

**FACULTY OF ENGINEERING,
COMPUTING AND THE ENVIRONMENT**

School of Computer Science and Mathematics

**MSc DEGREE
IN
DATA SCIENCE**

Name : Jeevan Mohan Pawar
ID Number : K2242210
Project Title : Assessing Knee Ligament Tears Using
Deep Learning
Date : 18th September 2023
Supervisor : Dr. Jad Abbass

Kingston University London

WARRANTY STATEMENT

This is a student project. Therefore, neither the student nor Kingston University makes any warranty, express or implied, as to the accuracy of the data or conclusion of the work performed in the project and will not be held responsible for any consequences arising out of any inaccuracies or omissions therein.

Table of Contents

LIST OF FIGURES.....	3
LIST OF TABLES.....	5
GLOSSARY.....	6
ABSTRACT.....	7
ACKNOWLEDGEMENT.....	8
CHAPTER 1 : INTRODUCTION.....	9
1.1 BACKGROUND.....	9
1.2 RESEARCH IDEA AND MOTIVATION.....	10
1.3 LIGAMENT INJURIES IN KNEE JOINT.....	11
1.4 AIMS AND OBJECTIVES.....	12
1.5 ETHICS AND LEGAL RELEVANCE.....	13
1.6 RESEARCH METHOD AND WORK PLAN.....	13
1.7 THESIS OUTLINE.....	15
CHAPTER 2 : LITERATURE REVIEW.....	16
2.1 KNEE LIGAMENT INJURIES AND DIAGNOSIS.....	16
2.2 KNEE MRIS AND MEDICAL IMAGING.....	17
2.3 DEEP LEARNING IN MEDICAL IMAGING.....	18
2.4 KNEE LIGAMENT INJURIES AND DEEP LEARNING.....	19
2.5 RESEARCH GAP AND CHALLENGES.....	21
CHAPTER 3 : CONTRIBUTION.....	23
3.1 OVERVIEW.....	23
3.2 TECHNOLOGIES AND RESOURCES.....	24
3.3 DATASETS.....	25
3.4 EXPLORATORY DATA ANALYSIS.....	27
3.5 DATA PREPROCESSING.....	33
3.6 CONVOLUTIONAL NEURAL NETWORKS.....	38
3.7 TRAINING PRE-REQUISITES.....	40
3.8 NEED FOR GPUs.....	41
3.9 BATCH PROCESSING.....	43
3.10 EVALUATION METRICS.....	44
3.11 MODELS.....	46
3.12 GRAPHICAL USER INTERFACE.....	64
CHAPTER 4 : CONCLUSION.....	66
4.1 SUMMARY OF WORK.....	66
4.2 CHALLENGES.....	66
4.3 CRITICAL REVIEW AND REFLECTION ON WORK.....	67
4.4 FUTURE SCOPE.....	68
REFERENCES.....	69
APPENDIX.....	72

LIST OF FIGURES

FIGURE 1: DATA SCIENCE (CHAMCHOUN, 2022).....	9
FIGURE 2 : ANATOMY OF THE KNEE (MULCAHEY, 2022).....	11
FIGURE 3 : GANTT CHART (CROFT, 2012).....	14
FIGURE 4 : METHODOLOGY.....	23
FIGURE 5 : MRNET DATASET SAMPLE (BIEN, ET AL., 2018).....	25
FIGURE 6 : KNEEMRI DATASET SAMPLES (ŠTAJDUHAR, ET AL., 2017).....	26
FIGURE 7 : SOURCE FILES IN MRNET DATASET.....	28
FIGURE 8 : EXPLORING MRNET SAMPLE MRI.....	28
FIGURE 9 : ANALYSIS OF SLICES IN MRNET TRAIN MRI SCANS.....	29
FIGURE 10 : ANALYSIS OF SLICES IN MRNET VALID MRI SCANS.....	29
FIGURE 11 : NUMBER OF SAMPLES IN MRNET DATASET (TRAIN + VALID).....	30
FIGURE 12 : ACL CLASS IMBALANCE IN MRNET DATASET (TRAIN + VALID).....	30
FIGURE 13 : SOURCE FILES IN KNEEMRI DATASET.....	31
FIGURE 14 : EXPLORING KNEEMRI SAMPLE MRI.....	31
FIGURE 15 : ANALYSIS OF SLICES IN MRNET VALID MRI SCANS.....	32
FIGURE 16 : NUMBER OF SAMPLES IN KNEEMRI DATASET.....	32
FIGURE 17 : ACL CLASS IMBALANCE IN KNEEMRI DATASET.....	33
FIGURE 18 : ORIGINAL MRI VOLUME SHAPE AND RESIZED VOLUME SHAPE.....	34
FIGURE 19 : DENOISING OF MRI SAMPLE.....	34
FIGURE 20 : TIME COMPARISON OF BIAS FIELD CORRECTIONS.....	34
FIGURE 21 : BIAS FIELD CORRECTION OF MRI SAMPLE.....	35
FIGURE 22 : RANDOM ROTATION DATA AUGMENTATION OF MRI SAMPLE.....	36
FIGURE 23 : RANDOM HORIZONTAL FLIP DATA AUGMENTATION OF MRI SAMPLE.....	36
FIGURE 24 : MRNET DATASET AFTER DATA AUGMENTATION.....	37
FIGURE 25 : MRNET WITH REDUCED ACL CLASS IMBALANCE.....	37
FIGURE 26 : KNEEMRI DATASET AFTER DATA AUGMENTATION.....	37
FIGURE 27 : KNEEMRI WITH REDUCED CLASS IMBALANCE.....	38
FIGURE 28 : TYPICAL CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE (HOESER & KUENZER, 2020).....	38
FIGURE 29 : COMPARISON OF 2D AND 3D CONVOLUTIONS (TRAN, ET AL., 2015).....	39
FIGURE 30 : TRAINING ON CPU.....	42
FIGURE 31 : TRAINING ON GPU.....	42
FIGURE 32 : USAGE OF GPU (NVIDIA RTX A5000).....	43
FIGURE 33 : EXAMPLE OF A CONFUSION MATRIX.....	44
FIGURE 34 : MODEL1 FOR MRNET AND KNEEMRI.....	47
FIGURE 35 : MODEL1 TRAINING FOR MRNET AND KNEEMRI.....	47
FIGURE 36 : MODEL1 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	48
FIGURE 37 : MODEL2 FOR MRNET AND KNEEMRI.....	48
FIGURE 38 : MODEL2 TRAINING FOR MRNET AND KNEEMRI.....	49
FIGURE 39 : MODEL2 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	49
FIGURE 40 : MODEL3 FOR MRNET AND KNEEMRI.....	50
FIGURE 41 : MODEL3 TRAINING FOR MRNET AND KNEEMRI.....	51
FIGURE 42 : MODEL3 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	51
FIGURE 43 : MODEL4 FOR MRNET AND KNEEMRI.....	52
FIGURE 44 : MODEL4 TRAINING FOR MRNET AND KNEEMRI.....	52
FIGURE 45 : MODEL4 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	53
FIGURE 46 : MODEL5 FOR MRNET AND KNEEMRI.....	53
FIGURE 47 : MODEL5 TRAINING FOR MRNET AND KNEEMRI.....	54
FIGURE 48 : MODEL5 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	54
FIGURE 49 : MODEL6 FOR MRNET AND KNEEMRI.....	55
FIGURE 50 : MODEL6 TRAINING FOR MRNET AND KNEEMRI.....	56
FIGURE 51 : MODEL6 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	56

FIGURE 52 : MODEL_TF_3_VGG16 FOR MRNET AND KNEEMRI.....	57
FIGURE 53 : MODEL_TF_3_VGG16 TRAINING FOR MRNET AND KNEEMRI.....	57
FIGURE 54 : MODEL_TF_3_VGG16 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	58
FIGURE 55 : MODEL_TF_3_XCEPTION FOR MRNET AND KNEEMRI.....	58
FIGURE 56 : MODEL_TF_3_XCEPTION TRAINING FOR MRNET AND KNEEMRI.....	59
FIGURE 57 : MODEL_TF_3_XCEPTION CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	59
FIGURE 58 : MODEL_TF_3_ResNet50 FOR MRNET AND KNEEMRI.....	60
FIGURE 59 : MODEL_TF_3_ResNet50 TRAINING FOR MRNET AND KNEEMRI.....	60
FIGURE 60 : MODEL_TF_3_ResNet50 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	60
FIGURE 61 : MODEL_TF_5_VGG16 FOR MRNET AND KNEEMRI.....	61
FIGURE 62 : MODEL_TF_5_VGG16 TRAINING FOR MRNET AND KNEEMRI.....	61
FIGURE 63 : MODEL_TF_5_VGG16 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	62
FIGURE 64 : MODEL_TF_5_ResNet50 FOR MRNET AND KNEEMRI.....	62
FIGURE 65 : MODEL_TF_5_ResNet50 TRAINING FOR MRNET AND KNEEMRI.....	62
FIGURE 66 : MODEL_TF_5_ResNet50 CONFUSION MATRICES FOR MRNET AND KNEEMRI.....	63
FIGURE 67 : USER INTERFACE WITH HEALTHY ACL PREDICTION FROM AN UPLOADED MRI.....	64
FIGURE 68 : USER INTERFACE WITH PARTIAL ACL TEAR PREDICTION FROM AN UPLOADED MRI.....	65
FIGURE 69 : USER INTERFACE WITH COMPLETE ACL TEAR PREDICTION FROM AN UPLOADED MRI.....	65

LIST OF TABLES

TABLE 1 : OVERVIEW OF MRNET DATASET (BIEN, ET AL., 2018).....	25
TABLE 2 : OVERVIEW OF KNEEMRI DATASET (ŠTAJDUHAR, ET AL., 2017).....	27
TABLE 3 : SAMPLES IN MRNET DATASET.....	30
TABLE 4 : CO-OCCURRENCE OF LABELS IN MRNET SAMPLES.....	31
TABLE 5 : SAMPLES IN KNEEMRI DATASET.....	33
TABLE 6 : COMPARING EVALUATION METRICS OF THE MODELS TRAINED ON MRNET DATASET.....	63
TABLE 7 : COMPARING EVALUATION METRICS OF THE MODELS TRAINED ON KNEEMRI DATASET.....	63

GLOSSARY

2D	Two Dimensional
3D	Three Dimensional
ACL	Anterior Cruciate Ligament
ACLR	Anterior Cruciate Ligament Reconstruction
API	Application Programming Interface
CAM	Class Activation Map
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma-separated Values
CT	Computed Tomography
CUDA	Compute Unified Device Architecture
DICOM	Digital Imaging and Communications in Medicine
GB	Gigabyte
GE	General Electric
GPU	Graphical Processing Unit
GUI	Graphical User Interface
JPEG	Joint Photographic Expert Group
LCL	Lateral Collateral Ligament
MCL	Medial Collateral Ligament
MRI	Magnetic Resonance Imaging
PCL	Posterior Cruciate Ligament
PD	Proton Density
PNG	Portable Network Graphics
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RGB	Red, Green and Blue Channels
ROI	Region of Interest
RTX	Ray Tracing Texel eXtreme
VGG	Visual Geometry Group
Xception	Xtreme Inception
HOG	Histogram of Gradients
SVM	Support Vector Machine

DenseNet	Densely Connected Convolutional Network
YOLO	You Only Look Once

ABSTRACT

Abstract - Knee ligament injuries, pervasive and incapacitating, wield the potential to substantially degrade an individual's quality of life. Prompt diagnosis and treatment is crucial to prevent long-term disability. Magnetic resonance imaging is commonly used for diagnosing knee ligament injuries unless there are contraindications for its use. Although MRI is non-invasive and safe for patients, it requires expert interpretation and can be time-consuming. In recent years, there has been significant progress in utilizing deep learning techniques to analyse medical images and provide automated diagnoses for knee ligament injuries. This study aims to explore the application of deep convolutional neural network architectures in accurately predicting ligament tears based on MRI scans. To support our findings, we have incorporated two widely available knee MRI datasets, MRNet and KneeMRI, which serve as valuable resources for this project. The utilization of deep learning models in knee joint MRI image analysis has significant implications. It not only speeds up the generation of MRI reports, but also provides diagnostic assistance to clinicians. Furthermore, this research lays the foundation for similar diagnostic models that could be used for addressing medical conditions in other parts of the body.

Keywords— *Deep Learning, Convolutional Neural Network (CNN), Ligament Tear, Magnetic Resonance Imaging (MRI)*

ACKNOWLEDGEMENT

I am glad for the opportunity to recognise the major contributions and unwavering support that have helped me complete this dissertation successfully. This academic endeavour has been a transforming educational experience, and I want to express my heartfelt gratitude to everyone who helped make it a reality.

I am extremely grateful to Dr. Jad Abbass for his unwavering support, direction, and knowledge during this endeavour. His outstanding suggestions and persistent encouragement have significantly influenced the course of my research.

I extend my sincere appreciation to the esteemed faculty and dedicated staff members of Kingston University's School of Computer Science and Mathematics for fostering an intellectually rigorous atmosphere and empowering me with the essential tools needed for this academic endeavour.

I am deeply appreciative of the support, patience, and understanding provided by my family and friends during this undertaking. Their unwavering faith in me has been a constant source of motivation and invaluable aid.

Finally, I'd want to recognise the invaluable contributions made by students, specialists, and professionals in this field of research. Their trailblazing work has had a substantial impact on the advancement of my research.

As I bring this chapter of my academic journey to a close, I am grateful for the invaluable support and contributions from everyone mentioned earlier. Your involvement has played a crucial role in helping me reach this significant milestone in my life.

Chapter 1 : INTRODUCTION

1.1 Background

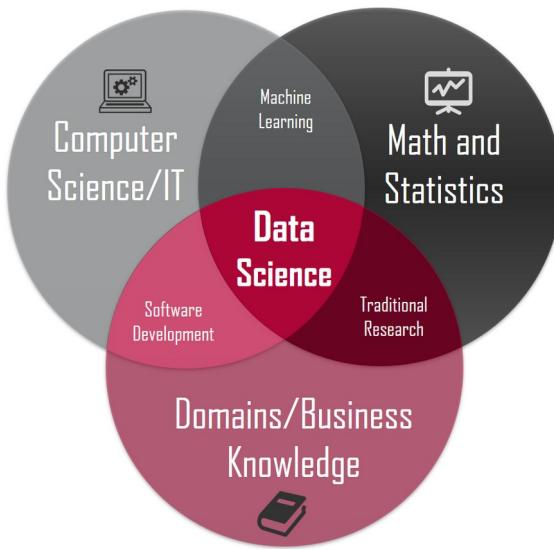


Figure 1: Data Science (Chamchoun, 2022)

Data Science is a term that originated with simple analysis and interpretation of data through statistical methods, and eventually grew into a multidisciplinary subject as it evolved through technical advancements and strong potential applications across all sectors. It harnesses the power of computer science, techniques from statistics, and domain expertise to derive insights and understanding from extensive and intricate datasets.

It encompasses the application of diverse approaches, algorithms, and technologies for examining and interpreting data with the goal of facilitating informed decision-making processes and constructing predictive models.

It has evolved from its origins in statistical analysis to become a multifaceted discipline that is deeply integrated into various aspects of our daily lives. Its vast range of applications in sectors such as healthcare, finance, business, logistics, environmental science, and others consistently offer ground-breaking solutions, data-informed decision-making processes, and even surpass human capabilities with the aid of artificial intelligence.

Biomedical science has made huge leaps through the application of advanced-level machine learning and deep-learning data science projects. Some of the top contributions include drug discovery, diagnostics, virtual assistants, predictive analytics as well as medical image analysis. During the COVID-19 outbreak, data science proved to be an instrumental factor, organizations like the World Health Organization used it for performing clinical trials and identifying effective treatments. Looking forward, ethical implications, interdisciplinary collaboration, and continuous technological advancements are key factors that will shape the future landscape of data science.

1.2 Research Idea and Motivation

Deep learning has provided ways for the analysis of medical images, allowing for automated assessment of complex imaging modalities like X-rays, MRIs, and CT scans. These models can accurately detect and

classify various conditions such as tumours, fractures, and other abnormalities. It provides healthcare professionals with more precise diagnoses promptly and, in turn significantly enhances patient care by facilitating earlier detection, improved treatment planning, and alleviating the workload of medical experts.

Ligaments are a strong and flexible fibrous band of tissue that connects bones to other bones in joints such as the knees, ankles, and wrists, providing the needed stability and strength for the crucial movement of the human body. Ligaments consist of collagen fibres, which provide them with their robustness, enabling them to endure the tension and the stresses exerted on joints during physical activity. It is responsible for maintaining the alignment of the bones, preventing dislocations in joints, and enabling controlled movement within a safe range.

Ligament injuries can occur if these are stretched excessively beyond their capacity, this usually happens due to sudden impacts, twisting motions, changes of direction, or unnatural landings. Knee ligament injuries are commonly observed in individuals who engage in physical activities, particularly athletes and sportsmen. Additionally, individuals can also suffer such injuries in car accidents, trips, falls, overexertion of joints, improper technique during physical activities, or weak joints from old age. The severity of these injuries varies from mild sprains to complete tears of the ligaments, which can significantly impact an individual's ability to partake in both recreational activities and routine tasks.

Accurate interpretation of knee MRI scans is a time-consuming and subjective process, even for experienced medical experts or radiologists. It requires multiple reviews and analysis of the complex anatomy of the joint, considering any prior history of injuries or medical conditions. Only after this comprehensive assessment can an appropriate treatment plan be conveyed to the patient.

Deep learning models have emerged as a ground-breaking method for addressing knee ligament injuries and possess a lot of potential in medical imaging. By leveraging sophisticated algorithms, these models can analyse extensive collections of medical images, including MRI scans of the knee joint. Through their ability to recognize complex patterns and intricate features across diverse datasets, deep learning models can effectively identify subtle irregularities, tears, or structural modifications within ligaments.

Developing accurate deep-learning models for diagnosing knee ligament injuries using MRI scans is a multifaceted endeavour that presents various challenges. A primary obstacle lies in acquiring a robust dataset of high-quality MRI images with precise annotations, encompassing different types and levels of severity for ligament injuries. Additionally, the intricate nature of knee anatomy and the variability in injury manifestations underscore the need for comprehensive and meticulously annotated data.

Deep learning has been applied to MRI image processing and analysis which has proven to be effective in various applications, such as image segmentation, registration, injury diagnosis, denoising, and scan quality enhancement. However, the success of MRI-based deep learning applications has been hindered by two main challenges: a small dataset size and class imbalance. In order to address these issues, we will explore strategies like data augmentation and combining datasets to increase the samples while also taking into account the class imbalance during model training.

When working with medical images, it is important to consider their three-dimensional nature. Traditional computer vision models used for tasks like object recognition are typically designed for 2D images. However, in the case of medical imaging such as MRI scans, the volumetric information can

provide valuable insights. In these situations, a 3D convolutional neural network can be advantageous as it can leverage both spatial features and anatomical structure across the entire volume to capture more comprehensive information. It should be noted that using a 3D CNN comes with certain drawbacks including increased memory requirements, computational power demands, and longer training times.

Currently, there is limited existing research and literature on the utilization of deep learning for assessing knee ligament injuries. The basis for this dissertation arises from the convergence of medical science, technological advancement, and patient well-being. Additionally, the current difficulties serve as a strong incentive to explore the creation of models capable of evaluating knee ligament injuries. Knee ligament injuries are a prominent health issue that impacts individuals across different age groups and lifestyles. These injuries can result in pain, limited mobility, and compromised quality of life. The current approaches to evaluating these injuries typically involve a combination of clinical examination, interpretation of imaging tests, and subjective judgment.

1.3 Ligament Injuries in Knee Joint

The knee joint is an essential hinge joint that connects the femur (thigh bone) to the tibia (shin bone). It also houses the patella, or kneecap, which serves as a protective covering at the front of the joint. Smooth articular cartilage covers the ends of both bones, allowing for fluid movement and reducing friction during motion. Additionally, C-shaped pieces of cartilage called menisci function as shock absorbers and contribute to stability within the joint. Working in harmony, ligaments, tendons, and muscles support overall stability while enabling a wide range of motion and providing strength to the knee joint. The biological anatomy of the knee is presented in Fig 2. from different views for a better understanding below:

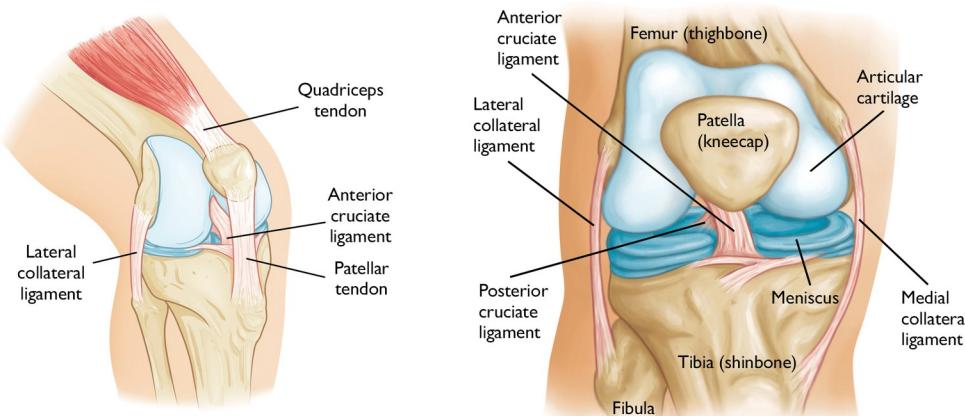


Figure 2 : Anatomy of the knee (Mulcahey, 2022)

The knee joints are reinforced by four primary ligaments, the anterior cruciate ligament (ACL), posterior cruciate ligament (PCL), medial collateral ligament (MCL), and lateral collateral ligament (LCL), which play important roles in providing stability and strength to the joint.

