DEEP LEARNING MODELS FOR IMAGE-BASED DISEASE CLASSIFICATION AND ASSISTIVE TECHNOLOGY RELATED TO ALZHEIMER'S DISEASE

by

Ke Xu

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Engineering Technology

West Lafayette, Indiana

August 2019

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Suranjan Panigrahi, Chair
School of Electrical Engineering Technology

Dr. Ragu Athinarayanan
School of Electrical Engineering Technology

Dr. Daniel Leon-Salas
School of Electrical Engineering Technology

Dr. Frederick Berry
School of Mechanical Engineering Technology

Approved by:

Dr. Kathryne Newton
Head of the Graduate Program

For my wife, Na Li and my parents

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my major professor, Dr. Suranjan Panigrahi, for providing me opportunity and guidance in my Ph.D. research. I have learned many treasured academic and life experience from him. I would also like to thank all my committee members, Dr. Ragu Athinarayanan, Dr. Daniel Leon-Salas, and Dr. Frederick Berry for providing valuable suggestion in my research.

I would also like to extend my gratitude to Purdue Rosen Center for Advanced Computing (RCAC) and Mr. Lev Gorenstein, the computation specialist of RCAC, for providing me the computational resource and support throughout the process. My special thanks to the Alzheimer's Disease Neuroimaging Initiative (ADNI) for providing the research evidence; The Engineering Computer Network (ECN), for providing the IT support.

I also want to thank all my fellow colleagues and staff member in the department: Mr. Xiaoyu Yu, Ms Ridhi Dep, Ms Niedra Mcleland, Ms Amanda Wilson and Ms Debbie Hulsey, for their help and kindness. I would like to thank my family and friends for their unlimited support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AD	Alzheimer's Disease
ADNI	Alzheimer's Disease Neuroimaging Initiative
CNN	Convolutional Neural Network
MCI	Mild Cognitive Impairment
MRI	Magnetic Resonance Imaging
NC	Normal Control

ABSTRACT

Author: Xu, Ke. PhD
Institution: Purdue University
Degree Received: August 2019
Title: Deep Learning Models for Image-Based Disease Classification and Assistive Technology Related to Alzheimer's Disease
Committee Chair: Suranjan Panigrahi

Alzheimer's disease (AD), is a devastating neurodegenerative disorder that destroys the patient's ability to perform daily living task and eventually, takes their lives. Currently, there are 5.8 million people in North America that suffer from AD. This number is projected to by 13.8 million by the year of 2050. For many years, researchers have been dedicated on performing automated diagnosis based on neuroimaging. There are critical needs in two aspects of AD: 1) computer-based AD classification with MRI images; 2) computer-based tools/system to enhance the AD patient's quality of life. We are addressing these two gaps via two specific objectives in this study.

For objective 1, the task is to develop a machine-learning based intelligent model for classification of AD conditions (Normal Control [NC], Mild Cognitive Impairment [MCI], Alzheimer's disease [AD]) based on MRI images. Specifically, four different deep learning models were developed and assessed. The overall average accuracy for AD classification is 81.5%, provided by Multi-Layer-Output model.

. For objective 2, a deep learning model was developed and evaluated to recognitze three specific type of indoor scenes (bedroom, living room and dining room). An accuracy of 97% was obtained.

This study showed the potential of application in deep learning models for two different aspects of AD - disease classification and intelligent model-based assistive device for AD patients. Further research and development activities are recommended to further validate these findings on larger and different datasets.

CHAPTER 1. INTRODUCTION

1.1 Introduction

For many years, the temptation of achieving machine intelligence has drawn the interests of many researchers and the public. Many films on artificial intelligence have been made, demonstrating people's constant attention on the subject matter. However, not many advances have been accomplished in the meantime. This is due to two major obstacles. One obstacle is the difficulty in performing accurate and efficient feature engineering (Domingos, 2012). Feature engineering refers to the process of mathematical characterization of unique evidences that relate to a target task. It involves the utilization of domain knowledge, which is problematic because it requires the developer to be an expert in the target domain. Even if the developer did acquire such experience, it could still be difficult if the scientific understanding of such a domain was still indefinite. The features engineered from uncertain observations and questionable speculations would not lead to a discriminative representation. Most of the time, what happened to manual feature engineering was constantly tuning the crafted features to find the best approximation. The second obstacle is the inability to process large amounts of data and complex feature representation. Many classification tasks are very complex. Naturally, for such a task, we anticipate a feature representation with huge complexity or, its most straightforward form, a representation with a huge number of features. Statistically, to train a classifier to use such a huge feature representation, a gigantic dataset is needed to avoid the overfitting problem. Unfortunately, computation hardware back in the day was not capable of handling heavy computation load like this.

Since 2008, an advanced form of machine learning known as deep learning has caught the interest of many researchers and industries. The idea of deep learning is to perform multiple

levels of feature extraction, creating a high-level feature embedding automatically. Compared with conventional machine learning, deep learning applies classification using similar methods such as artificial neural network (ANN), support vector machine (SVM) and so on. The difference is deep learning performs feature extraction on the raw data while the conventional machine learning uses manually crafted features. This advance from shallow learning to deep learning theoretically solved the feature engineering problem mentioned earlier. Another critical factor behind this advancement was the improvement on the computing hardware, as shown in Figure 1.1. Given this opportunity, many deep learning models have been introduced and proved to be very effective in many tasks, such as AlexNet (Krizhevsky, Sutskever, & Hinton, 2012), GoogLeNet (Szegedy, Liu, Jia, & Sermanet, 2015), ResNet (He, Zhang, Ren, & Sun , 2016), and DenseNet (Huang, Liu, Van Der Maaten, & Weinberger, 2017). The rapid development of deep learning has altered the landscape of many scientific and industrial areas. Machine intelligence techniques have boosted the performance of facial recognition, speech recognition, natural language processing, and many other applications. Among other applications, computer vision related machine intelligence is one the most popular research field. Visual cognition is the most common and direct method of how human interact with the world. Ideally, by mimicking human visual perception, we could teach machine recognize the world the same way as we do.

One of the rising research foci in imaging powered machine learning is the learning of medical imaging. To care the wellness of human being is always one of the most significant tasks of scientists. Medical imaging, benefited from the advancement of equipment, has become more and more favored by physicians. Instead of looking at data from charts, medical imaging now can help physicians visually reconstruct patients' situation from two or even three

dimensions. Also, medical imaging brings less discomfort to the patients because it is non-invasive.

Of all the human organs and systems, the brain and nervous system benefit the most from advanced medical imaging. Being the most important and delicate organ in humans, the brain is always prominent in medical research. Consisting of numerous brain cells, matters, and bodily fluids, the brain controls and coordinates the human body to work collectively and form the world's most robust system. The study of the brain, or neurology, has always been limited by how to effectively examine the brain. So far, other than postmortem dissection, neuroimaging is the most common technique of obtaining evidence from the brain. The common imaging acquisition includes magnetic resonance imaging (MRI) and positron emission tomography (PET).

One common brain disease that is worth investigating is Alzheimer's disease (AD). Alzheimer's disease is a chronic neurodegenerative disease that slowly destroys the patient's brain. It was named after Dr. Alois Alzheimer who gave the first description of the disease in 1906 (Berchtold & Cotman, 1998). Alzheimer's is the most common form of dementia, contributing to 60% to 80% of the cases (World Health Organization, 2018). Currently, the clinical diagnosis of AD is determined by measuring the patient's cognitive decline. However, studies show that a definite diagnosis of cognitive impairment appears in the late stage of AD (Pawlowski, Meuth, & Duning, 2017). If neurologists could manage to provide a diagnosis in AD's moderate stage, or mild cognitive impairment (MCI), treatment and preventive care could be provided to the patients, improving the family's wellness.

Most of the research done in this field focuses on using biomarkers in the nervous system to perform an automated diagnosis. According to our current biochemical and pathological

understanding, abnormal existence of biomarkers could be found in patients of AD (Hashimoto, Rockenstein, Crews, & Masliah, 2003; Wenk, 2003). However, given the limited understanding on the disease, these biomarkers might not be accurately related to the disease, leading to the relative unreliability of these methods.

Aside from the damage caused by the disease to the patients, it also brings massive damage to the caregivers and the families. Alzheimer's patients gradually lose their ability to live by themselves. Sooner or later, caregivers need to intervene in patients' daily life. However, in the early stage of AD, patients tend to have more lucidity than confusion, meaning that, technically, they do not require as much attention as patients in late stage AD. In reality, the caregivers, who often are unpaid or are family members, would not take the risk and provide the care anyway to be safe. This process can be very prolonged because the progressiveness of AD is highly unpredictable, leading to anxiety and agony for both patients and caregivers.

Therefore, in this work, I reported several models targeted on improving the wellness of both AD patients and their caregivers. Specifically, several deep learning models were developed and investigated to increase the performance of the AD stage classification (normal control [NC], mild cognitive impairment [MCI], Alzheimer's disease [AD]). Additionally, an intelligent model was developed specifically for identifying general indoor environments which would serve as a segment of AD assistive system. I report the development of the indoor scene understanding model based on deep learning, targeting three indoor room types: living room, bedroom, and dining room. The proposed model will be capable of identifying these typical indoor room types based on a single image. The model hyper-parameters were selected through a grid search to obtain the optimal combination.

1.2 Organization

This dissertation contains five major chapters and several appendices. Chapter 1 provides an overall introduction to the research area and a general statement to the objectives of this work.

Chapter 2 provides a thorough review of related works. It begins with the pathology and biochemistry of Alzheimer's disease and the current development of Alzheimer's caregiving and assistive devices. Next, prior research on the automated diagnosis of AD is reviewed from both shallow learning and deep learning perspectives. Then, the prior research on indoor scene understanding is outlined. Finally, a detailed analysis of current research illustrates the existing gaps in the target research domain.

Chapter 3 elaborates the objectives of this dissertation.

Chapter 4 reports the proposed methodologies for the mentioned objectives. In this chapter, the deep 3D convolutional neural network (CNN) model and the deep 3D residual neural network (ResNet) model are first elaborated. The novel revision of Multi-Layer-Output (MLO) is presented for each of the two proposed models. Finally, a deep 2D ResNet model is proposed for indoor scene understanding.

Chapter 5 describes the experiment results and discusses the observed data. A comparison between the proposed methods and other state-of-the-art frameworks is also conducted in Chapter 5.

Chapter 6 provides the conclusion of the proposed work, discusses the limitations of the framework, and provides recommendations for future research.

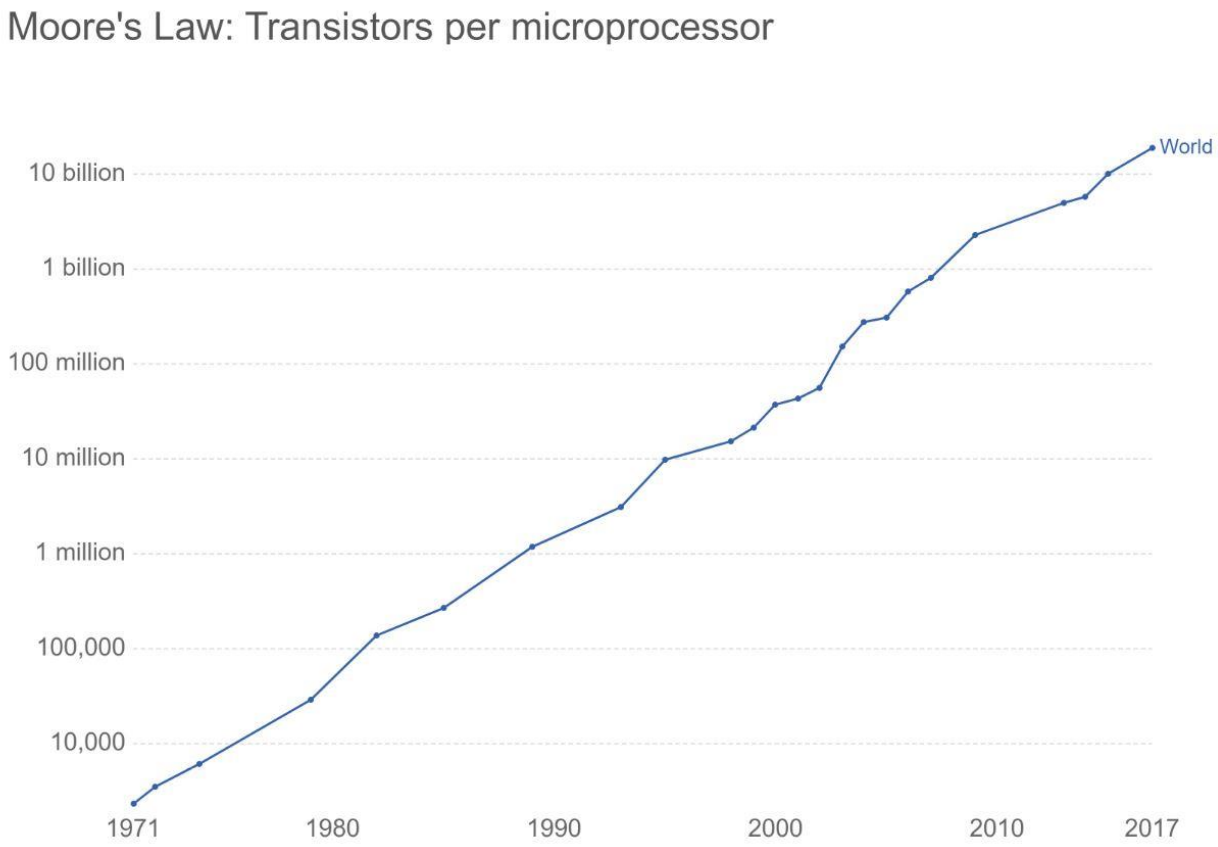


Figure 1.1 Transistors per microprocessor.

CHAPTER 2. LITERATURE REVIEW

Being able to differentiate the progressiveness of Alzheimer's disease patients has always been challenging. Due to our lack of understanding on this disease, such judgement can only be done by very experienced neurologists with unavoidable deviation. Therefore, to automatically perform such judgement through computer-aided system is very valuable. On the other hand, to assist early stage Alzheimer's patients, to enhance their ability of individual living, has both scientific and realistic implications.

In this chapter, the basic pathology and biochemistry of AD were first briefly reviewed. After that, the clinical diagnosis criteria of AD was introduced. Following was the review on the previous conducted research, from both shallow learning perspective and deep learning perspective. Finally, the review of AD assistive device and indoor scene understanding was provided.

2.1 Automated Diagnosis of Alzheimer's Disease

The Alzheimer's disease (AD), the major form elder cognitive impairment, is an irreversible, progressive neurodegenerative brain disorder that slowly destroys patients' cognitive function and eventually, takes their lives. For the past 20 years, the reported death from AD has increased by 145% while other major disease showed significant decreases (Gaugler et al., 2019). It is reported that there are 5.9 million adults aged 65 and more, with Alzheimer's disease and related dementia (ADRD) in North America, and this number is estimated to be 13.9 million by the year 2060 (Gaugler et al., 2019). This translates into a new case every 33 seconds (Corrada et al., 2010; Hebert et al., 2001). Given its significance, the World Health Organization officially recognized AD as the most common form of dementia and possibly contributes to 60% - 70% of cases (World Health Organization, 2012).

2.1.1 Pathology and Biochemistry

In 2008, the World Health Organization (WHO) declared that dementia is a priority condition through the Mental Health Gap Action Program. Duthey (2013) gave a comprehensive overview on the current mental healthcare situation and related research. The author emphasized the significance in early mental risk identification and diagnosis. The author also identified that Alzheimer's disease is the most common and arduous form of dementia because Alzheimer's disease is currently irreversible and incurable. To make things worse, there are no biomarkers that could identify Alzheimer's disease perfectly before the disease progresses. The most common approaches are combining brain imaging and clinical memory assessment. The common symptoms of Alzheimer's disease was shown in Table 2.1 (World Health Organization, 2012).

The current pathological understanding of AD is, abnormal processing of the transmembrane Aβ precursor protein, causing a family of peptides that form the Beta-amyloids. The insoluble of these peptides, mostly Aβ42, have a propensity for self-aggregation into fibrils that form the senile plaques. Also the abnormal aggregation of microtubule-associated tau protein, will disrupt and kill the nerve cells (Iqbal, Liu, Gong, & Grundke-Iqbal, 2010; Medeiros, BagliettoVargas, & LaFerla, 2011; Mietelska-Porowska et al., 2014; Singh, Srivastav, Yadav, Srikrishna, & Perry, 2016). Observation of damaged neurons and synapses in the cerebral cortex and subcortical regions can be found from patients. These damages caused the degeneration in the brain, affecting patient's cognitive function.

2.1.2 Clinical Diagnosis of Alzheimer's Disease

The major difficulty in prodromal Alzheimer's disease diagnosis is because there is no clear biomarker to identify it. According to WHO (World Health Organization, 2012),

Alzheimer's disease is most likely to develop after age 65. Clinical reports showed that prodromal Alzheimer's disease, amnestic mild cognitive impairment and aging cognitive normal share very similar symptoms. Petersen et al. (1999) launched a research targeted on characterizing clinical subjects with mild cognitive impairment. The researchers conducted an experiment among 76 mild cognitive impairment patients with 234 healthy control subjects and 106 mild Alzheimer's disease patients. The test subjects were divided into three groups. Six mental function evaluations were conducted on the subjects. The classification of dementia and Alzheimer's disease were done using criteria published by Spitzer and Williams (1980). The results showed that the primary difference between healthy normal and mild cognitive impairment is the appearance of amnesia while other cognitive functions showed similar result. For mild cognitive impairment and mild Alzheimer's disease, these patients had similar evaluation in amnesia but Alzheimer's disease patients also had other cognitive difficulties. Dubois and Albert (2004) also conducted research on the topic with biological interpretation rather than clinical performance. The researcher later revised the diagnostic criteria of Alzheimer's disease on the basis of National Institute of Neurological Disorders and Stroke Alzheimer Disease and Related Disorders [NINCDS—ADRDA] criteria (Dubois et al., 2007). A detailed clinical diagnostic criteria can be found in Table 2.2. The new criteria group the symptoms into one core diagnostic criteria and four supportive features. A subject with a symptom from core criteria (cognitive impairments) and one or more supportive features (e.g., abnormal biomarker, cerebrospinal fluid, functional/structural MRI, etc.) could be diagnosed as Alzheimer's disease. This set of diagnostic criteria has been widely adopted by future research.

To perform the clinical diagnosis of AD, according to the National Institute of Aging's revised core diagnostic criteria, largely relies on the observation of cognitive function

impairment, deficit and insidious onset (McKhann et al., 2011). Another way for definite diagnosis is through postmortem autopsy (Klppel et al., 2008). The biomarker evidence, such as low cerebrospinal fluid (CSF) Aβ42 and elevated CSF tau, can help increase the accuracy of diagnosing the clinical dementia is AD pathophysiological process. But it is not suggested to add AD biomarker test to routine diagnostic process due to lack of understanding and therefore need further research (McKhann et al., 2011). The objectives of AD research partially focus on prodromal diagnosis using noninvasive examination. Recently, researchers have been investigating the possibility of using neuroimaging as evidence to perform AD diagnosis and have made great progress. These work are inspired by the massive tangible and latent information embedded in different imaging modalities like Magnetic Resonance Imaging (MRI) and Positron Emission Tomography (PET). Some early successes were achieved using statistical based method like SVM and Bayesian classifier (Plant et al., 2010). And soon the community has shifted towards deep learning oriented research.

2.1.3 Automated Diagnosis by Machine Learning

2.1.3.1 Shallow Learning

The use of machine learning and neuroimaging had been very popular since 21st century. This is mostly because machine learning is capable of processing massive high-dimensional data while humans perform diagnosis on very limited amount of data and personal experience.

In early years, most of researchers used methods based on volumetric measurement of regions of interest (Convit et al., 2000; Juottonen et al., 1998). De Leon et al. (1997) proposed to use the volume reduction of hippocampal as major criteria to distinguish dementia of the Alzheimer's type (DAT) from mild cognitive normal (MCI) and elderly (NL). The researchers conducted the experiment on three groups of screened age- and education-matched subjects.

Additionally, when discriminating DAT from MCI subjects, the use of gyrus volume had further improved the classification accuracy. Thus they reached a conclusion that the hippocampal volume reduction in temporal lobe and the gyrus volume reduction in the lateral lobe could be adopted as a major feature in prodromal Alzheimer's disease prediction. Similar results were obtained by other researchers also (Colliot et al., 2008; Gerardin et al., 2009).

The second category used voxel-based methods and it has been the most popular structural MRI based approach. In this class, the features are created by the density of voxel. Many research had been done using this method (Davatzikos, Fan, Wu, Shen, & Resnick, 2008; Good et al., 2002; Lao et al., 2004; Misra, Fan, & Davatzikos, 2009).

A straightforward voxel-based was introduced by Kloppel et al. (2008). In their research, the tissue density of grey matter was directly extracted as a feature. The researcher adopted support vector machine as classifier and first ran the test using feature extracted from whole brain MRI. Ninety-six percent of the verified Alzheimer's subjects were correctly discriminated. They also introduced extended version which they train the classifier only with the features from frontotemporal lobe image acquired by one equipment, and later tested this classifier with data collected from another scanner. The results showed a discrimination of 89% correctness.

Fan, Shen, Gur, Gur, and Davatzikos (2007) had introduced another representative voxel-based algorithm called COMPARE. In their work, the brain MRIs were segmented into differentiated regions and the total voxel values were extracted as feature representations. These regions were segmented according to the density map of tissue class in the brain (grey matter, white matter and cerebrospinal fluid) and the regions that highly correlated with classification were discriminated. The selected feature representations were fed to a support-vector-machine-based classifier and tested using a leave-one-out cross-validation strategy. The test results

showed that their approach achieved a 91.8% accuracy for females and 90.8% accuracy for males.

The third category is based on cortical thickness (Desikan et al., 2009; Lerch et al., 2005). The features are usually the value of thickness at designated surface, which represent the atrophy of brain. Desikan et al. (2009) conducted an experiment on 313 subjects from two separate populations and their volume and mean of thickness were measured. The cortical thickness, the hippocampal volume and the gyrus thickness were combined to form a feature space for classification. The test in first group had a specificity of 94% and a sensitivity of 74%. The test in second group had a specificity of 91% and a sensitivity of 90% when discriminating mild cognitive impairment subject. The results showed that this proposed cortex-based approach is a cost-effective and efficient method in prodromal Alzheimer's diagnosis.

2.1.3.2 Deep Learning

The concept of artificial neural networks has been popular through 1980s but then gradually replaced by statistical based approaches like support vector machine. But recently with the explosion of massive data, the drawback of SVM has been revealed. When facing high-dimensional data, it is very difficult to create kernels to separate inputs. Hinton and Salakhutdinov (2006) proposed a neural network based learning method which is later known as deep learning. In their research, the high-dimensional inputs could be converted to low-dimensional feature representation through multilayer training. Followed by fine-tuning weights using gradient descent, this stacked autoencoder could reconstruct the input vector and was claimed to be more efficient and robust than principal components analysis.

Given this observation, researchers have used deep learning based methodologies to further explore the possibility of automated AD diagnosis and have reached substantial results. These work could be further grouped into three categories based on their applied modality.

2.1.3.2.1 Structural Information

Structural information used in deep learning aided diagnosis is mostly to be structural Magnetic Resonance Imaging (sMRI). Researchers found that AD process will cause damage to synapses, axons and perikaryon (Serrano-Pozo, Frosch, Masliah, & Hyman, 2011) and is more likely to be severe in temporal gyri, hippocampus and precuneus (Baron et al., 2001; Busatto et al., 2003). Structural brain imaging contains typical anatomical and morphological brain features such as ventricle size, hippocampus shape, cortical thickness and brain volume. Thus sMRI is often used in automated diagnosis research (Suk, Lee, & Shen, 2017).

Some early success have been achieved using Stacked Auto-Encoder (SAE). SAE is mostly used as feature extraction given it can be effectively used as input reconstruction (Supratak, Li, & Guo, 2014; Vincent, Larochelle, Lajoie, Bengio, & Manzagol, 2010; Xing, Ma, & Yang, 2016). B. Shi, Chen, Zhang, Smith, and Liu (2017) used a multimodel SAE to perform feature extraction and feature fusion. The features, which learnt from different Grey Matter (GM) and Deformation Magnitude (DM) patches, includes many nondiscriminative components and therefore were further transformed or so-called denoised with a SAE to obtain the final feature representations.

Other than SAE based frameworks, deep convolutional neural network (CNN; Krizhevsky, Sutskever, & Hinton, 2012) models are also very popular, given the desire of building deeper architecture. It is wide believed that deeper models have better ability to capture

latent feature representation. Given that SAEs layer-wised fully connected structure, training deep SAE would result in huge amount of network parameters as opposed to CNN architecture.

Billones, Demetria, Hostallero, and Naval (2016) modified VGG architecture into a 16-layered 2D convolutional neural network to perform three-way classification (AD/MCI/NC). Each of the 20 slices in the center of sMRI scan were input to the network separately. Aderghal, Benois-Pineau, Afdel, and Gwenalle (2017) proposed to train three 2D-CNNs separately on three different projections of the hippocampus: Sagittal, Coronal and Axial. Their outputs were later fused in a fully connected layer to generate the final output score. Ortiz-Suarez et al

Ortiz-Surez, Ramos-Polln, and Romero (2017) also used the projections of brain and 2D-CNN to differentiate which ROI is more discriminative for AD diagnosis. The authors concluded that the frontal pole region demonstrate the greatest discriminative power and thus provide validity in using CNN for AD diagnosis. Hosseini-Asl et al. (2016) used a deep 3D convolutional neural network to extract the morphological feature of subjects structural MRI and the classification was done using 3D adaptable Convolutional Neural Network. This classifier applied Net2Net initialization which accelerate update training when having a pretrained different model by avoids the brief period of low performance exists in methods that initialize some layers of a deeper network from a trained network and others randomly. This implementation makes it adaptable to different data size and increase the ability to generalize learnt feature. Li et al. (2017) created a multimode scheme using two different feature extracting method: multiscale convolutional autoencoder and a 3D convolutional neural network.

Those features were later concatenated and used as the input of upper-fully connected Multi-Layer Perceptron for classification. Payan and Montana (2015) used a two-stage approach which used sparse autoencoder initially to learn filters for convolution operations, whose result

was later used as the first layer of the convolutional neural network. Specifically, a 3-layer sparse autoencoder was adopted to extract features from images. For all the basis of the learned sparse encoder, the authors used the set of learned weights of that basis as a 3D filter of a 3D convolution. The feature maps obtained from the convolution layer were fed into a max-pooling layer to reduce the size of the feature maps. Other work has reported similar result (Karasawa, Liu, & Ohwada, 2018).

These mentioned studies used structural neuroimaging as classification evidence and have proven achieving great success. As we can observe, there is a clear favor in convolutional neural network architecture in these studies and the trend of migrating from 2D to 3D topology. This is largely due to the believing of 3D predictive models could cope with the dimensionality of used evidence and thus be capable of extract higher-leveled feature representation. By doing so, voxel-based features, which are invisible in 2D images, now become available for training. A detail studies comparison is shown in the Table 2.3.

2.1.3.2.2 Functional Information

As opposed to the structural information, functional information reveals the functional activities in the brain and the connectivity between brain regions. As mentioned earlier, the AD introduces great changes in the cognitive functions and therefore thought to be reflected on the functional measurements. These studies majorly used functional neuroimaging whilst some adoption of EEG recordings.

Hu, Ju, Shen, Zhou, and Li (2016) divided subjects fMRI into 90 regions of interest and the correlation between regions were obtained using Pearsons correlation coefficient, thus forming a correlation matrix which was treated as features. The network was later optimized using Limited BroydenFletcherGoldfarbShanno algorithm (L-BFGS). Morabito et al. (2016)

used EEG to reveal the functional information of brain. To cope with nature of EEG data, the classification was done using multichannel CNN which is able to handle multivariate time-series data. The EEG recording was decomposed into 5s epochs for each channel, and the time-frequency representation was computed in each channel using continues wavelet transform. Sarraf and Tofighi (2016) achieved the highest reported accuracy with functional data of 96.85% over cross-validation using a modified LeNet-5 model (Lecun, Bottou, Bengio, & Haffner, 1998). Cheng and Liu (2017) combined CNN and Recurrent Neural Network (RNN) to capture both intra and inter slice features in batches of PET slices. Suk, Wee, Lee, and Shen (2016) proposed a methodological architecture that combines deep learning and state-space modelling, and apply it to resting-state fMRI based Mild Cognitive Impairment (MCI) diagnosis. In this study, the author used two independent datasets, a public ADNI2 dataset and in-house dataset. The mean time series of ROIs were extracted from the preprocessed images. Then the mean intensities of ROIs in a volume at one time point were used as inputs to a deep Auto-Encoder (DAE). This is to discover the nonlinear association among ROIs in an unsupervised and hierarchical manner. To learn these encoded time series of ROIs, the authors trained two state-space model with hidden Markov model for functional dynamics in resting-state fMRI of NC and MCI. The performance comparison was conducted on the two mentioned dataset and between the proposed method and three other methods and the baseline. In both of the datasets, the proposed method achieved the best diagnostic accuracy (72.58% in ADNI2, 81.08% in in-house). This could be because of this proposed method considered the potential functional dynamics inherent in the rs-fMRI.

In general, researchers rarely used functional information solely in their work and also the achieved performance is comparably worse than the works using structural data. This could

be because the resting state functional neuroimaging, in fact, could not reflect the entire functional connectives in the brain. Strong intra/inter subjects variation is repeatedly reported by researchers (Adelstein et al., 2011; Heuvel, Stam, Kahn, & Pol, 2009; Honey et al., 2009; Meindl et al., 2010; Song et al., 2008; Wei et al., 2011). Also these proposed frameworks usually have much smaller data size compared with structural studies. Summarized result in shown in the Table 2.4.

2.1.3.2.3 Multimodality Information

As mentioned earlier, both structural and functional information oriented methods have certain validity in performing accurate diagnosis. Given this circumstances, frameworks adopted multimodality features are now thriving in the research community, in the scope of finding better feature representations (Bhatkoti & Paul, 2016; Feng et al., 2018; Liu, Cheng, Wang, Wang, & the Alzheimers Disease Neuroimaging Initiative, 2018; Suk et al., 2017; Suk, Lee, & Shen, 2014;). Ortiz, Munilla, Gorriz, and Ramirez (2016) preselected the discriminative regions among those defined by Automated Anatomical Labeling (AAL) Atlas (Tzourio-Mazoyer et al., 2002) from both MRI and PET imaging, and applied them to a Deep Belief Network (DBN). J. Shi, Zheng, Li, Zhang, and Ying (2018) investigated the possibility of solving this task by adopting stacked deep polynomial network that trained on MRI and PET.

Additionally, clinical data such as mental assessments were considered valuable and thus were included in studies. Li et al. (2015) used multimodality data of structural MRI and PET. Initially, the researchers used principle component analysis to extract PCs from ROI and biomarkers as features. Then they used stability selection techniques with the least absolute shrinkage for a further selection. These neuroimaging features were later combined with subjects' mental assessments as the input to a stack of Restricted Boltzmann machine with

dropout mechanism. Suk and Shen (2015) performed 3 two-way binary classifications (i.e., AD vs HC, MCI vs HC, and pMCI vs sMCI) using stacked auto-encoder, multikernel support vector machine and deep Boltzmann machine (DBM). Specifically, they used baseline MRI, 18-fluoro-deoxygllucose PET and cerebral spinal fluid data acquired from ADNI dataset. Other than the neuroimaging data, two clinical scores, mini-mental state examination (MMSE) and Alzheimer's disease Assessment Scale-Cognitive subscale (ADAS-Cog), were also included as the low-level features. The three modalities were used on three separate SAE models to extract the high-level feature representations. These representations were later concatenated with the original low-level features and thus constructed an augmented feature vector. A deep Boltzmann machine was adopted to remedy the drawback of the ROI-based method which is failing to handle subtle changes in an ROI or across ROIs. Moreover, because simple concatenation of the features of multiple modalities in a shallow architecture can cause strong connections among the variables of an individual modality and failed to find intermodality relations, the authors devised a discriminative multimodal DBM which yield to better result. Khvostikov, Aderghal, Krylov, Catheline, and Benois-Pineau (2018) proposed a modified 3D version of inception-based CNN using both sMRI and Mean Diffusivity (MD)-DTI. Both sMRI and MD-DTI have been segmented into different region of interests (ROIs) and each ROI was considered as the input of a pipeline of inception modules. The output features were eventually processed by a 3D average-pooling layer as oppose to the conventional fully connected layer, then concatenated to produce the classification results.

Generally, studies that using multimodality information outperformed the ones with single modality, especially for studies that added clinical information. Intuitively, modalities that reveal either functional or structural information of the brain, when jointly analyzed together,

should provide the classification with strengthened evidence and latent intramodalities features. Thus, it would make up the oversight of pathological changes that only partially captured by single modality. Summarized works that used multimodal information was shown in Table 2.5.

2.1.4 Summary

In this section, I have introduced some examples of deep learning based frameworks in AD automated diagnosis. Recently, there is clear shift in the research community from shallow machine learning toward deep learning due to the following reasons:

1. The complex nature of evidence

Restricted by our limited understanding of AD, feature engineering is far more difficult this particular research area compared with other fields of application. Considering the limited standardization of variated brain structure, hand-engineered features suffer from insufficient feature representation. Deep learning on the other hand, could automatically learn end-to-end feature without too much preprocessing.

2. The supremacy of algorithm

Deep learning has been proven to outperform previous existing algorithm in other data-driven field of research. Ideally, we believe that deeper architecture has better capability to capture latent features, assuming the architecture topology and hyper-parameter are carefully optimized. Moreover, advanced architectures and computation power have further enabled the exploration of deep learning based method.

However, there are still several existing concerns and pitfalls, despite the reported excellent performance:

1. Classification comparison and evaluation metrics

The reviewed works are purposed to provide an overview of the framework and not for comparison. In fact, authors stated that the proposed research outperform existing ones for having only higher accuracies, for most of the time. Others may include sensitivity and specificity in their evaluation metrics. However, given that most of the studies have very limited and imbalanced datasets, an area under the curve (AUC) for receiver operating characteristic (ROC) or F1- score (harmonic mean of precision and recall) should also be reported for a more intuitionistic result.

2. Sample size and generalization

Compared to other data-driven research, the available data for AD research are very limited and imbalanced. The major drawback is that one can argue that the population characteristic could not be reflected from reached conclusion.

This is due to not being able to capture enough variation in the population from questionable dataset and thus lack of ability for data generalization.

Additionally, another aspect of data generalization is using decentralized data. Currently, the majority of the researchers used pre-existing dataset in their work. However, whether their work can be extended and operates on data collected from different machines and protocols is untested.

3. Classification of disease subtypes

Most of the reviewed work have reported single or more binary classification (AD/HC, AD/MCI, MCI/HC). These results seemed promising but lack of real-world implications. Instead of trying distinguish patients from healthy normal, researches should focus more on the method to differentiate alternative diseases or progressiveness of disease at various stages. This resulting

in the need of performing multiclass classification, and subtype classifications (AD,sMCI, pMCI).

4. Further clinical data

In the reviewed work we have reviewed several frameworks that included clinical testing data as part of the prediction evidence. These works averagely showed superior results compared with neuroimaging-only works. This brought the attention of adding more clinical data into the process of classification. This may include but not limited to genome data, behavioral data and family history data.

5. Model optimization and overfitting

As come to the model architecture, despite the recent huge success achieved in basically all data-driven fields of research, deep learning is known for being a black-box system. Its decision-making process cannot be intuitively explained, used separately, or back-traced to original input. This nature of lacking transparency has made model optimization difficult and uninformative. Currently there is no systematic guideline for model design, optimization or hyper-parameter tuning.

One other challenge is overfitting. While overfitting is common for machine learning models, the high dimensionality of medical neuroimaging and limited dataset has made overfitting incredibly crucial for this particular field of research. So far, standard strategies like regularization and dimension reduction are usually applied while more sophisticated methods need to be investigated.

In the reviewed work, we have introduced an aspect of deep learning methods that were adopted in the research of Alzheimer's disease diagnosis. These works both achieved promising result and raised further questions. As mentioned in the previous section, couple of the key

challenges need to be taken with full consideration. Meanwhile, scientists should also focus on the caregiving application and techniques for AD related personals.

2.2 Assistive Technology for Alzheimer's Disease Patients

Recently, the development of smart accessories has caught the attention of both research and industries. By actively connecting your devices to a smart IoT system, home living can now be a lot more convenient and efficient than ever before. Especially, assistive living system for cognitively impaired elders can significantly benefit from such application.

Among other aspects, the ability to automatically identify the indoor room is a very important feature of the assistive living system for Alzheimer patients. By accurately identifying the patient's whereabouts, the smart home system could effectively estimate their potential action and provide help accordingly, or send alert to the caregivers in an emergency situation.

2.2.1 Assistive Technology

Lately, the prosperity of machine learning has advanced people's everyday living significantly from many directions. Technology realizations like smart home have been widely popular and implemented. Such system controls all the smart devices in the household through home automation to improve efficiency, security and convenience. User can easily setup the smart system through customized user preferences and monitor/supervise the operation from their control hub. Moreover, devices that connected together through IoT system, can actively share the information between each other and collaborate. For example, a passive motion detector could pick up the irregular movement and wake up the video surveillance camera to capture the appearance of this intruder. The system could then run facial recognition with known authorized personnel to determine if law enforcement needs to be alerted. Other than security,

another important feature of smart home system is to enhance people's quality of life. The standard usage includes but not limit to smart lighting control and thermal control. Smart lighting system can adjust the lighting condition in the household either based on customized preference, or adapt to the situation. For instance, a smart lighting system could automatically detect the amount of occupants in the room and adjust illuminance, or regulate lighting condition based on daylight illuminance. A more common application is smart thermostats. These thermostats could detect owner's activities based on action heat map and adjust temperature in each room respectively. They can also learn owner's activity pattern such as when do they leave for/ return from work, and thus adjust temperature accordingly.

One of the most significant advantage of smart home is to reduce owner's involvement in everyday life. A well-established smart home system will allow the owner to easily interact with home appliance without too much tending of the equipment. This feature has made such system capable of housekeeping, home security, and providing full home functionalities to person with less ability to maneuver home appliances such as seniors, children and disabilities.

2.2.2 Assistive Technology for Alzheimer's Disease Patients

The cognitively impaired seniors, due to their severely low capabilities of handling everyday tasks, are the ones that would benefit the most from smart home, or smart assistive system in this case, among all the aforementioned groups. The Alzheimer's disease (AD), the major form elder cognitive impairment, is an irreversible, progressive neurodegenerative brain disorder that slowly destroys patients' cognitive function and eventually, takes their lives. For the past 20 years, the reported death from AD has increased by 145% while other major disease showed significant decreases (Gaugler et al., 2019). It is reported that there are 5.9 million adults aged 65 and more, with Alzheimer's disease and related dementia (ADRD) in North America,

and this number is estimated to be 13.9 million by the year 2060 (Gaugler et al., 2019). This translates into a new case every 33 seconds (Corrada et al., 2010; Hebert et al., 2001). Given its significance, the World Health Organization officially recognized AD as the most common form of dementia and possibly contributes to 60% - 70% of cases (World Health Organization, 2012).

Alzheimer's disease has massive impacts on the patients, families, both physically, and financially. Aside from the disease itself, caregiving is an equally concerned aspect. It is reported that, in 2018, more than 18.5 billion hours of informal care were provided by the caregivers (with a worth of $234 billion). Among all the caregivers, 83% of them are unpaid caregivers like family members (Gaugler et al., 2019) and caregiving was a huge burden for them. Surprisingly, research showed that the burden on the caregivers was largely contributed by the emotional stress (Sales et al., 2016) rather than the physical stress. In the early stages, caregivers tend to consistently worry about the patients' capability of individual living and struggled about how to balance between their own life and providing care.

To that end, one common solution is to provide the patients with assistive technology that can improve their quality of life, and enhance their ability of independent living. These technologies include, but not limited to video surveillance, assistive phone system, or geo-fence. However such solutions usually need continuous attention from the caregivers and therefore would not help with relieving the aforementioned stress on the caregivers.

2.2.3 Related Work

Given these circumstances, automated event detection for patients with cognitive impairment is worth the investigation and research. Research in this domain involves many aspects including wandering detection (Kim et al., 2009; Lin et al., 2012; Vuong et al., 2011), fall detection (Ko et al., 2014; Wang et al., 2017) and indoor scene understanding (Gupta et al.,

2015; Khan et al., 2016; Zhu et al., 2016). Among other aspects, scene understanding based on pervasive information is very useful as it could utilize the ubiquitous information instead of requiring carefully deployed sensors or knowledge about room layout. Identification of the patient's location or the surrounding environment can assist the patient in making a better decision. Many research have been done in this field. Typically, the identification of the scene was done by classification based on various evidence. Imaging for example, is proven to be very efficient since it is the same way human use to identify their whereabouts. Classification though imaging and computer vision can be categorized as learning detailed features that are significant to their respective classes. One of the early successes was achieved by Mozos et al. (2005), where the researchers used Adaboost (Freund & Schapire, 1997) to classify different laser range scans into their respective semantic categories. Specifically, each data observation contained a set of beams from a 360° field of view range sensor. These raw beams and the area covered by these beam were obtained as raw data. To calculate the features, a set of nine simple geometric features were captured from the raw beams of the training samples. Furthermore, another 13 features were calculated from the area covered by the beams in each training sample. These features were combined together as the final feature representation. Six binary classifications were then conducted using this representation between three indoor class – rooms, corridors, and doorways. The classification accuracy ranged from 80.10% (for corridor – doorway) to 93.94% (for room – doorway). To expand their method from binary classification to multiclass classification, a sequence of proposed Adaboost classifiers were arranged to form a decision list for the final outcome. For the three provided multiclass test scenarios, the multi-Adaboost model achieved 89.52%, 92.10%, and 93.94% on the classification accuracies.

Later, the authors further expanded their research to using multimodel data (i.e., imaging and range data; Mozos et al., 2007; Rottmann et al., 2005). In these subsequent studies, geometric features and imaging features were both adopted to find better representation. In short, 321 geometric features were calculated using range sensor data in the same way that described previously. For imaging, a panoramic view, which was consisted of a set of eight images for each data observation, was captured using cameras. From there, eight carefully selected common objects in these scenes were targeted to find their descriptive features. These features represented the evidential presence of such object in the scene and therefore used as supplementary feature for the overall classification task. Additionally, the authors incorporated a hidden Markov model (HMM) in their classifier. This was due to the desire of not only predict the room type based on the current data, but the previous data as well. The idea was that certain room type transition would be less possible (e.g., from kitchen to office directly). Specifically, the probable class of current location was estimated by the previous predicted room class and the transition probabilities between rooms. Such probabilities were estimated by running numerous simulations. The author summarized that the room types between two time stamps were most likely to remain the same. The next highest probable transition was from a room to doorway while the least likely transition was from one room to another room. This meant the person needed to get to the doorway first if he wanted to switch room. As a result, the authors achieved 75.4% without HMM and 91.2% with HMM.

Related works also used Adaboost only on imaging data (Ayers & Boutell, 2007). In their work, the Scale – Invariant Feature Transform (SIFT; Lowe, 1999) was applied to the data to extract key points as features and achieved average accuracy of 77% on binary classification between seven classes.

These early attempts (Ayers & Boutell, 2007; Mozos et al., 2005; Mozos et al., 2007; Rottmann et al., 2005) on the indoor scene type classification showed promising efforts and valuable ideas on the task. However, also they showed some serious drawbacks. First of all, all the features were created very carefully with specific and complicated requirements on the hardware setup. A very sophisticated system could neither be necessary, nor be feasible to deploy on other related task. Secondly, the calculated features were very specific to the test environments. The authors mentioned the classification accuracy dropped from 91.83% to 82.23% when moved to a different building (Mozos et al., 2005).

For the past 10 years, convolutional neural network (CNN) inspired models have shown tremendous success in many research fields especially in imaging based studies (Rawat & Wang, 2017). A properly tuned deep CNN model could effectively learn features from raw input without carefully performing feature engineering. Another key advantage is that convolutional layer, compared with traditional fully connected layer, is computationally less intensive. Ursic et al. (2016) developed a hybrid - CNN to perform classification on part-based images. The dataset used in this work is the MIT Indoor67 dataset (Quattoni & Torralba, 2009). The authors first extracted image regions using a gestalt-principle-inspired selective search (Uijlings et al., 2013). Such method was capable of preforming more robust and accurate generation of discriminative regions. Thus, images were transformed into collections of unordered segmented image regions. The model was then trained to learn the discriminative regions that were significant to the target classes. The same model was also tested on the modified dataset where images were distorted to create noises. This method achieved 85.16% accuracy on the original images and average 72.58% accuracy on all distorted datasets. Zhou et al. (2014) explore the classification performance between scene-centric database and object-centric database. The proposed hybrid-

CNN model achieved 70.80% on MIT Indoor67 and 91.59% on 15-Scene (Fei-Fei & Perona, 2005; Lazebnik et al., 2006; Oliva & Torralba, 2001).

The accuracies of these reported neural networks have ranged from 70 to 90. This supports the common challenge of recognizing natural scene with high accuracies all the time. In other word, natural scene recognition is a complex task. Thus, recent research effort has been made on developing intelligent model/techniques for recognition of natural scenes with high accuracies. This process can be very case specific. Therefore, one aspect of this research is to use a large rang of natural scene.

2.3 Background Information

In this section, I would briefly introduce the common technique and components. Specifically, this review would focus on the information related to the deep neural network. This would serve as the background information for the latter chapter, which would not further elaborate on the details.

2.3.1 Network Layers

Neural networks are mostly consisted of several or more layers. An early example, the "perceptron," was developed by Rosenblatt (1961). Such model only contained two layers – one input layer and one output layer, as shown in Figure 2.1. Inspired by this work, some more complex models were introduced, such as "Multilayer Perceptron" (Rumelhart, Hinton, & Williams, 1985). The simplest multilayer perceptron, compared with the original perceptron, has one additional hidden layer between the input and the output layer. Depending on the application and training methods, more hidden layers can be added as well. An example of MLP with two hidden layers was shown in Figure 2.2. The total amount of layers has become bigger as the

technology advances. Network nowadays can have thousands of layers of different types. In this section, all the neural network layers that were adopted in this research were introduced.

2.3.1.1 Fully Connected Layer

The most fundamental layer is the fully connected layer. As the name suggested, each neurons in the FC layer connects with all the neuro in the previous layer. Each of these neurons is an instance of the perceptron. In short, each neuron receives information from all the neurons from previous layer in an unordered manner. As shown in Figure 2.2, the two hidden layers in the middle were the examples of fully connected layer. The first FC layer has 12 neurons and the second FC layer has eight neurons. The total amount of parameters in these two layers is $9 \times 12 + 12 + 12 \times 8 + 8 = 224$.

2.3.1.2 Convolutional Layer

In imaging related classification task, convolutional layers were massively favored for feature extraction. FC layer on the other hand, has several drawbacks when dealing with image-based classification for three reasons. One is that the computation burden is very huge. Imagine an input grayscale image with size of $m \times n$, connects to a hidden layer with i hidden neurons and then finally connects to an output layer with j output nodes. This simple structure contains a total number of $m \times n \times i \times j$ parameters that need to be calculated. Secondly, this structure suggested the model treats the input as a one-dimensional vector which would ignore the patterns in the surrounding neighborhood. Finally, this structure needs to see the whole image input before producing an output. However, in most of the cases, one could and should be able to identify the input by its fragments. Therefore, convolutional layer was introduced to that end (LeCun et al., 1998). Following, several variations of convolutional layers were explained.

