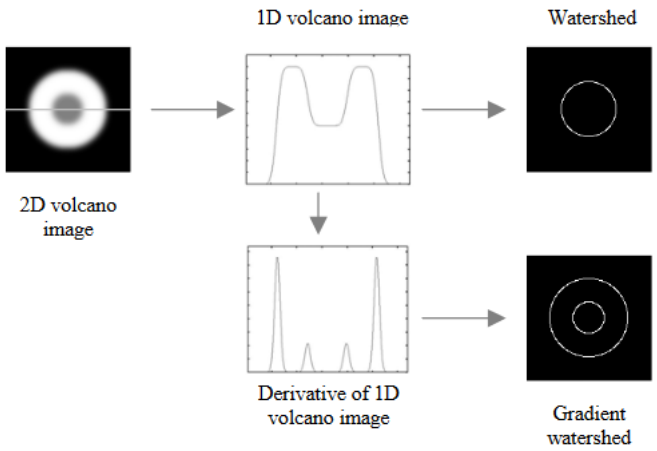

Figure 2.1: Simple example of the watershed algorithm [71].

2.1.2 Classification and clustering

Recent brain lesion segmentation algorithms are largely based on classification or clustering methods. In supervised learning, covariates and labels are known for each observation and used. Classification methods are representative of supervised learning, and segmentation can be seen as pixel-wise classification. Unsupervised methods don't make use of labels, and often treat them as latent variables involved in the image generation process. Clustering methods are representative of unsupervised learning. Semi-supervised methods use labeled and unlabeled datasets, in an effort to deal with costly or unavailable labels. Although some of the methods presented can fall into exactly one of these categories, others can not be easily classified as such.

2.1.2.1 Fuzzy clustering

Fuzzy clustering has been one of the early methods used in unsupervised medical image segmentation. Proposed by Dunn et al. [49], the fuzzy c-means algorithm allows each voxel the capability of belonging to all clusters. The association of a voxel to a cluster is independent of other clusters, and its strength is measured by an image-based similarity coefficient, often distance or intensity-based. In the traditional form, the distance between the voxel intensity and mean intensity in each cluster is calculated. The smaller the deviance, the more likely the voxel belongs to a cluster. Yet, by construction, there is no penalty in the strength of association to any cluster, allowing voxels to be similar to more than one cluster. Phillips et al. [87] demonstrated the strength of the original distance-based fuzzy c-means on a single patient with glioblastoma multiforme with swelling of the surrounding tissue. Fuzzy clustering was effective in differentiating swollen tissue from the boundary of the tumor, showing the potential to guide a subsequent histopathology.

Another tangent of contributions combined additional features in the fuzzy c-means algorithm, while using region growing or connected components as a finishing

step [40, 53, 72]. Generation of labeled supervoxels, a sub-unit obtained by grouping several voxels together, is another technique used in conjunction with fuzzy clustering. Initially proposed by Shi et al.[105], supervoxels reduce computational cost and memory usage, thereby improving computational speed, and minimizing the risk of assigning the wrong label. Achanta et al.[11] generated supervoxels via their proposed simple linear iterative clustering (SLIC) algorithm for segmentation of non-medical images, but Verma et al.[115] extended the method to be applied to brain images. Other examples of generating supervoxels via unsupervised methods are those of Tian et al. [112], Felzenszwalb and Huttenlocher [51], and Ren et al.[92]. Ji et al. [62] combined fuzzy clustering with supervoxels for a novel unsupervised medical image segmentation method.

Suckling et al.[108] made further refinements and modifications for brain tissue segmentation by localizing the algorithm in a moving window scheme to account for reduced contrast at image extremities. They also used a normal mixture model to identify extreme gray and white matter voxels, thereby guiding mean estimation in fuzzy clustering. Dealing with other aspects of the algorithm, Saha et al.[99] experimented with a symmetry-based distance coefficient. Despite its effectiveness, fuzzy clustering is an iterative algorithm and can be time consuming. Efforts have also been made to reduce the computational investment involved [110, 29].

2.1.2.2 Support-vector machines

The standard version of support-vector machines (SVMs) was proposed by Cortes and Vapnik [42], as a parametric kernel-based method. SVMs are a supervised classification method that estimates hyperplanes to separate the feature space and by doing so, classify the response values. Hyperplanes are constructed so as to maximize the distance to each set of features associated with a class. In some cases, the original feature space is not so easily separable, and a transformation to another feature

space via the kernel is required. Zhou et al. [125] applied SVMs to brain tumor segmentation for the binary class segmentation task on 24 patients with only one MR modality. By reasoning that there exists a function that can separate the features into 1 (tumor) and -1 (background), they conclude that maximizing the distance of only tumor data to the origin is sufficient. They then proceed to apply SVM with the radius basis function as the kernel, and demonstrated that the method out performed a semi-automated fuzzy clustering algorithm.

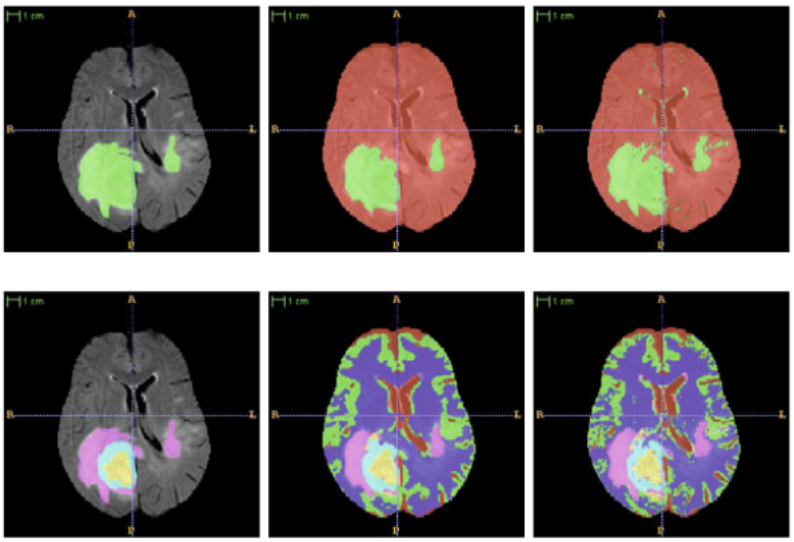

Figure 2.2: Hierarchical segmentation of brains using SVM and CRF [23].

Sub-compartment segmentation of tumors and healthy regions was possible by adding a higher number of MRI modalities to SVM [28, 116]. Illustrating the importance of feature selection, the initial work of Ruan et al.[96] used a lower number of modalities and was only able to segment the tumor from background. However, their subsequent work [97] saw an improvement in results by incorporating more modalities and using kernel class separability for feature selection. Zhang et al. [123] used a multi-kernel SVM combined with feature fusion, and demonstrated an improvement in results over traditional single-kernel approaches. Bauer et al. [23] used MRI

intensities along with first order texture features such as mean, variance, skewness, kurtosis, energy, and entropy in a fully automatic SVM-based method. In addition, they used hierarchical conditional random fields (CRF) regularization to include spatial dependence in classification. Their method performed well in the segmentation of brains into cerebrospinal fluid, gray matter, white matter, active, and edema, as can be seen in Fig 2.2.

2.1.2.3 Registration-only

According to Bauer et al. [24], registration-only methods can mainly be divided into intrapatient and interpatient. Intrapatient registration methods aim to align images accross modalities or over time in an effort to delineate lesions. These methods don't make use of any manual segmentations, and are therefore unsupervised in nature. The alignment task itself, without any consideration to tumor growth or deformation, has been extensively studied by Mang et al. [78], who found that affine registration with a MI metric performs best. A major issue with the alignment task is that modalities are often obtained with different anisotropic resolution and orientation. This problem has been addressed for the general case using interpolation by Thevenaz et al.[111]. Fast and efficient longitudinal intrapatient registration has largely been helpful in resection surgeries [41, 15]. In these studies, pre-operative scans are registered to scans obtained during surgery or shortly after to help practitioners visualize the resection. Another application of longitudinal registration is that of monitoring tumor volume over time [69].

Interpatient registration is largely used in atlas-based segmentation. These segmentation methods have largely been captured in the review by Cabezas et al. [27]. In these methods, the atlas is defined to be the combination of an image, usually MRI, and a manual segmentation of the image. Given an image to segment (target image), the atlas intensity image is deformed and spatially aligned to the target image so as to

be aligned. In doing so, the labels of the manual segmentation are also propagated to the target image, producing a segmentation. However, the drawbacks of atlas-based segmentation methods are that they are time-consuming in the alignment step and they require a large repository of labeled datasets to account for various lesion types.

The pioneering work of joint-registration and segmentation was undertaken by Ashburner et al. [16], but was applied to the segmentation of healthy brains. They proposed a Bayesian framework to estimate posterior probabilities of tissue classes while simultaneously incorporating registration parameters to deform the tissue maps. Pohl et al. [88] challenged the practicality of the assumption that the atlas is globally aligned to the image space in the initial work and made further refinements for healthy brain tissue segmentation. More recently, Gooya et al. [56] and Parisot et al. [83] have been among the first to extend and refine the method to be applied to brain lesions.

2.1.2.4 Convolutional neural networks

Convolutional neural networks (CNNs) have consistently ranked on the top of the leaderboard in many recent brain lesion segmentation challenges such as the ISLES 2015 [77], ISLES 2016/2017 [119] and BRATS 2018 [22]. See Chapter 3 for a detailed introduction of CNNs. This is due to their powerful ability to process and learn image-related features. CNNs include a sequence of convolutional, pooling, and activation tensor transformations. There has been variation in the final step, which can either consist of a more recently used final convolutional layer (fully convolutional network), or an older non-convolutional fully connected layer. The models are commonly formulated by assuming a voxel-wise multinoulli distribution, conditional on the MRI intensities of neighboring voxels where the parameters are obtained and estimated with a CNN. Provided the manual segmentations, they employ iterative stochastic optimization on a log-likelihood objective to update the network parame-

ters. Popular examples of efficient and well-performing CNNs are those of Kamnitsas et al.[64], Havaei et al. [59], and Ronneberger et al. [94]. There has been much experimentation with CNN architectures. CNN architectures can be single/multi path, fully convolutional, cascaded, or any combination of the three [75].

Single/multi-path: Single-path neural networks contain only one flow of data processing, where the flow usually begins by extracting features from the input image with a convolution. Subsequently, pooling and activation layers are included and the cycle repeated. The main advantage of single-path networks are their computational efficiency, and this is the reason they have been used in many works [126, 114, 26]. Despite this perk, multi-path networks make use of multiple data processing pathways, incorporating features from different scales. The features are then combined via concatenation for final label predictions. In one common framework, the same image can be fed into multiple pathways, but each pathway has varying convolutional kernel sizes. Larger kernel sizes are associated with global features such as positioning of objects, whereas smaller kernel sizes are associated with local features such as texture, size, and boundary details. Features are simultaneously trained in the training phase. In the research work of Moeskops et al. [81], three pathways with kernel sizes of 5×5 , 7×7, and 13×13 were used. Havaei et al.[59] proposed a novel two-path architecture with kernels of 7×7 and 13×13, and further experimented with cascaded architectures that use output probability tensors as features in the training of another two-path CNN architecture. Instead of varying the kernel-size, another approach is to vary the dimensions of the input image across the pathways and use scaling techniques to reduce parameter costs. For instance, Kamnitsas et al. [64] proposed a dual-pathway CNN, DeepMedic, with a similar global and local pathway structure. The architecture takes normal resolution image segments of size $25 \times 25 \times 25$ down one pathway and downscaled low resolution inputs of size $19 \times 19 \times 19$ down the second pathway. DeepMedic is a deep CNN with several layers of $3 \times 3 \times$

3 convolution kernels, and requires upscaling the second pathway before concatenation. Their network is fully convolutional and is trained on image segments that are comparatively larger than the receptive field to enable dense training, which yields a more favorable proportion of training labels. Zhao et al. [124] also varied image segment sizes, but used three pathways: a large scale pathway with inputs of 48 × 48, a medium scale pathway with inputs of 18 × 18, and a small scale pathway with inputs of size 12 × 12. The networks of Sedlar et al.[103] and Choi et al. [38] involve individual architectures to account for a certain feature scale, and combine them for accurate segmentation in tumor and ischemic stroke respectively. Two examples of architectures are shown in Figures 2.3 and 2.4.

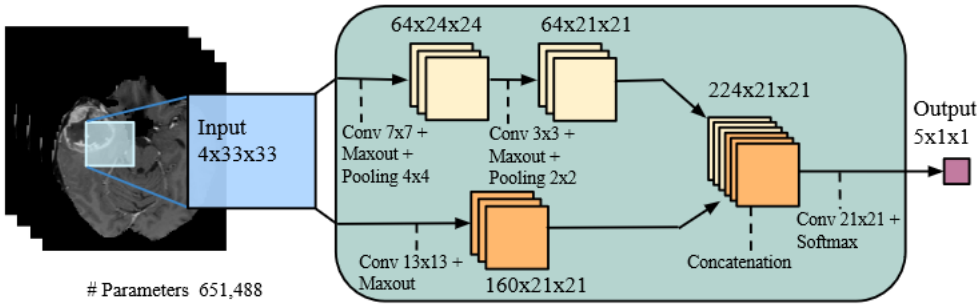

Figure 2.3: Dual-pathway network TwoPathCNN of Havaei et al. [59] .

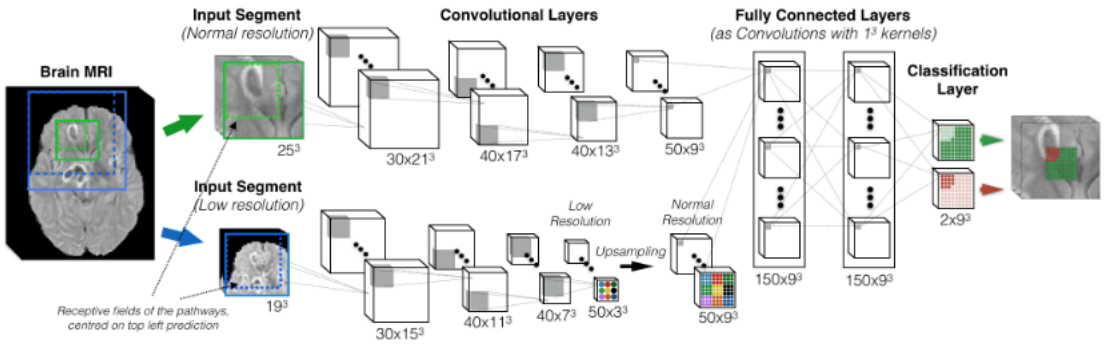

Figure 2.4: Dual-pathway network DeepMedic of Kamnitsas et al. [64] .

Fully convolutional network: Traditionally, CNNs would cast label predictions via a non-convolutional fully connected layer. The output of the CNN would be the label prediction, but due to this non-convolutional casting, the CNN became a patch classifier predicting one voxel at a time. That is, to obtain the classification for another voxel, the CNN has to be run again with the new covariates. This made it difficult to produce entire segmentations of medical images in a timely manner. Long et al. [76] noticed this issue, and proposed to replace the fully connected layer with an upsampling procedure followed by a final label-casting convolutional output layer, which resulted in the creation of the fully convolutional network (FCN). The upsampling procedure ensures that the contracted features are of the same spatial size as the original image. However, since deeper features are obtained from successive layers, localization information is also lost. For this reason, upsampling and fusion is often performed across shallower layers to feed more precise and useful features into the final convolutional layer. FCNs preserve the translational invariance property of CNNs and allow label casting for image segments larger than the receptive field of the network, with the ultimate benefit being applied to entire images at test time. One disadvantage of FCNs are their inability to directly model the contexts of the label domain. To account for this, research works like Jesson et al. [61] and Shen et al. [104] experimented with the loss function.

A popular FCN in brain lesion segmentation is U-Net [94], as shown in Figure 2.5 . U-Net is composed of a connected contracting path and a symmetric expanding path, thus forming a U-shaped architecture. The contracting path learns deeper features with convolution and pooling operation, while the expanding path upsamples to preserve localization information. A unique feature of U-Net are its skip connections which directly combine features from symmetric layers of the architecture to help the expanding path repair detail. There is no fully connected layer, and images are processed directly into segmentations through convolution. To deal with the natural loss of border segmentation seen in FCNs, U-Net extrapolates the missing context by

mirroring the input image and using a tiling strategy. Several variants of U-Net have also been used in brain tumor segmentation [37, 45, 68].

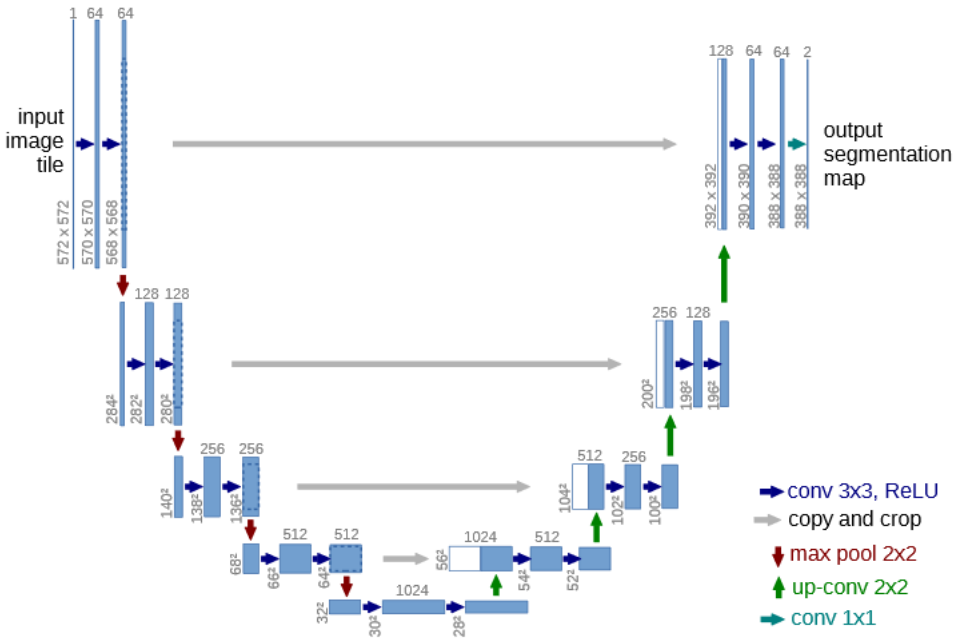

Figure 2.5: U-Net architecture [94] .

Cascaded architectures: Cascaded CNN architectures function by using the outputs of one CNN as the inputs of another, by training the networks independently or jointly. They have largely been developed to incorporate spatial prior information, such as tumors being surrounded by edema, and to deal with the imbalance of healthy and pathological voxels commonly seen in brain lesion segmentation. In this way, cascaded CNNs can help reduce false positive errors. Havaei et al. [59] developed three cascaded CNNs to model direct dependencies of spatially close labels. In particular, the softmax activations of the image from TwoPathCNN were added as features into three different levels of a second TwoPathCNN: InputCascadeCNN concatenated the features with the input image, LocalCascadeCNN concatenated the features with the features from the first convolution of the local pathway, and MFCascadeCNN

concatenated the features with the pre-output features that were obtained from the two pathways. InputCascadeCNN ended up performing the best, with the cascaded architectures almost always besting the original TwoPathCNN on BRATS2013.

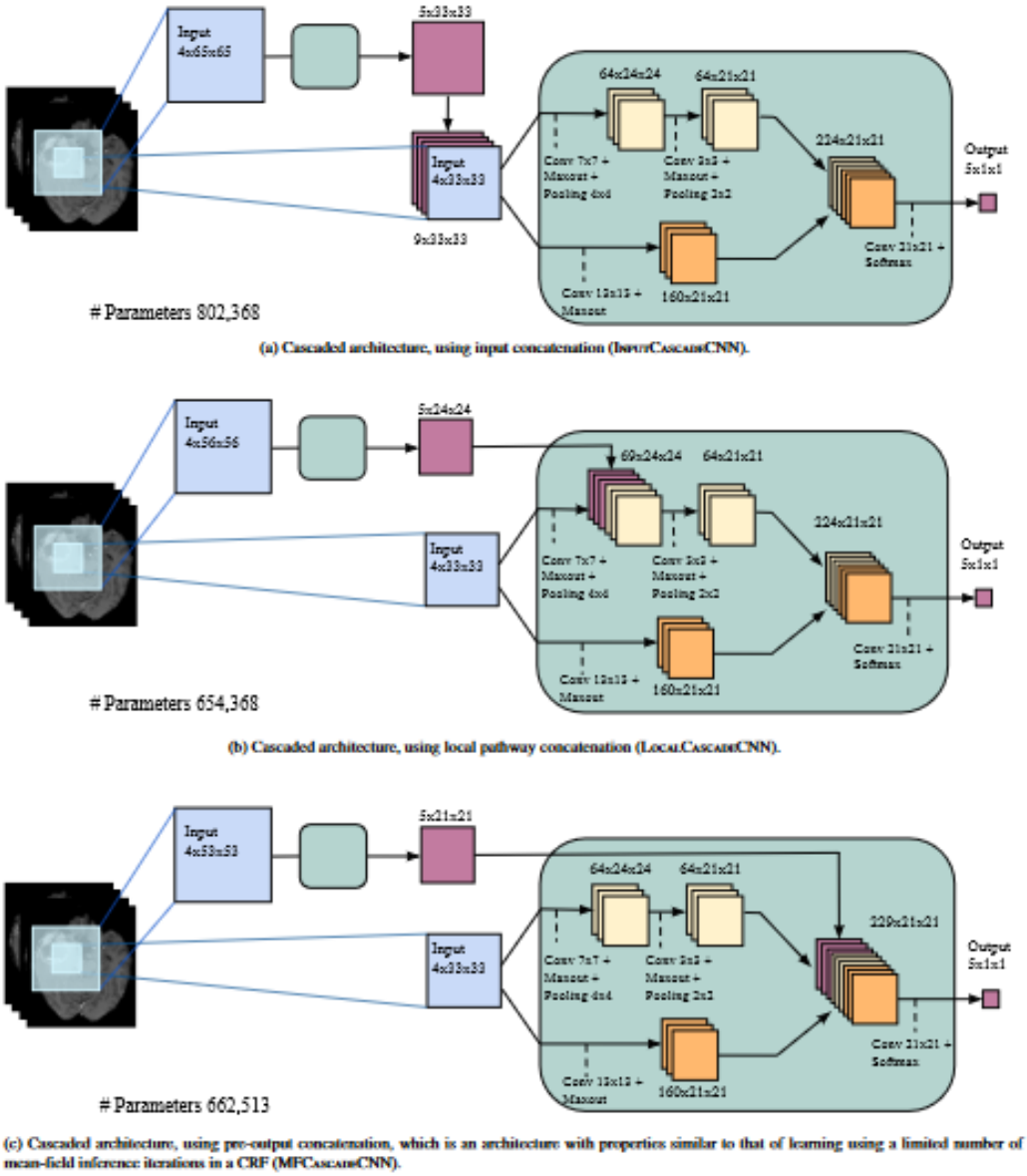

Figure 2.6: Cascaded networks of Havaei et al. [59] .

Another advantage of cascade CNNs are their ability to turn a multi-class seg-

mentation task into a sequence of binary segmentation tasks. One example is the CNN of Wang et al. [118], who used a sequence of three cascaded networks coupled with subregion hierarchal information. In particular, the first network (WNet) cropped the tumor from the medical image. The second network (TNet) took as input the cropped image and then segmented the tumor core from the rest of the tumor. The final network (ENet) worked similarly by taking the segmented tumor core as input and segmenting the enhancing core from the tumor core. Other examples of cascaded architectures are those of Casamitjana [31], Chen [36] and Liu [73].

2.1.2.5 Auto-encoders

Auto-encoders work by transforming high-dimensional image data into a lower dimensional latent space via the encoder, and from the latent space reconstructs the image via the decoder. The decoder and encoder networks are usually trained by minimizing distance-based metrics from the reconstructed and input images. As in Chen et al. [35], auto encoders are usually trained on healthy brains. In doing so, lesioned-brains will be reconstructed as if they were filled with healthy tissue instead. Analysis of the voxel-wise difference between the reconstructed and input image will lead to a segmentation of the lesion. Recently proposed by Diedrik et al. [66], variational auto encoders (VAEs) attempt to improve the healthy image reconstruction from the latent space by combining stochastic inference with the auto-encoder framework. The recent contributions of Baur et al. [25] and Kohl et al. [67] are examples of VAEs applied to medical image segmentation. In Baur et al., the reconstructed image is subtracted voxel-wise from the original image to produce a segmentation, while Kohl et al., used a probabilistic U-Net and introduced a posterior network as a segmentation variant.

2.2 Lesion volume estimation

Another brain lesion delineation task is the direct modelling of lesion volume. In a clinical study, Alexander et al. [14] showed that lesion volume is a significant covariate for understanding ischemic stroke deficits after the initial onset. In application, lesion volume has generally been a dependable factor in the prognosis of ischemic stroke [79, 93] and multiple sclerosis [127, 21]. In a study by Erskine et al. [50] comparing the effects on volume estimation by using different magnetic resonance imaging scanners, lesion volume was estimated from a computer-assisted segmentation tool. Other methods for estimating lesion volume favored a geometric approach, wherein the lesions surface area per slice is calculated and the estimate is derived by summing across slices [84, 52].

Chapter 3

A primer on CNNs

3.1 Tensors

The term tensor has different meanings to mathematicians, statisticians, and data scientists. In the machine learning community, particularly as part of the open source CNN building software Tensorflow [10], a tensor is a multidimensional array.

Definition 1. *A Tensor $T_{S_1 \times S_2 \times ... \times S_k}$, or simply T, is a multidimensional array with real number entries given by $T_{S_1 \times S_2 \times ... \times S_k}(s_1, .., s_k)$ for $s_1 \in \{1, ..., S_1\}$, $s_2 \in \{1, ..., S_2\}$,...,$s_k \in \{1, ..., S_k\}$.*

Remark 1. *Here k is called the dimension of the tensor and the S_i are the lengths of the axes.*

Remark 2. *When $k = 1, 2$ the tensor is commonly known as a vector and matrix respectively .*

Image data can often be conveniently expressed as tensors, and CNNs work entirely on tensors. The ability of CNNs to readily discover underlying patterns in images relies on being able to manipulate and transform tensors in a detailed and useful manner.

3.2 Tensor operations

As with vectors and matrices, algebraic operations can be performed on tensors of arbitrary dimensions. That is, tensors can be added, subtracted, multiplied, and divided element-wise.

Definition 2. *Let $T_{S_1 \times S_2 \times \ldots \times S_k}$ and $G_{S_1 \times S_2 \times \ldots \times S_k}$ be two k-dimensional tensors. Then,*

$$O_{S_1 \times S_2 \times \ldots \times S_k} := T_{S_1 \times S_2 \times \ldots \times S_k} + G_{S_1 \times S_2 \times \ldots \times S_k} \tag{3.2.1}$$

where,

$$O_{S_1 \times S_2 \times \ldots \times S_k}(s_1, .., s_k) = T_{S_1 \times S_2 \times \ldots \times S_k}(s_1, .., s_k) + G_{S_1 \times S_2 \times \ldots \times S_k}(s_1, ..s_k), \forall (s_1, .., s_k) \tag{3.2.2}$$

Remark 3. *Note the dimension and length of the axes must be the same across the two tensors.*

Remark 4. *This definition can be extended to subtraction, multiplication, and division by simply substituting in the required operation. For instance, $T_{S_1 \times S_2 \times \ldots \times S_k} \times G_{S_1 \times S_2 \times \ldots \times S_k}$ refers to the element-wise multiplication of the tensors.*

Prior to defining the convolution operation, we need the dot product tensor operation, which entails multiplying corresponding entries in the tensors and summing over all entries.

Definition 3. *Let $T_{S_1 \times S_2 \times \ldots \times S_k}$ and $G_{S_1 \times S_2 \times \ldots \times S_k}$ be two k-dimensional tensors. Then,*

$$O_1 := T_{S_1 \times S_2 \times \ldots \times S_k} \bullet G_{S_1 \times S_2 \times \ldots \times S_k} \tag{3.2.3}$$

where,

$$O_1 = \sum_{(s_1, .., s_k)} T_{S_1 \times S_2 \times \ldots \times S_k}(s_1, .., s_k) \times G_{S_1 \times S_2 \times \ldots \times S_k}(s_1, ..s_k) \tag{3.2.4}$$

Remark 5. *Note the dimension and length of the axes must be the same across the two tensors.*

Remark 6. *The result of the dot product operation, is O_1. As per definition 1, this is a real number.*

The convolution operation is the backbone of CNN architectures, since it is an effective way of transforming image features to capture detailed underlying patterns. We will limit our attention to 3D convolutions, since it is rare to see convolutions for larger dimensions. For instance, the Tensorflow program only has convolution methods defined up to 3 dimensions. Though, it is possible to extend this definition to higher dimensions.

Definition 4. *Given two 3D tensors $T_{S\times S\times S}$ and $W_{P\times P\times P}$ with $P \leq S$ the convolution with stride length 1, is*

$$O_{S-P+1\times S-P+1\times S-P+1} := T_{S\times S\times S} \star W_{P\times P\times P} \tag{3.2.5}$$

where,

$$O_{S-P+1\times S-P+1\times S-P+1}(s_1, s_2, s_3) = T_{S\times S\times S}([s_1, s_1{+}P], [s2, s_2{+}P], [s_3, s_3{+}P]) \bullet W_{P\times P\times P} \tag{3.2.6}$$

$$\forall (s_1, s_2, s_3) \in \{1, .., S - P + 1\}^3$$

and $T_{S\times S\times S}([s_1, s_1{+}P], [s2, s_2{+}P], [s_3, s_3{+}P])$ represents a sub-tensor extracted from $T_{S\times S\times S}$ of dimensions $P \times P \times P$ at location s_1, s_2, s_3.

Remark 7. *For this definition, we insisted that T and W have uniform dimensions for simplicity. This need not be the case, but uniform axes are more common in practice.*

Remark 8. *The tensor $W_{P\times P\times P}$ is an example of a trainable parameter tensor introduced by the CNN called the kernel or filter. The size, P, is termed the kernel or filter size.*

Another useful tensor operation is the pooling operation . This operation is a way of incorporating dimension reduction while insisting a larger context of information is used. Again, we will consider the 3D case, but we note this operation can be extended to arbitrary dimensions.

Definition 5. *Given a 3D tensor $T_{S\times S\times S}$ and pooling parameter p, the pooling operation with stride length* 1 *outputs*

$$O_{S-p+1\times S-p+1\times S-p+1} := \mathcal{P}(T_{S\times S\times S}; p) \tag{3.2.7}$$

where,

$$O_{S-p+1\times S-p+1\times S-p+1}(s_1, s_2, s_3) = \max_{\substack{i\in[s_1,s_1+p]\\ j\in[s_2,s_2+p]\\ k\in[s_3,s_3+p]}} T_{S\times S\times S}(i, j, k) \tag{3.2.8}$$

$$\forall(s_1, s_2, s_3) \in \{1, .., S-p+1\}^3$$

Remark 9. *For this definition, we insisted that T have uniform dimensions for simplicity. This need not be the case, but uniform axes are more common in practice.*

The final major tensor operation is the max-out operation . This operation introduces a non-linearity into the tensor transformations for deeper and more complex pattern recognition. There are many variants of max-out, but we define the basic version. For this operation, since it is routinely applied to tensors of arbitrary dimensions, we define it similarly.

Definition 6. *Given a N-D tensors $T_{S_1\times S_2\times\ldots\times S_N}$, the max-out operation with parameter k outputs*

$$O_{S_1\times S_2\times\ldots\times S_N} := \mathcal{M}(T_{S_1\times S_2\times\ldots\times S_N}; k) \tag{3.2.9}$$

where,

$$O_{S_1\times S_2\times ...\times S_N}(s_1, s_2, ..., s_N) = \max_{\omega\in\{0,..,k\}} T_{S_1\times S_2\times ...\times S_N}(s_1, s_2, .., s_N + \omega) \tag{3.2.10}$$

$$\forall(s_1, s_2, .., s_N)$$

Remark 10. *As we see, max-out is an operation that mainly deals with the last dimension of the incoming tensor, but preserves the tensor's dimensions.*

Remark 11. *The set of elements in* $\{T_{S_1\times S_2\times ...\times S_N}(s_1, s_2, .., s_N+\omega); \omega = 0, .., k\}$ *may have cardinality* 1 *to* k*, depending on the proximity of the position* $(s_1, s_2, .., s_N)$ *to the boundary.*

3.3 CNN architectures

As we have discussed common tensor operations, we can now define the CNN architecture .

Definition 7. *Given an input tensor* $X_{D_1\times D_2\times ...\times D_M}$*, a CNN architecture is a finite sequence of tensor operations, composed on one another, to yield an output tensor* $\eta_{H_1\times H_2\times ...\times H_K}$*. That is,*

$$CNN(X, \Theta) = F_N(F_{N-1}(...(F_1(X; \Theta_1)...); \Theta_{N-1}), ; \Theta_N) = \eta \tag{3.3.1}$$

where, $\Theta = (\Theta_1, \Theta_2, .., \Theta_N)$ *represents all the trainable parameters of the CNN, and* Θ_i *represents the trainable parameters for tensor operation* F_i*.*

Remark 12. *It is possible, as in the case of pooling or max-out, for a tensor operation to not introduce any trainable parameters. Trainable parameters are commonly found in the convolution operation.*

Remark 13. *The output tensor η, is usually just a real number. That is, $K = 1$, and $H_1 = 1$*

3.4 Statistical modelling

In a regression framework , the goal is to predict and model a response random variable, given its fixed covariates . In definition 7, the input tensor X can also be interpreted as realized covariates which are jointly distributed with a random response tensor Y, for which inference is desired. In a discriminative model, a parametric conditional distribution is assumed. That is,

$$Y|X \sim f(y; \eta(X)), \tag{3.4.1}$$

where η, a function of the covariates, is the parameter of the conditional distribution. In this manner, the CNN can serve as a way to estimate the conditional parameter from its covariates.

$$Y|X \sim f(y; CNN(X, \Theta)). \tag{3.4.2}$$

In practice, Θ is estimated during training by an estimator $\hat{\Theta}$ that is obtained by maximizing the likelihood. Let $\{(x_1, y_1), ..., (x_N, y_N)\}$ be all training observations. Assuming conditional independence across units, $\hat{\Theta}$ can be found by

$$\hat{\Theta} = \arg\max_{\Theta} L(\Theta), \tag{3.4.3}$$

$$L(\Theta) = \prod_{i=1}^{N} f(y_i; CNN(x_i, \Theta)). \tag{3.4.4}$$

To locate this maximum, numerical methods like gradient descent are used. Sometimes N is too large and would require too many computational resources to estimate

Θ. In these cases, it is customary to randomly sample a subset of training observations B, called the batch size, and perform gradient descent for an iteration. The next iteration would correspond to a new sample and so on. Further training details of the respective projects can be found in Chapter 4, 5, 6 and 7.

Chapter 4

Exploiting bilateral symmetry in brain lesion segmentation with reflective registration

Exploiting bilateral symmetry in brain lesion segmentation with reflective registration

Kevin Raina[1][a], Uladzimir Yahorau[1][b] and Tanya Schmah[1][c]
[1]*Department of Mathematics and Statistics, University of Ottawa, Ontario, Canada*
{}*@uottawa.ca, @gmail.com*

Keywords: Stroke, Brain Lesions, Lesion Mapping, Image Segmentation, MRI, Convolutional Neural Network.