Both the ACL and PCL pass through the inside of the knee joint, forming a crossing pattern that resembles an "X." The ACL is positioned at the front, preventing excessive forward movement of the tibia.

Conversely, located at the rear, the PCL prevents excessive backward movement of the tibia. These ligaments have a crucial role in maintaining stability during rotational motions and abrupt shifts in direction.

The MCL and LCL are crucial structures located on the sides of the knee joint. These ligaments play a vital role in providing stability by limiting excessive inward or outward bending of the knee. Specifically, the MCL is situated on the inner side of the knee and is commonly injured during contact sports or activities that involve an impact on the outside area of the knee. In contrast, injuries to LCL are relatively less common since it is positioned on the outer side of the knee.

Ligament injuries in the knee joint are classified according to the severity of the injury and the extent of ligament damage:

- Grade 1 (Mild): Ligament injuries in this category are often referred to as "sprains." It typically involves slight stretching or microscopic tears of the ligament fibres, resulting in minimal instability and potential mild discomfort, but overall function is not significantly affected.
- Grade 2 (Moderate): Injuries in this classification occur when there is a partial tearing of the ligament fibres. The affected joint may experience instability, moderate pain, swelling, and limitations in range of motion. These injuries are indicative of a more substantial compromise to the integrity of the ligaments, joints may buckle with movements and cause further damage.
- Grade 3 (Severe): Complete tears of the ligament fall into this category. This leads to instability in the joint, along with symptoms such as pain, swelling, and restricted range of motion. There may be instances where the joint feels loose or gives way easily.

Magnetic Resonance Imaging (MRI) is a valuable modality for precisely assessing knee ligament injuries. In instances where there is suspicion of ligament damage, an MRI scan is commonly employed to visualize the internal structures of the knee joint. Through detailed cross-sectional imaging, MRI offers substantial insights into soft tissues such as ligaments, tendons, cartilage, and more. The non-invasive nature and high level of detail make MRI particularly effective in diagnosing ligament injuries. It provides multidimensional views of the knee joint that enable healthcare providers to accurately determine both the location and extent of the ligament damage. This information plays a critical role in guiding appropriate treatment interventions ranging from conservative methods like physical therapy to more invasive approaches such as surgery.

1.4 Aims and Objectives

The purpose of this project is to develop a deep learning model that can analyse knee MRI scans and accurately assess knee ligament injuries. This has several potential benefits:

- a) It can expedite the report generation process during emergency situations.
- b) It may assist healthcare professionals in efficiently diagnosing these injuries.
- c) Patients would have the option to seek a second opinion or gain additional understanding about their injury while waiting for test results and medical appointments.

To outline the project objectives using the SMART framework for goals:

- a) Specific: This study aims to develop deep learning models by training them on datasets of knee MRIs to accurately assess ligament damage following a knee injury.
- b) Measurable: Our goal is to achieve a high level of accuracy, with at least 90%, in accurately determining the grade of ligament injury through our developed model.
- c) Achievable: Through an extensive review and analysis of existing literature, we aim to enhance current deep learning techniques, methodologies, architectures, and available datasets related to knee MRIs. This improvement will contribute towards training a more effective model for ligament injury assessment.
- d) Relevant: The utilization of deep learning models in detecting knee ligament injuries holds significant relevance as it enhances diagnostic precision while potentially reducing reliance on invasive procedures such as arthroscopy. Moreover, this approach offers faster MRI report generation times compared to traditional methods.
- e) Time-bound: Within the designated timeframe, my objective is to create and test a functional prototype for this project. This will involve pre-processing the dataset, refining its quality, constructing different deep learning model frameworks, training these models extensively, and finally verifying their effectiveness in achieving the desired objectives.

1.5 Ethics and Legal Relevance

The ethical and legal implications surrounding the use of deep learning to predict knee ligament tears from MRI scans are important. These considerations are vital for ensuring that the project is conducted responsibly and within the confines of the law, particularly when working with sensitive medical data and potential impacts on patient care:

- The datasets to be utilized in this venture are freely accessible for non-commercial research purposes only, specifically for educational projects. It is strictly prohibited to distribute these datasets to any external party without obtaining explicit permission from the original providers. Moreover, it is essential to appropriately reference or cite any dataset used in the project.
- The dataset consists of MRI scans from individuals whose identities have been anonymized. These data subjects will not be identified, and the scans will only be used for training the models.
- Each user utilizing our developed project will need to upload their MRI scan for analysis using the trained model. Rest assured that all data will be handled securely and will not be stored or saved anywhere beyond the immediate processing needs.
- It is imperative to mitigate any potential adverse effects that may arise from the utilization of the deep learning model, including misdiagnoses resulting in inappropriate treatment. Given that this initiative serves an educational purpose, it is essential to provide disclaimers to users emphasizing that the system's results should not replace medical advice or treatment from a doctor.

During the development of this project, there are no significant legal or ethical concerns that need to be addressed. It is crucial to acknowledge that all work and progress made under this project is strictly for educational purposes. In essence, adhering to responsible research practices and complying with established

laws and regulations ensures both the ethical integrity and legal significance of this dissertation project. By considering these factors, the project can have a beneficial impact on medical diagnostics while prioritizing patient rights and ensuring the well-being of healthcare stakeholders.

1.6 Research Method and Work Plan

The methods employed to carry out research studies are commonly known as research methodologies. The specific methodology and plan for the development of this project are as follows:

a) Qualitative Research

- Qualitative research typically entails the examination of text, images, and observations in order to gain an understanding of concepts, thoughts, or experiences that are not easily quantifiable.
- Reviewing scholarly literature to explore concepts and theories is a widely used qualitative method. This approach entails examining academic sources to gain a comprehensive understanding and identify relevant methodologies or solutions.
- This study employs deep learning techniques to improve the evaluation of knee ligament injuries based on MRI scans. Convolutional neural network architectures will be created and trained using publicly accessible knee MRI datasets. The CNN model will learn patterns from the images to effectively predict the presence and severity of ligament tears.
- The procedure consists of pre-processing the data, wherein MRI images are prepared to be input into the convolutional neural network. The architecture of the model will be devised, considering factors such as depth, layers, and filters to capture pertinent features from the images. To ensure accurate evaluation and generalization of the model, datasets will be divided into training, validation, and testing sets. Labelled images will be utilized for training purposes while assessing performance metrics like accuracy, precision, and recall.

b) Agile Project Development

- Agile project development is a project management methodology that emphasizes flexibility and adaptability through iterative processes.
- The project is segmented into smaller, more manageable portions known as sprints in accordance with the Agile methodology. Each sprint serves a milestone and specific deliverables.
- Agile development necessitates effective communication, collaboration, and feedback mechanisms that enable prompt adaptation to change and implementation of necessary enhancements throughout the project.
- Regular discussions with supervisors and iterative cycles of development, testing, and feedback are well-suited for the fast progress of a deep learning project.

The project's work plan is visually represented by a Gantt chart, which illustrates the scheduled timelines for each of the project deliverables. Additionally, it outlines an estimated time commitment for each task on a weekly basis. The specific Gantt chart for our research study can be found in Fig 3. below:

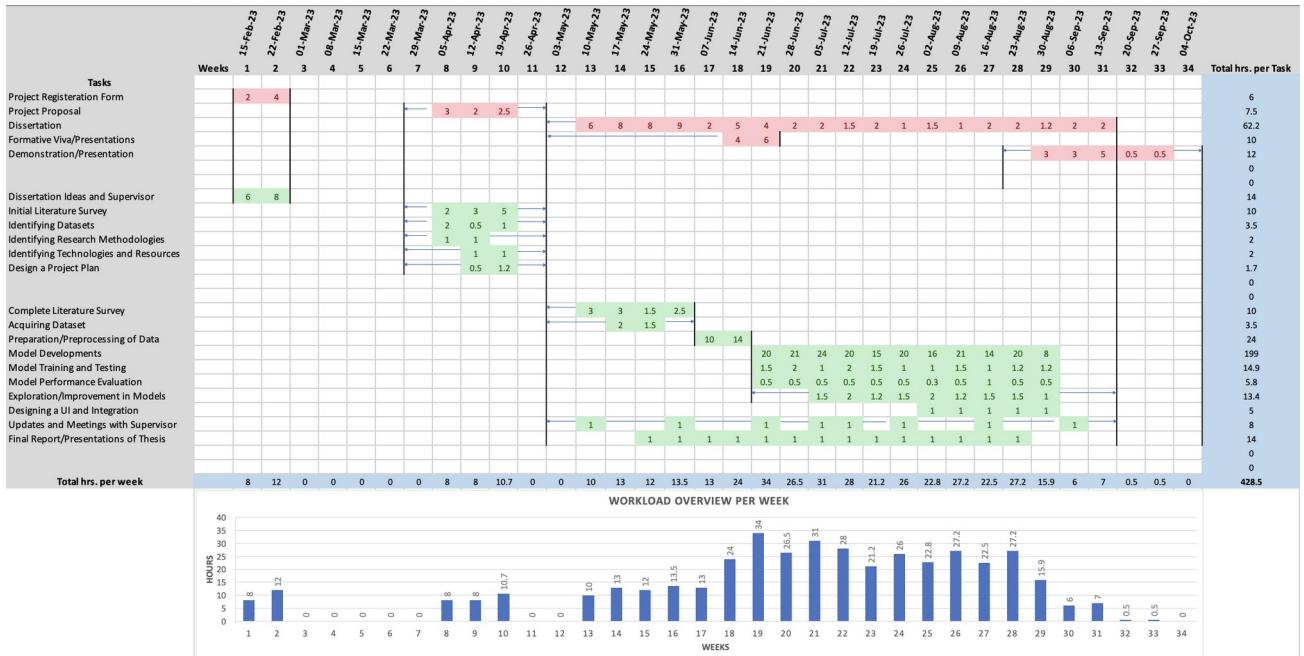


Figure 3 : Gantt Chart (Croft, 2012)

1.7 Thesis Outline

The dissertation is organized into four chapters, each dedicated to exploring various aspects of the research study:

- Chapter 1: Introduction

This chapter explores the field of data science and its wide-ranging applications, with a specific focus on medical imaging in relation to knee ligament examination. It also discusses the research concept and objectives, emphasizing the significance of thorough knee ligament assessment. Additionally, it highlights the unique features of knee ligament injuries and underscores the challenges associated with their evaluation using conventional techniques.

- Chapter 2: Literature Review

This chapter presents a thorough examination of the current body of literature regarding knee ligament assessment, MRI scans, and deep learning techniques. It reviews relevant studies and research papers that have investigated the application of MRI scans and deep learning algorithms in medical imaging mainly based on knee MRIs. The aim is to understand the progress made in this area of research, identify any existing challenges, and propose new ideas for further development.

- Chapter 3: Contribution

This chapter presents the methodology employed in this research study. It outlines the datasets utilized, including a comprehensive analysis of MRI images and labels. The pre-processing and preparation of these datasets are discussed following an initial exploratory data analysis. Subsequently, diverse deep-learning models are trained for the purpose of this study along with their evaluations.

- Chapter 4: Conclusion

This section presents a concise overview of the previous chapter's research. It commences by outlining the significant findings and subsequently evaluates any challenges encountered during the project's progression. A meticulous evaluation of existing literature is succeeded by several suggestions for potential avenues of future investigation.

Chapter 2 : LITERATURE REVIEW

2.1 *Knee Ligament Injuries and Diagnosis*

Knee ligament injuries are a frequent orthopaedic issue requiring prompt diagnosis and treatment to minimize further consequences or complications. Clinical examinations and radiological studies, such as MRI scans, are popularly used in diagnosis. These procedures, however, are subjective and are dependent on the clinician's experience. With the advancement of deep learning techniques, various research has investigated the use of deep learning algorithms for the detection of knee ligament damage.

It has been estimated that approximately 200,000 ACL reconstructions are performed each year, with an incidence rate of about 1 in every 3,000 people annually. The implications of an ACL injury extend beyond immediate consequences and can have long-term effects on a person's quality of life, increasing their risk for osteoarthritis and disability over time. Currently, the recommended approach to treating an ACL tear is through ACLR surgery due to evidence supporting improved stability measures and the desire for patients to resume sports activities while also reducing future knee injuries. (Spindler, 2007)

Another research identified the main sources of ligament injuries in the knees, ACL being the most significant and common injuries involving surgeries. An estimated 65% of the cases involved sports activities as the primary reason behind ligament injuries requiring surgery. (Gianotti, et al., 2009)

Research has shown that while males tend to have a higher overall injury rate in the general population, studies focusing on specific sports reveal that females consistently face a greater risk. This difference can be attributed to factors such as the types of physical activities involved, including cutting and jumping, which are more common among males, as well as participation in contact sports. (The female ACL: Why is it more prone to injury?, 2016)

Medical experts point out that the anatomical differences in males and females could be one of the main reasons for females being in more danger of getting a ligament injury. The female pelvis is wider, which affects the mechanism between the thigh bone, tibia, and femur putting more pressure on soft tissues in the joint. Moreover, women have lesser muscle mass surrounding the knee joints leading to more chances of knee joint instability and higher chances of ligament tears. Hormonal differences are also another reason, testosterone is responsible for increasing muscle density, but these are much lower in a female's body. (Macmillan, 2020)

A study revealed that individuals in their early twenties who engaged in sports and physical activities favoured the use of MRI for diagnosing knee problems due to its greater accuracy compared to relying solely on clinical examination. While arthroscopy is a traditional and invasive methodology, MRI offers a comprehensive evaluation of the issue, aiding in determining whether surgical intervention is necessary for patients. (Bryan, et al., 1998)

Physical examination and medical history of a patient also play important roles in assessing the extent of a ligament injury. Before the advent of advanced medical imaging techniques simple physical tests that involved applying pressure on the knee joint and testing the range of motion, included the Lachman Test or Drawer tests, and pivot-shift tests. (Bronstein & Schaffer, 2017)

A combination of physical examinations, medical history, and medical imaging such as the MRI helps to decide the course of action to treat a patient. A patient's age, history of non-surgical measures, presence of arthritis as well as activity levels are important considerations for treating ACL injuries. (Marx, et al., 2016)

2.2 *Knee MRIs and Medical Imaging*

Medical imaging refers to the technique of creating visual representations of the interior of the human body or other living organisms for clinical analysis and medical diagnosis. These images help healthcare professionals visualize and assess the structure and function of organs, tissues, and other anatomical features. Medical imaging plays a crucial role in the diagnosis, monitoring, and treatment of various medical conditions.

Magnetic resonance imaging has become an invaluable diagnostic tool within the medical imaging field. This technique boasts numerous benefits, such as its ability to provide exceptional contrast for soft tissues, its lack of ionizing radiation exposure, and its capability to visualize structures from various angles. (Farshad-Amacker & Potter, 2013)

Magnetic resonance imaging is considered the preferred method for assessing knee ligament structures and diagnosing knee ligament instability. Through analysing MRI signal characteristics, this study enhances our understanding of the intricate anatomy of the knee joint. It is important to focus on specific knee

ligaments based on the mechanism of injury. As advancements in orthopaedic surgical techniques continue to evolve, it is crucial for practitioners to stay updated regarding graft options, tunnel placement, and graft mechanics to ensure accurate interpretation of MRI images. (Farshad-Amacker & Potter, 2013)

Several studies show that MRI is accurate and needful to effectively assess knee ligament injuries. (Potter, et al., 2002). Sagittal MR images have been commonly used to evaluate ACL injuries, but axial and coronal planes can add some useful information as well. Most ACL tears occur in the middle portion of the ligament, less frequently at the femoral or tibial attachment. (Remer, et al., 1992)

In instances of intricate or multi-ligamentous injuries, it is crucial to thoroughly evaluate MRI scans in order to develop an effective treatment plan. Furthermore, the arthroscopic findings during surgery play a significant role and contribute valuable information to guide the management of these types of injuries. (Stephenson, et al., 2010)

Radiologists and medical experts use MRI medical imaging to assess the ligament injuries sustained to a knee joint with high accuracy. Accurate interpretation of the MRI examination requires a meticulous MRI technique, knowledge of the imaging anatomy, and an understanding of the lesion's appearance. It also assists in identifying injuries sustained to other crucial parts of the joint such as the collateral ligaments, bones, and cartilages. The PCL is twice as strong as the ACL, and it's less frequently torn. Physical examination of the knee joints to assess injuries can be difficult sometimes due to pain, in such cases, MRI scans can be relied on to effectively assess the injury. (Kam, et al., 2010)

MRI medical imaging provides superior soft tissue visualisations and intricate details of the knee joint. Unlike X-rays or CT medical imaging techniques, it does not involve any ionizing radiation and it's much safer. MRI also provides a multi-planar view of the joint from the following three orthogonal planes: axial, coronal, and sagittal. A dedicated extremity coil is essential to obtain high-resolution MRI. T1-weighted MRI highlights structural details, making it suitable for assessing bones, ligaments, and cartilage. It is used in at least one of the planes, while T2-weighted MRI emphasizes fluid content, aiding in the detection of soft tissue abnormalities like ligament and meniscal tears. Fat-suppressed T2-weighted MRIs are most common as they can prevent the hyperintense appearance of fats. Proton Density (PD)-weighted MRI strikes a balance between T1 and T2, providing moderate tissue contrast, and is used to evaluate the menisci. (Kam, et al., 2010)

Research indicates that MRI imaging has demonstrated efficacy in accurately locating and determining the extent of tears in soft tissues, as well as identifying bone injuries and neural damage resulting from multiple ligament injuries or knee dislocations. This finding was confirmed through subsequent assessment of arthroscopic surgeries following MRI scans. (Potter, et al., 2002)

2.3 Deep Learning in Medical Imaging

Deep learning has overtaken machine learning techniques to solve various previously intractable problems. Medical image processing is another field of work where deep learning has shown great potential. Computer vision problems such as the detection of objects, face recognition, human pose estimation, emotion detection, and various other complex tasks which were difficult to conquer earlier have been

revolutionised by deep learning models. These models can automatically identify the required features that help to solve the problem from the dataset, making them much more complex and powerful. (Voulovodimos, et al., 2018)

Another survey highlights the work done in computer vision, speech recognition, and pattern recognition using various deep learning architectures, some of the most used ones are CNNs, autoencoders, deep belief networks, etc. emphasizing the capabilities of deep learning and potential application in medical imaging. (Liu, et al., 2017)

A research study examines the applications of deep learning in medical imaging and explores both current achievements and future possibilities. Deep learning has been employed in two main areas within this field: firstly, addressing research problems related to disease detection, classification, and prediction of different diseases and their associated factors; secondly, utilizing deep learning for image processing tasks such as registration (aligning multiple slices of medical images spatially for comparison), segmentation (identifying regions of interest as targets or backgrounds), and generating higher resolution images. The findings suggest that applying deep learning approaches can greatly benefit medical image processing by leveraging its impressive performance demonstrated in non-medical imaging studies when compared to traditional machine learning methods. (Kim, et al., 2018)

Another study gives an overview and compares how the medical imaging field was impacted by machine learning and what changed after the introduction of deep learning. It highlights and compares traditional models that rely on the features extracted from images rather than the images itself as found in deep learning models, this prevents any information lost by just using the extracted features. However, deep learning models can be computationally heavy and require larger size of datasets for good results. (Suzuki, 2017)

Recent progress in medical image analysis using deep learning is the subject of another scholarly paper. The study highlights the advancements made by convolutional neural network-based techniques in clinical applications. These cutting-edge applications primarily focus on four major human body systems: cardiovascular, nervous, digestive, and skeletal systems. Specifically, these CNN-based approaches are utilized for brain-related issues, heart ailments, liver disorders, and orthopaedic diseases. Image classification, object detection, segmentation, and registration are some of the areas where CNN-based methods have demonstrated remarkable success. Advanced deep learning has greatly contributed to significant achievements in biomedical science through the effective utilization of medical images. (Liu, et al., 2021)

Another research paper also gives an overview of deep learning and its applications in medical imaging. It gives a different perspective of the need for radiologists to be aware of advancements made in this field as major tech companies are also heavily investing in this research area for a promising future. It highlights the non-medical, radiologic as well as automatic labelling and captioning. (Lee, et al., 2017)

In recent years, 3D CNNs have gained popularity with medical imaging due to their ability to process volumetric data such as MRI, this has been fuelled by better processing capabilities in terms of powerful GPUs and widespread application of deep learning to improve as well as assist medical experts. A review study of 3D deep learning for medical images highlights the 3D CNNs and their application in medical imaging, pre-processing pipelines for such networks, and their medical applications. Multi-modal medical images need to be pre-processed for image registration, artefact removal, normalization, and bias field

correction. 3D CNNs have been applied for lesion segmentation, disease classification such as dementia and Alzheimer's disease, localization, or detection of tumours, and even registration. Despite of high computational complexity, 3D CNNs have been fruitful in taking the analysis of medical images further. (Singh, et al., 2020)

2.4 *Knee Ligament Injuries and Deep Learning*

In one of the research papers, a custom dataset made from a total case of 230 with half the cases representing ACL tears and the other half with healthy ACL was used to develop and train deep learning models. A separate bunch of 60 cases was used for testing. The models were trained on different inputs such as the cropped region of interests, single slices, and multiple slices from the volumetric samples. In order to generate more samples from the cases dynamic patch-based algorithm was used and pre-processing was done to resample volumes to 256x256 voxels. The research concluded that using multiple slices or 3D inputs as well as cropping the region of interest improves the model's performance. The best model was trained using five slices from the volumes and dynamic patch-based sampling, it gave a 96% test set accuracy. (Chang, et al., 2019)