1. 2D Convolutional Layer

A 2D convolutional layer convolves in both axes in a 2D plane with given filter size. Convolution in imaging-related tasks often relates to extracting features such as edge detection. The result is usually referred to as a *feature map*. When more convolutional layers are applied, the embedded information of the original images gets transformed from raw information like pixel intensity value, to some intermediate features such as edge, shape (lower level), and eventually into high-level features (deeper level). See Figures 2.3 and 2.4 for 2D convolution demonstration. Theoretically, to calculate the 2D convolution y[m,n] for a given 2D signal x[m,n] using 2D filter h[m,n] (Nielsen, 2015):

$$y[m,n] = x\,[m,n] * h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j] \qquad (1)$$

In reality, the filter would slide from the top left corner of the image, all way to the bottom right corner. If the step size, or stride, of this sliding operation is one, then this operation would result in a feature map that is smaller than the original input. However, the underlying problem is that the feature for all the pixels on the edge will not be calculated. By stacking convolutional layers like this, we will end up with a model that will keep losing edge features. A typical solution to this problem (Figure 2.3), requires the filter to calculate the pixel values that are out of the image boundary. To do so, we will pad zeros to the perimeter of the original image to increase its size. In summary, the following equation 2 can be used to calculate the output feature map size O, given W as the input dimension on given axis, F as the filter size, P as the padded amount and S as the stride (Nielson, 2015).

$$O = \frac{W - F + (2 \times P)}{S} + 1 \qquad (2)$$

Here (Figure 2.3), the input image size is 5 × 5 with filter size of 3 × 3. After zero padding, the input image is now 7 × 7, which would bring the output size to 5 × 5 ($O = \frac{5-3+(2\times1)}{1} + 1 = 5$).

By performing 2D convolution, not only the feature from the pixel itself, but also the spatial features from its surround neighborhood, are all captured by the filter. Moreover, a neuron with a filter size of m × n, only needs to update its own weights and a bias. This would bring the total number of parameters that need to be trained to (m × n) + 1, which is much less than aforementioned Multilayer Perceptron.

2. 3D Convolutional Layer

When target task evolved from 2D to 3D domain, naturally the filters need to be expand to 3D domain as well. As shown in Figure 2.5, a 3D convolution is the generalized version of 2D convolution. In such operation, the 3D filter performs similar operation as the 2D filter. Except the filter would move at three dimensions and convolves around three axes instead of two.

2.3.1.3 Pooling and Flatten

Pooling layers are used to downsample feature map to reduce computation burden and avoid overfitting. Also, in computer vision, pooling layers could effectively reduce the size of receptive field (Nielson, 2015). Receptive field was defined as “a portion of sensory space that can elicit neuronal responses when stimulated” (Alonso & Chen, 2009, para. 2). Cognitive theorists suggest that, features in bigger receptive fields tends to be low-level features. As the receptive field get smaller, the level of extracted features become more sophisticated. This way, the model could imitate how human vision system extract feature and thus produce better feature representation. Specifically, pooling layer works similarly as the convolutional layer, also perform sliding window operation on the feature map and output the maximum/average value in

the window. However, the stride of pooling layer is the same as the filter size to effectively downsample the feature map. A special type of pooling layer, the global average pooling layer is often applied as the last layer in the feature extraction model. The difference between a global pooling layer and a standard pooling layer is, the global pooling layer has a filter with its size same as the feature map. This way, the output size of each feature map is 1×1. An example of the average pooling, the max pooling, and the global max pooling was shown in Figure 2.6.

Flatten layer usually follows the last pooling layer in the feature extraction model. The purpose of the flatten layer is to provide a feature representation for the classifier. An example of flatten layer was shown in Figure 2.7.

2.3.1.4 Dropout

The dropout layer was introduced by Srivastava et al. (2014) for the purpose of reducing overfitting. Overfitting is mostly due to model learnt unwanted statistical noise during training. Thus fail to perform adequately when evaluating new data. When connecting the output feature maps to the dropout layer, as shown in Figure 2.8, it would randomly reset selective amount of feature maps to zero, also called deactivate a selection of nodes/neurons, according to the preset drop rate *p*. Then these neurons would be reactivated during testing but suffers a penalty of p on activation. In a way, dropout forces the model to learn different combinations of features parallelly, and later average these models during testing.

2.3.2 Activation

As the name suggested, the artificial neural network was inspired by the biological form of human brain neurons. So similarly, the artificial neurons can also be fired up if given the correct input. And activation function is what we used to quantify the rate of action. Given the

neuron describe in Figure 2.1, the output of the neuron is calculated by equation (3), where x_i are the inputs from previous layer, w_i are the corresponding weights to x_i and b is the bias of the neuron:

$$y = \sum_i w_i x_i + b \tag{3}$$

The activation function is then applied to the output of the neuron. Typically, there are four major types of activation function: Sigmoid, Hyperbolic Tangent (tanh), Rectified Linear Unit (ReLU), and Soft Argument of the Maxima (softmax).

2.3.2.1 Sigmoid and Tanh

The sigmoid function was described in equation (4), where z denotes the input:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

The hyperbolic tangent function was described in equation (5), where z denotes the input:

$$h(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{5}$$

Detailed plots of the sigmoid function and the tanh function were shown in Figure 2.9 and 2.10. In short, the sigmoid function maps the inputs to outputs ranged from 0 to 1, while the tanh produces the outputs to [-1, 1]. Both functions symbolize the activation of an artificial neuron. The difference is tanh has slightly stronger gradient than sigmoid (Nielson, 2015).

Sigmoid and tanh functions have been widely used in the early machine learning works but quickly has been abandoned in the deep learning regime. This is due to their glaring disadvantage, which is the “vanishing gradient” problem. It is well-known that neural networks are trained using gradient based methods. Therefore, whether the gradient can be effectively

propagated back to the layers in the network is very important. According to the equations(4) and (5), their derivative were describe in equation (6) and (7):

$$g'(z) = g(z)\big(1 - g(z)\big) \quad (6)$$

$$h'(z) = 1 - h^2(z) \quad (7)$$

It is clear that when the input of activation function is strong positive or strong negative, the gradient of the function tends to approach 0, which would cause the network not being able to update parameters. This "vanishing gradient" problem is exceptionally destructive in deep learning, as there are much more layers than the conventional neural network.

2.3.2.2 ReLU and Leaky ReLU

To effectively train deep network, the Rectified Liner Unit (ReLU) was introduced by Nair and Hinton (2010). ReLU is rectified for input below zero and produce identical output when input is greater than zero. The diagram of ReLU layer was shown in Figure 2.11. Its mathematic equation can be interpreted as (Nair & Hinton, 2010):

$$R = \max(0, input) \quad (8)$$

Compared with sigmoid-like function, ReLU has several advantages. First of all, as shown in the right half of the plane in Figure 2.11, when input is greater than zero, the gradient of the function is constant 1. This solves the vanishing gradient problem. Second of all, as shown in the left half of the plane, when input is less than zero, the gradient of the function is rectified to zero. This results a very sparse feature representation, because only the positive inputs will contribute to the representation. On the contrary, sigmoid-like function will calculate a feature for every input, leaving a heavily dense representation. Moreover, the computation burden of ReLU is trivial compared with sigmoid function, given no costly operations are required like exponential function. In summary, ReLU activation is more suitable for deep learning model and

was proven to have much better performance and convergence speed (Krizhevsky, Sutskever, & Hinton, 2012).

Despite the astonishing success of ReLU in deep learning, it suffers from a dramatic drawback known as the "Dying ReLU." Basically, a neuron that was rectified by ReLU will remain in the "dead" situation and can no longer participate in training. A variation of the ReLU, known as the "Leaky ReLU" (Maas, Hannun, & Ng, 2013) was then created to resolve this issue. In this variation, as shown in Figure 2.12, a very tiny gradient was given to the negative section of the function. Mathematically, the Leaky ReLU function can be described as equation (9):

$$LR = \begin{cases} x, & if\ x > 0 \\ \alpha \cdot x, & if\ x \leq 0 \end{cases} \qquad (9)$$

Such design would allow the rectified neurons to have small gradients, that could bring them back to life, in long term of parameter updates.

2.3.2.3 Softmax

A softmax function, also known as softargmax function, is a mathematic function that converts a series of real numbers that indicates the likelihoods of each represented category, into normalized probabilities. The function was derived from argmax function, or arguments of the maxima. Such function would find the maxima among inputs and its corresponding index. Then the function would return an output vector with the same length as the input. In this vector, all the elements would be 0 except the element bears the previous found index, which would be 1. For example, for a given input vector [1, 0, 5, 4], the output would be [0, 0, 1, 0]. A softmax function is basically a smoothed arg max function. In short, it converts the input values to probabilities in the range of [0, 1] and all the probabilities would add up to 1. Softmax function

was extensively used in machine learning because it is capable of converting the outputs of the model to probability distribution over the classification task categories.

2.3.3 Regularization

Overfitting is the most common and significant problem when trying to train a machine learning model. Often times, restricted by either nature of data set or poor generalization, the model would try to converge to every data point in the training set and thus lose the ability to predict new, un-seen data. To avoid overfitting, the common methodology is to penalize the weights update in the training process, which is known as "Regularization." Here, two regularization methods that were used in this research are introduced.

2.3.3.1 L2 Regularization

The L2 Regularization provides a weight penalty to the update process. Such regularization, depending on the sum of the square of feature weights, applies a decay factor, or λ, to the penalty term. This penalty would force the weight decays towards zero. In some way, the less import features would thus become negligible and therefore are practically removed from the model. On the other hand, if the decay factor was selected too big, some significant features would also be penalized and therefore leads to an under-fitting. Currently, the selection of λ is done by conducting grid search.

2.3.3.2 Batch Normalization

The purpose of batch normalization is to reduce the covariate shift existing between layers. The covariate shift in artificial neural network refers to the fact that the input distribution of each layers changes drastically during training, causing each layer has to adjust the parameter greatly to adapt to this change (Ioffe & Szegedy, 2015). This situation is even more dire when

training deep network because the changes in distribution would enlarge greatly when propagate through layers. By doing batch normalization, the activation is effectively scaled to have zero mean and unit variance, which would allow the training to be more effective. Specifically, given a batch with m data points, the batch mean μ_B and batch variance σ^2B were calculated (Ioffe & Szegedy, 2015):

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{10}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{11}$$

Then input is normalized to:

$$\widehat{x_\iota} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{12}$$

Table 2.1 Common Symptoms of AD Patients in different stages

Early Stage	Middle Stage	Late Stage

The early stage is often overlooked. Relatives and friends (and sometimes professionals as well) see it as "old age," just a normal part of ageing process. Because the onset of the disease is gradual, it is difficult to be sure exactly when it begins.
• Become forgetful, especially regarding things that just happened
• May have some difficulty with communication, such as difficulty in finding words
• Become lost in familiar places
• Lose track of the time, including time of day, month, year, season
• Have difficulty making decisions and handling personal finances
• Have difficulty carrying out complex household tasks
• Mood and behavior:
  • may become less active and motivated and lose interest in activities and hobbies
  • may show mood changes, including depression or anxiety
  • may react unusually angrily or aggressively on occasion
	As the disease progresses, limitations become clearer and more restricting.
	• Become very forgetful, especially of recent events and people's names
	• Have difficulty comprehending time, date, place, and events; may become lost at home and in the community
	• Have increasing difficulty with communication (speech and comprehension)
	• Need help with personal care (e.g., toileting, washing, dressing)
	• Unable to successfully prepare food, cook, clean or shop
	• Unable to live alone safely without considerable support
	• Behavior changes may include wandering, repeated questioning, calling out, clinging, disturbed sleeping, hallucinations (seeing or hearing things which are not there)
	• May display inappropriate behavior in the home or in the community (e.g., disinhibition, aggression)
		The last stage is one of nearly total dependence and inactivity. Memory disturbances are very serious and the physical side of the disease becomes more obvious.
		• Usually unaware of time and place
		• Have difficulty understanding what is happening around them
		• Unable to recognize relatives, friends and familiar objects
		• Unable to eat without assistance, may have difficulty in swallowing
		• Increasing need for assisted self-care (bathing and toileting)
		• May have bladder and bowel incontinence
		• Change in mobility, may be unable to walk or be confined to a wheelchair or bed
		• Behavior changes, may escalate and include aggression towards carer, nonverbal agitation (kicking, hitting, screaming or moaning)
		• Unable to find their way around in the home

Table 2.2 Clinical Diagnostic Criteria for AD

Core Diagnostic Criteria	Supportive Features	Exclusion Criteria
A. Presence of an early and significant episodic memory impairment that includes the following features: 1. Gradual and progressive change in memory function reported by patients or informants over more than 6 months 2. Objective evidence of significantly impaired episodic memory on testing: this generally consists of recall deficit that does not improve significantly or does not normalize with cueing or recognition testing and after effective encoding of information has been previously controlled 3. The episodic memory impairment can be isolated or associated with other cognitive changes at the onset of AD or as AD advances	B. Presence of medial temporal lobe atrophy • Volume loss of hippocampi, entorhinal cortex, amygdala evidenced on MRI with qualitative ratings using visual scoring (referenced to well characterized population with age norms) or quantitative volumetry of regions of interest (referenced to well characterized population with age norms C. Abnormal cerebrospinal fluid biomarker • Low amyloid β1–42 concentrations, increased total tau concentrations, or increased phosphor-tau concentrations, or combinations of the three • Other well validated markers to be discovered in the future D. Specific pattern on functional neuroimaging with PET • Reduced glucose metabolism in bilateral temporal parietal regions • Other well validated ligands, including those that foreseeably will emerge such as Pittsburg compound B or FDDNP E. Proven AD autosomal dominant mutation in the immediate family	History • Sudden onset • Early occurrence of the following symptoms: gait disturbances, seizures, behavioral changes Clinical feature • Focal neurological features including hemiparesis, sensory loss, visual field deficits • Early extrapyramidal signs Other medical disorders severe enough to account for memory and related symptoms • Non-AD dementia • Major depression • Cerebrovascular disease • Toxic and metabolic abnormalities, all of which may require specific investigations • MRI FLAIR or T2 signal abnormalities in the medial temporal lobe that are consistent with infectious or vascular insults

Table 2.3 Related Work Using Structural Information

Author	Evaluation		Architecture	Sample Size	Dataset
B. Shi et al. (2017)	AD/HC	89.0	SAE	338	ADNI
	MCI/HC	81.7			
Billones et al. (2016)	AD/HC	98.33	CNN	900	ADNI
	AD/MCI	93.89			
	MCI/HC	91.67			
	AD/MCI/HC	91.85			
Aderghal et al. (2017)	AD/HC	91.41	CNN	815	ADNI
	AD/MCI	69.53			
	AD/MCI/HC	65.62			
Ortiz-Surez et al. (2017)	HC/Mild Dementia	87.5	CNN	86	OASIS
Hosseini-Asl, Keynto, and El-Baz (2016)	AD/HC	99.3	CNN/CAE	30+210	CADDementia; ADNI
	MCI/HC	94.2			
	AD/MCI	100			
	AD/MCI/HC	94.8			
Li, Cheng, and Liu (2017)	AD/NC	88.31	CNN/CAE	428	ADNI
Payan and Montana (2015)	AD/HC	95.4	CNN	2265	ADNI
	MCI/HC	92.1			
	AD/MCI	86.8			
	AD/MCI/HC	89.5			

Table 2.4 Related Works Using Functional Information

Author	Evaluation		Architecture	Sample Size	Dataset
Hu et al. (2016)	MCI/HC	87.58	SAE	100	ADNI
Morabito et al. (2016)	AD/HC	85	CNN	119	Self-Recruited
	MCI/HC	85			
	AD/MCI	78			
	AD/MCI/HC	82			
Sarraf and Tofighi (2016)	AD/HC	96.85	CNN	43	ADNI
Suk et al. (2016)	MCI/HC	72.6 81.1	SAE	62+37	ADNI Self-Recruited
Cheng and Liu (2017)	AD/HC	91.19	CNN+RNN	339	ADNI
	MCI/HC	78.86			

Table 2.5 Related Works Using Multimodal Information

Author	Evaluation		Architecture	Sample Size	Dataset
Li et al. (2015)	AD/HC	91.4	RBM	202	ADNI
	MCI/HC	77.4			
	AD/MCI	70.1			
	MCI.Convert/MCI.Stable	57.4			
Ortiz et al. (2016)	AD/HC	90	DBN	818	ADNI
	MCI.Stable/HC	80			
	MCI.Convert/HC	83			
	AD/MCI.Stable	84			
	MCI.Stable/MCI.Convert	78			
J. Shi et al. (2018)	AD/HC	97.13	DPN	202	ADNI
	MCI/HC	87.24			
	MCI.Stable/MCI.Convert	78.88			
	AD/HC/MCI.Stable/MCI.Convert	57.00			
Suk and Shen (2015)	MCI/HC	88.8	SAE+DBM	202	ADNI
	MCI.Stable/MCI.Convert	77.9			
Khvostikov et al. (2018)	AD/HC	93.3	CNN	531	ADNI
	AD/MCI	86.7			
	MCI/HC	73.3			
	AD/MCI/HC	68.9			

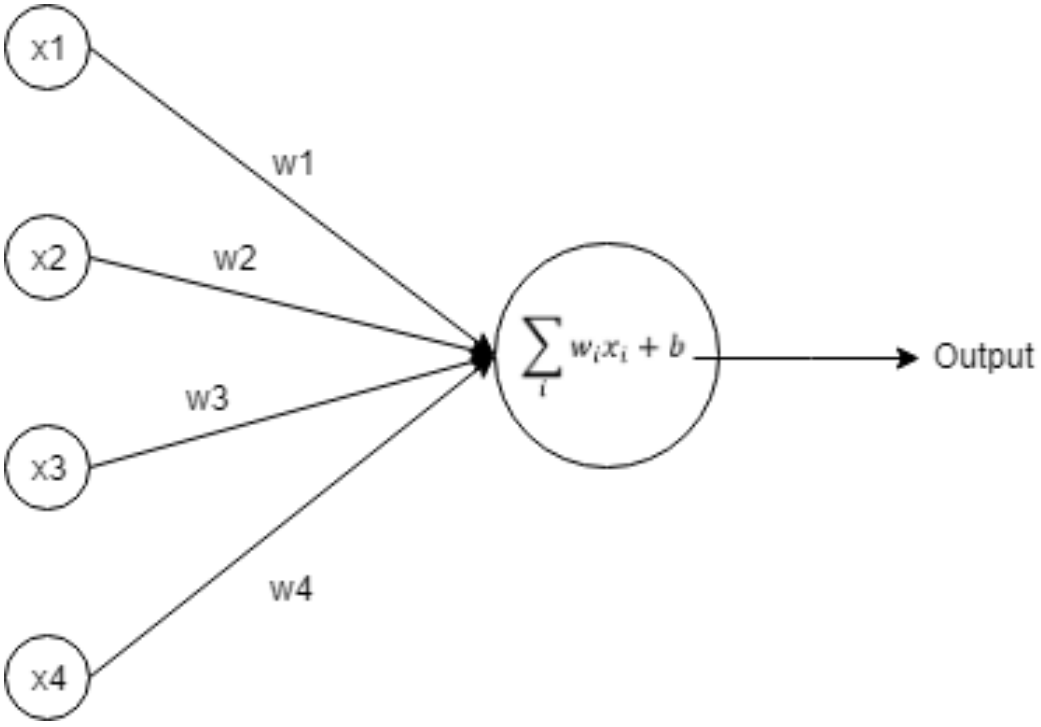


Figure 2.1 Illustration of perceptron.

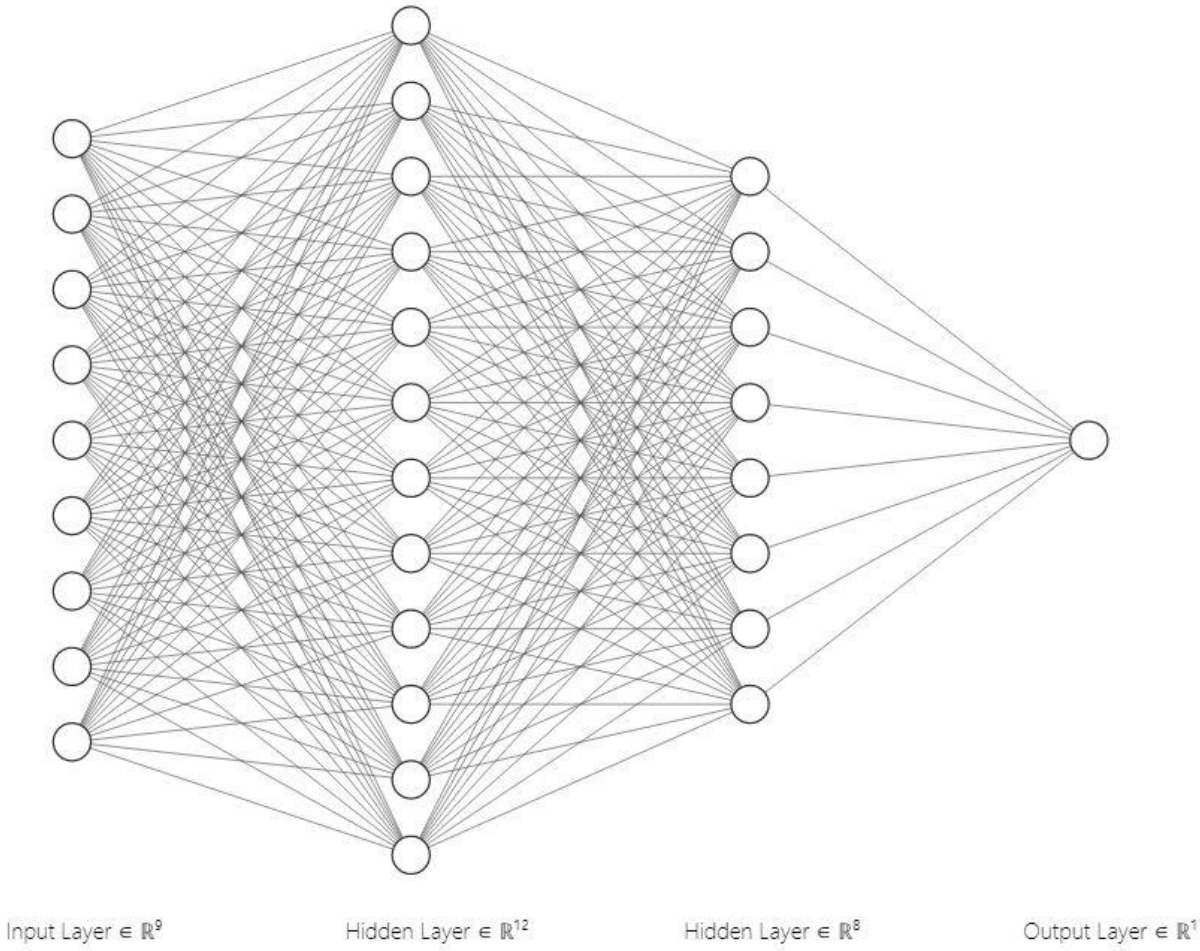


Figure 2.2 Illustration of multilayer perceptron.

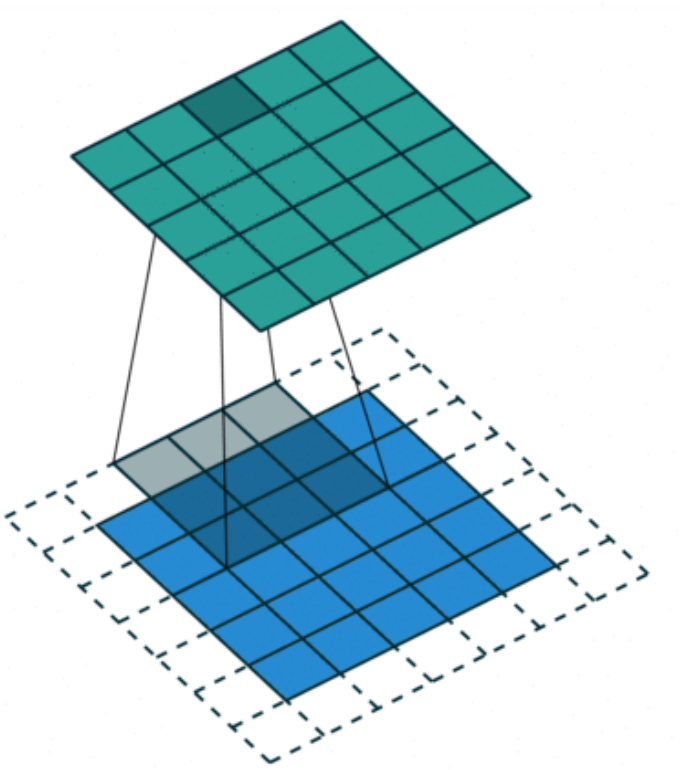


Figure 2.3 Window operation of 2D convolution.

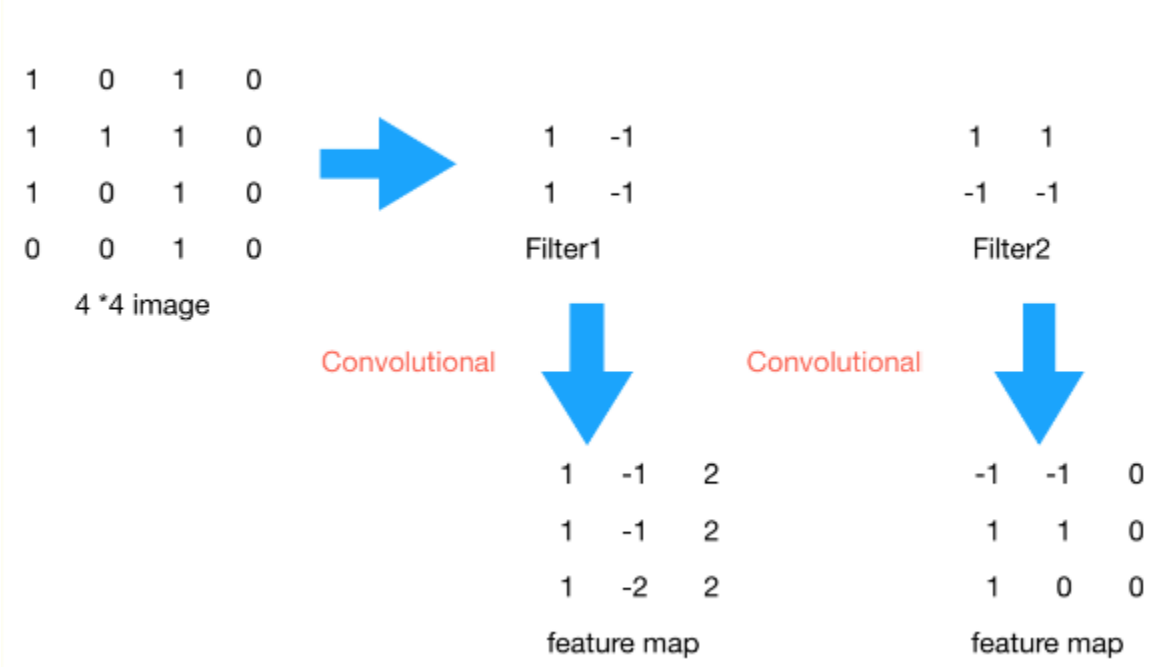


Figure 2.4 2D convolution computation.

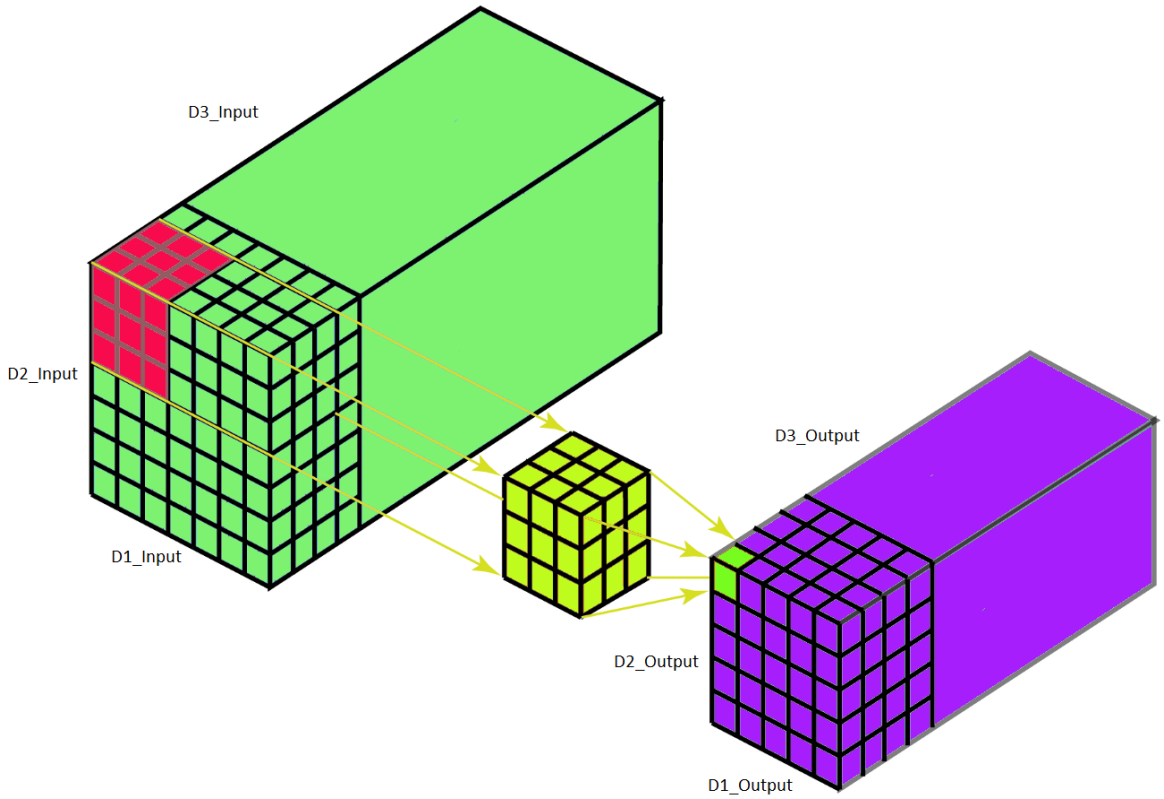


Figure 2.5 Illustration of 3D convolution.

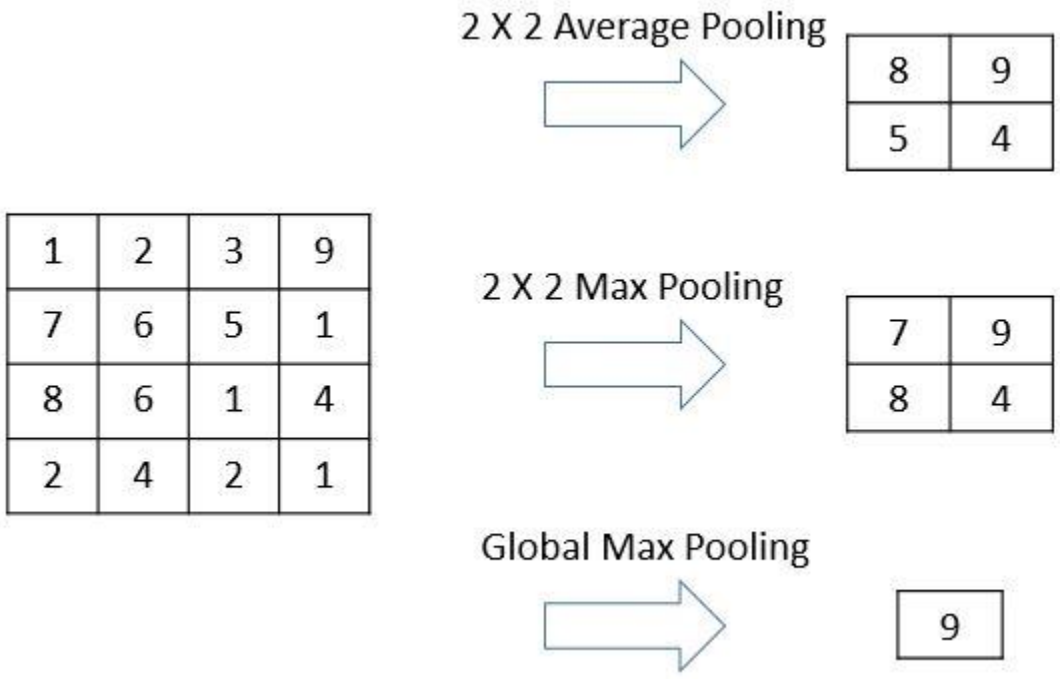


Figure 2.6 Max pooling, average pooling, and global max pooling.

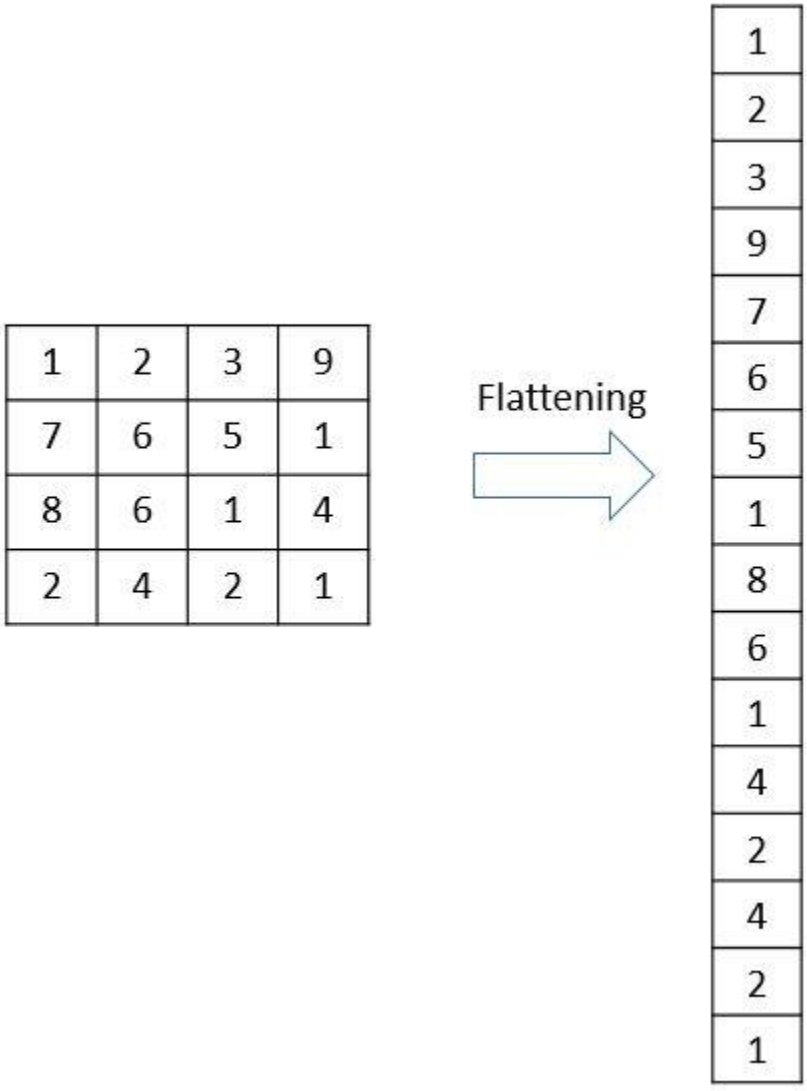


Figure 2.7 Example of flatten layer.

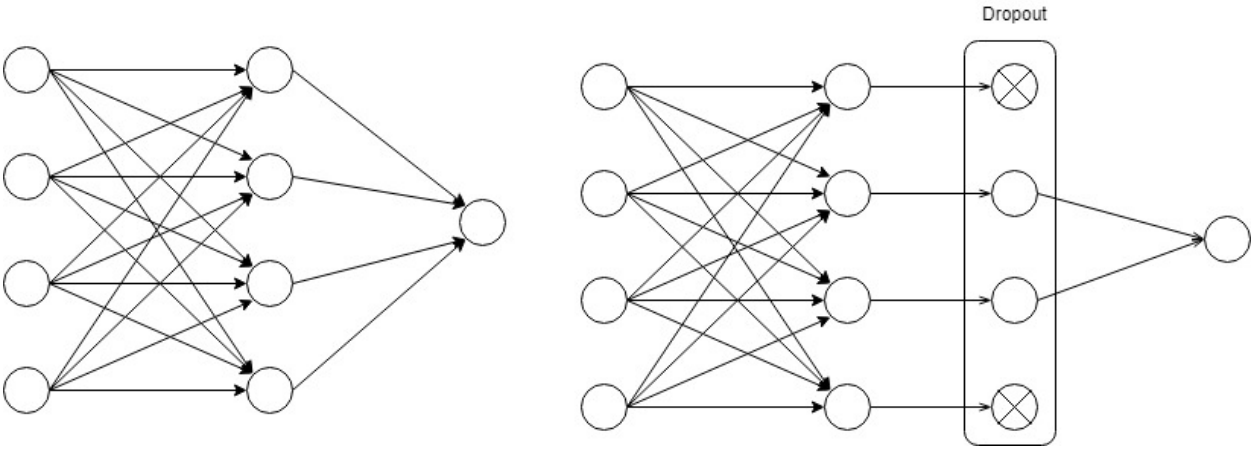


Figure 2.8 Model without dropout (left) vs. model with dropout (right).

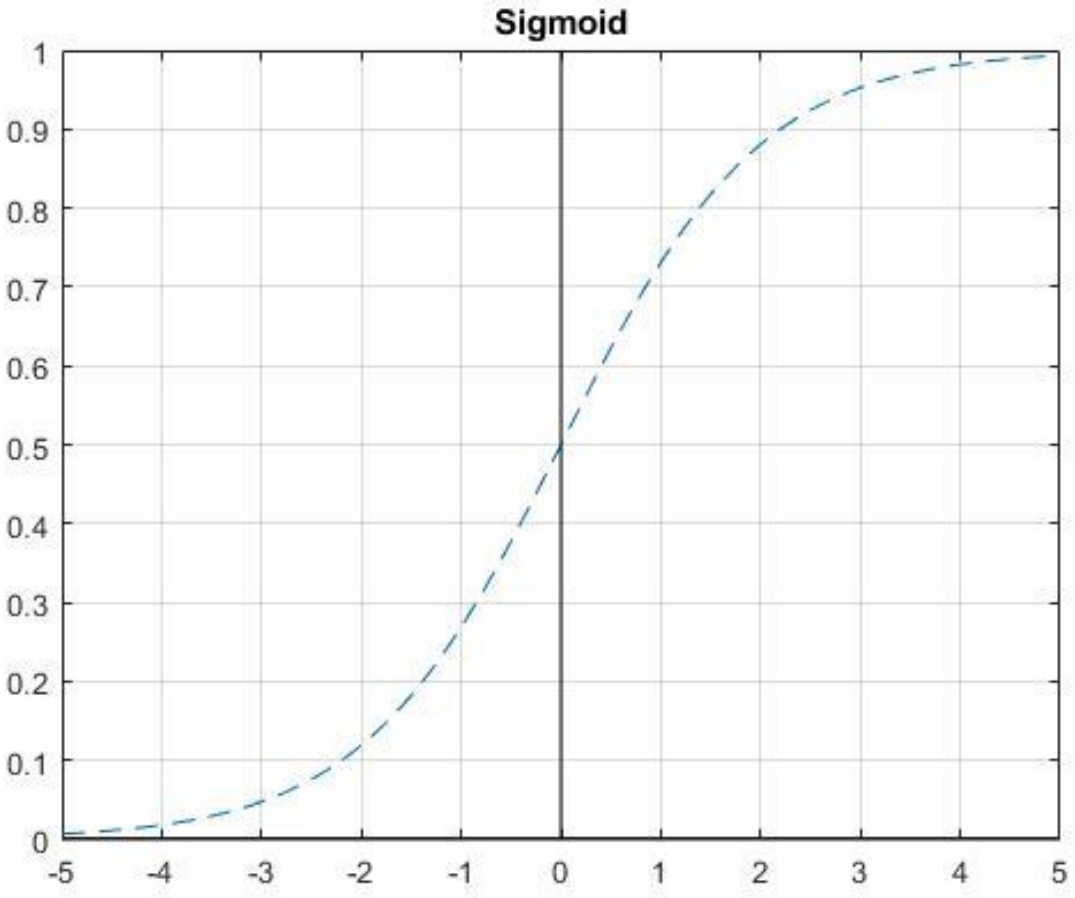


Figure 2.9 Sigmoid activation function.

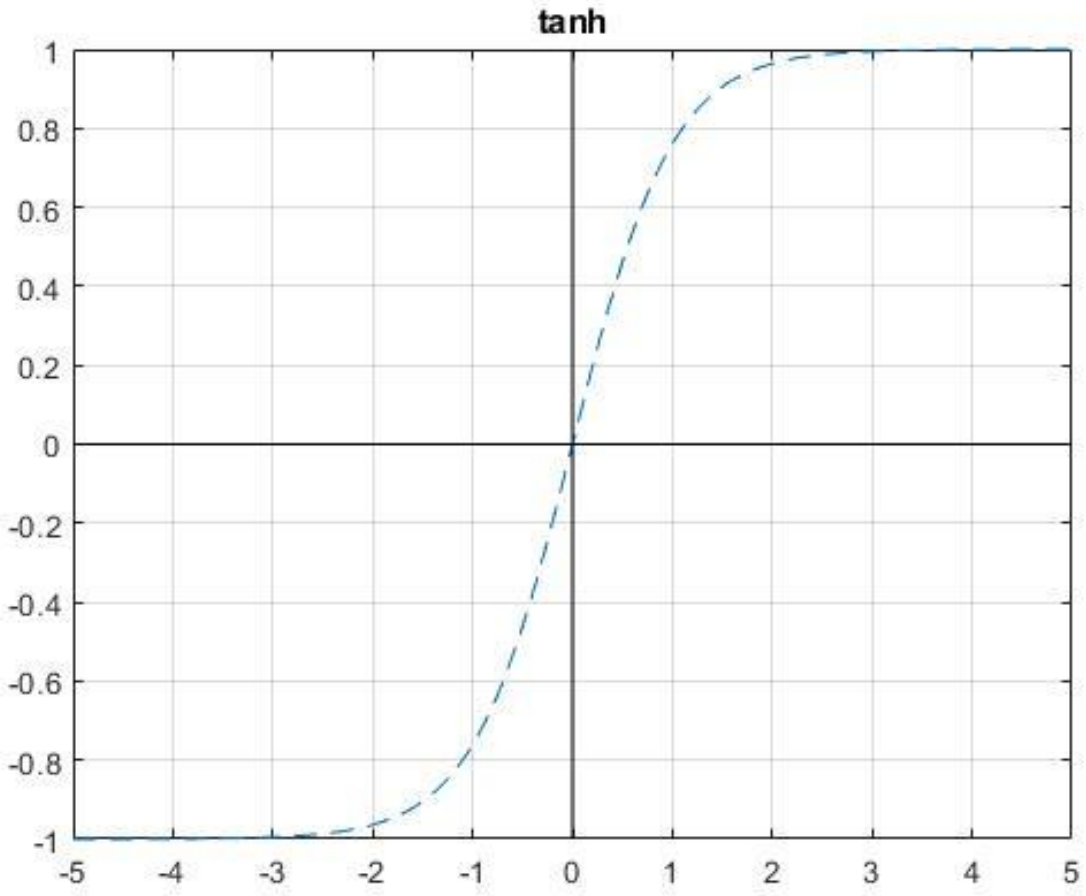


Figure 2.10 Hyperbolic tangent activation function.

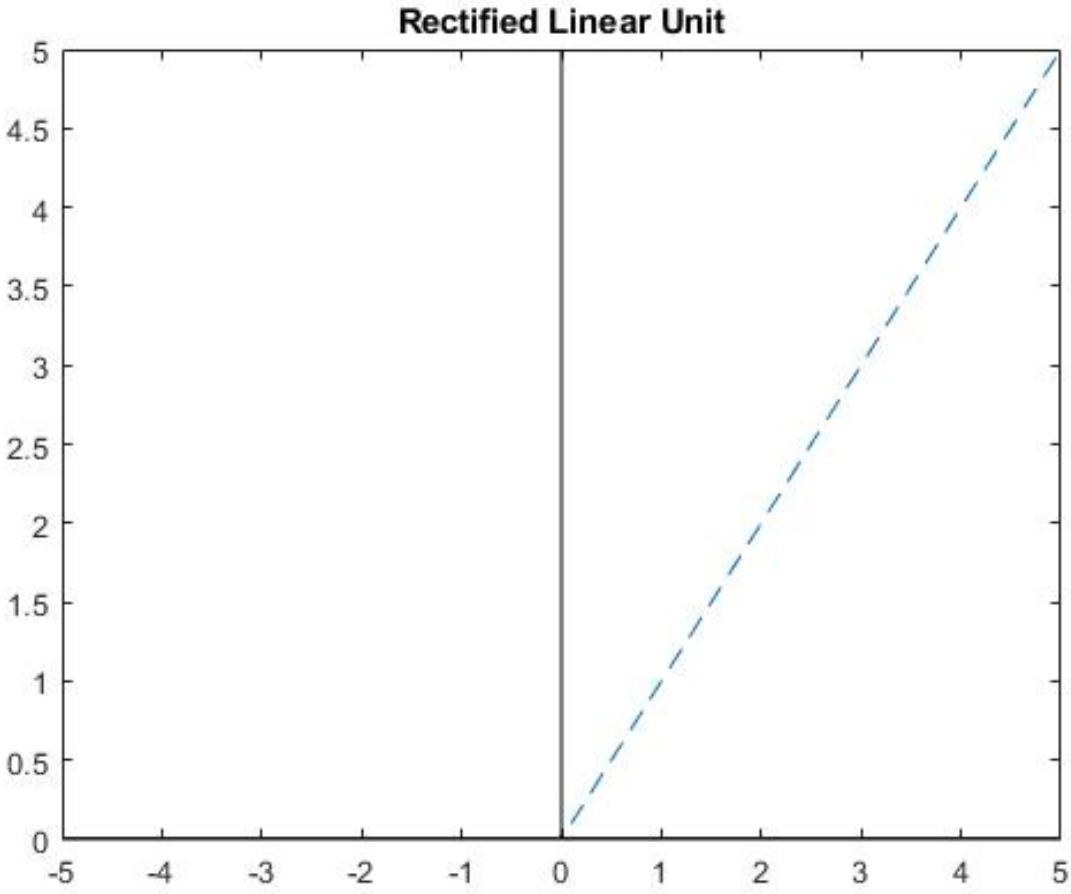


Figure 2.11 Rectified linear unit (ReLU) activation function.