Abstract: Brain lesions, including stroke lesions and tumours, have a high degree of variability in terms of location, size, intensity and form, making automatic segmentation difficult. We propose an improvement to existing segmentation methods by exploiting the bilateral quasi-symmetry of healthy brains, which breaks down when lesions are present. Specifically, we use nonlinear registration of a neuroimage to a reflected version of itself ("reflective registration") to determine for each voxel its homologous (corresponding) voxel in the other hemisphere. A patch around the homologous voxel is added as a set of new features to the segmentation algorithm. To evaluate this method, we implemented two different CNN-based multimodal MRI stroke lesion segmentation algorithms, and then augmented them by adding extra symmetry features using the reflective registration method described above. For each architecture, we compared the performance with and without symmetry augmentation, on the SISS Training dataset of the Ischemic Stroke Lesion Segmentation Challenge (ISLES) 2015 challenge. Using linear reflective registration improves performance over baseline, but nonlinear reflective registration gives significantly better results: an improvement in Dice coefficient of 13 percentage points over baseline for one architecture and 9 points for the other. We argue for the broad applicability of adding symmetric features to existing segmentation algorithms, specifically using the proposed nonlinear, template-free method.

1 INTRODUCTION

Segmentation of lesions in neuroimages, also called lesion mapping, can give valuable information for prognosis, treatment planning and monitoring of disease progression. The "gold standard" for lesion segmentation is still manual delineation by a human expert, going through each of the horizontal slices of the three-dimensional image and labeling each separate voxel as either healthy or belonging to a lesion. This is tedious, time-consuming, and often impractical, and therefore in practice, a human expert usually gives only a qualitative assessment of lesions. Further, there is inter-observer variability; the size of this variability varies significantly by task, but we note that an average Dice score of 0.58 for overlap of manually-outlined lesions by two raters was reported for the ISLES2016 challenge (Winzeck et al., 2018). These observations indicate a need for automatic brain lesion segmentation algorithms. However, accurate lesion segmentation is a challenging task for many reasons, including large variability in location, size, shape and frequency of lesions across patients.

While a plethora of automatic lesion segmentation methods has been proposed, most of the currently leading methods are based on convolutional neural networks (CNN) (Winzeck et al., 2018). Many of these use 2D CNNs, where the 3D neuroimage is processed as a sequence of independent 2D slices. These approaches are arguably suboptimal, since they do not take into account the 3D spatial structure of the data. Nonetheless, many 2D methods have shown promising results, including the methods of Havaei et al. (Havaei et al., 2017) and Kamnitsas et al. (Kamnitsas et al., 2017), which we use as baseline architectures in the present paper. Some other works have used CNNs with an input of three orthogonal patches around each voxel being classified, thus incorporating some 3D information, however this significantly increased memory requirements and computational complexity. The technique of dense inference greatly sped up inference time, and led to several successful 3D segmenta-

[a] https://orcid.org/0000-0002-6240-9675
[b] https://orcid.org/0000-0002-0522-4148
[c] https://orcid.org/0000-0002-0404-8824

tion methods, see discussion and references in Kamnitsas et al. (Kamnitsas et al., 2017).

We propose a general method for improving existing segmentation algorithms, including all of the CNN-based methods mentioned above, by exploiting the bilateral quasi-symmetry of healthy brains. This symmetry often breaks down when lesions are present. In particular, stroke lesions usually affect only one hemisphere; while for some other lesion types such as tumours, lesions may be present in both hemispheres but any symmetry is coincidental and rare. The basic idea is illustrated in Figure 1. The first subfigure shows an axial slice of a brain with a stroke lesion in one hemisphere, and two homologous ("mirror") voxels, i.e voxels in corresponding parts of the brain but in opposite hemispheres. Throughout this paper, homologous voxels are determined using "reflective registration", which is registration of an image to its own reflection, as detailed in the next section. In healthy normal brains, there is a strong correlation between intensities of homologous voxels. In lesioned brains, voxels in a lesion often have intensities very different from the intensities of their homologous voxels, as shown in Fig. 1 (b). On the other hand, lesions typically represent a small proportion of total brain volume, so non-lesion voxels typically have non-lesion mirror voxels as well, typically resulting in small intensity differences (if the mirror voxels have been accurately located). The distribution of these intensity differences, for both lesion and non-lesion voxels, is illustrated in Figures 1 (c) and (d). The difference between these two subfigures is in the nature of the reflective registration used to identify homologous voxels: in (c), "linear" (affine) registration is used, while in (d), nonlinear registration is used. The increase in mass around zero for non-lesions when using nonlinear registration (d) in comparison to linear registration (c) suggests the superiority of nonlinear reflective registration in locating mirror voxels. In both cases, the pattern of intensity differences can be used to aid the classification of a voxel as lesion or non-lesion. This method is inspired by the clinical practice of radiologists, who make frequent use of comparisons with homologous areas to detect abnormalities.

Our method, explained in more detail below, uses 3D nonlinear registration of a neuroimage with a reflected version of itself to determine for each voxel its homologous voxel in the other hemisphere. A patch around the homologous voxel is added as a set of new features to the segmentation algorithm. To evaluate this method, we implemented two baseline multimodal MRI stroke lesion segmentation algorithms, both based on 2D CNNs, following Havaei et al. (Havaei et al., 2017) and Kamnitsas et al. (Kamnitsas et al., 2017), and then augmented them by adding extra symmetry features as described above. For each architecture, we evaluated the baseline method and two versions of symmetry augmentation: one using "linear" (affine) registration only, and one using nonlinear registration. We compared the performance of these three segmentation methods on the SISS Training dataset of the Ischemic Stroke Lesion Segmentation Challenge (ISLES 2015)(Maier et al., 2017). Though our experiments use 2D CNNs, our method can be applied without modification to 3D CNNs.

We are aware of three prior works that have also used brain quasi-symmetry to improve the performance of CNN based methods: (Shen et al., 2017), (Wang et al., 2016) and (Zhang et al., 2017). Shen et al. use the SIFT-based method of Loy and Eklundh (Loy and Eklundh, 2006) to identify homologous voxels, and report a mean improvement of 3% in Dice scores on the high-grade (HG) BraTS2013 (brain tumor) Training set. Wang et al. report a higher increase in mean Dice scores, from 0.63 to 0.78, on a private test set of 8 brains with chronic stroke lesions. However the method they use is unclear; the absence of such an explanation suggests a simple linear transformation, perhaps a reflection in the medial (mid-sagittal) plane. Zhang et al. uses a reflection in the medial plane of images that have already been registered to a template (with the registration method unspecified). Evaluations on the BraTS2017 dataset show that adding symmetry features speeds convergence of the algorithm, however there is no consistent improvement in segmentation accuracy. All three methods require homologous voxels to be in the same axial plane, a restriction that our method does not have.

We note that the idea of using symmetry also appeared in early literature, prior to the widespread use of neural networks, for example in (Meier et al., 2014), (Schmidt et al., 2005) and (Dvorak et al., 2013). These works are based on an initial linear registration of each subject's brain to a template. This is also true of (Tustison et al., 2015), with the major differences being the use of multiple modalities and nonlinear registration. Our contributions are thus two-fold: (1) Using template-free registration of an image with a reflected version of itself, called *reflective registration*, to produce supplementary features for use by segmentation algorithms; and (2) demonstrating that *nonlinear* reflective registration is better than linear reflective registration for locating mirror voxels, as judged by improved segmentation results.

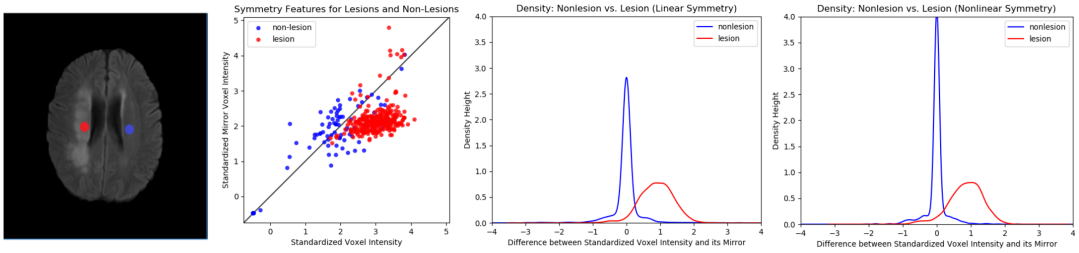

Figure 1: The quasi-symmetry property of the normal brain can be used to aid lesion segmentation. Subfigure (a) shows a lesion voxel (red) and its non-lesion (blue) mirror voxel (projected onto the same axial slice); (b) shows the voxel intensity plotted against the intensity of its mirror for a sample of 600 voxels taken from the same brain and equally divided between lesions and non-lesions; (c) and (d) show superimposed densities for the difference between standardized intensities of a voxel and its mirror, based on a sample of 2000 voxels taken from the same brain and equally divided amongst lesions and non-lesions, where homologous voxels have been identified using linear and nonlinear reflective registration in (c) and (d) respectively.

2 METHODS

Our two baseline algorithms are slight modifications of: (i) TwoPathCNN by Havaei et al. (Havaei et al., 2017), see Figure 2; and (ii) Wider2dSeg by Kamnitsas et al. (Kamnitsas et al., 2017). All of the 2D architectures in Kamnitsas et al. (Kamnitsas et al., 2017), including Wider2dSeg, are two dimensional variants of their 3D deepMedic architecture. The 2D architectures vary in the number of layers, feature maps (FMs) per layer, and FMs per hidden layer. See Table B.1 in Appendix B of Kamnitsas et al. (Kamnitsas et al., 2017) for more details. We first describe the architectures and training of these baseline models, and then describe how to compute and append symmetric features so as to preserve dense inference on 2D images of arbitrary size.

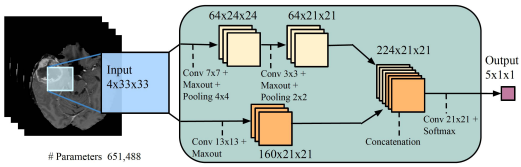

Figure 2: TwoPathCNN architecture, reproduced from Havaei et al. (Havaei et al., 2017) with permission of the authors. Note that the final output is a $2 \times 1 \times 1$ tensor for our application.

2.1 Baseline Models

TwoPathCNN and Wider2dSeg are convolutional neural networks. Both architectures take as input one or, as in the case of Kamnitsas' 2D architectures, two stacks of four patches from different MRI modalities. The networks branch into two pathways. TwoPathCNN consists of three convolutional blocks of sizes shown in Fig. 2, which also shows the locations of maxout and max pooling operations. Wider2dSeg is a deeper architecture with 16 convolutional blocks that makes use of multiscale features through downsampling, convolution and upsampling back to the original scale. This allows for a larger area of information to be used. For more details on the architectures, see (Havaei et al., 2017) and (Kamnitsas et al., 2017). An important feature of these architectures is that all the layers of the network are convolutional, enabling dense inference on full images or image segments.

2.1.1 Training

We interpret the output of the CNNs as predicted label probabilities for lesion, and define a training loss function consisting of the negative log likelihood with both L1 and L2 regularization. This loss is minimised by following a stochastic gradient descent approach on randomly selected minibatches of patches within each brain.

Performance of CNNs depends greatly on the distribution of the training samples. A commonly used approach is to train a classifier on the same number of image patches from each of the classes per minibatch. However since the classes are imbalanced, this approach biases the classifier towards making false positive predictions.

In TwoPathCNN, we follow the two-phase training proposed in [3] with minibatches of one labeled sample per training instance. In the first phase, the mini-batches are equally divided among lesions and backround. In the second phase, we keep the weights of all layers fixed and retrain only the final layer on patches uniformly extracted to be closer to the true

data distribution.

For Wider2dSeg, the patch size fed down each pathway is larger than the network's receptive field (Kamnitsas et al., 2017). This technique, called dense training, increases the effective batch size by constructing mini-batches of more than one labeled sample per training instance since it allows the network to segment a neighborhood of voxels sorrounding the central voxel. (Kamnitsas et al., 2017). This makes the patch size, also called image segment size, an important parameter to tune since larger patch sizes capture more background voxels than smaller patch sizes.

2.2 Implementation Details

We implemented the models using Tensorflow (Abadi et al., 2016). We apply only minimal pre-processing: we normalize within each input channel by subtracting its mean and dividing by its standard deviation. Unlike (Havaei et al., 2017), we did not use N41TK bias correction and we did not remove the 1% highest and lowest intensities. Similarily we didn't use batch normalization as proposed in (Kamnitsas et al., 2017) since it is more of a requirement for 3D architectures. We use standard momentum and fix the momentum coefficient $\mu = 0.6$ throughout training for all architectures.

2.2.1 TwoPathCNN

Weights are initialized from a uniform distribution on $(-0.005, 0.005)$, as in (Havaei et al., 2017), and biases are initialized to zero. Every epoch consists of 10,000 iterations of stochastic gradient descent with momentum on mini-batches of 10 labeled samples. Each sample consists of 4 stacked patches of size 33×33, each patch correspond to a different MRI modality, and a label, which is the ground truth label for the central voxel in the patch. The first phase of training consists of 50,000 iterations or 5 epochs. The minibatches at this stage contain equal number of positive and negative examples. The learning rate is set to 0.001 decays by a factor of 0.1 (Havaei et al., 2017) starting from the third epoch. The second phase of training consists of another 4 epochs of 10,000 iterations each. The minibatches at this stage have the property that approximately 2% of samples presented in them are labeled as negative. The learning rate is reset to 0.001 and decays by a factor of 0.1 after each epoch. Thus, in total, the model is trained on 900,000 samples. The L_1 regularization constant is 10^{-6} and the L_2 regularization constant is 10^{-4}. For further regularization, dropout at a rate of 0.5 was applied on hidden layers of the local pathway. In all of these implementation details, we follow (Havaei et al., 2017), except that the regularization constants were inspired from (Kamnitsas et al., 2017) (which used the same dataset as we do), but changed to account for the increased number of parameters.

2.2.2 Wider2dSeg:

As in (Kamnitsas et al., 2017), we use the weight initiliazation method of He et al. (He et al., 2015), since deeper architectures are prone to larger signal variance. The bias terms are initialized to zero. We used the RMSProp optimizer for a total of 80,000 iterations with a learning rate of 0.001 and decayed it by a factor of 0.5 (Kamnitsas et al., 2017) at the the following iterations: 25,000, 39,000, 49,000, 59,000, 71,000, and 75,000. In contrast to (Kamnitsas et al., 2017), we use an image segment size of 43 for the first pathway and 75 for the second pathway, which segments the 27^2 neighborhood around the central voxel per training instance. Mini-batches are of size 12 and equally divided amongst lesions and background. The batch and image segment sizes were chosen to achieve the same effective batch size shown in Table B.1 of Appendix B from (Kamnitsas et al., 2017). To regularize the network we follow (Kamnitsas et al., 2017) and set the L_1 constant to 10^{-8}, the L_2 constant to 10^{-6}, and apply dropout at a rate of 0.5 on the last two hidden layers.

2.3 Symmetry-augmented methods (LSymm and NLSymm)

For each subject, we augment the four image modalities by four "mirror" images produced as follows. We begin by producing a reflected FLAIR image by reversing the orientation of the x (left-right) axis, using the fslorient tool of FSL (Smith et al., 2001). Since the original images are linearly co-registered, this step is approximately a reflection in the median, i.e. midsagittal, plane. (FLAIR was chosen due to its frequent use in lesion segmentation, however we intend in later work to compare the use of T1 or multiple modalities in this step.) We align the result with the original FLAIR image using either "linear"/affine (LSymm) or nonlinear (NLSymm) registration. This step uses the SynQuick method in the ANTs package (Avants et al., 2009). For LSymm, the "-t a" option was used, giving a 2-stage rigid+affine registration. For NLSymm, the default options were used, giving a 3-stage rigid + affine + nonlinear ("SyN") registration. In either case, the resulting transformation, composed with a reflection, produces a symmetry transformation $T(x,y,z)$ that associates to each voxel its corresponding "mirror" voxel in the opposite hemisphere.

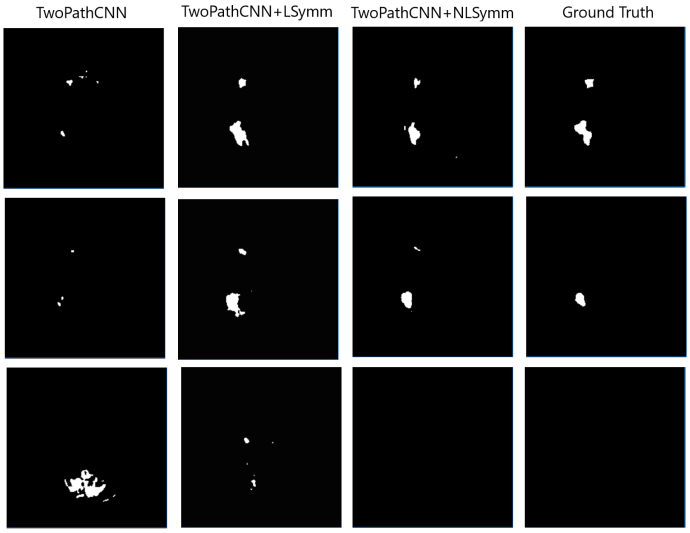

Figure 3: An example of different segmentations of Brain #8 of the SISS2015 Training set from the ISLES2015 Challenge. From left to right, the columns show segmentations produced by the TwoPathCNN, TwoPathCNN+LSymm and TwoPathCNN+NLSymm methods, and the ground truth.

Table 1: Performance of TwoPathCNN and Wider2dSeg based on a 7-fold cross-validation for baseline, NLSymm and LSymm on the ISLES2015 (SISS) training data. Results for Dice, Recall and Precision (Havaei et al., 2017) are shown as mean (std. dev.).

Architecture	Dice	Recall	Precision
TwoPathCNN	0.45(0.25)	0.59(0.22)	0.45(0.29)
TwoPathCNN+LSymm	0.52(0.23)	0.63(0.23)	0.50(0.28)
TwoPathCNN+NLSymm	**0.54(0.21)**	**0.65(0.22)**	**0.52(0.26)**
Wider2dSeg	0.49(0.25)	0.53(0.28)	0.54(0.25)
Wider2dSeg+LSymm	0.61(0.22)	0.58(0.25)	0.67(0.22)
Wider2dSeg+NLSymm	**0.62(0.22)**	**0.60(0.25)**	**0.68(0.21)**

Once we have obtained our linear or nonlinear symmetry transformation for each subject, we use it to construct a Symmetry Difference Image for each modality, by subtracting from each voxel's standardized intensity the standardized intensity of the "mirror" voxel, $S_r(x,y,z) = I_r(x,y,z) - I_r(T(x,y,z))$.

This results in 4 Symmetry Difference Images (SDIs), one for each modality, which we use to augment the original 4 images. For instance in the baseline TwoPathCNN model, for each voxel, one 33×33 patch is extracted from each of the 4 MR images and combined into a $4 \times 33 \times 33$ tensor; in LSymm and NLSymm, one 33×33 patch is extracted from each of the 8 MR images (originals plus SDIs) and combined into a $8 \times 33 \times 33$ tensor. This double-size tensor is fed into the both the local and global pathways of the TwoPathCNN architecture. Apart from doubling the number of images from 4 to 8, all details of architecture and training are exactly as in the baseline methods.

3 EXPERIMENTAL RESULTS

3.1 Dataset

We evaluated our methods on the ISLES2015 (SISS) training data. The training data consists of FLAIR, DWI ,T1 and T1-contrast images of size $230 \times 230 \times 154$, for each of 28 patients with sub-acute ischemic stroke lesions. All images are skull-stripped and have isotropic $1mm^3$ voxel resolution.

3.2 Experiment

For each of the two architectures, we compare the three methods described above: baseline, baseline

with LSymm, and baseline with NLSymm, using 7-fold cross-validation on the Training dataset of 28 subjects. All methods are run with the same hyper-parameters, on the same pseudo-random sequence of training patches.

3.3 Results

Example segmentations produced by the three methods on TwoPathCNN are shown in Fig. 3. The main results are summarised in Table 1. Adding linearly or nonlinearly registered symmetry features (LSymm or NLSymm) to the baseline architectures consistently improves mean Dice coefficient, Recall and Precision, showing the effectiveness of reflective registration. For the Dice coefficient, we performed one-sided paired t-tests for symmetry-augmented vs. baseline methods, and found that the resulting p-values were always less than 10^{-5}, for both LSymm and NLSymm and for both baseline architectures. Moreover, nonlinearly registered symmetry features (NLSymm) consistently produced higher Dice, Recall and Precision scores compared to linearly registered symmetry features ($p < 0.001$ for the TwoPathCNN architecture, and $p = 0.08$ for the Wider2dSeg architecture). Of the two architectures evaluated, Wider2dSeg benefitted more from the symmetry augmentation, however the difference between LSymm and NLSymm was not significant; both differences were perhaps due to its deeper architecture.

4 CONCLUSIONS

We have proposed an improvement to existing segmentation methods by exploiting the bilateral quasi-symmetry of healthy brains. Our method, which does not require a template, consists of augmenting the input images to a CNN with extra Symmetry Difference Images, which are intensity differences between homologous ("mirror") voxels in different hemispheres. We showed how to incorporate these symmetric features into the increasingly popular patch-based CNNs so as to preserve dense inference. In an experiment on the ISLES2015 SISS dataset, we found that adding symmetric features generated using nonlinear reflective registration (the "NLSymm" method) consistently resulted in a mean improvement in Dice coefficient, Recall and Precision. Using linear reflective registration instead gave consistently smaller improvements over baseline, showing that nonlinear registration is superior in this application. For the Dice coefficient, improvement over baseline was significant ($p < 10^{-5}$) for both linear and nonlinear symmetric features. The nonlinear method was significantly better than the linear one ($p < 0.001$) for one baseline architecture (TwoPathCNN) but not the other (Wider2dSeg).

While our numerical results are not directly comparable with those of the three preceding studies of symmetric feature augmentation for CNNs mentioned in the Introduction, we note that our improvements in Dice scores of 9 to 13% on an open dataset compare favourably to earlier results.

We have shown that the brain's quasi-symmetry property is a valuable tool for brain lesion segmentation. The ease of application of symmetry augmentation to most existing CNN methods and many other methods suggests a potentially wide-ranging utility of the method.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Avants, B. B., Tustison, N. J., Song, G., and Gee, J. C. (2009). ANTS: Open-source tools for normalization and neuroanatomy. *Transac Med Imagins Penn Image Comput Sci Lab*.

Dvorak, P., Bartusek, K., and Kropatsch, W. (2013). Automated segmentation of brain tumour edema in flair mri using symmetry and thresholding. *PIERS Proceedings, Stockholm, Sweden*.

Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Pal, C., Jodoin, P.-M., and Larochelle, H. (2017). Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Kamnitsas, K., Ledig, C., Newcombe, V. F., Simpson, J. P., Kane, A. D., Menon, D. K., Rueckert, D., and Glocker, B. (2017). Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78.

Loy, G. and Eklundh, J.-O. (2006). Detecting symmetry and symmetric constellations of features. In *European Conference on Computer Vision*, pages 508–521. Springer.

Maier, O., Menze, B. H., von der Gablentz, J., Häni, L., Heinrich, M. P., Liebrand, M., Winzeck, S., Basit, A., Bentley, P., Chen, L., et al. (2017). ISLES 2015 - a

public evaluation benchmark for ischemic stroke lesion segmentation from multispectral MRI. *Medical image analysis*, 35:250–269.

Meier, R., Bauer, S., Slotboom, J., Wiest, R., and Reyes, M. (2014). Appearance-and context-sensitive features for brain tumor segmentation. *Proceedings of MICCAI BRATS Challenge*, pages 020–026.

Schmidt, M., Levner, I., Greiner, R., Murtha, A., and Bistritz, A. (2005). Segmenting brain tumors using alignment-based features. In *Fourth International Conference on Machine Learning and Applications (ICMLA'05)*, pages 6–pp. IEEE.

Shen, H., Zhang, J., and Zheng, W. (2017). Efficient symmetry-driven fully convolutional network for multimodal brain tumor segmentation. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3864–3868. IEEE.

Smith, S., Bannister, P. R., Beckmann, C., Brady, M., Clare, S., Flitney, D., Hansen, P., Jenkinson, M., Leibovici, D., Ripley, B., et al. (2001). FSL: New tools for functional and structural brain image analysis. *NeuroImage*, 13(6):249.

Tustison, N. J., Shrinidhi, K., Wintermark, M., Durst, C. R., Kandel, B. M., Gee, J. C., Grossman, M. C., and Avants, B. B. (2015). Optimal symmetric multimodal templates and concatenated random forests for supervised brain tumor segmentation (simplified) with antsr. *Neuroinformatics*, 13(2):209–225.

Wang, Y., Katsaggelos, A. K., Wang, X., and Parrish, T. B. (2016). A deep symmetry convnet for stroke lesion segmentation. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 111–115. IEEE.

Winzeck, S., Hakim, A., McKinley, R., Pinto, J. A., Alves, V., Silva, C., Pisov, M., Krivov, E., Belyaev, M., Monteiro, M., et al. (2018). ISLES 2016 and 2017-benchmarking ischemic stroke lesion outcome prediction based on multispectral MRI. *Frontiers in neurology*, 9.

Zhang, H., Zhu, X., and Willke, T. L. (2017). Segmenting brain tumors with symmetry. *arXiv preprint arXiv:1711.06636*. NIPS ML4H Workshop 2017.

Chapter 5

Modelling brain lesion volume in patches with CNN-based Poisson Regression

Modelling brain lesion volume in patches with CNN-based Poisson Regression

Kevin Raina [1] [a]
[1] *Department of Mathematics and Statistics, University of Ottawa, Ontario, Canada*

Keywords: Stroke, Brain Lesions, MRI, Poisson Regression (PR), Convolutional Neural Network.

Abstract: Monitoring the progression of lesions is important for clinical response. Summary statistics such as lesion volume are objective and easy to interpret, which can help clinicians assess lesion growth or decay. CNNs are commonly used in medical image segmentation for their ability to produce useful features within large contexts and their associated efficient iterative patch-based training. Many CNN architectures require hundreds of thousands parameters to yield a good segmentation. In this work, an efficient, computationally inexpensive CNN is implemented to estimate the number of lesion voxels in a predefined patch size from magnetic resonance (MR) images. The output of the CNN is interpreted as the conditional Poisson parameter over the patch, allowing standard mini-batch gradient descent to be employed. The ISLES2015 (SISS) data is used to train and evaluate the model, which by estimating lesion volume from raw features, accurately identified the lesion image with the larger lesion volume for 86% of paired sample patches. An argument for the development and use of estimating lesion volumes to also aid in model selection for segmentation is made.

1 INTRODUCTION

Many segmentation challenges have been undertaken recently, showing the need for automated models in clinical settings (Maier et al., 2017; Winzeck et al., 2018; Bakas et al., 2018). Along with strong predictive power, these challenges stress the importance of fast inference, as lesions can quickly spread. For instance, ischemic stroke lesions cause increasing tissue death within hours of onset, requiring reperfusion therapies around this time. The stroke progresses through acute, sub-acute and chronic stages within days. Gliomas, the most common type of malignant brain tumour, grows at a rate of increasing severity depending on the grade. As the tumour gets larger, symptoms often worsen, reinforcing the need to monitor lesion growth.

A simple, yet powerful, summary statistic is the lesion volume, or when the brain is represented as labelled voxels, the lesion label count. In a clinical study, (Alexander et al., 2010) showed that lesion volume is a significant covariate for understanding ischemic stroke deficits after the initial onset. In application, lesion volume has generally been a dependable factor in the prognosis of ischemic stroke (Merino et al., 2007; Rivers et al., 2006) and multiple sclerosis (Zivadinov et al., 2012; Bagnato et al., 2011). The objectivity of counts enables straightforward inference: a higher lesion label count means the lesion has grown.

In comparison to segmenting entire 3D medical images, directly estimating the number of lesion voxels in the brain from raw features should be expected to require less computational resources since it is no longer necessary to provide detailed information about the lesion's appearance. Nonetheless, in a study by (Erskine et al., 2005) comparing the effects on volume estimation by using different magnetic resonance imaging scanners, lesion volume was estimated from a computer-assisted segmentation tool. Other methods for estimating lesion volume favored a geometric approach, wherein the lesion's surface area per slice is calculated and the estimate is derived by summing across slices (Park et al., 2013; Filippi et al., 1995). In contrast, the output of our proposed statistical direct lesion counting model is a single non-negative integer that doesn't require sophisticated viewing software or significant memory usage.

CNNs have shown promising results in lesion segmentation, as in the works of (Kamnitsas et al., 2017), (Havaei et al., 2017) and (Ronneberger et al., 2015), due to their ability to produce useful features from

[a] https://orcid.org/0000-0002-6240-9675

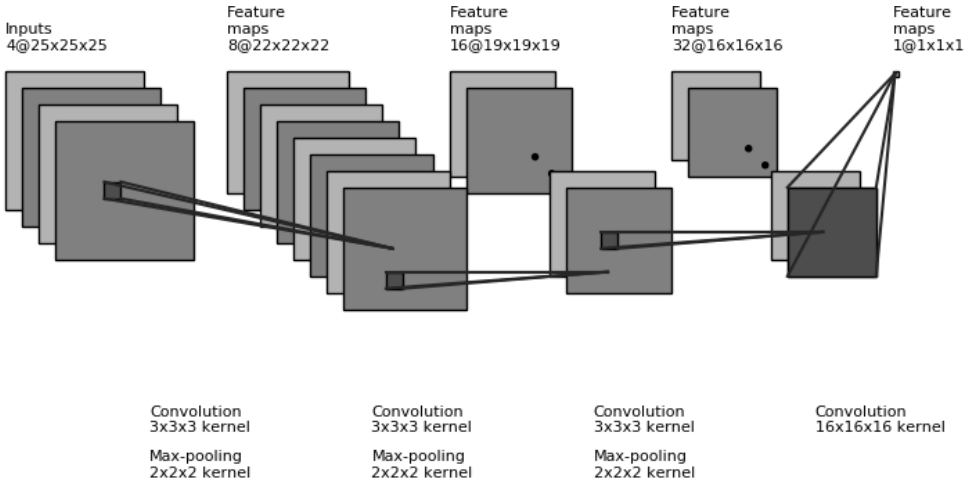

Figure 1: 3D architecture employed for counting lesions. The input tensor is obtained by stacking patches from the patient's brain MRI over 4 different modalities. After applying convolution and pooling operations, the final output is a real number.

large visual spatial contexts combined with efficient iterative patch-based training and dense inference. The output of the CNN is often interpeted as the parameter of a conditional distribution. For instance, in (Kamnitsas et al., 2017; Havaei et al., 2017), the output at each voxel is the parameter of a Bernoulli conditional distribution. The Poisson distribution is generally well known for modelling counts over time and space, and particularly has been applied to modelling the count of multiple sclerosis lesions over time (Altman and Petkau, 2005; Albert et al., 1994). For this reason, we propose the lesion label counts, or equivalently lesion volume, in a predefined patch size is assumed to follow a Poisson distribution conditional on the patch features. The CNNs of (Kamnitsas et al., 2017) and (Havaei et al., 2017) use hundreds of thousands of parameters for segmentation. Using CNNs, coupled with good distributional assumptions, should allow for smaller architectures and faster convergence on the counting task.

One prior related work by (Dubost et al., 2017) used a 3D CNN, similar to U-Net (Ronneberger et al., 2015) to predict global lesion label count, but produces a segmentation when testing by using a removable global pooling layer. A drawback of estimating global lesion label counts is the need for more patient brain images, since an entire brain serves as a single sample. In their study, training was performed on $1,289$ 3D PD-weighted MRI scans, whereas some challenges provide limited training instances (Maier et al., 2017). Another challenge with global lesion counts is being able to efficiently produce scalar outputs from larger 3D information, which would require additional preprocessing and transformations. Estimation of counts in patches can help in these situations.

This paper is organized as follows: Section 2 describes the methods, Section 3 presents the results, and a discussion follows in Section 4.

2 METHODS

2.1 Architecture

The proposed network is shown in Figure 1. As input it stacks $25 \times 25 \times 25$ patches from each MR sequence, runs 3 layers of convolution and max pooling which have sizes $3 \times 3 \times 3$ and $2 \times 2 \times 2$ respectively, followed by a final convolution of size $16 \times 16 \times 16$ to output one real number. In addition to the convolution and max pooling operations, Leaky ReLU nonlinearity was used. It is important to note the number of output activations at each layer are considerably small to reduce the total number of parameters. The patch is not only a parameter of the training features, but is intertwined with the task as well, since it delineates the region over which the lesion label count is estimated.

Training : A block diagram of the methodology is shown in Figure 2. In accordance with the notation of Figure 2, the architecture associates one real number N to each input tensor. Then, the model can be formulated as: $c|X \sim Pois(e^{N(X,\Theta)})$, where c is the lesion label count over the patch, X are the input features used in the architecture, and Θ are the parameters of the

Figure 2: Block diagram representation of the CNN-based Poisson Regression model. The predicted count is obtained by flooring the estimated conditional Poisson parameter (λ).

architecture. In order to train the parameters from observed counts c_i (assumed to Poisson distributed with rate λ_i), mini-batch gradient descent with a batch size b is used to minimize the average negative log-likelihood, $-\frac{1}{b}\sum_{i=1}^{b} log(\frac{\lambda_i^{c_i} e^{-\lambda_i}}{c_i!})$, plus additional L1 and L2 regularization terms to prevent overfitting. The samples used in the mini-batch were taken so as to ensure the lesion count was non-zero by insisting the central voxel be lesion. Since training only samples non-zero counts, it will not be efficient at predicting counts for completely randomly sampled patches, for which zero counts are more frequent. A possible workaround for this task is using a zero-inflated Poisson model (ZIP) (Lambert, 1992), which is suggested for a future study.

2.2 Implementation Details

The open-source software Tensorflow was used to implement the model (Abadi et al., 2016). Non-zero counts were sampled in mini-batches of size 10. Weights were initialized under a Gaussian with mean 0 and standard deviation of 0.001, while biases are initialized to 0. Moreover, the Adam Optimizer was used with initial learning rate of 10^{-4}, and training was stopped when the average cost over $1,000$ iterations increased. This always happened within $15,000$ iterations. In comparison, the segmentation CNN of (Kamnitsas et al., 2017) has default training configurations set to $70,000$ iterations, demonstrating a quick ability to learn for direct counting models. L1 regularization and L2 regularization were used and set to 10^{-8} and 10^{-6} respectively. In addition to regularization, dropout on all hidden layers was employed at a rate ot 0.5. At prediction time, the mean Poisson rate was floored to provide an integer estimate. Table 1 summarizes the implementation details.

Table 1: Numerical summary of implementation details.

Implementation Detail	Value
Batch size (b)	10
Kernel initialization mean(std.)	$0(0.001)$
Learning rate	10^{-4}
L1, L2 coefficients	$10^{-8}, 10^{-6}$
Dropout	0.5

3 Experiments and Results

3.1 Dataset

The architecture and model were trained and evaluated on the ISLES2015 (SISS) training data, which consists of 28 patients with sub-acute ischemic stroke. All data come from the same clinical center, which provided 4 MR sequences for each patient: FLAIR, DWI, T1 and T1-contrast. Images are of size $230 \times 230 \times 153/154$, are processed to not contain the skull and have isotropic $1mm^3$ voxel resolution. Sub-acute ischemic stroke lesions have large variation in size. For instance, in the dataset, the smallest lesion consists of 106 voxels, and the largest consists of $233,547$ voxels. From the 28 brains, 20 were randomly selected to form the training set, and the remaining 8 formed the validation set. This selection was carried out once, and the same split was used across all experiments. It should be acknowledged that the choice of the training and validation split will have an effect on performance due to the aforementioned variability in lesion count.

3.2 Model Performance

To evaluate the performance of the architecture, several metrics are calculated on $10,000$ patches sampled to have a non-zero lesion label count from the validation set. Sampling was done by first selecting the validation brain and then selecting a patch from the brain. Estimated counts that surpassed the possible count in the predefined patch size were adjusted to predict the maximum possible count. In the experiment applied to the ISLES2015 (SISS) data, the mean absolute error (MAE) rounded up to the nearest integer was computed to be $1,458$. In addition, over the same samples the average estimated count to true count ratio was computed to be 1.15. Finally, the mean relative error (MRE) was 0.42 for the ischemic stroke lesions. True patch lesion label counts vary from a few hundred to $15,000$, indicating a promising initial result. Figure 3 plots the estimated and true counts for 200 samples.