Another research done on this subject involved developing a robust model trained on a big dataset. MRI samples of almost 20,000 cases were gathered from 12 different imaging centres for patients 16 years old and above, these scans were done between 2009 and 2020. Total samples were split into train, valid, and test data in the ratio of 7:2:1 respectively. The samples were labelled for an ACL tear or no tear using the Natural Language Processing (NLP) of the reports. A model was trained using the coronal and sagittal planes, first part of the model localised the area of the MRI for detecting tears, resized, and pre-processed to be fed further. The second part of the model classified the MRI with tear or no tear. External validation on datasets from different continents was done by MRNet, USA and KneeMRI, Croatia. Heatmaps of the part of an MRI that was being used to classify a sample were also observed to identify whether a model was truly interpreting the classifications. The model gave classification accuracy of 90% on their dataset, and after retraining on the external datasets it gave accuracies of 87% approximately. (Tran, et al., 2022)

One of the other studies carried out on this matter involved experimenting with a 2D CNN by incorporating transfer learning and a 3D CNN that were trained on custom datasets obtained from three prior research studies, these were labelled by expert radiologists with a minimum of five years of experience. A total of 1273 MRI samples with sagittal plane sequences were in this dataset. A hierarchical approach was adopted to classify the lesion severity as a complete tear, partial tear, or no tear. The framework consisted of a segmentation module that identified 11 anatomical areas of the scan and localized the ACL in the slices. This was further fed into convolutional networks, the 3D CNN and 2D CNN with transfer learning had an accuracy of 89% and 92% respectively. Despite 3D CNN's capability to gauge volumetric data better, the 2D CNN outperformed using the concept of transfer learning as it was trained on ImageNet (Stanford Vision Lab, 2020), a dataset of almost 14 million images. (Namiri, et al., 2020)

Another study researched the diagnostic performance of a deep learning model and radiologists in assessing knee ligament tears. The proposed work collected sagittal MRI samples of patients with an arthroscopic confirmation of ACL tear, in total 175 samples with tears and 175 without tears were part of

the dataset. The proposed work consisted of three different CNNs. The first CNN was inspired by Lenet-5 and it was responsible for identifying slices of MRI containing information about the ACL, the second CNN adapted from YOLO localised the parts of slices with ACL, and the third CNN based on DenseNet identified if the tear is present or not. The research didn't find major statistical differences in the radiologist and deep learning model's analysis. The model achieved an AUC of 0.98 for detecting whether a tear is present or not. (Liu, et al., 2019)

One of the major research projects done on this topic was at Stanford University. The dataset used was generated from MRI exams conducted at the Stanford University Medical Centre, it consisted of a total of 1,370 samples consisting of views from three different planes. MSK radiologists assessed and labelled the samples for three different things with binary labels for abnormalities, ACL tears, and meniscal tears. They developed a convolutional neural network called MRNet, a combination of predictions made by multiple CNNs using a linear regression model for three labels. These CNNs were trained on the samples by utilising Alexnet as a feature extractor, a model was trained for each of the three planes and tasks. Overall, for each task (abnormality, ACL tear, meniscal tear) and series type (sagittal, coronal, axial), the training yielded nine distinct MRNets, whose probabilities were later combined using a logistic regression classifier. They also reviewed the class activation maps (CAMs) of these models to identify what the model picked up from the MRI scan. The model achieves ROC AUC scores of 0.937, 0.965, and 0.847 respectively to detect abnormalities, ACL tears, and meniscus tears. The radiologists had a higher accuracy in detecting tears in MRI at 90% as compared to the MRNet model at 85%. The research also concluded the assistance of a deep learning model improved the radiologists' interpretation of MRIs. (Bien, et al., 2018)

In another study, the sagittal planes of MRI scans collected at Hospital Centre Rijeka, Croatia were used to assess knee ligament injuries by applying some machine learning techniques. A total of 969 samples were obtained with labels for healthy, partially torn, and completely torn. Regions of interest were applied with two feature extraction techniques, histogram of gradients (HOG) and scene spatial envelope descriptor (gist descriptor). These features were used with two different classification techniques, namely support vector machines (SVM) and random forests (RF) which resulted in a total of 4 models. Results showed the best model was achieved using HOG with SVM which gave an AUC of 0.943 for the binary problem of detecting a tear, whereas this dropped to an AUC of 0.894. This was probably because of the complexity involved in distinguishing between a partial tear and a healthy ACL. (Štajduhar, et al., 2017)

In a study based upon the multimodal feature extraction using deep neural networks after pre-processing, the extracted features were consequently processed by a multi-layered neural network for correlations fusion of the extracted features before classification. This research highlighted that the sagittal plane of MRIs has greater benefit and gives more accuracy (92.68%) as compared to the other planes for ACL tear detection. Additionally, the group of thirty patients with knee joint injuries was examined through MRI and arthroscopy. The results indicated that individuals with ACL injuries accompanied by bone contusion and medial collateral ligament damage had a higher likelihood of positive findings compared to those with chronic injury alone. Additionally, the prevalence of chronic ACL injury combined with meniscus tear and cartilage damage was observed to be considerably greater than that of acute injuries. (Li, et al., 2021)

Some of the brilliant work done in this subject is the ELNet, an efficiently layered CNN that has blocks of 2D convolutions, multi-slice normalisation, and ReLu activation to add non-linearities. These blocks

increase the spatial dimensions which were controlled using BlurPool, this kept the network quite lightweight with 0.2M parameters which could be trained from scratch using just a single stack of MRI images. Heatmaps of the trained model showed the areas of the image used to detect tears, despite the imbalance of datasets used such as MRNet and KneeMRI, ELNet gave good accuracy of approximately 90%. (Tsai, et al., 2020)

A research study that worked on a lightweight as well as interpretable 3D CNN gave a different direction to the work done previously. It proposed to solve two of the issues, one was the use of CAMs for visualising or identifying the areas being observed in an MRI by the models however this technique fails to identify ACL accurately and all the models worked upon previously are heavily parameterised or computational. The proposed model takes knee MRIs and Gaussian position features, it learns to classify the ACL tear as well as identifies the location. The typical convolutional layers are replaced by squeeze modules to reduce the size of the model, and attention modules that generate attention maps to learn similarity between features. This architecture resulted in a very lightweight model with just 43K trainable parameters achieving an ROC AUC of 0.98 on the MRNet dataset. (Jeon, et al., 2021)

Another research presented a compact parallel deep convolutional neural network (CPDCNN) consisting of three layers with differently sized kernel filters to capture various details from the frames of MRI. In the first, second, and third CNN layers of each parallel segment, there are a total of 32, 64, and 128 convolutional filters used respectively. It showcased a high accuracy of 96.60% while having a much lower number of trainable parameters than the state-of-the-art methods, it has reduced the complexity as well as removed the need for hyper-parameter tuning. (Joshi & Suganthi, 2022)

A systematic examination of the literature on the detection of knee ligament tears reveals extensive research in this area. The scholars undertook a thorough investigation into the implementation of deep learning algorithms for diagnosing knee injuries. Their analysis encompassed studies published between 2013 and 2021. Most of the work conducted in this field can be categorized as machine learning, deep learning with transfer learning, or custom-designed architectures employing deep neural networks. A common approach taken by researchers involved pre-processing data followed by an optional step to localize specific areas before applying a classifier. Many papers incorporated data augmentation techniques such as shifting, flipping, and random rotations. Additionally, it was found that extracting regions of interest proved beneficial in most studies along with suggestions for utilizing CAMs generated from trained models to identify key features sought after by the model when interpreting MRI slices for classification purposes. (Siouras, et al., 2022)

2.5 Research Gap and Challenges

The existing research focuses mainly on determining whether an ACL tear exists or not, with a binary approach. Additionally, the use of MRNet and custom datasets appears to be more prevalent. However, KneeMRI offers a considerable number of samples for identifying different grades of ACL tears, suggesting potential for further exploration in this field. While numerous studies have been conducted using brain MRIs and CT scans, there is still growing interest in applying deep learning techniques to knee MRI analysis. This emerging area has already produced noteworthy results. By leveraging deep learning models,

clinical experts, and radiologists can enhance the diagnostic process and generate accurate reports while mitigating errors caused by fatigue or distractions during MRI assessments. (Liu, et al., 2019)

Even though MRI-based deep learning applications have achieved great results they have been affected by several issues which are listed below (Lee, et al., 2017):

- Legal and ethical issues involving the medical field hinder the wider availability of open-access datasets.
- Dependencies of the deep neural networks on high-quality data and bigger size of the datasets.
- Limited availability of knee MRI datasets as well as high imbalance creates a bottleneck.
- Most of the datasets have homogeneous samples, this makes the dataset lack a wider variety of samples that are present in the practical world. (Kim, et al., 2018)
- The unperceived black-box nature of deep learning models has created a need to obtain interpretable models to observe what a model learns from given samples.
- Samples might be mislabelled depending on the quality of the methods applied to create the dataset.

Several methodologies can be applied to overcome these issues, obtaining data from multiple sites to create a bigger dataset as well as to introduce heterogeneity, data augmentation can be applied to overcome imbalance, more robust ways to label the samples correctly should be adopted and innovative ways of representing what features a model picks up are essential tools for understanding models.

It is also important to take into consideration the three-dimensional nature of medical images when using deep-learning models for analysis. Most conventional computer vision models have been trained on 2D images, but with volumetric information in medical scans such as MRI, valuable insights can be gained. In these cases, a 3D CNN can effectively leverage both spatial features and anatomical structure across the entire volume to capture more comprehensive information. However, it is worth noting that this approach requires higher memory capacity, computational power, and training time compared to traditional methods. (Singh, et al., 2020)

This gives us a motivation to try to use both the publicly available datasets: MRNet and KneeMRI. We will try to augment new data by over-sampling the classes with a low number of cases, this will prevent any loss of information for the majority class if under-sampling is followed for them instead. Class imbalance can also be handled during the training process of the models by calculating class weights. We apply a hybrid approach of augmentation and multi-slice-based transfer learning for our models.

Chapter 3 : CONTRIBUTION

3.1 *Overview*

This chapter explores the fundamental methodology utilized in this dissertation, which centres around improving the evaluation of knee ligament injuries through deep learning. It also covers the details about technologies and resources used in the development of this project, various challenges faced alongside their solutions, and evaluations of the trained models.

The proposed methodology in this study aims to address the research objective of accurately assessing knee ligament injuries using MRI scans and deep learning techniques. One of the major challenges is the lack of medical imaging datasets to address this problem, further, this is complicated by the amount of complexity and processing power involved in training models based on three-dimensional or volumetric data. The combination of different datasets as well as performing augmentations can provide a diverse collection of knee MRI scans, allowing for comprehensive training of the deep learning model.

An overview of the methodology followed in the research is given in the Fig 4. below:

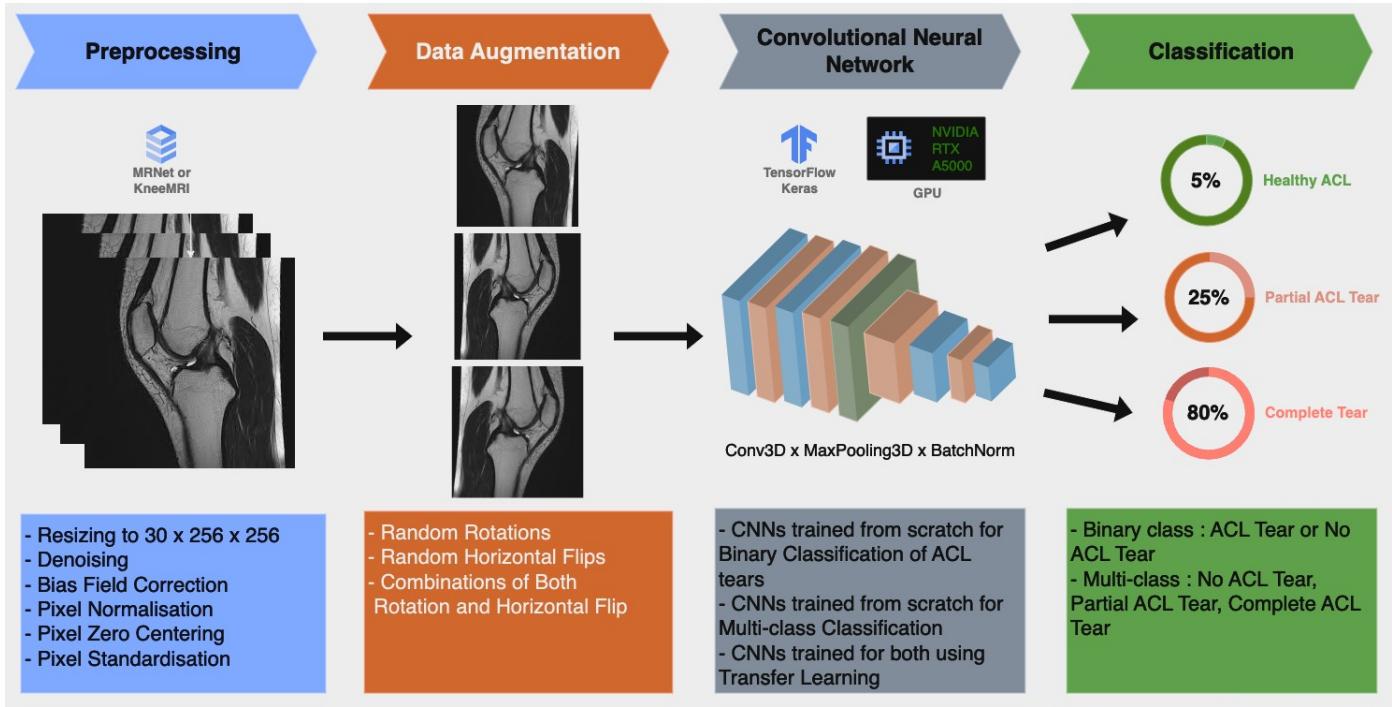


Figure 4 : Methodology

This work employed a comprehensive approach to develop deep learning models for assessing knee ligament injuries using MRI scans. It encompassed several crucial steps, including data acquisition, data pre-processing to ensure uniform quality and format of the MRI scans, training different model architectures, and evaluating the performance of these trained models through metrics such as accuracy, precision, recall, and F1 score for effective classification of various extents of anterior cruciate ligament injuries. Furthermore, the effectiveness of the developed deep learning model was assessed by comparing it with existing approaches in knee ligament assessment.

3.2 Technologies And Resources

The primary technological tools and resources needed for this project are detailed below:

a) Software

a. Python

- Python is widely favoured as the programming language of choice for developing machine learning and deep learning projects. Its simplicity, flexibility, and extensive library support make it a popular choice among developers in these domains.
- It offers deep learning frameworks consisting of a multitude of tools for designing and training neural networks. These frameworks provide a high level of abstraction through APIs, simplifying the process of constructing complex model architectures and facilitating training and evaluation tasks.
- The project will be implemented using Keras, which is a popular framework among others such as TensorFlow, PyTorch, and Theano.

- Python offers a wide range of libraries for pre-processing, transforming, augmenting, and producing visualisations. Prominent packages utilized in computer vision applications include NumPy, SciPy, SimpleITK, OpenCV, PIL, scikit-learn, and scikit-image.
 - Furthermore, Python offers a range of packages like Streamlit that enable the efficient creation and distribution of interactive web applications for machine learning and deep learning endeavours.
- b. Git
- Git is a powerful version control system that facilitates the tracking of changes made in a code repository. Additionally, it aids in versioning and facilitating project sharing for reviews or collaborative purposes.
 - To streamline and enhance the development process of the project, we will incorporate a version control system. Git services such as GitHub and GitLab are widely used for this purpose, and we will select one of them to establish the code repository for our project.

b) Hardware

- a. Personal laptop
 - The MacBook Air, equipped with the advanced Apple M1 chip boasting an 8-core CPU and a 7-Core GPU, is highly capable of training deep learning models. With its substantial specifications including 8GB of RAM, this personal laptop offers efficient handling for the training process of deep learning models.
- b. Google Collab
 - Google Collab offers a Jupyter Notebook interface, allowing users to code and train deep learning models on a virtual machine hosted by the Google Cloud Platform.
 - It offers access to high-performance resources including Nvidia K80 GPU or T4 GPU with a RAM capacity of 12GB.
- c. Kingston University Computers
 - In the event that more powerful hardware is needed, there are high-end computers equipped with robust chips like the Intel Core i7 or AMD Thread-ripper Pro, along with 32GB of RAM or greater.
 - In addition, these computers are equipped with specialized GPUs from Nvidia such as the RTX A5000 and GeForce RTX 3090, which have a dedicated memory capacity of 24GB.

3.3 *Datasets*

The research conducted in this dissertation heavily relied on the utilization of two publicly accessible knee MRI datasets:

a) MRNet



Figure 5 : MRNet Dataset Sample (Bien, et al., 2018)

- The Stanford University Machine Learning Group curated and labelled MRNet, a dataset of knee MRI scans, making it available to the public. (Bien, et al., 2018)
- The MRI scans were performed using GE scanners, specifically the GE Discovery models from GE Healthcare located in Waukesha, WI. These scanners have magnetic fields of both 3.0T and 1.5T.
- In order to compile this dataset, reports from knee MRI examinations conducted at the Stanford University Medical Centre between 1st January 2001 to 31st December 2012 were thoroughly examined by medical experts.
- The dataset includes 1,370 knee MRI scans consisting of details across all three planes: sagittal, coronal, and axial. Skilled radiologists annotated the dataset to indicate the presence of ACL tears, meniscal tears, and other abnormalities.
- An overview of the dataset is given in Table 1. below:

Table 1 : Overview of MRNet Dataset (Bien, et al., 2018)

Property	MRNet												
Origin	The source of the data is as follows: <ul style="list-style-type: none"> - The images were obtained from the knee MRI scanners using DICOM files in three planes: axial, coronal, and sagittal. - Pre-processing techniques were applied to resize the images to 256 x 256 dimensions. Subsequently, intensity standardization was performed followed by pixel value clipping between the range of 0-255. - All samples have been stored as NumPy array files. 												
Channels	1 (grayscale images)												
Samples	A grand total of 1,370 samples were divided into training, validation, and hidden test sets utilizing stratified random sampling methodology. The purpose behind this approach was to guarantee the presence of a minimum of 50 positive instances for each label category (abnormality, ACL tear, and meniscal tear) in every set: <table border="1"> <thead> <tr> <th>Set</th><th>Samples</th><th>Patients</th></tr> </thead> <tbody> <tr> <td>Training</td><td>1,130</td><td>1,088</td></tr> <tr> <td>Validation</td><td>120</td><td>111</td></tr> <tr> <td>Hidden Test</td><td>120</td><td>113</td></tr> </tbody> </table>	Set	Samples	Patients	Training	1,130	1,088	Validation	120	111	Hidden Test	120	113
Set	Samples	Patients											
Training	1,130	1,088											
Validation	120	111											
Hidden Test	120	113											

Labels	Each sample has been labelled with three binary labels:	
	Label	Meaning
	Abnormal	<ul style="list-style-type: none"> - 0 indicates normal (all images reviewed are free of abnormalities) - 1 represents abnormal (These included conditions such as osteoarthritis, effusion, iliotibial band syndrome, posterior cruciate ligament tear, fracture, contusion, plica, and medial collateral ligament sprain)
	ACL tear	<ul style="list-style-type: none"> - 0 indicates intact (normal, mucoid degeneration, mild sprains) - 1 represents tear (grade 1 partial tear with half of fibres torn, high-grade partial tear with more than half of fibres torn, complete tear)
	Meniscal tear	<ul style="list-style-type: none"> - 0 indicates intact (normal, degenerative changes without tear, postsurgical changes without tear) - 1 represents tear (increased signal reaching the articular surface on at least 2 slices or morphologic deformity)
Size on Disk	7.76 GB	

b) KneeMRI



a) Healthy ACL (No tear)

b) Partial ACL tear

c) Complete ACL tear

Figure 6 : KneeMRI Dataset Samples (Štajduhar, et al., 2017)

- The KneeMRI dataset is a comprehensive collection of knee MRI images sourced from the Clinical Hospital Centre Rijeka in Croatia. (Štajduhar, et al., 2017)
- The examinations were performed utilizing a Siemens Avanto 1.5T MR scanner.
- The dataset was obtained retrospectively from examination records at the Clinical Hospital Centre Rijeka, Croatia during the period spanning 2007 to 2014.

- The dataset comprises of 917 knee volumes, each comprising grayscale images with a pixel depth of 12 bits. The categorization labels for these images correspond to the level of severity in ligament injury, specifically classified as healthy, partially injured, or completely ruptured.
- A larger rectangular area of interest (ROI) was manually selected from the original volumes and is also marked with annotations.
- This dataset comes with a metadata csv file that contains information about each MRI file as well as the labels. It provides the ACL diagnosis, whether it is a left or right knee and ROI coordinates across the three dimensions.
- An overview of the dataset is given in Table 2. below:

Table 2 : Overview of KneeMRI Dataset (Štajduhar, et al., 2017)

Property	KneeMRI										
Origin	The source of the data is as follows: - Images were extracted from the sagittal DICOM files gathered from the knee MRI Scanners. - Four experienced radiologists helped in labelling the dataset. - All samples have been stored as Python Pickle objects.										
Channels	1 (grayscale images)										
Samples	A total of 917 samples of left and right knees with labels (not injured/healthy, partially injured, and completely ruptured): <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Set</th> <th>Samples</th> <th>Patients</th> </tr> </thead> <tbody> <tr> <td>Total</td> <td>917</td> <td>Unknown</td> </tr> </tbody> </table>			Set	Samples	Patients	Total	917	Unknown		
Set	Samples	Patients									
Total	917	Unknown									
Labels	Each sample has been labelled according to the ligament condition: <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Label</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>Healthy</td> <td>- 0 represent healthy or not injured ligaments.</td> </tr> <tr> <td>Partially injured</td> <td>- 1 represents a partial tear in the ligament.</td> </tr> <tr> <td>Completely ruptured</td> <td>- 2 represents a complete tear in the ligament.</td> </tr> </tbody> </table>			Label	Meaning	Healthy	- 0 represent healthy or not injured ligaments.	Partially injured	- 1 represents a partial tear in the ligament.	Completely ruptured	- 2 represents a complete tear in the ligament.
Label	Meaning										
Healthy	- 0 represent healthy or not injured ligaments.										
Partially injured	- 1 represents a partial tear in the ligament.										
Completely ruptured	- 2 represents a complete tear in the ligament.										
Size on Disk	6.68 GB										

The integration of these two datasets provides a significant and varied collection of knee MRI scans. By utilizing these resources, the deep learning model was trained to identify complex patterns that indicate ligament tears, enhancing the precision and efficacy of knee injury evaluations. The incorporation of these datasets in this study showcases the significance of openly accessible medical data in propelling research and fostering innovation within the healthcare field.