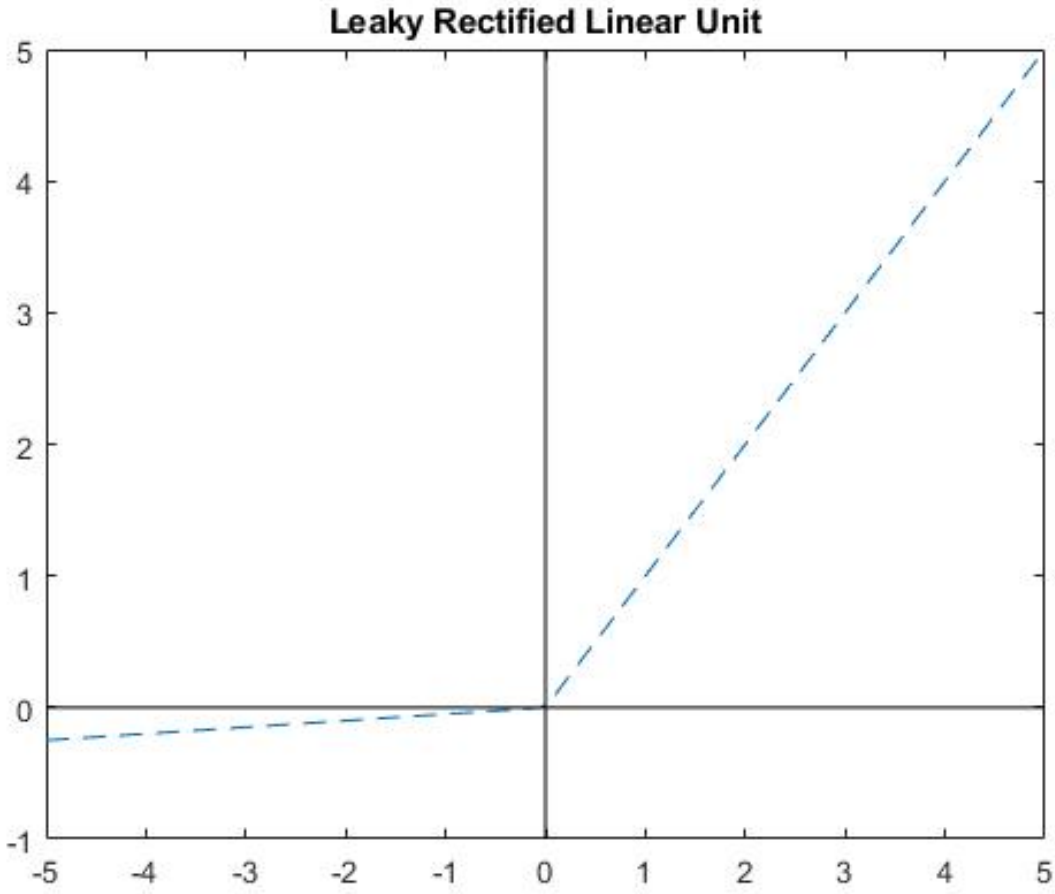


Figure 2.12 Leaky ReLU activation function.

CHAPTER 3. OBJECTIVES

Based on the introduction and discussion, the specific objectives of this work are:

1. Improving Alzheimer's disease diagnostic

To develop a machine-learning based intelligent model for prediction of specific disease conditions associated with Alzheimer's disease. The specific tasks associated with this objective are:

   a. Implement and evaluate a deep 3D convolutional neural network model for prediction of Alzheimer's disease stages between Health Control (HC)/Mild Cognitive Impairment (MCI) /Alzheimer's disease (AD)
   b. Implement and evaluate a deep 3D residual neural network model for the prediction of AD stages (HC / MCI / AD).
   c. Implement the Multi-Layer-Output (MLO) architecture on the models in (a) and (b). Evaluate the performance improvement on the models.

2. Improving Alzheimer's patients' quality of individual living

To develop an indoor scene understanding model that would predict the patients' whereabouts with single taken images, without facilitation of the sensor network or prior knowledge of the room layout. This idea was inspired by Dr.Panigrahi.

CHAPTER 4. METHODOLOGY

4.1 Deep 3D Convolutional Neural Network for AD classification

This work explores, investigates, and reports four classification models for multiclass Alzheimer's disease diagnosis. These models use structural magnetic resonance imaging (sMRI) as classification evidence and is capable of revealing the latent causality between patients' structural changes in their brain and their progressiveness of the disease.

In this study, the sMRI acquired from Alzheimer's Disease Neuroimaging Initiatives (ADNI) were used to conduct experiments in conjunction with two 3D convolutional neural network-based models and effect of Multi-Layer-Output mechanism. The classification is performed by a fully connected classifier. Their performances were compared between their own variations and other state-of-art methodologies.

The result obtained from this study revealed that the proposed 3D models showed good accuracy on 3-class (Normal Control, Mild Cognitive Impairment, and Alzheimer's disease) classification. The proposed Multi-Layer-Output structure also boosted the performance of classification, compared with the native models. The detailed program implementation can be found in Appendix A.

4.1.1 Data Acquisition and Preprocessing

4.1.1.1 Dataset

The dataset used in this work was the ADNI1: Annual 2 Yr 1.5T dataset. These data were obtained from the Alzheimer's disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu). As per the condition of the acknowledgement for the use of this data set, the following specific italicized texts are presented below within the quotation marks.

"The ADNI was launched in 2003 by the National Institute on Aging (NIA), the National Institute of Biomedical Imaging and Bioengineering (NIBIB), the Food and Drug Administration (FDA), private pharmaceutical companies and nonprofit organizations, as a $60 million, 5-year public-private partnership. The primary goal of ADNI has been to test whether serial magnetic resonance imaging (MRI), positron emission tomography (PET), other biological markers, and clinical and neuropsychological assessment can be combined to measure the progression of mild cognitive impairment (MCI) and early Alzheimer's disease (AD). Determination of sensitive and specific markers of very early AD progression is intended to aid researchers and clinicians in developing new treatments, monitoring their effectiveness, and lessening the time and cost of clinical trials.

The principal investigator of this initiative is Michael W. Weiner, MD, VA Medical Center and University of California - San Francisco. ADNI is the result of efforts of many co-investigators from a broad range of academic institutions and private corporations, and subjects have been recruited from over 50 sites across the United States and Canada. The initial goal of ADNI was to recruit 800 subjects but ADNI has been followed by ADNI-GO and ADNI-2. The follow up duration of each group is specified in the protocols for ADNI-1, ADNI-2 and ADNI-GO. Subjects originally recruited for ADNI-1 and ADNI-GO had the option to be followed in ADNI-2."

The dataset obtained in this work is a subset of the entire ADNI dataset, namely ADNI1 dataset. It contained 1725 structural MRI scans, with 346 AD scans, 573 NC scans and 806 MCI scans. The MRIs were collected with 1.5T scanner using T1-Weighted sequence. A detailed demographics of this dataset was shown in Table 4.1. Sample images of NC, MCI and AD was shown in Figure 4.1.

This work was approved by Purdue University Institutional Review Board (IRB) protocol #1906022293 under determined exemption (Category 4) (Appendix D).

4.1.1.2 Data Preprocessing

The MRI scans from ADNI were reportedly acquired using Magnetization Prepared Rapid Gradient Echo (MPRAGE; Mugler & Brookeman, 1990). MPRAGE is a fast 3D gradient echo pulse sequence that considerably providing excellent scan quality. Each MPRAGE scan in the dataset were then corrected by the routines provided by the vendor of scanner (GE) when they were collected by the researchers in ADNI. These routines included Gradwarp, B1 Correction and N3 Correction. In our research, preprocessing was not conducted at Purdue.

4.1.1.3 Data Postprocessing

4.1.1.3.1 Denoising

The brain MRI acquisition is long process and the image is reconstructed by stacking the frames of scanning. The usual noises in MRI are Gaussian and Rician (Gudbjartsson & Patz, 1995; Macovski, 1996). Therefore, removal of these noises and slices registration are very crucial but often neglected. This is due to the level of denoising is hard to calibrated thus endanger the information integrity.

Nevertheless, the denoising method proposed in this framework is the optimized nonlocal mean 3D filter that described in Buades, Coll, and Morel (2005). This method uses the redundancy of information in the image under study to remove the noise. A routine developed by Manjon et al. (2010) was run in MATLAB R2018b at Purdue. Specifically, a MRI scan was divided into blocks with overlap voxels. For a given voxel included in several blocks, the nonlocal mean restorations were computed and averaged to obtain the final estimation. This routine was

implemented by Manjon et al. (2010) and run in MATLAB R2018b. A pre/post denoising comparison was shown in Figure 4.2.

4.1.1.3.2 Non-Brain Tissue Removal

Typical non-brain tissue removal involves skull stripping and cerebellum removal. Without non-brain tissue removal, the follow up procedures like co-registration and normalization could be complicated. Detailed comparisons between various brain extraction methods have been conducted by Iglesias, Liu, and Thompson (2011), leading to a conclusion that all algorithms (e.g., Statistical parametric mapping [SPM; Penny et al., 2011], Brain extraction tool [BET; Smith, 2002], Brain surface extractor [BSE; Shattuck, Sandor-Leahy, Schaper, Rottenberg, & Leahy, 2001], and Minneapolis Consensus Strip [Rehm et al., 2004]) perform similarly and are subject to small variation depending on the specific dataset. The tool used in this proposed work is the brain extraction tool (BET; Smith, 2002), embedded in FSL neuroimaging library. In this routine, the first step was to use the histogram of voxel intensities to determine the threshold of the intensity values. From there, the center of gravity was estimated, as was an initial sphere. Finally, the tessellated sphere was gradually decomposed outward towards the surface of the brain, until the contoured region reached the outliers in the histogram (Figure 4.3). This procedure was completed in Vmware workstation 15 with CentOS virtual Linux system.

Another typical tissue often proposed to remove is the cerebellum (Zhang et al., 2011; Zhang, Shen, & Alzheimer's Disease Neuroimaging Initiative, 2012) as it was considered mainly controlling human motor functions. However, lately, researcher started to investigate the role of cerebellum plays in cognitive functions (Jacobs et al., 2017). Therefore, in this proposed research, the cerebellum will be preserved for feature analysis. Masked views of brain matters included

white matter (WM), grey matter (GM), cerebral spinal fluid (CSF) and skull were shown in Figures 4.4 – 4.7. The final skull-striped MRI scan was shown in the top of Figure 4.8

4.1.1.3.3 Resize and Normalization

The original MRI scans were stored as nifti files. The nifti format, or the Neuroimaging Informatics Technology Initiative, is an advanced version of widely used ANALYZE 7.5 format, with additional supplementary information in the header such as affine coordinates that mapping between voxels and spatial location. Each MRI scan obtained from ADNI-1 dataset was of size of 256 by 256 by 166. Due to memory constraint on the computation unit, these original scans were first resized to remove the background. Specifically, a new 3D scan with a size of 96 by 96 by 64 was cropped from each of the original scan. The cropped image was centered at its hippocampus and dilated from three axes. An example of resized image was shown in the bottom of Figure 4.8.

Afterward, a standard normalization was adopted on these cropped scans. Specifically, for each scan, its data object was first read as 3D numpy array with data type of 32-bit float. These data arrays were then scaled to the range of [0, 1]. Finally, the means of each scans were subtracted from these images. Denotes the cropped MRI scan data array as I, the scaled array as I_{scale} and the final centered image as $I_{centered}$, the steps was shown in equation (13) and (14):

$$I_{scale} = \frac{I - \min(I)}{\max(I) - \min(I)} = \frac{I}{\max(I)} \qquad (13)$$

$$I_{centered} = I_{norm} - \text{mean}(I_{scale}) \qquad (14)$$

After completion of these processing steps, the final output format of each MRI scan was 3D data array with a size of 96 by 96 by 64. Each of the voxels in these data arrays contained an intensity value with zero mean and unit variance. The overall processing flowchart was shown in Figure 4.9.

4.1.2 Model Architecture

To develop accurate predictive model of AD stages was the scope of this objective. In this research, various model was implemented to examine the performance on the targeted task. Specifically, two deep 3D convolutional neural network models were proposed and tested at first. Later on, the ideology of Multi-Layer-Output (MLO) was introduced and explained. Finally, the MLO was implemented on the existing two models to boost their performance. The detailed code implementation could be found in Appendix A.

4.1.2.1 Deep 3D Neural Network Model (Model A)

The detailed graphical model overview was shown in Figure 4.10. This conventional deep neural network contained seven trainable hidden layers and various other components. Systematically, this model could be divided into a feature extraction model and an artificial neural network classifier. Some global setting was implemented for all the layers that applied. First of all, zero padding (see Section 2.3.1.2) was applied to all the convolutional layers to make sure all the voxels had been used in the convolution. Secondly, L2 regularization (see Section 2.3.3) was applied to all the convolutional layers to further reduce overfitting. In the rest of the paper, this model will be referred as Model A. The entire model can be described in a top to bottom manner, as following:

1. Input layer – Output size: (96 × 96 × 64, 1 channel)

The input layer read the data array from the batches that were prepared for the model. As discussed in Section 4.1.1, the MRI scans had dimension of 96 × 96 × 64. The input layer thus had the same size as the input data.

2. 3D convolutional layer: Conv_1 – Output size: (48 × 48 × 32, 32 filters)

A 3D convolutional layer was connected to the input layer. Such layer had a kernel of size 7 × 7 × 7 and stride of (2, 2, 2). A kernel with stride of 2 would reduce the dimension on the given axis by half. The intension of giving big receptive field to this layer was to quickly extract low-level feature. Additionally, a batch normalization was added to reduce covariant shift. This was because batch normalization could scale the outputs of the neurons to have zero mean and unit variance. After that, a leaky ReLU activation (which had a very small gradient for the negative inputs compared with ReLU), was added. A total number of 32 filters were included in this layer, creating an output size of 48 × 48 × 32, 32 filters (reduce by half from 96 × 96 × 64 on all axes).

3. Average pooling layer – Output size: (24 × 24 × 16, 32 filters)

An average pooling layer was then connected to Conv_1. As described in Section 2.3.1, the pooling layer can effectively downsample the size of feature map while preserving the information. In this case, the layer had a kernel size of 5 × 5 × 5 and a stride of (2, 2, 2). The output size was then downsampled to 24 × 24 × 16, 32 filters (reduce by half from 48 × 48 × 32 on all axes).

4. 3D convolutional layers: Conv_2, Conv_3, Conv_4 – Output size: (3 × 3 × 2, 256 filters)

Sequentially, three 3D convolutional layers were attached to the previous layer. These layers served the purpose of gradually increase the level of extracted features. Therefore, they had relatively smaller receptive field (3 × 3 × 3) than the previous convolutional layer (Conv_1). The number of filters in each layer also gradually increased, namely, 64, 128 and 256. The same stride of (2, 2, 2) was applied to the convolutional layers, the same batch normalization, and the leaky ReLU. The output sizes of the convolutional layers were:

Conv_2: 12 × 12 × 8, 64 filters (reduce by half from 24 × 24 × 16 on all axes).

Conv_3: 6 × 6 × 4, 128 filters (reduce by half from 12 × 12 × 8 on all axes).

Conv_4: 3 × 3 × 2, 256 filters (reduce by half from 6 × 6 × 4 on all axes).

5. Global average pooling layer – Output size : (256)

The final feature maps were pooled by a global average pooling layer (see Section 2.3.1.3) afterwards. This layer was to produce a feasible feature representation for the upcoming classification. The output size of this layer was a vector of 256 elements (Each 3 × 3 × 2 feature map produce one feature value).

6. Dropout layer

The dropout layer was the final layer of the feature extraction model. This layer aimed at reducing overfitting by penalizing the feature maps. It would randomly reset selective amount of feature maps to zero, also called deactivate a selection of nodes/neurons, according to the preset drop rate *dp*. The output size of this layer was still 256.

7. Fully Connected layers –Output size : (3)

Three fully connected layers consisted the classifier in this model. Effectively, such classifier can be described as a multilayer perceptron (see Section 2.3.1) with three hidden layers, where the input was obtained from the feature extraction model. The first two layers had ReLU activation. They had 128 and 64 neurons respectively. The last layer had three neurons with softmax activation (see Section 2.3.2) to produce the output probability.

8. Output layer – Output size: (3)

Based on the calculated probability, the final prediction was conducted by applying argmax function to the probabilities.

4.1.2.2 Deep 3D Residual Neural Network Model (Model B)

To examine whether increasing the depth of network could improve the classification performance, a much deeper 3D Residual neural network was derived from He et al. (2016). This model expand the idea of residual block to the 3D environment to enable us building very deep neural network. Compared with the model proposed earlier, this 3D residual network had 13 trainable layers. A detailed model was shown in Figure 4.11. In the rest of the work this model will be referred as model B.

To compare with result from previous model, the majority of the model remained the same. That included the first three layers and the last six layers. The difference between them was the new model replaced the three convolutional layers with three "ResBlock." As demonstrated in Figure 4.12, the residual block, or "ResBlock," had three convolutional layers in it. The first convolutional layer in the main branch had a kernel size of $3 \times 3 \times 3$, an adjustable number of filters, n1, and a stride of (2, 2, 2). The second convolutional layer had a kernel size of $3 \times 3 \times 3$, an adjustable number of filters, n2, and a stride of (1, 1, 1). The third convolutional layer that was on the shortcut branch had a kernel size of $1 \times 1 \times 1$, the same number of filters, n2, as the second layer, and a stride of (2, 2, 2).

By giving n1 and n2 values, multiple instances of ResBlock can be implemented in the model. In this case, three ResBlocks were declared with [32, 32], [64, 64] and [128, 128] filters, respectively. The output feature map size was $3 \times 3 \times 2$, which was the same as the first model.

4.1.2.3 Deep 3D Models with Multi-Layer-Output (Models C and D)

Moreover, a new concept of Multi-Layer-Output (MLO) was developed for this application. The goal was to explore if combining features that were created at different level can increase the classification accuracy.

4.1.2.3.1 Intuition

Generally, there is a perception that deep learning performs better than shallow learning because its ability to automatically learn high level features. Such features were either too costly for human to manually engineer, or researchers didn't understand the task well enough to create discriminant features. Often times, restrained by the limited observation, researcher constantly drew conclusions that seemed convincing at the moment but overtime got proven to be wrong. Nowadays, empowered by the exploding amount of data, deep learning models can effectively draw conclusion from data without human conjecture, such as speech recognition and natural language processing. However, deeper models were not always better. Model optimization is always a challenging research focus that needs constant attention. Despite the consistent attempts on building deeper network, as shown in Figure 4.13, results showed that simply increase the depth of the network may cause the performance to decline (He et al., 2016). The same researchers made another interesting observation that a well-trained model would quickly degrade if some plain identity layers were added to it.

These observation indicated that even though deep models showed strong performance in automatic learning, the tuning of model topology was still a challenging job for researchers (He et al., 2016). Given the fact that there are millions of tasks in the world, it is unthinkable to request a fine-tuned model for every one of them. However, if we could provide the classifier with a combination of multiple feature representations, it is possible for the classifier to select the best features based on the training data.

4.1.2.3.2 Implementation

Inspired by the discussion, two modification were applied to the existing models. In Figure 4.14, the new model (Model C), which was originated from the model in Figure 4.10

(Model A), had output three feature representations separately from the three convolutional layers. As mentioned in Section 4.1.2.1, item 4, the out sizes of these layers were 12 × 12 × 8 × 64, 6 × 6 × 4 × 128, and 3 × 3 × 2 × 256. To produce a useable feature representation, these feature maps were connected with global average pooling layers. Therefore, three feature vectors, with sizes of 64, 128 and 256, were created. Later on, these feature vectors were concatenated sequentially and resulted in the final feature vector which had a size of 64 + 128 + 256 = 448. The rest of the model was the same setup as the Model A except the size of the dropout layer was now 448 to adapt the changes in the feature vector. In the rest of the work, this model will be referred as Model C.

Similarly, as shown in Figure 4.15, the model was modified based on the model in Figure 4.11. The output feature maps of the three 3D ResBlocks had sizes of 12 × 12 × 8 × 32, 6 × 6 × 4 × 64, and 3 × 3 × 2 × 128. After separated pooling operations, three feature vectors of size 32, 64, and 128 were created. After concatenation, the final feature vector was thus created with a size of 32 + 64 + 128 = 224. The rest of the model was the same as the model in Figure 4.11 except the size of the dropout layer was now 224. In the rest of the work, this model will be referred as Model D.

4.1.3 Grid Search

Other than the specifications mentioned in Section 4.1.2, there were many hyper-parameters that were not specified. In machine learning, “parameter” refers to the characteristic of the model that can be gradually approximated by optimized training process. An example of parameter can be the weights in a kernel, which is gradually derived and updated based on the model performance on the training data (Nielson, 2015). On the other hand, “hyper-parameter” refers to the set of model specifications that were determined before the training process and

cannot be further tuned by the training data (Kuhn & Johnson, 2013). Examples of the hyper-parameters can be number of layers in the network or the value of the drop rate in a dropout layer. To find the optimal combination of the hyper-parameters, grid search was often adopted in the training process. The fundamental idea of grid search was to conduct exhaust search on all the combinations in a given hyper-parameter space. This process would return the combination that achieve the highest performance on the monitored metrics.

In this work, three hyper-parameters were remained to be determined by grid search, namely, the drop rate, *dp*, the coefficient of the L2 regularizations, λ (equation 15), and the leak rate of the leakyReLU units, *leakR* (equation 16).

$$Error_{new} = Error + \lambda \sum \beta \qquad (15)$$

$$output = \begin{cases} x, & if\ x > 0 \\ leakR \cdot x, & if\ x \leq 0 \end{cases} \qquad (16)$$

The scan range of *dp* was [0.1, 0.7] with step size of 0.1. The scan range of λ was [0.01, 0.05] with step size of 0.01. The scan range of *leakR* was [0.01, 0.03] with step size of 0.01. Given the targeted classification task was multiclass classification, the monitored metric will be the average accuracy of cross validation. For each combination, a 5-fold cross validation was conducted and their accuracies will be recorded. The combination of hyper-parameter that achieved the highest average accuracy was selected for the final model.

4.1.4 Train and Testing Method

4.1.4.1 Training

Due to the limited amount of data, a 5-fold validation was conducted for all the proposed model to reduce bias. The training scheme was shown in Figure 4.16. The global training hyper-parameters were:

- Learning rate:

  The initial learning rate was $1e^{-4}$. A learning rate reduce on plateau was incorporated. That is, reduce the learning rate by a factor of 0.5 when validation loss doesn't decrease for three consecutive epoch. The minimum learning rate was $1e^{-5}$.

- Epoch and batch size:

  The total number of epochs was 80. The batch size was 20

- Optimizer

  The optimizer adopted in this work was adaptive moment estimation (ADAM; Kingma & Ba, 2014). Additionally, a save-best strategy was adopted, which would only update the weights when validation loss improves.

  The whole process can be broken down into five steps.

Step A - The dataset used was the processed ADNI1 dataset (totally 1725 scans). Eighty-five percent of the data were randomly sampled from the entire dataset (totally 1466 scans). This proportion was used for cross validation while the remainder was preserved for testing (totally 259 scans). These 85% of the data were divided into five folds evenly (17% of the data in each fold).

Step B - The prepared five folds of data were reorganized as shown in Figure 4.16. The idea was to make sure that each fold had been selected for validation just once. This way, all the data observation are validated once and therefore, reduce the bias that came with the imbalanced dataset.

Step C - The training then began with one of the combinations in Step B. To relieve the computation burden and accelerate the learning, a progressive loading routine, denoted as "Data Generator" was adopted in this work. Basically, instead of reading all the data into memory, the

training and the validation data were prepared into batches of 20 scans. In all time, only 10 batches were read into the memory simultaneously. The detailed implementation of the data generator could be found in Appendix A.

Step D - The prepared batches of data were then fed into the model, one at a time. The model to be trained was one of the four models proposed in Section 4.1.2 (Models A-D). The grid search methods, described in Section 4.1.3, was implemented in this step. One combination from the hyper-parameters space was used throughout the cross validation. As one batch of training data finished, the training accuracy and the training loss was recorded for evaluation purpose. This process was repeated until the entire training data had been used. After that, a similar progressive batch loading was adopted for validation data as well. The final validation accuracy was recorded when all the validation batches had been used. This entire maneuver, also known as one epoch, was repeated 80 times, giving the model enough time to converge. Upon competition of all the epochs, one of the 5- fold cross validation was considered as completed. The next step was to route back to Step B, where a new data split was prepared for the next cross validation test. The entire process (Steps B through D) were repeated five times to complete the cross validation.

Step E – One set of 5-fold cross validation was conducted for each combination in the hyper-parameter space. As described in Section 4.1.3, the hyper-parameters to be determined by grid search were drop rate, *dp*; Leak rate, *leakR*; and the L2 coefficient, λ. Based on the search range, each model was trained 7 (# of *dp*)× 3 (# of *leakR*) × 5 (# of λ) = 105 times. The corresponding training performance was recorded to determine which hyper-parameter setting achieved the best classification accuracy.

4.1.4.2 Testing

As shown in Figure 4.17, 15% data that were hold out in the training were used for testing. Progressive loading was used to prepare the batches for testing. For each model proposed in Section 4.1.2, among its 105 trainings, one model that achieved the best average validation accuracy was selected for testing. The models predicted the labels (Stages of subject – NC, MCI, or AD) of the testing data and then later compare with the ground truth labels. The test accuracy will be recorded for comparison.

4.2 Indoor Scene Understanding

4.2.1 Dataset

In this study, we focused on identifying the most common home indoor scene. Typically, datasets with the image collection of bedroom, dining room, and living room were considered. To thoroughly test the performance of our model and further test the model's generalization ability, we collected the following datasets:

1) **Large-scale Scene Understanding** (LSUN; Yu et al., 2015)
    - Includes 10 indoor/outdoor scene categories and 20 object categories. See Figure 4.18.
    - The total number of images with indoor scenes is over five million.
2) **MIT Indoor67** (Quattoni & Torralba, 2009)
    - Includes 67 indoor scene categories. See Figure 4.19.
    - The total number of images with indoor scenes is 15,620.
3) **15-Scene** (Fei-Fei & Perona, 2005; Lazebnik Schmid, & Ponce, 2016; Oliva & Torralba, 2001)
    - Includes 15 indoor/outdoor scene categories. See Figure 4.20.

- The total number of images is 4490.

In our experiment, the data from LSUN dataset was used for fine-tuning the model and testing. The data from MIT Indoor67 and 15-Scene were prepared for validating the model's ability for data generalization. Three sets of approaches were designed to thoroughly test the performance.

Approach A:

Specifically, 30,000 images were randomly sampled out of the LSUN dataset with 10,000 images from each categories respectively. These images were later divided and resulted in 3,000 images for testing and 27,000 for training and validation. The model was then fine-tuned and validated using 5-fold cross validation.

Approach B:

Moreover the entire LSUN dataset which contains 5,006,415 images belonging to three indoor scene categories (bedroom, living room, dining room) were used for training. These images were randomly split into training, validation, and testing with 60%, 20%, and 20% ratios respectively. This resulted in a training size of 3,151,092, and validation/test size of 938,472.

Generalization tests were conducted in both approaches. Extensively, 211 images were collected from the MIT Indoor67 dataset and 505 images were collected from the 15-Scene dataset that belonged to the target classes.

Approaches C:

To avoid the collection bias in the existing dataset and to better simulate the real-world environment for testing. 15 real-life images were collected using various daily equipment such as cellphone camera and webcam. These images were used finally on the optimized model. See Figure 4.21.

4.2.2 Model Architecture

In this work, we transferred a deep Residual Neural Network (ResNet; He et al., 2016) that was intended for object detection to our targeted domain for feature extraction (see detailed code implementation in Appendix B). The reason of choosing ResNet – based model is because it has two major advantages. Firstly, the ResNet model, compared with other popular architectures, has much smaller model size considering its depth and performance. As shown in the Figure 4.22 (Canziani, Paszke & Culurciello, 2016), compared with VGG models (Simonyan & Zisserman, 2014) that had approximately 138 million parameters, ResNet models had model sizes ranged from 25 million to 65 million parameter depending on specific variation. This would greatly accelerate the training speed when trying to go deeper on the model. Secondly, the shortcut connection in each block could effectively solve another major problem when having very deep model, which is the gradient vanishing problem (He et al., 2016).

Deep Residual Neural Network (ResNet) has various variation and topology/depth. Depending on the specific task, the ResNet model can 18 layers, 34 layers, 50 layers, 101 layers and 152 layers. The detailed models were summarized in Table 4.2. The model adopted in this work was a native ResNet-50 model (He et al., 2016) that was pretrained on ImageNet. The idea of transferring learning is, if the source classification task is similar to the target classification task, the knowledge learned from the source task would ideally be very similar to the knowledge required to perform the target classification task. Similarly, in machine learning, transfer learning refers to the method that adopt the prior knowledge of another task that was related to the current target task. To do so, the trained weights in the source model will be copied to the target classification model as the weigh initialization. The model with this transferred weights will then be fine-tuned with the data in the target domain. This would significantly improve the training convergence and the classification performance.

Naturally, the complexity of interior design has brought very serious challenges for feature extraction given the diverse intraclass and interclass variation (Khan et al., 2015). Simple low-level semantic understanding of the objects in the scene would not contribute greatly to the scene classification due to its inconsistency (Khan et al., 2015). Therefore, high-level feature representation needs to be established to capture such characteristic. In our circumstances, the pretrained model was targeted on object detection. However, considering the large resemblance on low-level features between the source domain and the targeted domain, it was reasonable to transfer the prior knowledge from object detection to scene understanding with proper domain adaption.

Traditionally, neural network with very deep architecture can be severely problematic in training. The gradient vanishing problem (He et al., 2016) would cause the gradient to become very slow as it transmitted through layers. This would lead to the weights in the bottom layers to be not updating effectively. Conversely, a deeper network was believed to has better ability to extract high-level features (Eldan & Shamir, 2016). To that end, we adopted a ResNet-50 model which would be able to extract high-level features through very deep network while avoiding the vanishing gradient problem. ResNet model has been proven very powerful in many machine learning scenario. Its strength comes from the unique shortcut connection and the residual block design. Model with such topology could effectively solve the gradient vanishing problem when using very deep network to extract high-level feature (Akiba, Suzuki & Fukuda, 2017; Jung et al., 2017; Xie et al., 2017). In this case, the gradients would not pass through all the layers but instead, would transmit through the identity mapping. So in a way, the model is both deep when performing feature extraction and shallow when back-propagating the gradients (He et al., 2016).

4.2.2.1 Model Summary

A detailed training flowchart can be found in left segment of Figure 4.23. Specifically, the following describe the model:

1. Segment A: input layer

This layer read the input data from the batches prepared for the model. The output shape is 224 × 224 pixels to comply with the size of images.

2. Segment B: 2D Convolution with padding

This layer was a 2D convolutional layer with kernel size of 7 × 7. The stride of the filters was 2 and the number of filters was 64 (layer denoted as conv1). Zero padding of 3 was applied to the layer before convolution. Therefore, the actual input size of 2D convolution is 230 × 230 (224 + 3 + 3 = 230). After the convolution, the output feature map size was changed from 224 to 112 (see equation 15) and resulted in an output size of 112 × 112 × 64.

$$o = \frac{224 - 7 + (3 \times 2)}{2} + 1 = 112.5 \approx 112 \qquad (17)$$

3. Segment C: 2D Max pooling with padding

This layer was a 2D max pooling layer with kernel size of 3 × 3 and stride of 2. Similarly, zero padding of 1 was applied to the layer, increased the input size to 114 × 114 × 64 (112 +1 +1 = 114). The max pooling was then applied and produced output feature maps with a size of 56 × 56 × 64 (see equation 16).

$$o = \frac{112 - 3 + (1 \times 2)}{2} + 1 = 56.5 \approx 56 \qquad (18)$$

4. Segment D: Bottleneck Residual Blocks (He et al., 2016)

As shown in Figure 4.25, a bottleneck residual block consist of three 2D convolutional layers: two layers with filter size of 1 × 1, and one intermediate layer with filter size of 3 × 3 in between. Additionally, a short cut connection (with an optional 2D convolutional layer with filter

size of 1 × 1, when the input filter number is different from the output filter numbers) from the input to the output is also included. 16 residual blocks were concatenated sequentially with different number of filters. Specifically, four different types of residual block were created in this worked, namely, conv2 to conv5 (see Table 4.2).

Conv2 blocks have 64 1 × 1 filters in the first layer, 64 3 × 3 filters in the second layer and 256 1 × 1 filters in the third layer. Conv2 block had repeated three times. The first convolution was performed with a stride of 2 while the rest of the convolutions had stride of 1. The final output size of conv2 blocks was 56 × 56 × 256.

Conv3 blocks have 128 1 × 1 filters in the first layer, 128 3 × 3 filters in the second layer and 512 1 × 1 filters in the third layer. Conv3 block had repeated four times. The first convolution was performed with a stride of 2 while the rest of the convolutions had stride of 1. The final output size of conv3 blocks was 28 × 28 × 512.

Conv4 blocks have 256 1 × 1 filters in the first layer, 256 3 × 3 filters in the second layer and 1024 1 × 1 filters in the third layer. Conv4 block had repeated six times. The first convolution was performed with a stride of 2 while the rest of the convolutions had stride of 1. The final output size of conv4 blocks was 14 × 14 × 1024.

Conv5 blocks have 512 1 × 1 filters in the first layer, 512 3 × 3 filters in the second layer and 2048 1 × 1 filters in the third layer. Conv5 block had repeated three times. The first convolution was performed with a stride of 2 while the rest of the convolutions had stride of 1. The final output size of conv5 blocks was 7 × 7 × 2048.

5. Segment E: Global average pooling

This layer performed global average pooling on the output of segment D. A global average pooling layer (see Section 2.3.1.3) will extract the average value across the entire feature

map. Given the input size was 7 × 7 × 2048, for each feature map (7 × 7), one average value will be calculate and therefore resulted in an output size of 1 × 1 × 2048.

6. Segment F: Dropout

This layer performed Dropout operation on the feature representation to avoid overfitting. The dropout rate, dp, was remained to be selected using grid search techniques. The output size of this layer remained 1 × 1 × 2048.

7. Segment G: Fully Connected Layer

Following the dropout layer was a fully connected layer with ReLU activation (see Section 2.3.2). All the neurons in this layer connected to all the 2048 features. This constructed a linear classifier. The number of the filters in this layer, nf, was remained to be selected using grid search techniques. The output size of this layer was 1 × 1 × nf.

8. Segment H: Fully Connected Layer and Output

The final output layer was a fully connected layer with three neurons that used softmax activation (see Section 2.3.2). This layer would use the output of the previous layer and calculate the probabilities of input belonging to the 3 targeted categories – bedroom, living room and dining room. This input data would then be labeled with the category that has the highest probability.

4.2.2.2 Grid Search for Hyper-parameter

As mentioned in Section 4.1.3, “hyper-parameter” refers to the set of model specifications that were determined before the training process and cannot be further tuned by the training data (Kuhn & Johnson, 2013). Therefore, hyper-parameters needs to be preemptively decided.

Tuning the hyper parameters of the model can be very arbitrary. So far, there is no systematic approach in finding the optimal combination of parameters except grid search. In this study, two hyper parameters are left to be determined by grid search to find the model with optimal performance. Namely, the drop rate of the dropout layer and the number of filters in the fully connected layer. The testing range for dropout is [0.1, 0.5] with step size of 0.1. The testing range for number of filters is [100, 500, 1000].

4.2.3 Training and Testing Process

The right segment of Figure 4.23 graphically depicted the training and the validation of the given dataset. Specifically, the dataset was first divided into training set, validation set and testing set. The training and the validation of this model used the LSUN dataset (Yu et al., 2015). The whole LUSN dataset contains more than five million images. By image preprocessing, each image was formatted into a 224 × 224 grayscale image with 8bit integer pixel intensity. To accelerate the training, all the pixels are then scaled to [0,1] and subtract from their mean. Therefore, as a result, each image is stored as a 2D numpy array with size of 224 × 224, data type of float32, zero mean and unit variance. A processed image like this takes up 224 × 224 × 32 = 1605632 bits = 200704 bytes which is 0.2 MB. Attempting to read all five million images into memory would then requires 0.2MB*5000000= 1000GB. Such hardware requirement is impossible to meet. Therefore, a customized data streamer (denoted as “Data Generator”) was used to create batches of data and streamed to the model, batch by batch. Each batch contained images of size 224 × 224, accompanied by their ground truth labels. In all time, only 10 batches of images would be queued and stored in the memory for the model. The training and the validation were then performed orderly in four steps.

4.2.3.1 Step A: Cross validation with LSUN subsets.

To accelerate training and to perform rapid model validation, we randomly sampled five subsets from the whole LSUN dataset. Each subset contained 30,000 images with 10,000 from each categories respectively. These data subsets, namely LSUN-1 through LSUN-5, were then used for initial model training and validation. A 5-fold stratified Cross Validation was conducted on each of the data subsets to reduce overfitting. Specifically, for each LSUN subset (totally 30,000 images), 3000 images (1000 images for each class) were reserved for testing. The rest of the images (27,000 images – 9000 images for each class) were used for training and validation. In each fold of cross validation, 21600 images (7200 images for each class) were used for training and 5400 images (1800 images for each class) were used for validation.

For each fold of the validation, the training was done the same way described in right segment of Figure 4.23. That is, 21600 images were used for training and 5400 images were used for validation. The model was optimized by Stochastic Gradient Decent (SGD; Robbins & Monro, 1951) optimizer, the initial learning rate was 0.0001 and the training run 10 epochs. After each epoch, the training accuracy, training loss, validation accuracy and validation loss would be recorded in the training history. Among them, the validation loss would be used as the benchmark for training performance. Neural network training is known for easily converge to local minimum on the loss function instead of the global minimum. Therefore a learning rate scheduler was used when the monitored metric is on plateau. In this case, the learning rate will decayed by a factor of 0.2 when the monitored metric, validation loss, never improved for three consecutive epochs. The minimum learning rate was set to 1e-5 and cannot be decayed further.

Once the training and validation was finished, the rest of the data in LSUN subsets (3000 images) would be used for testing. Also the MIT Indoor67 dataset and the 15-Scene dataset would be used for additional testing to examine the model's performance on the unseen data.

Generally, this was to test whether the model could generalize new data. Hence, all the tests conducted in this work that did not use the LSUN dataset would be categorized as "generalization tests." The testing process was shown in Figure 4.24.

In summary, for each LUSN subset, five models were trained using cross validation. Each trained model was tested with the remainder of the LSUN subset, the MIT Indoor67 dataset and the 15-Scene dataset. The batch size for all LSUN datasets (training, validation, testing) was 60 and the batch size for all generalization tests (MIT Indoor67, 15-Scene) was 10.

The recorded metrics included training accuracies, training losses, validation accuracies, validation losses and testing accuracies.

4.2.3.2 Step B: Grid Search

As described in Section 4.2.2.2, grid searches were conducted to obtain the optimal hyper-parameters for this particular model. In this study, two hyper-parameters were considered to be adjustable, namely the drop rate (Step F in Section 4.2.2.1) and the number of neurons in the first fully connected layer (*nf*; Step G in Section 4.2.2.1). The tested values for drop rate ranged from 0.1 to 0.7 with a step size of 0.1. The tested values for *nf* were [100, 500, 1000]. This translated into 21 grid search experiments. To further increase the confidence of this grid search, a new, larger subset of LSUN, which contained 300,000 images (100,000 for each class), were randomly sampled from the entire dataset. This dataset was divided into training set, validation set and testing set with the sizes of 240000, 3000 and 3000 images respectively. The training (without cross-validation) and the testing was then conducted the same way described in Figures 4.23 and 4.24. The additional generalization tests were also conducted with MIT Indoor67 and 15-Scene.

In summary, 21 models with different combination of hyper-parameters were trained on LSUN subset and tested on LSUN subset, MIT Indoor67 dataset and 15-Scene dataset. The batch size for all LSUN datasets (training, validation, testing) was 60 and the batch size for all generalization tests (MIT Indoor67, 15-Scene) was 10. The testing accuracies were recorded as the benchmarks of model performance. The ideal combination of hyper-parameter should achieve good performance on all the test sets.

4.2.3.3 Step C: Fine-tune with entire LSUN dataset

Later on, a more comprehension model training was conducted using the whole LSUN dataset. The dataset was divided into training set, validation set, and testing set with ratios of 60%, 20%, and 20%. This resulted in a training set with a size of 3,151,092 images, a validation set with a size of 938,472 images, and a test set with a size of 938,472 images. Because of the ample data size, no k-fold validation was conducted for this experiment. Moreover, dataset with such gigantic size was most likely to increase the duration of training and testing. To accelerate the process, the batch size of all LSUN dataset was increased from 60 to 200. The training and testing were then conducted with this updated hyper-parameter and the optimal hyper-parameter obtained in Step B (dp, nf). Similarly, the generalization tests with the MIT Indoor67 dataset and the 15-Scene dataset were also conducted afterwards. And the testing accuracies were recorded for performance evaluation.

4.2.3.4 Step D: Real scene

Lastly, 15 real life images from these three categories were also collected separately. All the images were taken using cameras of smartphones under natural environmental lighting, with no postprocessing. The reason of collecting such images was because a lot of the images in the LSUN dataset were very standard scenes that most likely taken from model show rooms. Traces

of photo editing can also be found in those images, such as lighting compensation. The most alarming concern, as shown in Figure 4.26, was the placements of objects were very clean and neat, which was uncommon in real life environment. Using the trained model from Step C, these 15 real-life images were tested to evaluate the performances of this model in classifying into one of the three scenes.

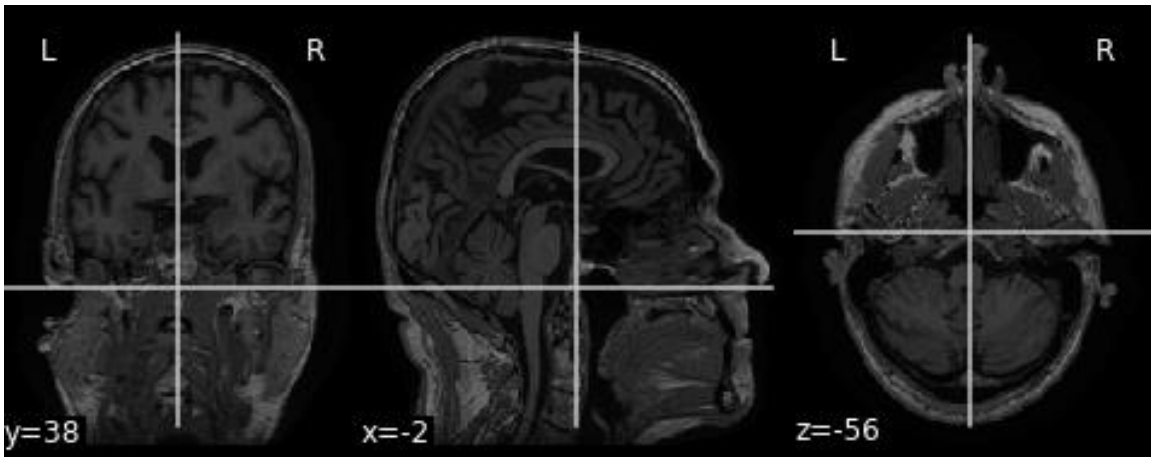
.

Table 4.1 Demographics of ADNI1 Dataset

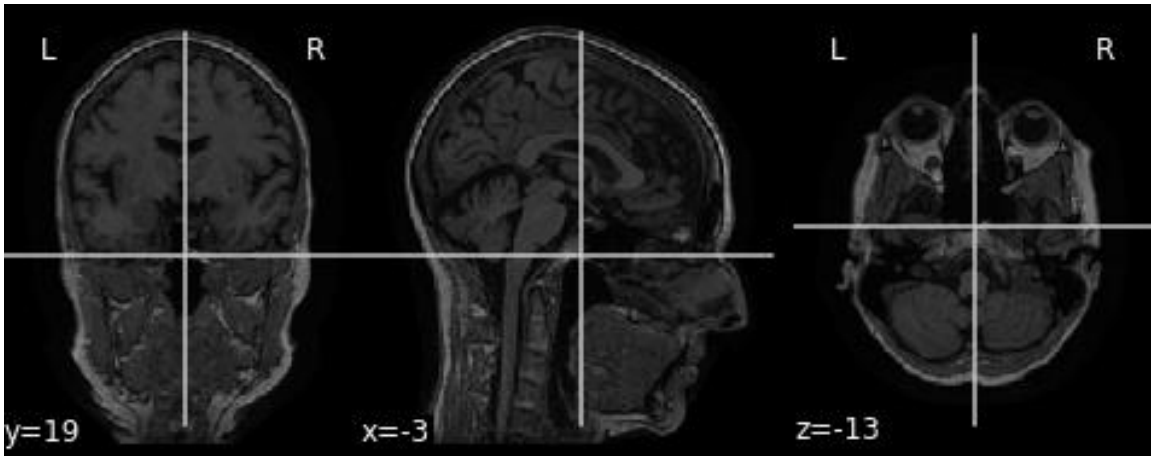
	ADNI-1
Acquisition	Scanner: 1.5T Voxel sizes: 1.2mm x 1.25mm x 1.25mm
Diagnosis	346 AD; 573 NC; 806 MCI;
Sex	Male: 949; Female: 776
Age in years (mean, stdev)	(75, 6.7)
Education in years (mean, stdev)	(15.5, 3.1)
ADAS-13 (mean, stdev, [min, max])	(18.4, 9.2, [1.0, 54.7])
MMSE (mean, stdev, [min, max])	(26.7, 2.7, [18.0, 30.0])

Table 4.2 Detailed Model Architecture for Native ResNet Models (He et al., 2016)

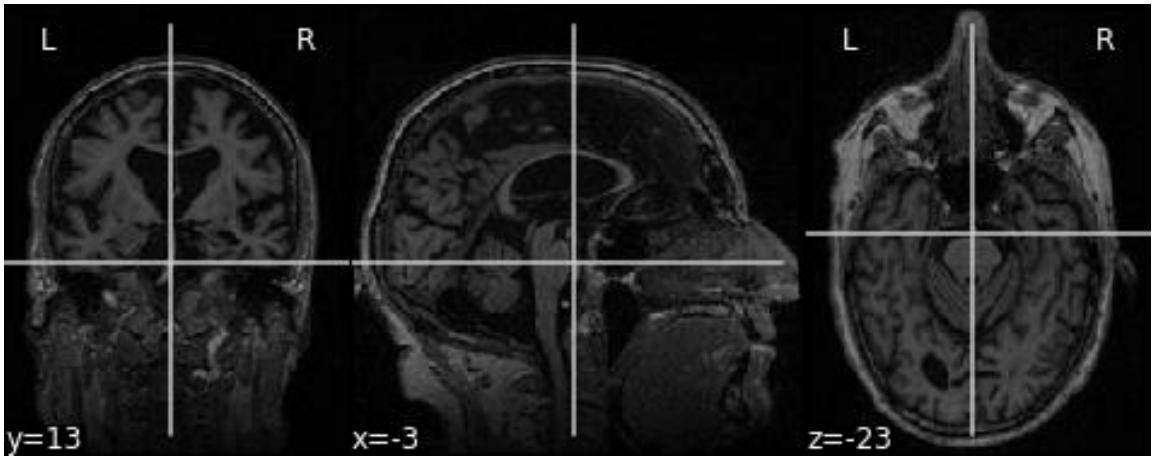
Layer Name	Output Size	ResNet-18	ResNet-34	ResNet-50	ResNet-101	ResNet-152
Conv1	112 × 112	7 × 7, 64 filters, stride 2				
		3 × 3, max pooling, stride 2				
Conv2_x	56 × 56	$\begin{bmatrix} 3\times3 & 64 \\ 3\times3 & 64 \end{bmatrix}\times 2$	$\begin{bmatrix} 3\times3 & 64 \\ 3\times3 & 64 \end{bmatrix}\times 3$	$\begin{bmatrix} 1\times1 & 64 \\ 3\times3 & 64 \\ 1\times1 & 256 \end{bmatrix}\times 3$	$\begin{bmatrix} 1\times1 & 64 \\ 3\times3 & 64 \\ 1\times1 & 256 \end{bmatrix}\times 3$	$\begin{bmatrix} 1\times1 & 64 \\ 3\times3 & 64 \\ 1\times1 & 256 \end{bmatrix}\times 3$
Conv3_x	28 × 28	$\begin{bmatrix} 3\times3 & 128 \\ 3\times3 & 128 \end{bmatrix}\times 2$	$\begin{bmatrix} 3\times3 & 128 \\ 3\times3 & 128 \end{bmatrix}\times 4$	$\begin{bmatrix} 1\times1 & 128 \\ 3\times3 & 128 \\ 1\times1 & 512 \end{bmatrix}\times 4$	$\begin{bmatrix} 1\times1 & 128 \\ 3\times3 & 128 \\ 1\times1 & 512 \end{bmatrix}\times 4$	$\begin{bmatrix} 1\times1 & 128 \\ 3\times3 & 128 \\ 1\times1 & 512 \end{bmatrix}\times 8$
Conv4_x	14 × 14	$\begin{bmatrix} 3\times3 & 256 \\ 3\times3 & 256 \end{bmatrix}\times 2$	$\begin{bmatrix} 3\times3 & 256 \\ 3\times3 & 256 \end{bmatrix}\times 6$	$\begin{bmatrix} 1\times1 & 256 \\ 3\times3 & 256 \\ 1\times1 & 1024 \end{bmatrix}\times 6$	$\begin{bmatrix} 1\times1 & 256 \\ 3\times3 & 256 \\ 1\times1 & 1024 \end{bmatrix}\times 23$	$\begin{bmatrix} 1\times1 & 256 \\ 3\times3 & 256 \\ 1\times1 & 1024 \end{bmatrix}\times 36$
Conv5_x	7 × 7	$\begin{bmatrix} 3\times3 & 512 \\ 3\times3 & 512 \end{bmatrix}\times 2$	$\begin{bmatrix} 3\times3 & 512 \\ 3\times3 & 512 \end{bmatrix}\times 3$	$\begin{bmatrix} 1\times1 & 512 \\ 3\times3 & 512 \\ 1\times1 & 2048 \end{bmatrix}\times 3$	$\begin{bmatrix} 1\times1 & 512 \\ 3\times3 & 512 \\ 1\times1 & 2048 \end{bmatrix}\times 3$	$\begin{bmatrix} 1\times1 & 512 \\ 3\times3 & 512 \\ 1\times1 & 2048 \end{bmatrix}\times 3$
Classifier	1	Global Average Pooling Fully Connected Layer, 1000, Softmax Output label				



(Normal Control / NC)



(Mild Cognitive Impairment / MCI)



(Alzheimer's Disease / AD)

Figure 4.1 Example the subjects in different stages.