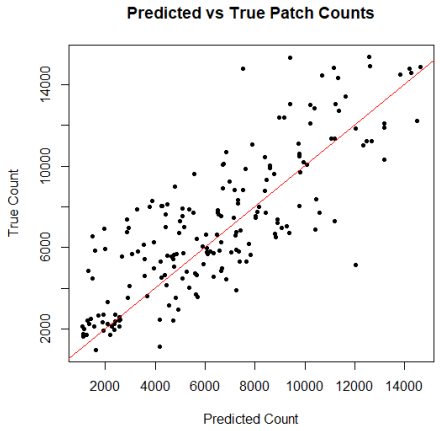

Figure 3: Plot of true count and estimated count for 200 lesion patch samples. Coefficient of multiple correlation (Pearson's correlation coefficient between predicted and actual values) of $R = 0.81$.

3.3 Predicting count order

The second experiment was to order pairs of patches by lesion label count, which can be applied in a clinical setting to compare lesion images over time and assess growth or decay. Given any two image patches containing lesions, the goal is to evaluate the model's ability to identify the image with the larger (equivalently smaller) lesion volume using the proposed estimation. In the experiment, $10,000$ pairs of non-zero count patches are sampled from the validation fold. A sample is counted as correct if the predicted counts preserve the same order as the true counts. Running the experiment on the ISLES2015 (SISS) data gives a correct order prediction for 86 % of samples, demonstrating good ordering capabilities. Figure 4 shows an accurately predicted sample.

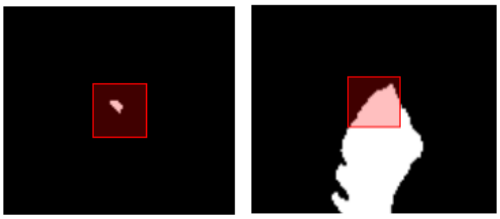

Figure 4: Example of predicting count order, where the red square outline represents the middle slice of the 25^3 patch. The true counts from left to right are: 356 and 5297. The predicted counts from left to right are: 501 and 5640. In this case, the model accurately identifies the left image as having the smaller lesion volume.

4 FUTURE WORK

4.1 Extension to Arbitrary Patches

The analysis and modelling undertaken in the previous sections were done on patches that contain lesion voxels. That is, the patches were sampled to contain lesion voxels. Although it was shown that this has the potential for modelling lesion growth or decay by first estimating lesion volume, relaxing this restriction allows for the prediction of counts in arbitrary patches, which are more frequently zero counts. Due to the unbalanced nature of the data, one proposition for a future study is to combine a zero-inflated Poisson model which is known to account for excess zero counts in data, with CNNs.

4.2 Location Detection

Being able to predict counts in arbitrary patches can form the basis for lesion location detection. A possible algorithm could be to randomly sample patches from one brain, predict their counts, and record the central voxel position for the patch with the maximum predicted count. The voxel position returned by the algorithm should identify an area of significant lesion presence in the brain, assuming the model is well-tuned. Though simple, the algorithm is theoretically stable when applied on a single lesion, since applying true counts in place of predictions will always return the location of a lesion provided a lesion is present.

Rather than recording just the maximum predicted count from the given sample, another option would be to order the predicted counts for samples drawn from one brain. From a pre-defined quantile, larger counts a long with their central voxel positions will delineate the lesion.

4.3 Model Selection for Segmentation

The size of a lesion is one of the factors that can allow some configurations of a segmentation algorithm to perform better than others (Kamnitsas et al., 2017). Smaller lesions, often found in sub-acute ischemic stroke, are usually tougher to segment and produce lower dice coefficients (Havaei et al., 2017) than larger lesions due to a relatively higher number of false positives compared to true positives. For CNNs, configurations mainly pertain to the architecture employed. Given that some architectures may segment smaller lesions better, one option is to regress the architecture employed for a brain, on an estimate of the global lesion label count in the brain.

5 DISCUSSION

Direct counting models might be useful in clinical settings. Some potential examples are but not limited to: monitoring and locating lesions. They can also aid segmentation by selecting configurations of segmentation algorithms based on global lesion size, from raw data, and reducing the required segmentation area through lesion detection. Further developments need to be made to predict patch counts, including: improving accuracy in the predictions of non-zero counts, accounting for highly imbalanced zero counts, developing sampling-based algorithms for lesion location detection, and providing aggregate patch measures to predict global lesion count. This will increase the effectiveness and broaden the applications of direct counting models.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Albert, P. S., McFarland, H. F., Smith, M. E., and Frank, J. A. (1994). Time series for modelling counts from a relapsing-remitting disease: application to modelling disease activity in multiple sclerosis. *Statistics in Medicine*, 13(5-7):453–466.

Alexander, L. D., Black, S. E., Gao, F., Szilagyi, G., Danells, C. J., and McIlroy, W. E. (2010). Correlating lesion size and location to deficits after ischemic stroke: the influence of accounting for altered perinecrotic tissue and incidental silent infarcts. *Behavioral and brain functions*, 6(1):6.

Altman, R. M. and Petkau, A. J. (2005). Application of hidden markov models to multiple sclerosis lesion count data. *Statistics in Medicine*, 24(15):2335–2344.

Bagnato, F., Ikonomidou, V. N., van Gelderen, P., Auh, S., Hanafy, J., Cantor, F. K., Ohayon, J., Richert, N., and Duyn, J. (2011). Lesions by tissue specific imaging characterize multiple sclerosis patients with more advanced disease. *Multiple Sclerosis Journal*, 17(12):1424–1431.

Bakas, S., Reyes, M., Jakab, A., Bauer, S., Rempfler, M., Crimi, A., Shinohara, R. T., Berger, C., Ha, S. M., Rozycki, M., et al. (2018). Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the brats challenge. *arXiv preprint arXiv:1811.02629*.

Dubost, F., Bortsova, G., Adams, H., Ikram, A., Niessen, W. J., Vernooij, M., and De Bruijne, M. (2017). Gp-unet: Lesion detection from weak labels with a 3d regression network. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 214–221. Springer.

Erskine, M., Cook, L., Riddle, K., Mitchell, J. R., and Karlik, S. J. (2005). Resolution-dependent estimates of multiple sclerosis lesion loads. *Canadian journal of neurological sciences*, 32(2):205–212.

Filippi, M., Horsfield, M., Campi, A., Mammi, S., Pereira, C., and Comi, G. (1995). Resolution-dependent estimates of lesion volumes in magnetic resonance imaging studies of the brain in multiple sclerosis. *Annals of Neurology: Official Journal of the American Neurological Association and the Child Neurology Society*, 38(5):749–754.

Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Pal, C., Jodoin, P.-M., and Larochelle, H. (2017). Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31.

Kamnitsas, K., Ledig, C., Newcombe, V. F., Simpson, J. P., Kane, A. D., Menon, D. K., Rueckert, D., and Glocker, B. (2017). Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78.

Lambert, D. (1992). Zero-inflated poisson regression, with an application to defects in manufacturing. *Technometrics*, 34(1):1–14.

Maier, O., Menze, B. H., von der Gablentz, J., Häni, L., Heinrich, M. P., Liebrand, M., Winzeck, S., Basit, A., Bentley, P., Chen, L., et al. (2017). ISLES 2015 - a public evaluation benchmark for ischemic stroke lesion segmentation from multispectral MRI. *Medical image analysis*, 35:250–269.

Merino, J. G., Latour, L. L., Todd, J. W., Luby, M., Schellinger, P. D., Kang, D.-W., and Warach, S. (2007). Lesion volume change after treatment with tissue plasminogen activator can discriminate clinical responders from nonresponders. *Stroke*, 38(11):2919–2923.

Park, H.-J., Machado, A. G., Cooperrider, J., Truong-Furmaga, H., Johnson, M., Krishna, V., Chen, Z., and Gale, J. T. (2013). Semi-automated method for estimating lesion volumes. *Journal of neuroscience methods*, 213(1):76–83.

Rivers, C., Wardlaw, J., Armitage, P., Bastin, M., Carpenter, T., Cvoro, V., Hand, P., and Dennis, M. (2006). Do acute diffusion-and perfusion-weighted mri lesions identify final infarct volume in ischemic stroke? *Stroke*, 37(1):98–104.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

Winzeck, S., Hakim, A., McKinley, R., Pinto, J. A., Alves, V., Silva, C., Pisov, M., Krivov, E., Belyaev, M., Monteiro, M., et al. (2018). ISLES 2016 and 2017-benchmarking ischemic stroke lesion outcome prediction based on multispectral MRI. *Frontiers in neurology*, 9.

Zivadinov, R., Heininen-Brown, M., Schirda, C. V., Poloni, G. U., Bergsland, N., Magnano, C. R., Durfee, J.,

Kennedy, C., Carl, E., Hagemeier, J., et al. (2012). Abnormal subcortical deep-gray matter susceptibility-weighted imaging filtered phase measurements in patients with multiple sclerosis: a case-control study. *Neuroimage*, 59(1):331–339.

Chapter 6

Statistical inference of the inter-sample Dice distribution for discriminative CNN brain lesion segmentation models

6.1 Introduction

Discriminative CNNs, such as those constructed by [64, 59, 94], have consistently ranked on the top of the leaderboard in many brain lesion segmentation challenges [77, 119, 22]. Commonly, they are formulated by assuming a voxel-wise multinoulli distribution, conditional on the MRI intensities of neighboring voxels where the parameters are obtained and estimated with a CNN. For the case of 2 labels, if x_j denotes the 3D MRI intensities used in the prediction of voxel j over a discrete domain $\Omega \subset \mathbb{R}^3$, Y_j represents the random label at voxel j, and Θ are the parameters of a CNN architecture, then the model is formulated as

$$Y_j|x_j \sim Bernoulli(\pi_j = CNN(\Theta, x_j)). \tag{6.1.1}$$

At prediction time, the translational invariance property of the CNN allows for a fast way to estimate all the voxel-wise conditional distributions, without having to separately feed their respective covariates. When a model is fully trained, the parameters in (6.1.1) are replaced with an estimate: $\hat{\Theta}$. In this way, discriminative CNNs are capable of sampling segmentations by sampling the associated label at each voxel, conditionally independent of each other. Despite this source of variability, a decision rule of selecting the label with the highest probability is often applied rendering the segmentation deterministic.

Measuring the performance variation of trained discriminative CNNs in brain lesion segmentation on a patient by patient basis can help clinical practitioners instill a degree of confidence in the use of automated segmentation methods. One metric of model performance is the Dice coefficient [46, 106], which measures the similarity between two 3D voxel-wise labelled images: S_1 and S_2. The Dice coefficient is a real number in $[0, 1]$ given by

$$Dice(S_1, S_2) = \frac{2TP}{2TP + FN + FP}, \tag{6.1.2}$$

where a value of 0 indicates no overlap and a value of 1 indicates perfect similarity. TP, FP and FN refer to the number of true positive, false positive, and false negative voxels in the medical image respectively. Formulating this incentive mathematically, if D denotes the Dice coefficient random variable which incorporates randomness across patients, and across model segmentations, and X_p collects all the covariates in (6.1.1) accross all voxels in the image for patient p to from a high dimensional vector, then the interest lies in measuring

$$Var(D|X_p, \hat{\Theta}). \tag{6.1.3}$$

Here the Dice random variable is theoretically measured on the true segmentation, which is never known, and the random segmentations produced by a trained discriminative CNN model on the particular patient. Intuitively, the observations of D can be generated by: observing a new random patient, randomly generating a segmentation from the trained discriminative CNN given their radiological features, and calculating the Dice coefficient against the ground truth. In (6.1.3), the first source of variation is eliminated by conditioning on the patient. Estimating (6.1.3) can also help suggest the complete abandonment of an algorithm in favor of another, as some CNNs may be better tuned to deal with specific brain lesion characterstics like small lesion volume.

Despite these advantages, some brain lesion segmentation challenges compute variability in the Dice metric accross patient predictions [77]. Moreover, for a particular patient, only a single deterministic segmentation is produced by some decision rule. That is, $Var(D|\hat{\Theta})$ becomes the measure to be estimated, but this estimate does not account for model variability. It is important to note this measure is unconditional on the patient's radiological features, and thus inherently incorporates additional variability. The merit of this metric is in its applicability, as not all segmentation algorithm are capable of sampling segmentations. However in the case of discriminative CNNs, by its very nature, this metric can solely be used to construct a one size fits all confidence interval and could completely discard a model that may in fact perform well on certain kinds of patients. Another disadvantage is that this metric is obtained from the training set, and then must be generalized to arbitrary cases, which may present significant differences. Analysis of model performance variance on a single patient (6.1.3), conditional on their radiological features, eliminates these extra sources of variation, and can be more useful in clinical practice by providing specialized patient care.

One step towards this direction was taken in the BraTS 2019 challenge, which is an extension of [22]. In particular, voxel-wise measures of segmentation uncertainty ranging from 0 to 100 were calculated from 3D MRIs for all patients individually.

Then, at specified thresholds, uncertain voxels were filtered out in the calculation of the Dice coefficient. Depending on the structure of uncertainties, this method can reward or penalize the Dice score, but this can never be confirmed in practice since we never know the ground truth segmentation.

Segmentation sampling and consequent analysis of the inter-sample Dice distribution has been undertaken for generative models, for instance, by Lê et al. [70]. In their work, they use a Gaussian process to produce segmentation samples based on a single expert manual segmentation of grade 4 gliomas. The mean of the inter-sample Dice distribution and variability of the segmentations can be controlled by a single model parameter. The samples can then be used in radiotherapy planning by delivering radiation to certain voxels and avoiding dose to uncertain voxels, where perhaps there are more sensitive tissues.

The contribution mentions the applicability of their method in evaluating uncertainty in the performance of segmentation algorithms, by repeatedly sampling segmentations off the ground truth and calculating the variability against a deterministic predicted segmentation. Though this method can be applied to arbitrary segmentation algorithms, the source of variation is not produced by the model predictions, as in the case of discriminative CNNs. As a consequence, the method is effective in assessing the effectiveness of a particular segmentation produced from a model, but not the model itself. Another related work by Roy et al. [95], shows that the inter-sample Dice coefficient correlates with Dice performance using a discriminative CNN for the segmentation of brain scans from children with phsychiatric disorders (CANDI-13 dataset), and suggests it as a measure for quantifying uncertainty. Figure 6.1 shows the correlation of the metrics organized by brain region.

In this proposed work, the inter-sample Dice observations are shown to be independent and identically distributed samples from a distribution with finite variance and mean, under certain conditions. Segmentations are sampled directly from a discriminative CNN. The mean with a confidence interval of the inter-sample Dice

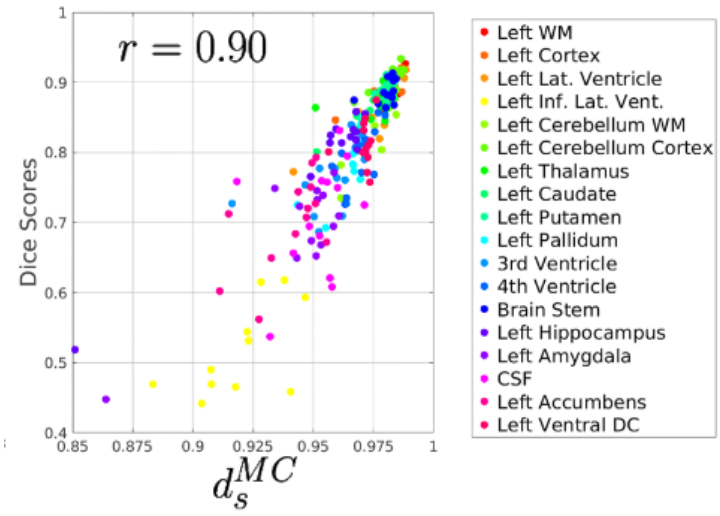

Figure 6.1: Correlation of inter-sample Dice coefficient and Dice performance against ground truth on the CANDI-13 dataset from Roy et al. [95].

distribution is then estimated by the central limit theorem and used in place of (6.1.3) to decide on whether to reject or accept the CNN model on patients with ischemic stroke. This chapter is organized as follows: Section 2 describes the methods, Section 3 presents the results, and a discussion follows in Section 4.

6.2 Methods

6.2.1 Architecture

The CNN architecture considered is Wider2DSeg, which was originally constructed by Kamnitsas et al. [64] and is a two dimensional variant of their 3D deepMedic architecture. Raina et al. [90] tuned this architecture in junction with additional symmetry covariates obtained from a reflective registration step to yield a 0.62 Dice coefficient over a 7-fold cross validation on the ISLES 2015 SISS dataset. All the implementation details in this work are exactly as in the Wider2DSeg implementation of Raina et al. [90] with nonlinear symmetry covariates.

6.2.2 Segmentation sampling

Referencing eqaution (6.1.1), suppose there are V voxels in an image. The attention is restricted to the case of 2 voxel labels, though this analysis can extend to an arbitrary finite number of labels. Let $S \in \mathbb{R}^V$ represent the segmentation random vector of a trained discriminative CNN model, where each element $S_k = Y_j$, for unique $j \in \Omega$. The segmentation vector is unconditional and incorporates randomness across patients. Although in what follows it does not matter which permutation in the assignment of voxels to array elements is used, the assignment must be chosen and fixed. Then, denote X as a patient's radiological features as formulated in equation (6.1.3) , and $\hat{\Theta}$ as the estimated parameters of the CNN. Segmentations can be sampled conditional on a patient's radiological features if conditional independence is assumed accross voxels. In this case, the distribution is completely known by

$$S|X \sim f(s; X, \hat{\Theta}) = \prod_{j=1}^{V} \pi_j(x_j, \hat{\Theta})^{s_j}(1 - \pi_j(x_j, \hat{\Theta}))^{1-s_j}. \tag{6.2.1}$$

6.2.3 Inter-sample Dice distribution

Let S_1^* and S_2^* represent two iid model segmentation random vectors for a given patient, each distributed according to (6.2.1). By independence, the joint distribution of these samples is given by

$$(S_1^*, S_2^*) \sim f(s_1^*; X, \hat{\Theta}) f(s_2^*; X, \hat{\Theta}). \tag{6.2.2}$$

In this way, the inter-sample Dice random variable is just a function of the random vectors. In particular, if S_2^* is seen as the ground truth, then the Dice coefficient of S_1^* can be computed against it, and vice-versa. Define Γ to be the inter-sample Dice random variable. Then

$$\Gamma = Dice(S_1^*, S_2^*). \tag{6.2.3}$$

In fact, the complete distribution of Γ can be obtained, but can be quite computationally expensive as the number of voxels are often large. That is, if $D_{(\cdot)}$ represents the support of a random variable, then $|D_{S_1^*}| \leq 2^V$, where the upper bound is obtained by having $0 < \pi_j < 1$ for all voxels in the image. By independence of S_1^* and S_2^*, this further implies that $|D_{(S_1^*, S_2^*)}| \leq 2^{2V}$. Subsequently, after applying the Dice transformation (6.2.3) which may at most associate a unique Dice value for each element in $D_{(S_1^*, S_2^*)}$, the inequality $|D_\Gamma| \leq 2^{2V}$ is obtained. Hence, computing the exact Dice distribution may require the calculation of at most 2^{2V} probabilties, where V tends to be in the millions. Though, since D_Γ is bounded by 2^{2V}, Γ has a discrete distribution with a finite and countable support.

In order to conduct statistical inference on $E[\Gamma]$ it must also be finite, however, this is not the case. Notice that if $(S_1^*, S_2^*) = (0, 0)$, Γ is undefined, and this outcome always has a non-zero probability. In fact, this is the only way for Γ to be undefined. To correct this, two necessary modifications are introduced: 1) $\exists\ \pi_j \neq 0$, for some j and 2) The inter-sample Dice random variable is redefined as

$$\Gamma^* = \Gamma | (S_1^*, S_2^*) \neq (0, 0). \tag{6.2.4}$$

The first condition ensures that $(0, 0)$ can never be the only sample generated, thereby permitting its removal and still retaining a distribution through the redefinition in the second condition. The second condition redefines the inter-sample Dice random variable to be conditional on all samples drawn from (6.2.2) except $(0, 0)$. Now, $\forall \gamma \in D_{\Gamma^*}, 0 \leq \gamma \leq 1$, where $D_{\Gamma^*} \neq \emptyset$. Then

$$E[(\Gamma^*)^2] < \infty. \tag{6.2.5}$$

As a consequence, sampling observations from this distribution, and estimating its mean from a sample can be used. Independent and identically distributed pairs of segmentations can be sampled from (6.2.2), and can generate an iid sample of inter-sample Dice observations. To adjust for the second modification, any $(0, 0)$ sample, or equivalently undefined Dice score, is removed in the sampling phase. Moreover, since the inter-sample Dice distribution has a finite mean and variance, the central limit theorem can be applied, and associated confidence intervals can be constructed.

Computationally, this method can be undertaken by first producing the estimated probability tensors. At each voxel in the probability tensor are the estimated label probabilities obtained from the features and CNN parameter estimates. Once this tensor is calculated, segmentations are sampled by sampling from the associated multinoulli distribution and iterating over all voxels. Naturally, these will produce segmentation samples that can be appropriately viewed as such since the format of the probability tensors align with that of the actual image.

6.2.4 Decision rule

Let $\gamma_1, .., \gamma_n$ be an iid sample of realized inter-sample Dice coefficients for a given patient, and construct a $(1 - \alpha)\%$ approximate confidence interval, based on the central limit theorem. For a specified threshold, reject the use of a discriminative CNN model on a patient if the confidence interval is entirely below the threshold. In this manner, the clinician can justify with $(1 - \alpha)\%$ confidence that the true mean of the inter-sample Dice distribution is below the given threshold, and appropriate actions to reject the model in favor of another can be undertaken.

6.3 Experiments and results

6.3.1 Dataset

The discriminative CNN model is trained and evaluated on the ISLES2015 (SISS) training data, which consists of 28 patients with sub-acute ischemic stroke. The radiological features X for a patient are 4 MR sequences: FLAIR, DWI, T1 and T1-contrast. Each image has a total of $230 \times 230 \times 154$ voxels. At each voxel, there are only two possible labels classes: lesion or non-lesion.

6.3.2 Efficacy of decision rule

The proposed segmentation sampling based decision rule is applied to the results of Raina et al. [90], which yielded an average Dice coefficient (predicted against ground truth) of 0.62 over a 7-fold cross-validation from single deterministic segmentations obtained by selecting the label with the highest probability. For each fold in the cross-validation and for each brain in the validation fold, 30 inter-sample Dice observations are sampled, and the central limit theorem confidence interval is computed. Table 6.1 displays the patient by patient results. The Pearson correlation between mean ISD and Dice over the 28 patients was computed to be $r = 0.81$. Figure 6.2 plots the mean ISD against Dice score for each patient, and depicts the fitted regression line. The threshold was set to 0.85 with reference to the regression line in Figure 6.2, and corresponds to a Dice score of 0.60. In addition, the confidence level is 95%. By removing the rejected predictions, the average Dice coefficient increased from 0.62 to 0.74.

Table 6.1: Case by case hypothesis testing for Wider2dSeg on ISLES2015 SISS. The inter-sample Dice confidence interval is computed using the central limit theorem with 30 samples.

Patient No.	ISD confidence	Dice	Rejection Status
1	0.940339 ± 0.000116	0.866798	Fail to Reject
2	0.955828 ± 0.000243	0.815299	Fail to Reject
3	0.897922 ± 0.001002	0.736231	Fail to Reject
4	0.946641 ± 0.000122	0.797607	Fail to Reject
5	0.944956 ± 0.000133	0.857821	Fail to Reject
6	0.965364 ± 0.000146	0.905630	Fail to Reject
7	0.943856 ± 0.000178	0.822416	Fail to Reject
8	0.909659 ± 0.000349	0.702697	Fail to Reject
9	0.969197 ± 0.000076	0.854602	Fail to Reject
10	0.875187 ± 0.000340	0.592438	Fail to Reject
11	0.933384 ± 0.000399	0.775896	Fail to Reject
12	0.888211 ± 0.000810	0.517170	Fail to Reject
13	0.850324 ± 0.001449	0.277828	Fail to Reject
14	0.980536 ± 0.000095	0.815283	Fail to Reject
15	0.974488 ± 0.000179	0.887611	Fail to Reject
16	**0.580227 ± 0.004708**	**0.009584**	**Reject**
17	**0.508344 ± 0.005390**	**0.164190**	**Reject**
18	0.949571 ± 0.000435	0.730066	Fail to Reject
19	**0.730039 ± 0.003967**	**0.487031**	**Reject**
20	0.950651 ± 0.000441	0.795393	Fail to Reject
21	**0.687765 ± 0.003033**	**0.434674**	**Reject**
22	0.863127 ± 0.000823	0.685439	Fail to Reject
23	**0.754482 ± 0.002522**	**0.634320**	**Reject**
24	0.861986 ± 0.001278	0.507946	Fail to Reject
25	0.916227 ± 0.001130	0.761670	Fail to Reject
26	**0.598502 ± 0.006355**	**0.191637**	**Reject**
27	**0.824157 ± 0.000633**	**0.000156**	**Reject**
28	0.887825 ± 0.001183	0.732581	Fail toReject

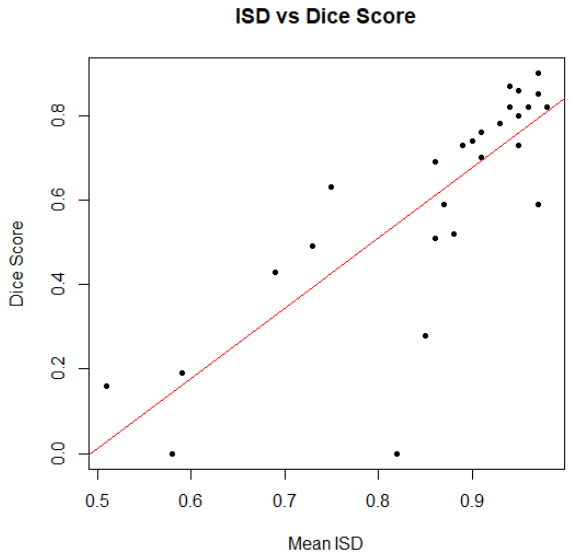

Figure 6.2: Plot of mean ISD against Dice score with regression line: $y = 1.66x - 0.82$, where y is Dice score and x is mean ISD. The F-test yielded a p-value of 1.4×10^{-7}.

6.4 Limitations and future work

The reason as to why ISD is highly correlated to Dice performance for discriminative CNNs, and can be used to detect weak segmentations is not entirely clear. As a counter-example, consider a model that always predicts $P(lesion) = 0$ for all but one voxel, which instead has $P(lesion) = 1$. Then, the ISD would always be 1, since the model produces exactly the same segmentation with probability 1, regardless of the input. However, it would be unexpected to see a high Dice score for this model. The coupling of CNN outputs and ISD for detecting uncertain segmentations requires further investigation for a deeper understanding of its performance.

One important remark is that the Dice metric can be substituted by other metrics such as the sensitivity, specificity, mean squared error, or precision and the preceding analysis would also follow for these distributions, thereby permitting hypothesis testing. Moreover, the computations considered in this work were over the entire brain,

but could also be calculated on specific regions of interest (ROIs). Deciding on which metrics to use, and applying them to more detailed brain sub-regions could improve the decision-making potential, and is a possible area of development.

Another point to remark is that the proposed method can be used to rigorously test competing discriminative models based on their respective inter-sample mean Dice confidence intervals, and select the most robust one on an individualized patient basis. Applying this unifying technique for all competing CNNs in a brain lesion challenge may exhibit the best possible performance, without any consideration to the ground truth. Segmentation challenges have recently begun to incorporate uncertainty analysis, but further work is required to apply these techniques on various types of brain lesion structures.

Chapter 7

Unsupervised invariant information clustering applied to MRI brain lesion segmentation

7.1 Introduction

The need for large and costly annotated datasets by supervised deep learning segmentation methods affects their efficiency and applicability [63]. In such cases unsupervised algorithms, which don't use any labels, are useful. Efforts have been made to combine deep learning with unsupervised clustering algorithms [122, 57, 30], but often this combination leads to degenerate solutions, which is the tendency for all data samples to be classified into a single cluster. Caron et al. [30] reasons that solutions are typically based on penalizing the number of data samples per cluster. Over smaller datasets, these terms can be computed easily, but are not applicable in the training of large-scale datasets that CNNs are often applied to. In addition, some distributions are dominated by only a few classes, which leads the CNN to exclusively discriminate between them. To prevent this issue, many artificial pre-

processing and post-training steps have been developed [122, 30, 47, 48]. Invariant information clustering (IIC), proposed by Ji et al. [63], is an unsupervised image segmentation method designed to deal with solution degeneracy. The basic idea is to train on pairs of similar inputs, and maximize mutual information between the (soft) labels assigned to the two elements of each pair. In Ji et al. [63] and in the present work, the inputs are images and labels are assigned by a CNN. Although Ji et al. [63] showed the efficacy of IIC on RGB images, in this section we show that applying IIC to MR brain lesion segmentation is more challenging due to image contrast, and suggest methods for improvement.

7.2 Method

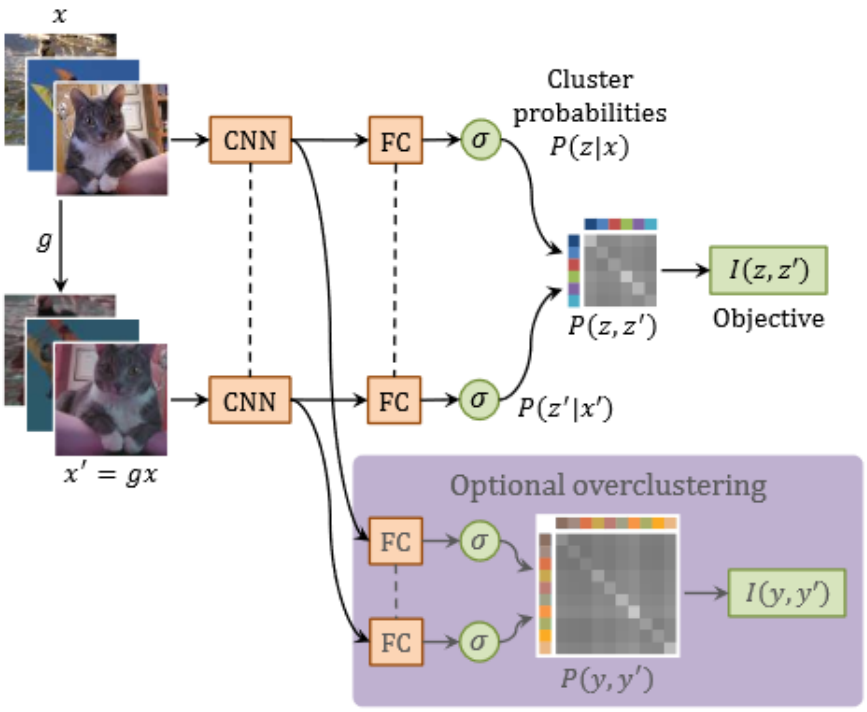

Figure 7.1: IIC framework [63] .

As per the notation of Ji et al., let $x \in \mathcal{X}$ be image features used by a CNN

for classification. In addition, let $x' \in \mathcal{X}$ be the result of a transformation of the image x that is considered to leave the content of the image intact, i.e. the content is invariant to the transformation. For example, the transformation could be scaling, skewing, rotation, flipping, or changing color contrast. The pair of random vectors (x, x') can be sampled from their joint distribution $P(x, x')$. The marginal distributions of x and x' are identical (if we ignore image boundary effects), but the variables are highly dependent, in fact x' depends deterministically on x. In the case of 2D segmentation, x is usually an image patch centered on a voxel $u \in \{1, .., H\} \times \{1, .., W\}$. If $t \in T \subset \mathbb{Z}^2$ is a small 2 dimensional shift vector, then x' can be the image patch centered on a neighboring voxel $u + t$. Thus, synthetic perturbations are not the only way of generating pairs of images in segmentation. In particular, the segmentation framework provides access to the spatial relationship between patches, and local spatial invariance can also be included. In fact, Ji et al. [63] experimented with incorporating both synthetic and spatial transformations simultaneously. Moreover, for a classification of C classes, let $\phi(x) : \mathcal{X} \to \{1, .., C\}$ represent the CNN architecture applied to image feature x. Note that the parameters of the CNN, Θ, are hidden in the notation. Since $\phi(x) \in [0, 1]^C$ by construction, by referencing chapter 3 it can be interpreted as the distribution of a discrete random variable Z over the C classes, given the image features. In particular, $P(Z = c|x) = \phi_c(x)$. However, in IIC we are also interested in the unconditioned random variable Z whose distribution is the output of the CNN based on the random input vector x. Similarly, Z' is the random variable whose distribution is the output of the CNN associated with inputs x'. By conditional independence the conditional joint distribution is given by $P(Z = c, Z' = c'|x, x') = \phi_c(x)\phi_c(x')$. It is important to note that despite the conditional independence, Z and Z' are dependent on each other when marginalizing over the paired feature random vectors (x, x'). That is

$$p(z, z') = \int_x \int_{x'} p(z, z', x, x')dx'dx = \int_x \int_{x'} p(x, x')p(z, z'|x, x')dx'dx.$$

In practice, the joint distribution is estimated from a batch of observations $\{(x_i, x_i')\}_{i=1}^n$, which are treated as having a discrete distribution with mass $1/n$ at each observation. The discrete version of the above marginalization is

$$p(z, z') = \sum_{i=1}^n p(x_i, x_i')p(z, z'|x_i, x_i') = \frac{1}{n}\sum_{i=1}^n p(z, z'|x_i, x_i').$$

Ji et al. transforms this equation into a matrix formulation for an easier computational implementation. In particular the probability distribution of (Z, Z') can be re-written as the sum of matrices P_i obtained from the outer product of the vectors $\phi(x)$ and $\phi(x')$:

$$P_i = \phi(x_i)\phi(x_i')^T = [P(Z = c, Z' = c'|x_i, x_i')]_{(c,c')\in\{1..C\}\times\{1..C\}}.$$

Thus, $p(z, z')$ can be reformulated as

$$P = [P(Z = c, Z' = c')]_{(c,c')\in\{1..C\}\times\{1..C\}} = \frac{1}{n}\sum_{i=1}^n Pi.$$

The matrix P is then symmetrized by replacing it with $(P+P^T)/2$, and the marginals $P_c = P(Z = c)$ and $P_c' = P(Z = c')$ can then be obtained by summing accross rows or columns. Then, the mutual information between Z and Z' is defined as

$$I(Z; Z') = I(P) = \sum_{c=1}^C \sum_{c'=1}^C P_{c,c'} ln(\frac{P_{c,c'}}{P_c P_{c'}}),$$

and is a non-negative [43] measure of dependence between the random variables. In addition, the mutual information is 0 if and only if Z and Z' are independent [43]. The goal of IIC is to adapt the parameters of the CNN in order to maximize the mutual information or dependency in the classification of two variants of the same image so as to learn differences that may arise in a similarly labeled images. By comparing an image with a transformed version of itself, the method eliminates the need for labels. Thus, notationally, the objective can be written as

$$\max_{\phi} I(\phi(x), \phi(x').$$

Mutual information can be expanded into the difference of two terms: $I(z, z') = H(z) - H(z|z')$. In particular, $H(z)$ is the individual cluster assignment entropy and $H(z|z')$ is the conditional cluster assignment entropy. Hence, maximizing mutual information is equivalent to maximizing $H(z)$ and minimizing $H(z|z')$ simultaneously. However, this dual optimization presents a trade-off. $H(z)$ is maximized when all clusters are equally likely to be picked, while $H(z|z')$ is minimized when the cluster assignments are entirely predictable from each other. This leads to the balancing of mass equalization and prediction reinforcement for avoiding degeneracy.