3.4 Exploratory Data Analysis

The exploratory analysis of data plays a crucial role in fields including computer vision challenges such as those involving medical imaging data. EDA helps to develop a comprehensive understanding of the dataset's properties and reveals valuable insights that can guide further analysis and model-building processes.

a) EDA of MRNet

a. Source files and directories:

- After extracting the downloaded zip file provided by Stanford ML Group. The source files are present in the form of NumPy array files.
- They are already divided into train and valid directories, which further contain the files divided according to three planes axial, coronal, and sagittal.
- Labels are provided in the accompanying CSV files for the three labels, each sample has an entry in three CSV files determining whether the condition is observed or not in that sample denoted by 0 or 1.

Inspect the files

```
# Explore the directory structure and files in the dataset MRNet
!tree -L 2 Data/MRNet-v1.0/
Data/MRNet-v1.0/
└── train
    ├── axial
    ├── coronal
    └── sagittal
    ├── train-abnormal.csv
    ├── train-acl.csv
    ├── train-meniscus.csv
    └── valid
        ├── axial
        ├── coronal
        └── sagittal
        ├── valid-abnormal.csv
        ├── valid-acl.csv
        └── valid-meniscus.csv
9 directories, 6 files
```

Figure 7 : Source files in MRNet Dataset

b. Visualising samples

- Each MRI scan data is a collection of multiple slices, this forms a three-dimensional volume as compared to the usual two-dimensional images used in solving various computer vision tasks using deep learning.
- In order to view sample data, we developed a function in Python that allows us to observe a given sample slice by slice using a slider instead of plotting all the slices of the sample.
- This creates a visually interactive and more intuitive way to view an MRI sample.

```

mrnet_sample_path = 'Data/MRNet-v1.0/train/sagittal/1099.npy'
mrnet_mri_vol = np.load(mrnet_sample_path)
mrnet_mri_vol = mrnet_mri_vol.astype(np.float64) #Change the dtype to float64

explore_3D_volume(mrnet_mri_vol)

```

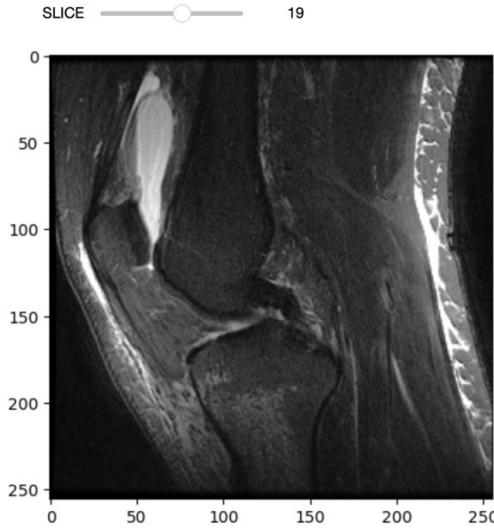


Figure 8 : Exploring MRNet Sample MRI

c. Dimensions of sample

- Each sample has all the slices of size 256 x 256.
- The number of total slices which also constitutes the depth of each sample is not consistent and varies from one sample to another.
- This needs to be handled during the pre-processing stage to make all the samples of consistent dimensions *depth x width x height*.
- We observed some statistics about the number of slices:

```

# TRAIN DATASET
plot_slices_per_exam(datasets['train'])

For DATA/MRNET-V1.0\TRAIN axial plane min : 19, max : 61, avg : 34.316814159292036
For DATA/MRNET-V1.0\TRAIN coronal plane min : 17, max : 58, avg : 29.77787610619469
For DATA/MRNET-V1.0\TRAIN sagittal plane min : 17, max : 51, avg : 30.41592920353982

```

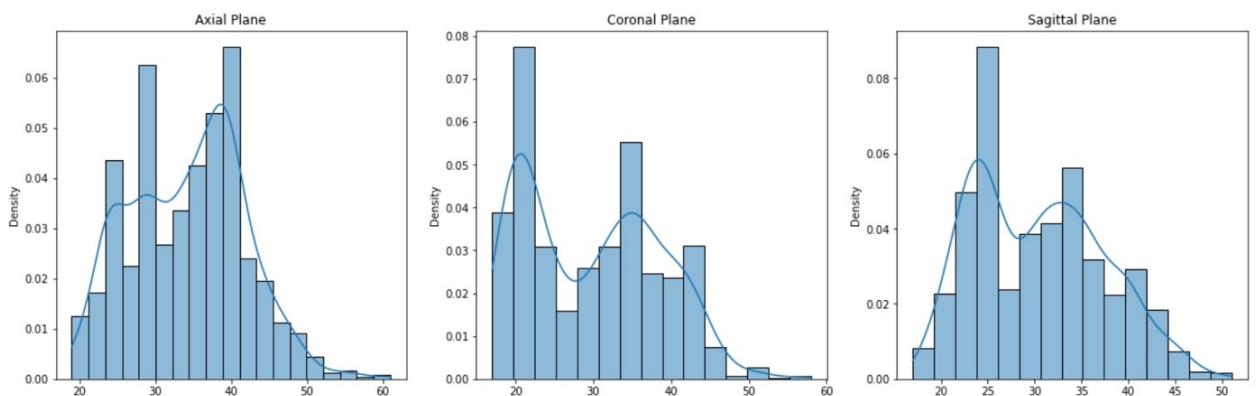


Figure 9 : Analysis of slices in MRNet Train MRI Scans

```
# VALID DATASET
```

```
plot_slices_per_exam(datasets['valid'])
```

```
For DATA/MRNET-V1.0\VALID axial plane min : 20, max : 52, avg : 34.316666666666667
```

```
For DATA/MRNET-V1.0\VALID coronal plane min : 17, max : 48, avg : 29.425
```

```
For DATA/MRNET-V1.0\VALID sagittal plane min : 21, max : 45, avg : 30.525
```

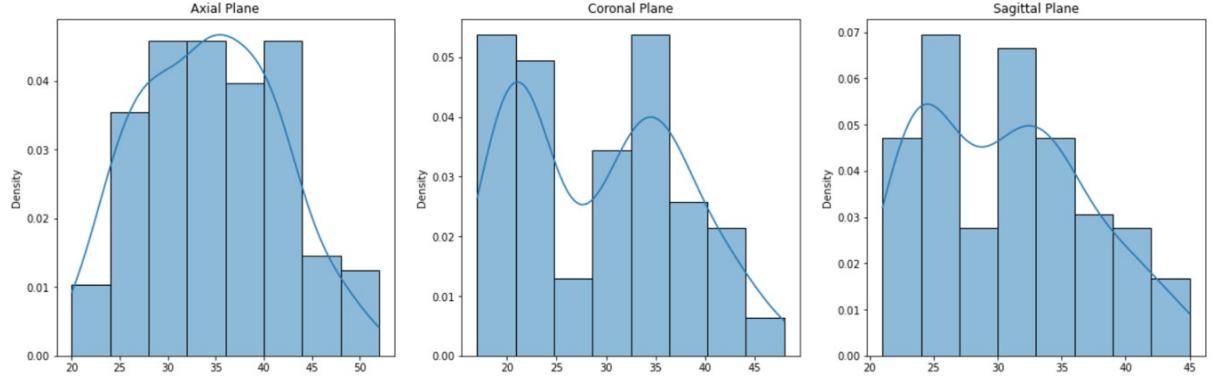


Figure 10 : Analysis of slices in MRNet Valid MRI Scans

- It was observed that both train and valid splits of dataset had similar number of slices in their samples, ranging mostly between 20 to 40 slices. Overall, we observed an average number of 30 slices, and we will resize all the samples for the same during the pre-processing.
- d. Number of samples
- The dataset consists of a total of 1130 samples in train and 120 samples in valid splits.
 - Out of these, the number of samples corresponding to the respective labels and split is as follows:

Table 3 : Samples in MRNet Dataset

Dataset	Samples		
Train	Abnormal 0: 217 1: 913	ACL 0: 922 1: 208	Meniscus 0: 733 1: 397
Valid	Abnormal 0: 25 1: 95	ACL 0: 66 1: 54	Meniscus 0: 68 1: 52
Total	Abnormal 0: 242 1: 1008	ACL 0: 988 1: 262	Meniscus 0: 801 1: 449

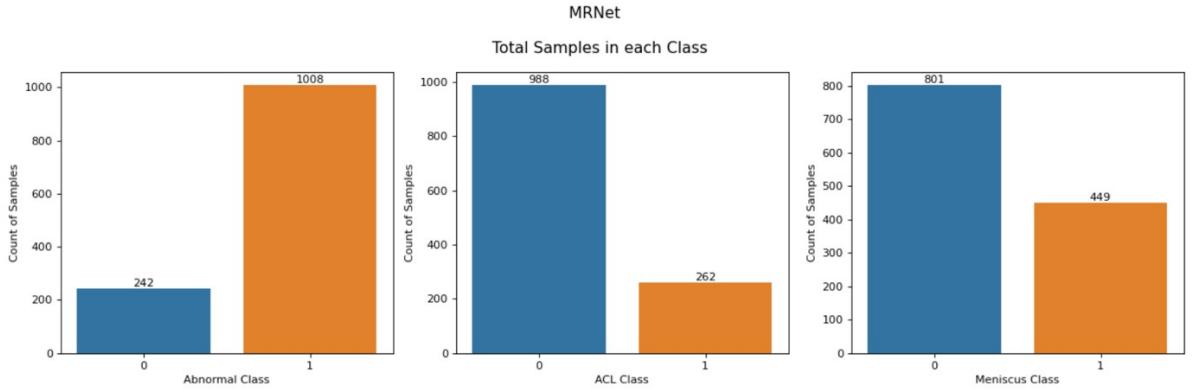


Figure 11 : Number of samples in MRNet Dataset (Train + Valid)

MRNet
Pie Chart of ACL Class Imbalance

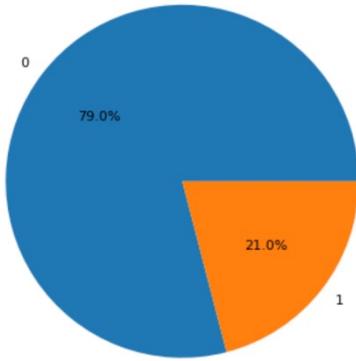


Figure 12 : ACL Class imbalance in MRNet Dataset (Train + Valid)

- We observe an imbalance in the class label ACL, the number of samples with a tear is way lower than the number of samples without a tear.
- e. Co-occurrence of labels
- In MRNet dataset, a sample can be labelled to have all three conditions and they are not mutually exclusive. However, each condition is a binary label where 0 means not present and 1 means present.

Table 4 : Co-occurrence of labels in MRNet Samples

Sr. No.	Abnormal	ACL	Meniscus	Total Cases	Percentage %
1	0	0	0	217	19.20 %
2	1	0	0	433	38.32 %
3	1	0	1	272	24.07 %
4	1	1	0	83	07.35 %

5	1	1	1	125	11.06 %
---	---	---	---	-----	---------

- This gives an overview of the three medical conditions and the number of samples where the multiple labels occur together.

b) EDA of KneeMRI

a. Source files and directories:

- The dataset is available as ten zip files that can be downloaded from the source
- Alongside a CSV file is provided that contains information regarding whether it was a left or right leg knee, a diagnosis label that states the condition of ACL, and region of interest.
- All the cases are made available as Python pickle files.

Inspect the files

```
# Explore the directory structure and files in the dataset MRNet
!tree -L 1 Data/KneeMRI/
```

```
Data/KneeMRI/
├── metadata-aug.csv
└── metadata.csv
    ├── vol01
    ├── vol02
    ├── vol03
    ├── vol04
    ├── vol05
    ├── vol06
    ├── vol07
    ├── vol08
    ├── vol09
    └── vol10
```

11 directories, 2 files

Figure 13 : Source files in KneeMRI Dataset

b. Visualising samples

- We can use the function developed earlier to visualise MRI samples interactively.

```
kneemri_sample_path = 'Data/KneeMRI/vol01/491177-5.pck'
with open(kneemri_sample_path, 'rb') as file_handler: # Must use 'rb' as the data is binary
    kneemri_vol = pickle.load(file_handler)
    kneemri_vol = kneemri_vol.astype(np.float64) # Change the dtype to float64
```

```
explore_3D_volume(kneemri_vol)
```

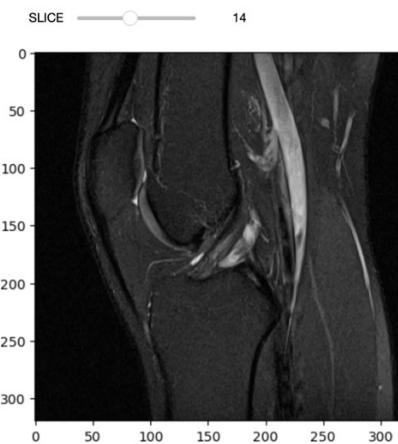


Figure 14 : Exploring KneeMRI Sample MRI

c. Dimensions of sample

- MRI scans in this dataset have slices of dimension 320x320 in all the samples.

- Here as well, we observe the depth of samples varies and needs to be fixed in the pre-processing of the dataset to get a similar shape for all the scans.

```
plot_slices_per_exam(mri_vol_paths)
For KneeMRI Sagittal plane min : 21, max : 45, avg : 30.925845147219192
```

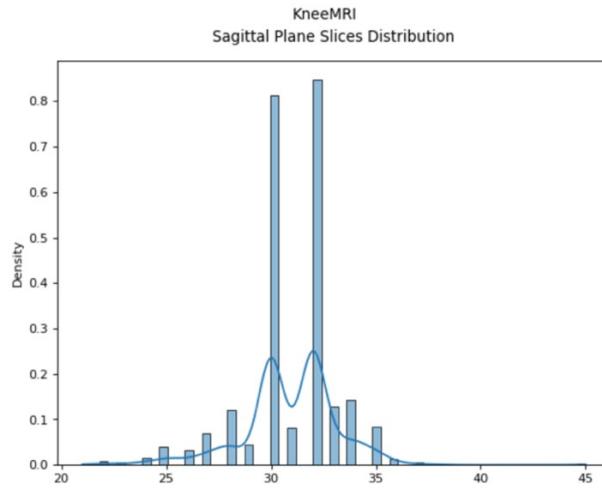


Figure 15 : Analysis of slices in MRNet Valid MRI Scans

- We observe the number of slices in the MRI samples of this dataset also lies between 20 to 40, with an average number of 30 slices. We will resize all the samples in this dataset as well to match dimensions as 30x256x256.
- d. Number of samples
- A total of 917 samples are present in this dataset, there's no prior division or splitting into train, valid or test datasets.

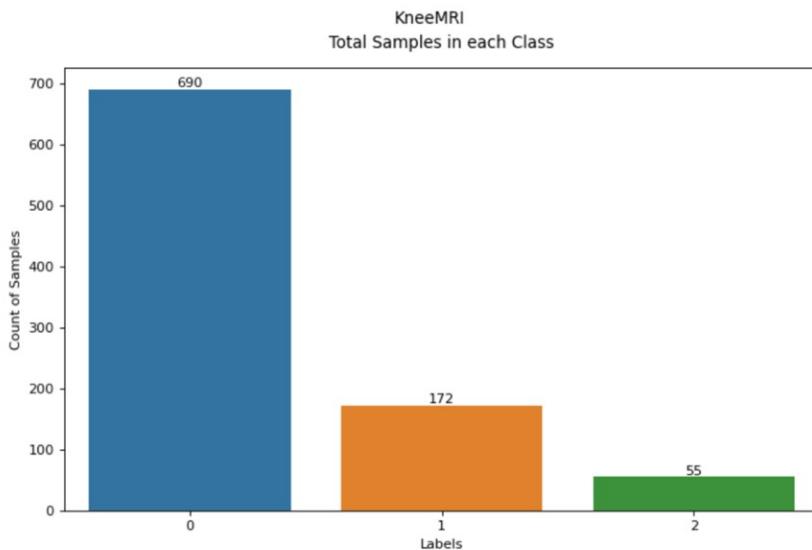


Figure 16 : Number of samples in KneeMRI Dataset

- The samples are labelled for one of the three conditions applying to ACL knee ligament health status:

Table 5 : Samples in KneeMRI Dataset

Label	Samples	Percentage %
Healthy	690	75.24%
Partial tear	172	18.76%
Complete tear	55	6%

- This highlights the class imbalance in the dataset that needs to be considered while training.

KneeMRI
Pie Chart of Class Imbalance

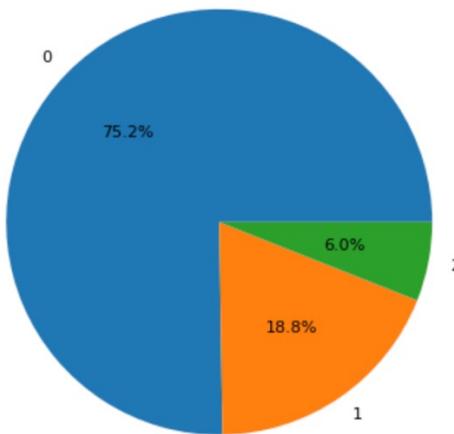


Figure 17 : ACL Class imbalance in KneeMRI Dataset

e. Co-occurrence of labels

- The samples in this dataset have been labelled for one of the conditions that applies, the labels are mutually exclusive and denotes only one of the conditions of the ACL ligament.

3.5 Data Preprocessing

Data pre-processing is an extremely important part of the process of developing good models, it ensures the quality and consistency of the data we feed for training, evaluation, and testing. We must follow the same steps and pre-process any sample before using the model for predictions.

The exploratory analysis of the two datasets yielded significant findings that necessitate resolving through the application of essential pre-processing techniques specific to MRI scans:

a) Resizing

- As observed in EDA both the datasets have MRI samples with an inconsistent number of slices, we will reshape them to have consistent data.
- The average number of slices across both datasets is 30. Hence, we will reshape the samples into a shape of 30 x 256 x 256.

```

print(f"Knee MRI Sample shape : {kneemri_vol.shape}")
kneemri_vol_resized = resize_3D_volume(kneemri_vol)
print(f"Resized Knee MRI Sample shape : {kneemri_vol_resized.shape}")

Knee MRI Sample shape : (33, 320, 320)
Resized Knee MRI Sample shape : (30, 256, 256)

```

Figure 18 : Original MRI volume shape and resized volume shape

b) Denoising

- One of the crucial steps in preparing any dataset for computer vision tasks is to denoise the samples. We applied the denoising on the entire MRI scan volume after resizing it to the desired shape.

```

mrnet_mri_vol_denoised = denoise_3D_volume(mrnet_mri_vol)
compare_3D_volume(mrnet_mri_vol, mrnet_mri_vol_denoised)

```

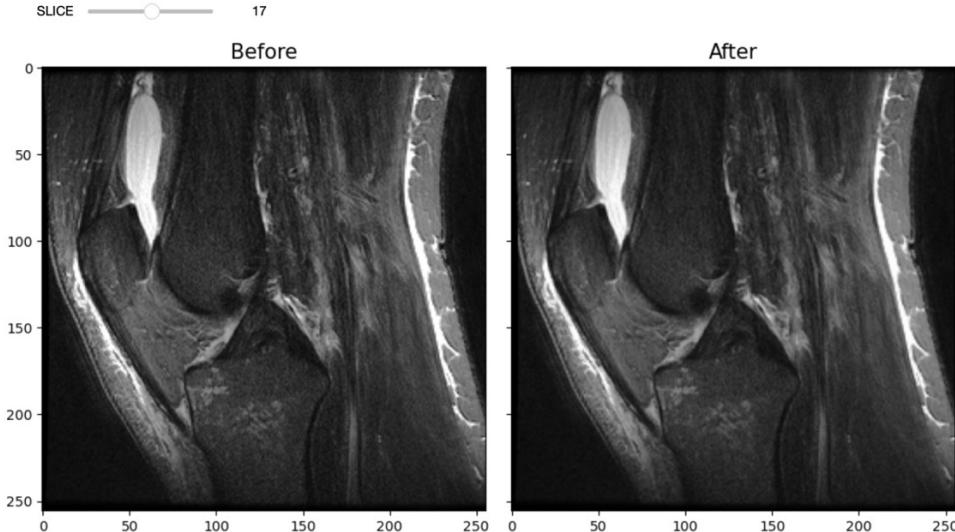


Figure 19 : Denoising of MRI Sample

- We observe that denoising the MRI sample smoothens it and helps to improve the texture, it improves the quality of MRI scan by removing noise as seen in Fig 19.

c) Bias field correction

- Another crucial part of pre-processing MRI scans is to perform bias field correction. Bias fields are low-frequency signals that corrupt MRIs.
- These introduce gradual or non-uniform changes in pixel intensities, which are usually introduced by inadequate acquisition of scans or some external disturbances to the magnetic fields of the MRI machines.
- A bias field can lead to blurring of the image and degrade the edges or contours in the scans. This leads to different pixel intensities of the same anatomical tissues.
- We applied the N4 bias field correction algorithm, however, it's a heavy operation and takes a considerable amount of time to process a sample.

```
%time
knee_mri_vol_bfc = bias_field_correction_volume(kneemri_vol_denoised)
```

```
CPU times: user 7min 32s, sys: 2.85 s, total: 7min 35s
Wall time: 1min 8s
```

```
%time
knee_mri_vol_eff_bfc = efficient_bias_field_correction_volume(kneemri_vol_denoised)
```

```
CPU times: user 4 s, sys: 207 ms, total: 4.21 s
Wall time: 1.18 s
```