Figure 4.2 Denoised MRI scan.

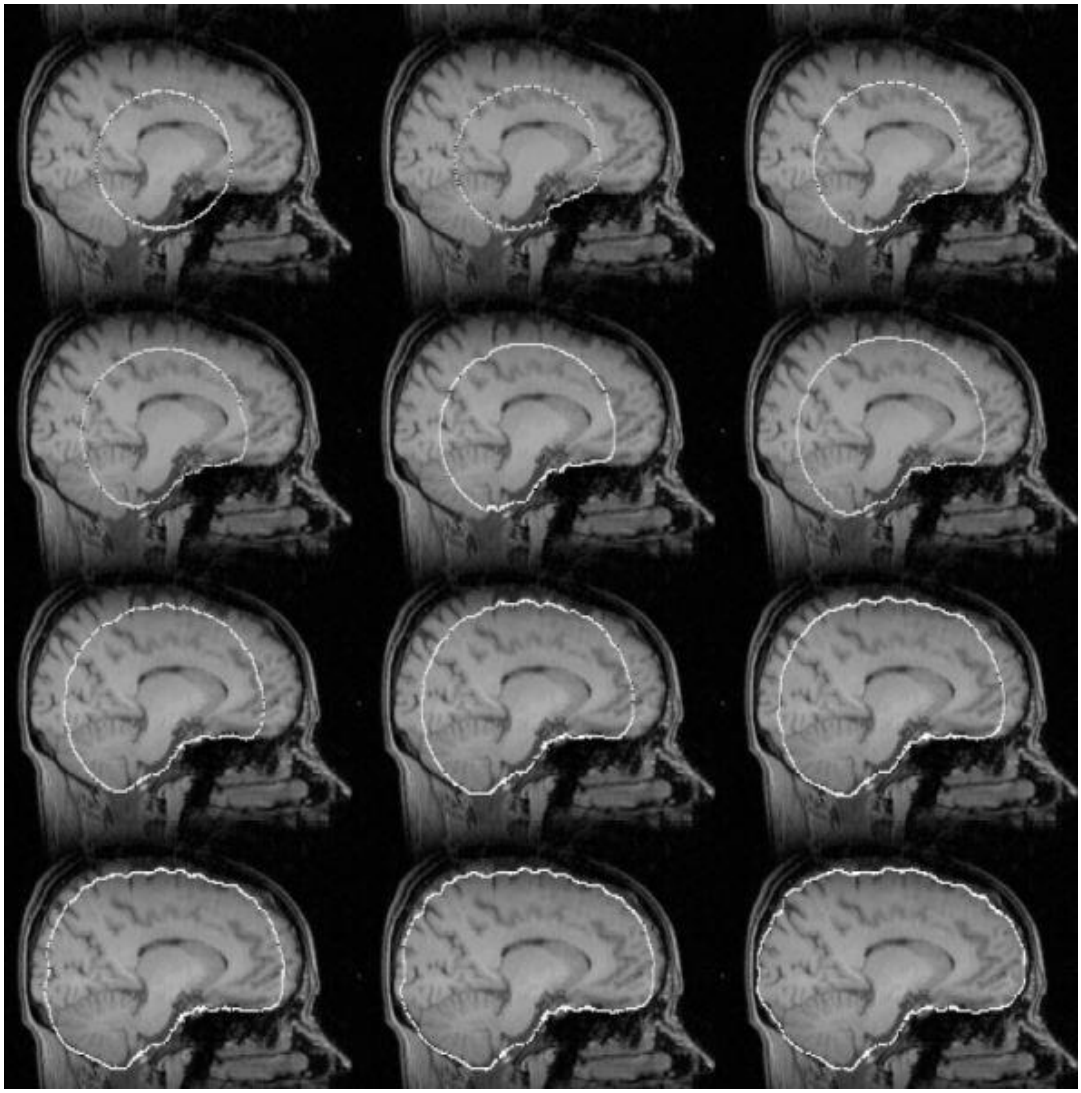


Figure 4.3 Demonstration of brain extraction tool (BET; Smith, 2002).

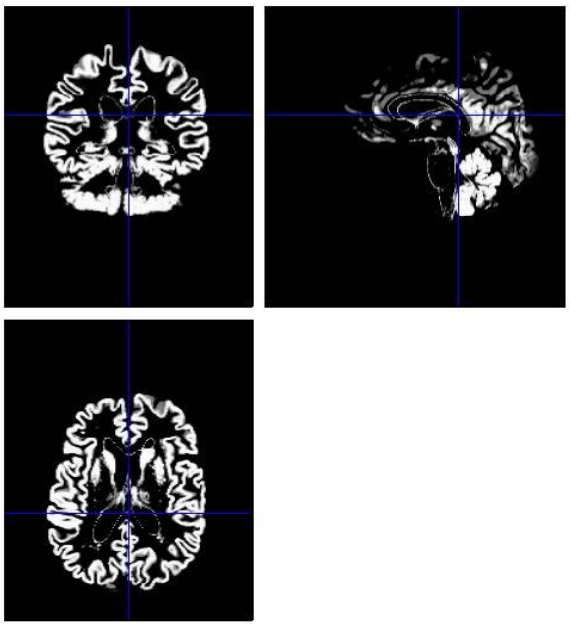


Figure 4.4 Isolated grey matter (top left: coronal plane; top right: sagittal plane; bottom left: axial plane).

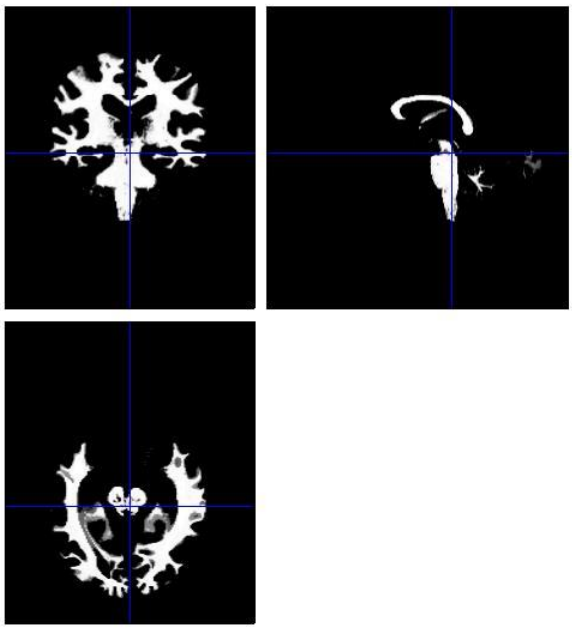


Figure 4.5 Isolated white matters (top left: coronal plane; top right: sagittal plane; bottom left: axial plane).

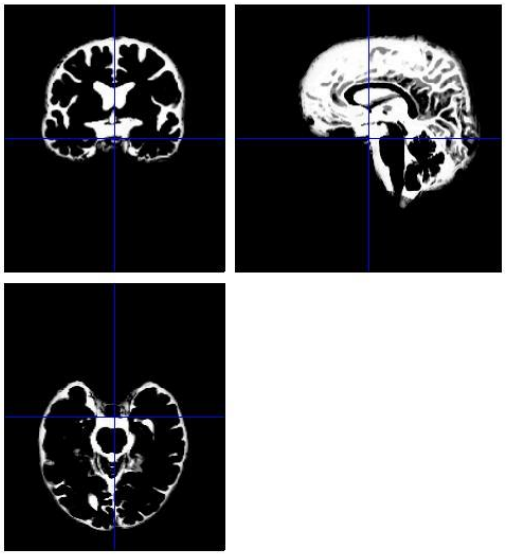


Figure 4.6 Isolated cerebral spinal fluid (top left: coronal plane; top right: sagittal plane; bottom left: axial plane).

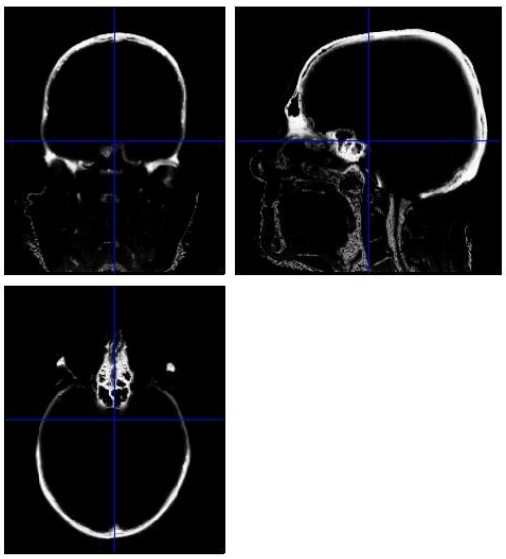


Figure 4.7 Isolated skull and other non-brain (top left: coronal plane; top right: sagittal plane; bottom left: axial plane).

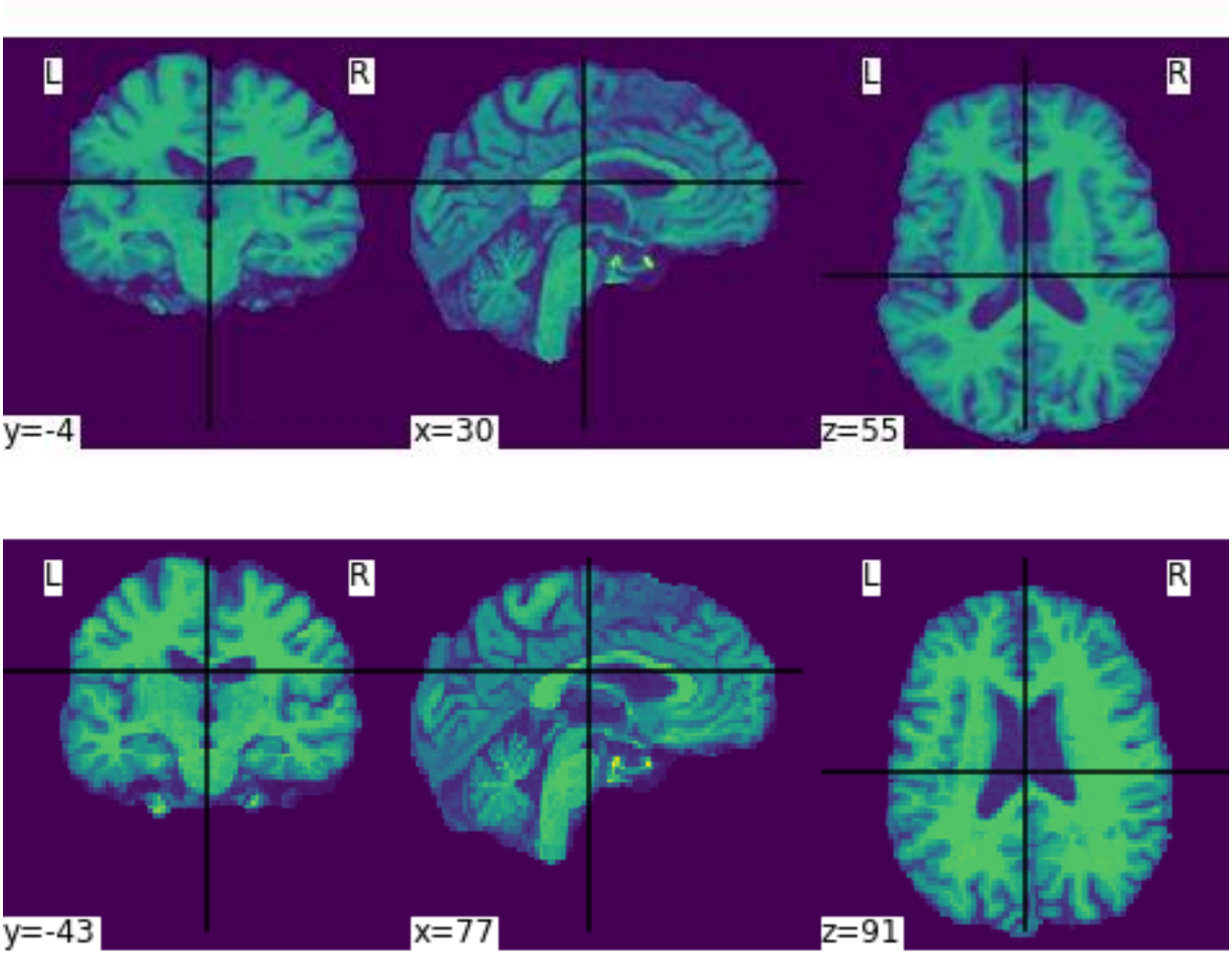


Figure 4.8 Effect of data processing. Original (top) versus processed (Bottom).

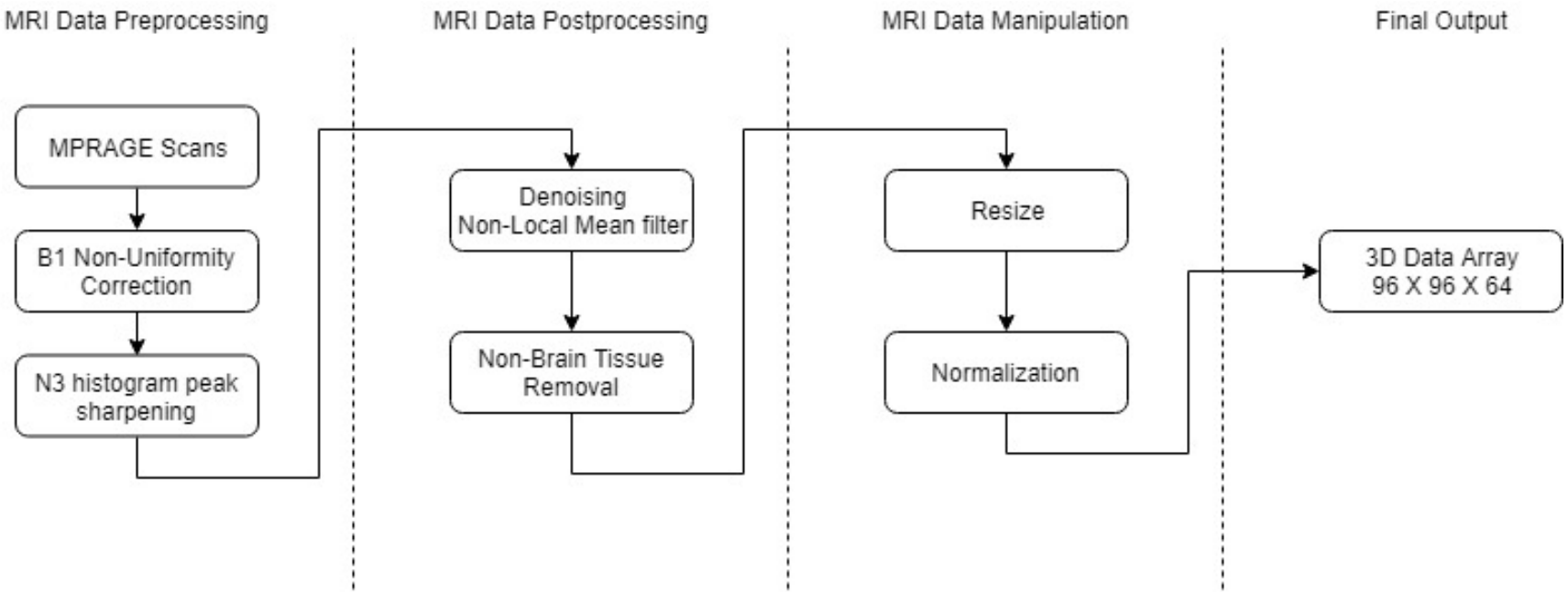


Figure 4.9 MRI data processing flowchart.

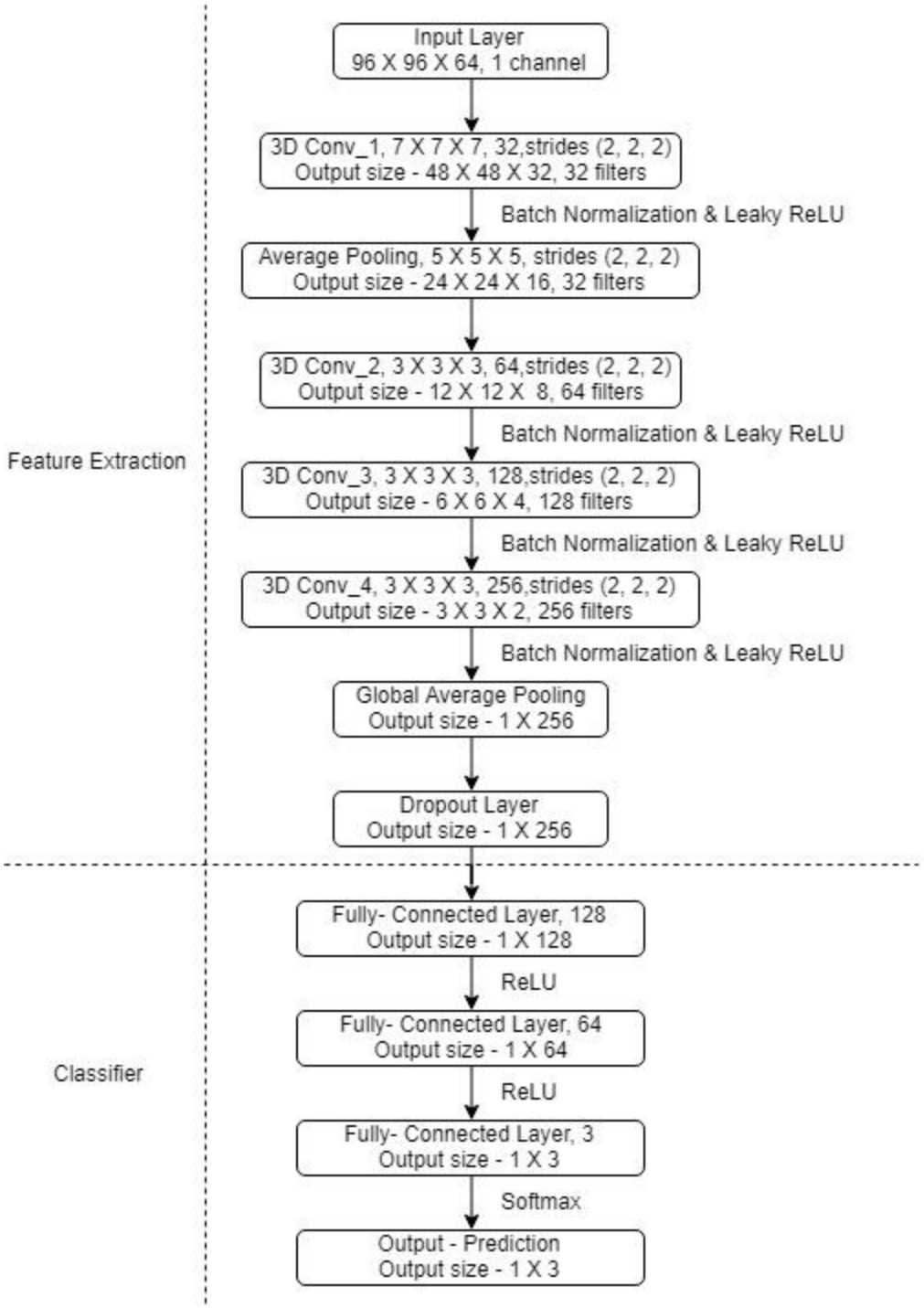


Figure 4.10 Model architecture for deep 3D neural network – Model A, with output shape of each layer listed in the parentheses.

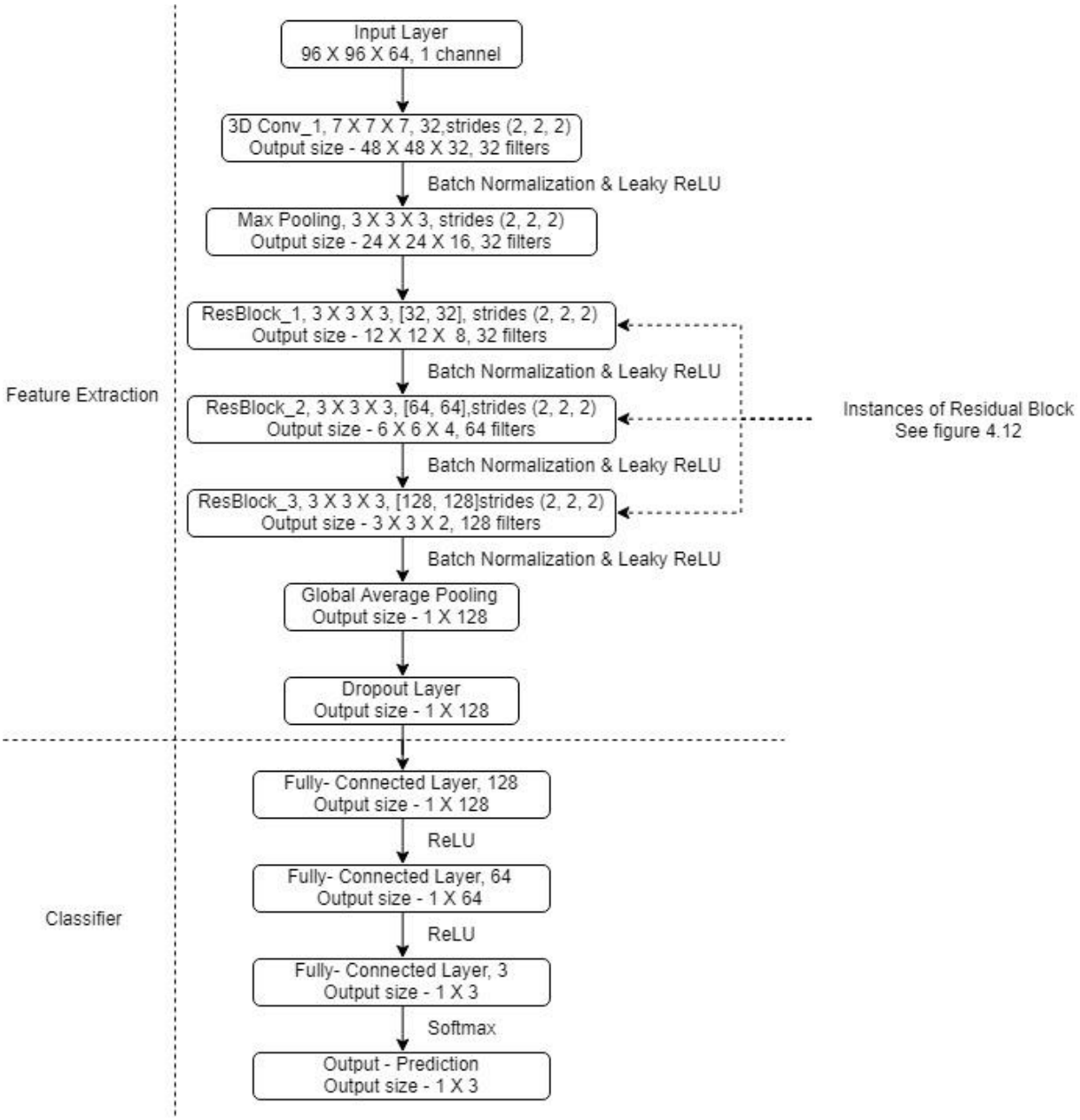


Figure 4.11 Model architecture for 3D Residual neural network – Model B, with output shape of each layer listed in the parentheses.

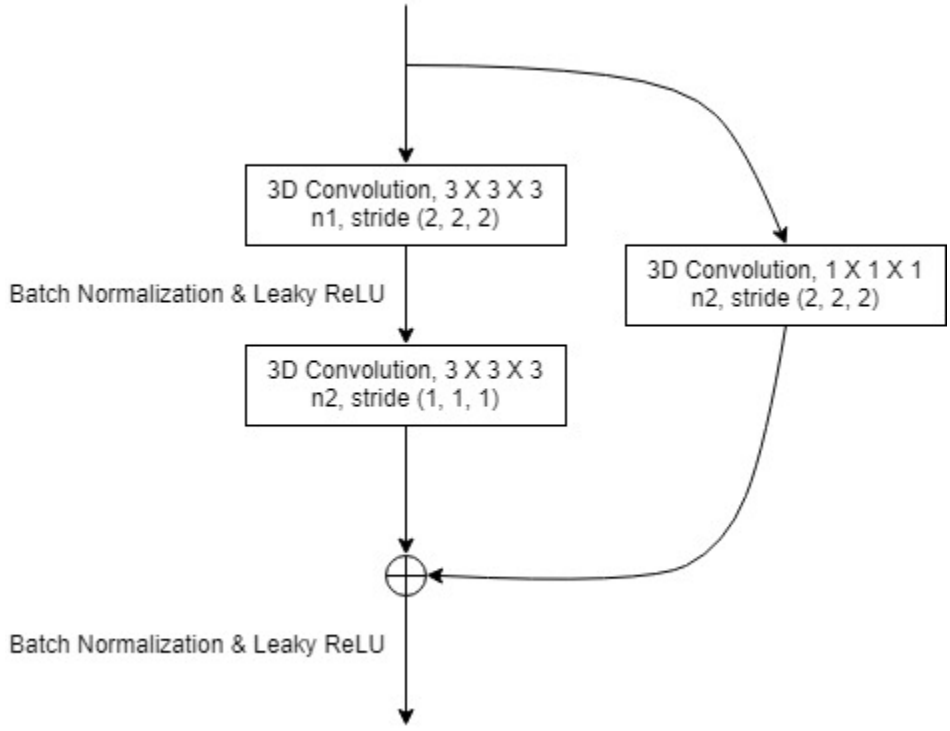


Figure 4.12 3D Residual Block, with n1 and n2 being the number of filters.

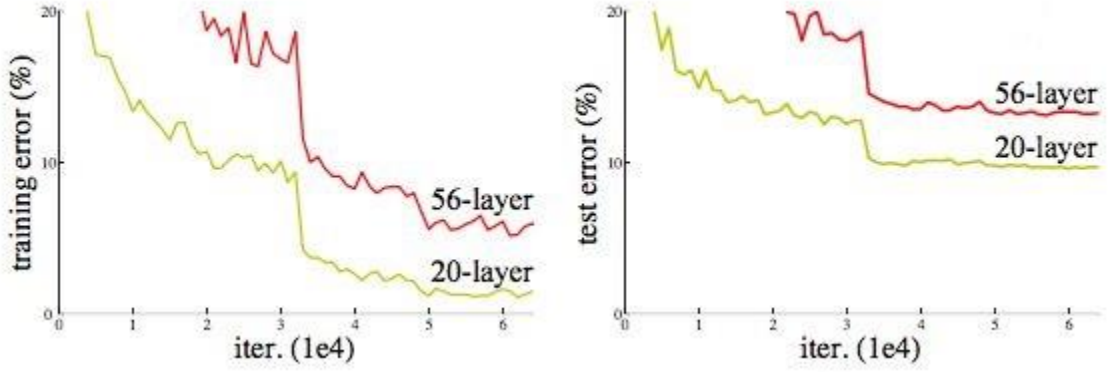


Figure 4.13 Error increased when simply increased depth (He et al., 2016).

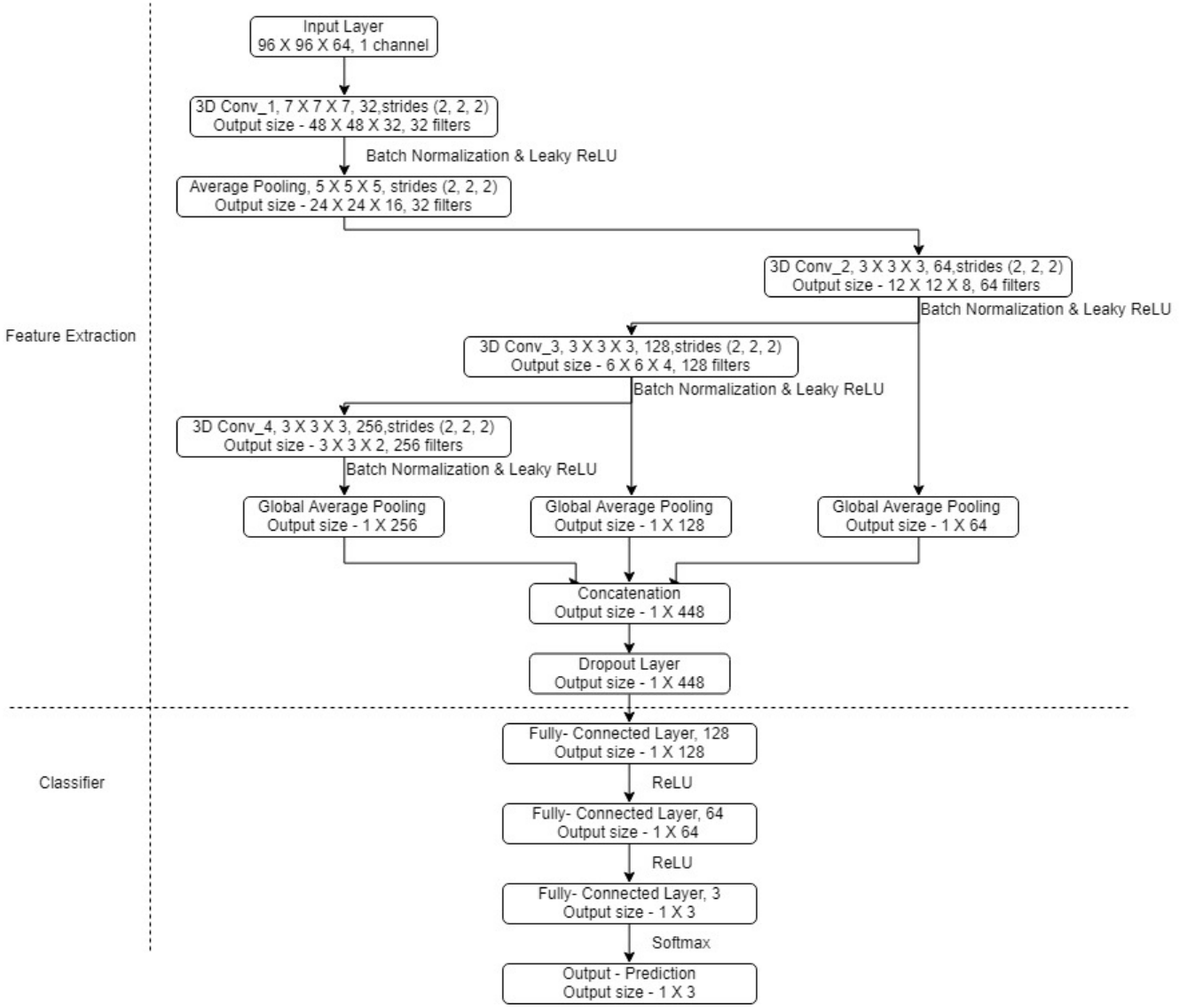


Figure 4.14 Deep 3D convolutional network with Multi-Layer-Output (Model C).

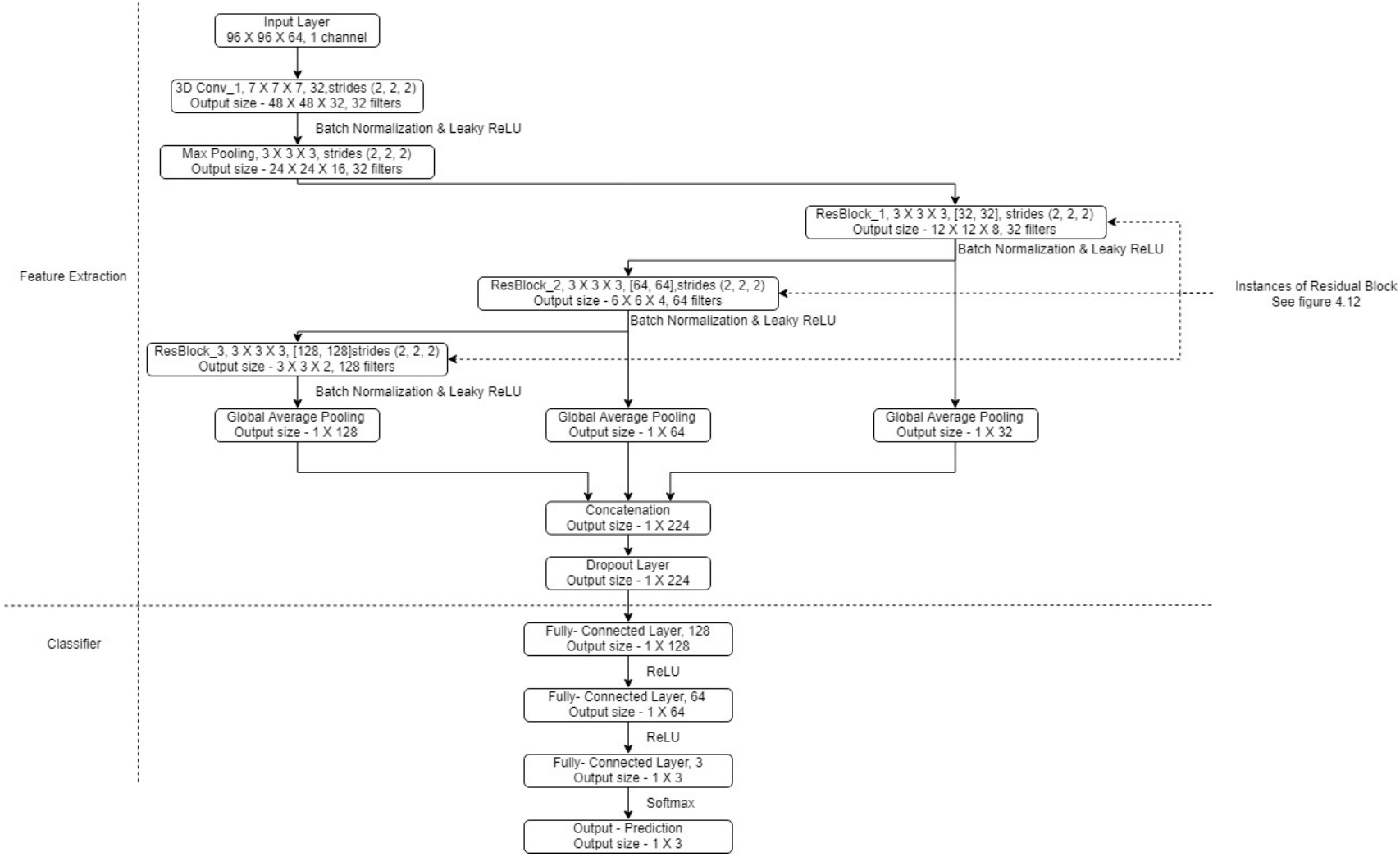


Figure 4.15 3D ResNet model with MLO (Model D)..

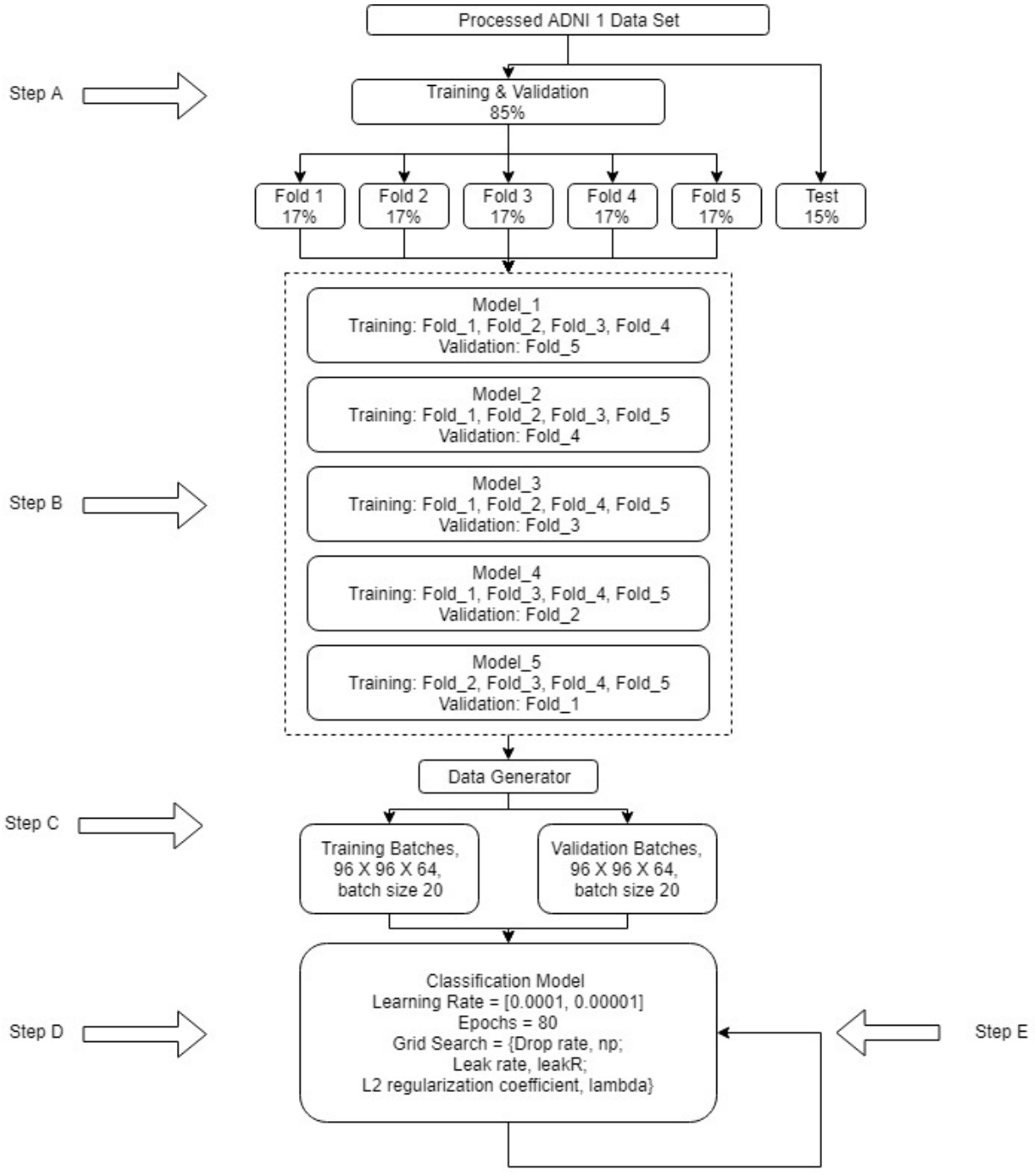


Figure 4.16 Demonstration of cross validation for AD diagnosis models.

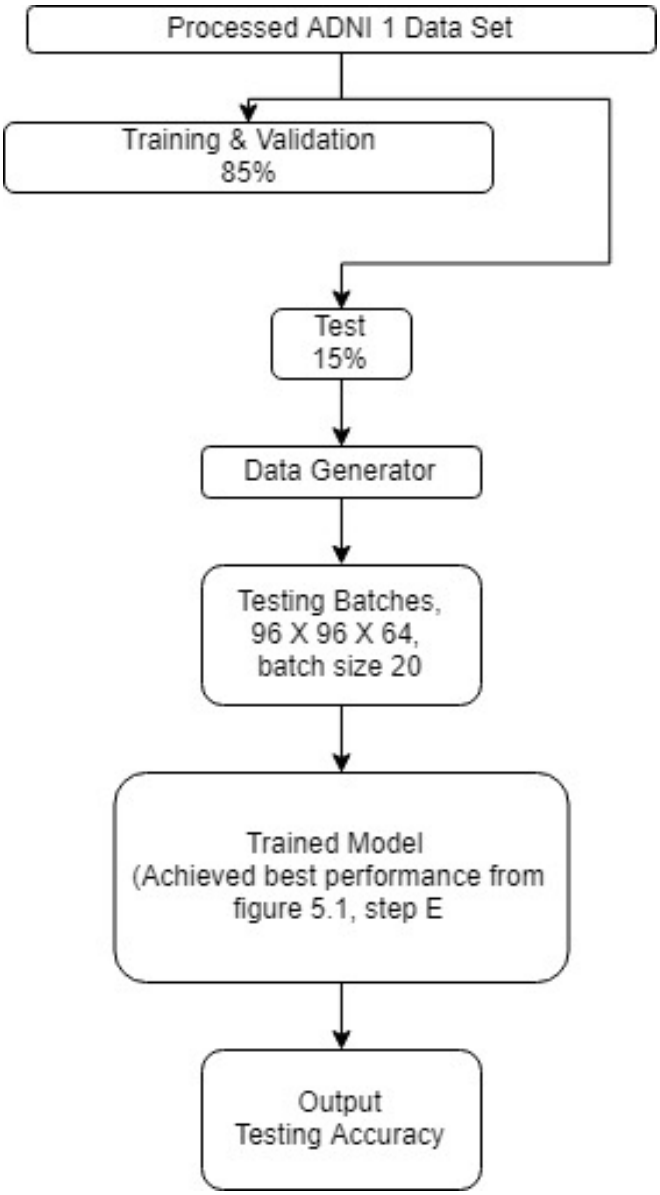


Figure 4.17 Demonstration of model testing.

Figure 4.18 Typical indoor scene from LSUN dataset (Yu et al., 2015).

Figure 4.19 Typical indoor scene from MIT Indoor67 dataset (Quattoni & Torralba, 2009).

Figure 4.20 Typical indoor scene from 15-Scene dataset (Fei-Fei & Perona, 2005; Lazebnik, Schmid, & Ponce, 2016; Oliva & Torralba, 2001).

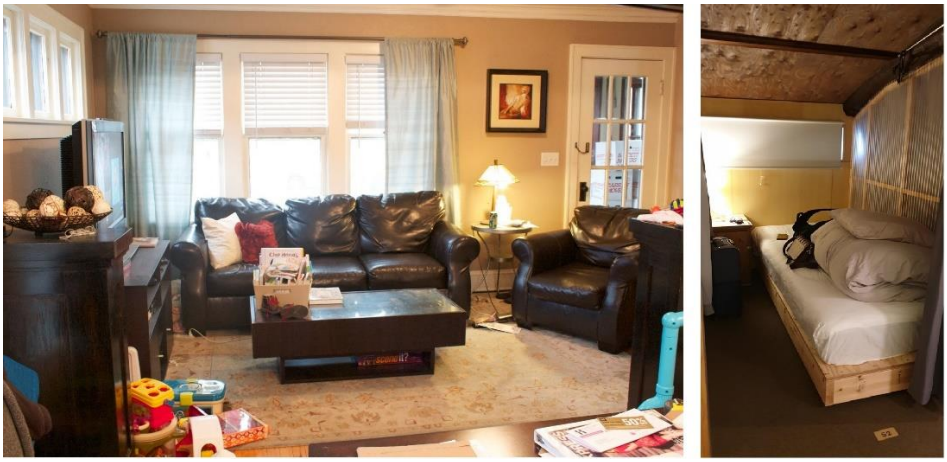


Figure 4.21 Examples from self-collect dataset.

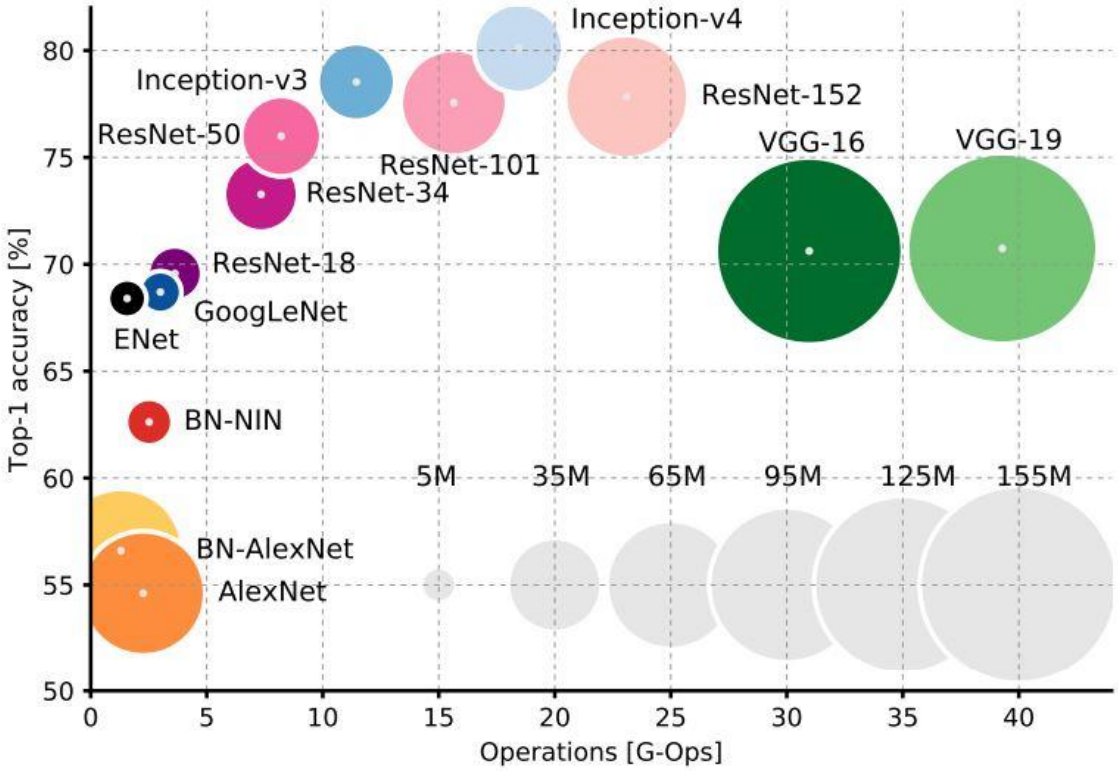


Figure 4.22 Model comparison on top-1 accuracy, number of operations and model size (Canziani, Paszke, & Culurciello, 2016).