7.3 Implementation details

The model was implemented using Tensorflow [10] (See Appendix for the IIC code). Minimal pre-processing is applied by normalizing within each input channel by subtracting its mean and dividing by its standard deviation. Batch normalization was not used as proposed in [64], since it is more of a requirement for 3D architectures. Standard momentum and a fixed momentum coefficient $\mu = 0.6$ was used throughout training for all architectures. As in Kamnitsas et al., the weight initialization method of He et al. [60] is used, since deeper architectures are prone to larger signal variance. The bias terms are initialized to zero. The Adam optimizer ran for a total of $20,000$ iterations with a learning rate of 0.0001 before it converged. Following Kamnitsas et al., we use an image segment size of 25 for the first pathway and 57 for the second pathway, which segments the 9^2 neighborhood around the central voxel per training instance. Mini-batches are of size 10 and randomly sampled from cropped images. To regularize the network we follow Kamnitsas et al., and set the L1 constant to 10^{-8}, the L2 constant to 10^{-6}, and apply dropout at a rate of 0.5 on the last two hidden layers. The last fully convolutional layer is treated as the single sub-head ($h = 1$), and there are no sample repeats[63]. Furthermore, the shift size was set to $t = 2$, and $k = 10$ classes were assumed with the intention of segmenting the lesion, but also identifying

CSF, gray matter, white matter, and any other undiscovered but prominent clusters. No synthetic transformations were applied for these experiments.

7.4 Experiments and results

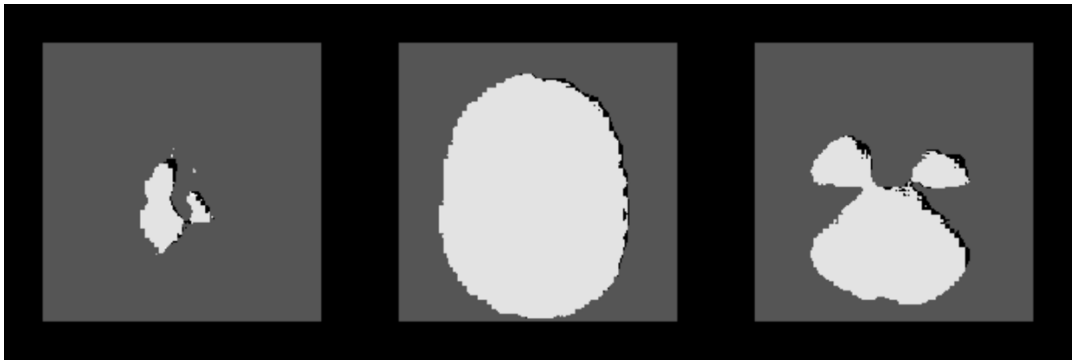

Figure 7.2: IIC segmentation results for selected slices from a patient. Only three clusters were discovered: background (dark-gray), black, and brain (light-gray).

7.4.1 Dataset

IIC was experimented on the ISLES2015 (SISS) training data. The training data consists of FLAIR, DWI ,T1 and T1-contrast images of size $230 \times 230 \times 154$, for each of 28 patients with sub-acute ischemic stroke lesions. All images are skull-stripped and have isotropic $1mm^3$ voxel resolution.

7.4.2 Results and discussion

As can be seen in Fig 7.2, IIC was mainly able to separate brain tissue from background, while also outlining a third unknown cluster. Lesions have very similar intensities to healthy tissue, and IIC was not able to differentiate this. Image preprocessing methods amongst different MRI modalities could help to improve contrast

and segmentation results. For instance, Harris et al. [58] noted that CSF pixels appear brighter in T2 images, and in proton-weighted images they appear darker. Thus, by subtracting proton-weighted images from T2 images and adding a constant to ensure positive pixel values, the CSF is highlighted. Similar pre-processing methods for lesions, gray or white matter could improve IIC medical image segmentation results. In addition, x' was obtained only from shifts, which contains slightly different content than x. This breaks the model of x and x' always having the same content, which is a conceptual issue of IIC segmentation. Moreover, including synthetic transformations with or without shifts may lead to different results. In the future, we will investigate adapting IIC to address the conceptual issue, while also verifying our implementation.

Appendix A

TwoPathCNN Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 21 13:47:09 2019

@author: Kevin Raina
"""

import nibabel as nib
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import numpy as np
from random import randint
from random import seed
from pylab import *

class process:



    def load_data(data_path , modality, range1,range2 ):
        """
        Parameters:
        data_path: path to the folder "Training" with brain images
        """
        data = []
        for brain_id in range(range1, range2+1):
            if brain_id < 10:
                brain_id = '0' + str(brain_id)
            folder = os.path.join(data_path, str(brain_id))
            for file in os.listdir(folder):
                if modality in file:
                    filefile = file + '/' + file +'.nii'
                    file_name = os.path.join(folder, filefile)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)
        normalized1 = process.standard_normalization(data)
        return normalized1

    def load_header(data_path , modality, range1, range2):
        for brain_id in range(range1, range2+1):
            if brain_id < 10:
                brain_id = str(brain_id)
            folder = os.path.join(data_path, str(brain_id))
            for file in os.listdir(folder):
                if modality in file:
                    filefile = file + '/' + file +'.nii'
                    file_name = os.path.join(folder, filefile)
```

```
                image = nib.load(file_name)
                img_header = image.header
        return img_header

    def load_test_data(data_path , modality, range1,range2 ):
        """
        Parameters:
        data_path: path to the folder "Training" with brain images
        """
        data = []
        for brain_id in range(range1, range2+1):
            if brain_id < 10:
                brain_id = str(brain_id)
            folder = os.path.join(data_path, str(brain_id))
            for file in os.listdir(folder):
                if modality in file:
                    filefile = file + '/' + file +'.nii'
                    file_name = os.path.join(folder, filefile)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)
        normalized1 = process.standard_normalization(data)
        return normalized1


    def dataorganize(array,testsize,valsize,trainsize,modality,mode):
        data = []
        if mode == "Validation":
            for j in range(0,valsize):
                data.append(process.load_data("/data/SISS2015/Training",modality, array[j],array[j])[0])
        if mode == "Test":
            for j in range(valsize,valsize+testsize):
                data.append(process.load_test_data("/data/SISS2015/Testing",modality, array[j],array[j])
[0])
        if mode == "Training":
            for j in range(valsize + testsize, valsize + trainsize + testsize):
                data.append(process.load_data("/data/SISS2015/Training",modality, array[j],array[j])[0])
        return(data)

    def labelorganize(array,testsize,valsize,trainsize,mode):
        labels = []
        if mode == "Validation":
            for k in range(0,valsize):
                labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
        if mode == "Test":
            for k in range(valsize,valsize + testsize):
                labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
        if mode == "Training":
            for k in range(valsize + testsize,valsize + testsize + trainsize):
```

```
            labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
        return(labels)

    def load_symmdata(data_path, modality):
        """Modality is always Flair"""
        data = []
        if modality == "Flair":
            indexes =
np.array([70614,70620,70626,70632,70638,70644,70650,70656,70662,70668,70674,70680,70686,706
92,70708,70718,
                      70730,70736,70750,70768,70774,70780,70786,70792,70798,70804,70810,70816])
        if modality == "T1":
            indexes =
np.array([70615,70621,70627,70633,70639,70645,70651,70657,70663,70669,70675,70681,70687,706
93,70709,70725,
                      70731,70737,70753,70769,70775,70781,70787,70793,70799,70805,70811,70817])

        i=0
        while len(data) < 28 and i < 28 :
            for file in os.listdir(data_path):
                if i == 28:
                    i = 28
                elif modality in file and str(indexes[i]) in file:
                    i=i+1
                    file_name = os.path.join(data_path,file)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)
        return(data)

    def load_symmdata_test(data_path, modality):
        data = []
        if modality == "T1":
            indexes =
np.array([86843,86849,86855,86861,86867,86873,86879,86885,86891,86897,86903,86909,86915,869
21,86927,86933,

86939,86945,86951,86957,86963,86969,86975,86981,86987,86993,86999,87005,92727,92740,92746,
92752,92765,
                      92928,92734,92759])
        if modality == "Flair":
            indexes =
np.array([86842,86848,86854,86860,86866,86872,86878,86884,86890,86896,86902,86908,86914,869
20,86926,86932,

86938,86944,86950,86956,86962,86968,86974,86980,86986,86992,86998,87004,92726,92739,92745,
92751,92764,
                      92927,92733,92758])
        i=0
```

```
        while len(data) < 36 and i < 36 :
            for file in os.listdir(data_path):
                if i == 36:
                    i = 36
                elif modality in file and str(indexes[i]) in file:
                    i=i+1
                    file_name = os.path.join(data_path,file)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)
        print(data[35].shape)
        print(data[30].shape)
        print(data[20].shape)
        print(data[1].shape)
        return(data)

    def dataorganizesymm(array,mode,modality,linear):
        data = []
        if linear == False:
            if mode == "Training":
                if modality == "Flair":
                    symmdata = process.load_symmdata("/home/krain033/homLR","Flair")
                if modality == "T1":
                    symmdata = process.load_symmdata("/home//SymmCNN/homLR/T1all","T1")
                for k in range(0,28):
                    data.append(symmdata[array[k]-1])
            if mode == "Test":
                if modality == "Flair":
                    symmdata =
process.load_symmdata_test("/home//SymmCNN/homLR/FlairTesting","Flair")
                if modality == "T1":
                    symmdata =
process.load_symmdata_test("/home//SymmCNN/homLR/T1all","T1")
                for k in range(0,36):
                    data.append(symmdata[array[k]-1])
        if linear == True:
            if mode == "Training":
                if modality == "Flair":
                    symmdata =
process.load_symmdata("/home//SymmCNN/homLR_linear/allFlair","Flair")
                if modality == "T1":
                    symmdata =
process.load_symmdata("/home//SymmCNN/homLR_linear/allT1","T1")
                for k in range(0,28):
                    data.append(symmdata[array[k]-1])

            if mode == "Test":
                if modality == "Flair":
                    symmdata =
```

```
process.load_symmdata_test("/home//SymmCNN/homLR_linear/allFlair","Flair")
                if modality == "T1":
                    symmdata =
process.load_symmdata_test("/home//SymmCNN/homLR_linear/allT1","T1")
                for k in range(0,36):
                    data.append(symmdata[array[k]-1])

        return(data)

    def load_labels(data_path, range1, range2):

        labels = []
        for brain_id in range(range1, range2+1):
            if brain_id < 10:
                brain_id = '0' + str(brain_id)
            folder = os.path.join(data_path, str(brain_id))
            for file in os.listdir(folder):
                if "OT" in file:
                    filefile = file + '/' + file +'.nii'
                    file_name = os.path.join(folder, filefile)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    labels.append(np_image)

        return labels

    def standard_normalization(data):
        """
        Here the input is a list of brains and the function normalize each of this brains
        to have a mean 0 and standard deviation 1.
        """
        normalized = []
        for image in data:
            m = np.mean(image)
            image = image - m
            st_dev = np.std(image)
            image = image/st_dev
            normalized.append(image)
        return normalized

    def one_hot_encode(labels, number_of_labels=2):
        """
        One hot encode a list of sample labels. Return a one-hot encoded vector for each label.
        : labels: List of sample Labels
        : return: Numpy array of one-hot encoded labels
        """
        labels = np.array(labels)
        one_hot = np.zeros((labels.size, number_of_labels), dtype = np.int)
        one_hot[np.arange(labels.size), labels] = 1
```

```
        return one_hot

    def one_hot_encode_dense(labels, number_of_labels=2):
        """
        One hot encode a list of sample labels. Return a one-hot encoded vector for each label.
        : labels: List of sample Labels
        : return: Numpy array of one-hot encoded labels
        """
        labels = np.array(labels)
        one_hot = np.zeros(labels.shape + (2,), dtype = np.int)
        one_hot[process.all_idx(labels, axis=3)] = 1
        return one_hot

    def all_idx(idx, axis):
        grid = np.ogrid[tuple(map(slice, idx.shape))]
        grid.insert(axis, idx)
        return tuple(grid)

    def
get_samples(data_I,data_I_symm,data_F_symm,data_D_symm,data_T1_symm,data_T2_symm,data_F,
data_D,data_T1,data_T2, labels, from_brains, size , dim, pi,symmetry,joint,marginal,interaction):
        """
        Parameters:
        data: list of 3D numpy arrays representing an image of the brain
        labels: list of 3D numpy array representing a voxelwise segmentation of the brain in data
        from_brains:
        size: number of extracted patches
        dim: patch has a shape (dim, dim)

        Returns: tuple, namely, (array of patches, array of labels of the central voxels in the patches).

        Samples are taken from the uniform distribution.
        """
        batch_data_F = []
        batch_data_D = []
        batch_data_T1 = []
        batch_data_T2 = []
        batch_data_F_symm = []
        batch_data_D_symm = []
        batch_data_T1_symm = []
        batch_data_T2_symm = []
        batch_data_I = []
        batch_data_I_symm =[]
        batch_labels = []


        while len(batch_data_F) < size:
            bernoulli = np.random.binomial(size=1,n=1,p=pi)
            bernoulli = int(bernoulli)
```

```
            found = 0
            index = randint(0,from_brains)
            while found == 0:
               if bernoulli == 0:
                  x = randint(int(dim/2), (data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                  y = randint(int(dim/2), (data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                  z = randint(int(dim/2), (data_F[index].shape[2] - 1) - (int(dim/2) + 1) + 1 + 1)
               if  bernoulli == 1:
                  x = randint(int(dim/2), (data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                  y = randint(int(dim/2), (data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                  z = randint(0,data_F[index].shape[2])
               if data_F[index][x][y][z] != 0 and labels[index][x][y][z] == bernoulli:
                  found = 1
                  patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                  """
                  when you subet an array the upper dimension is + 1 above the true size
                  """
                  patch_F.reshape((dim, dim, 1))
                  batch_data_F.append(patch_F)
                  patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                  patch_D.reshape((dim, dim, 1))
                  batch_data_D.append(patch_D)
                  patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                  patch_T1.reshape((dim, dim, 1))
                  batch_data_T1.append(patch_T1)
                  patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                  patch_T2.reshape((dim, dim, 1))
                  batch_data_T2.append(patch_T2)

                  patch_F = data_F_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                  patch_F.reshape((dim, dim, 1))
                  batch_data_F_symm.append(patch_F)
                  patch_D = data_D_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                  patch_D.reshape((dim, dim, 1))
                  batch_data_D_symm.append(patch_D)
                  patch_T1 = data_T1_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                  patch_T1.reshape((dim, dim, 1))
                  batch_data_T1_symm.append(patch_T1)
                  patch_T2 = data_T2_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                  patch_T2.reshape((dim, dim, 1))
                  batch_data_T2_symm.append(patch_T2)
```

```
                patch_I = data_I[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1, z]
                patch_I.reshape((dim,dim,1))
                batch_data_I.append(patch_I)
                patch_I_symm = data_I_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                patch_I_symm.reshape((dim,dim,1))
                batch_data_I_symm.append(patch_I_symm)

                label = labels[index][x][y][z]
                batch_labels.append(label)
        batch_data_F = np.array(batch_data_F)
        batch_data_D = np.array(batch_data_D)
        batch_data_T1 = np.array(batch_data_T1)
        batch_data_T2 = np.array(batch_data_T2)
        batch_data_F_symm = np.array(batch_data_F_symm)
        batch_data_D_symm = np.array(batch_data_D_symm)
        batch_data_T1_symm = np.array(batch_data_T1_symm)
        batch_data_T2_symm = np.array(batch_data_T2_symm)
        batch_labels = np.array(process.one_hot_encode(batch_labels))
        batch_data_I = np.array(batch_data_I)
        batch_data_I_symm = np.array(batch_data_I_symm)
        #batch_labels=np.array(batch_labels)
        if symmetry == True:
            if joint == True:
                if interaction == True:
                    samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_F_symm,batch_data_
D_symm,batch_data_T1_symm,batch_data_T2_symm,batch_data_I,batch_data_I_symm])
                if interaction == False:
                    samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_F_symm,batch_data_
D_symm,batch_data_T1_symm,batch_data_T2_symm])
            if marginal == "Flair" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_F_symm])
            if marginal == "DWI" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_D_symm])
            if marginal == "T1" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_T1_symm])
            if marginal == "T2" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_T2_symm])

        if symmetry == False:
            samples = np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2])
```

```
        return samples, batch_labels

    def
get_uniform_samples_dense_batch(data_I,data_I_symm,data_F_symm,data_D_symm,data_T1_symm,
data_T2_symm,data_F,data_D, data_T1,data_T2, labels, from_brains , size , dim
,archsize,symmetry,joint,marginal,interaction):
        """
        Parameters:
        data: list of 3D numpy arrays representing an image of the brain
        labels: list of 3D numpy array representing a voxelwise segmentation of the brain in data
        from_brains:
        size: number of extracted patches
        dim: patch has a shape (dim, dim)

        Returns: tuple, namely, (array of patches, array of labels of the central voxels in the patches).

        There are equal number of samples labeled 1 and 0.
        """
        batch_data_F = []
        batch_data_D = []
        batch_data_T1 = []
        batch_data_T2 = []
        batch_data_F_symm = []
        batch_data_D_symm = []
        batch_data_T1_symm = []
        batch_data_T2_symm = []
        batch_data_I = []
        batch_data_I_symm = []
        batch_labels = []
        while len(batch_data_F) < size/2:
            index = randint(0,from_brains)
            found = 0
            while found == 0:
                x = randint(int(dim/2),(data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                y = randint(int(dim/2),(data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                z = randint(int(dim/2),(data_F[index].shape[2] - 1) - (int(dim/2) + 1) + 1 + 1)
                if data_F[index][x][y][z] != 0 and labels[index][x][y][z] == 0:
                    found = 1
                    patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    """
                    when you subet an array the upper dimension is + 1 above the true size
                    """
                    batch_data_F.append(patch_F)
                    patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    batch_data_D.append(patch_D)
                    patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
```

```
                batch_data_T1.append(patch_T1)
                patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                batch_data_T2.append(patch_T2)

                patch_F = data_F_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_F_symm.append(patch_F)
                patch_D = data_D_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_D_symm.append(patch_D)
                patch_T1 = data_T1_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_T1_symm.append(patch_T1)
                patch_T2 = data_T2_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_T2_symm.append(patch_T2)

                patch_I = data_I[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1, z]
                batch_data_I.append(patch_I)
                patch_I_symm = data_I_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_I_symm.append(patch_I_symm)


                label = labels[index][x - int(dim/2) + int(archsize/2): x + int(dim/2) - int(archsize/2) +
1,y - int(dim/2) + int(archsize/2): y + int(dim/2) - int(archsize/2) + 1 ,z]
                batch_labels.append(label)
        while len(batch_data_F) < size:
            index = randint(0,from_brains)
            found = 0
            while found == 0:
                x = randint(int(dim/2),(data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                y = randint(int(dim/2),(data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                z = randint(0, (data_F[index].shape[2]))
                if data_F[index][x][y][z] != 0 and labels[index][x][y][z] == 1:
                    found = 1
                    patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    """
                    when you subet an array the upper dimension is + 1 above the true size
                    """
                    batch_data_F.append(patch_F)
                    patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    batch_data_D.append(patch_D)
                    patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                    batch_data_T1.append(patch_T1)
```

```
                patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                batch_data_T2.append(patch_T2)

                patch_F = data_F_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_F_symm.append(patch_F)
                patch_D = data_D_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_D_symm.append(patch_D)
                patch_T1 = data_T1_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_T1_symm.append(patch_T1)
                patch_T2 = data_T2_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_T2_symm.append(patch_T2)

                patch_I = data_I[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1, z]
                batch_data_I.append(patch_I)
                patch_I_symm = data_I_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_I_symm.append(patch_I_symm)

                label = labels[index][x - int(dim/2) + int(archsize/2): x + int(dim/2) - int(archsize/2) +
1,y - int(dim/2) + int(archsize/2): y + int(dim/2) - int(archsize/2) + 1 ,z]
                batch_labels.append(label)
        batch_data_F = np.array(batch_data_F)
        batch_data_D = np.array(batch_data_D)
        batch_data_T1 = np.array(batch_data_T1)
        batch_data_T2 = np.array(batch_data_T2)
        batch_data_F_symm = np.array(batch_data_F_symm)
        batch_data_D_symm = np.array(batch_data_D_symm)
        batch_data_T1_symm = np.array(batch_data_T1_symm)
        batch_data_T2_symm = np.array(batch_data_T2_symm)
        batch_data_I = np.array(batch_data_I)
        batch_data_I_symm = np.array(batch_data_I_symm)


        # batch_labels = np.array(process.one_hot_encode(batch_labels))
        # batch_labels = np.array(batch_labels)

        batch_labels = np.array(process.one_hot_encode_dense(batch_labels))

        #   randomize = np.arange(size)
        #   np.random.shuffle(randomize)
        if symmetry == True:
            if joint == True:
                if interaction == True:
                    samples =
```

```
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_F_symm,batch_data_
D_symm,batch_data_T1_symm,batch_data_T2_symm,batch_data_I,batch_data_I_symm])
                if interaction == False:
                    samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_F_symm,batch_data_
D_symm,batch_data_T1_symm,batch_data_T2_symm])
            if marginal == "Flair" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_F_symm])
            if marginal == "DWI" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_D_symm])
            if marginal == "T1" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_T1_symm])
            if marginal == "T2" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_T2_symm])

        if symmetry == False:
            samples = np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2])

        return samples,batch_labels

    def create_new_conv_layer(input_data, num_input_channels, num_filters, filter_shape,
pool_shape, K, name, patch_size, addpool, keep_prob):
        # setup the filter input shape for tf.nn.conv_2d
        conv_filt_shape = [filter_shape[0], filter_shape[1], num_input_channels,num_filters]

        # initialise weights and bias for the filter
        weights = tf.Variable(tf.random_uniform(conv_filt_shape, minval = -0.005,maxval = 0.005,
seed = 1967),name=name+'_W')
        bias = tf.Variable(tf.zeros([num_filters],name = name + '_b'))
        #bias = tf.Variable(tf.truncated_normal([num_filters],seed = 1953), name=name+'_b')

        # setup the convolutional layer operation
        if addpool == False:
            out_layer = tf.nn.conv2d(input_data, weights, [1, 1, 1, 1],padding = 'VALID')

            # add the bias
            out_layer += bias

            # apply a ReLU non-linear activation
            out_layer = tf.nn.relu(out_layer)
            out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1992)
            return out_layer
        if addpool == True:
            out_layer = tf.nn.conv2d(input_data, weights, [1, 1, 1, 1],padding = 'VALID')
```

```
        # add the bias
        out_layer += bias

        # apply a ReLU non-linear activation
        out_layer = tf.nn.relu(out_layer)               # now perform max pooling
        ksize = [1, pool_shape[0], pool_shape[1], 1]
        strides = [1, 1, 1, 1]
        out_layer = tf.nn.max_pool(out_layer, ksize=ksize, strides=strides, padding = 'VALID')
        out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1995)
        return out_layer

    def maxout(x,num_filters,patch_size,K):

        a1,a2,a3,a4 = x.get_shape().as_list()
        N = a1
        if N == None:
            print("Goodfellow initialization")


        else:
            for n in range(1,N + 1):
                for l in range(1,patch_size + 1):
                    for w in range(1,patch_size + 1):
                        for s in range(1,num_filters - K + 2):
                            n1,n2,n3 = tf.split(x,[n-1,1,N-(n-1)-1],0)
                            l1,l2,l3 = tf.split(n2,[l-1,1,patch_size-(l-1)-1],1)
                            w1,w2,w3 = tf.split(l2,[w-1,1,patch_size-(w-1)-1],2)
                            p1,p2,p3 = tf.split(w2,[s-1,1,num_filters-(s-1)-1],3)
                            s1,s2,s3 = tf.split(w2,[s-1,K,num_filters-(s-1)-K],3)
                            d = tf.reduce_max(s2,3,keep_dims = True)
                            r = tf.concat([p1,d,p3],3)
                            t = tf.concat([w1,r,w3],2)
                            m = tf.concat([l1,t,l3],1)
                            g = tf.concat([n1,m,n3],0)
                            x = g
        return(x)

    def
crossval(linear,symmtrainingmodality,symmtestingmodality,barray,nval,ntrain,ntest,j,mu,d,g1,g2,b,n,la
mbda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondphase,tsize,
pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction):
        """
        for jval in range(1,j+1):
            print("Sample:",j)
            array = process.randomvaltrain(choicearray = barray , nbrains=28, testsize = 0, valsize = 3)
            print("Array Setup:",array)
        """
        seed(13031953)
        array = np.array([15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,1,28,7,10,23,6,16,24,18,9,2,4])
```

```
        print("Fold 1",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction)

        seed(13031953)
        array = np.array([17,19,22,21,15,20,14,8,25,5,13,26,12,3,11,27,1,28,7,10,23,6,16,24,18,9,2,4])
        print("Fold 2",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction)

        seed(13031953)
        array = np.array([25,5,13,26,15,20,14,8,17,19,22,21,12,3,11,27,1,28,7,10,23,6,16,24,18,9,2,4])
        print("Fold 3",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction)
        seed(13031953)
        array = np.array([12,3,11,27,15,20,14,8,17,19,22,21,25,5,13,26,1,28,7,10,23,6,16,24,18,9,2,4])

        print("Fold 4",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction)
        seed(13031953)
        array = np.array([1,28,7,10,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,23,6,16,24,18,9,2,4])
        print("Fold 5",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction)
        seed(13031953)
        array = np.array([23,6,16,24,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,1,28,7,10,18,9,2,4])
        print("Fold 6",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction)
        seed(13031953)
        array = np.array([18,9,2,4,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,1,28,7,10,23,6,16,24])
        print("Fold 7",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
```

```
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction)
        """
        for jval in range(17,29):
            seed(13031953)
            print("Case",jval)
            componentarr = np.arange(1,29)
            componentarr[0] = jval
            componentarr[jval-1] = 1
            print("Array:",componentarr,"Left-out Case:",componentarr[0])

process.mainrandom(componentarr,nval,ntrain,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpo
int,dsize,costinit,costcheckpt,burnin1,burnin2,secondphase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,
joint,marginal,interaction)
        """



    def
test(linear,symmtrainingmodality,symmtestingmodality,barray,nval,ntrain,ntest,j,mu,d,g1,g2,b,n,lambda
1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondphase,tsize,pi,dro
p1,drop2,drop3,drop4,symmetry,joint,marginal,interaction):
        seed(13031953)
        array = np.arange(1,29)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction)



    """Kevin Raina"""
    def
mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1,g2,b,n,l
ambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondphase,tsize
,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal,interaction):
        for v in range(1,2):
            print("Validation: Fold #",v)

            if nval != 0:
                data_F_T = process.dataorganize(array,ntest,nval,ntrain,"Flair","Training")
                data_D_T = process.dataorganize(array,ntest,nval,ntrain,"DWI","Training")
                data_T1_T = process.dataorganize(array,ntest,nval,ntrain,"T1","Training")
                data_T2_T = process.dataorganize(array,ntest,nval,ntrain,"T2","Training")
                labels_T = process.labelorganize(array,ntest,nval,ntrain,"Training")

                data_F_V = process.dataorganize(array,ntest,nval,ntrain,"Flair","Validation")
                data_D_V = process.dataorganize(array,ntest,nval,ntrain,"DWI","Validation")
                data_T1_V = process.dataorganize(array,ntest,nval,ntrain,"T1","Validation")
                data_T2_V = process.dataorganize(array,ntest,nval,ntrain,"T2","Validation")
```

```
    labels_V = process.labelorganize(array,ntest,nval,ntrain,"Validation")
    for w in range(0,4):
        print(data_F_V[w].shape)
if nval ==0:
    data_F_T = process.dataorganize(array,0,nval,ntrain,"Flair","Training")
    data_D_T = process.dataorganize(array,0,nval,ntrain,"DWI","Training")
    data_T1_T = process.dataorganize(array,0,nval,ntrain,"T1","Training")
    data_T2_T = process.dataorganize(array,0,nval,ntrain,"T2","Training")
    labels_T = process.labelorganize(array,0,nval,ntrain,"Training")
    array2 = np.arange(1,37)
    data_F_V = process.dataorganize(array2,ntest,nval,0,"Flair","Test")
    data_D_V = process.dataorganize(array2,ntest,nval,0,"DWI","Test")
    data_T1_V = process.dataorganize(array2,ntest,nval,0,"T1","Test")
    data_T2_V = process.dataorganize(array2,ntest,nval,0,"T2","Test")
    for w in range(0,4):
        print(data_F_V[w].shape)



if symmetry == True:
    print("Loading symmetric data...")
    if nval !=0:
        if linear == False:
            if symmtrainingmodality == "Flair":
                SymmData = process.dataorganizesymm(array,"Training","Flair",linear)
            if symmtrainingmodality == "T1":
                SymmData = process.dataorganizesymm(array,"Training","T1",linear)
        if linear == True:
            if symmtrainingmodality == "Flair":
                SymmData = process.dataorganizesymm(array,"Training","Flair",linear)
            if symmtrainingmodality == "T1":
                SymmData = process.dataorganizesymm(array,"Training","T1",linear)

            for w in range(0,4):
                print(SymmData[w].shape)
    if nval == 0:
        if linear == False:
            if symmtrainingmodality == "T1":
                SymmDataTrain = process.dataorganizesymm(array,"Training","T1",linear)
            if symmtrainingmodality == "Flair":
                SymmDataTrain = process.dataorganizesymm(array,"Training","Flair",linear)
            if symmtestingmodality == "T1":
                SymmDataTest = process.dataorganizesymm(array2,"Test","T1",linear)
            if symmtestingmodality == "Flair":
                SymmDataTest = process.dataorganizesymm(array2,"Test","Flair",linear)
        if linear == True:
            if symmtrainingmodality == "Flair":
                SymmDataTrain = process.dataorganizesymm(array,"Training","Flair",linear)
```

```
                    if symmtestingmodality == "Flair":
                        SymmDataTest = process.dataorganizesymm(array2,"Test","Flair",linear)
                    if symmtrainingmodality == "T1":
                        SymmDataTrain = process.dataorganizesymm(array,"Training","T1",linear)
                    if symmtestingmodality == "T1":
                        SymmDataTest = process.dataorganizesymm(array2,"Test","T1",linear)



                print("Done Loading")
                if joint == True:
                    print("Building Symmetric Features...")
                    if nval != 0:
                        data_F_V_symm =process.buildsymm(data_F_V,SymmData,nval,ntrain,"Validation")
                        data_D_V_symm
=process.buildsymm(data_D_V,SymmData,nval,ntrain,"Validation")
                        data_T1_V_symm
=process.buildsymm(data_T1_V,SymmData,nval,ntrain,"Validation")
                        data_T2_V_symm
=process.buildsymm(data_T2_V,SymmData,nval,ntrain,"Validation")

                        data_F_T_symm =process.buildsymm(data_F_T,SymmData,nval,ntrain,"Training")
                        data_D_T_symm =process.buildsymm(data_D_T,SymmData,nval,ntrain,"Training")
                        data_T1_T_symm
=process.buildsymm(data_T1_T,SymmData,nval,ntrain,"Training")
                        data_T2_T_symm
=process.buildsymm(data_T2_T,SymmData,nval,ntrain,"Training")

                    if nval == 0:
                        data_F_V_symm =process.buildsymm(data_F_V,SymmDataTest,0,ntest,"Training")
                        data_D_V_symm =process.buildsymm(data_D_V,SymmDataTest,0,ntest,"Training")
                        data_T1_V_symm
=process.buildsymm(data_T1_V,SymmDataTest,0,ntest,"Training")
                        data_T2_V_symm
=process.buildsymm(data_T2_V,SymmDataTest,0,ntest,"Training")

                        data_F_T_symm =process.buildsymm(data_F_T,SymmDataTrain,0,ntrain,"Training")
                        data_D_T_symm
=process.buildsymm(data_D_T,SymmDataTrain,0,ntrain,"Training")
                        data_T1_T_symm
=process.buildsymm(data_T1_T,SymmDataTrain,0,ntrain,"Training")
                        data_T2_T_symm
=process.buildsymm(data_T2_T,SymmDataTrain,0,ntrain,"Training")

                    mods=8
                    if interaction == True:
                        print("Building Interaction Features...")
                        data_I_T =
```

```
process.buildInteractions(data_F_T,data_D_T,data_T1_T,data_T2_T,nval,ntrain,"Training")
                    data_I_T_symm =
process.buildInteractions(data_F_T_symm,data_D_T_symm,data_T1_T_symm,data_T2_T_symm,nval
,ntrain,"Training")
                    data_I_V =
process.buildInteractions(data_F_V,data_D_V,data_T1_V,data_T2_V,nval,ntrain,"Validation")
                    data_I_V_symm =
process.buildInteractions(data_F_V_symm,data_D_V_symm,data_T1_V_symm,data_T2_V_symm,nv
al,ntrain,"Validation")
                    mods = 10
                if marginal == "Flair" and joint == False:
                    data_F_V_symm =process.buildsymm(data_F_V,SymmData,nval,ntrain,"Validation")
                    data_F_T_symm =process.buildsymm(data_F_T,SymmData,nval,ntrain,"Training")
                    mods = 5
                if marginal == "DWI" and joint == False:
                    data_D_V_symm =process.buildsymm(data_D_V,SymmData,nval,ntrain,"Validation")
                    data_D_T_symm =process.buildsymm(data_D_T,SymmData,nval,ntrain,"Training")
                    mods = 5
                if marginal == "T1" and joint == False:
                    data_T1_V_symm =process.buildsymm(data_T1_V,SymmData,nval,ntrain,"Validation")
                    data_T1_T_symm =process.buildsymm(data_T1_T,SymmData,nval,ntrain,"Training")
                    mods = 5
                if marginal == "T2" and joint == False:
                    data_T2_V_symm =process.buildsymm(data_T2_V,SymmData,nval,ntrain,"Validation")
                    data_T2_T_symm =process.buildsymm(data_T2_T,SymmData,nval,ntrain,"Training")
                    mods = 5
            if symmetry == False:
                mods = 4


            """
            data_D_V_symm =process.buildsymm(data_D_V,SymmData,nval,ntrain,"Validation")
            data_T1_V_symm =process.buildsymm(data_T1_V,SymmData,nval,ntrain,"Validation")
            data_T2_V_symm =process.buildsymm(data_T2_V,SymmData,nval,ntrain,"Validation")

            data_D_T_symm =process.buildsymm(data_D_T,SymmData,nval,ntrain,"Training")
            data_T1_T_symm =process.buildsymm(data_T1_T,SymmData,nval,ntrain,"Training")
            data_T2_T_symm =process.buildsymm(data_T2_T,SymmData,nval,ntrain,"Training")
            """



            tf.reset_default_graph()

            x = tf.placeholder(tf.float32)
            y = tf.placeholder(tf.float32)
            probl1 = tf.placeholder_with_default(1.0,shape=())
            probl2 = tf.placeholder_with_default(1.0,shape=())
            probl3 = tf.placeholder_with_default(1.0,shape=())
```

```
        probl4 = tf.placeholder_with_default(1.0,shape=())
        """patch-6,patch-11,patch-12,1,[patch-12,patch-12], 24,21,21,1,[21,21]"""
        """4,64,7,4,64,64,3,2,4,160,13,13,1,224,2,1,1 """
        layer1 = process.create_new_conv_layer(x,mods,64,[7,7],[4,4],2,'layer1',patch-6,True,probl1)
        layer2 = process.create_new_conv_layer(layer1,64,64,[3,3],[2,2],2,'layer2',patch-
11,True,probl2)
        layer3 = process.create_new_conv_layer(x,mods,160,[13,13],[1,1],2,'layer3',patch-
12,False,probl3)
        conc = tf.concat([layer3, layer2], axis = 3)
        layer4 = process.create_new_conv_layer(conc,224,2,[patch-12,patch-12],
[1,1],2,'layer4',1,False,probl4)
        logits = tf.squeeze(layer4)
        """layer 4: 1 by 198 by 198 by 2, logits: 198 by 198 by 2"""
        y_ = tf.nn.softmax(logits)
        """ 154 by 230 by 230 by 2 m,idx,count = tf.unique_with_counts(tf.argmax(y,3))"""
        iteration = int(len(tf.trainable_variables())/8)

        reg1 = tf.reduce_sum(tf.abs(tf.trainable_variables()[iteration*8-2])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[iteration*8-4]))+
tf.reduce_sum(tf.abs(tf.trainable_variables()[iteration*8-6]))
+tf.reduce_sum(tf.abs(tf.trainable_variables()[iteration*8-8]))
        reg2 = tf.nn.l2_loss(tf.trainable_variables()[iteration*8-2]) +
tf.nn.l2_loss(tf.trainable_variables()[iteration*8-4])+tf.nn.l2_loss(tf.trainable_variables()[iteration*8-
6])+tf.nn.l2_loss(tf.trainable_variables()[iteration*8-8])
        cross_entropy= tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits =
logits,labels = y))
        loss = cross_entropy + lambda1*reg1 + lambda2*reg2
        global_step = tf.Variable(0,trainable = False)