Figure 20 : Time comparison of Bias Field Corrections

- After some research, we improved the way of applying the algorithm by shrinking the volume and retaining the original shape back again after the process was done, this significantly reduced the processing time to a few seconds for each sample as seen in Fig 20.

```
mrnet_mri_vol_bfc = efficient_bias_field_correction_volume(mrnet_mri_vol_denoised)
```

```
compare_3D_volume(mrnet_mri_vol_denoised, mrnet_mri_vol_bfc, cmap='nipy_spectral')
```

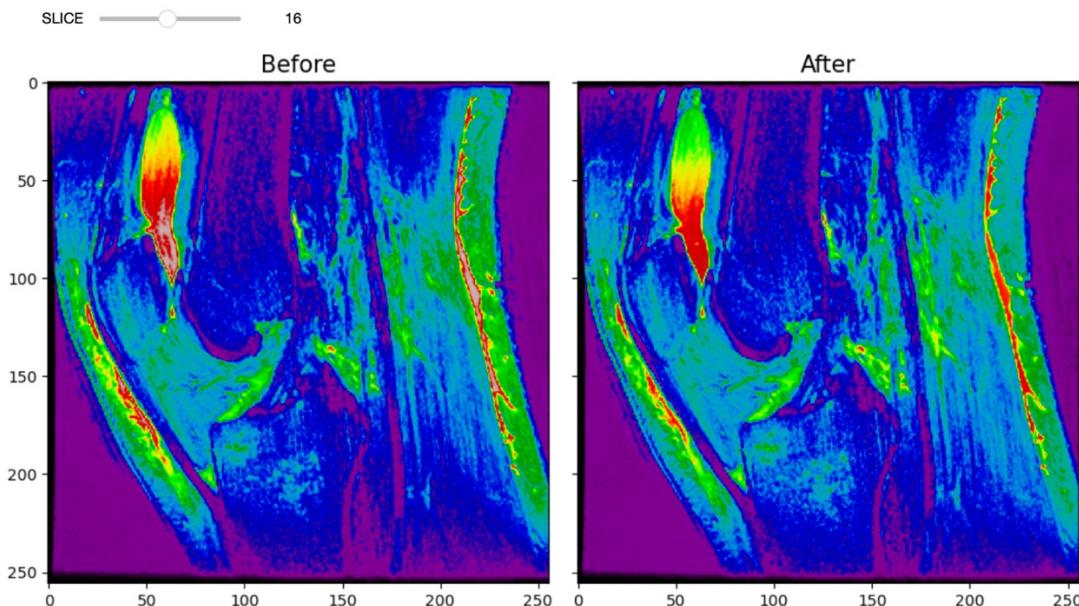


Figure 21 : Bias Field Correction of MRI Sample

- We can observe from Fig 21. that bias field correction has improved the intensity of similar tissues in the scan, this helps the model to identify the features in an MRI scan more easily.

d) Pixel intensity normalisation

- Pixel intensity normalisation helps to accelerate model learning, a common technique that scales the pixel values in the range between 0 to 1. This process involves dividing each individual pixel value of an image by the maximum possible value for that particular type of pixel (e.g., 255 for an 8-bit image).

e) Pixel standardisation

- Standardisation of the pixels helps to achieve a mean value of zero and a unit standard deviation, converting the distribution of pixels into a Gaussian distribution.

f) Pixel zero centering

- Zero centering is preferred when activation functions such as sigmoid and ReLu are used as it can help prevent gradient saturation.

- The average of the pixels is subtracted from making it centered around zero i.e. a mean of zero.

g) Data augmentation

- We observed the class imbalance during the EDA of the datasets, in order to generate more samples of the labels that are less.
- We applied data augmentations on the classes with less number of samples, random horizontal flips with only a 5% chance of augmentation if the labels were from classes having a high number of cases, and a 100% chance for the label that has a minority of samples.
- In MRNet, we augmented a random negative sample with only a single augmentation of flipping and rotation. However, for each positive sample, we created three different augmentations by using similar augmentations.
- On the other hand, we augmented KneeMRI in a similar fashion for negative samples but for each positive sample belonging to partial tears, we generated two augmentations and for a completely ruptured sample we augmented five more samples.
- We implemented the data augmentations for the two datasets by applying these two techniques:

a. Random rotation

- We tried to apply random rotations to augment the data samples, this resulted in samples with newer perspectives after rotations as seen below.
- We randomly rotated a given sample with degrees between -2 and +2 to avoid too many deviations from an original scan.

```
mrnet_mri_vol_rotated = random_rotation(mrnet_mri_vol_bfc)
compare_3D_volume(mrnet_mri_vol_bfc, mrnet_mri_vol_rotated)
```

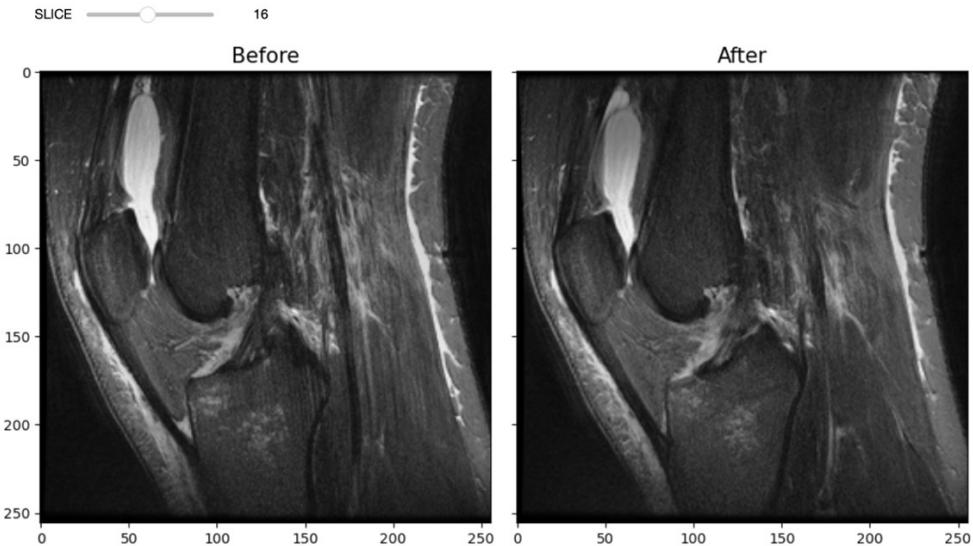


Figure 22 : Random rotation data augmentation of MRI sample

b. Random horizontal flips

- Flipping the entire MRI volume horizontally turned out to be a good way to augment more samples.
- It can apparently generate samples of the opposite knee if not present in the datasets, a left knee MRI can become a right knee MRI and vice versa.

```

mrnet_mri_vol_flipped = random_horizontal_flip(mrnet_mri_vol_bfc)
compare_3D_volume(mrnet_mri_vol_bfc, mrnet_mri_vol_flipped)

```

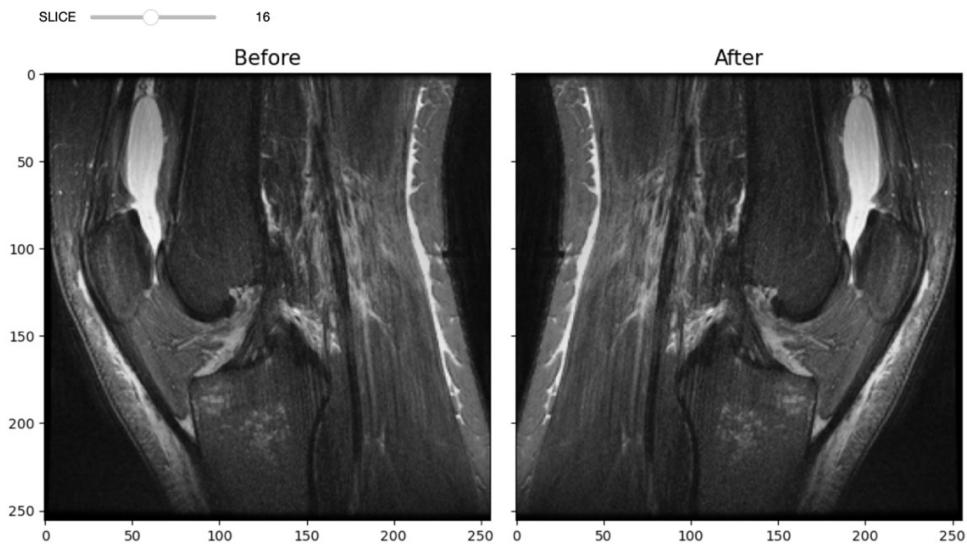


Figure 23 : Random horizontal flip data augmentation of MRI sample

- After applying data augmentation, the number of samples in the MRNet dataset from each of the classes have been represented graphically in the Fig. below:

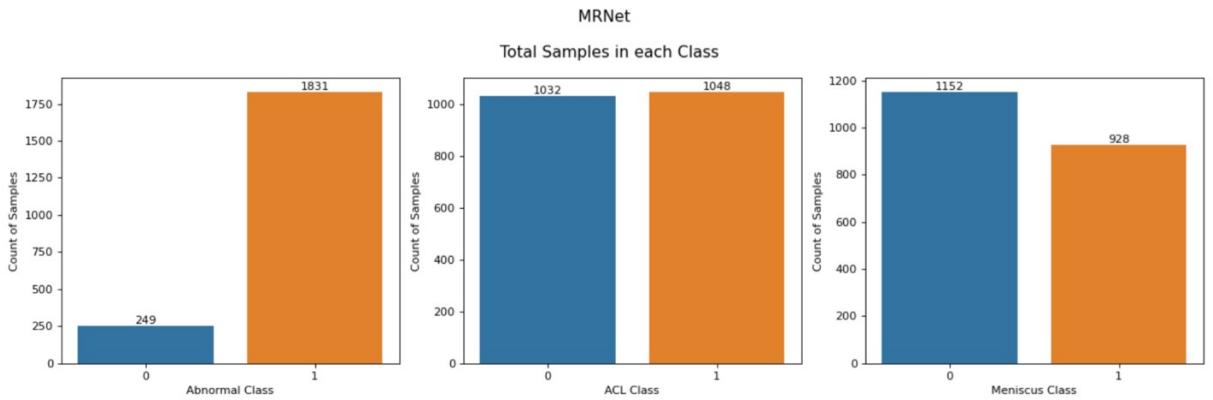


Figure 24 : MRNet Dataset after Data Augmentation

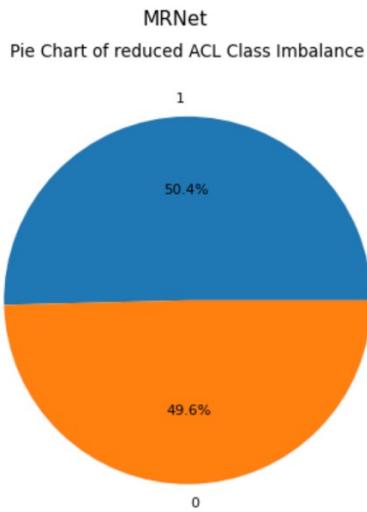


Figure 25 : MRNet with reduced ACL Class Imbalance

- Similarly, the data augmentation on the KneeMRI dataset gave us the following number of samples belonging to each class represented in the Fig. below:

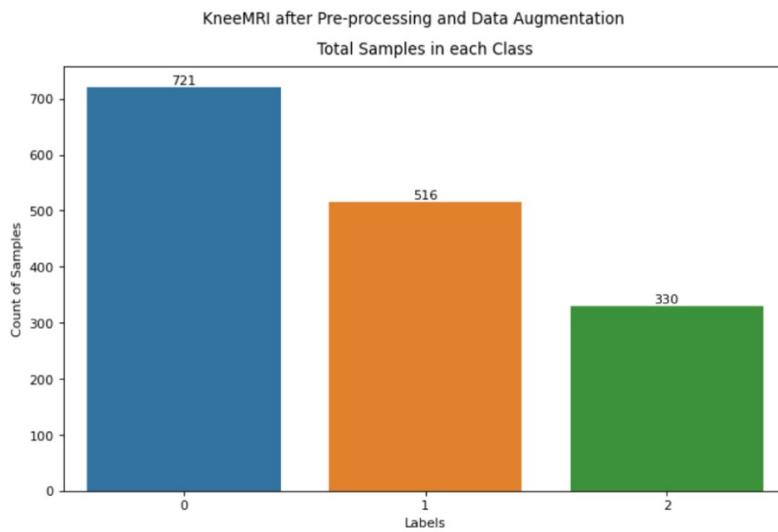


Figure 26 : KneeMRI Dataset after Data Augmentation

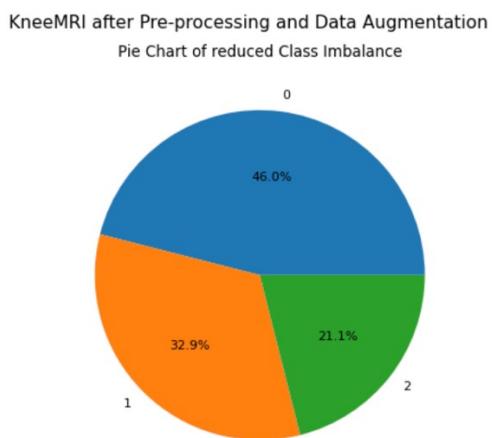


Figure 27 : KneeMRI with reduced Class Imbalance

3.6 Convolutional Neural Networks

Deep Learning has significantly impacted computer vision tasks, including image classification, object detection, and medical imaging. Convolutional Neural Networks (CNNs) have allowed the processing of grid-like data such as images and even volumes to learn features that are essential to tackle tasks such as classification.

CNNs that work with volumetric data are called three-dimensional Convolutional Neural Networks (3D CNNs). These expand on the concept of 2D CNNs by working with three-dimensional data, such as volumetric images or video sequences. While 2D CNNs process two-dimensional images, 3D CNNs operate on entire three-dimensional volumes such as MRI or CT scans.

A typical architecture of CNNs is as follows with some additional information about the layers we have applied in our model architectures is given below:

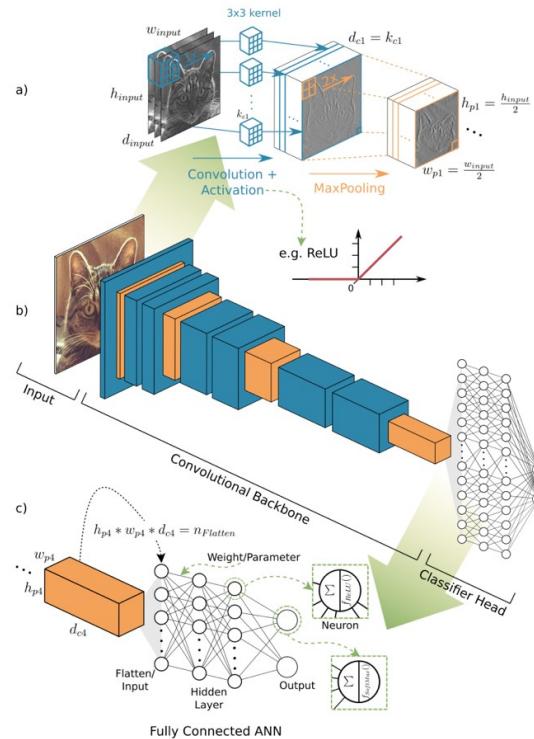


Figure 28 : Typical Convolutional Neural Network Architecture (Hoeser & Kuenzer, 2020)

a) Input Layer

- This is the initial layer of a model; it is responsible for consuming the input data and providing it to the further layers for extracting valuable features of the input data and learning.
- It's generally a good practice to add any additional pre-processing layers after the input layer, but in our case, we pre-processed the data as it was huge, and it can consume a lot of time with every model being trained. Hence, we have pre-processed the datasets separately.

b) Convolutional Layer

- This is considered the core layer of CNN architecture and does most of the work when we are training a model to do a certain task.
 - The convolutional layer consists of kernels or filters, which are nothing but matrices that slide over an input data image over each channel of the input to learn features. The model adjusts the weights of this kernel to reduce the loss function and identify features that are important.
 - With every epoch, the model goes over the training dataset and the loss is backpropagated through the layers leading to updates in the weights.
 - Convolutional layers at the start of a model may learn some basic detection of edges or patterns, as the layers go deeper into the model these learn more complex things such as identifying a face or certain type of object. (CS231n, 2021)
 - 3D convolutional layers can traverse the entire space or volume, giving it the ability to learn not only the spatial features but also temporal features. This makes it a powerful tool that has helped not only in medical imaging but also in action or gesture recognition through video analysis, pose estimation or object recognition, and autonomous vehicles to detect their surroundings.

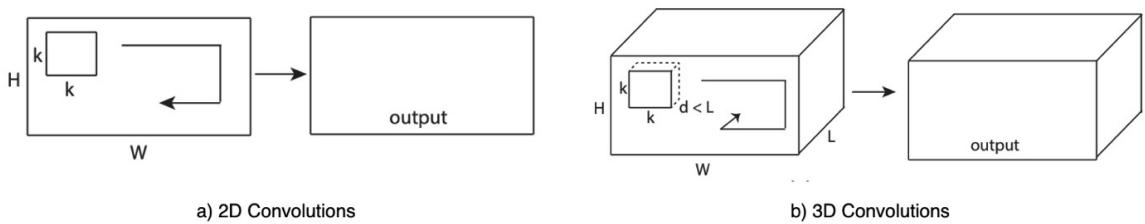


Figure 29 : Comparison of 2D and 3D Convolutions (Tran, et al., 2015)

c) Activation Layer

- Following each convolutional operation, a non-linear activation function (typically ReLU - Rectified Linear Unit) is utilized to introduce non-linearity. This allows the network to effectively capture intricate relationships within the data.

d) Pooling Layer

- Pooling layers such as max-pooling or average-pooling aid in decreasing the spatial dimensions of feature maps while preserving crucial information. This offers computational efficiency by reducing the number of parameters and allows the network to prioritize vital features.

e) Batch-Normalisation Layer

- Batch normalisation layers are used in training deep neural networks that standardizes the inputs within each mini-batch by re-centering and re-scaling to adjust the mean output to 0 and a standard deviation close to 1.
 - It helps to stabilize the learning process and significantly reduces the number of training epochs needed for successful training of deep networks.

f) Dropout Layer

- Dropout layers are a regularisation technique to prevent overfitting, which happens when a model becomes too specialized in performing well on the training data but fails to generalize

effectively to new, unseen data. Dropout layers play a crucial role in implementing this technique.

- It randomly sets certain connections or nodes to zero while training in order to influence the model architecture to learn with each training pass.
- In general, these are added after the last pooling layer in the model architecture.

g) Dense or Fully Connected Layer

- Fully connected layers, also known as dense layers, link all neurons from the previous layer to the current layer in order to make decisions. These layers focus on learning and classifying input data based on the features that have been extracted throughout the network.

h) Output Layer

- This makes up the final layer of a CNN, it typically consists of one or more neurons, depending on the classification task.
- It uses an appropriate activation function such as Sigmoid in binary classification or SoftMax for multi-classification in order to produce the network's output probabilities or scores.

3.7 Training Pre-requisites

In order to start training our models for the classification problem about knee ligament injuries, we split the dataset and set some important call-back functions in Keras whose details are given below:

a) Dataset Splits

- In order to start training our models on the data, we are required to split our dataset ideally into three portions for training, validation, and testing.
- Training split is the data actually used to train a model, whereas validation split is used to see the performance of the model in each pass or epoch of training.
- Lastly, a testing dataset split is something not seen by the model and helps to gauge the performance of the model for unseen data and assists in identifying if the model performs practically.
- We use stratification when splitting the dataset to ensure all the splits have the same ratio of classes, this helps to make sure the model learns on all the labels that are present.

b) Call-back Functions

- Keras provides an option to write certain functions known as call-back functions. These call-back functions play a vital role in customizing the training process of neural networks in Keras.
- These functions are executed at specific stages during training and allow for various actions such as monitoring metrics, adjusting learning rates, saving checkpoints, and more. By implementing call-backs, users can enhance the model's performance and improve the overall efficiency of the training process with increased flexibility and customization options.
- Some of the call-back functions that we have used:
 - a. Learning Rate Scheduling
 - This function allows us to set an initial learning rate before starting the training, and dynamically control it as the training progresses.

- The learning rate decays further as the model comes close to convergence and helps the model to learn better.

b. Early Stopping

- It is one of the crucial call-back functions that allows to set the number of epochs to some arbitrarily large number and monitors a validation metric (e.g., validation loss or accuracy) to stop the training when it can't be improved further, preventing overfitting and saves time.
- We monitor the validation loss in our models during the training, to stop the training once the validation loss stops improving, we also restore the weights from the best epoch.

c. Checkpoint

- This call-back facilitates saving the to save the weights of a model at specific intervals throughout training.
- This method is advantageous as it allows for the restoration of the optimal model weights in case of interruption during training or for performing inference using the best model.
- It monitors the validation loss by default, we used it to save the best model along with its weights from the epoch exemplifying minimum validation loss in the required model directory.

c) Class weights

- As we observed there was an imbalance in the dataset, we applied various augmentations to create a greater number of samples for classes with lower data samples to overcome the imbalance.
- Determining class weights is another technique to deal with class imbalance, it involves assigning varying weights to different classes based on their distribution in the dataset.
- This assists the model in providing increased importance to minority classes during training, it can potentially improve overall performance, particularly when certain classes are not well-represented in the data.