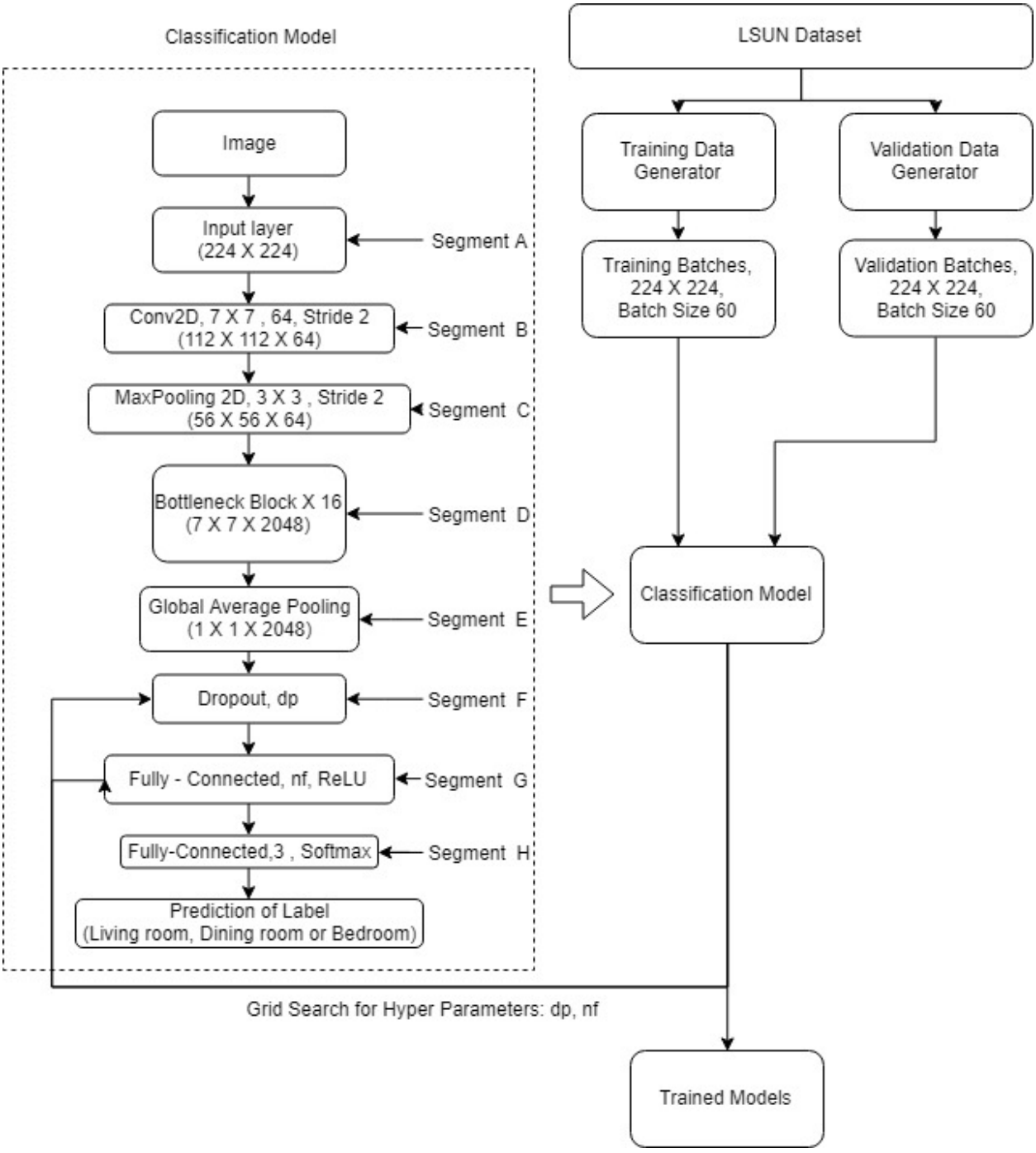


Figure 4.23 Training flowchart for scene understanding.

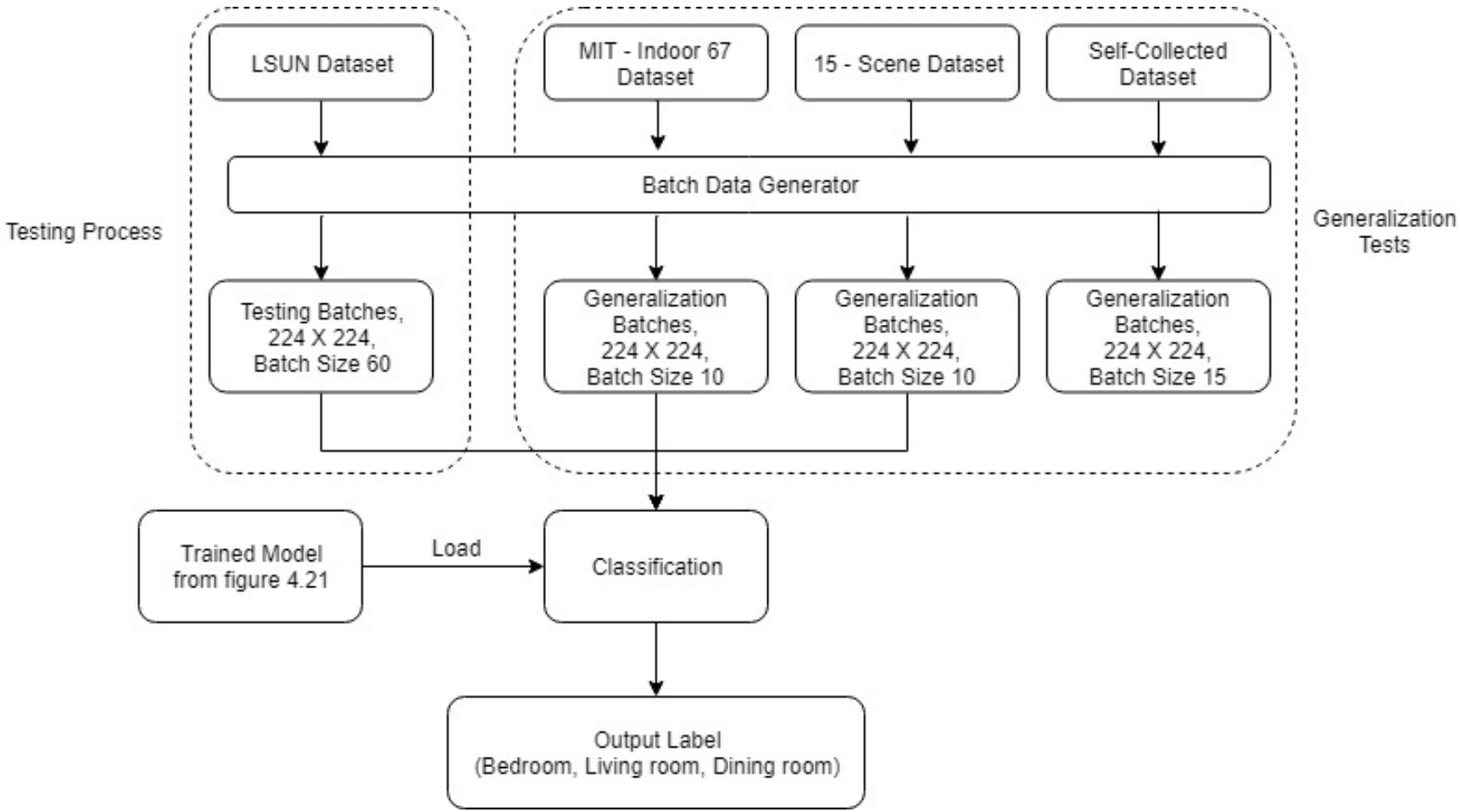


Figure 4.24 Testing flowchart for scene understanding.

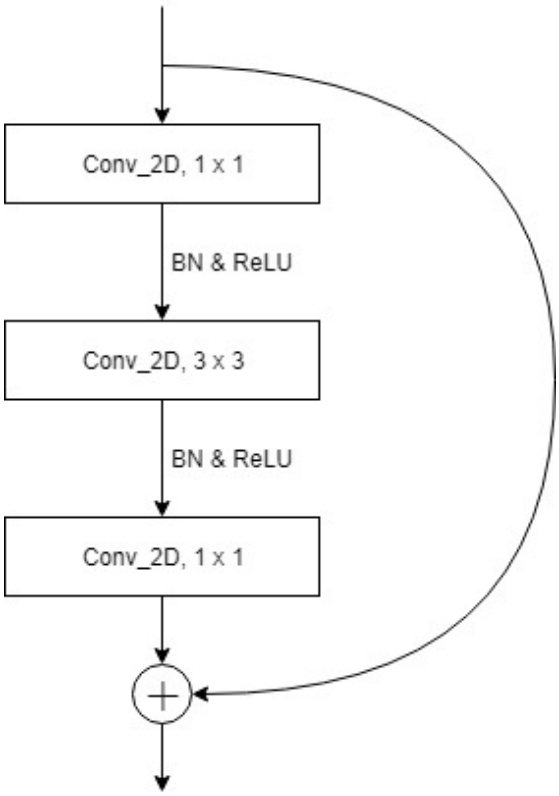


Figure 4.25 Bottleneck residual block.

Figure 4.26 Comparison between showroom image (top) and real-life image (bottom).

CHAPTER 5. RESULT AND ANALYSIS

5.1 Development Environment

The model proposed in this work were developed and tested in Python 3.6.6. The major supporting development libraries included: Tensorflow 1.12.0 (Abadi et al., 2016), Keras 2.2.4 (Chollet, 2015), Scikit-Learn 0.20.1 (Pedregos et al., 2011), NumPy 1.12.1 (Oliphant, 2006), and Nibabel 2.3.1 (Brett et al., 2018). The front-end independent development environment was Spyder 3.2.4 under Window 10 Education operating system.

With the courtesy of Purdue University Rosen Center for Advanced Computing (RCAC), all the test and analysis were done on their "Gilbreth" back-end computation cluster. For every job submitted, two computation nodes were requested which contained 384 GB of memory and two Nvidia Tesla P100 graphic computation units. This cluster was run with CentOS 7.6 operation system. The graphic computation was supported by CuDA 9.0.176 (Nvidia, 2010) development kit and CuDNN 9.0 deep neural network optimization library (Chetlur et al., 2014).

5.2 Experiments Results Deep 3D Convolutional Neural Network for AD classification

5.2.1 Grid Search Result

As described in Section 4.1.4, grid search with 5-fold cross validations were conducted to find the best hyper-parameter for the model. The search range for drop rate (*dp*) was [0.1, 0.7] with a step size of 0.1. The search range for L2 coefficient, λ, was [0.01, 0.05] with a step size of 0.01. The search range for leak rate (*leakR*) was [0.01, 0.03] with a step size of 0.01. The total size of ADNI1 dataset was 1,725. Fifteen percent of the data were reserved for testing, which was $1725 * 0.15 = 258.75 \approx 259$ scans. The rest of 1467 scans were used for a 5-fold cross validation. The detail results were reported later.

5.2.1.1 Deep 3D Convolutional Neural Network – Model A

As listed in Tables 5.1 to 5.7, the average accuracies of Model A in 5-fold cross validations were reported. The highest average validation accuracy was 77.1%, achieved with *dp* = 0.1, *leakR* = 0.02 and λ = 0.02. Although, the leak rate and the L2 coefficient didn't showed significant impact on the performance. However, an obvious decreasing trend was observed in the accuracy when the drop rate increased. The accuracies started to dramatically decrease (from around 75.0% to around 55.0%) when drop rate was set to greater than 0.3. The final model, whose *dp* = 0.7, showed accuracies only between 50% and 60%, and an overall minimum accuracy of 50.0%. This minimum performance occurred when *dp* = 0.7, *leakR* = 0.02 and λ = 0.01. The training and validation performance of the best model was shown in Figure 5.1.

5.2.1.2 Deep 3D Residual Neural Network – Model B

As listed in Table 5.8 to 5.14, the average accuracies of model B in cross validations were reported. The highest accuracy was 72.5%, achieved with *dp* = 0.1, *leakR* = 0.01 and λ = 0.01. Throughout the entire set of validations, no obvious trend in validation accuracy could be observed. The majority of the validation accuracies were around 70%. The lowest accuracy was 62.5%, achieved with *dp* = 0.6, *leakR* = 0.03 and λ = 0.05. The training and validation performance of the best model was shown in Figure 5.2.

5.2.1.3 Deep 3D Convolutional Neural Network with Multi-Layer-Output – Model C

As listed in Tables 5.15 to 5.21, the average accuracies of Model C in cross validations were reported. The highest accuracy was 80.4%, achieved with *dp* = 0.1, *leakR* = 0.03 and λ = 0.03. Validation accuracies started to decrease when the drop rate was greater than 0.3. The training and the validation performance of the best model was shown in Figure 5.3.

5.2.1.4 Deep 3D Residual Neural Network with Multi-Layer-Output –Model D

As listed in Tables 5.22 to 5.28, the average accuracies of Model D in cross validations were reported. The highest accuracy was 82.0%, achieved with *dp* = 0.2, *leakR* = 0.01 and λ = 0.04. Validation accuracies showed no significant trend when changing hyper-parameters. The majority of the models showed validation accuracies around 78%. The training and the validation performance of the best model was shown in Figure 5.4

5.2.2 Testing Results

Based on the cross-validation results, four best models were selected from the grid search cross validation. These four models were:

- Model A with *dp* = 0.1, *leakR* = 0.02 and λ = 0.02. Average validation accuracy 77.1%.
- Model B with *dp* = 0.1, *leakR* = 0.01 and λ = 0.01. Average validation accuracy 72.5%.
- Model C with *dp* = 0.1, *leakR* = 0.03 and λ = 0.03. Average validation accuracy 80.4%.
- Model D with *dp* = 0.2, *leakR* = 0.01 and λ = 0.04. Average validation accuracy 82.0%.

The same testing dataset was used to evaluate the classification accuracies of these models. The detailed performance was shown in Table 5.29. Additionally, six rounds of testing were conducted to find out the average testing performance (the result for these additional tests were shown in Appendix C).

Overall, on the testing set that contained 259 scans (88 NC, 117 MCI and 54 AD), the testing accuracies were summarized (The detailed confusion matrices for these tests can be found in Figures 5.5 through 5.8):

Model A achieved 74.9% multiclass classification accuracy. For in-class accuracy, 61 out of total 88 NC (69.3%), 92 out of total 117 MCI (78.6%) and 41 out of total 54 AD (75.9%) were

classified correctly. For additional tests, the average accuracy was 75.4% and the standard deviation was 0.8%.

Model B achieved 71.4% multiclass classification accuracy. For in-class accuracy, 59 out of total 88 NC (67.0%), 89 out of total 117 MCI (76.1%) and 37 out of total 54 AD (68.5%) were classified correctly. For additional tests, the average accuracy was 69.6% and the standard deviation was 1.4%.

Model C achieved 79.9% multiclass classification accuracy. For in-class accuracy, 74 out of total 88 NC (84.1%), 93 out of total 117 MCI (79.5%) and 40 out of total 54 AD (74.1%) were classified correctly. For additional tests, the average accuracy was 78.0% and the standard deviation was 1.1%.

Model D achieved 81.5% multiclass classification accuracy. For in-class accuracy, 72 out of total 88 NC (81.8%), 102 out of total 117 MCI (87.2%) and 37 out of total 54 AD (68.5%) were classified correctly. For additional tests, the average accuracy was 80.5% and the standard deviation was 1.4%.

5.2.3 Analysis and Discussion

5.2.3.1 Grid Search

In this work, the proposed models were first trained and validated on 85% of the entire ADNI1 dataset. There were 105 different variation for each model. This brought the total amount of models to 105 × 4 = 420 and the total amount of training to 420 × 5 (5-fold cross validation) = 2100. Several observations can be made from these 5-fold cross validation result.

1. Between native models (Model A and Model B)

Model A (7 hidden layers) and model B (13 hidden layers) were designed to explore the effect of simply increasing the depth of the network. Based on the results listed in Tables 5.1 to

5.14, the best average validation accuracy achieved by Model A was 77.1% which was better than that of model B (72.5%). Evidently, this suggested that, for this particular dataset, deeper network did not necessarily extract better feature than shallower network.

However, another interesting fact was that as the drop rate increased, Model A showed significant decreased classification accuracies (from around 75% to around 55%). Meanwhile, Model B showed no such significant drop (from around 70% to around 67%). This indicated that even though the features extracted by Model B were not very significant, the majority of them were still related to the classification task. This relation could explain why penalizing the features would not affect the classification much. On the other hand, for Model A, the feature representations were more likely to contain several significant features while the rest of them were mostly nuisance to the task. Therefore, when the significant features got penalized, the rest of the features could not support the same classification performance as before. In a word, even though the high-level features extracted by the deeper network was not necessarily better, it could still serve as strong supporting features to the classification task.

2. Between native models and their variations (Model A/C, Model B/D)

Compared with the two native models, the two models with MLO showed an observable improvement on the classification accuracy (A/C: from 77.1% to 80.4%, B/D: from 72.5% to 82.0%). Especially, the improvement for Model D was very significant. As we suspected earlier, the high-level features from deeper network may not be very discriminative but can still serve as supporting features. In Model D, feature representations were created every two convolutional layers and concatenated together. Because of this two layers between each representation, more feature transformations were completed and representation was more refined. This could be the reason of why the performance showed better improvement than Model C.

3. Between MLO models (Model C and Model D)

Model C and Model D were the advanced modification of their native models (Model A and Model B). The idea was to examine if combining high-level features with low-level features could improve the classification accuracies. The results listed in Tables 5.15 to 5.28 indicated an observable improvement on the metrics. Unlike their native models, Model D showed better performance (82.0%) than Model C (80.4%). This result supported the previous proposition that high-level and low-level features should be combined. Both models showed improved accuracy when combining different levels of representation, especially for deeper networks. We can then speculate that the deeper network may not be better at producing final feature representation, but did perform better feature extraction throughout the entire process.

5.2.3.2 Testing

After the hyper-parameter were selected, the models were tested with the remaining 15% of data (259 scans, 88 NC, 117 MCI, and 54 AD) for overall and in-class accuracy. Model D showed the best performance (81.5%) and Model C achieved the second (79.9%). Both of the models showed improvements compared with their original models (5% improvement between Model A/C, and 10% improvement between Model B/D). Also, although misclassification happened in all models, it was noticed that the misclassification was more likely to happen between classes with similarity. In all models, when NC subjects were wrongly classified, they were majorly classified as MCI and not AD. Similar thing happened to MCI subjects and AD subjects. MCI misclassifications were majorly NC and AD misclassifications were majorly MCI. Considering the gradual structural changes when AD progress, these results suggested that the model successfully captured a moderately discriminative feature representation.

For AD classification, all models showed similar accuracies (37-41 correct classifications out of 54). This could be a sign of model underfitting (the feature cannot fully describe the characteristic of the class). However, this was considerably understandable because there were only 346 AD subjects in the entire dataset. Such characteristic might not be presented in the dataset in the first place.

For NC and MCI classification, models showed significant improvements when adopted MLO structures. For Model A/C, the in-class accuracy boosted from 69.3% (NC) and 78.6% (MCI) to 84.1% (NC) and (79.5%). For Model B/D, the in-class accuracy boosted from 67.0% (NC) and 76.1% (MCI) to 81.8% (NC) and 87.2% (MCI). These results suggested that when combining low-level and high-level features, the final feature representation might have better ability to interpret the data.

Compared with other popular works, the proposed models did not showed improvements. Billones et al. (2016) used 2D slices of MRI scan (around hippocampus) in a 17-layer convolutional neural network and achieved 91.8% accuracy on three-way classification (AD/MCI/NC). Hosseini-Asl et al.(2016) first extracted preliminary features using an autoencoder, then performed three-way classification through a 6-layer convolutional neural network. The classification results reached 89.1%. Based on their results and the results obtained in this work, several conjectures can be made: First, even though the models developed in this work did not achieve very high performance, they still achieved an accuracy greater than 81%. The proposed Multi-Layer-Output (MLO) structure significantly increased the classification accuracies of the conventional models. The standard models (model A and model B) when trained without MLO, only showed 74.9% and 71.4% respectively. After adopted the MLO architecture, the models (model C and model D) showed 5% and 10% improvement respectively,

on the overall classification accuracies (model C - 79.9% and model D - 81.5%). The advancement was, in model C and D, the features extracted at different levels were concatenated to the final feature representation. The models were them be able to learn much more comprehensive features and therefore led to a significant performance improvement. This architecture demonstrated a foreseeable potential in refining the model. For future work, instead of trying to increase the depth, it is worthwhile to explore how to optimize the model's topology so that it could use the evidence to the fullest.

Secondly, deep learning was not necessarily better than other shallow learning methods in this field of research. Deep learning is known for being an end-to-end methodology, which means to draw final conclusions from raw input. However, the collection of AD evidence is known for being notoriously difficult and costly. ADNI, being the largest organization for conducting such data collection, is well-funded by the National Institute of Aging (NIA) and National Institute of Health (NIH) and only recruited about 2,000 subjects in their 15-year study. Given this limited data size, deep learning models are not likely to be able to effectively refine the model and perform accurate classification.

Lastly, instead of looking at the whole brain, methods that narrow the region of interest seem to have better performance. Billones et al. (2016) only focused on the slices that contained the hippocampus. Other research has used the density of brain matter as the evidence and achieved good results on binary classification (Davatzikos et al., 2008; Good et al., 2002; Lao et al., 2004; Misra et al., 2009). These methods were all inspired by neurological observation. Therefore, it might be helpful to obtain advice from medical professionals when developing models.

5.3 Indoor Scene Understanding

5.3.1 Cross Validation

In this experiment setup, the average training time for one epoch is 274s. The total training time for each 5-fold validation test is about 4 hours. As shown in the Table 5.30, the average training accuracy for the five LSUN subsets were, 99.8%, 99.8%, 99.8%, 99.7%, and 99.7%. The total average training accuracy is 99.8% Also, the average testing accuracy for the five LSUN subsets are 94.3%, 94.1%, 94.4%, 94.6%, and 93.9% respectively. The total average testing accuracy is 94.3%. Finally, the average in-class accuracies for these three scene categories are 95.6%, 95.2%, and 91.9%.

The generalization tests (testing model with unseen datasets) were also conducted using the trained models (Table 5.31). The average test accuracy for the MIT Indoor67 was 87.7% with in-class accuracies being 81.2%, 91.7%, and 90.0%. The average test accuracy for the 15-Scene was 89.7% with in-class accuracies being 96.3% (Bedroom) and 84.8% (Dining Room).

5.3.2 Grid Search

Mentioned in Section 4.2.2, grid searches were conducted to find the optimal hyper-parameter settings. The results were shown in Figures 5.5 through 5.7, Tables 5.32 to 5.34.

In summary, on the LSUN subset, as shown in Table 5.32, the testing accuracies ranged from 95.2% (achieved with dp = 0.1 and nf = 100), to 96.7% (achieved with dp = 0.7, and nf = 500). The best achieved accuracies were around 96.7%. Generally, an obvious improvement on the accuracies can be observed when the drop rate increased (Figure 5.5). The highest accuracies occurred when the hyper-parameters were set to (0.7, 500), (0.6, 100), and (0.7, 100).

On the MIT Indoor67 dataset, as shown in Table 5.33, the testing accuracies ranged from 86.7% (achieved with dp = 0.1, nf = 500), to 92.4% (achieved with dp = 0.7, nf = 100). From

Figure 5.6, no obvious performance adjustment could be observed when changing hyper-parameter. However, the model did show good performance when the drop rate was greater than 0.2 and *nf* was less than or equal to 500.

On the 15-Scene dataset, as shown in Table 5.34, the testing accuracies ranged from 87.7% (achieved with *dp* = 0.1, *nf* = 100), to 95.6 % (achieved with *dp* = 0.4, *nf* =500 and *dp* = 0.5, *nf* = 100). As shown in Figure 5.7, the model showed particular poor performance when the drop rate was set to be less than 0.3. As long as the drop rate was greater than or equal to 0.3, the model achieved 95% accuracies with no obvious fluctuation.

Aggregated the testing results from the three datasets (LSUN, MIT Indoor67 and 15-Scene), we could conclude that for this particular model to achieve good classification performance, the drop rate of the model should be greater than 0.2 and the *nf* should be less than or equal to 500. Among all the combinations, the model with *dp* = 0.7 and *nf* =100 seemed to achieve the best performance on all three datasets. Therefore, this set of hyper-parameters was selected for the final model training.

5.3.3 Testing Results

After the hyper-parameter was selected, the model was fine-tuned with the entire LUSN dataset. Mentioned in Section 4.2.3, the model was trained with three million images (LSUN dataset) and validated with one million images (LSUN dataset). Upon completion of the training, the model was tested with the remainder of LSUN dataset (one million images). Additionally, three generalization tests (MIT Indoor67, 15-Scene, self-collected) were conducted. The results were summarized in Table 5.35.

As we can observe the proposed model achieved 97.2% overall accuracy on the LSUN testing set. It also achieved 93.8% and 96.0% accuracy on the generalization test sets. For in-

class accuracies (bedroom vs living room vs dining room), the model achieved 97.8%, 96.9%, and 95.4% on LSUN dataset. The model also achieved 91.3%, 91.7%, and 98.6% on MIT Indoor67 dataset. Finally, the model achieved 98.6% (bedroom) and 91.1 % (dining room) on the 15-Scene dataset. The detailed confusion matrixes on these three tests can be found in Figure 5.8. As for the test with self-collected real scenes, the model achieved 100% accuracy on the classification task. The model output of these 15 images were listed in Table 5.36. In short, the model predicted likelihood of the input image belong to each of the classes. Obviously, the model successfully classified the images into their ground truth class with extreme confidence (mostly more than 99% of confidence).

5.3.4 Analysis and Discussion

In these experiments, a set of small-scale experiments were first conducted to examine the validity of the model. By using small but balanced datasets parallelly, the proposed model was rapidly evaluated for its ability to converge on the target classification task. The model showed perfect convergence on the training sets but also a slightly decrease of approximately 5% on the testing sets. Furthermore, the generalization tests showed an acceptable performance but still decreased slightly from the original testing accuracies. These phenomena indicated that:

- The model showed excellent discriminant power on the proposed classification task by capturing significant features.
- The small decrease on the testing accuracies indicated that the model also captured a small proportion of features that were not relevant to the task but related to the bias came with the datasets.
- Such overfitting problem needed to be further reduced through hyper parameter tuning and increasing the dataset size.

From there, totally 21 grid search experiments were conducted to examine how tuning the hyper parameters could affect the classification performance. The model was tested with different combinations of drop rate and number of filters in the classifier, under three different datasets (LSUN, MIT Indoor67, and 15-Scene). This resulted in 63 measured metrics for performance evaluation. To goal is to find an optimal combination that would demonstrate excellent discrimination ability on all datasets. From Table 4.6, we can observe that the model showed very diminutive fluctuation when changing the number of filters in the classifier. However, changing drop rate showed great impact on the performance. Under all three datasets, the accuracies appeared to increase when the drop rate was increased. This phenomenon was particular significant on the generalization tests. When drop rate was changed from 0.1 to 0.7, the LSUN tests gained about 1% accuracy while both of the generalization tests gained about 4% accuracy. This proved that even though the model could generalize well on the unseen data from its own dataset (LSUN), it failed to achieve the similar ability on the two other datasets (MIT Indoor67, 15-Scene). Consequently, the majority of the features learnt were relevant to the task while a limited proportion of the features were the learnt bias noise from the dataset. And this bias noise was what causing the accuracies to drop on the two other tests. Nevertheless, such bias noise was inevitable in machine learning model, given that human even sometimes do the same on cognition.

To thoroughly test the performance of the model, the model was finally trained with the entire LSUN dataset (three million images) with optimal obtained hyper-parameter. As a result, the model showed 97.2% accuracy on the LSUN test set, 93.8% accuracy on the MIT Indoor67 test set and 96.0% on the 15-Scene test set. These accuracies were slightly better than the results obtained with LSUN subset. Several other works used the same dataset. Uršič, Leonardis, and

Kristan (2016) reached 85.15% on the MIT Indoor67 dataset. Zhou et al (2014) reached 70.8% on the MIT Indoor67 and 91.6% on the 15-Scene. Needless to say, the results obtained in this work outperformed them by more than 5%.

These results demonstrated that the model's excellent ability to discriminate typical indoor room types. However, misclassification of room types still happened. One possible reason could be these room types may share very similar graphical features, but such variation was not fully captured by the dataset. For example, if chair appeared very often in dining room scenes and living room scenes but not in the bedroom scenes, a bedroom photo with a chair in it would be more likely to be classified as a living room or dining room because of that. However, such bias was created by the collection bias of the dataset. Therefore, to further improve the performance, the model should ideally be able to capture more scene-centric features instead of the object-centric features. Secondly, given that the ground truth labels were manually assigned by human, there were chances that the some of the images were wrongly labeled or images themselves were utterly impossible to label, causing the confusion on the model. Like the example in Figure 5.9, the images were labeled as living room while in my opinion should be labeled as bedrooms. In Figure 5.10, the room clearly served both as living room and dining room but only labeled with dining room.

Despite all the mentioned problems, this proposed model still yielded to a very good classification performance. Moreover, even though the training of the model consumed gigantic computation resources, deploying such model would require a fairly insignificant amount of resources. The goal of this research was to develop an indoor scene understanding model that would not only achieve acceptable accuracy but also be feasible to be implemented on portable devices. One of the concerns was the execution time. The average runtime for each prediction of

images using this trained model was 0.35 seconds on desktop machines, which could be considered as fast for real-time execution. Further testing was needed to determine the runtime on micro-controllers. Another concern was the complexity of deployment. Currently, the proposed model contained more than 25 million parameters and the trained model would take 195 MB of storage space. Such model could be easily deployed on any micro-controller with some effort to set up the run-time environment (see Section 5.1 for the required packages). As shown in Figure 5.15, for a given platform, the prediction can be done with a simple program. In such program, the previously trained model would be preemptively loaded and wait for the input from I/O. Once the image was captured, the model would perform the prediction based on this image. The suspected label of the image would then be produced and be further used by other routines in the system such as action prediction.

Table 5.1 Model A 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.1

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	76.7%	**77.1%**	76.3%	76.7%	75.4%
0.02	74.6%	74.6%	75.4%	75.4%	74.2%
0.03	74.2%	75.4%	75.4%	75.8%	72.5%

Table 5.2 Model A 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.2

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	75.4%	75.0%	75.4%	74.6%	73.8%
0.02	72.9%	76.3%	76.3%	75.8%	74.6%
0.03	73.8%	75.4%	74.2%	75.4%	75.4%

Table 5.3 Model A 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.3

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	75.4%	75.4%	75.0%	75.0%	74.6%
0.02	73.3%	75.8%	75.4%	74.6%	75.4%
0.03	71.7%	74.2%	74.6%	74.6%	74.6%

Table 5.4 Model A 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.4

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	70.4%	74.2%	75.0%	72.9%	70.0%
0.02	68.3%	72.5%	72.9%	74.6%	75.0%
0.03	72.1%	74.2%	69.6%	73.3%	75.8%

Table 5.5 Model A 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.5

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	66.7%	70.8%	74.6%	70.0%	73.3%
0.02	66.3%	66.3%	70.0%	72.9%	75.4%
0.03	69.2%	71.7%	72.5%	73.8%	72.5%

Table 5.6 Model A 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.6

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	62.9%	60.0%	62.5%	68.8%	63.3%
0.02	62.9%	65.4%	61.3%	64.2%	71.7%
0.03	62.1%	64.2%	62.1%	63.8%	67.5%

Table 5.7 Model A 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.7

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	52.9%	50.8%	55.0%	57.8%	57.1%
0.02	50.0%	55.4%	54.6%	56.7%	60.4%
0.03	53.8%	54.6%	55.8%	59.2%	59.2%

Table 5.8 Model B 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.1

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	**72.5%**	70.4%	66.7%	68.3%	69.2%
0.02	68.3%	69.6%	67.9%	68.3%	66.7%
0.03	69.6%	68.3%	69.6%	67.9%	69.6%

Table 5.9 Model B 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.2

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	68.3%	68.3%	66.3%	68.8%	66.7%
0.02	67.5%	67.1%	67.9%	70.4%	66.3%
0.03	66.3%	69.6%	65.8%	70.4%	65.4%

Table 5.10 Model B 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.3

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	70.4%	65.8%	65.4%	69.6%	67.9%
0.02	70.0%	65.8%	67.9%	67.9%	71.3%
0.03	67.1%	66.3%	68.3%	68.3%	63.3%

Table 5.11 Model B 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.4

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	67.1%	66.7%	71.3%	69.6%	68.8%
0.02	70.0%	68.3%	68.8%	67.9%	70.0%
0.03	70.4%	69.2%	71.7%	70.0%	66.3%

Table 5.12 Model B 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.5

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	69.6%	69.6%	67.1%	67.5%	68.8%
0.02	65.8%	66.3%	69.6%	70.4%	70.4%
0.03	67.1%	68.8%	69.6%	70.4%	68.8%

Table 5.13 Model B 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.6

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	69.6%	68.3%	64.2%	68.8%	67.9%
0.02	67.9%	68.3%	70.4%	68.8%	69.2%
0.03	71.3%	67.5%	69.6%	70.8%	62.5%

Table 5.14 Model B 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.7

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	67.9%	65.4%	65.4%	67.5%	68.8%
0.02	67.1%	65.4%	69.6%	68.8%	70.4%
0.03	67.9%	63.3%	67.9%	69.6%	67.5%

Table 5.15 Model C 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.1

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	77.5%	72.9%	77.5%	79.6%	78.8%
0.02	76.7%	75.4%	76.7%	77.9%	76.3%
0.03	74.2%	76.7%	**80.4%**	75.4%	76.3%

Table 5.16 Model C 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.2

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	77.1%	77.9%	75.8%	79.6%	76.3%
0.02	75.8%	74.2%	75.8%	76.3%	71.3%
0.03	76.7%	75.0%	76.7%	77.9%	77.1%

Table 5.17 Model C 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.3

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	77.5%	74.6%	74.2%	78.8%	76.7%
0.02	75.0%	76.7%	76.7%	77.1%	73.3%
0.03	74.2%	74.2%	76.7%	77.5%	77.9%

Table 5.18 Model C 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.4

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	75.8%	72.9%	72.1%	76.7%	76.7%
0.02	72.1%	72.5%	75.4%	75.8%	75.8%
0.03	75.0%	74.2%	73.8%	75.4%	75.0%

Table 5.19 Model C 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.5

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	70.8%	70.0%	75.4%	74.6%	73.8%
0.02	69.6%	67.5%	76.3%	75.0%	75.8%
0.03	69.2%	69.6%	67.1%	75.8%	74.6%

Table 5.20 Model C 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.6

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	65.0%	62.9%	64.2%	67.5%	70.4%
0.02	52.1%	66.7%	67.1%	66.3%	72.9%
0.03	52.5%	64.6%	63.3%	63.8%	69.2%

Table 5.21 Model C 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.7

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	50.0%	54.6%	54.6%	40.8%	59.6%
0.02	54.2%	55.8%	55.8%	56.7%	58.8%
0.03	50.4%	53.3%	55.8%	57.9%	56.7%

Table 5.22 Model D 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.1

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	78.7%	76.8%	77.8%	80.1%	80.3%
0.02	78.7%	77.1%	76.3%	80.5%	80.8%
0.03	77.2%	77.4%	78.1%	78.3%	79.9%

Table 5.23 Model D 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.2

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	79.8%	77.5%	77.3%	**82.0%**	81.1%
0.02	77.4%	78.5%	78.3%	81.3%	79.9%
0.03	76.8%	78.6%	78.1%	81.1%	79.2%

Table 5.24 Model D 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.3

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	78.6%	79.0%	77.4%	79.3%	80.3%
0.02	77.9%	79.0%	77.3%	79.1%	80.1%
0.03	79.8%	78.8%	78.9%	79.3%	79.6%

Table 5.25 Model D 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.4

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	78.3%	77.8%	79.2%	80.8%	79.2%
0.02	78.8%	78.3%	77.3%	79.1%	79.2%
0.03	77.0%	77.8%	75.6%	79.7%	79.0%

Table 5.26 Model D 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.5

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	78.3%	77.6%	78.7%	79.3%	80.6%
0.02	78.7%	79.5%	78.6%	80.3%	80.7%
0.03	79.9%	79.1%	77.5%	80.3%	80.3%

Table 5.27 Model D 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.6

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	77.7%	79.8%	80.3%	80.3%	79.3%
0.02	78.3%	76.8%	77.6%	79.8%	80.9%
0.03	78.9%	78.7%	78.7%	79.5%	79.6%

Table 5.28 Model D 5-Fold Cross Validation Average Validation Accuracies, Drop rate, *dp* = 0.7

Leak Rate, *leakR*	L2 Coefficient, λ				
	0.01	0.02	0.03	0.04	0.05
0.01	78.5%	78.9%	79.8%	78.0%	80.0%
0.02	77.6%	78.3%	79.0%	80.4%	78.1%
0.03	78.5%	78.3%	80.2%	80.3%	79.3%

Table 5.29 Testing Result for the Best Models

Models	Testing Accuracy	In- Class Accuracy (Number of Scans)		
		NC (correct/total)	MCI (correct/total)	AD (correct/total)
Model A	74.9%	69.3% (61/88)	78.6% (92/117)	75.9% (41/54)
Model B	71.4%	67.0% (59/88)	76.1% (89/117)	68.5% (37/54)
Model C	79.9%	84.1% (74/88)	79.5% (93/117)	74.1% (40/54)
Model D	81.5%	81.8% (72/88)	87.2% (102/117)	68.5% (37/54)

Table 5.30 Cross Validation Performance on LSUN Subsets

	Avg. Training Accuracy (%) Training Size 21,600	Avg. Training Time (seconds per epoch)	Avg. Testing Accuracy (%) Testing Size 3,000
LSUN - 1	99.78	294	94.31
LSUN - 2	99.75	266	94.07
LSUN - 3	99.78	241	94.42
LSUN - 4	99.74	273	94.60
LSUN - 5	99.73	296	93.94
Overall	99.76	274	94.27

Table 5.31 Average Testing Performance on LSUN Subsets and Generalization Tests

Test Set (size)	Overall Accuracy	In- Class Accuracy		
		Bedroom	Living Room	Dining Room
LSUN (3,000)	94.27%	95.63%	95.23%	91.87%
MIT Indoor67 (211)	87.68%	81.16%	91.67%	90.0%
15 – Scene (505)	89.70%	96.30%	N/A	84.78%

Table 5.32 Grid Search Result on LSUN

Number of filters (*nf*)	Drop Rate (*dp*)						
	0.1	0.2	0.3	0.4	0.5	0.6	0.7
100	0.952121	0.960438	0.964043	0.964908	0.964783	**0.966616**	**0.966411**
500	0.955373	0.961785	0.963572	0.964510	0.964779	0.966116	**0.966928**
1000	0.954033	0.960794	0.961595	0.963431	0.964352	0.965939	0.964981

Table 5.33 Grid Search Result on MIT Indoor67

Number of filters (*nf*)	Drop Rate (*dp*)						
	0.1	0.2	0.3	0.4	0.5	0.6	0.7
100	0.886256	0.905213	**0.914692**	**0.914692**	0.895735	0.909953	**0.924171**
500	0.867299	0.890995	0.905213	0.890995	**0.914692**	0.905213	0.900474
1000	0.895735	0.895735	0.890995	0.895735	0.895735	0.905213	0.895735

Table 5.34 Grid Search Result on 15-Scene

Number of filters (*nf*)	Drop Rate (*dp*)						
	0.1	0.2	0.3	0.4	0.5	0.6	0.7
100	0.877228	0.938614	0.944554	0.952475	**0.956436**	0.946535	**0.954455**
500	0.916832	0.930693	**0.954455**	**0.956436**	0.950495	**0.954455**	0.948515
1000	0.934653	0.948515	0.950495	0.948515	0.944554	0.952475	0.946535

Table 5.35 Testing Performance on All Datasets

Test Set (size)	Overall Accuracy	In- Class Accuracy		
		Bedroom	Living Room	Dining Room
LSUN (938,472)	97.16%	97.80%	96.87%	95.42%
MIT Indoor67 (211)	93.84%	91.30%	91.67%	98.57%
15 – Scene (505)	96.04%	98.61%	N/A	94.12%
Self-collected (15)	100%	100%	100%	100%

Table 5.36 Prediction Confidence on the Real Scenes

Image Index	Prediction (Probability)			Ground Truth Label
	Bedroom (Class 0)	Living Room (Class 1)	Dining Room (Class 2)	
1	**0.998473**	2.55452e-07	0.00152729	0
2	**0.999976**	1.85987e-07	2.37907e-05	0
3	**0.999414**	2.05452e-09	0.000586021	0
4	**0.995284**	1.92134e-06	0.00471381	0
5	**0.932796**	0.00128309	0.0659209	0
6	1.98696e-05	**0.999937**	4.35367e-05	1
7	0.000944164	**0.994802**	0.00425355	1
8	0.015924	**0.981498**	0.0025777	1
9	5.98542e-05	**0.998747**	0.00119291	1
10	0.000258657	**0.999459**	0.000282051	1
11	0.00369191	0.00025444	**0.996054**	2
12	0.00247314	0.000732423	**0.996794**	2
13	0.000932294	0.00150781	**0.99756**	2
14	0.00087074	0.000553563	**0.998576**	2
15	0.000863768	0.00238603	**0.99675**	2

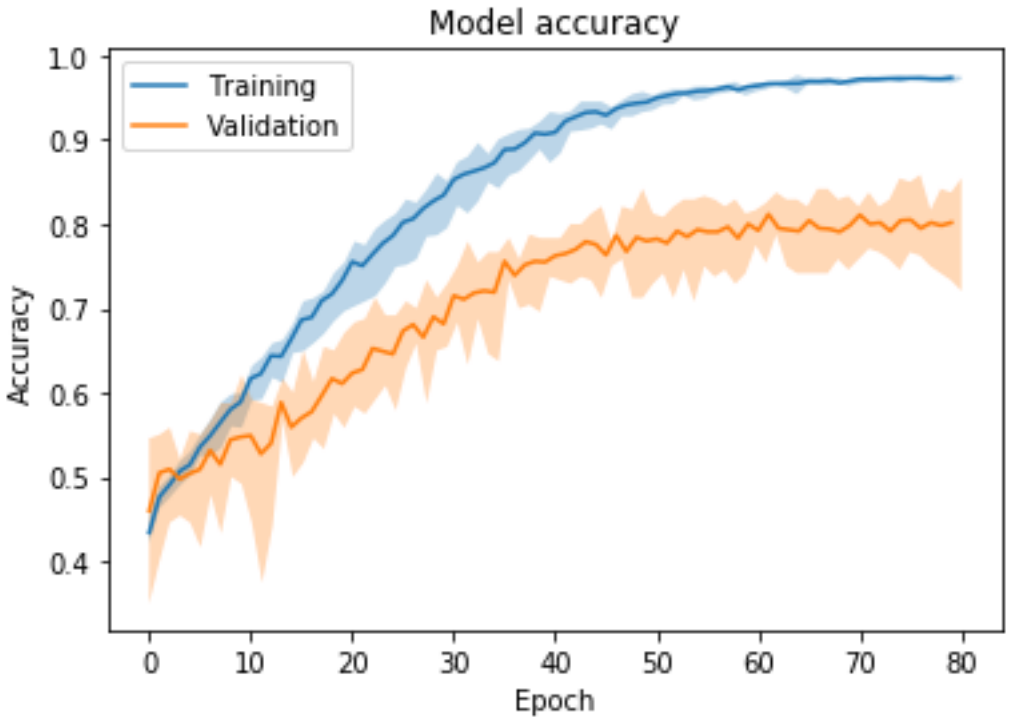


Figure 5.1 Best Model A- Average training and validation accuracies with error band

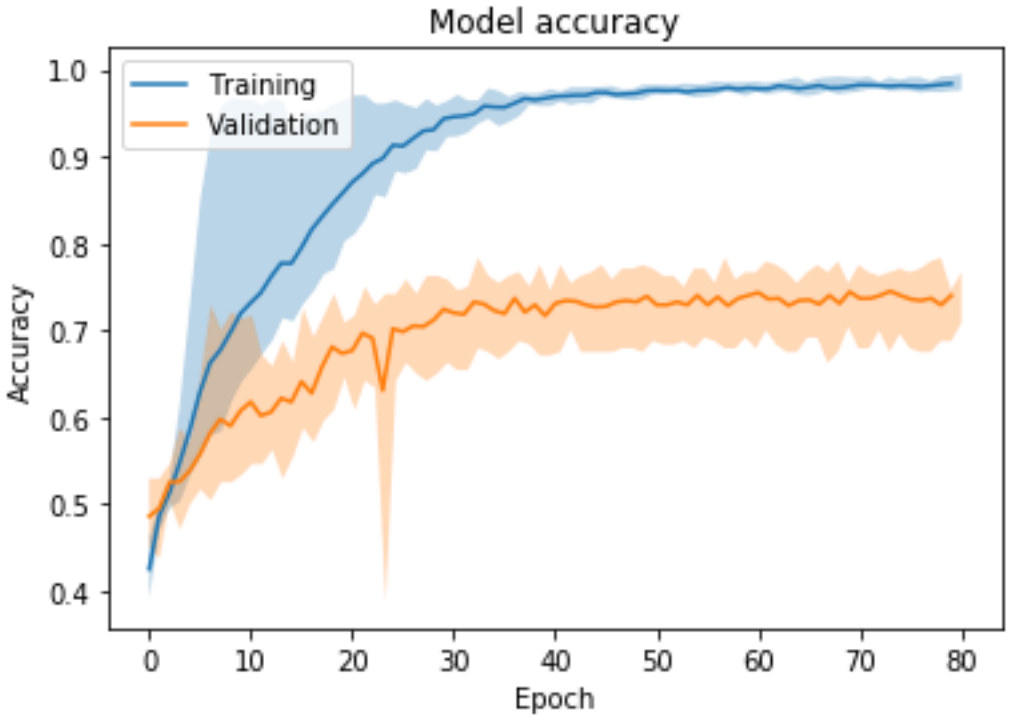


Figure 5.2 Best Model B – Average training and validation accuracies with error band