        """
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=g1).minimize(loss)


        """
        learningprime = tf.train.piecewise_constant(global_step, boundaries =
[20000,30000,40000,50000,60000,70000,80000,90000,100000],values = [g1,0.1*g1,(0.1**2)*g1,
(0.1**3)*g1, g2,(0.1)*g2,(0.1**2)*g2,(0.1**3)*g2,(0.1**4)*g2,(0.1**5)*g2])
        """
        learningprime = tf.train.exponential_decay(learning_rate = g1,global_step =
global_step,decay_steps = dsize,decay_rate = d)
        """
        optimizer = tf.train.MomentumOptimizer(learningprime,mu).minimize(loss, global_step =
global_step)

        optimizer2 = tf.train.MomentumOptimizer(learningprime,mu).minimize(loss, var_list =
tf.trainable_variables()[iteration*8-2:], global_step=global_step)

        gradients = tf.gradients(loss, tf.trainable_variables())
```

```
        paddedy_ = tf.pad(tf.argmax(y_,3),paddings = tf.constant([[0,0],[int(patch/2),int(patch/2)],
[int(patch/2),int(patch/2)]]),mode = "CONSTANT")
        tp = tf.metrics.true_positives(y,paddedy_,weights=None)[1]
        fp = tf.metrics.false_positives(y,paddedy_,weights=None)[1]
        fn = tf.metrics.false_negatives(y,paddedy_,weights=None)[1]




        init_op = tf.global_variables_initializer()
        init_l = tf.local_variables_initializer()
        with tf.Session() as sess:
            sess.run(init_op)
            sess.run(init_l)
            print("Training")
            total_batch = int(n)
            """
            switch = 0
            """
            for epoch in range(epochs):
                avg_cost = 0
                for i in range(1,total_batch + 1):

                    if (i/checkpoint).is_integer() and i > burnin1:
                        print("Overfit Checkpoint:",i/checkpoint)
                        print("Testing on Training Set:")

                        if nval == 0:
                            for m in range(1,ntest+1):
                                print("Validation Brain (not actual index)",m)
                                for s in range(0,2):
                                    if symmetry ==True:
                                        if joint == True:
                                            if interaction == True:
                                                data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1],data_D_V_symm[m-1],data_T1_V_symm[m-
1],data_T2_V_symm[m-1],data_I_V[m-1],data_I_V_symm[m-1]])
                                            if interaction == False:
                                                data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1],data_D_V_symm[m-1],data_T1_V_symm[m-
1],data_T2_V_symm[m-1]])
                                        if joint == False and marginal == "Flair":
                                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1]])
                                        if joint == False and marginal == "DWI":
                                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_D_V_symm[m-1]])
                                        if joint == False and marginal == "T1":
                                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
```

```
1],data_T2_V[m-1],data_T1_V_symm[m-1]])
                                if joint == False and marginal == "T2":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_T2_V_symm[m-1]])
                            if symmetry == False:
                                data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1]])
                            data_V_prime1 = np.transpose(data_V,(3,1,2,0))
                            dat1 = data_V_prime1[77*s:77*s + 77,:,:,:]

                            if s==0:
                                pred1 = sess.run(paddedy_, feed_dict = {x:dat1})
                                pred1 = np.asarray(pred1)
                                pred1 = np.transpose(pred1,(1,2,0))
                            if s==1:
                                pred2 = sess.run(paddedy_, feed_dict = {x:dat1})
                                pred2 = np.asarray(pred2)
                                pred2 = np.transpose(pred2,(1,2,0))
                                predf = np.concatenate((pred1,pred2),axis=2)
                                img_header = process.load_header("/data/SISS2015/Testing","Flair",
array2[m-1],array2[m-1])

                                predf1 = nib.Nifti1Image(predf,np.eye(4),img_header)
                                if symmetry == False:
                                    nib.save(predf1,'VSD.BL_'+str(m)+'.nii')
                                if symmetry == True:
                                    if linear == False:
                                        if symmtrainingmodality == "T1":
                                            nib.save(predf1,'VSD.NLS_T1_'+str(m)+'.nii')
                                        if symmtrainingmodality == "Flair":
                                            nib.save(predf1,'VSD.NLS_Flair_'+str(m)+'.nii')
                                    if linear == True:
                                        if symmtrainingmodality == "T1":
                                            nib.save(predf1,'VSD.LS_T1_'+str(m)+'.nii')
                                        if symmtrainingmodality == "Flair":
                                            nib.save(predf1,'VSD.LS_Flair_'+str(m)+'.nii')
                    if nval != 0:
                        for m in range(1,nval+1):
                            print("Validation Brain (not actual index)",m)
                            for s in range(0,2):
                                labels_Temp1 = labels_V[m-1]
                                labels_Temp_Prime1 = np.transpose(labels_Temp1,(2,0,1))
                                lab1 = labels_Temp_Prime1[77*s:77*s + 77,:,:]
                                if symmetry ==True:
                                    if joint == True:
                                        if interaction == True:
                                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1],data_D_V_symm[m-1],data_T1_V_symm[m-
1],data_T2_V_symm[m-1],data_I_V[m-1],data_I_V_symm[m-1]])
```

```
                        if interaction == False:
                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1],data_D_V_symm[m-1],data_T1_V_symm[m-
1],data_T2_V_symm[m-1]])
                        if joint == False and marginal == "Flair":
                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1]])
                        if joint == False and marginal == "DWI":
                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_D_V_symm[m-1]])
                        if joint == False and marginal == "T1":
                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_T1_V_symm[m-1]])
                        if joint == False and marginal == "T2":
                            data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_T2_V_symm[m-1]])
                    if symmetry == False:
                        data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1]])

                    data_V_prime1 = np.transpose(data_V,(3,1,2,0))
                    dat1 = data_V_prime1[77*s:77*s + 77,:,:,:]


                    if s==0:
                        pred1 = sess.run(paddedy_, feed_dict = {x:dat1,y:lab1})
                        pred1 = np.asarray(pred1)
                        pred1 = np.transpose(pred1,(1,2,0))
                    if s==1:
                        pred2 = sess.run(paddedy_, feed_dict = {x:dat1,y:lab1})
                        pred2 = np.asarray(pred2)
                        pred2 = np.transpose(pred2,(1,2,0))
                        predf = np.concatenate((pred1,pred2),axis=2)
                        predf1 = nib.Nifti1Image(predf,np.eye(4))

                        if symmetry == False:
                            nib.save(predf1,'VSD.BL_TRN_'+str(m)+'.nii')
                        if symmetry == True:
                            if linear == False:
                                if symmtrainingmodality == "T1":
                                    nib.save(predf1,'VSD.NLS_TRN_T1'+str(m)+'.nii')
                                if symmtrainingmodality == "Flair":
                                    nib.save(predf1,'VSD.NLS_TRN_Flair'+str(m)+'.nii')
                            if linear == True:
                                if symmtrainingmodality == "T1":
                                    nib.save(predf1,'VSD.LS_TRN_T1'+str(m)+'.nii')
                                if symmtrainingmodality == "Flair":
                                    nib.save(predf1,'VSD.LS_TRN_Flair'+str(m)+'.nii')
                    trup1 = sess.run(tp, feed_dict = {x:dat1,y:lab1})
```

```
                        trup1 = np.asarray(trup1)
                        falp1 = sess.run(fp, feed_dict = {x:dat1,y:lab1})
                        falp1 = np.asarray(falp1)
                        faln1 = sess.run(fn, feed_dict = {x:dat1,y:lab1})
                        faln1 = np.asarray(faln1)
                        np.save(file = "tp" + str(m) + str(s+1) + ".npy", arr = trup1)
                        np.save(file = "fp" + str(m) + str(s+1) + ".npy", arr = falp1)
                        np.save(file = "fn" + str(m) + str(s+1) + ".npy", arr = faln1)
                        sess.run(init_l)
                    print(process.dice(m))

            if i <= secondphase:
                if symmetry == True:
                    if joint == True:
                        if interaction == True:
                            train_samples =
process.get_uniform_samples_dense_batch(data_I_T,data_I_T_symm,data_F_T_symm,data_D_T_sym
m,data_T1_T_symm,data_T2_T_symm,data_F_T,data_D_T, data_T1_T,
data_T2_T,labels_T,ntrain,b,tsize,patch,symmetry,joint,marginal,interaction)
                        if interaction == False:
                            train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_F_T_symm,data_D_T_symm,dat
a_T1_T_symm,data_T2_T_symm,data_F_T,data_D_T, data_T1_T,
data_T2_T,labels_T,ntrain,b,tsize,patch,symmetry,joint,marginal,interaction)
                    if joint == False and marginal == "Flair":
                        train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_F_T_symm,data_F_T,data_F_T,d
ata_F_T,data_F_T,data_D_T, data_T1_T,
data_T2_T,labels_T,ntrain,b,tsize,patch,symmetry,joint,marginal,interaction)
                    if joint == False and marginal == "DWI":
                        train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_F_T,data_D_T_symm,data_F_T,d
ata_F_T,data_F_T,data_D_T, data_T1_T,
data_T2_T,labels_T,ntrain,b,tsize,patch,symmetry,joint,marginal,interaction)
                    if joint == False and marginal == "T1":
                        train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_F_T,data_F_T,data_T1_T_symm,
data_F_T,data_F_T,data_D_T, data_T1_T,
data_T2_T,labels_T,ntrain,b,tsize,patch,symmetry,joint,marginal,interaction)
                    if joint == False and marginal == "T2":
                        train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_F_T,data_F_T,data_F_T,data_T2_
T_symm,data_F_T,data_D_T, data_T1_T,
data_T2_T,labels_T,ntrain,b,tsize,patch,symmetry,joint,marginal,interaction)
                if symmetry == False:
                    train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_F_T,data_F_T,data_F_T,data_F_T
,data_F_T,data_D_T, data_T1_T,
data_T2_T,labels_T,ntrain,b,tsize,patch,symmetry,joint,marginal,interaction)
```

```
                    train_array = train_samples[0]
                    train_array = np.transpose(train_array,(1,2,3,0))
                    arrlab = train_samples[1]
                    dict = {x : train_array ,y : arrlab, probl1 : drop1 ,probl2 : drop2,probl3 :
drop3,probl4 : drop4}
                    temploss = sess.run(loss, feed_dict = dict)
                    _,c = sess.run([optimizer,loss],feed_dict=dict)
                    avg_cost += temploss/costcheckpt

                    if i < costinit or (i/costcheckpt).is_integer():
                        print(i,avg_cost)
                        if (i/costcheckpt).is_integer():
                            avg_cost = 0
                    carr = np.array([c])
                    if np.isnan(carr) == True:
                        print ("Cost is Infinite")

                if i > secondphase:
                    if i == secondphase + 1:
                        avg_cost = 0
                    if symmetry == True:
                        if joint == True:
                            if interaction == True:
                                train_samples =
process.get_samples(data_I_T,data_I_T_symm,data_F_T_symm,data_D_T_symm,data_T1_T_symm,d
ata_T2_T_symm,data_F_T,data_D_T,data_T1_T,
data_T2_T,labels_T,ntrain,b,patch,pi,symmetry,joint,marginal,interaction)
                            if interaction == False:
                                train_samples =
process.get_samples(data_F_T,data_F_T,data_F_T_symm,data_D_T_symm,data_T1_T_symm,data_T
2_T_symm,data_F_T,data_D_T,data_T1_T,
data_T2_T,labels_T,ntrain,b,patch,pi,symmetry,joint,marginal,interaction)
                        if joint == False and marginal == "Flair":
                            train_samples =
process.get_samples(data_F_T,data_F_T,data_F_T_symm,data_F_T,data_F_T,data_F_T,data_F_T,data
_D_T,data_T1_T, data_T2_T,labels_T,ntrain,b,patch,pi,symmetry,joint,marginal,interaction)
                        if joint == False and marginal == "DWI":
                            train_samples =
process.get_samples(data_F_T,data_F_T,data_F_T,data_D_T_symm,data_F_T,data_F_T,data_F_T,data
_D_T,data_T1_T, data_T2_T,labels_T,ntrain,b,patch,pi,symmetry,joint,marginal,interaction)
                        if joint == False and marginal == "T1":
                            train_samples =
process.get_samples(data_F_T,data_F_T,data_F_T,data_F_T,data_T1_T_symm,data_F_T,data_F_T,dat
a_D_T,data_T1_T, data_T2_T,labels_T,ntrain,b,patch,pi,symmetry,joint,marginal,interaction)
                        if joint == False and marginal == "T2":
                            train_samples =
process.get_samples(data_F_T,data_F_T,data_F_T,data_F_T,data_F_T,data_T2_T_symm,data_F_T,dat
a_D_T,data_T1_T, data_T2_T,labels_T,ntrain,b,patch,pi,symmetry,joint,marginal,interaction)
                    if symmetry == False:
```

```
                        train_samples =
process.get_samples(data_F_T,data_F_T,data_F_T,data_F_T,data_F_T,data_F_T,data_F_T,data_D_T,d
ata_T1_T, data_T2_T,labels_T,ntrain,b,patch,pi,symmetry,joint,marginal,interaction)
                    train_array = train_samples[0]
                    train_array = np.transpose(train_array,(1,2,3,0))
                    arrlab = train_samples[1]
                    dict = {x : train_array ,y : arrlab}
                    temploss = sess.run(loss, feed_dict = dict)
                    _,c = sess.run([optimizer2,loss],feed_dict=dict)
                    avg_cost += temploss/costcheckpt
                    if i < secondphase + costinit or (i/costcheckpt).is_integer():
                        print(i,avg_cost)
                        if (i/costcheckpt).is_integer():
                            avg_cost = 0
                    carr = np.array([c])
                    if np.isnan(carr) == True:
                        print ("Cost is Infinite")


            sess.close()


    def buildsymm(data_M_V,SymmData,nval,ntrain,mode):
        data = []
        if mode == "Validation":
            a = 0
            b = nval
        if mode == "Training":
            a = nval
            b = ntrain + nval
        for n in range(a,b):
            print("Loading Checkpoint (%):",100*(n+1)/b)
            data.append(np.zeros((data_M_V[n-a].shape[0],data_M_V[n-a].shape[1],data_M_V[n-
a].shape[2])))
            for i in range(0,data_M_V[n-a].shape[0]):
                for j in range(0,data_M_V[n-a].shape[1]):
                    for k in range(0,data_M_V[n-a].shape[2]):
                        xsymm = int(min(data_M_V[n-a].shape[0]-1,max(0,round(SymmData[n][i,j,k,0,0]))))
                        ysymm = int(min(data_M_V[n-a].shape[1]-1,max(0,round(SymmData[n][i,j,k,0,1]))))
                        zsymm = int(min(data_M_V[n-a].shape[2]-1,max(0,round(SymmData[n][i,j,k,0,2]))))
                        data[n-a][i,j,k] = data_M_V[n-a][i,j,k] - data_M_V[n-a][xsymm,ysymm,zsymm]


        return(data)

    def buildInteractions(data_M1,data_M2,data_M3, data_M4,nval,ntrain,mode):
        data = []
        if mode == "Validation":
            a = 0
```

```
            b = nval
        if mode == "Training":
            a = nval
            b = ntrain + nval
        for n in range(a,b):
            print("Loading Checkpoint (%):",100*(n+1)/b)
            data.append(np.zeros((data_M1[n-a].shape[0],data_M1[n-a].shape[1],data_M1[n-
a].shape[2])))
            for i in range(0,data_M1[n-a].shape[0]):
                for j in range(0,data_M1[n-a].shape[1]):
                    for k in range(0,data_M1[n-a].shape[2]):
                        data[n-a][i,j,k] = (data_M1[n-a][i,j,k]+data_M2[n-a][i,j,k]+data_M3[n-a][i,j,k]
+data_M4[n-a][i,j,k])/4
        return(data)



    def dice (m):
        tp1 = np.load("tp" + str(m) + str(1) + ".npy")
        tp2 = np.load("tp" + str(m) + str(2) + ".npy")
        fn1 = np.load("fn" + str(m) + str(1) + ".npy")
        fn2 = np.load("fn" + str(m) + str(2) + ".npy")
        fp1 = np.load("fp" + str(m) + str(1) + ".npy")
        fp2 = np.load("fp" + str(m) + str(2) + ".npy")
        tp = int(tp1) + int(tp2)
        fn = int(fn1) + int(fn2)
        fp = int(fp1) + int(fp2)
        print("True Positive Rate:", tp/(tp+fn))
        print("TP:",tp)
        print("FN:",fn)
        print("FP:",fp)
        if int(fn1) == 0 and int(fn2) == 0 and int(fp1) ==0 and int(fp2) == 0:
            dice = "Division by Zero in Dice Coefficient"
        else:
            dice = 2*(int(tp1) + int(tp2))/(2*(int(tp1) + int(tp2)) + int(fn1) + int(fn2) + int(fp1) +
int(fp2))
        return(dice)


process.crossval(linear = False,symmtrainingmodality = "Flair",symmtestingmodality ="Flair"
,barray=np.array([1]), j = 15, nval = 4,
            ntrain = 24,ntest = 0,mu = 0.6, d = 0.1, g1 = 0.001, g2= 0.001, b = 10, n = 90050, lambda1 =
0.000001,
            lambda2 = 0.0001, epochs = 1, patch = 33, checkpoint = 10000, dsize = 50000, costcheckpt
= 1000, costinit = 100,
            burnin1 = 85000, burnin2 = 100, secondphase = 50000, tsize = 33,pi=0.02,drop1=1,drop2
=0.5,drop3 =1,drop4=1,
            symmetry = False, joint = True, marginal = "T2",interaction = False)
```