3.8 Need for GPUs

Another challenge before we proceed to train the 3D CNNs is the sheer computational requirement of these models. Speed is a crucial advantage of using GPUs. They dramatically reduce training times compared to Central Processing Units (CPUs). What might take weeks to train on a CPU can be completed in mere hours or minutes on a GPU. This accelerated pace is vital for rapid model development and experimentation.

Training 3D Convolutional Neural Networks (3D CNNs) often necessitates considerable computational resources. The utilization of Graphics Processing Units plays a crucial role in this regard due to their outstanding capability for parallel processing. GPUs have been specifically optimized to perform matrix operations and convolutions, which are fundamental components of deep learning tasks. As a result, they exhibit exceptional efficiency when utilized in the training process of 3D CNNs.

The use of GPUs provides a significant advantage in terms of speed. Compared to Central Processing Units, GPUs can greatly reduce training times. What could take several weeks to train on a CPU can be completed in just hours or minutes with a GPU. This accelerated pace is crucial for efficient model development and experimentation processes.

Taking advantage of the Apple M1 GPU was sadly cut short due to incompatibility between TensorFlow Keras and most of the Apple MacBooks today. In order to effectively utilize GPU acceleration, it is generally necessary to have either an external GPU (eGPU) or a dedicated NVIDIA GPU. This is because TensorFlow predominantly supports GPUs that are compatible with CUDA. However, Apple discontinued the use of Nvidia Graphic Cards in 2014 and mostly comes with integrated GPUs now, which lacks the support of drivers to train deep neural networks.

TensorFlow Keras has great compatibility to train neural networks through most of the Nvidia GPUs, this gave me an opportunity to use the machines at Kingston University to train the required models. We set up one of the Windows machines in the lab by following the Windows Native TensorFlow guide and successfully tested the difference in time required for training models using GPU and CPU. (Tensorflow Installation Guide, 2023)

On CPU

```
%%time
with tf.device('/device:CPU:0'):
    history = model.fit(utils.batch_generator(X_train_temp, y_train_temp, BATCH_SIZE),
                         steps_per_epoch=len(X_train_temp)//BATCH_SIZE,
                         epochs=5,
                         validation_data=utils.batch_generator(X_valid_temp, y_valid_temp, BATCH_SIZE),
                         validation_steps=len(X_valid_temp)//BATCH_SIZE,
                         shuffle=True,
                         class_weight=class_weights_temp,
                         verbose=1,
                         callbacks=[utils.model_callback_checkpoint('CPUvsGPU'), utils.model_callback_earlystopping()])

Epoch 1/5
16/16 [=====] - 386s 24s/step - loss: 1.0875 - accuracy: 0.9297 - val_loss: 6.0008 - val_accuracy: 0.7188
Epoch 2/5
16/16 [=====] - 369s 23s/step - loss: 0.3787 - accuracy: 0.9375 - val_loss: 7.2662 - val_accuracy: 0.5625
Epoch 3/5
16/16 [=====] - 381s 24s/step - loss: 0.1403 - accuracy: 0.9766 - val_loss: 4.1612 - val_accuracy: 0.7812
Epoch 4/5
16/16 [=====] - 370s 23s/step - loss: 0.4838 - accuracy: 0.9219 - val_loss: 9.6759 - val_accuracy: 0.5938
Epoch 5/5
16/16 [=====] - 381s 24s/step - loss: 0.4018 - accuracy: 0.9453 - val_loss: 2.7049 - val_accuracy: 0.7812
Wall time: 31min 28s
```

Figure 30 : Training on CPU

On GPU

```
%time
with tf.device('/device:GPU:0'):
    history = model.fit(utils.batch_generator(X_train_temp, y_train_temp, BATCH_SIZE),
                         steps_per_epoch=len(X_train_temp)//BATCH_SIZE,
                         epochs=5,
                         validation_data=utils.batch_generator(X_valid_temp, y_valid_temp, BATCH_SIZE),
                         validation_steps=len(X_valid_temp)//BATCH_SIZE,
                         shuffle=True,
                         class_weight=class_weights_temp,
                         verbose=1,
                         callbacks=[utils.model_callback_checkpoint('CPUvsGPU'), utils.model_callback_earlystopping()])

Epoch 1/5
16/16 [=====] - 21s 1s/step - loss: 51.4816 - accuracy: 0.4453 - val_loss: 20.8152 - val_accuracy: 0.5625
Epoch 2/5
16/16 [=====] - 18s 1s/step - loss: 16.5783 - accuracy: 0.6250 - val_loss: 9.4362 - val_accuracy: 0.6562
Epoch 3/5
16/16 [=====] - 18s 1s/step - loss: 7.3881 - accuracy: 0.7031 - val_loss: 3.6138 - val_accuracy: 0.7188
Epoch 4/5
16/16 [=====] - 6s 361ms/step - loss: 2.4683 - accuracy: 0.8047 - val_loss: 4.9877 - val_accuracy: 0.6250
Epoch 5/5
16/16 [=====] - 6s 358ms/step - loss: 0.5582 - accuracy: 0.9219 - val_loss: 4.6749 - val_accuracy: 0.6875
Wall time: 1min 8s
```

Figure 31 : Training on GPU

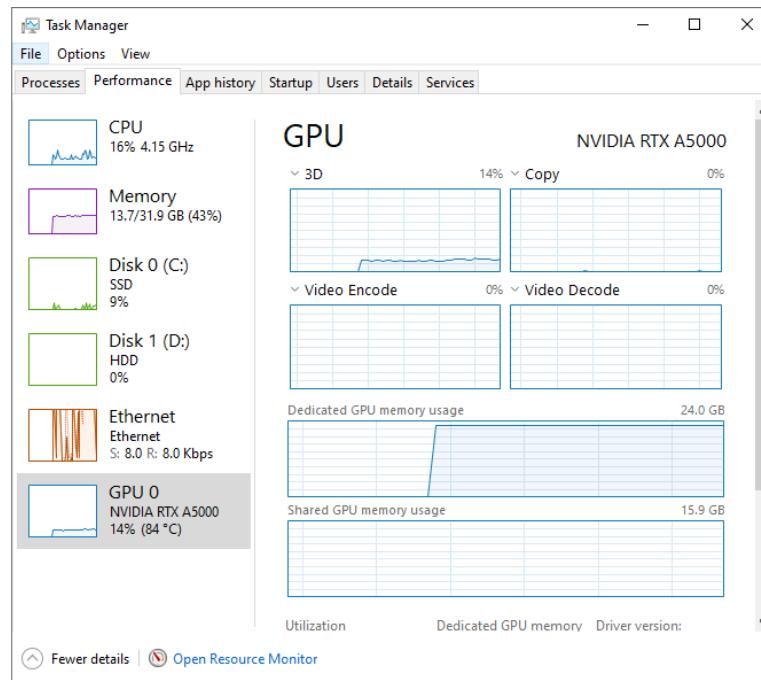


Figure 32 : Usage of GPU (Nvidia RTX A5000)

Comparing the time taken to train a simple model on a subset of the dataset clearly shows the need of GPU to train our models on both the datasets.

3.9 Batch Processing

The amount of RAM required to load such huge datasets can be significant and may not suffice to load all of the datasets together. In order to overcome this issue, we wrote a function that can load batches of the dataset as we train the model. This concept is known as batch training, it is a crucial method in deep learning that allows for the efficient handling of large datasets while considering memory limitations. (Brownlee, 2019)

To enhance the model's ability to generalize and prevent it from memorizing the data sequence, batches are frequently presented in a randomized or shuffled order during each epoch. This approach effectively mitigates overfitting, as it ensures that the model performs well not only on training data but also on unseen data.

In addition to its advantages in enabling parallel computations, batch training allows for the efficient utilization of modern multi-core CPUs and GPUs. This enables the preparation of other batches while one is being processed, thereby reducing the time required to complete a full epoch over the entire dataset and accelerating training.

To summarise, batch training turned out to be a crucial approach for efficiently training our deep neural networks with the MRI scan datasets. It allows efficient memory utilization, introduces randomness to improve convergence, and leverages parallelism to expedite the training process.

3.10 Evaluation Metrics

Evaluation metrics are crucial to assess the performance of trained models for a given task. The selection of appropriate metrics depends on various factors, including the problem's nature, class distribution, and specific application goals.

We are trying to train models that can identify knee ligament injuries from MRI scans, this falls under the classification task. Classification problems involve predicting the correct labels for the given data. It is an example of supervised learning; it may be dealing with just two classes resulting in binary classification or even more than two classes which leads to multi-class classification. However, the same evaluation metrics can be applied to both scenarios.

We used the following metrics to evaluate each of our models (Zheng, 2015):

a) Confusion Matrix

- The confusion matrix is not considered a precise metric, but rather it aids in observing the true labels and predicted labels of samples. Presented as a table, it provides an overview of the overall performance.

		Tear	Healthy
		1000 Samples TRUE POSITIVES	200 Samples FALSE NEGATIVES
True ACL Diagnosis	Tear		
	Healthy	800 Samples FALSE POSITIVES	8000 Samples TRUE NEGATIVES
Predicted ACL Diagnosis			

Figure 33 : Example of a Confusion Matrix

- Some important terms related to a confusion matrix are as follows:
 - True Positive (TP) is a sample that is predicted as positive, and it's originally a positive as well.
 - False Positive (FP) is a sample predicted as positive but it's originally a negative.
 - True Negative (TN) is a sample originally negative as well as predicted as negative.
 - False Negative (FN) is a sample originally positive but marked as negative in the prediction.

b) Accuracy

- Accuracy determines the overall number of correct predictions out of all the predictions made.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + FP + TN + FN} \quad (1)$$

- It is one of the commonly used evaluation metrics and is simple to grasp, its value lies between 0 to 100 in percentage or 0 to 1.
- However, accuracy only works well when the dataset is in fact balanced. In case of an imbalance, it is unable to capture the performance for each class.
- For example, a dataset with 98% of positive samples can lead to a model predicting all samples as positive and achieving great accuracy. But in reality, the model didn't learn to classify the negative samples.

c) Precision

- Precision allows to capture the proportion of correct positive predictions out of all the positive predictions.

$$\text{Precision} = \frac{\text{Number of correct positive predictions}}{\text{Total number of positive predictions}} = \frac{TP}{TP + FP} \quad (2)$$

- It's useful in scenarios where we can safely ignore or avoid False Negatives but can't tolerate False Positives.
- For example, for a spam detection model, it's fine if we get a couple of spam to the inbox but not if any important mail goes into the spam.

d) Recall

- Recall gives the proportion of correct positive predictions out of all actual positive data points; this gives an insight into the missed positive predictions.

$$Recall = \frac{Number\ of\ correct\ positive\ predictions}{Total\ number\ of\ actual\ positives} = \frac{TP}{TP+FN} \quad (3)$$

- It's helpful in situations where False Positives can be ignored, but False Negatives should not occur.
- For example, if we are identifying some disease then it's okay to predict some healthy people as ill but not an ill person as healthy.

e) F1 Score

- F1 Score is the harmonic mean of Precision and Recall, it combines to balance the trade-off between the two metrics. It has the highest possible value of 1, which denotes perfect precision and recall.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (4)$$

- F1 is a complex metric that can provide more accurate results in cases of imbalanced classification problems.

f) Area Under the Receiver Operating Characteristic Curve (AUC-ROC)

- The receiver operating characteristic curve displays the ability of the classifier to correctly identify positive cases compared to its tendency to incorrectly classify negative cases.
- In essence, it demonstrates the increase in correctly classified positive instances while allowing for an increased number of false positives.
- AUC for ROC plots is a quick way to compare with each other, the ideal value is 1.

g) Training/Validation Accuracy and Loss Plots

- Visualizations of the model accuracy and loss during the training on train and validation data can be crucial for assessing the effectiveness of the deep learning models.
- These plots offer valuable information about the model's ability to learn, as well as indications of potential overfitting or underfitting issues as the model goes through training epochs.
- Ideally, we want the model accuracy to go up as we train, and the loss to go down. If we can get these trends for both the training and validation data, we know the model is training well.

3.11 Models

This section covers all the deep learning models developed during this research study to help in the assessment of knee ligament injuries. After pre-processing and creating more samples through data augmentation for both datasets and setting up the prerequisites for training our 3D CNNs, we moved to designing and training the models based on these samples.

We developed all our model architectures using the TensorFlow Keras APIs. Overall, we developed models for the following two tasks:

- a) Identifying whether an ACL tear is present or not by utilizing the MRNet samples which have binary labels for this classification problem.
- b) Further, we want to assess the extent of damage to the ACL in case a tear is present, for this, we trained models using the KneeMRI data which has labels to identify healthy ligament, partial tear of ligament, or complete rupture.

We developed all our model architectures using the TensorFlow Keras APIs. Overall, we developed models using both datasets, these included model architectures trained from scratch as well as models created using the concept of transfer learning with the hybrid concept of using multiple channels.

The models were developed to train and learn the following two tasks:

- a) Binary classification: We used the MRNet dataset to train models for identifying if an ACL tear is present or not.
- b) Multi-class classification: We used the KneeMRI dataset to train models to identify the grade of an ACL tear if present.

All the trained models have been listed below along with the evaluation metrics, out of these the best-performing models will be integrated with the graphical user interface (GUI):

- a) Training models from scratch
 - In total, we trained six different 3D CNN architectures. The same model architecture will be used to train models on the two datasets, except for the final output layer depending on the classification task at hand.
 - The first three models are a combination of Conv3D, MaxPooling3D, and BatchNormalisation layers. These convolutional blocks are finally followed by a GlobalAveragePooling3D, some Dropout layers in between the Dense layers.
 - The depth of the three models follows an ascending order to increase the complexity and ability of the model to learn more from the samples.
 - Another couple of models, comprising typical Conv3D and MaxPooling3D convolutional blocks. These are followed by a Flatten, some Dropout layers amongst the Dense classification layers. Both models have an increasing order of depth in the architecture.
 - Lastly, the final model developed was a mix of both types of architecture, typical convolutional blocks of Conv3D and MaxPooling3D with BatchNormalisation layers in between some of the blocks. These are followed by a Flatten, with some Dropout layers in between the Dense layers.

- Model1

Model: "MRNet_Model1"			Model: "kneeMRI_Model1"		
Layer (type)	Output Shape	Param #	Layer (type)	Output Shape	Param #
conv3d (Conv3D)	(None, 30, 256, 256, 64)	1792	conv3d (Conv3D)	(None, 30, 256, 256, 64)	1792
max_pooling3d (MaxPooling3D)	(None, 15, 128, 128, 64)	0	max_pooling3d (MaxPooling3D)	(None, 15, 128, 128, 64)	0
batch_normalization (BatchN ormalization)	(None, 15, 128, 128, 64)	256	batch_normalization (BatchN ormalization)	(None, 15, 128, 128, 64)	256
conv3d_1 (Conv3D)	(None, 15, 128, 128, 64)	110656	conv3d_1 (Conv3D)	(None, 15, 128, 128, 64)	110656
max_pooling3d_1 (MaxPooling 3D)	(None, 8, 64, 64, 64)	0	max_pooling3d_1 (MaxPooling 3D)	(None, 8, 64, 64, 64)	0
batch_normalization_1 (Batch hNormalization)	(None, 8, 64, 64, 64)	256	batch_normalization_1 (Batch hNormalization)	(None, 8, 64, 64, 64)	256
conv3d_2 (Conv3D)	(None, 8, 64, 64, 128)	221312	conv3d_2 (Conv3D)	(None, 8, 64, 64, 128)	221312
max_pooling3d_2 (MaxPooling 3D)	(None, 4, 32, 32, 128)	0	max_pooling3d_2 (MaxPooling 3D)	(None, 4, 32, 32, 128)	0
batch_normalization_2 (Batch hNormalization)	(None, 4, 32, 32, 128)	512	batch_normalization_2 (Batch hNormalization)	(None, 4, 32, 32, 128)	512
conv3d_3 (Conv3D)	(None, 4, 32, 32, 128)	442496	conv3d_3 (Conv3D)	(None, 4, 32, 32, 128)	442496
max_pooling3d_3 (MaxPooling 3D)	(None, 2, 16, 16, 128)	0	max_pooling3d_3 (MaxPooling 3D)	(None, 2, 16, 16, 128)	0
batch_normalization_3 (Batch hNormalization)	(None, 2, 16, 16, 128)	512	batch_normalization_3 (Batch hNormalization)	(None, 2, 16, 16, 128)	512
conv3d_4 (Conv3D)	(None, 2, 16, 16, 256)	884992	conv3d_4 (Conv3D)	(None, 2, 16, 16, 256)	884992
max_pooling3d_4 (MaxPooling 3D)	(None, 1, 8, 8, 256)	0	max_pooling3d_4 (MaxPooling 3D)	(None, 1, 8, 8, 256)	0
batch_normalization_4 (Batch hNormalization)	(None, 1, 8, 8, 256)	1024	batch_normalization_4 (Batch hNormalization)	(None, 1, 8, 8, 256)	1024
global_average_pooling3d (G lobalAveragePooling3D)	(None, 256)	0	global_average_pooling3d (G lobalAveragePooling3D)	(None, 256)	0
dense (Dense)	(None, 512)	131584	dense (Dense)	(None, 512)	131584
dropout (Dropout)	(None, 512)	0	dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328	dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0	dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 1)	257	dense_2 (Dense)	(None, 3)	771
<hr/>					
Total params:	1,926,977		Total params:	1,927,491	
Trainable params:	1,925,697		Trainable params:	1,926,211	
Non-trainable params:	1,280		Non-trainable params:	1,280	

Figure 34 : Model1 for MRNet and KneeMRI

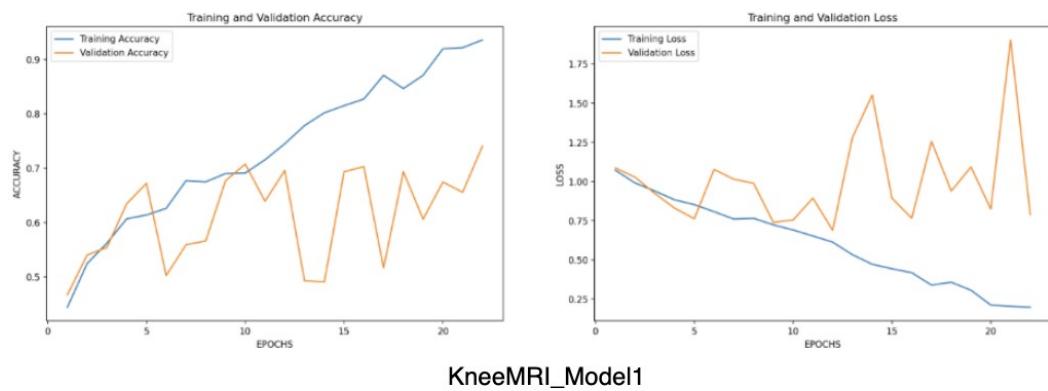
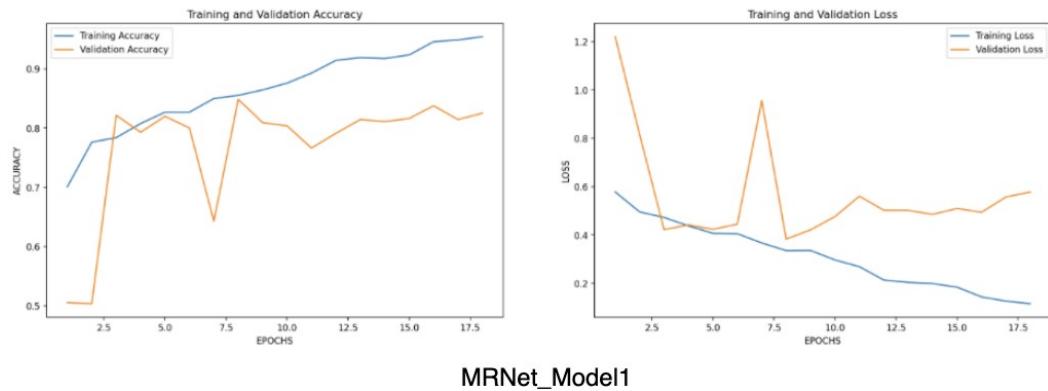


Figure 35 : Model1 Training for MRNet and KneeMRI

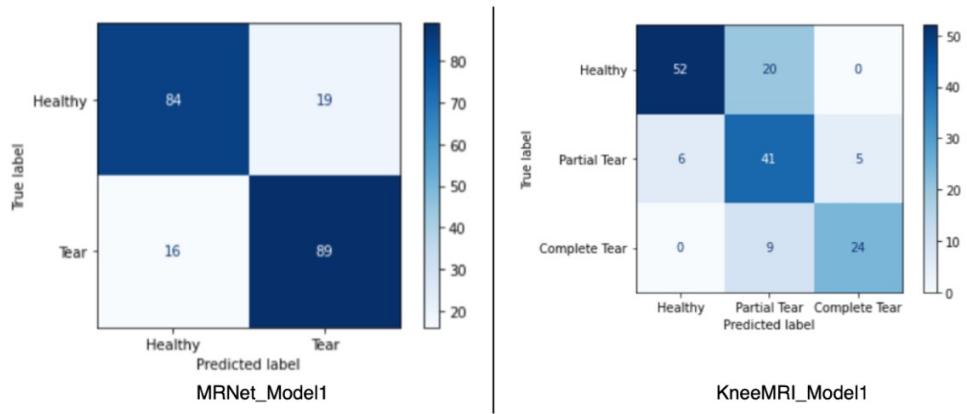


Figure 36 : Model1 Confusion Matrices for MRNet and KneeMRI

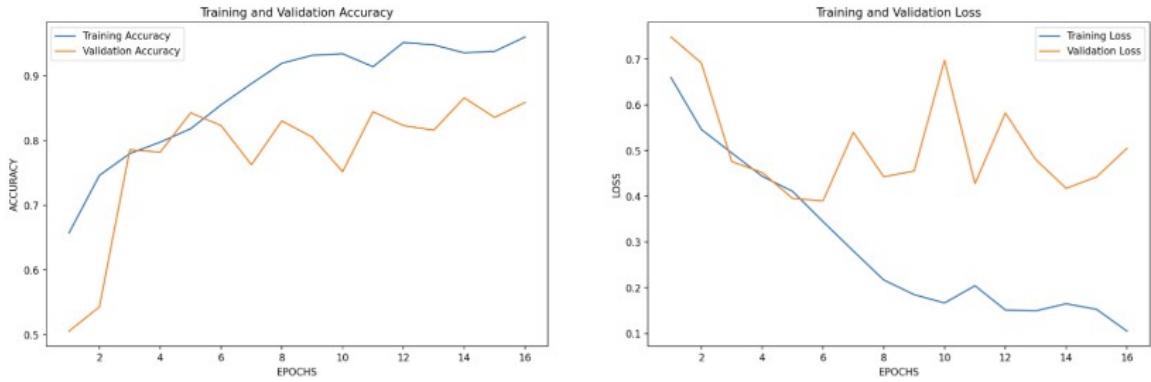
Model1 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 8 for MRNet and epoch 12 for KneeMRI.