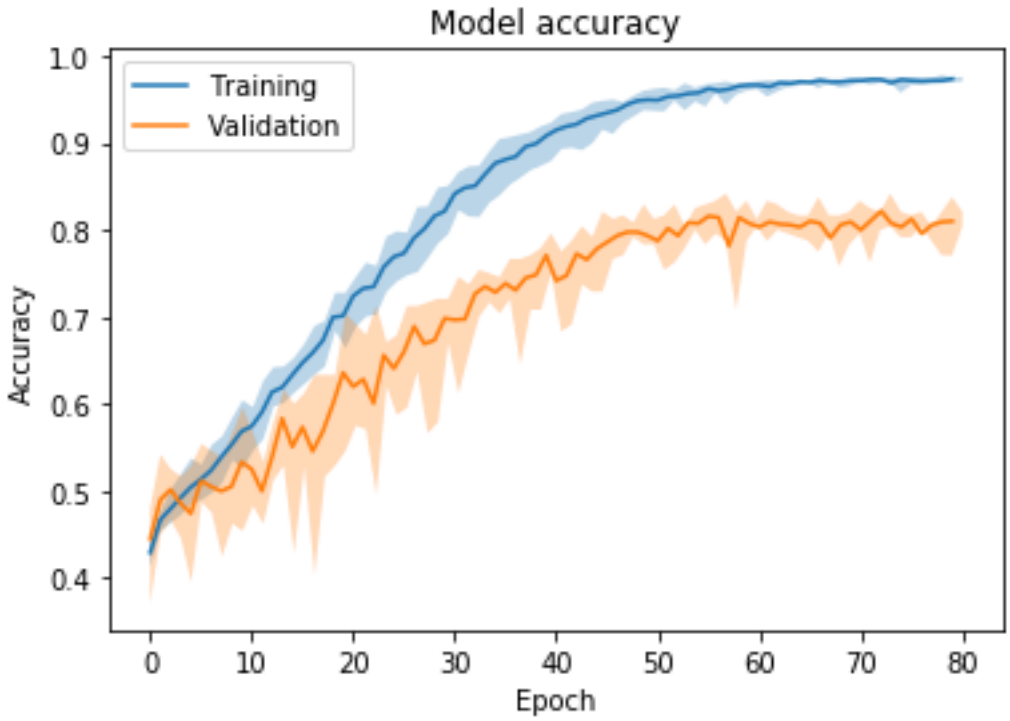


Figure 5.3 Best Model C- Average training and validation accuracies with error band

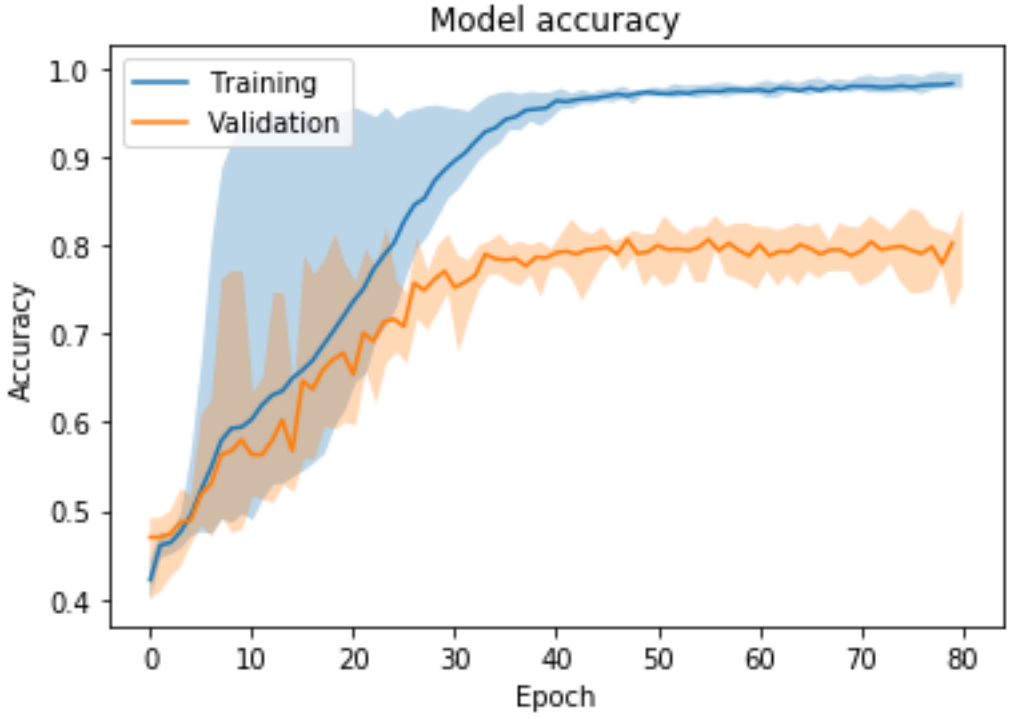


Figure 5.4 Best Model D- Average training and validation accuracies with error band

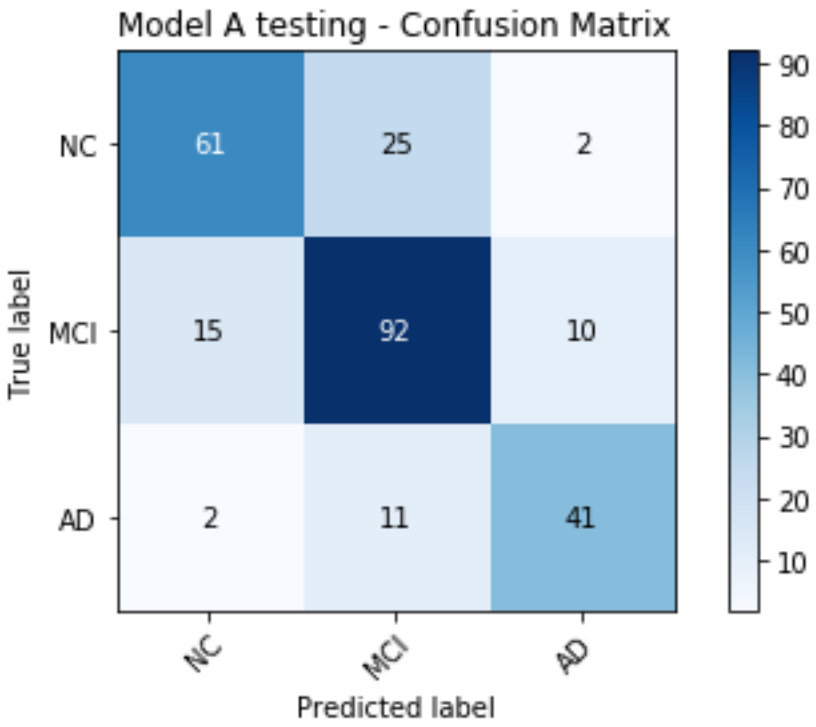


Figure 5.5 Confusion matrix for testing, Model A

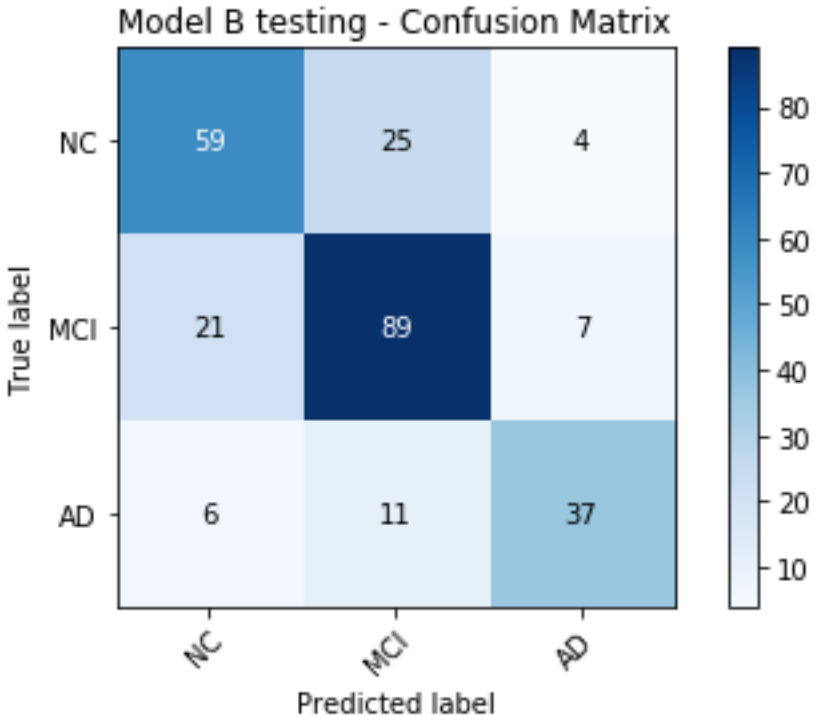


Figure 5.6 Confusion matrix for testing, Model B

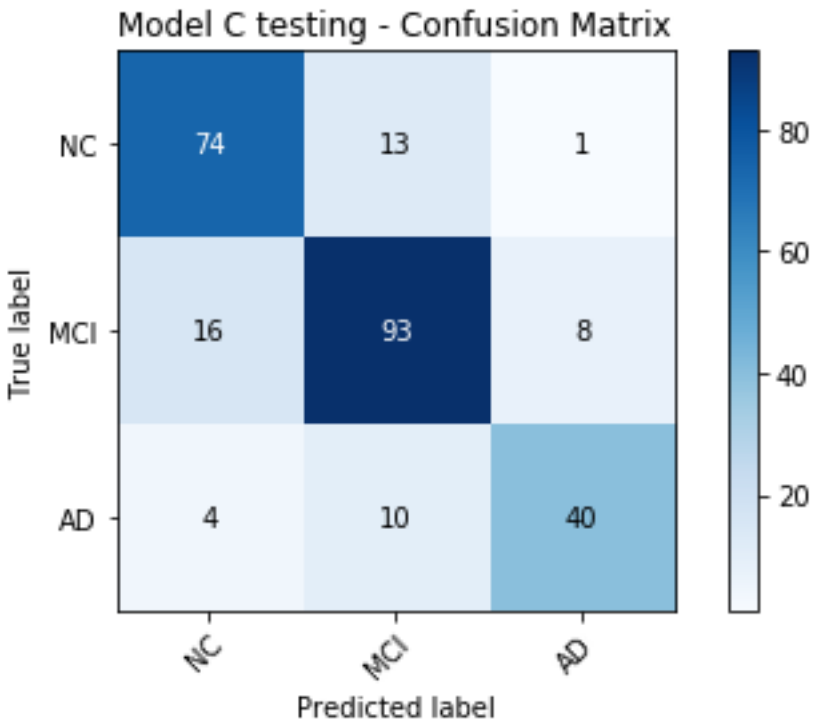


Figure 5.7 Confusion matrix for testing, Model C

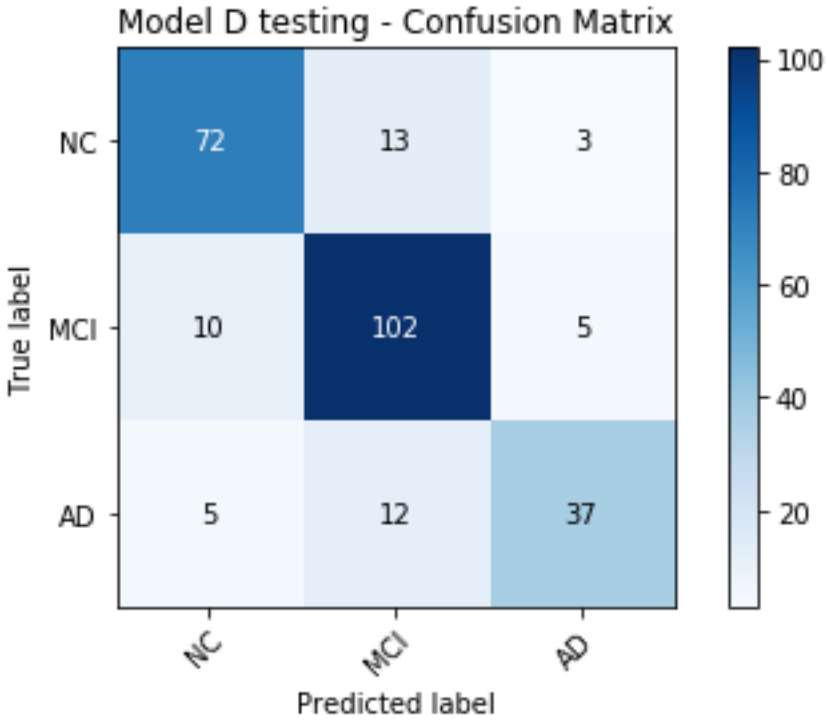


Figure 5.8 Confusion matrix for testing, Model D

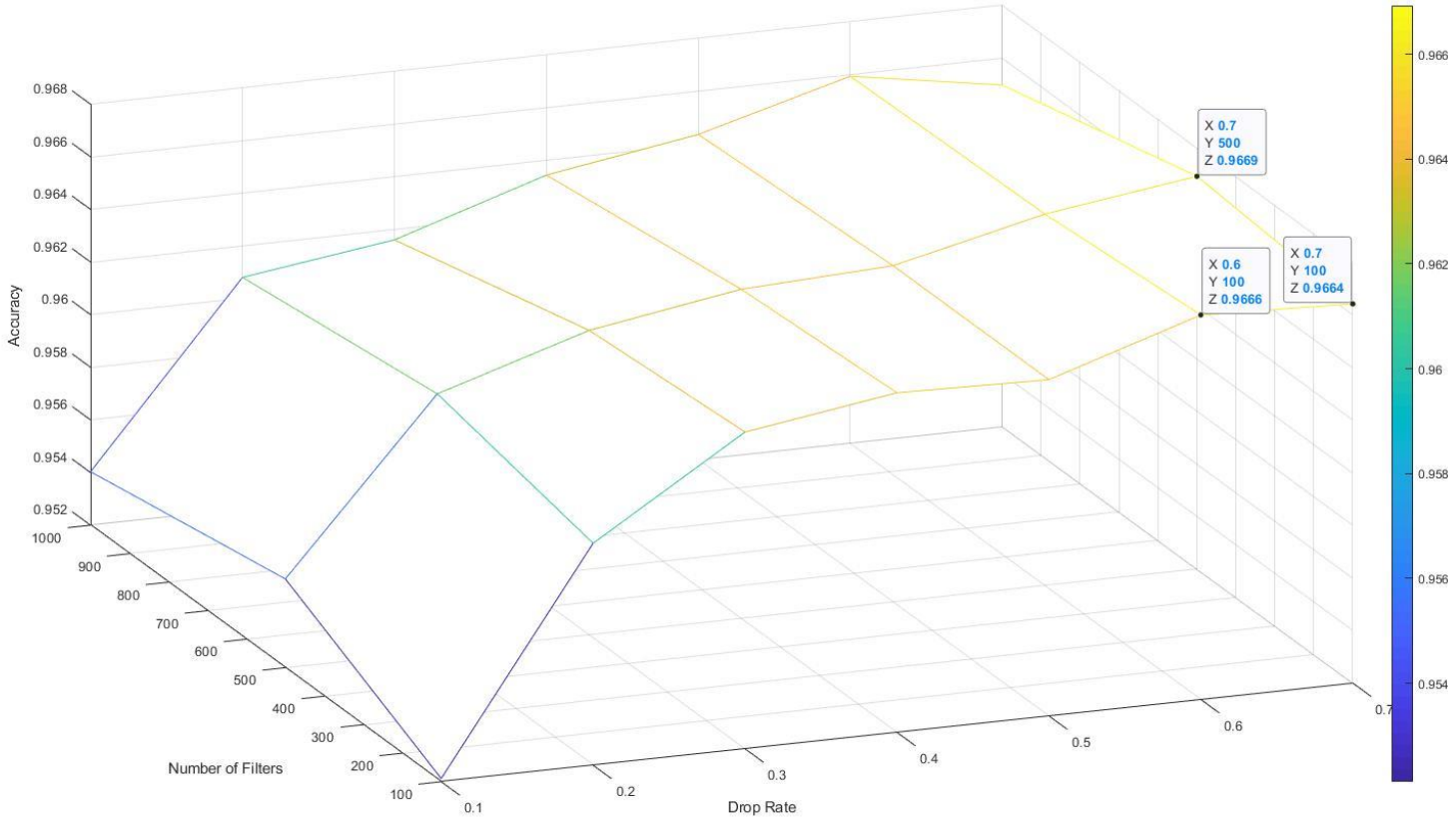


Figure 5.9 Grid search result with LSUN dataset. (Top-3 accuracies were labeled)

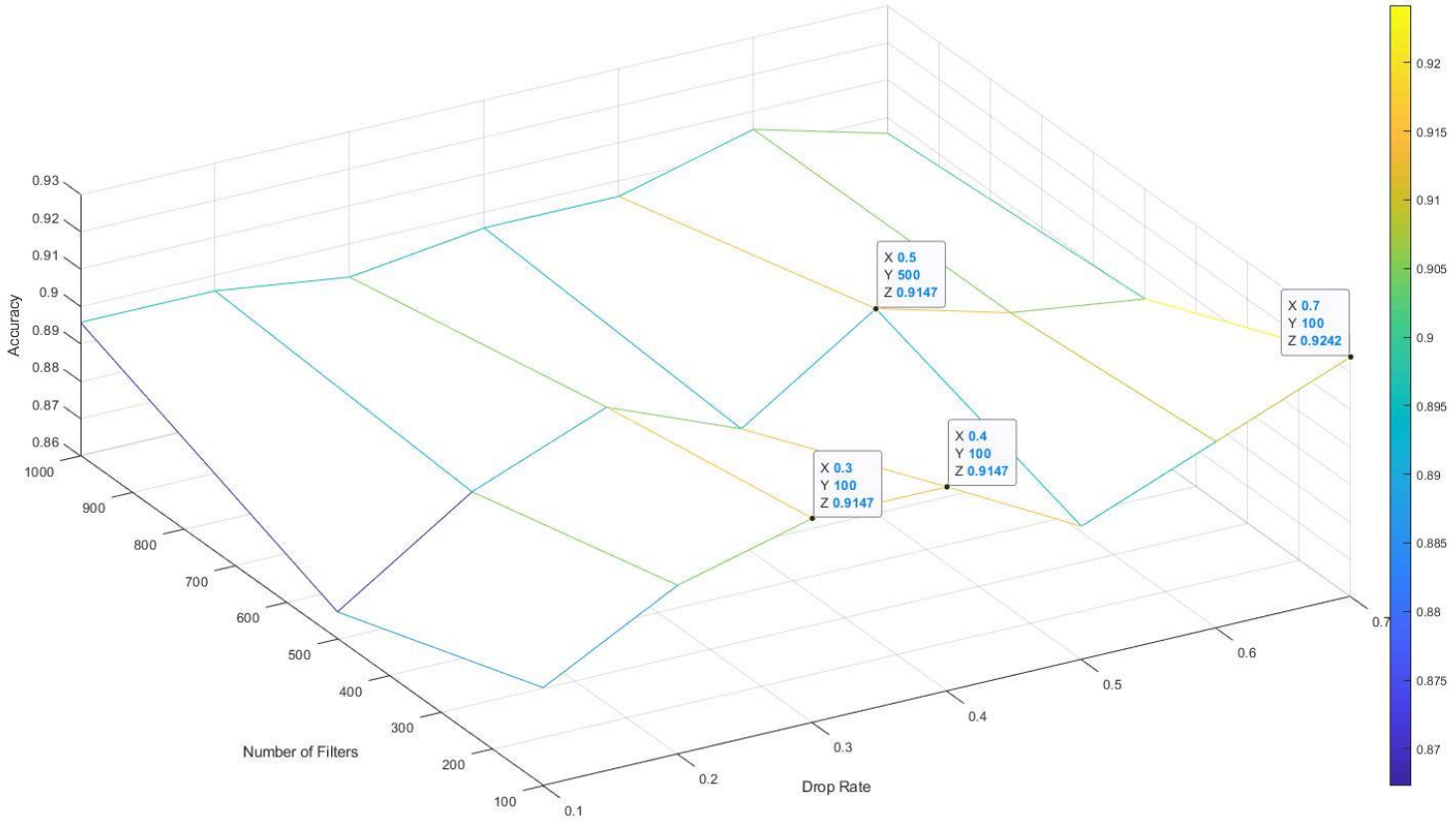


Figure 5.10 Grid search result with MIT Indoor67 dataset. (Top-3 accuracies were labeled)

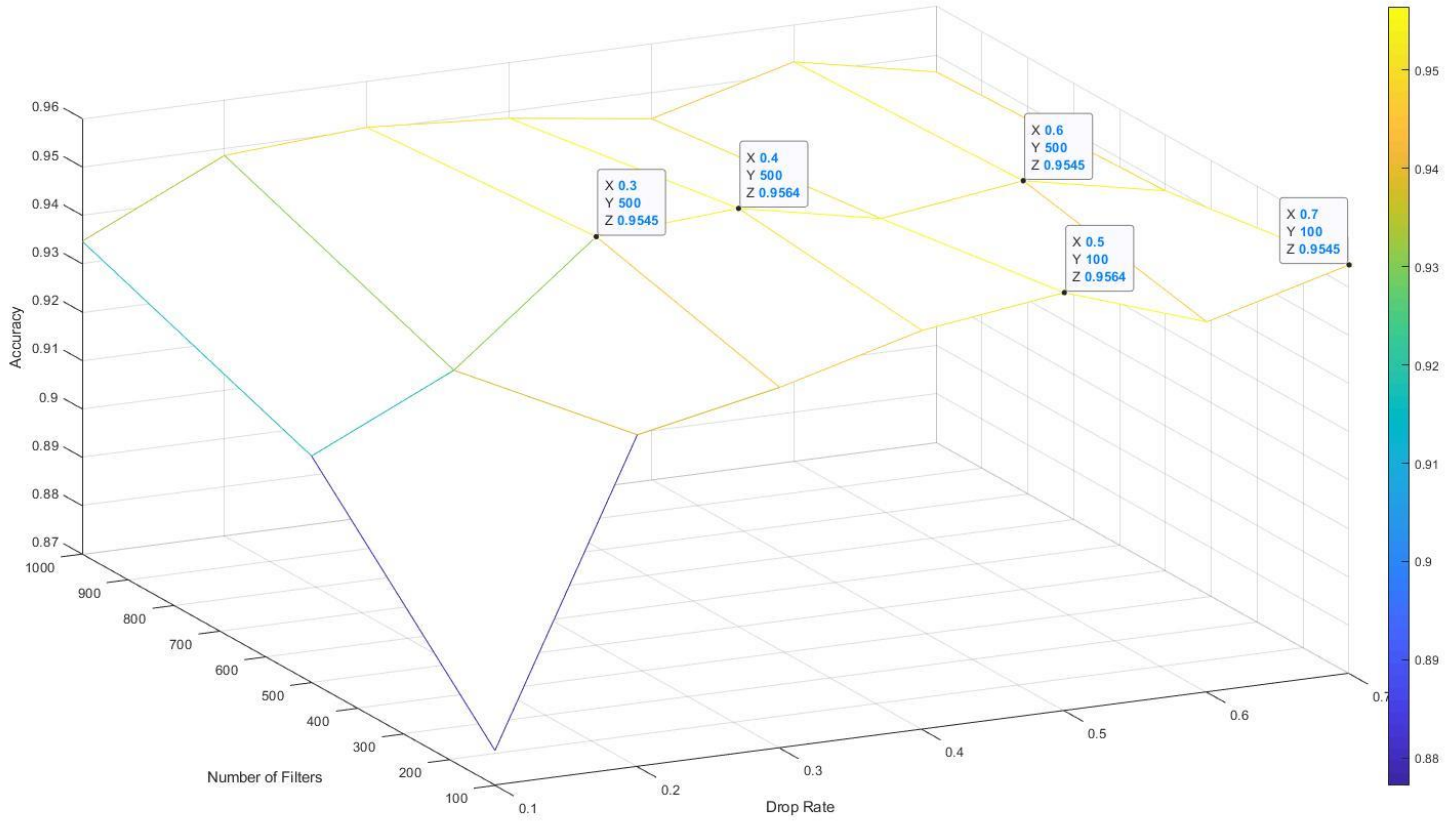


Figure 5.11 Grid search result with 15-Scene dataset. (Top-3 accuracies were labeled)

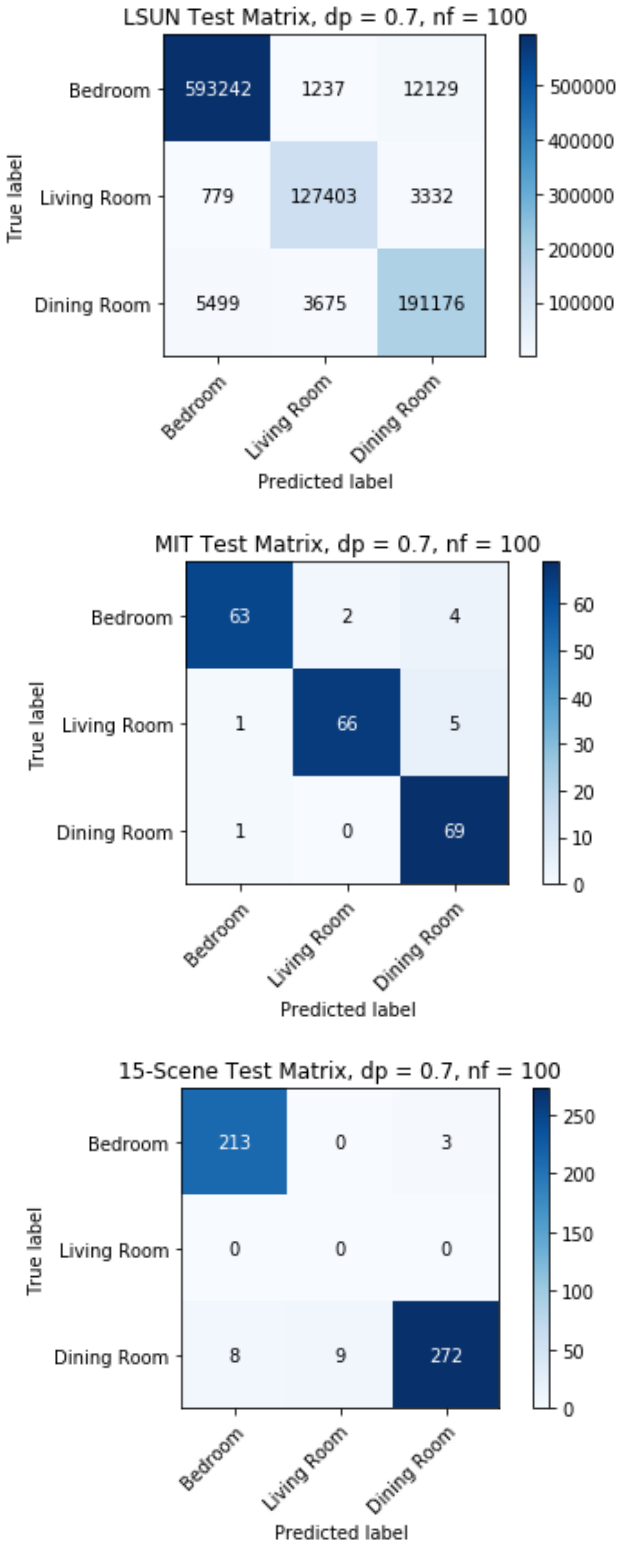


Figure 5.12 Confusion Matrix- Testing on the final model (LSUN, MIT Indoor67, 15-Scene)

Figure 5.13 Wrongly labeled images (labeled as living room)

Figure 5.14 Room with multiple functions

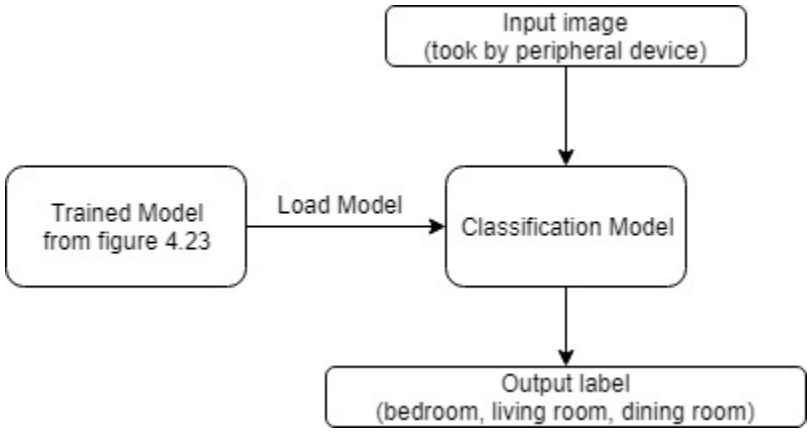


Figure 5.15 Flowchart of how to deploy classification model on portable platform

CHAPTER 6. CONCLUSION AND FUTURE WORK

In this work, two objectives related to Alzheimer's disease were proposed. The first objective target on the evaluation of the classification performance in automated Alzheimer's disease diagnosis using deep learning. The second objective involved performing quick indoor scene recognition using deep learning, which will serve as a segment of assistive system for Alzheimer's patients.

6.1 Automated Alzheimer's Disease Stages Classification

Four deep convolutional neural network models (model A, B, C and D) were developed in objective 1. This was to evaluate the multiclass classification accuracy and investigate why deep learning models struggled when perform such classification. In detail, Model A achieved an overall accuracy of 74.9% for classifying a give MRI scan into one of the three possible classes (NC: normal control, MCI: mild cognitive impairment, AD: Alzheimer's disease). The individual-class classification accuracies provided by this model were 69.3% for NC classification, 78.6% for MCI classification, and 75.9% for AD classification, respectively. Similarly, model B achieved an overall accuracy of 71.4% with individual-class classification accuracies of 67.0%, 76.1% and 68.5% for NC, MCI and AD, respectively. Model C (model A with MLO architecture) achieved an overall accuracy of 79.9% and individual-class classification accuracies of 84.1%, 79.5%, and 74.1% for NC, MCI and AD respectively. On the other hand, model D (model B with MLO architecture) achieved the highest overall accuracy of 81.5%. The individual-class accuracies for both NC and MCI were greater than 80% (81.8% and 87.2%, respectively). However, for classification of AD, the accuracy stayed at 68.5%.

Our analysis indicates that, because of the limited data, deep learning models might not have been able to effectively extract generalized high-level features and this might have caused overfitting. In this work, the proposed Multi-Layer-Output (MLO) mechanism (in model C and D) combined low-level features with high-level features. This combination helped to some extent in enhancing the performance of the model. While this work (models) is not ready for practical application. We believe that this platform/pathway has potential for further investigation and development.

The following suggestions are recommended for future research.

1. The developed models need to be further validated on larger and different datasets. Additional hyper-parameters could have been tuned for this model (such as learning rate, decay rate and different optimizers)

2. One may look into using data augmentation (Tanner & Wong, 1987) to answer the limited dataset problem. Standard data augmentation could also be adopted like rotation. Mirror flipping was on the other hand not recommended as hemispheres handled different functionalities. Some advanced technique such as Generative-Adversarial-Network (GAN; Goodfellow et al., 2014) have been adopted for other medical imaging like Computer Tomography (Mirsky et al., 2019).

3. Another thrust for future investigation could involve in refining the region of interest by adopting expert neurological suggestion.consultation. Hand-crafting the entire feature could be time-consuming and unreliable because it highly depends on the understanding of the subject (Domingos, 2012).

4. Lastly, one can combine the hand-crafted feature and deep-learning-extracted feature to assese their capability in enhancing the performance of the model.

The limitation of this work included the following. Firstly, the research was restrained by limited and unbalanced dataset. Secondly, the access to computation resources was limited.

6.2 Indoor Scene Understanding

In this work, we proposed a deep learning model inspired and transferred from ResNet-50 for indoor scene recognition. Such model was targeted on further helping cognitive impaired elders and enhance their quality of individual living by suggesting potential actions and as well automatically alert the caregivers. Three datasets were collected to fine-tune and test the proposed model. Different training schemes were implemented and their performance were assessed. The model showed 97.2% overall accuracy for three-class classification which is a state-of-the-art performance. Specifically, the individual-class classification accuracies were 97.8%, 96.9%, and 95.4% for bedroom, living room and dining room respectively. On the generalization datasets, the model achieved overall accuracy of 93.8% on MIT-indoor67 and 96.0% on 15-scene. For the test on the self-collected real life scenes, the model achieved overall accuracy of 100%.

This model shows very high potential for its use in real world setting. However additional research is needed for further validation of the model in using larger and different real wolf dataset. One could investigate how to improve the performance through online training. The concept is that the training data are gradually provided and the model was fine-tuned based on that. The idea was to create a case-specific model for the particular environment. Also, additional investigation is needed to implement the hardware system based on the developed model.

Despite the promising results obtained, there are some limitations to this study. In our model, all the hyper-parameters were frozen except the drop rate and the number of filters in the classifier. To further explore the model's ability on the subject matter, a more thorough hyper-

parameters search should be conducted, including but not limit to: number of layers, number of filters in each convolutional layer, different optimizers, and different learning rates. Furthermore, the model was only tested with three room categories on the collected datasets.

APPENDIX A SOURCE CODE FOR AUOTMATED AD DIAGNOSIS

```
########################################################################
#Major of the functions used were imported from Keras library (Chollet,F, et al. 2015). Please see https://keras.io/ for #more documentation.
########################################################################
import numpy as np
import os
from keras.models import Sequential
from keras.layers import Dense, Activation
from DataGenerator import DataGenerator
import csv
from keras.layers.convolutional import (
    Conv3D,
    AveragePooling3D,
    MaxPooling3D,
    ZeroPadding3D
)
from keras.layers.normalization import BatchNormalization
from keras.layers import Flatten, Activation, Input,Dropout, LeakyReLU, concatenate
from keras.layers.merge import add
from keras.models import Model, load_model
from keras.regularizers import l2
from keras.initializers import he_normal, Zeros
from keras.layers import Dense, GlobalAveragePooling3D
import nibabel as nib
from keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
import argparse
import datetime
import tensorflow as tf
```

```
from keras.utils import plot_model
from scipy.ndimage import rotate
import pickle

############################## Model A  figure 4.10 ###################################
def modela(dim1,dim2,dim3,dp=0.2,n_channels=1,num_classes=3,leakR=0.02,
reg_lambda=0.03):

#input arguments:
#      dim1,dim2,dim3:the size of 3D input tensor
#      dp: drop rate, default rate 0.2
#      n_channels: number of channels in the input,1-grayscal,3-color
#      num_classes: number of output classes, default numbe is 3
#      leakR: leak rate of the leaky relu unit. default rate is 0.02
#      reg_lambda: lambda coefficient of the L2 regularizer. default value is 0.03

    #define global parameter
    global reg, kinit, binit

    #define convolutional layer regularizer - L2 regularization with variable lambda
    reg=l2(reg_lambda)

    #define kernel weights initializer - glorot normal distribution
    kinit = glorot_normal()

    #define kernel bias initializer - all zeros
    binit = Zeros()

    #define batch normalization axis - 4 means on the 4th axis which is the axis representing color
    #channel
    axis_batchnorm =4
```

```
    #create input layer, size matches to the given input argument and the color channel
    tensor_input = Input(shape=(dim1,dim2,dim3,n_channels))

    #parse input tensor to the next layer, x is an iterable object to store intermediate object
    x = tensor_input

    #create Conv_1 layer (32 filters, 7 × 7 × 7 kernel size, strides = (2,2,2))
    #create its batch normalization layer and activation layer
    x = Conv3D(32,7,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv1_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_1')(x)
    x = LeakyReLU(leakR)(x)

    #create average pooling layer (kernel size 5 × 5 × 5, strides = (2,2,2)
    x = AveragePooling3D((5,5,5),strides=(2,2,2),padding='same')(x)

    #create conv_2 layer(64 filters, 3 × 3 × 3 kernel size, strides = (2,2,2))
    #create batch normalization layer and activation layer
    x = Conv3D(64,3,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv2_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_2')(x)
    x = LeakyReLU(leakR)(x)

    #create conv_3 layer(128 filters, 3 × 3 × 3 kernel size, strides = (2,2,2))
    #create batch normalization layer and activation layer
    x = Conv3D(128,3,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv3_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_3')(x)
    x = LeakyReLU(leakR)(x)
```

```
    #create conv_4 layer(256 filters, 3 × 3 × 3 kernel size, strides = (2,2,2))
    #create batch normalization layer and activation layer
    x = Conv3D(256,3,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv4_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name=‘batchnorm_4')(x)
    x = LeakyReLU(leakR)(x)

    #create global average pooling layer
    x = GlobalAveragePooling3D()(x)

    #create dropout layer with drop rate dp.
    x = Dropout(dp)(x)

    #create fully-connected layers with 128 neurons
    x = Dense(128,activation='relu')(x)

    #create fully-connected layers with 64 neurons
    x = Dense(64,activation='relu')(x)

    #create prediction layers with 3 neurons
    prediction = Dense(3,activation='softmax')(x)

    #envelop the model by defining overall input and output
    model = Model(inputs=tensor_input,outputs=prediction)

    return model

################################ Model B  figure 4.11################################

#define Residual Blk for model B, see figure 4.12
```

```
def conv_blk(tensor_in, kernel_size, filters, stg, blk,leakRate, strides=(2, 2, 2)):
#input arguments:
#       tensor_in: the input tensor object
#       kernel_size: the size of kernel in this blk. by default it is a 3D kernel with the same size #on all axes
#       filter: number of filters in the two conv layers in the blk
#       stg, blk: naming scheme. used as identifier when checking model
#       leakRate: leak rate of the leaky relu unit.
#       strides: strides for the conv layers. default value is (2,2,2)

    #define batch normalization axis - 4 means on the 4th axis which is the axis representing the
    #color channel
    axis_batchnorm =4

    #retrive filer number from input argument
    if len(filters)==3:
        nb_filter1, nb_filter2, nb_filter3 = filters
    elif len(filters)==2:
        nb_filter1, nb_filter2 = filters

    #define layer's naming pattern
    conv_root = 'res' + str(stg) + blk + '_branch'
    batchnorm_root = 'bn' + str(stg) + blk + '_branch'

    #create the first 3D convolutional layer on the main branch, as well as its batch normalization layer and activation layer
    x                                                                                                   =
Conv3D(nb_filter1,3,strides=2,padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name=conv_root + '2a')(tensor_in)
    x = BatchNormalization(axis=axis_batchnorm, name=batchnorm_root + '2a')(x)
```

```
    x = LeakyReLU(leakRate)(x)

    #create the second 3D convolutional layer on the main branch, as well as its batch normalization
layer
    x                                                                                    =
Conv3D(nb_filter2,3,strides=1,padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name=conv_root + '2b')(x)
    x = BatchNormalization(axis=axis_batchnorm, name=batchnorm_root + '2b')(x)

    #create the 3D convolutional layer on the shortcut, as well as its batch normalization layer
    shortcut = Conv3D(nb_filter2, 1, strides=2,kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit, name=conv_root + '1')(tensor_in)
    shortcut = BatchNormalization(axis=axis_batchnorm, name=batchnorm_root + '1')(shortcut)

    #merge the shortcut back to the main branch
    x = add([x, shortcut])

    #create activation
    x = LeakyReLU(leakRate)(x)

    return x

#define model B, see figure 4.11

def modelb(img_dim1,img_dim2,img_dim3, dp =0.2, color_type=1,
num_classes=3,leakR=0.02,reg_lambda=0.03):

#input arguments:
#       dim1,dim2,dim3:the size of 3D input tensor
#       dp: drop rate, default rate 0.2
#       n_channels: number of channels in the input,1-grayscal,3-color
```

```
#      num_classes: number of output classes, default numbe is 3
#      leakR: leak rate of the leaky relu unit. default rate is 0.02
#      reg_lambda: lambda coefficient of the L2 regularizer. default value is 0.03

    #define global parameter
    global reg, kinit, binit

    #define convolutional layer regularizer - L2 regularization with variable lambda
    reg=l2(reg_lambda)

    #define kernel weights initializer - glorot normal distribution
    kinit = glorot_normal()

    #define kernel bias initializer - all zeros
    binit = Zeros()

    #define batch normalization axis - 4 means on the 4th axis which is the axis representing the
    #color channel
    axis_batchnorm =4

    #create input layer
    tensor_input = Input(shape=(img_dim1,img_dim2,img_dim3,color_type))

    #parse input tensor to the next layer, x is an iterable object to store intermediate object
    x = tensor_input

    #create Conv_1 layer, its batch normalization layer and activation layer
    x = Conv3D(32,3,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv1_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_1')(x)
    x = LeakyReLU(leakR)(x)
```

```
#create max pooling layer
x = MaxPooling3D((3,3,3),strides=(2,2,2),padding='same')(x)

#create ResBlk_1, 3 × 3 × 3 kernel, [32,32]filters
x = conv_blk(x, 3, [32,32],stg=2,blk='a',leakRate=leakR)

#create ResBlk_2, 3 × 3 × 3 kernel, [64,64]filters
x = conv_blk(x, 3, [64,64],stg=3,blk='a',leakRate=leakR)

#create ResBlk_3, 3 × 3 × 3 kernel, [128,128]filters
x = conv_blk(x, 3, [128,128],stg=4,blk='a',leakRate=leakR)

#create global average pooling layer
x = GlobalAveragePooling3D()(x)

#create dropout layer with drop rate dp.
x = Dropout(dp)(x)

#create fully-connected layers with 128 neurons
x = Dense(128,activation='relu')(x)

#create fully-connected layers with 64 neurons
x = Dense(64,activation='relu')(x)

#create prediction layers with 3 neurons
prediction = Dense(3,activation='softmax')(x)

#envelop the model
new_model = Model(inputs=tensor_input, outputs=prediction)
```

```
    return new_model

##########################Model c see figure 4.14 ###################################

def modelc(dim1,dim2,dim3,dp=0.2,n_channels=1,num_classes=3,leakR=0.02,
reg_lambda=0.03):
#input arguments:
#       dim1,dim2,dim3:the size of 3D input tensor
#       dp: drop rate, default rate 0.2
#       n_channels: number of channels in the input,1-grayscal,3-color
#       num_classes: number of output classes, default numbe is 3
#       leakR: leak rate of the leaky relu unit. default rate is 0.02
#       reg_lambda: lambda coefficient of the L2 regularizer. default value is 0.03

    #define global parameter
    global reg, kinit, binit

    #define convolutional layer regularizer - L2 regularization with variable lambda
    reg=l2(reg_lambda)

    #define kernel weights initializer - glorot normal distribution
    kinit = glorot_normal()

    #define kernel bias initializer - all zeros
    binit = Zeros()

    #define batch normalization axis - 4 means on the 4th axis which is the axis representing the
    #color channel
    axis_batchnorm =4
```

```
    #create input layer
    tensor_input = Input(shape=(dim1,dim2,dim3,n_channels))

    #parse input tensor to the next layer, x is an iterable object to store intermediate object
    x = tensor_input

    #create Conv_1 layer, its batch normalization layer and activation layer
    x = Conv3D(32,7,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv1_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_1')(x)
    x = LeakyReLU(leakR)(x)

    #create average pooling layer
    x = AveragePooling3D((5,5,5),strides=(2,2,2),padding='same')(x)

    #create conv_2 layer, its batch normalization layer and activation layer
    x = Conv3D(64,3,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv2_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_2')(x)
    x = LeakyReLU(leakR)(x)

    #output the feature from conv_2 to tensor object x1
    x1 = GlobalAveragePooling3D()(x)

    #create conv_3 layer, its batch normalization layer and activation layer
    x = Conv3D(128,3,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv3_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_3')(x)
    x = LeakyReLU(leakR)(x)

    #output the feature from conv_3 to tensor object x2
```

```
    x2 = GlobalAveragePooling3D()(x)

    #create conv_4 layer, its batch normalization layer and activation layer
    x = Conv3D(256,3,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv4_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_4')(x)
    x = LeakyReLU(leakR)(x)

    #output the feature from conv_4 to tesnsor object x
    x = GlobalAveragePooling3D()(x)

    #concatenate features from different level to tensor object sx - sum of x
    sx = concatenate([x,x1,x2],axis=-1)

    #reate dropout layer with drop rate dp
    sx = Dropout(dp)(sx)

    #create fully-connected layers with 128 neurons
    sx = Dense(128,activation='relu')(sx)

    #create fully-connected layers with 64 neurons
    sx = Dense(64,activation='relu')(sx)

    #reate prediction layers with 3 neurons
    prediction = Dense(3,activation='softmax')(sx)

    #envelop the model
    model = Model(inputs=tensor_input,outputs=prediction)

    return model
##################### Model D see figure 4.15 ######################################
```

```
def conv_blk_3D(tensor_in, kernel_size, filters, stg, blk,leakRate, strides=(2, 2, 2)):
#input arguments:
#       tensor_in: the input tensor object
#       kernel_size: the size of kernel in this blk. by default it is a 3D kernel with the same size #on all axes
#       filter: number of filters in the two conv layers in the blk
#       stg, blk: naming scheme. used as identifier when checking model
#       leakRate: leak rate of the leaky relu unit.
#       strides: strides for the conv layers. default value is (2,2,2)

    #define batch normalization axis - 4 means on the 4th axis which is the axis representing the
    # color channel
    axis_batchnorm =4

    #retrive filer number from input argument
    if len(filters)==3:
        nb_filter1, nb_filter2, nb_filter3 = filters
    elif len(filters)==2:
        nb_filter1, nb_filter2 = filters

    #define layer's naming pattern
    conv_root = 'res' + str(stg) + blk + '_branch'
    batchnorm_root = 'bn' + str(stg) + blk + '_branch'

    #create the first 3D convolutional layer on the main branch, as well as its batch normalization layer and activation layer
    x =
Conv3D(nb_filter1,3,strides=2,padding='same',kernel_regularizer=reg,kernel_initializer=kinit, bias_initializer=binit,name=conv_root + '2a')(tensor_in)
    x = BatchNormalization(axis=axis_batchnorm, name=batchnorm_root + '2a')(x)
```

```
    x = LeakyReLU(leakRate)(x)

    #create the second 3D convolutional layer on the main branch, as well as its batch normalization
layer
    x =
Conv3D(nb_filter2,3,strides=1,padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name=conv_root + '2b')(x)
    x = BatchNormalization(axis=axis_batchnorm, name=batchnorm_root + '2b')(x)

    #create the 3D convolutional layer on the shortcut, as well as its batch normalization layer
    shortcut = Conv3D(nb_filter2, 1, strides=2,kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit, name=conv_root + '1')(tensor_in)
    shortcut = BatchNormalization(axis=axis_batchnorm, name=batchnorm_root + '1')(shortcut)

    #merge the shortcut back to the main branch
    x = add([x, shortcut])

    #create activation
    x = LeakyReLU(leakRate)(x)

    return x

def modeld(img_dim1,img_dim2,img_dim3, dp =0.2, color_type=1,
num_classes=3,leakR=0.02,reg_lambda=0.03):
#input arguments:
#       dim1,dim2,dim3:the size of 3D input tensor
#       dp: drop rate, default rate 0.2
#       color_type: number of channels in the input,1-grayscal,3-color
#       num_classes: number of output classes, default numbe is 3
#       leakR: leak rate of the leaky relu unit. default rate is 0.02
#       reg_lambda: lambda coefficient of the L2 regularizer. default value is 0.03
```

```
    #define global parameter
    global reg, kinit, binit

    #define convolutional layer regularizer - L2 regularization with variable lambda
    reg=l2(reg_lambda)

    #define kernel weights initializer - glorot normal distribution
    kinit = glorot_normal()

    #define kernel bias initializer - all zeros
    binit = Zeros()

    #define batch normalization axis - 4 means on the 4th axis which is the axis representing the
    #color channel
    axis_batchnorm =4

    #create input layer
    tensor_input = Input(shape=(img_dim1,img_dim2,img_dim3,color_type))

    #parse input tensor to the next layer, x is an iterable object to store intermediate object
    x = tensor_input

    #create Conv_1 layer, its batch normalization layer and activation layer
    x = Conv3D(32,3,strides=2, padding='same',kernel_regularizer=reg,kernel_initializer=kinit,
bias_initializer=binit,name='conv1_1')(x)
    x = BatchNormalization(axis=axis_batchnorm,name='batchnorm_1')(x)
    x = LeakyReLU(leakR)(x)

    #create max pooling layer
    x = MaxPooling3D((3,3,3),strides=(2,2,2),padding='same')(x)
```

```
#create ResBlk_1, 3X3X3 kernel, [32,32]filters
x = conv_blk_3D(x, 3, [32,32],stg=2,blk='a',leakRate=leakR)

#output the feature from Resblk_1 to tensor object x1
x1 = GlobalAveragePooling3D()(x)

#create ResBlk_2, 3X3X3 kernel, [64,64]filters
x = conv_blk_3D(x, 3, [64,64],stg=3,blk='a',leakRate=leakR)

#output the feature from Resblk_2 to tensor object x2
x2 = GlobalAveragePooling3D()(x)

#create ResBlk_3, 3X3X3 kernel, [128,128]filters
x = conv_blk_3D(x, 3, [128,128],stg=4,blk='a',leakRate=leakR)

#output the feature from ResBlk_3 to tesnsor object x
x = GlobalAveragePooling3D()(x)

#concatenate features from different level to tensor object sx - sum of x
x = concatenate([x,x1,x2],axis=-1)

#reate dropout layer with drop rate dp
x = Dropout(dp)(x)

#create fully-connected layers with 128 neurons
x = Dense(128,activation='relu')(x)

#create fully-connected layers with 64 neurons
x = Dense(64,activation='relu')(x)
```

```
    #reate prediction layers with 3 neurons
    prediction = Dense(3,activation='softmax')(x)

    #envelop the model
    new_model = Model(inputs=tensor_input, outputs=prediction)

    return new_model

################################################################################
#
#               End of model defination
#
################################################################################

################### Training process, see figure 4.16 ##########################

################### Initialize argument parser #################################

#create parser handle
parser = argparse.ArgumentParser(description="Hyper parameter for training")

#create parser argument: dp-drop rate, model-which model (a/b/c/d)
parser.add_argument('dp',type=float,help="drop rate")
parser.add_argument('model', type=str,help="model to be trained")
args = parser.parse_args()

#parse value from keyboard input to variable
dp_range = args.dp
model_type = args.model

##################### Step A&B #################################################
```

```
"""
prepare cross validation file list.
"""

#initilize ground truth label and ground truth id
gt_label = []
gt_id = []

#Read MRI scan index and their ground truth label
with    open('//scratch//gilbreth//xu640/ADNI//ADNI1_Annual_2_Yr_1.5T_4_29_2019.csv')    as
gt_csv:
    gt_reader = csv.reader(gt_csv)
    next(gt_reader)

    for line in gt_reader:
        ind = line[1]
        stg = line[2]

        if stg == 'CN':
            stg = 0
        elif stg == 'MCI':
            stg = 1
        else:
            stg = 2



        if ind in gt_id: continue
        gt_id.append(ind)
        gt_label.append(stg)
```

```
#store the data into a dictionay for datagenerator
labels = dict(zip(gt_id,gt_label))

######## Define Global Training Parameter ########

#data directorys:
#trained_model: directory to store models with trained weights
#test_output: directory to store testing outputs – validation accuracies, ground truth label,
#                  prediction labels.
#training_history: directory to store training history – training accuracies for each epoch
#file_path : directory to the data file

trained_model = '//scratch//gilbreth//xu640//ADNI_New//'+model_type+'//trained_model//'
test_output = '//scratch//gilbreth//xu640//ADNI_New//'+model_type+'//test_output//'
training_history = '//scratch//gilbreth//xu640//ADNI_New//'+model_type+'//training_history//'
file_path = '//scratch//burst//xu640//ADNI_New//cv//'

#Leak Rate search range
leakR_range = [0.01,0.02,0.03]

#L2 coefficient search range
lambda_range = [0.01,0.02,0.03,0.04,0.05]

#Cross Validation fold index
folds = [0,1,2,3,4]

#initial learning rate
init_lr=0.0001

#minimal learning rate
min_rate =0.00001
```

```
#training epochs
ep     =80

#setup Adam optimizer
op = Adam(lr=init_lr,beta_1=0.9, beta_2=0.999)

#parameter for the data generator
"""
dimension:tuple of integers, input tensor size
size:integer, number of training sample within one batch
n_categories: integer, number of classes in the data
color_mode: integer, number of channel: 1- grayscale; 3-color
is_shuffle:boolean, shullfe the data in the batch or not
"""
params = {'dimension': (96,96,64),
          size': 20,
          'n_categories': 3,
          'color_mode': 1,
          'is_shuffle': True}

#start training, loop through all possible hyper-parameter
for j in leakR_range:
    for k in lambda_range:
        for l in folds:

#####################################Step C ########################################

            #open the file handle of the data used in this training
            with open(os.path.join(file_path,'train_'+str(l)+'.pkl','rb')) as fp:
                train_data = pickle.load(fp)
```

```
    with open(os.path.join(file_path,'val_'+str(l)+'.pkl','rb')) as fp:
        val_data = pickle.load(fp)
    with open(os.path.join(file_path,'test.pkl','rb')) as fp:
        test_data = pickle.load(fp)

    #prepare batches for training
    train_batches = DataGenerator(train_data, labels, **params)
    val_batches = DataGenerator(val_data,labels,**params)
    test_batches = DataGenerator(test_data,labels,**params)

##################################Step D ######################################

    #create model instance, based on input argument

    if model_type =='modela':
        model = modela(96,96,64,dp=dp_range,leakR=j,reg_lambda=k)

    if model_type =='modelb':
        model = modelb(96,96,64,dp=dp_range,leakR=j,reg_lambda=k)

    if model_type =='modelc':
        model = modelc(96,96,64,dp=dp_range,leakR=j,reg_lambda=k)

    if model_type =='modeld':
        model = modeld(96,96,64,dp=dp_range,leakR=j,reg_lambda=k)



    #compile the model
    model.compile(op,loss='categorical_crossentropy', metrics=['accuracy'])

    #set up learning rate schedule
```

```
        reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=3,verbose=1,min_lr=min_rate)

        #set up model saving strategy
        weightfile =
trained_model+'dp_'+str(dp_range)+"_leakrate_"+str(j)+"_lambda_"+str(k)+'_fold_'+str(l)+'.h5'
        checkpointer = ModelCheckpoint(filepath=weightfile, save_best_only=True,verbose=1)

        #initialized training

history=model.fit_generator(generator=train_batches,epochs=ep,verbose=2,callbacks=[reduce_lr,
checkpointer],validation_data=val_batches,shuffle=True)

        #load the best model after training
        model=load_model(weightfile)

        #perform testing
        acc = model.evaluate_generator(test_batches,verbose=2)

        #save training history
        filename =
training_history+"dp_"+str(dp_range)+"_leakrate_"+str(j)+"_lambda_"+str(k)+'_fold_'+str(l)+".
pkl"
        f = open(filename,"wb")
        pickle.dump(history.history,f)
        f.close()

        #output testing result to CSV file
        csv_file = test_output+'dp_'+str(dp_range)+"_leakrate_"+str(j)+"_lambda_"+str(k)+'.csv'

        with open(csv_file, mode='a') as log:
```

```
        log_writer = csv.writer(log)
        log_writer.writerow(['CV Test, Fold='+str(l)])
        log_writer.writerow([str(acc[1])])


################################### Data Generator ##########################
# This Data Gnerator class was modified by the author, based on the tutorial provided at #https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly (Amidi, n.d.). This #routine would prepare training batches according to the given parameter – size, dimensionensions, #etc.
#############################################################################
import numpy as np
import keras
import os
import nibabel as nib

class StreamGeneratorNew(keras.utils.Sequence):
    def __init__(self, data_index, ground_truth_labels, size=4, dimension=(96,96,96), color_mode=1, n_categories=3, is_shuffle=True):
        'Initialization'
        self.dimension = dimension
        self.size = size
        self.ground_truth_labels = ground_truth_labels
        self.data_index = data_index
        self.color_mode = color_mode
        self.n_categories = n_categories
        self.is_shuffle = is_shuffle
        self.end_of_each_epoch()

    def __lenth__(self):
        #Calculate the amount of batches
```

```
    num_batches = int(np.floor(len(self.data_index) / self.size))
    return num_batches

def __genbatch__(self, index):
    # retrieve index for the current batch
    indexes = self.indexes[index*self.size:(index+1)*self.size]

    # Find corresponding data ID
    data_index_temp = [self.data_index[k] for k in indexes]

    # load data object
    Data_array, Label = self.__data_generation(data_index_temp)

    return Data_array, Label

def end_of_each_epoch(self):
    #Obtain new indexes for the next epoch
    self.indexes = np.arange(len(self.data_index))
    if self.is_shuffle == True:
        np.random.shuffle(self.indexes)

def __data_preparation(self, data_index_tmp):
    # Initialization
    Data_array = np.empty((self.size, *self.dimension, self.color_mode))
    Label = np.empty((self.size), dtype=int)

    # retrive data
    outdir = '//scratch//burst//xu640//ADNI1_2Y_NPY'

    for i, ID in enumerate(data_index_tmp):
```

```
xpath = os.path.join(outdir,ID)

#retrive subject ID from file name
tmp = ID.split('.')[0]
tmp = tmp.split('_')[1]+'_'+tmp.split('_')[2]+'_'+tmp.split('_')[3]

#load data based on the file format
if ID.split('.')[1]=='nii' or ID.split('.')[1]=='mgz':

    mri = nib.load(xpath)
    mri_array = np.array(mri.dataobj)
    # Store sample
    Data_array[i,] = mri_array[:,:,:,np.newaxis]
    Label[i]= self.ground_truth_labels[tmp]

elif ID.split('.')[1]=='npy':
    mri = np.load(xpath)

    Data_array[i,]= mri[:,:,:,np.newaxis]

    # Store class
    Label[i]= self.ground_truth_labels[tmp]



Labels = keras.utils.to_categorical(Label, num_classes=self.n_categories)
return Data, Labels
```