```
"""
process.mainrandom(array =
np.array([15,20,14,8,22,18,2,4,1,3,5,6,7,9,10,11,12,13,16,17,19,21,23,24,25,26,27,28]), nval = 8,
                ntrain = 20,mu = 0.6, d = 0.1, g1 = 0.001, g2= 0.1, b = 10, n = 13,
               lambda1 = 0.000001, lambda2 = 0.0001, epochs = 1, patch = 33, checkpoint = 10, dsize =
50000,
               costcheckpt = 1000, costinit = 100, burnin1 = 3, burnin2 = 100, secondphase = 5, tsize =
33,pi=0.02,drop1=1,
               drop2 =0.5,drop3 =1,drop4=1, symmetry = False, joint = True, marginal = "T2",interaction =
False)
"""
```

Appendix B

Wider2DSeg Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 21 13:47:09 2019

@author:  Kevin Raina
"""
import scipy
import cv2
import nibabel as nib
import os
import math as mat
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import numpy as np
from random import randint
from random import seed
from pylab import *

class process:



    def load_data(data_path , modality, range1,range2 ):
        """
        Parameters:
        data_path: path to the folder "Training" with brain images
        """
        data = []
        for brain_id in range(range1, range2+1):
            if brain_id < 10:
                brain_id = '0' + str(brain_id)
            folder = os.path.join(data_path, str(brain_id))
            for file in os.listdir(folder):
                if modality in file:
                    filefile = file + '/' + file +'.nii'
                    file_name = os.path.join(folder, filefile)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)
        normalized1 = process.standard_normalization(data)
        return normalized1

    def load_test_data(data_path , modality, range1,range2 ):
        """
        Parameters:
        data_path: path to the folder "Training" with brain images
        """
        data = []
        for brain_id in range(range1, range2+1):
```

```
        if brain_id < 10:
            brain_id = str(brain_id)
        folder = os.path.join(data_path, str(brain_id))
        for file in os.listdir(folder):
            if modality in file:
                filefile = file + '/' + file +'.nii'
                file_name = os.path.join(folder, filefile)
                image = nib.load(file_name)
                np_image = image.get_data()
                data.append(np_image)
    normalized1 = process.standard_normalization(data)
    return normalized1

def load_header(data_path , modality, range1, range2):
    for brain_id in range(range1, range2+1):
        if brain_id < 10:
            brain_id = str(brain_id)
        folder = os.path.join(data_path, str(brain_id))
        for file in os.listdir(folder):
            if modality in file:
                filefile = file + '/' + file +'.nii'
                file_name = os.path.join(folder, filefile)
                image = nib.load(file_name)
                img_header = image.header
    return img_header

def dataorganize(array,testsize,valsize,trainsize,modality,mode):
    data = []
    if mode == "Validation":
        for j in range(0,valsize):
            data.append(process.load_data("/data/SISS2015/Training",modality, array[j],array[j])[0])
    if mode == "Test":
        for j in range(valsize,valsize+testsize):
            data.append(process.load_data("/data/SISS2015/Training",modality, array[j],array[j])[0])
    if mode == "Training":
        for j in range(valsize + testsize, valsize + trainsize + testsize):
            data.append(process.load_data("/data/SISS2015/Training",modality, array[j],array[j])[0])
    return(data)

def labelorganize(array,testsize,valsize,trainsize,mode):
    labels = []
    if mode == "Validation":
        for k in range(0,valsize):
            labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
    if mode == "Test":
        for k in range(valsize,valsize + testsize):
            labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
    if mode == "Training":
        for k in range(valsize + testsize,valsize + testsize + trainsize):
```

```
            labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
        return(labels)

    def load_symmdata(data_path, modality):
        """Modality is always Flair"""
        data = []
        if modality == "Flair":
            indexes =
np.array([70614,70620,70626,70632,70638,70644,70650,70656,70662,70668,70674,70680,70686,706
92,70708,70718,
                      70730,70736,70750,70768,70774,70780,70786,70792,70798,70804,70810,70816])
        if modality == "T1":
            indexes =
np.array([70615,70621,70627,70633,70639,70645,70651,70657,70663,70669,70675,70681,70687,706
93,70709,70725,
                      70731,70737,70753,70769,70775,70781,70787,70793,70799,70805,70811,70817])

        i=0
        while len(data) < 28 and i < 28 :
            for file in os.listdir(data_path):
                if i == 28:
                    i = 28
                elif modality in file and str(indexes[i]) in file:
                    i=i+1
                    file_name = os.path.join(data_path,file)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)
        return(data)

    def load_symmdata_test(data_path, modality):
        data = []
        if modality == "T1":
            indexes =
np.array([86843,86849,86855,86861,86867,86873,86879,86885,86891,86897,86903,86909,86915,869
21,86927,86933,

86939,86945,86951,86957,86963,86969,86975,86981,86987,86993,86999,87005,92727,92740,92746,
92752,92765,
                      92928,92734,92759])
        if modality == "Flair":
            indexes =
np.array([86842,86848,86854,86860,86866,86872,86878,86884,86890,86896,86902,86908,86914,869
20,86926,86932,

86938,86944,86950,86956,86962,86968,86974,86980,86986,86992,86998,87004,92726,92739,92745,
92751,92764,
                      92927,92733,92758])
        i=0
```

```
        while len(data) < 36 and i < 36 :
            for file in os.listdir(data_path):
                if i == 36:
                    i = 36
                elif modality in file and str(indexes[i]) in file:
                    i=i+1
                    file_name = os.path.join(data_path,file)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)
        print(data[35].shape)
        print(data[30].shape)
        print(data[20].shape)
        print(data[1].shape)
        return(data)

    def dataorganizesymm(array,mode,modality,linear):
        data = []
        if linear == False:
            if mode == "Training":
                if modality == "Flair":
                    symmdata = process.load_symmdata("/home/UOTTAWA/krain033/homLR","Flair")
                if modality == "T1":
                    symmdata =
process.load_symmdata("/home/UOTTAWA//SymmCNN/homLR/T1all","T1")
                for k in range(0,28):
                    data.append(symmdata[array[k]-1])
            if mode == "Test":
                if modality == "Flair":
                    symmdata =
process.load_symmdata_test("/home/UOTTAWA//SymmCNN/homLR/FlairTesting","Flair")
                if modality == "T1":
                    symmdata =
process.load_symmdata_test("/home/UOTTAWA//SymmCNN/homLR/T1all","T1")
                for k in range(0,36):
                    data.append(symmdata[array[k]-1])
        if linear == True:
            if mode == "Training":
                if modality == "Flair":
                    symmdata =
process.load_symmdata("/home/UOTTAWA//SymmCNN/homLR_linear/allFlair","Flair")
                if modality == "T1":
                    symmdata =
process.load_symmdata("/home/UOTTAWA//SymmCNN/homLR_linear/allT1","T1")
                for k in range(0,28):
                    data.append(symmdata[array[k]-1])

            if mode == "Test":
                if modality == "Flair":
```

```
                symmdata =
process.load_symmdata_test("/home/UOTTAWA//SymmCNN/homLR_linear/allFlair","Flair")
            if modality == "T1":
                symmdata =
process.load_symmdata_test("/home/UOTTAWA//SymmCNN/homLR_linear/allT1","T1")
            for k in range(0,36):
                data.append(symmdata[array[k]-1])

        return(data)

    def load_labels(data_path, range1, range2):

        labels = []
        for brain_id in range(range1, range2+1):
            if brain_id < 10:
                brain_id = '0' + str(brain_id)
            folder = os.path.join(data_path, str(brain_id))
            for file in os.listdir(folder):
                if "OT" in file:
                    filefile = file + '/' + file +'.nii'
                    file_name = os.path.join(folder, filefile)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    labels.append(np_image)

        return labels

    def standard_normalization(data):
        """
        Here the input is a list of brains and the function normalize each of this brains
        to have a mean 0 and standard deviation 1.
        """
        normalized = []
        for image in data:
            m = np.mean(image)
            image = image - m
            st_dev = np.std(image)
            image = image/st_dev
            normalized.append(image)
        return normalized

    def one_hot_encode(labels, number_of_labels=2):
        """
        One hot encode a list of sample labels. Return a one-hot encoded vector for each label.
        : labels: List of sample Labels
        : return: Numpy array of one-hot encoded labels
        """
        labels = np.array(labels)
        one_hot = np.zeros((labels.size, number_of_labels), dtype = np.int)
```

```
        one_hot[np.arange(labels.size), labels] = 1
        return one_hot

    def one_hot_encode_dense(labels, number_of_labels=2):
        """
        One hot encode a list of sample labels. Return a one-hot encoded vector for each label.
        : labels: List of sample Labels
        : return: Numpy array of one-hot encoded labels
        """
        labels = np.array(labels)
        one_hot = np.zeros(labels.shape + (2,), dtype = np.int)
        one_hot[process.all_idx(labels, axis=3)] = 1
        return one_hot

    def all_idx(idx, axis):
        grid = np.ogrid[tuple(map(slice, idx.shape))]
        grid.insert(axis, idx)
        return tuple(grid)



    def
get_uniform_samples_dense_batch(data_F_symm,data_D_symm,data_T1_symm,data_T2_symm,data_
F,data_D, data_T1,data_T2, labels, from_brains , size , dim , dim2, archsize,symmetry,joint,marginal):
        """
        Parameters:
        data: list of 3D numpy arrays representing an image of the brain
        labels: list of 3D numpy array representing a voxelwise segmentation of the brain in data
        from_brains:
        size: number of extracted patches
        dim: patch has a shape (dim, dim)

        Returns: tuple, namely, (array of patches, array of labels of the central voxels in the patches).

        There are equal number of samples labeled 1 and 0.
        """
        batch_data_F = []
        batch_data_D = []
        batch_data_T1 = []
        batch_data_T2 = []
        batch_data_F_symm = []
        batch_data_D_symm = []
        batch_data_T1_symm = []
        batch_data_T2_symm = []

        batch_data_F2 = []
        batch_data_D2 = []
        batch_data_T12 = []
        batch_data_T22 = []
```

```
        batch_data_F_symm2 = []
        batch_data_D_symm2 = []
        batch_data_T1_symm2 = []
        batch_data_T2_symm2 = []

        batch_labels = []
        while len(batch_data_F) < size/2:
            index = randint(0,from_brains)
            found = 0
            while found == 0:
                x = randint(int(dim2/2),(data_F[index].shape[0] - 1) - (int(dim2/2) + 1) + 1 + 1)
                y = randint(int(dim2/2),(data_F[index].shape[0] - 1) - (int(dim2/2) + 1) + 1 + 1)
                z = randint(int(dim2/2),(data_F[index].shape[2] - 1) - (int(dim2/2) + 1) + 1 + 1)
                if data_F[index][x][y][z] != 0 and labels[index][x][y][z] == 0:
                    found = 1
                    patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    """
                    when you subet an array the upper dimension is + 1 above the true size
                    """
                    batch_data_F.append(patch_F)
                    patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    batch_data_D.append(patch_D)
                    patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                    batch_data_T1.append(patch_T1)
                    patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                    batch_data_T2.append(patch_T2)

                    patch_F = data_F_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                    batch_data_F_symm.append(patch_F)
                    patch_D = data_D_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                    batch_data_D_symm.append(patch_D)
                    patch_T1 = data_T1_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                    batch_data_T1_symm.append(patch_T1)
                    patch_T2 = data_T2_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                    batch_data_T2_symm.append(patch_T2)


                    patch_F = data_F[index][x-int(dim2/2):x+int(dim2/2) + 1, y-int(dim2/2):y+int(dim2/2)
+ 1, z]
                    batch_data_F2.append(patch_F)
```

```
                patch_D = data_D[index][x-int(dim2/2):x+int(dim2/2) + 1, y-int(dim2/2):y+int(dim2/2)
+ 1, z]
                batch_data_D2.append(patch_D)

                patch_T1 = data_T1[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T12.append(patch_T1)

                patch_T2 = data_T1[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T22.append(patch_T2)

                patch_F = data_F_symm[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_F_symm2.append(patch_F)

                patch_D = data_D_symm[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_D_symm2.append(patch_D)

                patch_T1 = data_T1_symm[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T1_symm2.append(patch_T1)

                patch_T2 = data_T2_symm[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T2_symm2.append(patch_T2)


                label = labels[index][x - int(dim/2) + int(archsize/2): x + int(dim/2) - int(archsize/2) +
1,y - int(dim/2) + int(archsize/2): y + int(dim/2) - int(archsize/2) + 1 ,z]
                batch_labels.append(label)
        while len(batch_data_F) < size:
            index = randint(0,from_brains)
            found = 0
            while found == 0:
                x = randint(int(dim2/2),(data_F[index].shape[0] - 1) - (int(dim2/2) + 1) + 1 + 1)
                y = randint(int(dim2/2),(data_F[index].shape[0] - 1) - (int(dim2/2) + 1) + 1 + 1)
                z = randint(0, (data_F[index].shape[2]))
                if data_F[index][x][y][z] != 0 and labels[index][x][y][z] == 1:
                    found = 1
                    patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    """
                    when you subet an array the upper dimension is + 1 above the true size
                    """
                    batch_data_F.append(patch_F)
                    patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
```

```
                batch_data_D.append(patch_D)
                patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                batch_data_T1.append(patch_T1)
                patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                batch_data_T2.append(patch_T2)

                patch_F = data_F_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_F_symm.append(patch_F)
                patch_D = data_D_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_D_symm.append(patch_D)
                patch_T1 = data_T1_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_T1_symm.append(patch_T1)
                patch_T2 = data_T2_symm[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]
                batch_data_T2_symm.append(patch_T2)


                patch_F = data_F[index][x-int(dim2/2):x+int(dim2/2) + 1, y-int(dim2/2):y+int(dim2/2)
+ 1, z]
                batch_data_F2.append(patch_F)

                patch_D = data_D[index][x-int(dim2/2):x+int(dim2/2) + 1, y-int(dim2/2):y+int(dim2/2)
+ 1, z]
                batch_data_D2.append(patch_D)

                patch_T1 = data_T1[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T12.append(patch_T1)

                patch_T2 = data_T1[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T22.append(patch_T2)

                patch_F = data_F_symm[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_F_symm2.append(patch_F)

                patch_D = data_D_symm[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_D_symm2.append(patch_D)

                patch_T1 = data_T1_symm[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T1_symm2.append(patch_T1)
```

```
                patch_T2 = data_T2_symm[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T2_symm2.append(patch_T2)


                label = labels[index][x - int(dim/2) + int(archsize/2): x + int(dim/2) - int(archsize/2) +
1,y - int(dim/2) + int(archsize/2): y + int(dim/2) - int(archsize/2) + 1 ,z]
                batch_labels.append(label)
        batch_data_F = np.array(batch_data_F)
        batch_data_D = np.array(batch_data_D)
        batch_data_T1 = np.array(batch_data_T1)
        batch_data_T2 = np.array(batch_data_T2)
        batch_data_F_symm = np.array(batch_data_F_symm)
        batch_data_D_symm = np.array(batch_data_D_symm)
        batch_data_T1_symm = np.array(batch_data_T1_symm)
        batch_data_T2_symm = np.array(batch_data_T2_symm)

        batch_data_F2 = np.array(batch_data_F2)
        batch_data_D2 = np.array(batch_data_D2)
        batch_data_T12 = np.array(batch_data_T12)
        batch_data_T22 = np.array(batch_data_T22)
        batch_data_F_symm2 = np.array(batch_data_F_symm2)
        batch_data_D_symm2 = np.array(batch_data_D_symm2)
        batch_data_T1_symm2 = np.array(batch_data_T1_symm2)
        batch_data_T2_symm2 = np.array(batch_data_T2_symm2)


        # batch_labels = np.array(process.one_hot_encode(batch_labels))
        # batch_labels = np.array(batch_labels)

        batch_labels = np.array(process.one_hot_encode_dense(batch_labels))
        #   randomize = np.arange(size)
        #   np.random.shuffle(randomize)
        if symmetry == True:
            if joint == True:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_F_symm,batch_data_
D_symm,batch_data_T1_symm,batch_data_T2_symm])
                samples2 =
np.array([batch_data_F2,batch_data_D2,batch_data_T12,batch_data_T22,batch_data_F_symm2,batch_
data_D_symm2,batch_data_T1_symm2,batch_data_T2_symm2])
            if marginal == "Flair" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_F_symm])
                samples2 =
np.array([batch_data_F2,batch_data_D2,batch_data_T12,batch_data_T22,batch_data_F_symm2])
            if marginal == "DWI" and joint == False:
                samples =
```

```
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_D_symm])
                samples2 =
np.array([batch_data_F2,batch_data_D2,batch_data_T12,batch_data_T22,batch_data_D_symm2])
            if marginal == "T1" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_T1_symm])
                samples2 =
np.array([batch_data_F2,batch_data_D2,batch_data_T12,batch_data_T22,batch_data_T1_symm2])
            if marginal == "T2" and joint == False:
                samples =
np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2,batch_data_T2_symm])
                samples2 =
np.array([batch_data_F2,batch_data_D2,batch_data_T12,batch_data_T22,batch_data_T2_symm2])

        if symmetry == False:
            samples = np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2])
            samples2 = np.array([batch_data_F2,batch_data_D2,batch_data_T12,batch_data_T22])


        return samples,samples2,batch_labels

    def create_new_conv_layer(input_data, num_input_channels, num_filters, filter_shape,
pool_shape, K, name, patch_size, addpool, keep_prob):
        # setup the filter input shape for tf.nn.conv_2d
        conv_filt_shape = [filter_shape[0], filter_shape[1], num_input_channels,num_filters]

        # initialise weights and bias for the filter
        weights = tf.Variable(tf.random_normal(conv_filt_shape, mean=0.0,stddev=(2/
(num_input_channels*filter_shape[0]*filter_shape[1]))**(1/2), seed = 1967),name=name+'_W')
        bias = tf.Variable(tf.zeros([num_filters], name=name+'_b'))

        # setup the convolutional layer operation
        if addpool == False:
            out_layer = tf.nn.conv2d(input_data, weights, [1, 1, 1, 1],padding = 'VALID')

            # add the bias
            out_layer += bias

            # apply a ReLU non-linear activation
            out_layer = tf.nn.leaky_relu(out_layer)
            out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1992)
            return out_layer
        if addpool == True:
            out_layer = tf.nn.conv2d(input_data, weights, [1, 1, 1, 1],padding = 'VALID')

            # add the bias
            out_layer += bias

            # apply a ReLU non-linear activation
```

```
            out_layer = tf.nn.leaky_relu(out_layer)
            # now perform max pooling


            out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1995)
            return out_layer



        def
crossval(linear,symmtrainingmodality,symmtestingmodality,barray,j,nval,ntrain,ntest,mu,d,g1,g2,b,n,la
mbda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsize2,pi,sym
metry,joint,marginal):
            """
            for jval in range(1,j+1):
               print("Sample:",j)
               array = process.randomvaltrain(choicearray = barray , nbrains=28, testsize = 0, valsize = 3)
               print("Array Setup:",array)
            """
            seed(13031952)
            array = np.array([15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,1,28,7,10,23,6,16,24,18,9,2,4])
            print("Fold 1",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsiz
e2,pi,symmetry,joint,marginal)
            """
            seed(13031953)
            array = np.array([17,19,22,21,15,20,14,8,25,5,13,26,12,3,11,27,1,28,7,10,23,6,16,24,18,9,2,4])
            print("Fold 2",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsiz
e2,pi,symmetry,joint,marginal)
            seed(13031953)
            array = np.array([25,5,13,26,15,20,14,8,17,19,22,21,12,3,11,27,1,28,7,10,23,6,16,24,18,9,2,4])
            print("Fold 3",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsiz
e2,pi,symmetry,joint,marginal)
            seed(13031953)
            array = np.array([12,3,11,27,15,20,14,8,17,19,22,21,25,5,13,26,1,28,7,10,23,6,16,24,18,9,2,4])
            print("Fold 4",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsiz
e2,pi,symmetry,joint,marginal)
            seed(13031953)
```

```
        array = np.array([1,28,7,10,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,23,6,16,24,18,9,2,4])
        print("Fold 5",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsiz
e2,pi,symmetry,joint,marginal)
        seed(13031953)
        array = np.array([23,6,16,24,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,1,28,7,10,18,9,2,4])
        print("Fold 6",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsiz
e2,pi,symmetry,joint,marginal)
        seed(13031953)
        array = np.array([18,9,2,4,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,1,28,7,10,23,6,16,24])
        print("Fold 7",array)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsiz
e2,pi,symmetry,joint,marginal)
        """

    def
test(linear,symmtrainingmodality,symmtestingmodality,barray,nval,ntrain,ntest,j,mu,d,g1,g2,b,n,lambda
1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondphase,tsize,pi,dro
p1,drop2,drop3,drop4,symmetry,joint,marginal):
        seed(13031953)
        array = np.arange(1,29)

process.mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1
,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,secondph
ase,tsize,pi,drop1,drop2,drop3,drop4,symmetry,joint,marginal)


    """Kevin Raina"""
    def
mainrandom(linear,symmtrainingmodality,symmtestingmodality,array,nval,ntrain,ntest,mu,d,g1,g2,b,n,l
ambda1,lambda2,epochs,patch,checkpoint,dsize,costinit,costcheckpt,burnin1,burnin2,tsize,tsize2,pi,sy
mmetry,joint,marginal):
        for v in range(1,2):
            print("Validation: Fold #",v)

            if nval != 0:
                data_F_T = process.dataorganize(array,ntest,nval,ntrain,"Flair","Training")
                data_D_T = process.dataorganize(array,ntest,nval,ntrain,"DWI","Training")
                data_T1_T = process.dataorganize(array,ntest,nval,ntrain,"T1","Training")
                data_T2_T = process.dataorganize(array,ntest,nval,ntrain,"T2","Training")
                labels_T = process.labelorganize(array,ntest,nval,ntrain,"Training")
```

```
    data_F_V = process.dataorganize(array,ntest,nval,ntrain,"Flair","Validation")
    data_D_V = process.dataorganize(array,ntest,nval,ntrain,"DWI","Validation")
    data_T1_V = process.dataorganize(array,ntest,nval,ntrain,"T1","Validation")
    data_T2_V = process.dataorganize(array,ntest,nval,ntrain,"T2","Validation")
    labels_V = process.labelorganize(array,ntest,nval,ntrain,"Validation")

if nval ==0:
    data_F_T = process.dataorganize(array,0,nval,ntrain,"Flair","Training")
    data_D_T = process.dataorganize(array,0,nval,ntrain,"DWI","Training")
    data_T1_T = process.dataorganize(array,0,nval,ntrain,"T1","Training")
    data_T2_T = process.dataorganize(array,0,nval,ntrain,"T2","Training")
    labels_T = process.labelorganize(array,0,nval,ntrain,"Training")
    array2 = np.arange(1,37)
    data_F_V = process.dataorganize(array2,ntest,nval,0,"Flair","Test")
    data_D_V = process.dataorganize(array2,ntest,nval,0,"DWI","Test")
    data_T1_V = process.dataorganize(array2,ntest,nval,0,"T1","Test")
    data_T2_V = process.dataorganize(array2,ntest,nval,0,"T2","Test")



if symmetry == True:
    print("Loading symmetric data...")
    if nval !=0:
        if linear == False:
            if symmtrainingmodality == "Flair":
                SymmData = process.dataorganizesymm(array,"Training","Flair",linear)
            if symmtrainingmodality == "T1":
                SymmData = process.dataorganizesymm(array,"Training","T1",linear)
        if linear == True:
            if symmtrainingmodality == "Flair":
                SymmData = process.dataorganizesymm(array,"Training","Flair",linear)
            if symmtrainingmodality == "T1":
                SymmData = process.dataorganizesymm(array,"Training","T1",linear)

    if nval == 0:
        if linear == False:
            if symmtrainingmodality == "T1":
                SymmDataTrain = process.dataorganizesymm(array,"Training","T1",linear)
            if symmtrainingmodality == "Flair":
                SymmDataTrain = process.dataorganizesymm(array,"Training","Flair",linear)
            if symmtestingmodality == "T1":
                SymmDataTest = process.dataorganizesymm(array2,"Test","T1",linear)
            if symmtestingmodality == "Flair":
                SymmDataTest = process.dataorganizesymm(array2,"Test","Flair",linear)
        if linear == True:
            if symmtrainingmodality == "Flair":
                SymmDataTrain = process.dataorganizesymm(array,"Training","Flair",linear)
            if symmtestingmodality == "Flair":
                SymmDataTest = process.dataorganizesymm(array2,"Test","Flair",linear)
```

```
                    if symmtrainingmodality == "T1":
                        SymmDataTrain = process.dataorganizesymm(array,"Training","T1",linear)
                    if symmtestingmodality == "T1":
                        SymmDataTest = process.dataorganizesymm(array2,"Test","T1",linear)


                print("Done Loading")
                if joint == True:
                    print("Building Symmetric Features...")
                    if nval != 0:
                        data_F_V_symm =process.buildsymm(data_F_V,SymmData,nval,ntrain,"Validation")
                        data_D_V_symm
=process.buildsymm(data_D_V,SymmData,nval,ntrain,"Validation")
                        data_T1_V_symm
=process.buildsymm(data_T1_V,SymmData,nval,ntrain,"Validation")
                        data_T2_V_symm
=process.buildsymm(data_T2_V,SymmData,nval,ntrain,"Validation")

                        data_F_T_symm =process.buildsymm(data_F_T,SymmData,nval,ntrain,"Training")
                        data_D_T_symm =process.buildsymm(data_D_T,SymmData,nval,ntrain,"Training")
                        data_T1_T_symm
=process.buildsymm(data_T1_T,SymmData,nval,ntrain,"Training")
                        data_T2_T_symm
=process.buildsymm(data_T2_T,SymmData,nval,ntrain,"Training")

                    if nval == 0:
                        data_F_V_symm =process.buildsymm(data_F_V,SymmDataTest,0,ntest,"Training")
                        data_D_V_symm =process.buildsymm(data_D_V,SymmDataTest,0,ntest,"Training")
                        data_T1_V_symm
=process.buildsymm(data_T1_V,SymmDataTest,0,ntest,"Training")
                        data_T2_V_symm
=process.buildsymm(data_T2_V,SymmDataTest,0,ntest,"Training")

                        data_F_T_symm =process.buildsymm(data_F_T,SymmDataTrain,0,ntrain,"Training")
                        data_D_T_symm
=process.buildsymm(data_D_T,SymmDataTrain,0,ntrain,"Training")
                        data_T1_T_symm
=process.buildsymm(data_T1_T,SymmDataTrain,0,ntrain,"Training")
                        data_T2_T_symm
=process.buildsymm(data_T2_T,SymmDataTrain,0,ntrain,"Training")

                    mods=8

                if marginal == "Flair" and joint == False:
                    data_F_V_symm =process.buildsymm(data_F_V,SymmData,nval,ntrain,"Validation")
                    data_F_T_symm =process.buildsymm(data_F_T,SymmData,nval,ntrain,"Training")
                    mods = 5
                if marginal == "DWI" and joint == False:
                    data_D_V_symm =process.buildsymm(data_D_V,SymmData,nval,ntrain,"Validation")
```

```
                data_D_T_symm =process.buildsymm(data_D_T,SymmData,nval,ntrain,"Training")
                mods = 5
            if marginal == "T1" and joint == False:
                data_T1_V_symm =process.buildsymm(data_T1_V,SymmData,nval,ntrain,"Validation")
                data_T1_T_symm =process.buildsymm(data_T1_T,SymmData,nval,ntrain,"Training")
                mods = 5
            if marginal == "T2" and joint == False:
                data_T2_V_symm =process.buildsymm(data_T2_V,SymmData,nval,ntrain,"Validation")
                data_T2_T_symm =process.buildsymm(data_T2_T,SymmData,nval,ntrain,"Training")
                mods = 5
        if symmetry == False:
            mods = 4


        """
        data_D_V_symm =process.buildsymm(data_D_V,SymmData,nval,ntrain,"Validation")
        data_T1_V_symm =process.buildsymm(data_T1_V,SymmData,nval,ntrain,"Validation")
        data_T2_V_symm =process.buildsymm(data_T2_V,SymmData,nval,ntrain,"Validation")

        data_D_T_symm =process.buildsymm(data_D_T,SymmData,nval,ntrain,"Training")
        data_T1_T_symm =process.buildsymm(data_T1_T,SymmData,nval,ntrain,"Training")
        data_T2_T_symm =process.buildsymm(data_T2_T,SymmData,nval,ntrain,"Training")
        """



        tf.reset_default_graph()
        x = tf.placeholder(tf.float32)

        x2 = tf.placeholder(tf.float32,shape=[None,None,None,None])
        y = tf.placeholder(tf.float32)
        probl1 = tf.placeholder_with_default(1.0,shape=())
        probl2 = tf.placeholder_with_default(1.0,shape=())
        #x2.set_shape([10,75,75,4])
        print(tf.shape(x2),tf.shape(x2))
        x2down = tf.image.resize_nearest_neighbor(x2,tf.cast(tf.stack([tf.shape(x2)[1]/3,tf.shape(x2)
[2]/3]),tf.int32),True)
        #x2.set_shape([None,None,None,None])
        """patch-6,patch-11,patch-12,1,[patch-12,patch-12], 24,21,21,1,[21,21]"""
        """4,64,7,4,64,64,3,2,4,160,13,13,1,224,2,1,1 """
        layer11 = process.create_new_conv_layer(x,mods,60,[3,3],[1,1],2,'layer11',patch-
2,True,probl1)
        layer12 = process.create_new_conv_layer(layer11,60,60,[3,3],[2,2],2,'layer12',patch-
11,True,probl1)
        layer13 = process.create_new_conv_layer(layer12,60,80,[3,3],[1,1],2,'layer13',patch-
12,False,probl1)
        layer14 = process.create_new_conv_layer(layer13,80,80,[3,3],[1,1],2,'layer14',patch-
12,False,probl1)
        layer15 = process.create_new_conv_layer(layer14,80,80,[3,3],[1,1],2,'layer15',patch-
```

```
12,False,probl1)
        layer16 = process.create_new_conv_layer(layer15,80,80,[3,3],[1,1],2,'layer16',patch-
12,False,probl1)
        layer17 = process.create_new_conv_layer(layer16,80,100,[3,3],[1,1],2,'layer17',patch-
12,False,probl1)
        layer18 = process.create_new_conv_layer(layer17,100,100,[3,3],[1,1],2,'layer18',patch-
12,False,probl1)

        layer21 = process.create_new_conv_layer(x2down,mods,60,[3,3],[1,1],2,'layer21',patch-
2,True,probl1)
        layer22 = process.create_new_conv_layer(layer21,60,60,[3,3],[2,2],2,'layer22',patch-
11,True,probl1)
        layer23 = process.create_new_conv_layer(layer22,60,80,[3,3],[1,1],2,'layer23',patch-
12,False,probl1)
        layer24 = process.create_new_conv_layer(layer23,80,80,[3,3],[1,1],2,'layer24',patch-
12,False,probl1)
        layer25 = process.create_new_conv_layer(layer24,80,80,[3,3],[1,1],2,'layer25',patch-
12,False,probl1)
        layer26 = process.create_new_conv_layer(layer25,80,80,[3,3],[1,1],2,'layer26',patch-
12,False,probl1)
        layer27 = process.create_new_conv_layer(layer26,80,100,[3,3],[1,1],2,'layer27',patch-
12,False,probl1)
        layer28 = process.create_new_conv_layer(layer27,100,100,[3,3],[1,1],2,'layer28',patch-
12,False,probl1)
        layer28up = tf.image.resize_nearest_neighbor(layer28,[tf.shape(layer28)
[1]*3,tf.shape(layer28)[2]*3],True)
        conc = tf.concat([layer28up, layer18], axis = 3)
        layerf1 = process.create_new_conv_layer(conc,200,200,[1,1],[1,1],2,'layerf1',1,False,probl2)
        layerf2 = process.create_new_conv_layer(layerf1,200,200,[1,1],
[1,1],2,'layerf2',1,False,probl2)
        layerf = process.create_new_conv_layer(layerf2,200,2,[1,1],[1,1],1,'layerf',1,False,probl1)
        logits = tf.squeeze(layerf)
        """layer 4: 1 by 198 by 198 by 2, logits: 198 by 198 by 2"""
        y_ = tf.nn.softmax(logits)
        """ 154 by 230 by 230 by 2 m,idx,count = tf.unique_with_counts(tf.argmax(y,3))"""

        reg1 = tf.reduce_sum(tf.abs(tf.trainable_variables()[0])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[2])) + tf.reduce_sum(tf.abs(tf.trainable_variables()[4])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[6])) + tf.reduce_sum(tf.abs(tf.trainable_variables()[8])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[10])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[12]))
+ tf.reduce_sum(tf.abs(tf.trainable_variables()[14])) +tf.reduce_sum(tf.abs(tf.trainable_variables()
[16])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[18]))
+tf.reduce_sum(tf.abs(tf.trainable_variables()[20])) + tf.reduce_sum(tf.abs(tf.trainable_variables()
[22])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[24])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[26])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[28]))
+ tf.reduce_sum(tf.abs(tf.trainable_variables()[30])) +tf.reduce_sum(tf.abs(tf.trainable_variables()
[32])) + tf.reduce_sum(tf.abs(tf.trainable_variables()[34])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[36]))
```

```
        reg2 = tf.nn.l2_loss(tf.trainable_variables()[0]) + tf.nn.l2_loss(tf.trainable_variables()[2])
+tf.nn.l2_loss(tf.trainable_variables()[4]) + tf.nn.l2_loss(tf.trainable_variables()[6]) +
tf.nn.l2_loss(tf.trainable_variables()[8]) + tf.nn.l2_loss(tf.trainable_variables()[10])
+tf.nn.l2_loss(tf.trainable_variables()[12]) + tf.nn.l2_loss(tf.trainable_variables()[14])
+tf.nn.l2_loss(tf.trainable_variables()[16]) + tf.nn.l2_loss(tf.trainable_variables()[18])
+tf.nn.l2_loss(tf.trainable_variables()[20]) + tf.nn.l2_loss(tf.trainable_variables()[22])
+tf.nn.l2_loss(tf.trainable_variables()[24]) + tf.nn.l2_loss(tf.trainable_variables()[26])
+tf.nn.l2_loss(tf.trainable_variables()[28]) + tf.nn.l2_loss(tf.trainable_variables()[30]) +
tf.nn.l2_loss(tf.trainable_variables()[32]) + tf.nn.l2_loss(tf.trainable_variables()[34]) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[36]))



        cross_entropy= tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits =
logits,labels = y))
        loss = cross_entropy + lambda1*reg1 + lambda2*reg2
        global_step = tf.Variable(0,trainable = False)

        """
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=g1).minimize(loss)


        """
        learningprime = tf.train.piecewise_constant(global_step, boundaries =
[25000,39000,49000,59000,71000,75000],values = [g1,g1/2,g1/4,g1/8,g1/16,g1/32,g1/64])
        """
        learningprime = tf.train.exponential_decay(learning_rate = g1,global_step =
global_step,decay_steps = dsize,decay_rate = d)
        """
        optimizer=
tf.train.RMSPropOptimizer(learning_rate=learningprime,momentum=mu,epsilon=1e-4).minimize(loss,
global_step=global_step)



        paddedy_ = tf.pad(tf.argmax(y_,3),paddings = tf.constant([[0,0],[int(51/2),int(51/2)],
[int(51/2),int(51/2)]]),mode = "CONSTANT")
        tp = tf.metrics.true_positives(y,paddedy_,weights=None)[1]
        fp = tf.metrics.false_positives(y,paddedy_,weights=None)[1]
        fn = tf.metrics.false_negatives(y,paddedy_,weights=None)[1]



        init_op = tf.global_variables_initializer()
        init_l = tf.local_variables_initializer()
        with tf.Session() as sess:
            sess.run(init_op)
            sess.run(init_l)
```

```
            print("Training")
            total_batch = int(n)
            """
            switch = 0
            """
            for epoch in range(epochs):
               avg_cost = 0
               for i in range(1,total_batch + 1):

                  if (i/checkpoint).is_integer() and i > burnin1:
                     print("Overfit Checkpoint:",i/checkpoint)
                     print("Testing on Training Set:")


                     if nval == 0:
                        for m in range(1,ntest+1):
                           print("Validation Brain (not actual index)",m)
                           for s in range(0,2):
                              if symmetry ==True:
                                 if joint == True:
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1],data_D_V_symm[m-1],data_T1_V_symm[m-
1],data_T2_V_symm[m-1]])
                                 if joint == False and marginal == "Flair":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1]])
                                 if joint == False and marginal == "DWI":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_D_V_symm[m-1]])
                                 if joint == False and marginal == "T1":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_T1_V_symm[m-1]])
                                 if joint == False and marginal == "T2":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_T2_V_symm[m-1]])
                              if symmetry == False:
                                 data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1]])

                              """
                              Arbitrary Image Dimensions
                              xdim = data_F_V[m-1].shape[0]
                              ydim = data_F_V[m-1].shape[1]

                              i_x = mat.floor(xdim/3) - 16
                              i_y = mat.floor(ydim/3) - 16

                              xsize1 = 16 + 3*i_x
                              xsize2 = 3*(16 + i_x)
```

```
                        ysize1 = 16 + 3*i_y
                        ysize2 = 3*(16 + i_y)

                        if xsize1 % 2 == 0:
                           xlim1 = (xsize1 + xdim)/2
                        """
                        data_V_prime1 = np.transpose(data_V,(3,1,2,0))


                        dat2 = data_V_prime1[77*s:77*s + 77,1:229,1:229,:]
                        dat1 = data_V_prime1[77*s:77*s+77,17:213,17:213,:]

                        if s==0:
                           pred1 = sess.run(paddedy_, feed_dict = {x:dat1})
                           pred1 = np.asarray(pred1)
                           pred1 = np.transpose(pred1,(1,2,0))
                        if s==1:
                           pred2 = sess.run(paddedy_, feed_dict = {x:dat1})
                           pred2 = np.asarray(pred2)
                           pred2 = np.transpose(pred2,(1,2,0))
                           predf = np.concatenate((pred1,pred2),axis=2)
                           img_header = process.load_header("/data/SISS2015/Testing","Flair",
array2[m-1],array2[m-1])

                           predf1 = nib.Nifti1Image(predf,np.eye(4),img_header)
                           if symmetry == False:
                              nib.save(predf1,'VSD.BL_'+str(m)+'.nii')
                           if symmetry == True:
                              if linear == False:
                                 if symmtrainingmodality == "T1":
                                    nib.save(predf1,'VSD.NLS_T1_'+str(m)+'.nii')
                                 if symmtrainingmodality == "Flair":
                                    nib.save(predf1,'VSD.NLS_Flair_'+str(m)+'.nii')
                              if linear == True:
                                 if symmtrainingmodality == "T1":
                                    nib.save(predf1,'VSD.LS_T1_'+str(m)+'.nii')
                                 if symmtrainingmodality == "Flair":
                                    nib.save(predf1,'VSD.LS_Flair_'+str(m)+'.nii')
                     if nval != 0:
                        for m in range(1,nval+1):
                           print("Validation Brain (not actual index)",m)
                           for s in range(0,2):
                              labels_Temp1 = labels_V[m-1]
                              labels_Temp_Prime1 = np.transpose(labels_Temp1,(2,0,1))
                              lab1 = labels_Temp_Prime1[77*s:77*s + 77,:,:]
                              if symmetry ==True:
                                 if joint == True:
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
```

```
1],data_T2_V[m-1],data_F_V_symm[m-1],data_D_V_symm[m-1],data_T1_V_symm[m-
1],data_T2_V_symm[m-1]])
                                if joint == False and marginal == "Flair":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_F_V_symm[m-1]])
                                if joint == False and marginal == "DWI":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_D_V_symm[m-1]])
                                if joint == False and marginal == "T1":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_T1_V_symm[m-1]])
                                if joint == False and marginal == "T2":
                                    data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1],data_T2_V_symm[m-1]])
                            if symmetry == False:
                                data_V = np.array([data_F_V[m-1],data_D_V[m-1],data_T1_V[m-
1],data_T2_V[m-1]])

                            data_V_prime1 = np.transpose(data_V,(3,1,2,0))
                            dat2 = data_V_prime1[77*s:77*s + 77,1:229,1:229,:]
                            dat1 = data_V_prime1[77*s:77*s+77,17:213,17:213,:]

                            if s==0:
                                pred1 = sess.run(paddedy_, feed_dict = {x:dat1,x2:dat2,y:lab1})
                                pred1 = np.asarray(pred1)
                                pred1 = np.transpose(pred1,(1,2,0))

                                pro1 = sess.run(y_,feed_dict = {x:dat1,x2:dat2,y:lab1})
                                pro1 = np.asarray(pro1)
                                pro1 = np.transpose(pro1,(1,2,0,3))
                            if s==1:
                                pred2 = sess.run(paddedy_, feed_dict = {x:dat1,x2:dat2,y:lab1})
                                pred2 = np.asarray(pred2)
                                pred2 = np.transpose(pred2,(1,2,0))

                                pro2 = sess.run(y_,feed_dict = {x:dat1,x2:dat2,y:lab1})
                                pro2 = np.asarray(pro2)
                                pro2 = np.transpose(pro2,(1,2,0,3))

                                predf = np.concatenate((pred1,pred2),axis=2)
                                predf1 = nib.Nifti1Image(predf,np.eye(4))

                                prof = np.concatenate((pro1,pro2),axis=2)
                                prof = nib.Nifti1Image(prof,np.eye(4))
                                #nib.save(predf1,'segment.nii')

                                if symmetry == False:
                                    nib.save(predf1,'VSD.BL_TRN_'+str(m)+'.nii')
                                    nib.save(prof,'Probabilities'+str(m)+'.nii')
```

```
                    if symmetry == True:
                        if linear == False:
                            if symmtrainingmodality == "T1":
                                nib.save(predf1,'VSD.NLS_TRN_T1'+str(m)+'.nii')
                            if symmtrainingmodality == "Flair":
                                nib.save(predf1,'VSD.NLS_TRN_Flair'+str(m)+'.nii')
                                nib.save(logif1,'Probabilities'+str(m)+'.nii')
                        if linear == True:
                            if symmtrainingmodality == "T1":
                                nib.save(predf1,'VSD.LS_TRN_T1'+str(m)+'.nii')
                            if symmtrainingmodality == "Flair":
                                nib.save(predf1,'VSD.LS_TRN_Flair'+str(m)+'.nii')

                    trup1 = sess.run(tp, feed_dict = {x:dat1,x2:dat2,y:lab1})
                    trup1 = np.asarray(trup1)
                    falp1 = sess.run(fp, feed_dict = {x:dat1,x2:dat2,y:lab1})
                    falp1 = np.asarray(falp1)
                    faln1 = sess.run(fn, feed_dict = {x:dat1,x2:dat2,y:lab1})
                    faln1 = np.asarray(faln1)
                    np.save(file = "tp" + str(m) + str(s+1) + ".npy", arr = trup1)
                    np.save(file = "fp" + str(m) + str(s+1) + ".npy", arr = falp1)
                    np.save(file = "fn" + str(m) + str(s+1) + ".npy", arr = faln1)
                    sess.run(init_l)
                print(process.dice(m))


            if symmetry == True:
                if joint == True:
                    train_samples =
process.get_uniform_samples_dense_batch(data_F_T_symm,data_D_T_symm,data_T1_T_symm,data
_T2_T_symm,data_F_T,data_D_T, data_T1_T,
data_T2_T,labels_T,ntrain,b,tsize,tsize2,patch,symmetry,joint,marginal)
                if joint == False and marginal == "Flair":
                    train_samples =
process.get_uniform_samples_dense_batch(data_F_T_symm,data_F_T,data_F_T,data_F_T,data_F_T,d
ata_D_T, data_T1_T, data_T2_T,labels_T,ntrain,b,tsize,tsize2,patch,symmetry,joint,marginal)
                if joint == False and marginal == "DWI":
                    train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_D_T_symm,data_F_T,data_F_T,data_F_T,d
ata_D_T, data_T1_T, data_T2_T,labels_T,ntrain,b,tsize,tsize2,patch,symmetry,joint,marginal)
                if joint == False and marginal == "T1":
                    train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_T1_T_symm,data_F_T,data_F_T,
data_D_T, data_T1_T, data_T2_T,labels_T,ntrain,b,tsize,tsize2,patch,symmetry,joint,marginal)
                if joint == False and marginal == "T2":
                    train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_F_T,data_T2_T_symm,data_F_T,
data_D_T, data_T1_T, data_T2_T,labels_T,ntrain,b,tsize,tsize2,patch,symmetry,joint,marginal)
```

```
                    if symmetry == False:
                        train_samples =
process.get_uniform_samples_dense_batch(data_F_T,data_F_T,data_F_T,data_F_T,data_F_T,data_D_
T, data_T1_T, data_T2_T,labels_T,ntrain,b,tsize,tsize2,patch,symmetry,joint,marginal)

                    train_array = train_samples[0]
                    train_array2 = train_samples[1]
                    train_array = np.transpose(train_array,(1,2,3,0))
                    train_array2 = np.transpose(train_array2,(1,2,3,0))
                    arrlab = train_samples[2]
                    dict = {x : train_array ,x2: train_array2, y : arrlab, probl1:1,probl2:0.5}

                    temploss = sess.run(loss, feed_dict = dict)
                    _,c = sess.run([optimizer,loss],feed_dict=dict)
                    avg_cost += temploss/costcheckpt

                    if i < costinit or (i/costcheckpt).is_integer():
                       print(i,avg_cost)
                       if (i/costcheckpt).is_integer():
                          avg_cost = 0
                    carr = np.array([c])
                    if np.isnan(carr) == True:
                       print ("Cost is Infinite")



            sess.close()


    def buildsymm(data_M_V,SymmData,nval,ntrain,mode):
        data = []
        if mode == "Validation":
           a = 0
           b = nval
        if mode == "Training":
           a = nval
           b = ntrain + nval
        for n in range(a,b):
           print("Loading Checkpoint (%):",100*(n+1)/b)
           data.append(np.zeros((data_M_V[n-a].shape[0],data_M_V[n-a].shape[1],data_M_V[n-
a].shape[2])))
           for i in range(0,data_M_V[n-a].shape[0]):
              for j in range(0,data_M_V[n-a].shape[1]):
                 for k in range(0,data_M_V[n-a].shape[2]):
                    xsymm = int(min(229,max(0,round(SymmData[n][i,j,k,0,0]))))
                    ysymm = int(min(229,max(0,round(SymmData[n][i,j,k,0,1]))))
                    zsymm = int(min(data_M_V[n-a].shape[2]-1,max(0,round(SymmData[n][i,j,k,0,2]))))
                    data[n-a][i,j,k] = data_M_V[n-a][i,j,k] - data_M_V[n-a][xsymm,ysymm,zsymm]
```

```
        return(data)


    def dice (m):
        tp1 = np.load("tp" + str(m) + str(1) + ".npy")
        tp2 = np.load("tp" + str(m) + str(2) + ".npy")
        fn1 = np.load("fn" + str(m) + str(1) + ".npy")
        fn2 = np.load("fn" + str(m) + str(2) + ".npy")
        fp1 = np.load("fp" + str(m) + str(1) + ".npy")
        fp2 = np.load("fp" + str(m) + str(2) + ".npy")
        tp = int(tp1) + int(tp2)
        fn = int(fn1) + int(fn2)
        fp = int(fp1) + int(fp2)
        print("True Positive Rate:", tp/(tp+fn))
        print("TP:",tp)
        print("FN:",fn)
        print("FP:",fp)
        if int(fn1) == 0 and int(fn2) == 0 and int(fp1) ==0 and int(fp2) == 0:
            dice = "Division by Zero in Dice Coefficient"
        else:
            dice = 2*(int(tp1) + int(tp2))/(2*(int(tp1) + int(tp2)) + int(fn1) + int(fn2) + int(fp1) +
int(fp2))
        return(dice)

print("Wider 2-D Segmentation")
process.crossval(linear = False,symmtrainingmodality = "Flair",symmtestingmodality = "Flair",
barray=np.array([1]), j = 15, nval = 4,
            ntrain = 24, ntest = 0, mu = 0.6, d = 0.1, g1 = 0.001, g2= 0.001, b = 12, n = 80005,
            lambda1 = 0.00000001, lambda2 = 0.000001, epochs = 1, patch = 17, checkpoint = 10000,
dsize = 50000,
            costcheckpt = 1000, costinit = 100, burnin1 = 75000, burnin2 = 100,tsize = 43,tsize2 = 75
,pi=0.02,
            symmetry = False, joint = True, marginal = "T2")
"""
seed(13031953)
process.crossval(linear = False,symmtrainingmodality = "Flair",symmtestingmodality = "Flair",
barray=np.array([1]), j = 15, nval = 4,
            ntrain = 24, ntest = 0, mu = 0.6, d = 0.1, g1 = 0.001, g2= 0.001, b = 12, n = 20,
            lambda1 = 0.00000001, lambda2 = 0.000001, epochs = 1, patch = 17, checkpoint = 15, dsize
= 50000,
            costcheckpt = 1000, costinit = 100, burnin1 = 10, burnin2 = 100,tsize = 43,tsize2 = 75
,pi=0.02,
            symmetry = False, joint = True, marginal = "T2")
"""
```

Appendix C

PRCountNet3D Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 21 13:47:09 2019
CountNet
@author:  Kevin Raina
"""
import scipy
import nibabel as nib
import os
import math
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import numpy as np
from random import randint
from random import seed
from pylab import *

class process:



    def load_data(data_path , modality, range1,range2 ):
        """
        Parameters:
        data_path: path to the folder "Training" with brain images
        """
        data = []
        for brain_id in range(range1, range2+1):
            if brain_id < 10:
                brain_id = '0' + str(brain_id)
            folder = os.path.join(data_path, str(brain_id))
            for file in os.listdir(folder):
                if modality in file:
                    filefile = file + '/' + file +'.nii'
                    file_name = os.path.join(folder, filefile)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)
        normalized1 = process.standard_normalization(data)
        return normalized1


    def dataorganize(array,testsize,valsize,trainsize,modality,mode):
        data = []
        if mode == "Validation":
            for j in range(0,valsize):
                data.append(process.load_data("/data/SISS2015/Training",modality, array[j],array[j])[0])
        if mode == "Test":
            for j in range(valsize,valsize+testsize):
```

```
            data.append(process.load_data("/data/SISS2015/Training",modality, array[j],array[j])[0])
    if mode == "Training":
        for j in range(valsize + testsize, valsize + trainsize + testsize):
            data.append(process.load_data("/data/SISS2015/Training",modality, array[j],array[j])[0])
    return(data)

def labelorganize(array,testsize,valsize,trainsize,mode):
    labels = []
    if mode == "Validation":
        for k in range(0,valsize):
            labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
    if mode == "Test":
        for k in range(valsize,valsize + testsize):
            labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
    if mode == "Training":
        for k in range(valsize + testsize,valsize + testsize + trainsize):
            labels.append(process.load_labels("/data/SISS2015/Training",array[k],array[k])[0])
    return(labels)


def load_labels(data_path, range1, range2):

    labels = []
    for brain_id in range(range1, range2+1):
        if brain_id < 10:
            brain_id = '0' + str(brain_id)
        folder = os.path.join(data_path, str(brain_id))
        for file in os.listdir(folder):
            if "OT" in file:
                filefile = file + '/' + file +'.nii'
                file_name = os.path.join(folder, filefile)
                image = nib.load(file_name)
                np_image = image.get_data()
                labels.append(np_image)

    return labels

def standard_normalization(data):
    """
    Here the input is a list of brains and the function normalize each of this brains
    to have a mean 0 and standard deviation 1.
    """
    normalized = []
    for image in data:
        m = np.mean(image)
        image = image - m
        st_dev = np.std(image)
        image = image/st_dev
        normalized.append(image)
```

```
    return normalized

def one_hot_encode(labels, number_of_labels=2):
    """
    One hot encode a list of sample labels. Return a one-hot encoded vector for each label.
    : labels: List of sample Labels
    : return: Numpy array of one-hot encoded labels
    """
    labels = np.array(labels)
    one_hot = np.zeros((labels.size, number_of_labels), dtype = np.int)
    one_hot[np.arange(labels.size), labels] = 1
    return one_hot

def one_hot_encode_dense(labels, number_of_labels=2):
    """
    One hot encode a list of sample labels. Return a one-hot encoded vector for each label.
    : labels: List of sample Labels
    : return: Numpy array of one-hot encoded labels
    """
    labels = np.array(labels)
    one_hot = np.zeros(labels.shape + (2,), dtype = np.int)
    one_hot[process.all_idx(labels, axis=3)] = 1
    return one_hot

def all_idx(idx, axis):
    grid = np.ogrid[tuple(map(slice, idx.shape))]
    grid.insert(axis, idx)
    return tuple(grid)



def get_training_samples(data_F,data_D, data_T1,data_T2, labels, from_brains , size , dim):

    batch_data_F = []
    batch_data_D = []
    batch_data_T1 = []
    batch_data_T2 = []

    batch_indices = []
    batch_labels = []


    while len(batch_data_F) < size:
        sample = np.random.binomial(n=1,p=1)
        index = randint(0,from_brains)
        found = 0
        while found == 0:
            x = randint(int(dim/2),(data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
            y = randint(int(dim/2),(data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
```

```
            z = randint(int(dim/2),(data_F[index].shape[2] - 1) - (int(dim/2) + 1) + 1 + 1)
            #if np.count_nonzero(labels[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z]) != 0:
            if sample == 1:
               if labels[index][x,y,z] == 1:
                  found = 1
                  patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z-int(dim/2):z+int(dim/2)+1]

                  batch_data_F.append(patch_F)

                  patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z-int(dim/2):z+int(dim/2)+1]
                  batch_data_D.append(patch_D)
                  patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2)
+ 1, z-int(dim/2):z+int(dim/2)+1]
                  batch_data_T1.append(patch_T1)
                  patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2)
+ 1, z-int(dim/2):z+int(dim/2)+1]
                  batch_data_T2.append(patch_T2)

                  label = np.array([np.count_nonzero(labels[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z-int(dim/2):z+int(dim/2)+1])])
                  batch_labels.append(label)
                  batch_indices.append(np.array([x,y,z]))

            if sample == 0:
               if np.count_nonzero(labels[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z-int(dim/2):z+int(dim/2)+1]) == 0:
                  found = 1
                  patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z-int(dim/2):z+int(dim/2)+1]

                  batch_data_F.append(patch_F)

                  patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z-int(dim/2):z+int(dim/2)+1]
                  batch_data_D.append(patch_D)
                  patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2)
+ 1, z-int(dim/2):z+int(dim/2)+1]
                  batch_data_T1.append(patch_T1)
                  patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2)
+ 1, z-int(dim/2):z+int(dim/2)+1]
                  batch_data_T2.append(patch_T2)

                  label = np.array([np.count_nonzero(labels[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z-int(dim/2):z+int(dim/2)+1])])
                  batch_labels.append(label)
                  batch_indices.append(np.array([x,y,z]))
```