■ Model2

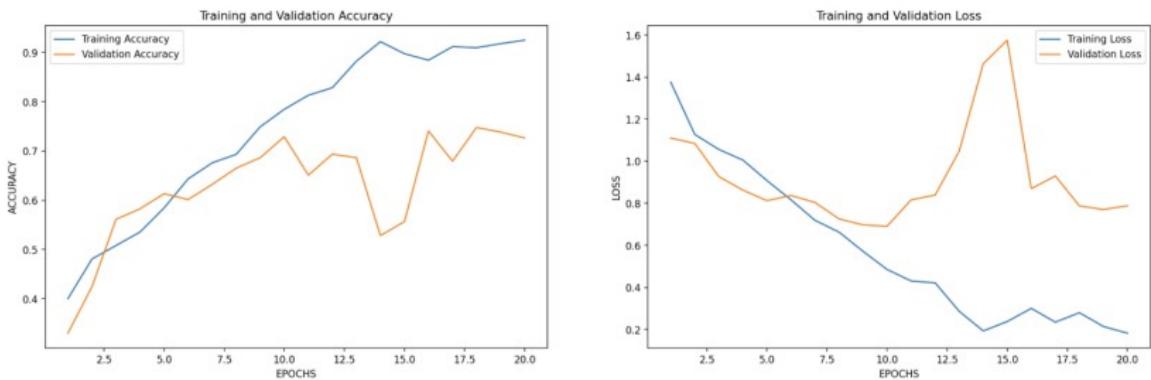
Model: "MRNet_Model2"		
Layer (type)	Output Shape	Param #
conv3d_5 (Conv3D)	(None, 30, 256, 256, 16)	448
max_pooling3d_5 (MaxPooling 3D)	(None, 15, 128, 128, 16)	0
batch_normalization_5 (Batch Normalization)	(None, 15, 128, 128, 16)	64
conv3d_6 (Conv3D)	(None, 15, 128, 128, 32)	13856
max_pooling3d_6 (MaxPooling 3D)	(None, 8, 64, 64, 32)	0
batch_normalization_6 (Batch Normalization)	(None, 8, 64, 64, 32)	128
conv3d_7 (Conv3D)	(None, 8, 64, 64, 32)	27680
max_pooling3d_7 (MaxPooling 3D)	(None, 4, 32, 32, 32)	0
batch_normalization_7 (Batch Normalization)	(None, 4, 32, 32, 32)	128
conv3d_8 (Conv3D)	(None, 4, 32, 32, 64)	55360
max_pooling3d_8 (MaxPooling 3D)	(None, 2, 16, 16, 64)	0
batch_normalization_8 (Batch Normalization)	(None, 2, 16, 16, 64)	256
conv3d_9 (Conv3D)	(None, 2, 16, 16, 64)	110656
max_pooling3d_9 (MaxPooling 3D)	(None, 1, 8, 8, 64)	0
batch_normalization_9 (Batch Normalization)	(None, 1, 8, 8, 64)	256
conv3d_10 (Conv3D)	(None, 1, 8, 8, 128)	221312
max_pooling3d_10 (MaxPoolin g3D)	(None, 1, 4, 4, 128)	0
batch_normalization_10 (Batch Normalization)	(None, 1, 4, 4, 128)	512
conv3d_11 (Conv3D)	(None, 1, 4, 4, 128)	442496
max_pooling3d_11 (MaxPoolin g3D)	(None, 1, 2, 2, 128)	0
batch_normalization_11 (Batch Normalization)	(None, 1, 2, 2, 128)	512
conv3d_12 (Conv3D)	(None, 1, 2, 2, 256)	884992
max_pooling3d_12 (MaxPoolin g3D)	(None, 1, 1, 1, 256)	0
batch_normalization_12 (Batch Normalization)	(None, 1, 1, 1, 256)	1024
global_average_pooling3d_1 (GlobalAveragePooling3D)	(None, 256)	0
dense_3 (Dense)	(None, 512)	131584
dropout_2 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131328
dropout_3 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 1)	257
<hr/>		
Total params: 2,022,849		
Trainable params: 2,021,489		
Non-trainable params: 1,440		

Model: "kneeMRI_Model2"		
Layer (type)	Output Shape	Param #
conv3d_5 (Conv3D)	(None, 30, 256, 256, 16)	448
max_pooling3d_5 (MaxPooling 3D)	(None, 15, 128, 128, 16)	0
batch_normalization_5 (Batch Normalization)	(None, 15, 128, 128, 16)	64
conv3d_6 (Conv3D)	(None, 15, 128, 128, 32)	13856
max_pooling3d_6 (MaxPooling 3D)	(None, 8, 64, 64, 32)	0
batch_normalization_6 (Batch Normalization)	(None, 8, 64, 64, 32)	128
conv3d_7 (Conv3D)	(None, 8, 64, 64, 32)	27680
max_pooling3d_7 (MaxPooling 3D)	(None, 4, 32, 32, 32)	0
batch_normalization_7 (Batch Normalization)	(None, 4, 32, 32, 32)	128
conv3d_8 (Conv3D)	(None, 4, 32, 32, 64)	55360
max_pooling3d_8 (MaxPooling 3D)	(None, 2, 16, 16, 64)	0
batch_normalization_8 (Batch Normalization)	(None, 2, 16, 16, 64)	256
conv3d_9 (Conv3D)	(None, 2, 16, 16, 64)	110656
max_pooling3d_9 (MaxPooling 3D)	(None, 1, 8, 8, 64)	0
batch_normalization_9 (Batch Normalization)	(None, 1, 8, 8, 64)	256
conv3d_10 (Conv3D)	(None, 1, 8, 8, 128)	221312
max_pooling3d_10 (MaxPoolin g3D)	(None, 1, 4, 4, 128)	0
batch_normalization_10 (Batch Normalization)	(None, 1, 4, 4, 128)	512
conv3d_11 (Conv3D)	(None, 1, 4, 4, 128)	442496
max_pooling3d_11 (MaxPoolin g3D)	(None, 1, 2, 2, 128)	0
batch_normalization_11 (Batch Normalization)	(None, 1, 2, 2, 128)	512
conv3d_12 (Conv3D)	(None, 1, 2, 2, 256)	884992
max_pooling3d_12 (MaxPoolin g3D)	(None, 1, 1, 1, 256)	0
batch_normalization_12 (Batch Normalization)	(None, 1, 1, 1, 256)	1024
global_average_pooling3d_1 (GlobalAveragePooling3D)	(None, 256)	0
dense_3 (Dense)	(None, 512)	131584
dropout_2 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131328
dropout_3 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 3)	771
<hr/>		
Total params: 2,023,363		
Trainable params: 2,021,923		
Non-trainable params: 1,440		

Figure 37 : Model2 for MRNet and KneeMRI



MRNet_Model2



KneeMRI_Model2

Figure 38 : Model2 Training for MRNet and KneeMRI

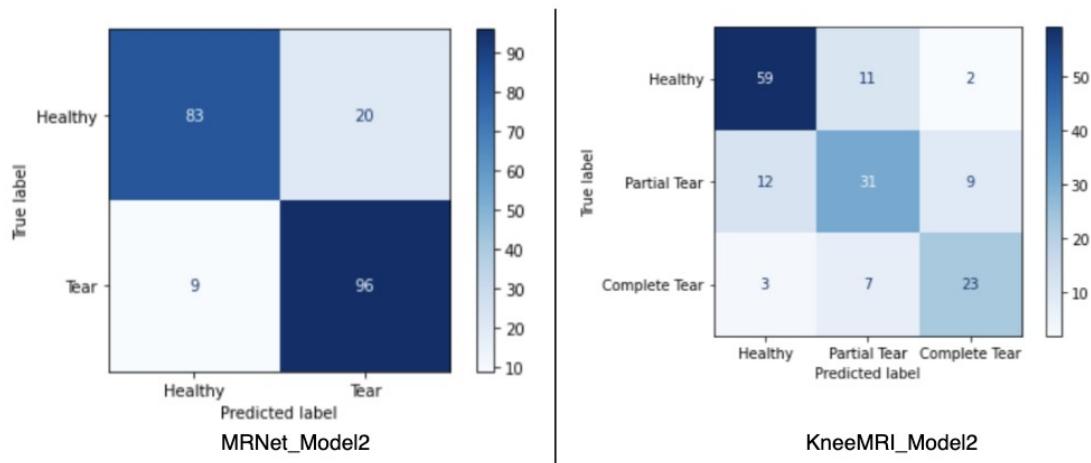


Figure 39 : Model2 Confusion Matrices for MRNet and KneeMRI

Model2 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 6 for MRNet and epoch 10 for KneeMRI.

- Model3

Model: "MRNet_Model3"		
Layer (type)	Output Shape	Param #
conv3d_13 (Conv3D)	(None, 30, 256, 256, 32)	896
max_pooling3d_13 (MaxPoolin g3D)	(None, 15, 128, 128, 32)	0
batch_normalization_13 (BatchNormalization)	(None, 15, 128, 128, 32)	128
conv3d_14 (Conv3D)	(None, 15, 128, 128, 32)	27680
max_pooling3d_14 (MaxPoolin g3D)	(None, 8, 64, 64, 32)	0
batch_normalization_14 (BatchNormalization)	(None, 8, 64, 64, 32)	128
conv3d_15 (Conv3D)	(None, 8, 64, 64, 32)	27680
max_pooling3d_15 (MaxPoolin g3D)	(None, 4, 32, 32, 32)	0
batch_normalization_15 (BatchNormalization)	(None, 4, 32, 32, 32)	128
conv3d_16 (Conv3D)	(None, 4, 32, 32, 64)	55360
max_pooling3d_16 (MaxPoolin g3D)	(None, 2, 16, 16, 64)	0
batch_normalization_16 (BatchNormalization)	(None, 2, 16, 16, 64)	256
conv3d_17 (Conv3D)	(None, 2, 16, 16, 64)	110656
max_pooling3d_17 (MaxPoolin g3D)	(None, 1, 8, 8, 64)	0
batch_normalization_17 (BatchNormalization)	(None, 1, 8, 8, 64)	256
conv3d_18 (Conv3D)	(None, 1, 8, 8, 64)	110656
max_pooling3d_18 (MaxPoolin g3D)	(None, 1, 4, 4, 64)	0
batch_normalization_18 (BatchNormalization)	(None, 1, 4, 4, 64)	256
conv3d_19 (Conv3D)	(None, 1, 4, 4, 128)	221312
max_pooling3d_19 (MaxPoolin g3D)	(None, 1, 2, 2, 128)	0
batch_normalization_19 (BatchNormalization)	(None, 1, 2, 2, 128)	512
conv3d_20 (Conv3D)	(None, 1, 2, 2, 128)	442496
max_pooling3d_20 (MaxPoolin g3D)	(None, 1, 1, 1, 128)	0
batch_normalization_20 (BatchNormalization)	(None, 1, 1, 1, 128)	512
conv3d_21 (Conv3D)	(None, 1, 1, 1, 128)	442496
max_pooling3d_21 (MaxPoolin g3D)	(None, 1, 1, 1, 128)	0
batch_normalization_21 (BatchNormalization)	(None, 1, 1, 1, 128)	512
conv3d_22 (Conv3D)	(None, 1, 1, 1, 256)	884992
max_pooling3d_22 (MaxPoolin g3D)	(None, 1, 1, 1, 256)	0
batch_normalization_22 (BatchNormalization)	(None, 1, 1, 1, 256)	1024
conv3d_23 (Conv3D)	(None, 1, 1, 1, 256)	1769728
max_pooling3d_23 (MaxPoolin g3D)	(None, 1, 1, 1, 256)	0
batch_normalization_23 (BatchNormalization)	(None, 1, 1, 1, 256)	1024
global_average_pooling3d_2 (GlobalAveragePooling3D)	(None, 256)	0
dense_6 (Dense)	(None, 512)	131584
dropout_4 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 256)	131328
dropout_6 (Dropout)	(None, 256)	0
dense_9 (Dense)	(None, 1)	257

Model: "kneeMRI_Model3"		
Layer (type)	Output Shape	Param #
conv3d_13 (Conv3D)	(None, 30, 256, 256, 32)	896
max_pooling3d_13 (MaxPoolin g3D)	(None, 15, 128, 128, 32)	0
batch_normalization_13 (BatchNormalization)	(None, 15, 128, 128, 32)	128
conv3d_14 (Conv3D)	(None, 15, 128, 128, 32)	27680
max_pooling3d_14 (MaxPoolin g3D)	(None, 8, 64, 64, 32)	0
batch_normalization_14 (BatchNormalization)	(None, 8, 64, 64, 32)	128
conv3d_15 (Conv3D)	(None, 8, 64, 64, 32)	27680
max_pooling3d_15 (MaxPoolin g3D)	(None, 4, 32, 32, 32)	0
batch_normalization_15 (BatchNormalization)	(None, 4, 32, 32, 32)	128
conv3d_16 (Conv3D)	(None, 4, 32, 32, 64)	55360
max_pooling3d_16 (MaxPoolin g3D)	(None, 2, 16, 16, 64)	0
batch_normalization_16 (BatchNormalization)	(None, 2, 16, 16, 64)	256
conv3d_17 (Conv3D)	(None, 2, 16, 16, 64)	110656
max_pooling3d_17 (MaxPoolin g3D)	(None, 1, 8, 8, 64)	0
batch_normalization_17 (BatchNormalization)	(None, 1, 8, 8, 64)	256
conv3d_18 (Conv3D)	(None, 1, 8, 8, 64)	110656
max_pooling3d_18 (MaxPoolin g3D)	(None, 1, 4, 4, 64)	0
batch_normalization_18 (BatchNormalization)	(None, 1, 4, 4, 64)	256
conv3d_19 (Conv3D)	(None, 1, 4, 4, 128)	221312
max_pooling3d_19 (MaxPoolin g3D)	(None, 1, 2, 2, 128)	0
batch_normalization_19 (BatchNormalization)	(None, 1, 2, 2, 128)	512
conv3d_20 (Conv3D)	(None, 1, 2, 2, 128)	442496
max_pooling3d_20 (MaxPoolin g3D)	(None, 1, 1, 1, 128)	0
batch_normalization_20 (BatchNormalization)	(None, 1, 1, 1, 128)	512
conv3d_21 (Conv3D)	(None, 1, 1, 1, 128)	442496
max_pooling3d_21 (MaxPoolin g3D)	(None, 1, 1, 1, 128)	0
batch_normalization_21 (BatchNormalization)	(None, 1, 1, 1, 128)	512
conv3d_22 (Conv3D)	(None, 1, 1, 1, 256)	884992
max_pooling3d_22 (MaxPoolin g3D)	(None, 1, 1, 1, 256)	0
batch_normalization_22 (BatchNormalization)	(None, 1, 1, 1, 256)	1024
conv3d_23 (Conv3D)	(None, 1, 1, 1, 256)	1769728
max_pooling3d_23 (MaxPoolin g3D)	(None, 1, 1, 1, 256)	0
batch_normalization_23 (BatchNormalization)	(None, 1, 1, 1, 256)	1024
global_average_pooling3d_2 (GlobalAveragePooling3D)	(None, 256)	0
dense_6 (Dense)	(None, 512)	131584
dropout_4 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 256)	131328
dropout_6 (Dropout)	(None, 256)	0
dense_9 (Dense)	(None, 3)	771

Figure 40 : Model3 for MRNet and KneeMRI

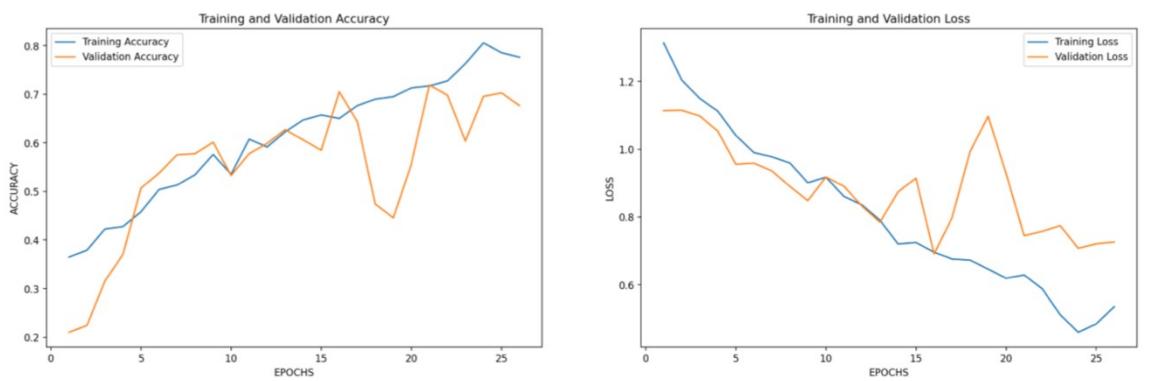
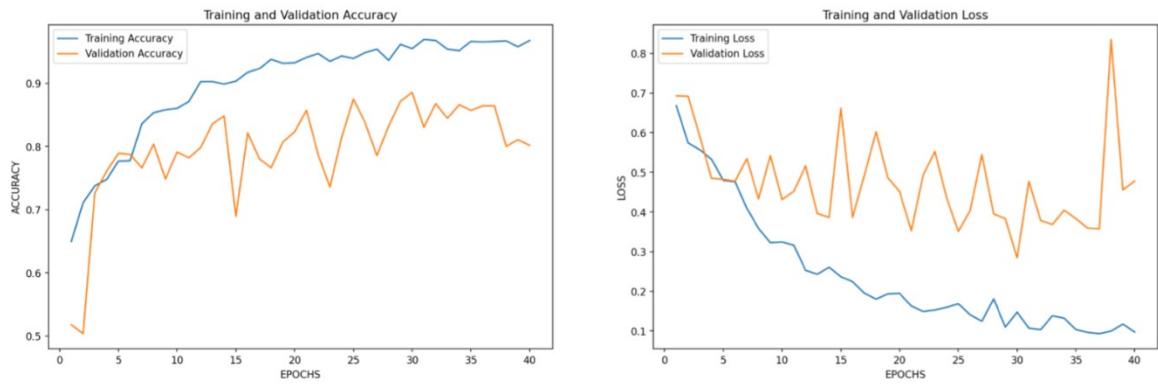


Figure 41 : Model3 Training for MRNet and KneeMRI

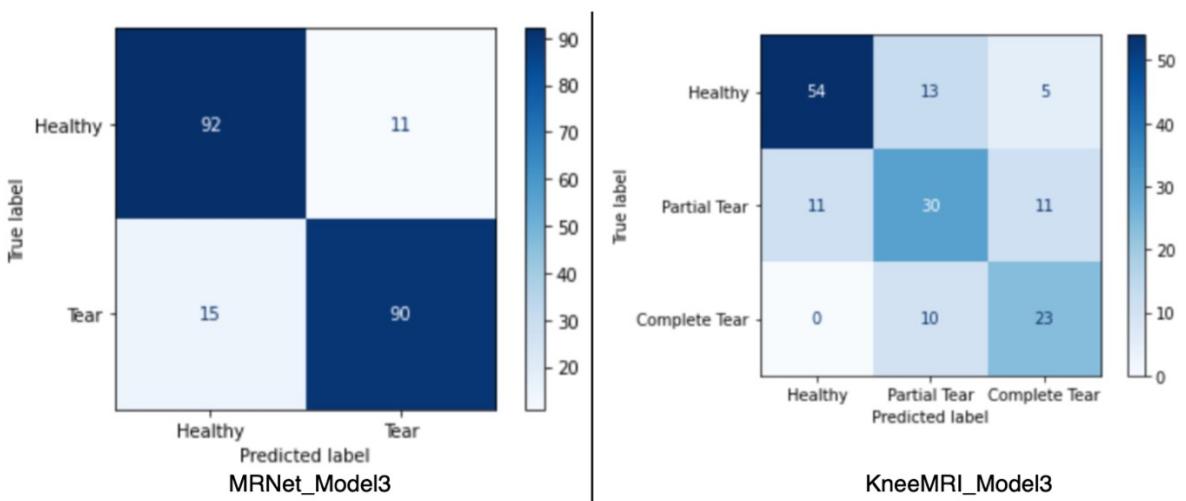


Figure 42 : Model3 Confusion Matrices for MRNet and KneeMRI

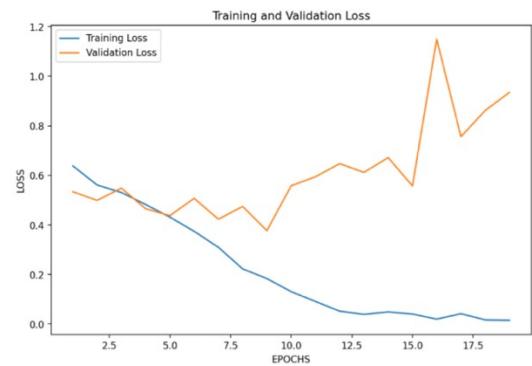
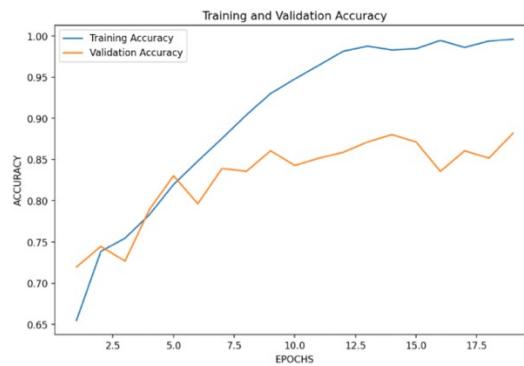
Model2 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 30 for MRNet and epoch 16 for KneeMRI.