APPENDIX B SOURCE CODE FOR INDOOR SCENE UNDERSTANDING

```
import numpy as np
import keras
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.resnet50 import ResNet50
from keras.layers.core import Dense, Activation, Dropout
from keras.models import Model, load_model
from keras.optimizers import SGD
from keras.utils import plot_model
from keras.callbacks import ModelCheckpoint, History
import pickle
import csv
import argparse

# Argument Parser for the program
parser = argparse.ArgumentParser(description="Hyper parameter for training")
parser.add_argument('dp',type=float,help="drop rate")
parser.add_argument('nf',type=int,help="Number of filters")
args = parser.parse_args()



dp_range = args.dp
num_filters = args.nf

#Define Global Parameters and Paths

#LSUN paths
train_path = '//scratch//burst//xu640//lsun//Train'
val_path = '//scratch//burst//xu640//lsun//Val'
```

```
test_path = '//scratch//burst//xu640//lsun//Test'

#Generalization test sets paths
mit_path = '//scratch//gilbreth//xu640//Gen_test'
fif_path = '//scratch//gilbreth//xu640//15_scene'

#Number of Training Epochs

num_ep = 10

#Notation for saving result

dataset = "lsun_whole"
option = "full_fine_tune"

#Preparing training batches/ validation batches / testing batches / generalization test batches
train_batches =
ImageDataGenerator().flow_from_directory(train_path,target_size=(224,224),batch_size=200)
test_batches =
ImageDataGenerator().flow_from_directory(test_path,target_size=(224,224),batch_size=60)
valid_batches =
ImageDataGenerator().flow_from_directory(val_path,target_size=(224,224),batch_size=60)
mit_test_batches =
ImageDataGenerator().flow_from_directory(mit_path,target_size=(224,224),batch_size=10)
fif_test_batches =
ImageDataGenerator().flow_from_directory(fif_path,target_size=(224,224),batch_size=10)

#Save ground truth labels for test sets
np.save('//scratch//gilbreth//xu640//LSUN_whole_result//final//LSUN_dp_'+str(dp_range)+'_nu
mfilter_'+str(num_filters)+'_test_label.npy',test_batches.classes)
```

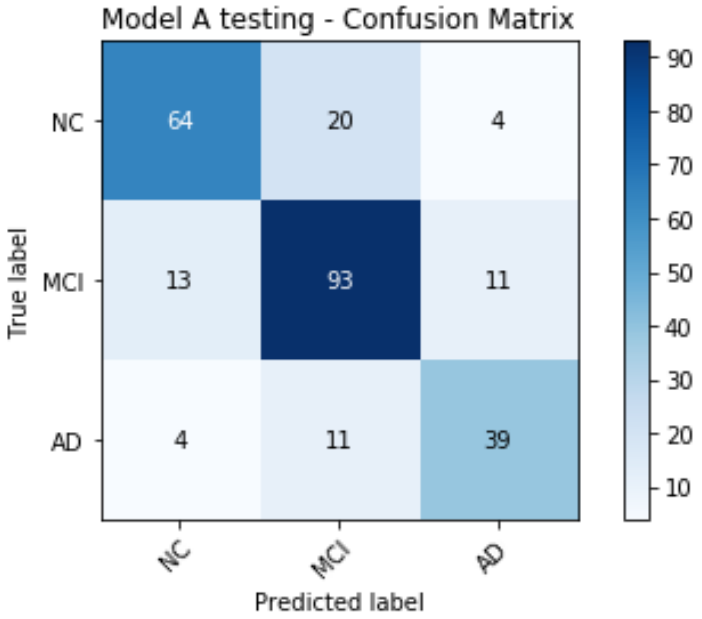
```
np.save('//scratch//gilbreth//xu640//LSUN_whole_result//final//MIT_dp_'+str(dp_range)+'_numf
ilter_'+str(num_filters)+'_test_label.npy',mit_test_batches.classes)
np.save('//scratch//gilbreth//xu640//LSUN_whole_result//final//FIF_dp_'+str(dp_range)+'_numfil
ter_'+str(num_filters)+'_test_label.npy',fif_test_batches.classes)

#Model Initilization
#Load Imagenet pre-trained ResNet 50
feature_model = ResNet50(include_top = True,weights='imagenet')

#remove the last two layers (classifer)
x = feature_model.layers[-2].output

#add dropout layer with dp = dp_range
x = Dropout(dp_range)(x)

#add fully-connected layer with filter amount = num_filters
x = Dense(num_filters, activation='relu')(x)

#add fully-connected layer with three neurons to perform three-class classification
prediction = Dense(3,activation='softmax')(x)

#Model Envelopped
new_model = Model(inputs=feature_model.input, outputs=prediction)

# Model Compilation
new_model.compile(SGD(lr=0.0001, decay=1e-6, momentum=0.9,
nesterov=False),loss='categorical_crossentropy', metrics=['accuracy'])

# Define Training Callbacks
# Define the weight matrix file which stores the best model during training
```

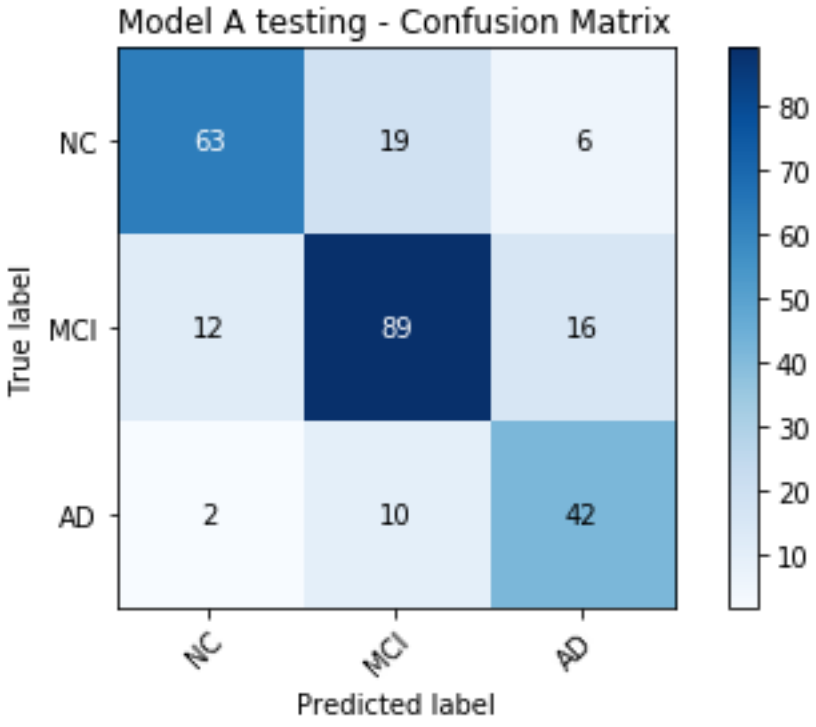
```
weightfile='//scratch//gilbreth//xu640//LSUN_whole_result//final//Model_epoch_'+str(num_ep)+
'_dp_'+str(dp_range)+'_numfilter_'+str(num_filters)+'.h5'

checkpointer = ModelCheckpoint(filepath=weightfile, save_best_only=True,verbose=1)

# Initilize Training
acc_hist=new_model.fit_generator(train_batches,steps_per_epoch=15000,
callbacks=[checkpointer],validation_data=valid_batches,  validation_steps=50,  epochs=num_ep,
verbose=1)

# Save Training History
filename                                                                                =
"//scratch//gilbreth//xu640//LSUN_whole_result//final//history_whole_epoch_"+str(num_ep)+"_
"+dataset+"_"+option+'_dp_'+str(dp_range)+'_numfilter_'+str(num_filters)+".pkl"

f = open(filename,"wb")
pickle.dump(acc_hist.history,f)
f.close()

#Load the best model for testing
model = load_model(weightfile)

#Initilize Testing with evaluation
acc1 = model.evaluate_generator(test_batches, steps=15642, verbose=2)
acc2 = model.evaluate_generator(mit_test_batches, steps=22, verbose=2)
acc3 = model.evaluate_generator(fif_test_batches, steps=51, verbose=2)

# Save Testing performances of the tests to CSV file
csv_file = '//scratch//gilbreth//xu640//LSUN_whole_result//final//output.csv'

with open(csv_file, mode='a') as log:
```

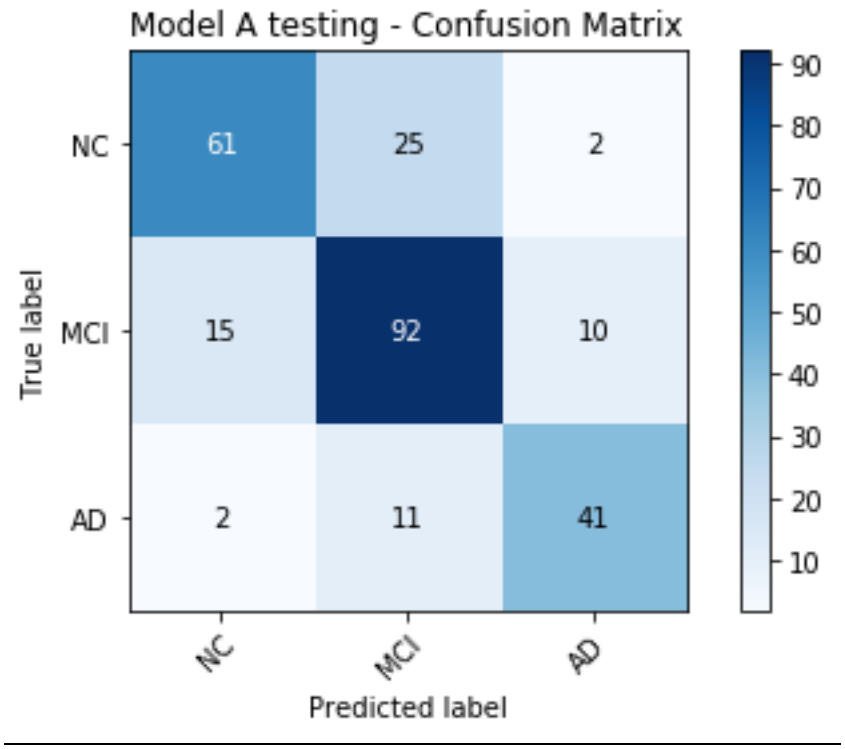
```
        log_writer = csv.writer(log)
        log_writer.writerow(['grid search,dp ='+str(dp_range)+',number of filters
='+str(num_filters)])
        log_writer.writerow([str(acc1[1])])
        log_writer.writerow([str(acc2[1])])
        log_writer.writerow([str(acc3[1])])

#Initilize label prediction for confusion matrix
predict1 = model.predict_generator(test_batches, steps=15642, verbose=2)
predict2 = model.predict_generator(mit_test_batches, steps=22, verbose=2)
predict3 = model.predict_generator(fif_test_batches, steps=51, verbose=2)

#Save predict label
np.save('//scratch//gilbreth//xu640//LSUN_whole_result//final//LSUN_dp_'+str(dp_range)+'_nu
mfilter_'+str(num_filters)+'.npy',predict1)
np.save('//scratch//gilbreth//xu640//LSUN_whole_result//final//MIT_dp_'+str(dp_range)+'_numf
ilter_'+str(num_filters)+'.npy',predict2)
np.save('//scratch//gilbreth//xu640//LSUN_whole_result//final//FIF_dp_'+str(dp_range)+'_numfil
ter_'+str(num_filters)+'.npy',predict3)
```

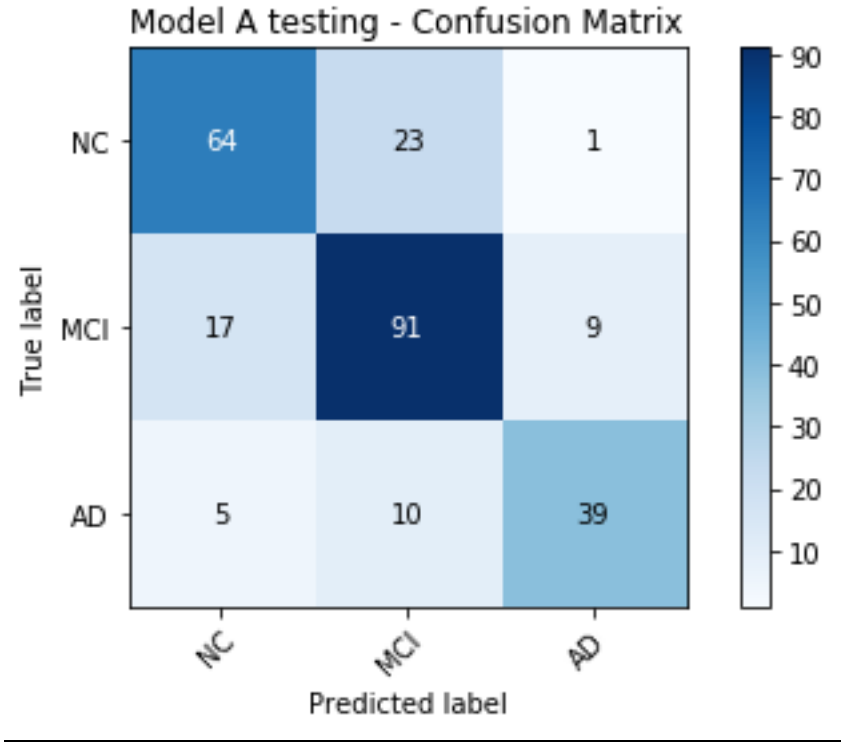
APPENDIX C EXTENDED TESTING RESULTS FOR AUTOMATED AD DIAGNOSIS

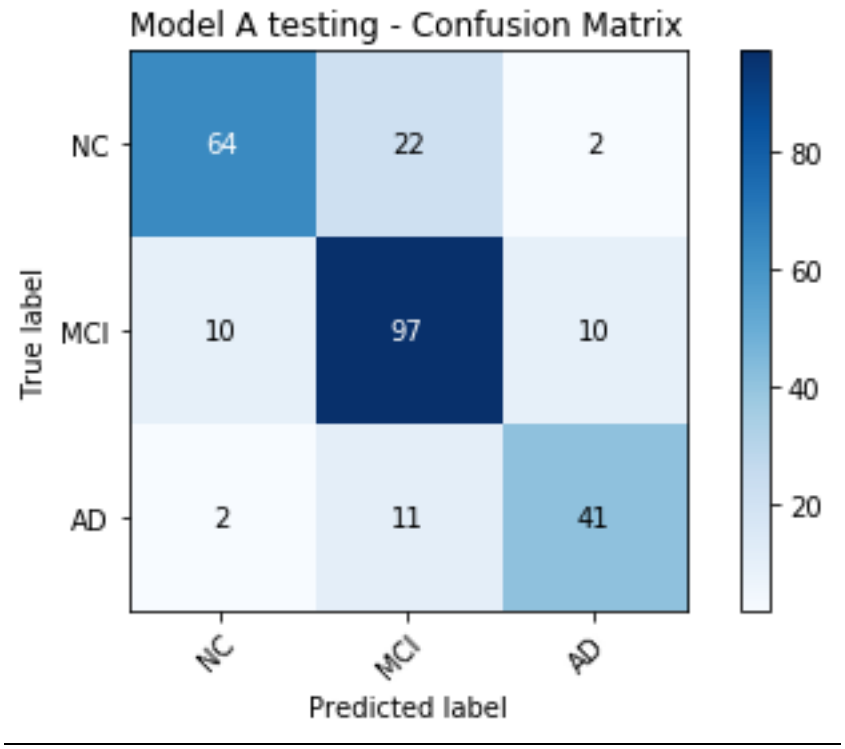
C-1. Model A testing results

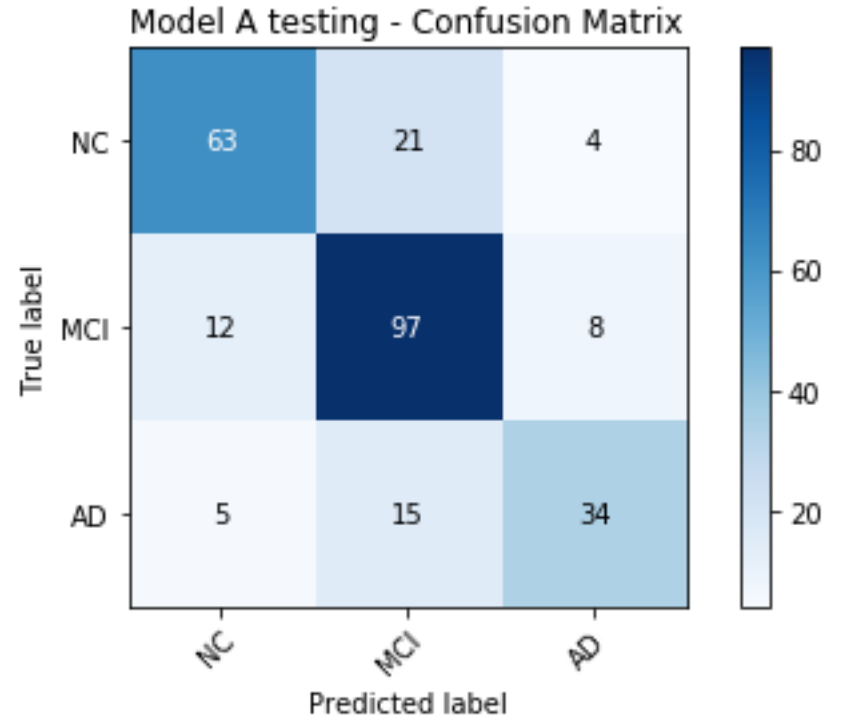




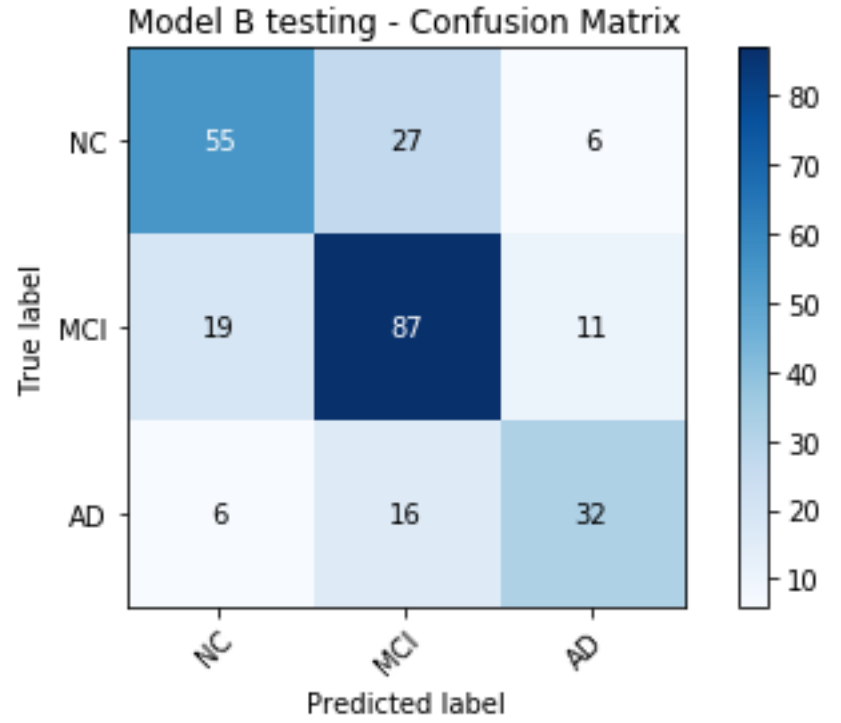

Model A testing - Confusion Matrix
NC
MCI
AD
True label
61
25
2
15
92
10
2
11
41
NC
MCI
AD
Predicted label
90
80
70
60
50
40
30
20
10

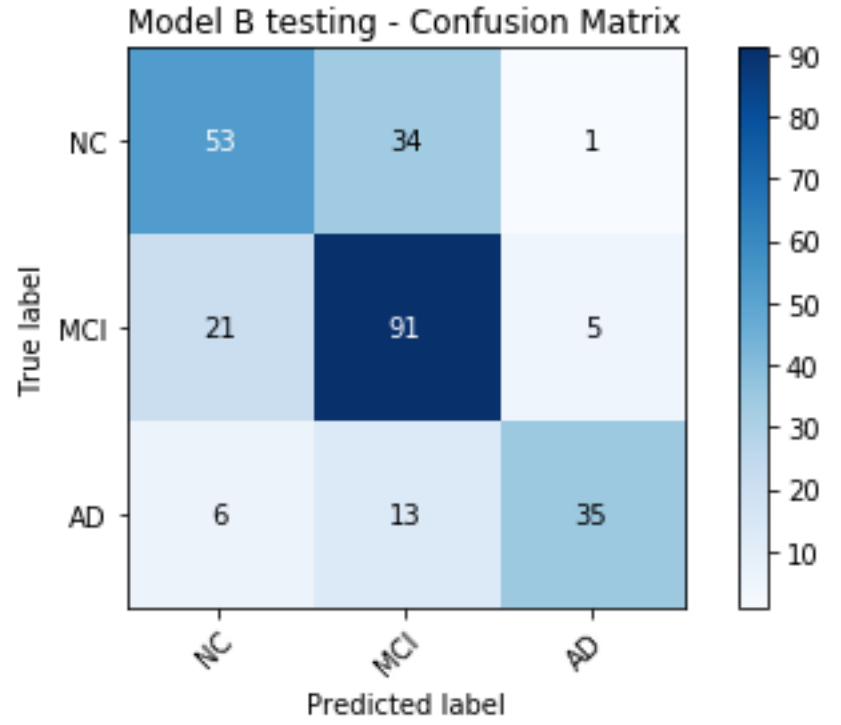

Model A testing - Confusion Matrix
NC
MCI
AD
True label
64
23
1
17
91
9
5
10
39
NC
MCI
AD
Predicted label
90
80
70
60
50
40
30
20
10

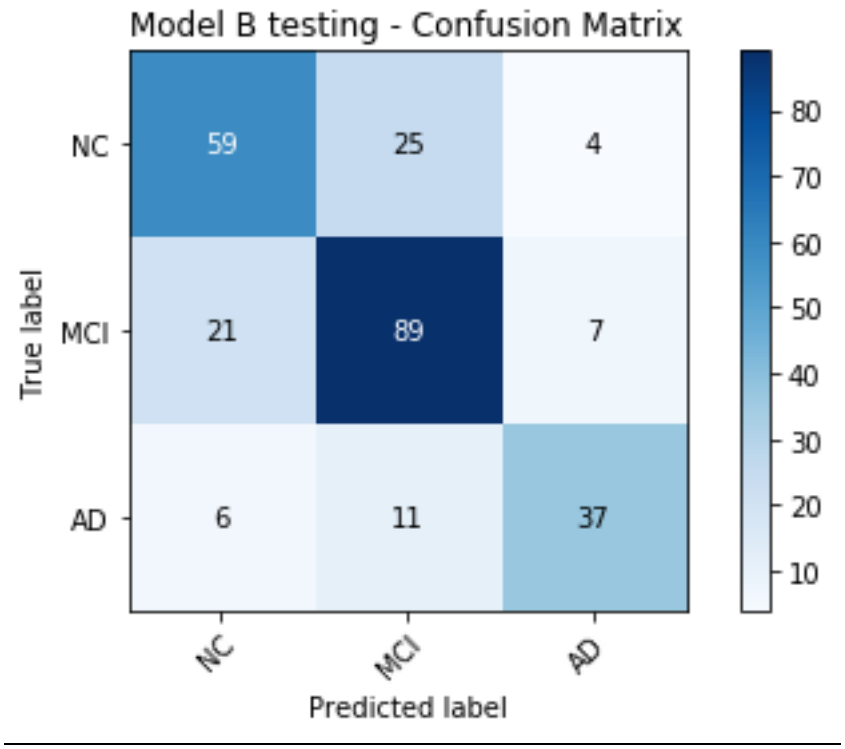

Model A testing - Confusion Matrix
NC
MCI
AD
True label
64
22
2
10
97
10
2
11
41
80
60
40
20
NC
MCI
AD
Predicted label

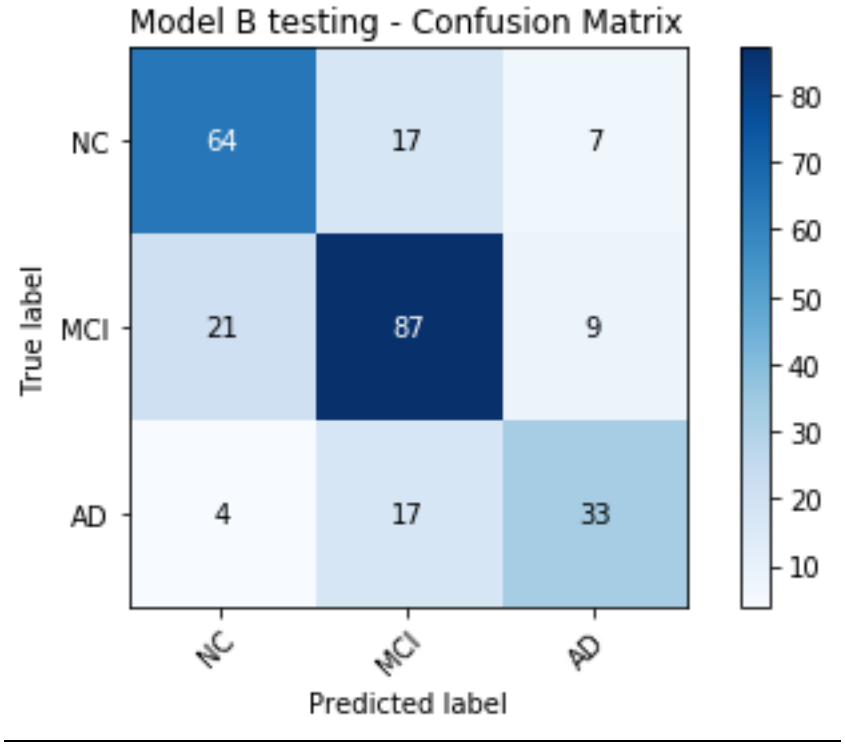

Model A testing - Confusion Matrix
NC
MCI
AD
True label
63
21
4
12
97
8
5
15
34
80
60
40
20
NC
MCI
AD
Predicted label

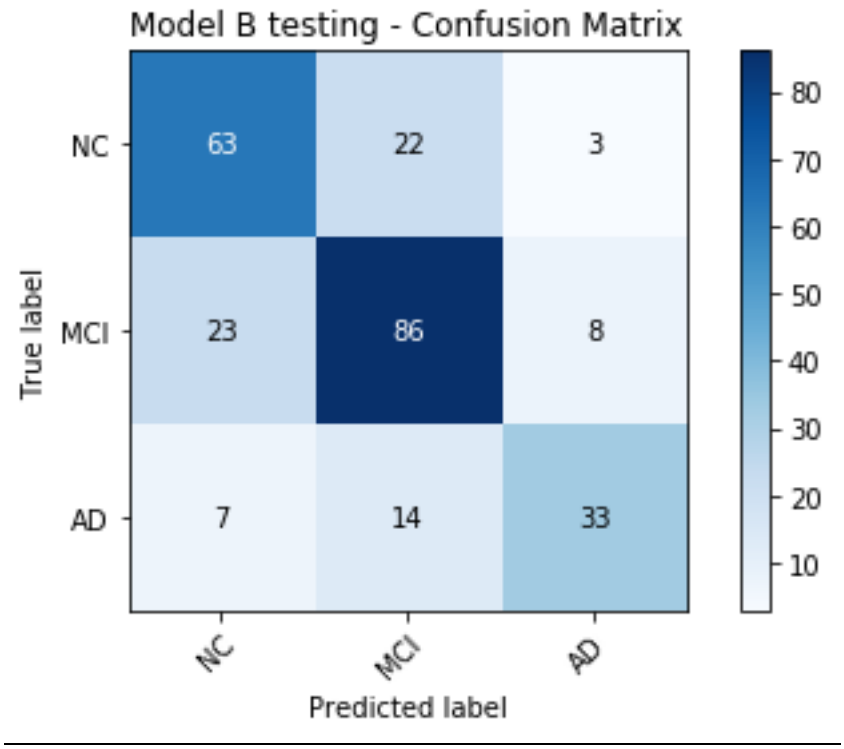
C-2. Model B testing results

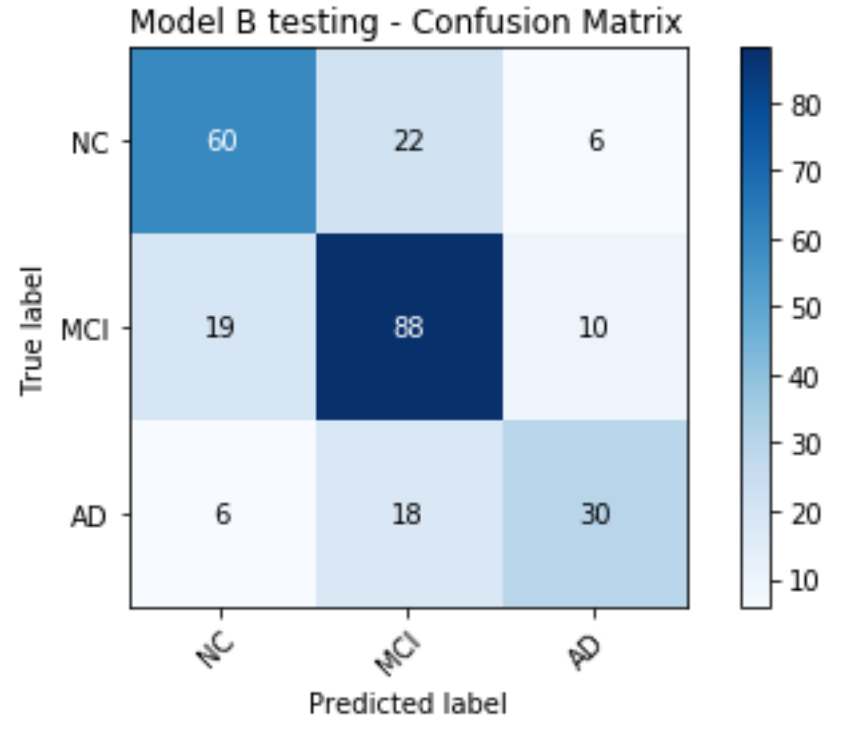




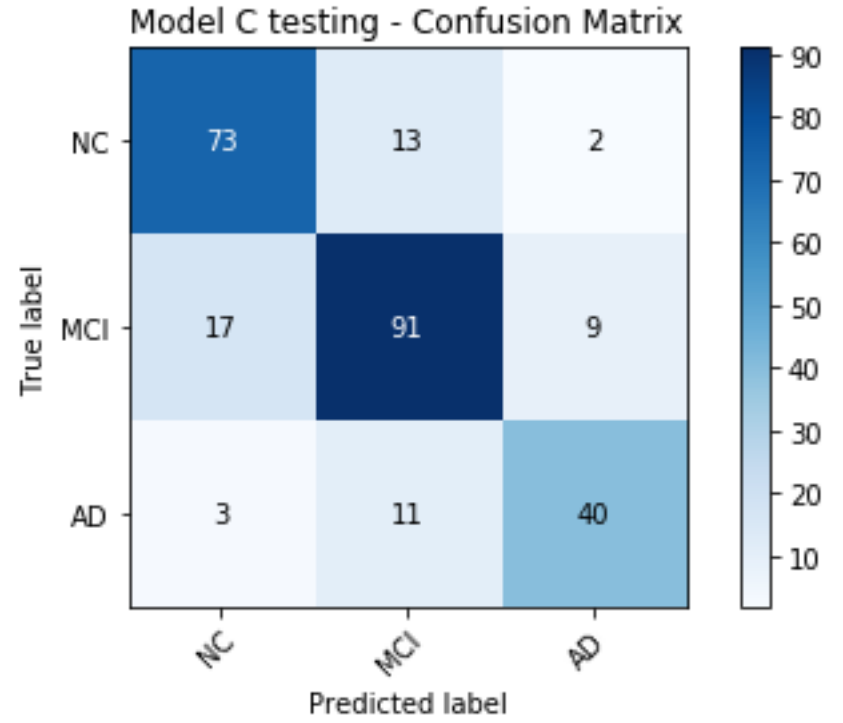

Model B testing - Confusion Matrix
NC
MCI
AD
True label
59
25
4
21
89
7
6
11
37
NC
MCI
AD
Predicted label
80
70
60
50
40
30
20
10

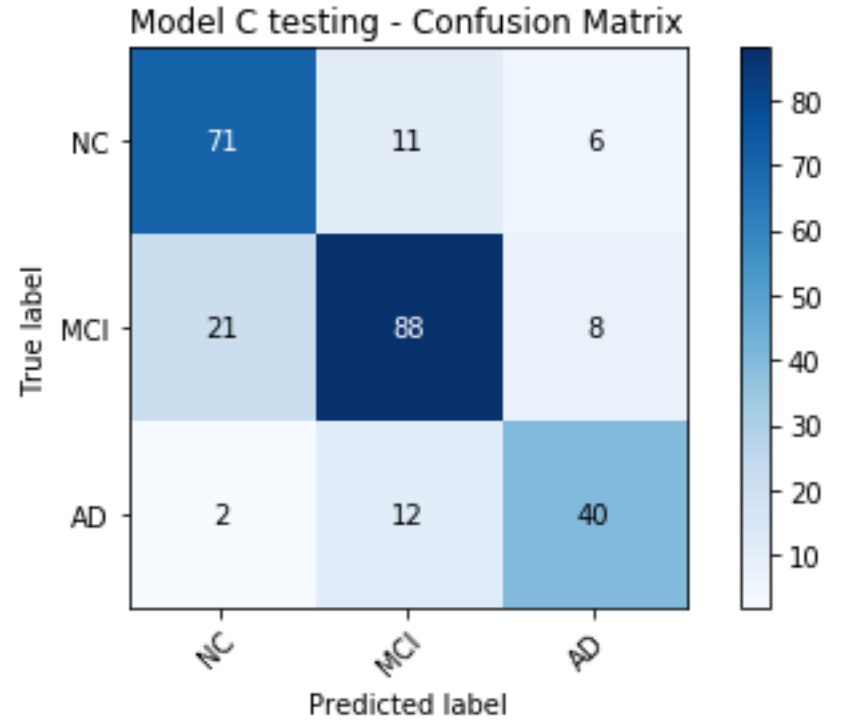

Model B testing - Confusion Matrix
NC
MCI
AD
True label
64
17
7
21
87
9
4
17
33
NC
MCI
AD
Predicted label
80
70
60
50
40
30
20
10

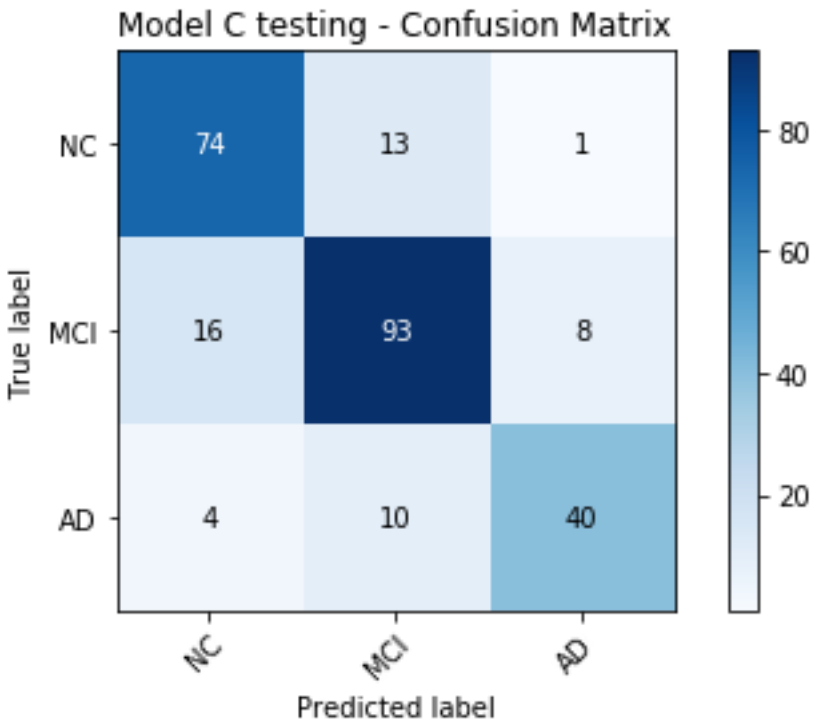

Model B testing - Confusion Matrix
NC
MCI
AD
True label
63
22
3
23
86
8
7
14
33
NC
MCI
AD
Predicted label
80
70
60
50
40
30
20
10

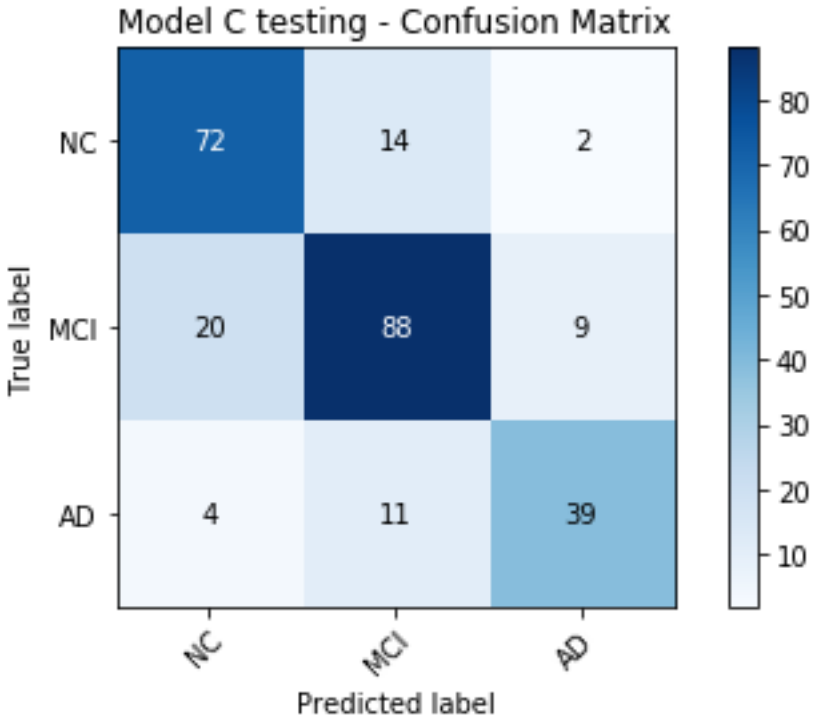

Model B testing - Confusion Matrix
NC
MCI
AD
True label
60
22
6
19
88
10
6
18
30
NC
MCI
AD
Predicted label
80
70
60
50
40
30
20
10