```
        batch_data_F = np.array(batch_data_F)

        batch_data_D = np.array(batch_data_D)
        batch_data_T1 = np.array(batch_data_T1)
        batch_data_T2 = np.array(batch_data_T2)
        batch_labels = np.array(batch_labels)
        batch_indices = np.array(batch_indices)

        # batch_labels = np.array(process.one_hot_encode(batch_labels))
        # batch_labels = np.array(batch_labels)

        #   randomize = np.arange(size)
        #   np.random.shuffle(randomize)
        samples = np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2])


        return samples,batch_labels,batch_indices

    def get_positive_samples(data_F,data_D, data_T1,data_T2, labels, from_brains , size , dim):

        batch_data_F = []
        batch_data_D = []
        batch_data_T1 = []
        batch_data_T2 = []


        batch_labels = []

        while len(batch_data_F) < size:
            index = randint(0,from_brains)
            found = 0
            while found == 0:
                x = randint(int(dim/2),(data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                y = randint(int(dim/2),(data_F[index].shape[0] - 1) - (int(dim/2) + 1) + 1 + 1)
                z = randint(int(dim/2),(data_F[index].shape[2] - 1) - (int(dim/2) + 1) + 1 + 1)

                if labels[index][x,y,z] == 1:

                    found = 1
                    #print("Sample",len(batch_data_F),index,x,y,z)
                    patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z-int(dim/2):z+int(dim/2)+1]
                    batch_data_F.append(patch_F)
                    patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z-int(dim/2):z+int(dim/2)+1]
                    batch_data_D.append(patch_D)
```

```
                patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z-int(dim/2):z+int(dim/2)+1]
                batch_data_T1.append(patch_T1)
                patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z-int(dim/2):z+int(dim/2)+1]
                batch_data_T2.append(patch_T2)



                label = np.array([np.count_nonzero(labels[index][x-int(dim/2):x+int(dim/2) + 1, y-
int(dim/2):y+int(dim/2) + 1, z-int(dim/2):z+int(dim/2)+1])])

                batch_labels.append(label)

        batch_data_F = np.array(batch_data_F)

        batch_data_D = np.array(batch_data_D)
        batch_data_T1 = np.array(batch_data_T1)
        batch_data_T2 = np.array(batch_data_T2)
        batch_labels = np.array(batch_labels)

        samples = np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2])


        return samples,batch_labels


    def create_new_conv_layer(input_data, num_input_channels, num_filters, filter_shape,
pool_shape, K, name, patch_size, addpool,maxout, keep_prob):
        # setup the filter input shape for tf.nn.conv_2d
        conv_filt_shape = [filter_shape[0], filter_shape[1], filter_shape[2],
num_input_channels,num_filters]

        # initialise weights and bias for the filter
        weights = tf.Variable(tf.random.normal(conv_filt_shape, mean=0.0,stddev = 0.001, seed =
1967),name=name+'_W')
        bias = tf.Variable(tf.zeros([num_filters]), name=name+'_b')

        if maxout == True:
           # setup the convolutional layer operation
           if addpool == False:
              out_layer = tf.nn.conv3d(input_data, weights, [1, 1, 1, 1, 1],padding = 'VALID')

              # add the bias
              out_layer += bias

              # apply a ReLU non-linear activation
              out_layer = tf.nn.leaky_relu(out_layer)
              out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1992)
```

```
                return out_layer
            if addpool == True:
                out_layer = tf.nn.conv3d(input_data, weights, [1, 1, 1, 1, 1],padding = 'VALID')

                # add the bias
                out_layer += bias

                # apply a ReLU non-linear activation
                out_layer = tf.nn.leaky_relu(out_layer)
                # now perform max pooling
                ksize = [1, pool_shape[0], pool_shape[1], pool_shape[2], 1]
                strides = [1, 1, 1, 1, 1]
                out_layer = tf.nn.max_pool3d(out_layer, ksize=ksize, strides=strides, padding = 'VALID')

                out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1995)
                return out_layer
        if maxout == False:
            # setup the convolutional layer operation
            if addpool == False:
                out_layer = tf.nn.conv3d(input_data, weights, [1, 1, 1, 1, 1],padding = 'VALID')

                # add the bias
                out_layer += bias

                # apply a ReLU non-linear activation
                out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1992)
                return out_layer
            if addpool == True:
                out_layer = tf.nn.conv3d(input_data, weights, [1, 1, 1, 1, 1],padding = 'VALID')

                # add the bias
                out_layer += bias

                # apply a ReLU non-linear activation
                # now perform max pooling
                ksize = [1, pool_shape[0], pool_shape[1], pool_shape[2], 1]
                strides = [1, 1, 1, 1, 1]
                out_layer = tf.nn.max_pool3d(out_layer, ksize=ksize, strides=strides, padding = 'VALID')

                out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1995)
                return out_layer


    def
crossval(barray,j,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,costi
nit,costcheckpt,burnin1,burnin2):
        """
        seed(13031952)
        array = np.array([15,20,14,8,2,6,7,17,19,22,21,25,5,13,26,12,3,11,27,1,28,10,23,16,24,18,9,4])
```

```
        print("Fold 1",array)

process.mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,
dsize,costinit,costcheckpt,burnin1,burnin2)
        """
        seed(13031952)
        array = np.array([15,19,14,16,1,6,7,21,19,22,21,25,5,13,26,12,3,11,27,1,28,10,23,16,24,18,9,4])
        print("Fold 2",array)

process.mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,
dsize,costinit,costcheckpt,burnin1,burnin2)

        """
        seed(13031953)
        array = np.array([17,19,22,21,15,20,14,8,25,5,13,26,12,3,11,27,1,28,7,10,23,6,16,24,18,9,2,4])
        print("Fold 2",array)

process.mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,
dsize,costinit,costcheckpt,burnin1,burnin2)

        seed(13031953)
        array = np.array([25,5,13,26,15,20,14,8,17,19,22,21,12,3,11,27,1,28,7,10,23,6,16,24,18,9,2,4])
        print("Fold 3",array)

process.mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,
dsize,costinit,costcheckpt,burnin1,burnin2)
        seed(13031953)
        array = np.array([12,3,11,27,15,20,14,8,17,19,22,21,25,5,13,26,1,28,7,10,23,6,16,24,18,9,2,4])
        print("Fold 4",array)

process.mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,
dsize,costinit,costcheckpt,burnin1,burnin2)
        seed(13031953)
        array = np.array([1,28,7,10,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,23,6,16,24,18,9,2,4])
        print("Fold 5",array)

process.mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,
dsize,costinit,costcheckpt,burnin1,burnin2)
        seed(13031953)
        array = np.array([23,6,16,24,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,1,28,7,10,18,9,2,4])
        print("Fold 6",array)

process.mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,
dsize,costinit,costcheckpt,burnin1,burnin2)
        seed(13031953)
        array = np.array([18,9,2,4,15,20,14,8,17,19,22,21,25,5,13,26,12,3,11,27,1,28,7,10,23,6,16,24])
        print("Fold 7",array)

process.mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,
```

```
dsize,costinit,costcheckpt,burnin1,burnin2)

        """

    """Kevin Raina"""
    def
mainrandom(array,nval,ntrain,ntest,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsize,co
stinit,costcheckpt,burnin1,burnin2):
        for v in range(1,2):
            print("Validation: Fold #",v)

            if nval != 0:
                data_F_T = process.dataorganize(array,ntest,nval,ntrain,"Flair","Training")
                data_D_T = process.dataorganize(array,ntest,nval,ntrain,"DWI","Training")
                data_T1_T = process.dataorganize(array,ntest,nval,ntrain,"T1","Training")
                data_T2_T = process.dataorganize(array,ntest,nval,ntrain,"T2","Training")
                labels_T = process.labelorganize(array,ntest,nval,ntrain,"Training")

                data_F_V = process.dataorganize(array,ntest,nval,ntrain,"Flair","Validation")
                data_D_V = process.dataorganize(array,ntest,nval,ntrain,"DWI","Validation")
                data_T1_V = process.dataorganize(array,ntest,nval,ntrain,"T1","Validation")
                data_T2_V = process.dataorganize(array,ntest,nval,ntrain,"T2","Validation")
                labels_V = process.labelorganize(array,ntest,nval,ntrain,"Validation")




            mods = 4



            tf.reset_default_graph()

            x = tf.placeholder(tf.float32)
            y = tf.placeholder(tf.float32)
            probl1 = tf.placeholder_with_default(1.0,shape=())
            probl2 = tf.placeholder_with_default(1.0,shape=())
            #probl2 had dropout 0.5 and probl1 has no dropout
            layer1 = process.create_new_conv_layer(x,mods,8,[3,3,3],[2,2,2],2,'layer1',patch-
3,True,True,probl1)
            layer2 = process.create_new_conv_layer(layer1,8,16,[3,3,3],[2,2,2],2,'layer2',patch-
4,True,True,probl2)
            layer3 = process.create_new_conv_layer(layer2,16,32,[3,3,3],[2,2,2],2,'layer3',patch-
6,True,True,probl2)

            layerf = process.create_new_conv_layer(layer3,32,1,[patch-9,patch-9,patch-9],
[2,2,2],2,'layerf',patch-12,False,True,probl1)
            logits = tf.squeeze(layerf)
            link = tf.exp(logits)
```

```
        with tf.get_default_graph().gradient_override_map({"Floor": "Identity"}):
            y_ = tf.floor(link)


        reg1 = tf.reduce_sum(tf.abs(tf.trainable_variables()[0])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[2])) + tf.reduce_sum(tf.abs(tf.trainable_variables()[4])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[6]))# + tf.reduce_sum(tf.abs(tf.trainable_variables()[8])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[10])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[12]))
+ tf.reduce_sum(tf.abs(tf.trainable_variables()[14])) +tf.reduce_sum(tf.abs(tf.trainable_variables()
[16])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[18]))
+tf.reduce_sum(tf.abs(tf.trainable_variables()[20]))

        reg2 = tf.nn.l2_loss(tf.trainable_variables()[0]) + tf.nn.l2_loss(tf.trainable_variables()[2])
+tf.nn.l2_loss(tf.trainable_variables()[4]) + tf.nn.l2_loss(tf.trainable_variables()[6]) #+
tf.nn.l2_loss(tf.trainable_variables()[8]) + tf.nn.l2_loss(tf.trainable_variables()[10])
+tf.nn.l2_loss(tf.trainable_variables()[12]) + tf.nn.l2_loss(tf.trainable_variables()[14])
+tf.nn.l2_loss(tf.trainable_variables()[16]) + tf.nn.l2_loss(tf.trainable_variables()[18])
+tf.nn.l2_loss(tf.trainable_variables()[20])



        average_log_loss = tf.reduce_mean(tf.nn.log_poisson_loss(targets = y,log_input =
logits,compute_full_loss=True))
        loss = average_log_loss + lambda1*reg1 + lambda2*reg2
        global_step = tf.Variable(0,trainable = False)

        optimizer = tf.train.AdamOptimizer(learning_rate=g1,epsilon=1.0).minimize(loss,
global_step = global_step)




        init_op = tf.global_variables_initializer()
        init_l = tf.local_variables_initializer()
        saver = tf.train.Saver()
        with tf.Session() as sess:
            sess.run(init_op)
            sess.run(init_l)
            #saver.restore(sess, './saved_variable')

            print("Training")
            total_batch = int(n)

            for epoch in range(epochs):
                avg_cost = 0
                prev_avg_cost = 0
                for i in range(1,total_batch + 1):
                    train_samples = process.get_training_samples(data_F_T,data_D_T, data_T1_T,
```

```
data_T2_T,labels_T,ntrain,b, patch)
                    train_array = train_samples[0]

                    train_array = np.transpose(train_array,(1,2,3,4,0))
                    arrlab = train_samples[1]
                    arrlab = np.squeeze(arrlab)
                    dict = {x : train_array , y : arrlab, probl1:1,probl2:0.5}

                    temploss = sess.run(loss, feed_dict = dict)
                    _,c = sess.run([optimizer,loss],feed_dict=dict)
                    avg_cost += temploss/costcheckpt

                    if i < costinit or (i/costcheckpt).is_integer():

                       if prev_avg_cost < avg_cost and i > 2*costcheckpt -1:
                       #if (i/checkpoint).is_integer():
                          save_path = saver.save(sess,'./saved_variable')
                          #print('Model saved in {}'.format(save_path))

                          #print(i,avg_cost,"Model Converged")

                          if nval != 0:

                             boundarray = np.array([0,250,500,750,1000,1500,2000])
                             for index_2 in range(0,1):



                                #Experiment 1
                                comp_accuracy = 0
                                N = 0
                                for m in range(1,10001):
                                   train_samples = process.get_positive_samples(data_F_V,data_D_V,
data_T1_V, data_T2_V,labels_V,nval, 2, patch)
                                   train_array = train_samples[0]
                                   train_array = np.transpose(train_array,(1,2,3,4,0))
                                   arrlab = train_samples[1]
                                   arrlab = np.squeeze(arrlab)
                                   dict = {x : train_array , y : arrlab}
                                   predictions = sess.run(y_,feed_dict=dict)
                                   if (abs(arrlab[0]-arrlab[1])>boundarray[0]):
                                      N += 1
                                      if ((predictions[0] > predictions[1] and arrlab[0] > arrlab[1]) or
(predictions[0] < predictions[1] and arrlab[0] < arrlab[1])):
                                         comp_accuracy += 1
                                         #print(predictions[0],arrlab[0],predictions[1],arrlab[1])
                                      else:
                                         comp_accuracy += 0
                                print(comp_accuracy/N)
```

```
                print(N)



                #Experiment 2

                MAE = 0
                MRE = 0
                RATIO = 0
                for m in range(1,1001):
                    train_samples = process.get_training_samples(data_F_V,data_D_V,
data_T1_V, data_T2_V,labels_V,nval,10, patch)
                    train_array = train_samples[0]
                    train_array = np.transpose(train_array,(1,2,3,4,0))
                    arrlab = train_samples[1]
                    arrlab = np.squeeze(arrlab)
                    #arrindex = train_samples[2]
                    dict = {x : train_array , y : arrlab}
                    """
                    if 1 <= m <= 20:
                        print("Sample:",m,"True Count:",arrlab,"Mean
Rate:",sess.run(y_,feed_dict=dict))
                    """

                    predictions = sess.run(y_,feed_dict=dict)

                    for s in range(0,len(predictions)):
                        if predictions[s] > patch*patch*patch:
                            predictions[s] = patch*patch*patch
                        RE = abs(predictions[s]-arrlab[s])/arrlab[s]
                        MRE += RE/(1000*10)
                        MAE += abs(predictions[s]-arrlab[s])/(1000*10)
                        RATIO += (predictions[s]/arrlab[s])/(1000*10)
                print("MRE:",MRE)
                print("MAE:",MAE)
                print("AVG RATIO:",RATIO)

                #Experiment 3
                fhat = np.zeros(100)
                truth = np.zeros(100)

                #Experiment 3
                #sizearray = np.array([5,13,15,17,19,21,25])
                #feature = np.empty([len(sizearray),25,25,25,4])
                #for alph in range(0,7):
                #    inndim = int(sizearray[alph])
                #    inner = np.full([inndim,inndim,inndim],2.5)
                #    padwith = 12-int(inndim/2)
```

```
                            #    mrseq = np.pad(inner,((padwith,padwith),(padwith,padwith),
(padwith,padwith)),'constant',constant_values=((1.5,1.5),(1.5,1.5),(1.5,1.5)))
                            #    obs = np.transpose(np.array([mrseq,mrseq,mrseq,mrseq]),(1,2,3,0))
                            #    obs = np.array([obs])
                            #    print(obs.shape)
                            #    if alph == 0:
                            #        temp = obs
                            #    if alph >= 1:
                            #        feature = np.append(temp,obs,axis=0)
                            #        temp = feature
                            #dict = {x: feature}
                            #predictions = sess.run(y_,feed_dict=dict)
                            #print("Estimation:",predictions)
                        #print(i,avg_cost)
                        prev_avg_cost = avg_cost
                        if (i/costcheckpt).is_integer():
                            avg_cost = 0
                    carr = np.array([c])
                    if np.isnan(carr) == True:
                        print ("Cost is Infinite")



            sess.close()







print("PRcount3D")



process.crossval(barray=np.array([1]), j = 15, nval = 8,ntrain = 20, ntest = 0, mu = 0.6, d = 0.1, g1 =
0.0001, g2= 0.001, b = 10,
            n = 15000, lambda1 = 10e-8, lambda2 = 10e-6, epochs = 1, patch = 25, checkpoint = 45000,
dsize = 50000,
            costcheckpt = 1000, costinit = 100, burnin1 = 15000, burnin2 = 100)
```

Appendix D

Wider2DSegIIC Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 21 13:47:09 2019

@author: Pandit Kevin Raina
"""
import sys
#import scipy
#import cv2
import nibabel as nib
import os
#import math as mat
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import numpy as np
from random import randint
from random import seed
#from pylab import *

class process:



    def load_data(data_path , modality, range1,range2):
        """
        Parameters:
        data_path: path to the folder "Training" with brain images
        """
        data = []
        for brain_id in range(range1, range2+1):
            if brain_id < 10:
                brain_id = '0' + str(brain_id)
            folder = os.path.join(data_path, str(brain_id))
            for file in os.listdir(folder):
                if modality in file:
                    filefile = file + '/' + file +'.nii'
                    file_name = os.path.join(folder, filefile)
                    image = nib.load(file_name)
                    np_image = image.get_data()
                    data.append(np_image)

        normalized1 = process.standard_normalization(data)

        return normalized1



    def standard_normalization(data):
        """
```

```
        Here the input is a list of brains and the function normalize each of this brains
        to have a mean 0 and standard deviation 1.
        """
        normalized = []
        for image in data:
            m = np.mean(image)
            image = image - m
            st_dev = np.std(image)
            image = image/st_dev
            normalized.append(image)
        return normalized



    def get_uniform_samples_dense_batch(data_F,data_D, data_T1,data_T2, size , dim , dim2,
archsize, shift_size):
        """
        Parameters:
        data: list of 3D numpy arrays representing an image of the brain
        labels: list of 3D numpy array representing a voxelwise segmentation of the brain in data
        from_brains:
        size: number of extracted patches
        dim: patch has a shape (dim, dim)

        Returns: tuple, namely, (array of patches, array of labels of the central voxels in the patches).

        There are equal number of samples labeled 1 and 0.
        """
        batch_data_F = []
        batch_data_D = []
        batch_data_T1 = []
        batch_data_T2 = []
        batch_data_F_shift = []
        batch_data_D_shift = []
        batch_data_T1_shift = []
        batch_data_T2_shift = []


        batch_data_F2 = []
        batch_data_D2 = []
        batch_data_T12 = []
        batch_data_T22 = []
        batch_data_F_shift2 = []
        batch_data_D_shift2 = []
        batch_data_T1_shift2 = []
        batch_data_T2_shift2 = []


        while len(batch_data_F) < size:
```

```
            index = 0
            found = 0
            while found == 0:
                #x = randint(int(dim2/2)+shift_size,data_F[index].shape[0] - int(dim2/2) - shift_size-1)
                #y = randint(int(dim2/2)+shift_size,data_F[index].shape[1] - int(dim2/2) - shift_size-1)
                #z = randint(0,data_F[index].shape[2] - 1)
                x = randint(43,187)
                y = randint(32,198)
                z = randint(5,147)
                if data_F[index][x][y][z] != 0:
                    t = randint(-shift_size,shift_size)
                    #print(x,y,z,t)
                    """
                    and labels[index][x][y][z] == 0:
                    """
                    found = 1
                    patch_F = data_F[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    """
                    when you subet an array the upper dimension is + 1 above the true size
                    """
                    batch_data_F.append(patch_F)
                    patch_D = data_D[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) + 1,
z]
                    batch_data_D.append(patch_D)
                    patch_T1 = data_T1[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                    batch_data_T1.append(patch_T1)
                    patch_T2 = data_T2[index][x-int(dim/2):x+int(dim/2) + 1, y-int(dim/2):y+int(dim/2) +
1, z]
                    batch_data_T2.append(patch_T2)

                    patch_F = data_F[index][x+t-int(dim/2):x+t+int(dim/2) + 1, y+t-
int(dim/2):y+t+int(dim/2) + 1, z]
                    batch_data_F_shift.append(patch_F)
                    patch_D = data_D[index][x+t-int(dim/2):x+t+int(dim/2) + 1, y+t-
int(dim/2):y+t+int(dim/2) + 1, z]
                    batch_data_D_shift.append(patch_D)
                    patch_T1 = data_T1[index][x+t-int(dim/2):x+t+int(dim/2) + 1, y+t-
int(dim/2):y+t+int(dim/2) + 1, z]
                    batch_data_T1_shift.append(patch_T1)
                    patch_T2 = data_T2[index][x+t-int(dim/2):x+t+int(dim/2) + 1, y+t-
int(dim/2):y+t+int(dim/2) + 1, z]
                    batch_data_T2_shift.append(patch_T2)



                    patch_F = data_F[index][x-int(dim2/2):x+int(dim2/2) + 1, y-int(dim2/2):y+int(dim2/2)
+ 1, z]
```

```
                batch_data_F2.append(patch_F)
                patch_D = data_D[index][x-int(dim2/2):x+int(dim2/2) + 1, y-int(dim2/2):y+int(dim2/2)
+ 1, z]
                batch_data_D2.append(patch_D)
                patch_T1 = data_T1[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T12.append(patch_T1)
                patch_T2 = data_T1[index][x-int(dim2/2):x+int(dim2/2) + 1, y-
int(dim2/2):y+int(dim2/2) + 1, z]
                batch_data_T22.append(patch_T2)

                patch_F = data_F[index][x+t-int(dim2/2):x+t+int(dim2/2) + 1, y+t-
int(dim2/2):y+t+int(dim2/2) + 1, z]
                batch_data_F_shift2.append(patch_F)
                patch_D = data_D[index][x+t-int(dim2/2):x+t+int(dim2/2) + 1, y+t-
int(dim2/2):y+t+int(dim2/2) + 1, z]
                batch_data_D_shift2.append(patch_D)
                patch_T1 = data_T1[index][x+t-int(dim2/2):x+t+int(dim2/2) + 1, y+t-
int(dim2/2):y+t+int(dim2/2) + 1, z]
                batch_data_T1_shift2.append(patch_T1)
                patch_T2 = data_T1[index][x+t-int(dim2/2):x+t+int(dim2/2) + 1, y+t-
int(dim2/2):y+t+int(dim2/2) + 1, z]
                batch_data_T2_shift2.append(patch_T2)




        batch_data_F = np.array(batch_data_F,dtype=object)
        batch_data_D = np.array(batch_data_D,dtype=object)
        batch_data_T1 = np.array(batch_data_T1,dtype=object)
        batch_data_T2 = np.array(batch_data_T2,dtype=object)
        batch_data_F_shift = np.array(batch_data_F_shift,dtype=object)
        batch_data_D_shift = np.array(batch_data_D_shift,dtype=object)
        batch_data_T1_shift = np.array(batch_data_T1_shift,dtype=object)
        batch_data_T2_shift = np.array(batch_data_T2_shift,dtype=object)


        batch_data_F2 = np.array(batch_data_F2,dtype=object)
        batch_data_D2 = np.array(batch_data_D2,dtype=object)
        batch_data_T12 = np.array(batch_data_T12,dtype=object)
        batch_data_T22 = np.array(batch_data_T22,dtype=object)
        batch_data_F_shift2 = np.array(batch_data_F_shift2,dtype=object)
        batch_data_D_shift2 = np.array(batch_data_D_shift2,dtype=object)
        batch_data_T1_shift2 = np.array(batch_data_T1_shift2,dtype=object)
        batch_data_T2_shift2 = np.array(batch_data_T2_shift2,dtype=object)
```

```
        # batch_labels = np.array(process.one_hot_encode(batch_labels))
        # batch_labels = np.array(batch_labels)

        #   randomize = np.arange(size)
        #   np.random.shuffle(randomize)

        samples = np.array([batch_data_F,batch_data_D,batch_data_T1,batch_data_T2])
        samples2 = np.array([batch_data_F2,batch_data_D2,batch_data_T12,batch_data_T22])
        samples3 =
np.array([batch_data_F_shift,batch_data_D_shift,batch_data_T1_shift,batch_data_T2_shift])
        samples4 =
np.array([batch_data_F_shift2,batch_data_D_shift2,batch_data_T1_shift2,batch_data_T2_shift2])
        #print(samples.shape,samples2.shape,samples3.shape,samples4.shape)
        return samples,samples2,samples3,samples4

    def create_new_conv_layer(input_data, input_data2, num_input_channels, num_filters,
filter_shape, pool_shape, K, name, patch_size, addpool, keep_prob):
        # setup the filter input shape for tf.nn.conv_2d
        conv_filt_shape = [filter_shape[0], filter_shape[1], num_input_channels,num_filters]

        # initialise weights and bias for the filter
        weights = tf.Variable(tf.random_normal(conv_filt_shape, mean=0.0,stddev=(2/
(num_input_channels*filter_shape[0]*filter_shape[1]))**(1/2), seed = 1967),name=name+'_W')
        bias = tf.Variable(tf.zeros([num_filters], name=name+'_b'))

        # setup the convolutional layer operation
        if addpool == False:
            out_layer = tf.nn.conv2d(input_data, weights, [1, 1, 1, 1],padding = 'VALID')

            # add the bias
            out_layer += bias

            # apply a ReLU non-linear activation
            out_layer = tf.nn.leaky_relu(out_layer)
            out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1992)

            out_layer2 = tf.nn.conv2d(input_data2, weights, [1, 1, 1, 1],padding = 'VALID')

            # add the bias
            out_layer2 += bias

            # apply a ReLU non-linear activation
            out_layer2 = tf.nn.leaky_relu(out_layer2)
            out_layer2 = tf.nn.dropout(out_layer,keep_prob,seed=1992)
            return out_layer,out_layer2
        if addpool == True:
            out_layer = tf.nn.conv2d(input_data, weights, [1, 1, 1, 1],padding = 'VALID')
```

```
            # add the bias
            out_layer += bias

            # apply a ReLU non-linear activation
            out_layer = tf.nn.leaky_relu(out_layer)
            # now perform max pooling


            out_layer = tf.nn.dropout(out_layer,keep_prob,seed=1995)
            out_layer2 = tf.nn.conv2d(input_data2, weights, [1, 1, 1, 1],padding = 'VALID')

            # add the bias
            out_layer2 += bias

            # apply a ReLU non-linear activation
            out_layer2 = tf.nn.leaky_relu(out_layer2)
            out_layer2 = tf.nn.dropout(out_layer,keep_prob,seed=1992)
            return out_layer,out_layer2



        """Kevin Raina"""
        def
mainframe(subjid,directory,mods,array,mu,d,g1,g2,b,n,lambda1,lambda2,epochs,patch,checkpoint,dsiz
e,costinit,costcheckpt,burnin1,burnin2,tsize,tsize2,shift_size,classes):
            for v in range(1,2):

                print("Wider 2D Segmentation IIC protocol")

                print("Loading Image:")

                data_F_T = process.load_data(directory,"Flair",subjid,subjid)
                data_T1_T = process.load_data(directory,"T1",subjid,subjid)
                data_T2_T = process.load_data(directory,"T2",subjid,subjid)
                data_D_T = process.load_data(directory,"DWI",subjid,subjid)

                data_F_V = process.load_data(directory,"Flair",subjid,subjid)
                data_T1_V = process.load_data(directory,"T1",subjid,subjid)
                data_T2_V = process.load_data(directory,"T2",subjid,subjid)
                data_D_V = process.load_data(directory,"DWI",subjid,subjid)




                tf.reset_default_graph()

                x = tf.placeholder(tf.float32)
                x2 = tf.placeholder(tf.float32)
                z = tf.placeholder(tf.float32)
```

```
        z2 = tf.placeholder(tf.float32)
        #y = tf.placeholder(tf.float32)

        probl1 = tf.placeholder_with_default(1.0,shape=())
        probl2 = tf.placeholder_with_default(1.0,shape=())
        x2down = tf.image.resize_nearest_neighbor(x2,tf.cast(tf.stack([tf.shape(x2)[1]/3,tf.shape(x2)
[2]/3]),tf.int32),True)
        z2down = tf.image.resize_nearest_neighbor(z2,tf.cast(tf.stack([tf.shape(z2)[1]/3,tf.shape(z2)
[2]/3]),tf.int32),True)

        """patch-6,patch-11,patch-12,1,[patch-12,patch-12], 24,21,21,1,[21,21]"""
        """4,64,7,4,64,64,3,2,4,160,13,13,1,224,2,1,1 """
        layer11,layer11p = process.create_new_conv_layer(x,z,mods,60,[3,3],[1,1],2,'layer11',patch-
2,True,probl1)
        layer12,layer12p = process.create_new_conv_layer(layer11,layer11p,60,60,[3,3],
[2,2],2,'layer12',patch-11,True,probl1)
        layer13,layer13p = process.create_new_conv_layer(layer12,layer12p,60,80,[3,3],
[1,1],2,'layer13',patch-12,False,probl1)
        layer14,layer14p = process.create_new_conv_layer(layer13,layer13p,80,80,[3,3],
[1,1],2,'layer14',patch-12,False,probl1)
        layer15,layer15p = process.create_new_conv_layer(layer14,layer14p,80,80,[3,3],
[1,1],2,'layer15',patch-12,False,probl1)
        layer16,layer16p = process.create_new_conv_layer(layer15,layer15p,80,80,[3,3],
[1,1],2,'layer16',patch-12,False,probl1)
        layer17,layer17p = process.create_new_conv_layer(layer16,layer16p,80,100,[3,3],
[1,1],2,'layer17',patch-12,False,probl1)
        layer18,layer18p = process.create_new_conv_layer(layer17,layer17p,100,100,[3,3],
[1,1],2,'layer18',patch-12,False,probl1)

        layer21,layer21p = process.create_new_conv_layer(x2down,z2down,mods,60,[3,3],
[1,1],2,'layer21',patch-2,True,probl1)
        layer22,layer22p = process.create_new_conv_layer(layer21,layer21p,60,60,[3,3],
[2,2],2,'layer22',patch-11,True,probl1)
        layer23,layer23p = process.create_new_conv_layer(layer22,layer22p,60,80,[3,3],
[1,1],2,'layer23',patch-12,False,probl1)
        layer24,layer24p = process.create_new_conv_layer(layer23,layer23p,80,80,[3,3],
[1,1],2,'layer24',patch-12,False,probl1)
        layer25,layer25p = process.create_new_conv_layer(layer24,layer24p,80,80,[3,3],
[1,1],2,'layer25',patch-12,False,probl1)
        layer26,layer26p = process.create_new_conv_layer(layer25,layer25p,80,80,[3,3],
[1,1],2,'layer26',patch-12,False,probl1)
        layer27,layer27p = process.create_new_conv_layer(layer26,layer26p,80,100,[3,3],
[1,1],2,'layer27',patch-12,False,probl1)
        layer28,layer28p = process.create_new_conv_layer(layer27,layer27p,100,100,[3,3],
[1,1],2,'layer28',patch-12,False,probl1)
        layer28up = tf.image.resize_nearest_neighbor(layer28,[tf.shape(layer28)
[1]*3,tf.shape(layer28)[2]*3],True)
        layer28upp = tf.image.resize_nearest_neighbor(layer28p,[tf.shape(layer28p)
[1]*3,tf.shape(layer28p)[2]*3],True)
```

```
            conc = tf.concat([layer28up, layer18], axis = 3)
            conc2 = tf.concat([layer28upp,layer18p], axis = 3)
            layerf1,layerf1p = process.create_new_conv_layer(conc,conc2,200,200,[1,1],
[1,1],2,'layerf1',1,False,probl2)
            layerf2,layerf2p = process.create_new_conv_layer(layerf1,layerf1p,200,200,[1,1],
[1,1],2,'layerf2',1,False,probl2)
            layerf,layerfp = process.create_new_conv_layer(layerf2,layerf2p,200,classes,[1,1],
[1,1],1,'layerf',1,False,probl1)

            logits = tf.squeeze(layerf)
            logitsp = tf.squeeze(layerfp)
            """layer 4: 1 by 198 by 198 by 2, logits: 198 by 198 by 2"""
            y_ = tf.nn.softmax(logits)
            y_p = tf.nn.softmax(logitsp)
            """ 154 by 230 by 230 by 2 m,idx,count = tf.unique_with_counts(tf.argmax(y,3))"""

            k = tf.shape(y_)[3]
            phi = tf.reshape(y_,[b*(tsize-16)**2,classes])
            phi_p = tf.reshape(y_p,[b*(tsize-16)**2,classes])
            phi = tf.transpose(phi,(1,0))
            p = tf.matmul(phi,phi_p)
            """
            p = tf.reduce_sum(tf.expand_dims(y_, 2) * tf.expand_dims(y_p, 1), 0)
            """
            p = (p + tf.transpose(p)) / 2

            p /= tf.reduce_sum(p)

            pi = tf.broadcast_to(tf.reshape(tf.reduce_sum(p, axis=1), [k, 1]), [k, k])
            pj = tf.broadcast_to(tf.reshape(tf.reduce_sum(p, axis=0), [1, k]), [k, k])
            EPS=sys.float_info.epsilon
            p = tf.clip_by_value(p, EPS, 1e9)
            pi = tf.clip_by_value(pi, EPS, 1e9)
            pj = tf.clip_by_value(pj, EPS, 1e9)
            """
            phi = tf.reshape(y_,[b*(tsize-16)**2,classes])
            phi_p = tf.reshape(y_p,[b*(tsize-16)**2,classes])
            phi = tf.transpose(phi,[1,0])
            p = tf.matmul(phi,phi_p)
            """
            #reg1 = tf.reduce_sum(tf.abs(tf.trainable_variables()[0])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[2])) + tf.reduce_sum(tf.abs(tf.trainable_variables()[4])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[6])) + tf.reduce_sum(tf.abs(tf.trainable_variables()[8])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[10])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[12]))
+ tf.reduce_sum(tf.abs(tf.trainable_variables()[14])) +tf.reduce_sum(tf.abs(tf.trainable_variables()
[16])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[18]))
+tf.reduce_sum(tf.abs(tf.trainable_variables()[20])) + tf.reduce_sum(tf.abs(tf.trainable_variables()
[22])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[24])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[26])) +tf.reduce_sum(tf.abs(tf.trainable_variables()[28]))
```

```
+ tf.reduce_sum(tf.abs(tf.trainable_variables()[30])) +tf.reduce_sum(tf.abs(tf.trainable_variables()
[32])) + tf.reduce_sum(tf.abs(tf.trainable_variables()[34])) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[36]))

        #reg2 = tf.nn.l2_loss(tf.trainable_variables()[0]) + tf.nn.l2_loss(tf.trainable_variables()[2])
+tf.nn.l2_loss(tf.trainable_variables()[4]) + tf.nn.l2_loss(tf.trainable_variables()[6]) +
tf.nn.l2_loss(tf.trainable_variables()[8]) + tf.nn.l2_loss(tf.trainable_variables()[10])
+tf.nn.l2_loss(tf.trainable_variables()[12]) + tf.nn.l2_loss(tf.trainable_variables()[14])
+tf.nn.l2_loss(tf.trainable_variables()[16]) + tf.nn.l2_loss(tf.trainable_variables()[18])
+tf.nn.l2_loss(tf.trainable_variables()[20]) + tf.nn.l2_loss(tf.trainable_variables()[22])
+tf.nn.l2_loss(tf.trainable_variables()[24]) + tf.nn.l2_loss(tf.trainable_variables()[26])
+tf.nn.l2_loss(tf.trainable_variables()[28]) + tf.nn.l2_loss(tf.trainable_variables()[30]) +
tf.nn.l2_loss(tf.trainable_variables()[32]) + tf.nn.l2_loss(tf.trainable_variables()[34]) +
tf.reduce_sum(tf.abs(tf.trainable_variables()[36]))


        """
        cross_entropy= tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits =
logits,labels = y))
        """
        loss = tf.reduce_sum( - p * (tf.math.log(p) - 1.0 * tf.math.log(pi) - 1.0 * tf.math.log(pj) ))
        global_step = tf.Variable(0,trainable = False)

        """
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=g1).minimize(loss)


        learningprime = tf.train.piecewise_constant(global_step, boundaries =
[25000,37000,49000,60000,71000,75000],values = [g1,g1/2,g1/4,g1/8,g1/16,g1/32,g1/64])

        learningprime = tf.train.exponential_decay(learning_rate = g1,global_step =
global_step,decay_steps = dsize,decay_rate = d)

        optimizer=
tf.train.RMSPropOptimizer(learning_rate=learningprime,momentum=mu,epsilon=1e-4).minimize(loss,
global_step=global_step)
        """
        optimizer = tf.train.AdamOptimizer(learning_rate=g1,epsilon=1.0).minimize(loss,var_list =
tf.trainable_variables()[0:36], global_step = global_step)


        paddedy_ = tf.pad(tf.argmax(y_,3),paddings = tf.constant([[0,0],[int(51/2),int(51/2)],
[int(51/2),int(51/2)]]),mode = "CONSTANT")
        #tp = tf.metrics.true_positives(y,paddedy_,weights=None)[1]
        #fp = tf.metrics.false_positives(y,paddedy_,weights=None)[1]
        #fn = tf.metrics.false_negatives(y,paddedy_,weights=None)[1]
```

```
init_op = tf.global_variables_initializer()
init_l = tf.local_variables_initializer()
with tf.Session() as sess:
   sess.run(init_op)
   sess.run(init_l)
   print("Training")
   total_batch = int(n)
   """
   switch = 0
   """
   for epoch in range(epochs):
      avg_cost = 0
      for i in range(1,total_batch + 1):

         if (i/checkpoint).is_integer() and i > burnin1:
            print("Overfit Checkpoint:",i/checkpoint)
            print("Computing Segmentation")
            data_V = np.array([data_F_V[0],data_D_V[0],data_T1_V[0],data_T2_V[0]])
            data_V_prime1 = np.transpose(data_V,(3,1,2,0))

            for s in range(0,2):

               dat2 = data_V_prime1[77*s:77*s + 77,1:229,1:229,:]
               dat1 = data_V_prime1[77*s:77*s+77,17:213,17:213,:]

               if s==0:
                  pred1 = sess.run(paddedy_, feed_dict = {x:dat1, x2:dat2})
                  pred1 = np.asarray(pred1)
                  pred1 = np.transpose(pred1,(1,2,0))
               if s==1:
                  pred2 = sess.run(paddedy_, feed_dict = {x:dat1, x2:dat2})
                  pred2 = np.asarray(pred2)
                  pred2 = np.transpose(pred2,(1,2,0))

            predf = np.concatenate((pred1,pred2),axis=2)

            predf1 = nib.Nifti1Image(predf,np.eye(4))

            nib.save(predf1,'IIC_Segmentation_'+str(subjid)+'.nii')



         train_samples = process.get_uniform_samples_dense_batch(data_F_T,data_D_T,
data_T1_T, data_T2_T,b,tsize,tsize2,patch,shift_size)

         train_array = train_samples[0]
         train_array2 = train_samples[1]
         train_array3 = train_samples[2]
```

```
                    train_array4 = train_samples[3]
                    train_array = np.transpose(train_array,(1,2,3,0))
                    train_array2 = np.transpose(train_array2,(1,2,3,0))
                    train_array3 = np.transpose(train_array3,(1,2,3,0))
                    train_array4 = np.transpose(train_array4,(1,2,3,0))
                    """
                    arrlab = train_samples[2]
                    """
                    dict = {x : train_array ,x2: train_array2, z: train_array3, z2: train_array4,
probl1:1,probl2:0.5}
                    """
                    dict2 = {x: train_array3 ,x2:train_array4, probl1:1,probl2:0.5}

                    phi_x = sess.run(y_,feed_dict = dict)
                    phi_x_prime = sess.run(y_,feed_dict = dict2)
                    phi_x = np.asarray(phi_x)
                    phi_x_prime = np.asarray(phi_x_prime)

                    phi_x = phi_x.reshape(-1,phi_x.shape[-1])
                    phi_x_prime = phi_x_prime.reshape(-1,phi_x_prime.shape[-1])
                    phi_x = np.transpose(phi_x,(1,0))
                    P = (1/(b*(tsize-16)**2))*np.matmul(phi_x,phi_x_prime)
                    P = (1/2)*(P + np.transpose(P,(1,0)))
                    P_C = np.sum(P,axis=0)
                    P_C_prime = np.sum(P,axis=1)
                    IICarray = 0
                    for C in range(0,classes):
                        for C_prime in range(0,classes):
                            IICarray += P[C,C_prime]*np.log(P[C,C_prime]/
(P_C[C]*P_C_prime[C_prime]))

                    phi = sess.run(y_,feed_dict=dict)
                    phi_p = sess.run(y_p,feed_dict=dict)
                    print(phi.shape,phi_p.shape)
                    """

                    temploss = sess.run(loss, feed_dict = dict)
                    _,c = sess.run([optimizer,loss],feed_dict=dict)
                    avg_cost += temploss/costcheckpt

                    if i < costinit or (i/costcheckpt).is_integer():
                        print(i,avg_cost)
                        if (i/costcheckpt).is_integer():
                            avg_cost = 0
                    carr = np.array([c])
                    if np.isnan(carr) == True:
                        print ("Cost is Infinite")
```

```
            sess.close()



print("Wider 2-D Segmentation IIC")
process.mainframe(subjid = 1, directory = "/data/SISS2015/Training", mods = 4, array=np.array([1]),
mu = 0.6, d = 0.1, g1 = 0.0001, g2= 0.001, b = 10, n = 20000,
            lambda1 = 0.00000001, lambda2 = 0.000001, epochs = 1, patch = 17, checkpoint = 10000,
dsize = 50000,
            costcheckpt = 1000, costinit = 100, burnin1 = 15000, burnin2 = 100,tsize = 25,tsize2 =
57,shift_size = 2, classes = 10)
"""
seed(13031953)
process.crossval(linear = False,symmtrainingmodality = "Flair",symmtestingmodality = "Flair",
barray=np.array([1]), j = 15, nval = 4,
            ntrain = 24, ntest = 0, mu = 0.6, d = 0.1, g1 = 0.001, g2= 0.001, b = 12, n = 20,
            lambda1 = 0.00000001, lambda2 = 0.000001, epochs = 1, patch = 17, checkpoint = 15, dsize
= 50000,
            costcheckpt = 1000, costinit = 100, burnin1 = 10, burnin2 = 100,tsize = 43,tsize2 = 75
,pi=0.02,
            symmetry = False, joint = True, marginal = "T2")
"""
```