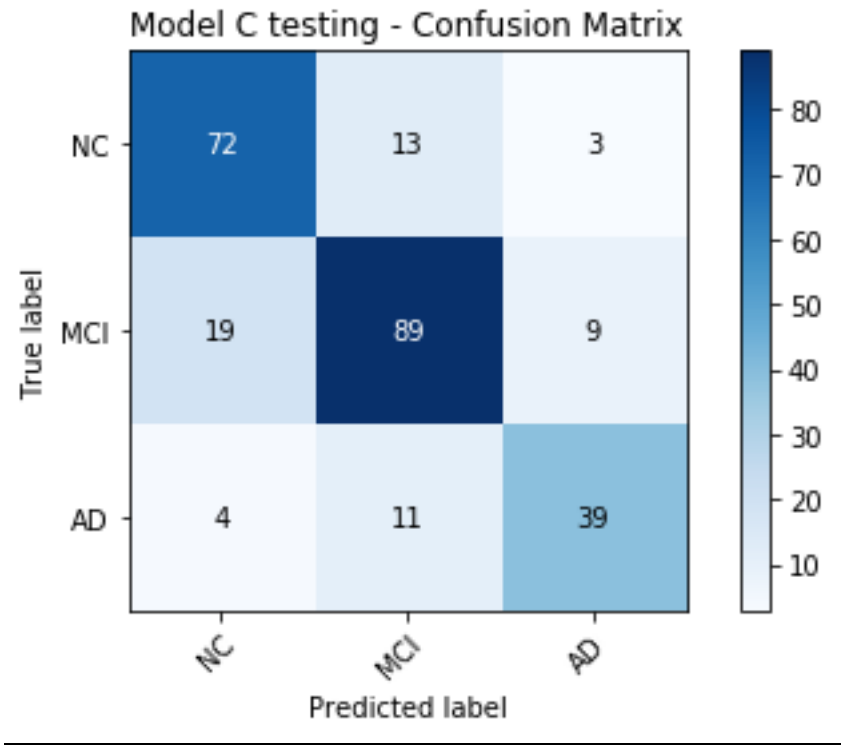
C-3. Model C testing results

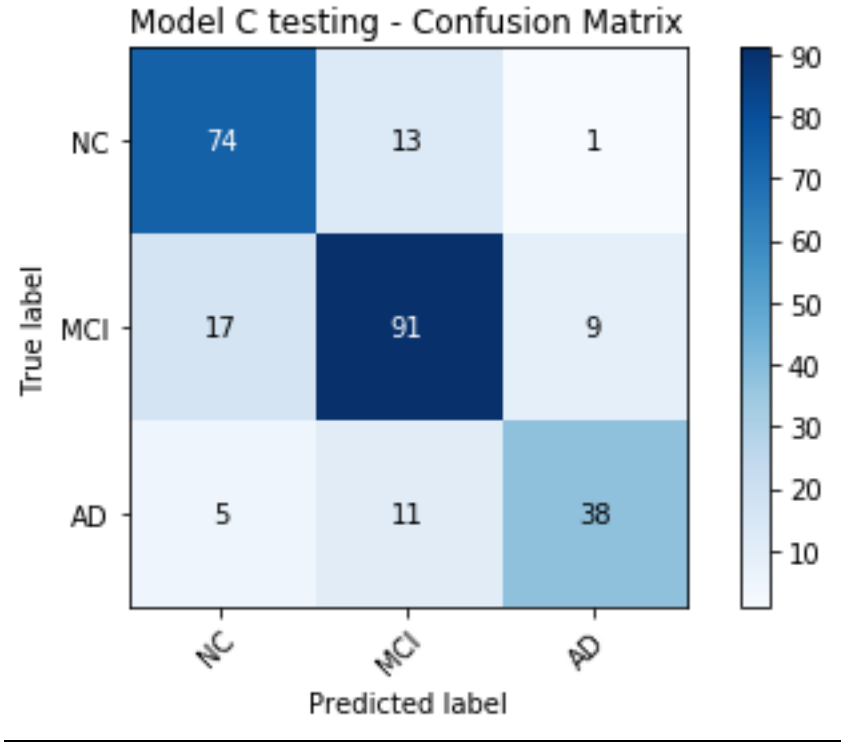




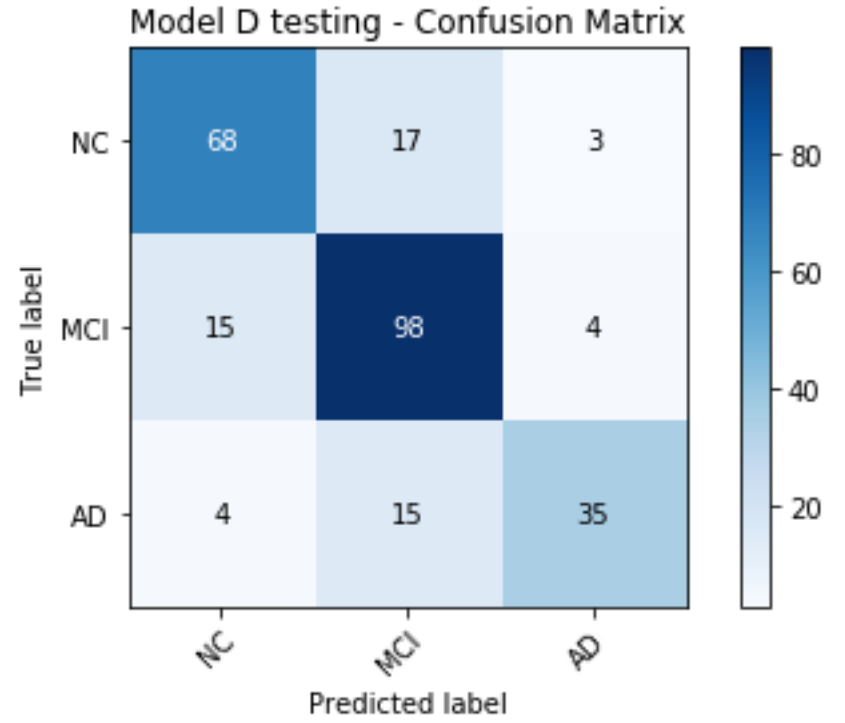

Model C testing - Confusion Matrix
NC
MCI
AD
True label
74
13
1
16
93
8
4
10
40
NC
MCI
AD
Predicted label
80
60
40
20

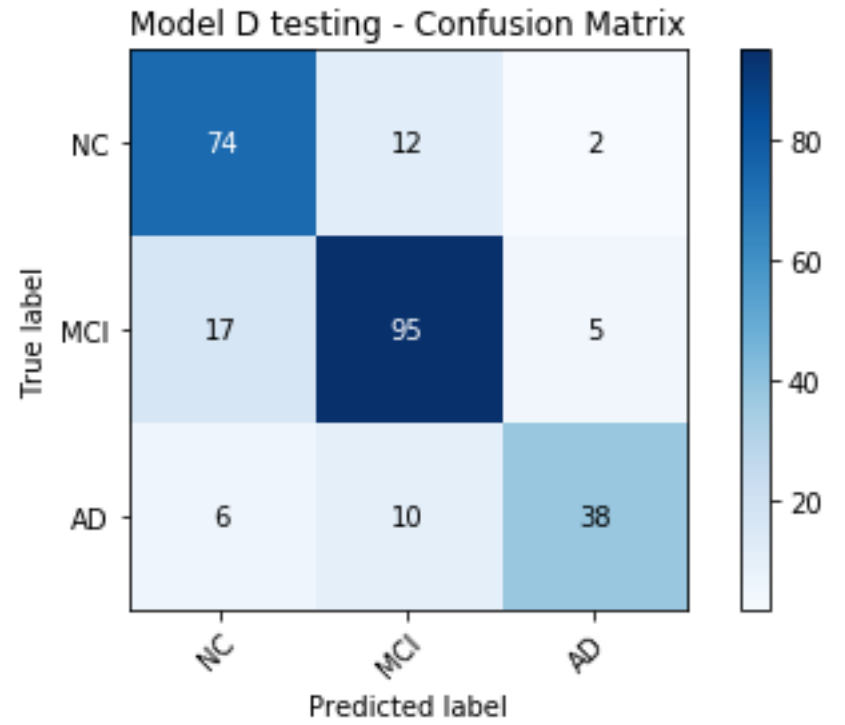

Model C testing - Confusion Matrix
NC
MCI
AD
True label
72
14
2
20
88
9
4
11
39
NC
MCI
AD
Predicted label
80
70
60
50
40
30
20
10

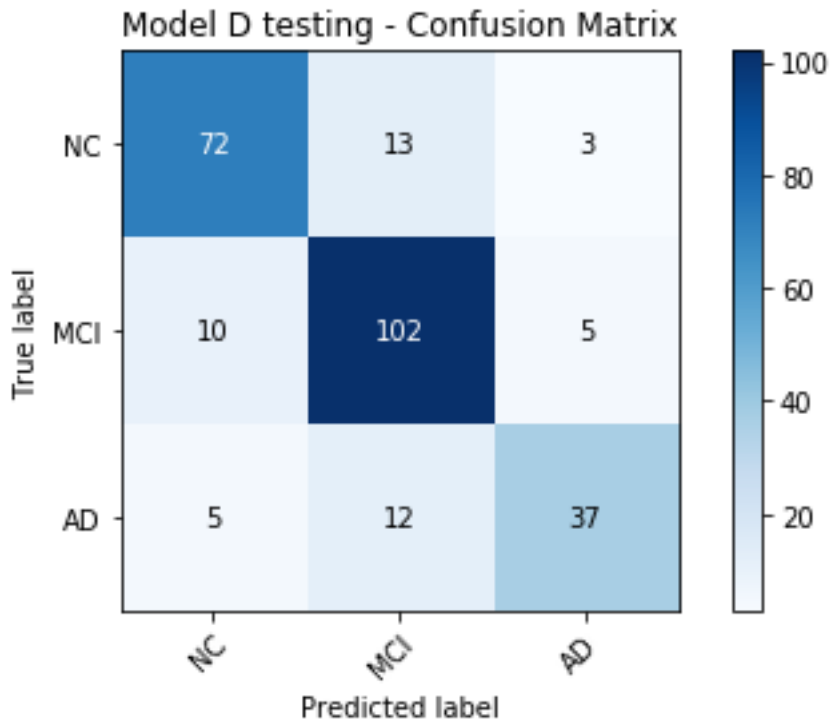

Model C testing - Confusion Matrix
NC
MCI
AD
True label
72
13
3
19
89
9
4
11
39
NC
MCI
AD
Predicted label
80
70
60
50
40
30
20
10

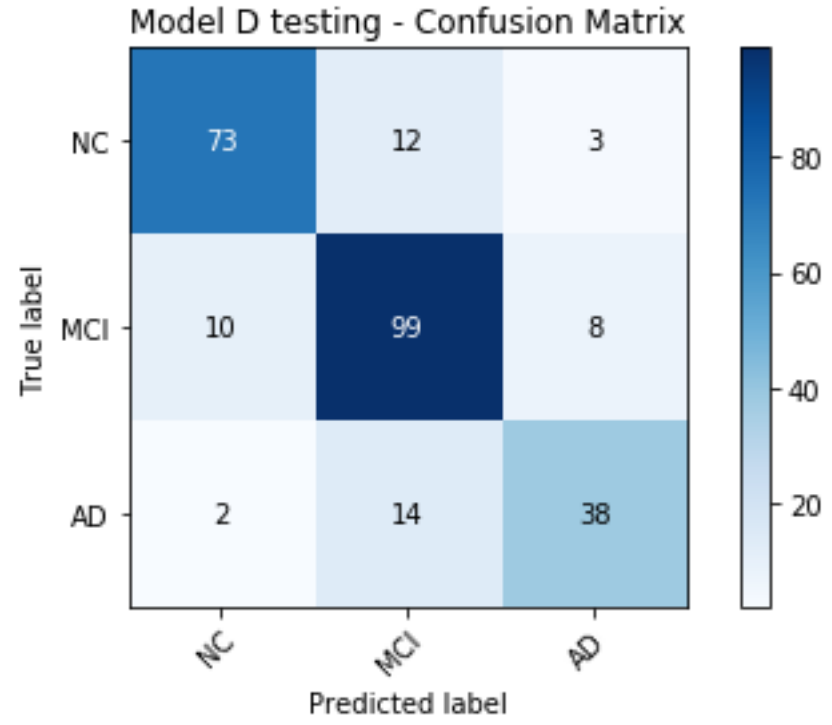

Model C testing - Confusion Matrix
NC
MCI
AD
True label
74
13
1
17
91
9
5
11
38
NC
MCI
AD
Predicted label
90
80
70
60
50
40
30
20
10

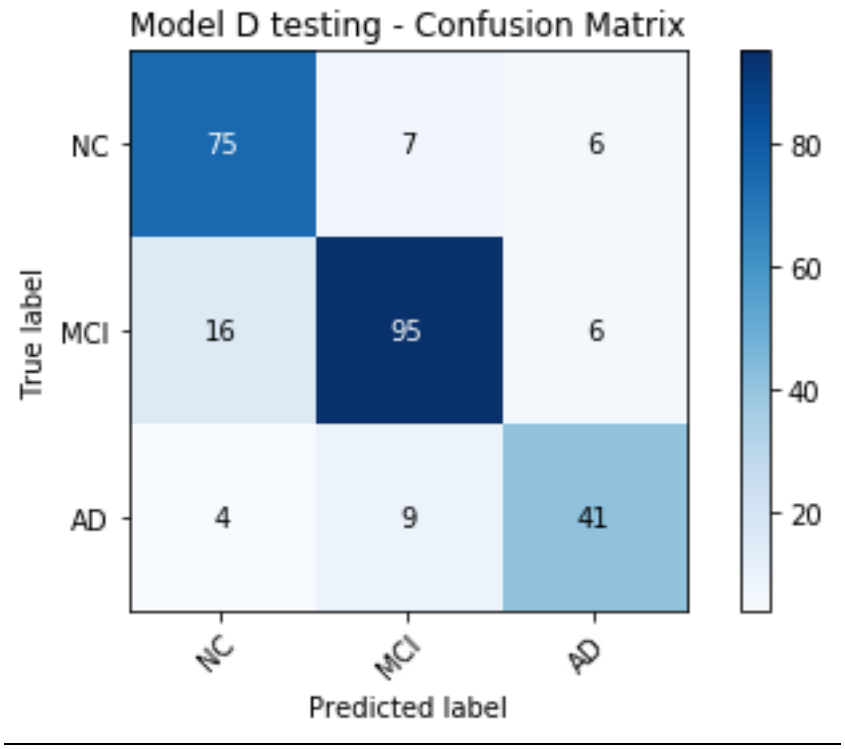
C-4. Model D testing results

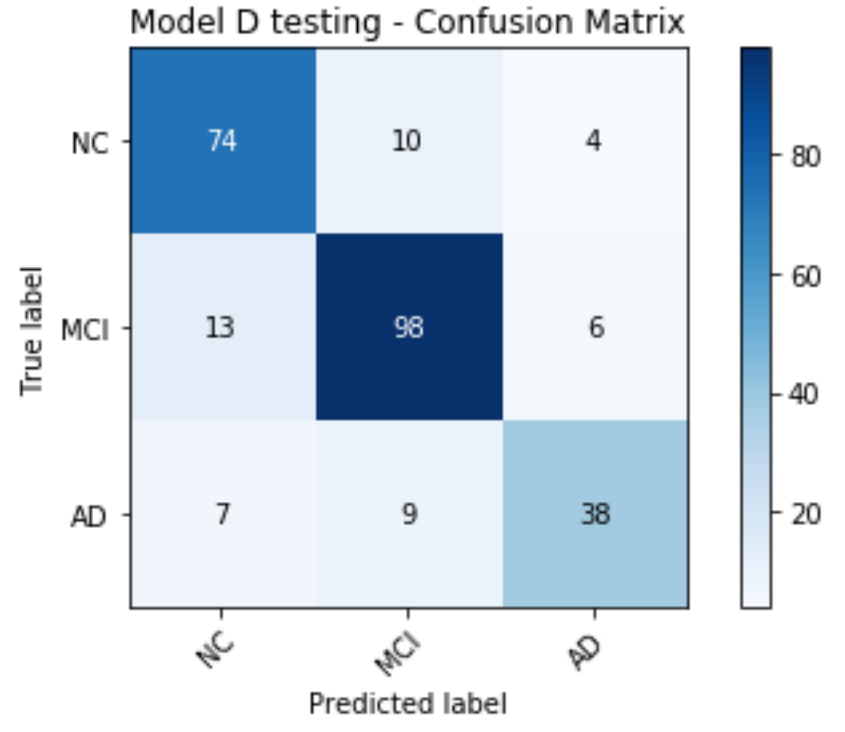





Model D testing - Confusion Matrix
NC
MCI
AD
True label
72
13
3
10
102
5
5
12
37
100
80
60
40
20
NC
MCI
AD
Predicted label


Model D testing - Confusion Matrix
NC
MCI
AD
True label
73
12
3
10
99
8
2
14
38
80
60
40
20
NC
MCI
AD
Predicted label


Model D testing - Confusion Matrix
NC
MCI
AD
True label
75
7
6
16
95
6
4
9
41
NC
MCI
AD
Predicted label
80
60
40
20


Model D testing - Confusion Matrix
NC
MCI
AD
True label
74
10
4
13
98
6
7
9
38
NC
MCI
AD
Predicted label
80
60
40
20

APPENDIX D IRB PROTOCOL

PURDUE
UNIVERSITY

HUMAN RESEARCH PROTECTION PROGRAM
INSTITUTIONAL REVIEW BOARDS

To:	PANIGRAHI, SURANJAN
From:	DICLEMENTI, JEANNIE D, Chair Social Science IRB
Date:	06/07/2019
Committee Action:(4)	Determined Exempt, Category (4)
IRB Action Date:	06 / 07 / 2019
IRB Protocol #:	1906022293
Study Title:	Multi-Class Alzheimer's Disease Diagnosis using Deep Learning

The Institutional Review Board (IRB) has reviewed the above-referenced study application and has determined that it meets the criteria for exemption under 45 CFR 46.101(b).

Before making changes to the study procedures, please submit an Amendment to ensure that the regulatory status of the study has not changed. Changes in key research personnel should also be submitted to the IRB through an amendment.

General

- To recruit from Purdue University classrooms, the instructor and all others associated with conduct of the course (e.g., teaching assistants) must not be present during announcement of the research opportunity or any recruitment activity. This may be accomplished by announcing, in advance, that class will either start later than usual or end earlier than usual so this activity may occur. It should be emphasized that attendance at the announcement and recruitment are voluntary and the student's attendance and enrollment decision will not be shared with those administering the course.
- If students earn extra credit towards their course grade through participation in a research project conducted by someone other than the course instructor(s), such as in the example above, the students participation should only be shared with the course instructor(s) at the end of the semester. Additionally, instructors who allow extra credit to be earned through participation in research must also provide an opportunity for students to earn comparable extra credit through a non-research activity requiring an amount of time and effort comparable to the research option.
- When conducting human subjects research at a non-Purdue college/university, investigators are urged to contact that institution's IRB to determine requirements for conducting research at that institution.
- When human subjects research will be conducted in schools or places of business, investigators must obtain written permission from an appropriate authority within the organization. If the written permission was not submitted with the study application at the time of IRB review (e.g., the school would not issue the letter without proof of IRB approval, etc.), the investigator must submit the written permission to the IRB prior to engaging in the research activities (e.g., recruitment, study procedures, etc.). Submit this documentation as an FYI through Coeus. This is an institutional requirement.

Categories 2 and 3

- Surveys and questionnaires should indicate
  - only participants 18 years of age and over are eligible to participate in the research; and
  - that participation is voluntary; and
  - that any questions may be skipped; and
  - include the investigator's name and contact information.
- Investigators should explain to participants the amount of time required to participate. Additionally, they should explain to participants how confidentiality will be maintained or if it will not be maintained.
- When conducting focus group research, investigators cannot guarantee that all participants in the focus group will maintain the confidentiality of other group participants. The investigator should make participants aware of this potential for breach of confidentiality.

Category 6

- Surveys and data collection instruments should note that participation is voluntary.
- Surveys and data collection instruments should note that participants may skip any questions.
- When taste testing foods which are highly allergenic (e.g., peanuts, milk, etc.) investigators should disclose the possibility of a reaction to potential subjects.

You are required to retain a copy of this letter for your records. We appreciate your commitment towards ensuring the ethical conduct of human subjects research and wish you luck with your study.

LIST OF REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., . . . Kudlur, M. (2016). Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (pp. 265-283).

Adelstein, J. S., Shehzad, Z., Mennes, M., DeYoung, C. G., Zuo, X. N., Kelly, C., . . . Milham, M. P. (2011). Personality is reflected in the brain's intrinsic functional architecture. PloS one, 6(11), e27633. doi: 10.1371/journal.pone.0027633

Aderghal, K., Benois-Pineau, J., Afdel, K., & Gwenaëlle, C. (2017, June). FuseMe: Classification of sMRI images by fusion of deep CNNs in 2D+ ϵ projections. In Proceedings of the 15th International Workshop on Content-Based Multimedia Indexing (p. 34). ACM.

Akiba, T., Suzuki, S., & Fukuda, K. (2017). Extremely large minibatch SGD: Training resnet-50 on imagenet in 15 minutes. arXiv:1711.04325.

Alonso, J. M., & Chen, Y. (2009). Receptive field. Scholarpedia, 4(1), 5393.

Amidi, S. (n.d.). A detailed example of how to use data generators with Keras. Retrieved from https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly. Accessed on April 2019.

Ayers, B., & Boutell, M. (2007, June). Home interior classification using SIFT keypoint histograms. In 2007 IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-6). IEEE.

Baron, J. C., Chetelat, G., Desgranges, B., Perchey, G., Landeau, B., De La Sayette, V., & Eustache, F. (2001). In vivo mapping of gray matter loss with voxel-based morphometry in mild Alzheimer's disease. Neuroimage, 14(2), 298-309. doi: 10.1006/nimg.2001.0848

Berchtold, N. C., & Cotman, C. W. (1998). Evolution in the conceptualization of dementia and Alzheimer's disease: Greco-Roman period to the 1960s. Neurobiology of Aging, 19(3), 173-189. doi: 10.1016/s0197-4580(98)00052-9

Bhatkoti, P., & Paul, M. (2016, November). Early diagnosis of Alzheimer's disease: A multi-class deep learning framework with modified k-sparse autoencoder classification. In 2016 International Conference on Image and Vision Computing New Zealand (pp. 1-5). IEEE.

Billones, C. D., Demetria, O. J. L. D., Hostallero, D. E. D., & Naval, P. C. (2016, November). DemNet: A convolutional neural network for the detection of Alzheimer's disease and mild cognitive impairment. In 2016 IEEE Region 10 Conference (pp. 3724-3727). IEEE.

Brett, M., Hanke, M., Markiewicz, C., Côté, M. A., McCarthy, P., Ghosh, S., & Wassermann, D. (2018). nipy/nibabel: 2.3.0 (Version 2.3.0). Zenodo. doi:10.5281/zenodo.1287921.

Busatto, G. F., Garrido, G. E., Almeida, O. P., Castro, C. C., Camargo, C. H., Cid, C. G., ... & Bottino, C. M. (2003). A voxel-based morphometry study of temporal lobe gray matter reductions in Alzheimer's disease. Neurobiology of Aging, 24(2), 221-231. doi: 10.1016/s0197-4580(02)00084-2

Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. arXiv:1605.07678.

Cheng, D., & Liu, M. (2017, October). Combining convolutional and recurrent neural networks for Alzheimer's disease diagnosis using PET images. In 2017 IEEE International Conference on Imaging Systems and Techniques (pp. 1-5). IEEE.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., & Shelhamer, E. (2014). cudnn: Efficient primitives for deep learning. arXiv:1410.0759.

Chollet, F and others. (2015). Keras. https://keras.io. Accessed from August 2018 to June 2019.

Colliot, O., Chételat, G., Chupin, M., Desgranges, B., Magnin, B., Benali, H., . . . Lehéricy, S. (2008). Discrimination between Alzheimer disease, mild cognitive impairment, and normal aging by using automated segmentation of the hippocampus. Radiology, 248(1), 194-201. doi: 10.1148/radiol.2481070876

Convit, A., De Asis, J., De Leon, M. J., Tarshish, C. Y., De Santi, S., & Rusinek, H. (2000). Atrophy of the medial occipitotemporal, inferior, and middle temporal gyri in non-demented elderly predict decline to Alzheimer's disease. Neurobiology of Aging, 21(1), 19-26. doi: 10.1016/s0197-4580(99)00107-4

Corrada, M. M., Brookmeyer, R., Paganini-Hill, A., Berlau, D., & Kawas, C. H. (2010). Dementia incidence continues to increase with age in the oldest old: the 90+ study. Annals of Neurology, 67(1), 114-121. doi: 10.1002/ana.21915

Davatzikos, C., Fan, Y., Wu, X., Shen, D., & Resnick, S. M. (2008). Detection of prodromal Alzheimer's disease via pattern classification of magnetic resonance imaging. Neurobiology of Aging, 29(4), 514-523. doi: 10.1016/j.neurobiolaging.2006.11.010

De Leon, M. J., George, A. E., Golomb, J., Tarshish, C., Convit, A., Kluger, A., ... & Ince, C. (1997). . Neurobiology of aging, 18(1), 1-11. doi: 10.1016/s0197-4580(96)00213-8

Desikan, R. S., Cabral, H. J., Hess, C. P., Dillon, W. P., Glastonbury, C. M., Weiner, M. W., . . . Fischl, B. (2009). Automated MRI measures identify individuals with mild cognitive impairment and Alzheimer's disease. Brain, 132(8), 2048-2057. doi: 10.1093/brain/awp123

Domingos, P. (2012). A few useful things to know about machine learning. Communications of the ACM, 55(10), 78. doi:10.1145/2347736.2347755

Dubois, B., & Albert, M. L. (2004). Amnestic MCI or prodromal Alzheimer's disease? The Lancet Neurology, 3(4), 246-248. doi: 10.1016/s1474-4422(04)00710-0

Dubois, B., Feldman, H. H., Jacova, C., DeKosky, S. T., Barberger-Gateau, P., Cummings, J., . . . Meguro, K. (2007). Research criteria for the diagnosis of Alzheimer's disease: revising the NINCDS–ADRDA criteria. The Lancet Neurology, 6(8), 734-746. doi: 10.1016/s1474-4422(07)70178-3

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.

Duthey, B. (2013). Background paper 6.11: Alzheimer disease and other dementias. Retrieved from World Health Organization website: https://www.who.int/medicines/areas/priority_medicines/BP6_11Alzheimer.pdf

Eldan, R., & Shamir, O. (2016, June). The power of depth for feedforward neural networks. In Conference on learning theory (pp. 907-940).

Fan, Y., Shen, D., Gur, R. C., Gur, R. E., & Davatzikos, C. (2006). COMPARE: classification of morphological patterns using adaptive regional elements. IEEE Transactions on Medical Imaging, 26(1), 93-105. doi:10.1109/tmi.2006.886812

Fei-Fei, L., & Perona, P. (2005, June). A bayesian hierarchical model for learning natural scene categories. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Vol. 2, pp. 524-531). IEEE.

Feng, C., Elazab, A., Yang, P., Wang, T., Lei, B., & Xiao, X. (2018, September). 3D convolutional neural network and stacked bidirectional recurrent neural network for Alzheimer’s disease diagnosis. In International Workshop on Predictive Intelligence In Medicine (pp. 138-146). Springer, Cham.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1), 119-139. doi:10.1006/jcss.1997.1504

Gaugler, J., James, B., Johnson, T., Marin, A., & Weuve, J. (2019). 2019 Alzheimer's disease facts and figures. Alzheimer's & Dementia, 15(3), 321-387. doi: 10.1016/j.jalz.2019.01.010

Gerardin, E., Chételat, G., Chupin, M., Cuingnet, R., Desgranges, B., Kim, H. S., . . . Eustache, F. (2009). Multidimensional classification of hippocampal shape features discriminates Alzheimer's disease and mild cognitive impairment from normal aging. Neuroimage, 47(4), 1476-1486. doi: 10.1016/j.neuroimage.2009.05.036

Good, C. D., Scahill, R. I., Fox, N. C., Ashburner, J., Friston, K. J., Chan, D., . . . Frackowiak, R. S. (2002). Automatic differentiation of anatomical patterns in the human brain: validation with studies of degenerative dementias. Neuroimage, 17(1), 29-46. doi: 10.1006/nimg.2002.1202

Gupta, S., Arbeláez, P., Girshick, R., & Malik, J. (2015). Indoor scene understanding with rgb-d images: Bottom-up segmentation, object detection and semantic segmentation. International Journal of Computer Vision, 112(2), 133-149. doi: 10.1007/s11263-014-0777-6

Hashimoto, M., Rockenstein, E., Crews, L., & Masliah, E. (2003). Role of protein aggregation in mitochondrial dysfunction and neurodegeneration in Alzheimer's and Parkinson's diseases. Neuromolecular Medicine, 4(1-2), 21-35. doi: 10.1385/nmm:4:1-2:21

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Hebert, L. E., Beckett, L. A., Scherr, P. A., & Evans, D. A. (2001). Annual incidence of Alzheimer disease in the United States projected to the years 2000 through 2050. Alzheimer Disease & Associated Disorders, 15(4), 169-173. doi: 10.1097/00002093-200110000-00002

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507. doi: 10.1126/science.1127647

Honey, C. J., Sporns, O., Cammoun, L., Gigandet, X., Thiran, J. P., Meuli, R., & Hagmann, P. (2009). Predicting human resting-state functional connectivity from structural connectivity. Proceedings of the National Academy of Sciences, 106(6), 2035-2040. doi: 10.1073/pnas.0811168106

Hosseini-Asl, E., Keynton, R., & El-Baz, A. (2016, September). Alzheimer's disease diagnostics by adaptation of 3D convolutional network. In 2016 IEEE international conference on image processing (pp. 126-130). IEEE.

Hu, C., Ju, R., Shen, Y., Zhou, P., & Li, Q. (2016, May). Clinical decision support for Alzheimer's disease based on deep learning and brain network. In 2016 IEEE international conference on communications (pp. 1-6). IEEE.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).

Iglesias, J. E., Liu, C. Y., Thompson, P. M., & Tu, Z. (2011). Robust brain extraction across datasets and comparison with publicly available methods. IEEE transactions on medical imaging, 30(9), 1617-1634.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Jung, H., Choi, M. K., Jung, J., Lee, J. H., Kwon, S., & Young Jung, W. (2017). ResNet-based vehicle classification and localization in traffic surveillance systems. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 61-67).

Juottonen, K., Laakso, M. P., Insausti, R., Lehtovirta, M., Pitkänen, A., Partanen, K., & Soininen, H. (1998). Volumes of the entorhinal and perirhinal cortices in Alzheimer's disease. Neurobiology of Aging, 19(1), 15-22. doi: 10.1016/s0197-4580(98)00007-4

Karasawa, H., Liu, C. L., & Ohwada, H. (2018, March). Deep 3d convolutional neural network architectures for Alzheimer's disease diagnosis. In Asian Conference on Intelligent Information and Database Systems (pp. 287-296). Springer, Cham.

Khan, S. H., Hayat, M., Bennamoun, M., Togneri, R., & Sohel, F. A. (2016). A discriminative representation of convolutional features for indoor scene recognition. IEEE Transactions on Image Processing, 25(7), 3372-3383. doi: 10.1109/tip.2016.2567076

Khvostikov, A., Aderghal, K., Krylov, A., Catheline, G., & Benois-Pineau, J. (2018). 3D Inception-based CNN with sMRI and MD-DTI data fusion for Alzheimer's disease diagnostics. arXiv:1809.03972.

Kim, K. J., Hassan, M. M., Na, S. H., & Huh, E. N. (2009, December). Dementia wandering detection and activity recognition algorithm using tri-axial accelerometer sensors. In Proceedings of the 4th International Conference on Ubiquitous Information Technologies & Applications (pp. 1-5). IEEE.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Klöppel, S., Stonnington, C. M., Chu, C., Draganski, B., Scahill, R. I., Rohrer, J. D., . . . Frackowiak, R. S. (2008). Automatic classification of MR scans in Alzheimer's disease. Brain, 131(3), 681-689. doi: 10.1093/brain/awm319

Ko, C. Y., Leu, F. Y., & Lin, I. T. (2014, November). A wandering path tracking and fall detection system for people with dementia. In 2014 Ninth International Conference on Broadband and Wireless Computing, Communication and Applications (pp. 306-311). IEEE.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

Kuhn, M., & Johnson, K. (2013). Applied predictive modeling (Vol. 26). New York: Springer.

Lao, Z., Shen, D., Xue, Z., Karacali, B., Resnick, S. M., & Davatzikos, C. (2004). Morphological classification of brains via high-dimensional shape transformations and machine learning methods. Neuroimage, 21(1), 46-57. doi: 10.1016/j.neuroimage.2003.09.027

Lazebnik, S., Schmid, C., & Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Vol. 2, pp. 2169-2178). IEEE.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. doi: 10.1109/5.726791

Lerch, J. P., Pruessner, J. C., Zijdenbos, A., Hampel, H., Teipel, S. J., & Evans, A. C. (2004). Focal decline of cortical thickness in Alzheimer's disease identified by computational neuroanatomy. Cerebral Cortex, 15(7), 995-1001. doi: 10.1093/cercor/bhh200

Li, F., Cheng, D., & Liu, M. (2017, October). Alzheimer's disease classification based on combination of multi-model convolutional networks. In 2017 IEEE International Conference on Imaging Systems and Techniques (pp. 1-5). IEEE.

Li, F., Tran, L., Thung, K. H., Ji, S., Shen, D., & Li, J. (2015). A robust deep model for improved classification of AD/MCI patients. IEEE Journal of Biomedical and Health Informatics, 19(5), 1610-1616. doi: 10.1109/jbhi.2015.2429556

Lin, Q., Zhang, D., Huang, X., Ni, H., & Zhou, X. (2012, December). Detecting wandering behavior based on GPS traces for elders with dementia. In 2012 12th International Conference on Control Automation Robotics & Vision (pp. 672-677). IEEE.

Liu, M., Cheng, D., Wang, K., Wang, Y., & Alzheimer’s Disease Neuroimaging Initiative. (2018). Multi-modality cascaded convolutional neural networks for Alzheimer’s disease diagnosis. Neuroinformatics, 1-14. doi:10.1007/s12021-018-9370-4

Lowe, D. G. (1999, September). Object recognition from local scale-invariant features. In International Conference on Computer Vision (Vol. 99, No. 2, pp. 1150-1157).

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013, June). Rectifier nonlinearities improve neural network acoustic models. In Proc. icml (Vol. 30, No. 1, p. 3).

Manjón, J. V., Coupé, P., Buades, A., Collins, D. L., & Robles, M. (2012). New methods for MRI denoising based on sparseness and self-similarity. Medical Image Analysis, 16(1), 18-27. doi: 10.1016/j.media.2011.04.003

McKhann, G. M., Knopman, D. S., Chertkow, H., Hyman, B. T., Jack Jr, C. R., Kawas, C. H., . . . Mohs, R. C. (2011). The diagnosis of dementia due to Alzheimer's disease: Recommendations from the National Institute on Aging-Alzheimer's Association workgroups on diagnostic guidelines for Alzheimer's disease. Alzheimer's & Dementia, 7(3), 263-269. doi: 10.1016/j.jalz.2011.03.005

Meindl, T., Teipel, S., Elmouden, R., Mueller, S., Koch, W., Dietrich, O., . . . Glaser, C. (2010). Test–retest reproducibility of the default-mode network in healthy individuals. Human Brain Mapping, 31(2), 237-246. doi:10.1002/hbm.20860

Mirsky, Y., Mahler, T., Shelef, I., & Elovici, Y. (2019). CT-GAN: Malicious tampering of 3D medical imagery using deep learning. In 28th USENIX security symposium (USENIX Security 19). Berkeley, CA: USENIX Association. arXiv:1901.03597.

Misra, C., Fan, Y., & Davatzikos, C. (2009). Baseline and longitudinal patterns of brain atrophy in MCI patients, and their use in prediction of short-term conversion to AD: results from ADNI. Neuroimage, 44(4), 1415-1422. doi: 10.1016/j.neuroimage.2008.10.031

Morabito, F. C., Campolo, M., Ieracitano, C., Ebadi, J. M., Bonanno, L., Bramanti, A., . . . Bramanti, P. (2016, September). Deep convolutional neural networks for classification of mild cognitive impaired and Alzheimer's disease patients from scalp EEG recordings. In 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (pp. 1-6). IEEE.

Mozos, O. M., Stachniss, C., & Burgard, W. (2005, April). Supervised learning of places from range data using adaboost. In Proceedings of the 2005 IEEE International Conference on Robotics and Automation (pp. 1730-1735). IEEE.

Mozos, O. M., Triebel, R., Jensfelt, P., Rottmann, A., & Burgard, W. (2007). Supervised semantic labeling of places using information extracted from sensor data. Robotics and Autonomous Systems, 55(5), 391-402. doi: 10.1016/j.robot.2006.12.003

Mugler, J. P., & Brookeman, J. R. (1990). Three-dimensional magnetization-prepared rapid gradient-echo imaging (3D MP RAGE). Magnetic Resonance in Medicine, 15(1), 152–157. doi:10.1002/mrm.1910150117

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (pp. 807-814).

Nielsen, M. A. (2015). Neural networks and deep learning (Vol. 25). San Francisco, CA: Determination Press.

Nvidia, C. U. D. A. (2010). Programming guide.

Oliphant, T. E. (2006). A guide to NumPy (Vol. 1, p. 85). USA: Trelgol.

Oliva, A., & Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. International Journal of Computer Vision, 42(3), 145-175. doi:#

Ortiz, A., Munilla, J., Gorriz, J. M., & Ramirez, J. (2016). Ensembles of deep learning architectures for the early diagnosis of the Alzheimer's disease. International Journal of Neural Systems, 26(7), 1650025. doi: 10.1142/s0129065716500258

Ortiz-Suárez, J. M., Ramos-Pollán, R., & Romero, E. (2017, January). Exploring Alzheimer's anatomical patterns through convolutional networks. In 12th International Symposium on Medical Information Processing and Analysis (Vol. 10160, p. 101600Z). International Society for Optics and Photonics.

Pawlowski, M., Meuth, S., & Duning, T. (2017). Cerebrospinal fluid biomarkers in Alzheimer's disease—From brain starch to bench and bedside. Diagnostics, 7(3), 42. doi: 10.3390/diagnostics7030042

Payan, A., & Montana, G. (2015). Predicting Alzheimer's disease: A neuroimaging study with 3D convolutional neural networks. arXiv:1502.02506.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

Penny, W. D., Friston, K. J., Ashburner, J. T., Kiebel, S. J., & Nichols, T. E. (Eds.). (2011). Statistical parametric mapping: The analysis of functional brain images. Elsevier.

Petersen, R. C., Smith, G. E., Waring, S. C., Ivnik, R. J., Tangalos, E. G., & Kokmen, E. (1999). Mild cognitive impairment: clinical characterization and outcome. Archives of Neurology, 56(3), 303-308. doi: 10.1001/archneur.56.3.303

Plant, C., Teipel, S. J., Oswald, A., Böhm, C., Meindl, T., Mourao-Miranda, J., . . . Ewers, M. (2010). Automated detection of brain atrophy patterns based on MRI for the prediction of Alzheimer's disease. Neuroimage, 50(1), 162-174. doi: 10.1016/j.neuroimage.2009.11.046

Quattoni, A., & Torralba, A. (2009, June). Recognizing indoor scenes. In 2009 IEEE Conference on Computer Vision and Pattern Recognition (pp. 413-420). IEEE.

Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. Neural Computation, 29(9), 2352-2449. doi: 10.1162/neco_a_00990

Rehm, K., Schaper, K., Anderson, J., Woods, R., Stoltzner, S., & Rottenberg, D. (2004). Putting our heads together: a consensus approach to brain/non-brain segmentation in T1-weighted MR volumes. NeuroImage, 22(3), 1262-1270. doi: 10.1016/j.neuroimage.2004.03.011

Robbins, H., & Monro, S. (1951). A stochastic approximation method. The annals of mathematical statistics, 400-407.

Rosenblatt, F. (1961). Principles of neurodynamics: Perceptrons and the theory of brain mechanisms (No. VG-1196-G-8). Buffalo, NY: Cornell Aeronautical Lab Inc.

Rottmann, A., Mozos, Ó. M., Stachniss, C., & Burgard, W. (2005, July). Semantic place classification of indoor environments with mobile robots using boosting. In AAAI (Vol. 5, pp. 1306-1311).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation (No. ICS-8506). La Jolla: Institute for Cognitive Science, University of California, San Diego.

Sales, A., Mayordomo, T., Redondo, R., Torres, M., & Bendicho, J. (2016). Psychological and Behavioral Disorders in Dementia. International Journal of Emergency Mental Health and Human Resilience, 18(2).

Sarraf, S., & Tofighi, G. (2016). Classification of Alzheimer's disease using fMRI data and deep learning convolutional neural networks. arXiv:1603.08631.

Serrano-Pozo, A., Frosch, M. P., Masliah, E., & Hyman, B. T. (2011). Neuropathological alterations in Alzheimer disease. Cold Spring Harbor Perspectives in Medicine, 1(1), a006189. doi:10.1101/cshperspect.a006189

Shi, B., Chen, Y., Zhang, P., Smith, C. D., Liu, J., & Alzheimer's Disease Neuroimaging Initiative. (2017). Nonlinear feature transformation and deep fusion for Alzheimer's Disease staging analysis. *Pattern recognition*, *63*, 487-498. doi: 10.1016/j.patcog.2016.09.032

Shi, J., Zheng, X., Li, Y., Zhang, Q., & Ying, S. (2017). Multimodal neuroimaging feature learning with multimodal stacked deep polynomial networks for diagnosis of Alzheimer's disease. IEEE Journal of Biomedical and Health Informatics, 22(1), 173-183. doi: doi: 10.1109/JBHI.2017.2655720

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556.

Song, M., Zhou, Y., Li, J., Liu, Y., Tian, L., Yu, C., & Jiang, T. (2008). Brain spontaneous functional connectivity and intelligence. Neuroimage, 41(3), 1168-1176. doi: 10.1016/j.neuroimage.2008.02.036

Spitzer, R. L., & Williams, J. B. (1980). Diagnostic and statistical manual of mental disorders. In American Psychiatric Association.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

Suk, H. I., Lee, S. W., Shen, D., & Alzheimer's Disease Neuroimaging Initiative. (2014). Hierarchical feature representation and multimodal fusion with deep learning for AD/MCI diagnosis. NeuroImage, 101, 569-582. doi: 10.1016/j.neuroimage.2014.06.077

Suk, H. I., Lee, S. W., Shen, D., & Alzheimer's Disease Neuroimaging Initiative. (2017). Deep ensemble learning of sparse regression models for brain disease diagnosis. Medical Image Analysis, 37, 101-113. doi: 10.1016/j.media.2017.01.008

Suk, H. I., Shen, D., & Alzheimer's Disease Neuroimaging Initiative. (2015). Deep learning in diagnosis of brain disorders. In Recent Progress in Brain and Cognitive Engineering (pp. 203-213). Dordrecht, Netherlands: Springer.

Suk, H. I., Wee, C. Y., Lee, S. W., & Shen, D. (2016). State-space model with deep learning for functional dynamics estimation in resting-state fMRI. NeuroImage, 129, 292-307. doi:#

Supratak, A., Li, L., & Guo, Y. (2014, August). Feature extraction with stacked autoencoders for epileptic seizure detection. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (pp. 4184-4187). IEEE. doi: 10.1109/EMBC.2014.6944546

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Tzourio-Mazoyer, N., Landeau, B., Papathanassiou, D., Crivello, F., Etard, O., Delcroix, N., . . . Joliot, M. (2002). Automated anatomical labeling of activations in SPM using a macroscopic anatomical parcellation of the MNI MRI single-subject brain. Neuroimage, 15(1), 273-289. doi: 10.1006/nimg.2001.0978

Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. International Journal of Computer Vision, 104(2), 154-171. doi: 10.1007/s11263-013-0620-5

Uršič, P., Leonardis, A., & Kristan, M. (2016, May). Part-based room categorization for household service robots. In 2016 IEEE International Conference on Robotics and Automation (pp. 2287-2294). IEEE.

Van Den Heuvel, M. P., Stam, C. J., Kahn, R. S., & Pol, H. E. H. (2009). Efficiency of functional brain networks and intellectual performance. Journal of Neuroscience, 29(23), 7619-7624. doi: 10.1523/jneurosci.1443-09.2009

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11, 3371-3408.

Vuong, N. K., Chan, S., Lau, C. T., & Lau, K. M. (2011, May). Feasibility study of a real-time wandering detection algorithm for dementia patients. In Proceedings of the First ACM MobiHoc Workshop on Pervasive Wireless Healthcare (p. 11). ACM

Wang, Y., Wu, K., & Ni, L. M. (2017). Wifall: Device-free fall detection by wireless networks. IEEE Transactions on Mobile Computing, 16(2), 581-594. doi: 10.1109/tmc.2016.2557792

Wei, L., Duan, X., Yang, Y., Liao, W., Gao, Q., Ding, J. R., . . . Chen, H. (2011). The synchronization of spontaneous BOLD activity predicts extraversion and neuroticism. Brain Research, 1419, 68-75. doi: 10.1016/j.brainres.2011.08.060

Wenk, G. L. (2003). Neuropathologic changes in Alzheimer's disease. Journal of Clinical Psychiatry, 64, 7-10.

World Health Organization. (2012). Dementia: a public health priority. Location: World Health Organization.

World Health Organization. (2018). Dementia fact sheet. 2017.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1492-1500).

Xing, C., Ma, L., & Yang, X. (2016). Stacked denoise autoencoder based feature extraction and classification for hyperspectral images. Journal of Sensors, 2016. doi:10.1155/2016/3632943

Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., & Xiao, J. (2015). Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. arXiv:1506.03365.

Zhang, D., Shen, D., & Alzheimer's Disease Neuroimaging Initiative. (2012). Multi-modal multi-task learning for joint prediction of multiple regression and classification variables in Alzheimer's disease. NeuroImage, 59(2), 895-907.

Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., & Oliva, A. (2014). Learning deep features for scene recognition using places database. In Advances in neural information processing systems (pp. 487-495).

Zhu, H., Weibel, J. B., & Lu, S. (2016). Discriminative multi-modal feature fusion for rgbd indoor scene recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2969-2976).