Appendix E

ISD Python Code

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

import nibabel as nib
import os
import numpy as np
from random import seed



def multinomial_rvs(n, p):
    """
    Sample from the multinomial distribution with multiple p vectors.

    * n must be a scalar.
    * p must an n-dimensional numpy array, n >= 1.  The last axis of p
      holds the sequence of probabilities for a multinomial distribution.

    The return value has the same shape as p.
    """
    count = np.full(p.shape[:-1], n)
    out = np.zeros(p.shape, dtype=int)
    ps = p.cumsum(axis=-1)
    # Conditional probabilities
    with np.errstate(divide='ignore', invalid='ignore'):
        condp = p / ps
    condp[np.isnan(condp)] = 0.0
    for i in range(p.shape[-1]-1, 0, -1):
        binsample = np.random.binomial(count, condp[..., i])
        out[..., i] = binsample
        count -= binsample
    out[..., 0] = count
    return out


#Brain Dice Confints
m = 27

seed(13031953)
path = 'C:/Users/krain/Dropbox/ISDInference/Probabilities/'
filename = os.path.join(path,'Probabilities' + str(m) +'.nii')
image = nib.load(filename)
prob_raw = image.get_fdata()
prob = np.pad(array = prob_raw,pad_width = ((25,25),(25,25),(0,0),(0,0)),mode='constant')
print("Brain id", m)
```

```
for i in range(1,6):
    samplei = multinomial_rvs(1,prob)
    samplej = multinomial_rvs(1,prob)

    sampi = np.argmax(samplei,axis=3)
    sampj = np.argmax(samplej,axis=3)

    dice = (np.sum(sampj[sampi==1])*2.0) / (np.sum(sampj) + np.sum(sampi))
    tp = (np.sum(sampi[sampj==1]==1))
    fp = (np.sum(sampj[sampi==1]==0))
    fn = (np.sum(sampj[sampi==0]==1))
    tn = (np.sum(sampj[sampi==0]==0))
    print (dice)
```

Bibliography

[1] Brain abscess. https://www.nhs.uk/conditions/brain-abscess/. Accessed January 19,2020.

[2] Brain lesion causes, symptoms, types and treatment. https://www.emedicinehealth.com/brain_lesions_lesions_on_the_brain/article_em.htm/. Accessed January 19,2020.

[3] Brain tumor types and symptoms. https://braintumor.org/brain-tumor-information/understanding-brain-tumors/tumor-types/. Accessed January 19,2020.

[4] Cancer. https://www.who.int/health-topics/cancer#tab=tab_1. Accessed January 19,2020.

[5] Chapter 28 - mass lesions - neoplasm. https://www.dartmouth.edu/~dons/part_3/chapter_28.html/. Accessed January 19,2020.

[6] Radiopaedia.org. https://radiopaedia.org/play/2310/entry/28396/case/4090/studies/6567. Accessed January 19,2020.

[7] Radiopaedia.org. https://radiopaedia.org/cases/cerebral-abscess-1?lang=us. Accessed January 19,2020.

[8] The top 10 causes of death. https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death. Accessed January 19,2020.

[9] Brain tumor - risk factors. `https://www.cancer.net/cancer-types/brain-tumor/risk-factors`, Mar 2019. Accessed January 19,2020.

[10] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[11] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. Technical report, 2010.

[12] Raj Acharya, Richard Wasserman, Jeffrey Stevens, and Carlos Hinojosa. Biomedical imaging modalities: a tutorial. *Computerized Medical Imaging and Graphics*, 19(1):3–25, 1995.

[13] Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on pattern analysis and machine intelligence*, 16(6):641–647, 1994.

[14] Lisa D Alexander, Sandra E Black, Fuqiang Gao, Gregory Szilagyi, Cynthia J Danells, and William E McIlroy. Correlating lesion size and location to deficits after ischemic stroke: the influence of accounting for altered peri-necrotic tissue and incidental silent infarcts. *Behavioral and brain functions*, 6(1):6, 2010.

[15] Neculai Archip, Olivier Clatz, Stephen Whalen, Dan Kacher, Andriy Fedorov, Andriy Kot, Nikos Chrisochoides, Ferenc Jolesz, Alexandra Golby, Peter M Black, et al. Non-rigid alignment of pre-operative mri, fmri, and dt-mri with intra-operative mri for enhanced visualization and navigation in image-guided neurosurgery. *Neuroimage*, 35(2):609–624, 2007.

[16] John Ashburner and Karl J Friston. Unified segmentation. *Neuroimage*, 26(3):839–851, 2005.

[17] American Brain Tumor Association. Brain tumors: A handoobk for the newly diagnosed. https://www.abta.org/wp-content/uploads/2018/03/newly-diagnosed.pdf. Accessed January 19,2020.

[18] American Heart Association. Risk factors for stroke. https://www.stroke.org/-/media/stroke-files/lets-talk-about-risk-factors-for-stroke-ucm_309713.pdf. Accessed January 19,2020.

[19] American Heart Association. Stroke, tia and warning signs. https://www.heart.org/-/media/stroke-files/stroke-resource-center/prevention/letstalkstrokeprevention/tia-and-warning-signs_309532.pdf?la=en&hash=EF6CD99B8420720FC5A2882583707001FC0B22FA. Accessed January 19,2020.

[20] American Stroke Association. Explaining stroke. https://www.stroke.org/-/media/stroke-files/stroke-resource-center/explaining_stroke_brochure_6_25_19.pdf?la=en&hash=086743F6D5F1A1E3C462B2AFB0C8360F16631787. Accessed January 19,2020.

[21] Francesca Bagnato, Vasiliki N Ikonomidou, Peter van Gelderen, Sungyoung Auh, Jailan Hanafy, Fredric K Cantor, Joan Ohayon, Nancy Richert, and Jeff Duyn. Lesions by tissue specific imaging characterize multiple sclerosis patients with more advanced disease. *Multiple Sclerosis Journal*, 17(12):1424–1431, 2011.

[22] Spyridon Bakas, Mauricio Reyes, Andras Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Takeshi Shinohara, Christoph Berger, Sung Min Ha, Martin Rozycki, et al. Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall

survival prediction in the brats challenge. *arXiv preprint arXiv:1811.02629*, 2018.

[23] Stefan Bauer, Lutz-P. Nolte, and Mauricio Reyes. Fully automatic segmentation of brain tumor images using support vector machine classification in combination with hierarchical conditional random field regularization. *Lecture Notes in Computer Science Medical Image Computing and Computer-Assisted Intervention MICCAI 2011*, page 354361, 2011.

[24] Stefan Bauer, Roland Wiest, Lutz-P Nolte, and Mauricio Reyes. A survey of mri-based medical image analysis for brain tumor studies. *Physics in Medicine & Biology*, 58(13):R97, 2013.

[25] Christoph Baur, Benedikt Wiestler, Shadi Albarqouni, and Nassir Navab. Deep autoencoding models for unsupervised anomaly segmentation in brain mr images. In *International MICCAI Brainlesion Workshop*, pages 161–169. Springer, 2018.

[26] Tom Brosch, Lisa YW Tang, Youngjin Yoo, David KB Li, Anthony Traboulsee, and Roger Tam. Deep 3d convolutional encoder networks with shortcuts for multiscale feature integration applied to multiple sclerosis lesion segmentation. *IEEE transactions on medical imaging*, 35(5):1229–1239, 2016.

[27] Mariano Cabezas, Arnau Oliver, Xavier Lladó, Jordi Freixenet, and Meritxell Bach Cuadra. A review of atlas-based segmentation for magnetic resonance brain images. *Computer methods and programs in biomedicine*, 104(3):e158–e177, 2011.

[28] Hongmin Cai, Ragini Verma, Yangming Ou, Seung-koo Lee, Elias R Melhem, and Christos Davatzikos. Probabilistic segmentation of brain tumors based

on multi-modality magnetic resonance images. In *2007 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 600–603. IEEE, 2007.

[29] Weiling Cai, Songcan Chen, and Daoqiang Zhang. Fast and robust fuzzy c-means clustering algorithms incorporating local information for image segmentation. *Pattern recognition*, 40(3):825–838, 2007.

[30] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018.

[31] Adrià Casamitjana, Marcel Catà, Irina Sánchez, Marc Combalia, and Verónica Vilaplana. Cascaded v-net using roi masks for brain tumor segmentation. In *International MICCAI Brainlesion Workshop*, pages 381–391. Springer, 2017.

[32] Joshua E Cates, Ross T Whitaker, and Greg M Jones. Case study: an evaluation of user-assisted hierarchical watershed segmentation. *Medical Image Analysis*, 9(6):566–578, 2005.

[33] Fusak Cheevasuvit, Henri Maitre, and Daniel Vidal-Madjar. A robust method for picture segmentation based on a split-and-merge procedure. *Computer Vision, Graphics, and Image Processing*, 34(3):268–281, 1986.

[34] Shiuh-Yung Chen, Wei-Chung Lin, and Chin-Tu Chen. Split-and-merge image segmentation based on localized feature analysis and statistical tests. *CVGIP: Graphical Models and Image Processing*, 53(5):457–475, 1991.

[35] Xiaoran Chen and Ender Konukoglu. Unsupervised detection of lesions in brain mri using constrained adversarial auto-encoders. *arXiv preprint arXiv:1806.04972*, 2018.

[36] Xuan Chen, Jun Hao Liew, Wei Xiong, Chee-Kong Chui, and Sim-Heng Ong. Focus, segment and erase: an efficient network for multi-label brain tumor segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 654–669, 2018.

[37] Yunjie Chen, Zhihui Cao, Chunzheng Cao, Jianwei Yang, and Jianwei Zhang. A modified u-net for brain mr image segmentation. In *International Conference on Cloud Computing and Security*, pages 233–242. Springer, 2018.

[38] Youngwon Choi, Yongchan Kwon, Hanbyul Lee, Beom Joon Kim, Myunghee Cho Paik, and Joong-Ho Won. Ensemble of deep convolutional neural networks for prognosis of ischemic stroke. In *International Workshop on Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 231–243. Springer, 2016.

[39] Vincent FH Chong, Jia-Yin Zhou, James BK Khoo, Jing Huang, and Tuan-Kay Lim. Tongue carcinoma: tumor volume measurement. *International Journal of Radiation Oncology* Biology* Physics*, 59(1):59–66, 2004.

[40] Matthew C Clark, Lawrence O Hall, Dmitry B Goldgof, Robert Velthuizen, F Reed Murtagh, and Martin S. Silbiger. Automatic tumor segmentation using knowledge-based techniques. *IEEE transactions on medical imaging*, 17(2):187–201, 1998.

[41] Olivier Clatz, Hervé Delingette, I-F Talos, Alexandra J Golby, Ron Kikinis, Ferenc A Jolesz, Nicholas Ayache, and Simon K Warfield. Robust nonrigid registration to capture brain shift from intraoperative mri. *IEEE transactions on medical imaging*, 24(11):1417–1427, 2005.

[42] Cortes Corinna and V Vapnik. Support-vector networks, machine learning. *Available from World Wide Web:¡ http://www. springerlink. com/content/k238jx04hm87j80g*, 1995.

[43] Thomas M Cover and Joy A Thomas. Elements of information theory second edition solutions to problems. *Internet Access*, 2006.

[44] Erik Dam, Marco Loog, and Marloes Letteboer. Integrating automatic and interactive brain tumor segmentation. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 790–793. IEEE, 2004.

[45] Yang Deng, Yao Sun, Yongpei Zhu, Mingwang Zhu, Wei Han, and Kehong Yuan. A strategy of mr brain tissue images' suggestive annotation based on modified u-net. *arXiv preprint arXiv:1807.07510*, 2018.

[46] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[47] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.

[48] Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1734–1747, 2015.

[49] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[50] MK Erskine, LL Cook, KE Riddle, Joseph Ross Mitchell, and Stephen J Karlik. Resolution-dependent estimates of multiple sclerosis lesion loads. *Canadian journal of neurological sciences*, 32(2):205–212, 2005.

[51] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[52] M Filippi, MA Horsfield, A Campi, S Mammi, C Pereira, and G Comi. Resolution-dependent estimates of lesion volumes in magnetic resonance imaging studies of the brain in multiple sclerosis. *Annals of Neurology: Official Journal of the American Neurological Association and the Child Neurology Society*, 38(5):749–754, 1995.

[53] Lynn M Fletcher-Heath, Lawrence O Hall, Dmitry B Goldgof, and F Reed Murtagh. Automatic segmentation of non-enhancing brain tumors in magnetic resonance images. *Artificial intelligence in medicine*, 21(1-3):43–63, 2001.

[54] Frank Gaillard. Intracranial compartments: Radiology reference article. `https://radiopaedia.org/articles/intracranial-compartments`. Accessed January 19,2020.

[55] Mina Ghaffari, Arcot Sowmya, and Ruth Oliver. Automated brain tumor segmentation using multimodal brain scans: A survey based on models submitted to the brats 2012–2018 challenges. *IEEE Reviews in Biomedical Engineering*, 13:156–168, 2019.

[56] Ali Gooya, Kilian M Pohl, Michel Bilello, George Biros, and Christos Davatzikos. Joint segmentation and deformable registration of brain scans guided by a tumor growth model. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 532–540. Springer, 2011.

[57] Philip Haeusser, Johannes Plapp, Vladimir Golkov, Elie Aljalbout, and Daniel Cremers. Associative deep clustering: Training a classification network with no labels. In *German Conference on Pattern Recognition*, pages 18–32. Springer, 2018.

[58] Gordon J Harris, Patrick E Barta, Luon W Peng, Seong Lee, Paul D Brettschneider, Amish Shah, Jeffery D Henderer, Thomas E Schlaepfer, and Godfrey D Pearlson. Mr volume segmentation of gray matter and white matter using manual thresholding: dependence on image brightness. *American journal of neuroradiology*, 15(2):225–230, 1994.

[59] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31, 2017.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[61] Andrew Jesson and Tal Arbel. Brain tumor segmentation using a 3d fcn with multi-scale loss. In *International MICCAI Brainlesion Workshop*, pages 392–402. Springer, 2017.

[62] Shiyong Ji, Benzheng Wei, Zhen Yu, Gongping Yang, and Yilong Yin. A new multistage medical segmentation method based on superpixel and fuzzy clustering. *Computational and mathematical methods in medicine*, 2014, 2014.

[63] Xu Ji, João F Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9865–9874, 2019.

[64] Konstantinos Kamnitsas, Christian Ledig, Virginia FJ Newcombe, Joanna P Simpson, Andrew D Kane, David K Menon, Daniel Rueckert, and Ben Glocker. Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78, 2017.

[65] Michael R Kaus, Simon K Warfield, Arya Nabavi, Peter M Black, Ferenc A Jolesz, and Ron Kikinis. Automated segmentation of mr images of brain tumors. *Radiology*, 218(2):586–591, 2001.

[66] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[67] Simon Kohl, Bernardino Romera-Paredes, Clemens Meyer, Jeffrey De Fauw, Joseph R Ledsam, Klaus Maier-Hein, SM Ali Eslami, Danilo Jimenez Rezende, and Olaf Ronneberger. A probabilistic u-net for segmentation of ambiguous images. In *Advances in Neural Information Processing Systems*, pages 6965–6975, 2018.

[68] Xiangmao Kong, Guoxia Sun, Qiang Wu, Ju Liu, and Fengming Lin. Hybrid pyramid u-net model for brain tumor segmentation. In *International conference on intelligent information processing*, pages 346–355. Springer, 2018.

[69] Ender Konukoglu, William M Wells, Sébastien Novellas, Nicholas Ayache, Ron Kikinis, Peter M Black, and Kilian M Pohl. Monitoring slowly evolving tumors. In *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 812–815. IEEE, 2008.

[70] Matthieu Lê, Jan Unkelbach, Nicholas Ayache, and Hervé Delingette. Sampling image segmentations for uncertainty quantification. *Medical image analysis*, 34:42–51, 2016.

[71] M Letteboer, W Niessen, P Willems, Erik B Dam, and M Viergever. Interactive multi-scale watershed segmentation of tumors in mr brain images. In *Proc. of the IMIVA workshop of MICCAI*. Citeseer, 2001.

[72] Geng-Cheng Lin, Wen-June Wang, Chung-Chia Kang, and Chuin-Mu Wang. Multispectral mr images segmentation based on fuzzy knowledge and modified seeded region growing. *Magnetic resonance imaging*, 30(2):230–246, 2012.

[73] Jia Liu, Fang Chen, Changcun Pan, Mingyu Zhu, Xinran Zhang, Liwei Zhang, and Hongen Liao. A cascaded deep convolutional neural network for joint segmentation and genotype prediction of brainstem gliomas. *IEEE Transactions on Biomedical Engineering*, 65(9):1943–1952, 2018.

[74] Jin Liu, Min Li, Jianxin Wang, Fangxiang Wu, Tianming Liu, and Yi Pan. A survey of mri-based brain tumor segmentation methods. *Tsinghua science and technology*, 19(6):578–595, 2014.

[75] Zhihua Liu, Long Chen, Lei Tong, Feixiang Zhou, Zheheng Jiang, Qianni Zhang, Caifeng Shan, Yinhai Wang, Xiangrong Zhang, Ling Li, et al. Deep learning based brain tumor segmentation: A survey. *arXiv preprint arXiv:2007.09479*, 2020.

[76] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[77] Oskar Maier, Bjoern H , Janina von der Gablentz, Levin Häni, Mattias P Heinrich, Matthias Liebrand, Stefan Winzeck, Abdul Basit, Paul Bentley, Liang

Chen, et al. ISLES 2015 - a public evaluation benchmark for ischemic stroke lesion segmentation from multispectral MRI. *Medical image analysis*, 35:250–269, 2017.

[78] Andreas Mang, Julia A Schnabel, William R Crum, Marc Modat, Oscar Camara-Rey, Christoph Palm, Gisele Brasil Caseiras, H Rolf Jäger, Sébastien Ourselin, Thorsten M Buzug, et al. Consistency of parametric registration in serial mri studies of brain tumor progression. *International Journal of Computer Assisted Radiology and Surgery*, 3(3-4):201–211, 2008.

[79] Jose G Merino, Lawrence L Latour, Jason W Todd, Marie Luby, Peter D Schellinger, Dong-Wha Kang, and Steven Warach. Lesion volume change after treatment with tissue plasminogen activator can discriminate clinical responders from nonresponders. *Stroke*, 38(11):2919–2923, 2007.

[80] Gangolf Mittelhaeusser and Frithjof Kruggel. Fast segmentation of brain magnetic resonance tomograms. In *International Conference on Computer Vision, Virtual Reality, and Robotics in Medicine*, pages 237–241. Springer, 1995.

[81] Pim Moeskops, Max A Viergever, Adriënne M Mendrik, Linda S De Vries, Manon JNL Benders, and Ivana Išgum. Automatic segmentation of mr brain images with a convolutional neural network. *IEEE transactions on medical imaging*, 35(5):1252–1261, 2016.

[82] Lewis B Morgenstern and Ralph F Frankowski. Brain tumor masquerading as stroke. *Journal of neuro-oncology*, 44(1):47–52, 1999.

[83] Sarah Parisot, Hugues Duffau, Stéphane Chemouny, and Nikos Paragios. Joint tumor segmentation and dense deformable registration of brain mr images. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 651–658. Springer, 2012.

[84] Hyun-Joo Park, Andre G Machado, Jessica Cooperrider, Havan Truong-Furmaga, Matthew Johnson, Vibhuti Krishna, Zhihong Chen, and John T Gale. Semi-automated method for estimating lesion volumes. *Journal of neuroscience methods*, 213(1):76–83, 2013.

[85] Kevin Patel and David B Clifford. Bacterial brain abscess. *The Neurohospitalist*, 4(4):196–204, 2014.

[86] Theodosios Pavlidis and Y-T Liow. Integrating region growing and edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):225–233, 1990.

[87] WE Phillips II, RP Velthuizen, S Phuphanich, LO Hall, LP Clarke, and ML Silbiger. Application of fuzzy c-means segmentation technique for tissue differentiation in mr images of a hemorrhagic glioblastoma multiforme. *Magnetic resonance imaging*, 13(2):277–290, 1995.

[88] Kilian M Pohl, John Fisher, W Eric L Grimson, Ron Kikinis, and William M Wells. A bayesian model for joint segmentation and registration. *NeuroImage*, 31(1):228–239, 2006.

[89] RadiologyInfo.org. Brain tumors. `https://www.radiologyinfo.org/en/pdf/braintumor.pdf`. Accessed January 19,2020.

[90] Kevin Raina., Uladzimir Yahorau., and Tanya Schmah. Exploiting bilateral symmetry in brain lesion segmentation with reflective registration. In *Proceedings of the 13th International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 2: BIOIMAGING,*, pages 116–122. INSTICC, SciTePress, 2020.

[91] Rajeev Ratan, Sanjay Sharma, and SK Sharma. Multiparameter segmentation and quantization of brain tumor from mri images. *Indian Journal of Science and Technology*, 2(2):11–15, 2009.

[92] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *null*, page 10. IEEE, 2003.

[93] CS Rivers, JM Wardlaw, PA Armitage, ME Bastin, TK Carpenter, V Cvoro, PJ Hand, and MS Dennis. Do acute diffusion-and perfusion-weighted mri lesions identify final infarct volume in ischemic stroke? *Stroke*, 37(1):98–104, 2006.

[94] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[95] Abhijit Guha Roy, Sailesh Conjeti, Nassir Navab, and Christian Wachinger. Inherent brain segmentation quality control from fully convnet monte carlo sampling. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 664–672. Springer, 2018.

[96] Su Ruan, Stéphane Lebonvallet, Abderrahim Merabet, and Jean-Marc Constans. Tumor segmentation from a multispectral mri images by using support vector machine classification. In *2007 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1236–1239. IEEE, 2007.

[97] Su Ruan, Nan Zhang, Qingmin Liao, and Yuemin Zhu. Image fusion for following-up brain tumor evolution. In *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 281–284. IEEE, 2011.

[98] Norhashimah Mohd Saad, Syed Abdul Rahman Syed Abu Bakar, Ahmad Sobri Muda, and Musa Mohd Mokji. Review of brain lesion detection and classification using neuroimaging analysis techniques. *Jurnal Teknologi*, 74(6), 2015.

[99] Sriparna Saha and Sanghamitra Bandyopadhyay. Mri brain image segmentation by fuzzy symmetry based genetic clustering technique. In *2007 IEEE Congress on Evolutionary Computation*, pages 4417–4424. IEEE, 2007.

[100] Saif D Salman and Ahmed A Bahrani. Segmentation of tumor tissue in gray medical images using watershed transformation method. *Int. J. Adv. Comp. Techn.*, 2(4):123–127, 2010.

[101] YM Salman, MA Assal, AM Badawi, SM Alian, and M El-M El-Bayome. Validation techniques for quantitative brain tumors measurements. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pages 7048–7051. IEEE, 2006.

[102] Gustav K Schulthess and Christoph L Zollikofer. *Diseases of the brain, head & neck, spine: diagnostic imaging and interventional techniques*. Springer Science & Business Media, 2008.

[103] Sara Sedlar. Brain tumor segmentation using a multi-path cnn based method. In *International MICCAI Brainlesion Workshop*, pages 403–422. Springer, 2017.

[104] Haocheng Shen, Ruixuan Wang, Jianguo Zhang, and Stephen J McKenna. Boundary-aware fully convolutional network for brain tumor segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 433–441. Springer, 2017.

[105] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[106] Thorvald Julius Sørensen. *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons*. I kommission hos E. Munksgaard, 1948.

[107] Andreas Stadlbauer, Ewald Moser, Stephan Gruber, Rolf Buslei, Christopher Nimsky, Rudolf Fahlbusch, and Oliver Ganslandt. Improved delineation of brain tumors: an automated method for segmentation based on pathologic changes of 1h-mrsi metabolites in gliomas. *Neuroimage*, 23(2):454–461, 2004.

[108] J Suckling, T Sigmundsson, K Greenwood, and ET Bullmore. A modified fuzzy clustering algorithm for operator independent brain tissue classification of dual echo mr images. *Magnetic resonance imaging*, 17(7):1065–1076, 1999.

[109] Yun-Chang Sung, Ki-Sun Han, Chang-Jun Song, Seung-Moo Noh, and Jong-Won Park. Threshold estimation for region segmentation on mr image of brain having the partial volume artifact. In *WCC 2000-ICSP 2000. 2000 5th International Conference on Signal Processing Proceedings. 16th World Computer Congress 2000*, volume 2, pages 1000–1009. IEEE, 2000.

[110] László Szilagyi, Zoltán Benyo, Sándor M Szilágyi, and HS Adam. Mr brain image segmentation using an enhanced fuzzy c-means algorithm. In *Proceedings of the 25th annual international conference of the IEEE engineering in medicine and biology society (IEEE Cat. No. 03CH37439)*, volume 1, pages 724–726. IEEE, 2003.

[111] Philippe Thévenaz, Thierry Blu, and Michael Unser. Interpolation revisited [medical images application]. *IEEE Transactions on medical imaging*, 19(7):739–758, 2000.

[112] Zhiqiang Tian, Lizhi Liu, Zhenfeng Zhang, and Baowei Fei. Superpixel-based segmentation for 3d prostate mr images. *IEEE transactions on medical imaging*, 35(3):791–801, 2015.

[113] Arti Tiwari, Shilpa Srivastava, and Millie Pant. Brain tumor segmentation and classification from magnetic resonance images: Review of selected methods from 2014 to 2019. *Pattern Recognition Letters*, 131:244–260, 2020.

[114] Gregor Urban, M Bendszus, F Hamprecht, and J Kleesiek. Multi-modal brain tumor segmentation using deep convolutional neural networks. *MICCAI BraTS (brain tumor segmentation) challenge. Proceedings, winning contribution*, pages 31–35, 2014.

[115] Nishant Verma, Matthew C Cowperthwaite, and Mia K Markey. Superpixels in brain mr image analysis. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 1077–1080. IEEE, 2013.

[116] Ragini Verma, Evangelia I Zacharaki, Yangming Ou, Hongmin Cai, Sanjeev Chawla, Seung-Koo Lee, Elias R Melhem, Ronald Wolf, and Christos Davatzikos. Multiparametric tissue characterization of brain neoplasms and their recurrence using pattern classification of mr images. *Academic radiology*, 15(8):966–977, 2008.

[117] C Vijayakumar and Damayanti Chandrashekhar Gharpure. Development of image-processing software for automatic segmentation of brain tumors in mr images. *Journal of medical physics/Association of Medical Physicists of India*, 36(3):147, 2011.

[118] Guotai Wang, Wenqi Li, Sébastien Ourselin, and Tom Vercauteren. Automatic brain tumor segmentation using cascaded anisotropic convolutional neural net-

works. In *International MICCAI brainlesion workshop*, pages 178–190. Springer, 2017.

[119] Stefan Winzeck, Arsany Hakim, Richard McKinley, José AADSR Pinto, Victor Alves, Carlos Silva, Maxim Pisov, Egor Krivov, Mikhail Belyaev, Miguel Monteiro, et al. ISLES 2016 and 2017-benchmarking ischemic stroke lesion outcome prediction based on multispectral MRI. *Frontiers in neurology*, 9, 2018.

[120] S. Wittenauer and L. Smith. Ischaemic and haemorrhagic stroke. `https://www.who.int/medicines/areas/priority_medicines/BP6_6Stroke.pdf`. Accessed January 19,2020.

[121] Koon-Pong Wong. Medical image segmentation: methods and applications in functional imaging. In *Handbook of biomedical image analysis*, pages 111–182. Springer, 2005.

[122] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, 2016.

[123] Nan Zhang, Su Ruan, Stéphane Lebonvallet, Qingmin Liao, and Yuemin Zhu. Multi-kernel svm based classification for brain tumor segmentation of mri multi-sequence. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3373–3376. IEEE, 2009.

[124] Liya Zhao and Kebin Jia. Multiscale cnns for brain tumor segmentation and diagnosis. *Computational and mathematical methods in medicine*, 2016, 2016.

[125] J Zhou, KL Chan, VFH Chong, and Shankar M Krishnan. Extraction of brain tumor from mr images using one-class support vector machine. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pages 6411–6414. IEEE, 2006.

[126] Darko Zikic, Yani Ioannou, Matthew Brown, and Antonio Criminisi. Segmentation of brain tumor tissues with convolutional neural networks. *Proceedings MICCAI-BRATS*, pages 36–39, 2014.

[127] Robert Zivadinov, Mari Heininen-Brown, Claudiu V Schirda, Guy U Poloni, Niels Bergsland, Christopher R Magnano, Jacqueline Durfee, Cheryl Kennedy, Ellen Carl, Jesper Hagemeier, et al. Abnormal subcortical deep-gray matter susceptibility-weighted imaging filtered phase measurements in patients with multiple sclerosis: a case-control study. *Neuroimage*, 59(1):331–339, 2012.

Index