- Model4

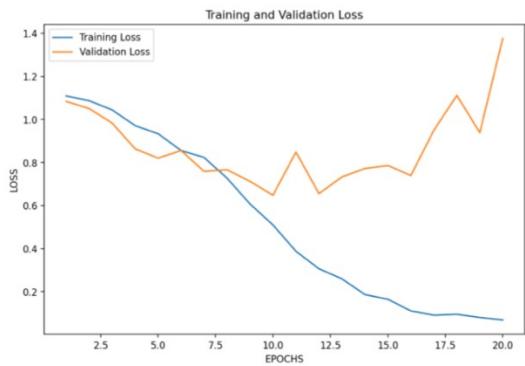
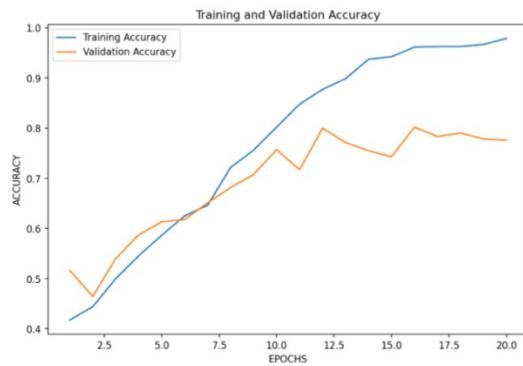
Model: "MRNet_Model4"			
Layer (type)	Output Shape	Param #	
conv3d_24 (Conv3D)	(None, 30, 256, 256, 32)	896	
conv3d_25 (Conv3D)	(None, 30, 256, 256, 32)	27680	
max_pooling3d_24 (MaxPoolin g3D)	(None, 15, 128, 128, 32)	0	
conv3d_26 (Conv3D)	(None, 15, 128, 128, 32)	27680	
conv3d_27 (Conv3D)	(None, 15, 128, 128, 64)	55360	
max_pooling3d_25 (MaxPoolin g3D)	(None, 8, 64, 64, 64)	0	
conv3d_28 (Conv3D)	(None, 8, 64, 64, 64)	110656	
conv3d_29 (Conv3D)	(None, 8, 64, 64, 128)	221312	
max_pooling3d_26 (MaxPoolin g3D)	(None, 4, 32, 32, 128)	0	
dropout_7 (Dropout)	(None, 4, 32, 32, 128)	0	
flatten (Flatten)	(None, 524288)	0	
dense_10 (Dense)	(None, 256)	134217984	
dropout_8 (Dropout)	(None, 256)	0	
dense_11 (Dense)	(None, 128)	32896	
dropout_9 (Dropout)	(None, 128)	0	
dense_12 (Dense)	(None, 1)	129	
<hr/>			
Total params: 134,694,593			
Trainable params: 134,694,593			
Non-trainable params: 0			

Model: "kneeMRI_Model4"			
Layer (type)	Output Shape	Param #	
conv3d_24 (Conv3D)	(None, 30, 256, 256, 32)	896	
conv3d_25 (Conv3D)	(None, 30, 256, 256, 32)	27680	
max_pooling3d_24 (MaxPoolin g3D)	(None, 15, 128, 128, 32)	0	
conv3d_26 (Conv3D)	(None, 15, 128, 128, 32)	27680	
conv3d_27 (Conv3D)	(None, 15, 128, 128, 64)	55360	
max_pooling3d_25 (MaxPoolin g3D)	(None, 8, 64, 64, 64)	0	
conv3d_28 (Conv3D)	(None, 8, 64, 64, 64)	110656	
conv3d_29 (Conv3D)	(None, 8, 64, 64, 128)	221312	
max_pooling3d_26 (MaxPoolin g3D)	(None, 4, 32, 32, 128)	0	
dropout_7 (Dropout)	(None, 4, 32, 32, 128)	0	
flatten (Flatten)	(None, 524288)	0	
dense_10 (Dense)	(None, 256)	134217984	
dropout_8 (Dropout)	(None, 256)	0	
dense_11 (Dense)	(None, 128)	32896	
dropout_9 (Dropout)	(None, 128)	0	
dense_12 (Dense)	(None, 3)	387	
<hr/>			
Total params: 134,694,851			
Trainable params: 134,694,851			
Non-trainable params: 0			

Figure 43 : Model4 for MRNet and KneeMRI



MRNet_Model4



KneeMRI_Model4

Figure 44 : Model4 Training for MRNet and KneeMRI

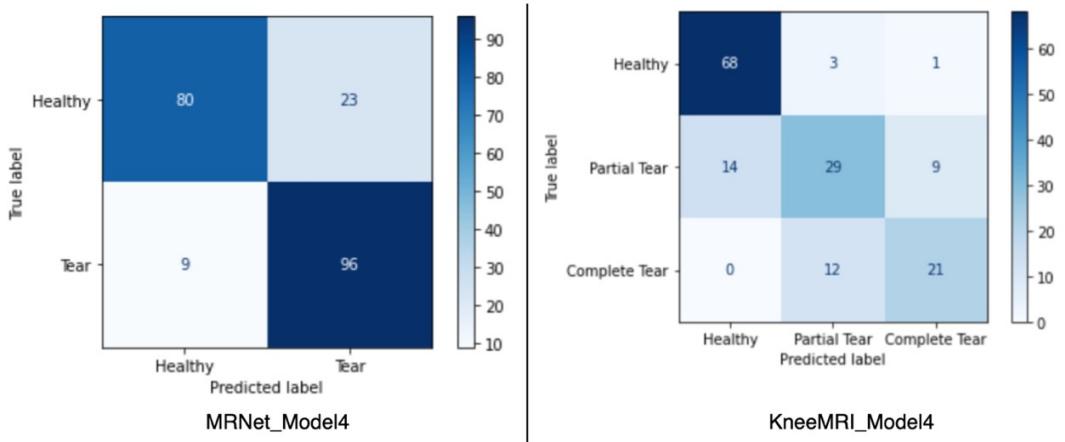


Figure 45 : Model4 Confusion Matrices for MRNet and KneeMRI

Model4 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 9 for MRNet and epoch 10 for KneeMRI.

- Model5

Model: "MRNet_Model5"

Layer (type)	Output Shape	Param #
<hr/>		
conv3d_30 (Conv3D)	(None, 30, 256, 256, 16)	448
max_pooling3d_27 (MaxPoolin g3D)	(None, 15, 128, 128, 16)	0
conv3d_31 (Conv3D)	(None, 15, 128, 128, 16)	6928
max_pooling3d_28 (MaxPoolin g3D)	(None, 8, 64, 64, 16)	0
conv3d_32 (Conv3D)	(None, 8, 64, 64, 32)	13856
max_pooling3d_29 (MaxPoolin g3D)	(None, 4, 32, 32, 32)	0
conv3d_33 (Conv3D)	(None, 4, 32, 32, 32)	27680
max_pooling3d_30 (MaxPoolin g3D)	(None, 2, 16, 16, 32)	0
conv3d_34 (Conv3D)	(None, 2, 16, 16, 32)	27680
max_pooling3d_31 (MaxPoolin g3D)	(None, 1, 8, 8, 32)	0
conv3d_35 (Conv3D)	(None, 1, 8, 8, 64)	55360
max_pooling3d_32 (MaxPoolin g3D)	(None, 1, 4, 4, 64)	0
conv3d_36 (Conv3D)	(None, 1, 4, 4, 64)	110656
max_pooling3d_33 (MaxPoolin g3D)	(None, 1, 2, 2, 64)	0
conv3d_37 (Conv3D)	(None, 1, 2, 2, 128)	221312
max_pooling3d_34 (MaxPoolin g3D)	(None, 1, 1, 1, 128)	0
conv3d_38 (Conv3D)	(None, 1, 1, 1, 128)	442496
max_pooling3d_35 (MaxPoolin g3D)	(None, 1, 1, 1, 128)	0
conv3d_39 (Conv3D)	(None, 1, 1, 1, 256)	884992
max_pooling3d_36 (MaxPoolin g3D)	(None, 1, 1, 1, 256)	0
dropout_10 (Dropout)	(None, 1, 1, 1, 256)	0
flatten_1 (Flatten)	(None, 256)	0
dense_13 (Dense)	(None, 512)	131584
dropout_11 (Dropout)	(None, 512)	0
dense_14 (Dense)	(None, 256)	131328
dropout_12 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 128)	32896
dropout_13 (Dropout)	(None, 128)	0
dense_16 (Dense)	(None, 1)	129
<hr/>		

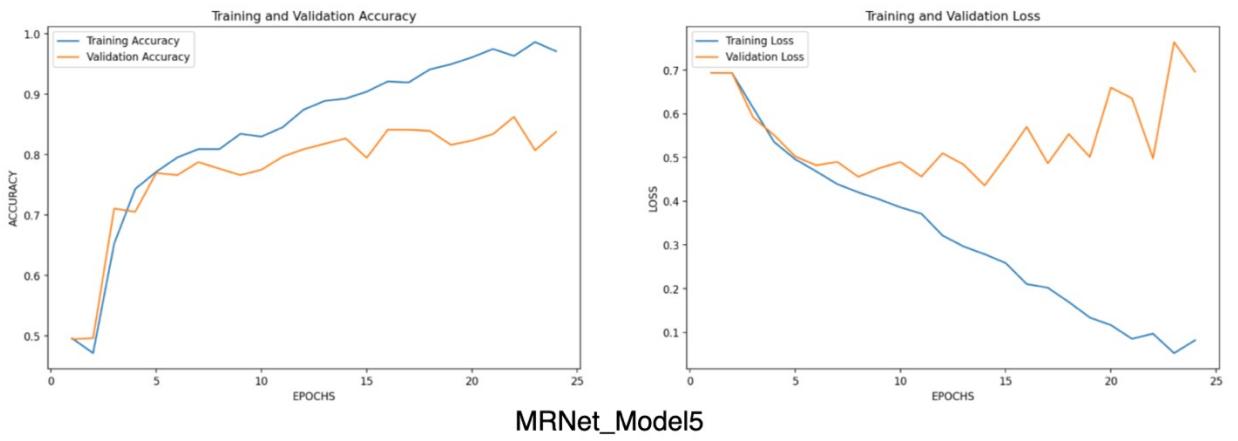
Total params: 2,087,345
Trainable params: 2,087,345
Non-trainable params: 0

Model: "kneeMRI_Model5"

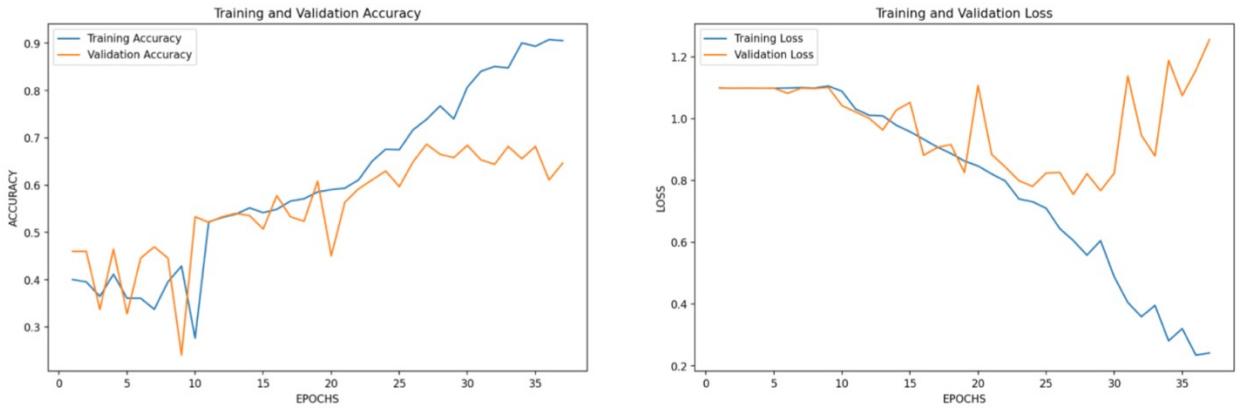
Layer (type)	Output Shape	Param #
<hr/>		
conv3d_30 (Conv3D)	(None, 30, 256, 256, 16)	448
max_pooling3d_27 (MaxPoolin g3D)	(None, 15, 128, 128, 16)	0
conv3d_31 (Conv3D)	(None, 15, 128, 128, 16)	6928
max_pooling3d_28 (MaxPoolin g3D)	(None, 8, 64, 64, 16)	0
conv3d_32 (Conv3D)	(None, 8, 64, 64, 32)	13856
max_pooling3d_29 (MaxPoolin g3D)	(None, 4, 32, 32, 32)	0
conv3d_33 (Conv3D)	(None, 4, 32, 32, 32)	27680
max_pooling3d_30 (MaxPoolin g3D)	(None, 2, 16, 16, 32)	0
conv3d_34 (Conv3D)	(None, 2, 16, 16, 32)	27680
max_pooling3d_31 (MaxPoolin g3D)	(None, 1, 8, 8, 32)	0
conv3d_35 (Conv3D)	(None, 1, 8, 8, 64)	55360
max_pooling3d_32 (MaxPoolin g3D)	(None, 1, 4, 4, 64)	0
conv3d_36 (Conv3D)	(None, 1, 4, 4, 64)	110656
max_pooling3d_33 (MaxPoolin g3D)	(None, 1, 2, 2, 64)	0
conv3d_37 (Conv3D)	(None, 1, 2, 2, 128)	221312
max_pooling3d_34 (MaxPoolin g3D)	(None, 1, 1, 1, 128)	0
conv3d_38 (Conv3D)	(None, 1, 1, 1, 128)	442496
max_pooling3d_35 (MaxPoolin g3D)	(None, 1, 1, 1, 128)	0
conv3d_39 (Conv3D)	(None, 1, 1, 1, 256)	884992
max_pooling3d_36 (MaxPoolin g3D)	(None, 1, 1, 1, 256)	0
dropout_10 (Dropout)	(None, 1, 1, 1, 256)	0
flatten_1 (Flatten)	(None, 256)	0
dense_13 (Dense)	(None, 512)	131584
dropout_11 (Dropout)	(None, 512)	0
dense_14 (Dense)	(None, 256)	131328
dropout_12 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 128)	32896
dropout_13 (Dropout)	(None, 128)	0
dense_16 (Dense)	(None, 3)	387
<hr/>		

Total params: 2,087,603
Trainable params: 2,087,603
Non-trainable params: 0

Figure 46 : Model5 for MRNet and KneeMRI



MRNet_Model5



KneeMRI_Model5

Figure 47 : Model5 Training for MRNet and KneeMRI

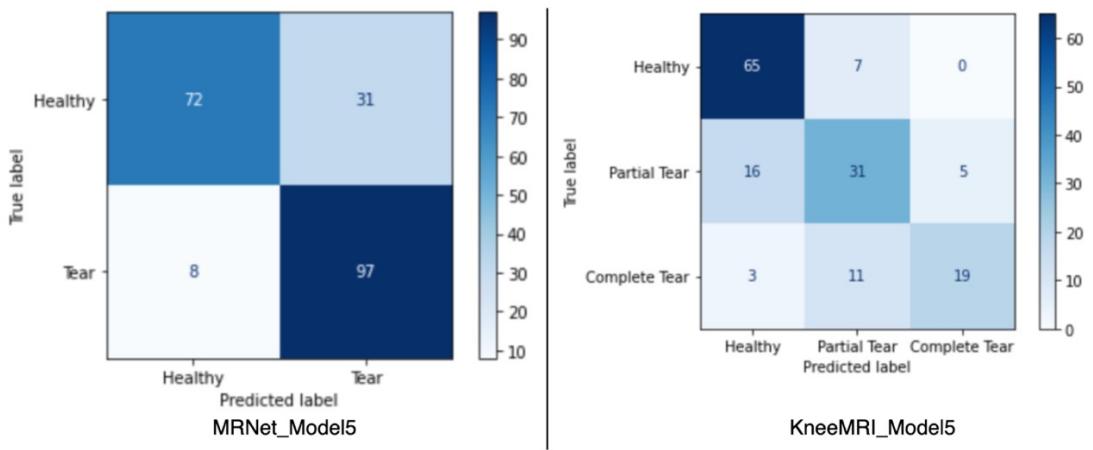


Figure 48 : Model5 Confusion Matrices for MRNet and KneeMRI

Model5 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 14 for MRNet and epoch 27 for KneeMRI.

- Model6

Model: "MRNet_Model6"

Layer (type)	Output Shape	Param #
conv3d (Conv3D)	(None, 30, 256, 256, 8)	224
conv3d_1 (Conv3D)	(None, 30, 256, 256, 16)	3472
max_pooling3d (MaxPooling3D)	(None, 15, 128, 128, 16)	0
conv3d_2 (Conv3D)	(None, 15, 128, 128, 16)	6928
conv3d_3 (Conv3D)	(None, 15, 128, 128, 16)	6928
max_pooling3d_1 (MaxPooling3D)	(None, 8, 64, 64, 16)	0
conv3d_4 (Conv3D)	(None, 8, 64, 64, 16)	6928
conv3d_5 (Conv3D)	(None, 8, 64, 64, 16)	6928
max_pooling3d_2 (MaxPooling3D)	(None, 4, 32, 32, 16)	0
batch_normalization (BatchNormalization)	(None, 4, 32, 32, 16)	64
conv3d_6 (Conv3D)	(None, 4, 32, 32, 32)	13856
conv3d_7 (Conv3D)	(None, 4, 32, 32, 32)	27680
max_pooling3d_3 (MaxPooling3D)	(None, 2, 16, 16, 32)	0
conv3d_8 (Conv3D)	(None, 2, 16, 16, 32)	27680
max_pooling3d_4 (MaxPooling3D)	(None, 1, 8, 8, 32)	0
batch_normalization_1 (BatchNormalization)	(None, 1, 8, 8, 32)	128
conv3d_9 (Conv3D)	(None, 1, 8, 8, 64)	55360
max_pooling3d_5 (MaxPooling3D)	(None, 1, 4, 4, 64)	0
conv3d_10 (Conv3D)	(None, 1, 4, 4, 64)	110656
max_pooling3d_6 (MaxPooling3D)	(None, 1, 2, 2, 64)	0
conv3d_11 (Conv3D)	(None, 1, 2, 2, 64)	110656
max_pooling3d_7 (MaxPooling3D)	(None, 1, 1, 1, 64)	0
batch_normalization_2 (BatchNormalization)	(None, 1, 1, 1, 64)	256
conv3d_12 (Conv3D)	(None, 1, 1, 1, 128)	221312
max_pooling3d_8 (MaxPooling3D)	(None, 1, 1, 1, 128)	0
conv3d_13 (Conv3D)	(None, 1, 1, 1, 128)	442496
max_pooling3d_9 (MaxPooling3D)	(None, 1, 1, 1, 128)	0
conv3d_14 (Conv3D)	(None, 1, 1, 1, 128)	442496
max_pooling3d_10 (MaxPooling3D)	(None, 1, 1, 1, 128)	0
batch_normalization_3 (BatchNormalization)	(None, 1, 1, 1, 128)	512
conv3d_15 (Conv3D)	(None, 1, 1, 1, 256)	884992
max_pooling3d_11 (MaxPooling3D)	(None, 1, 1, 1, 256)	0
conv3d_16 (Conv3D)	(None, 1, 1, 1, 256)	1769728
max_pooling3d_12 (MaxPooling3D)	(None, 1, 1, 1, 256)	0
conv3d_17 (Conv3D)	(None, 1, 1, 1, 256)	1769728
max_pooling3d_13 (MaxPooling3D)	(None, 1, 1, 1, 256)	0
dropout (Dropout)	(None, 1, 1, 1, 256)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 512)	131584
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129

Model: "kneeMRI_Model6"

Layer (type)	Output Shape	Param #
conv3d (Conv3D)	(None, 30, 256, 256, 8)	224
conv3d_1 (Conv3D)	(None, 30, 256, 256, 16)	3472
max_pooling3d (MaxPooling3D)	(None, 15, 128, 128, 16)	0
conv3d_2 (Conv3D)	(None, 15, 128, 128, 16)	6928
conv3d_3 (Conv3D)	(None, 15, 128, 128, 16)	6928
max_pooling3d_1 (MaxPooling3D)	(None, 8, 64, 64, 16)	0
conv3d_4 (Conv3D)	(None, 8, 64, 64, 16)	6928
conv3d_5 (Conv3D)	(None, 8, 64, 64, 16)	6928
max_pooling3d_2 (MaxPooling3D)	(None, 4, 32, 32, 16)	0
batch_normalization (BatchNormalization)	(None, 4, 32, 32, 16)	64
conv3d_6 (Conv3D)	(None, 4, 32, 32, 32)	13856
conv3d_7 (Conv3D)	(None, 4, 32, 32, 32)	27680
max_pooling3d_3 (MaxPooling3D)	(None, 2, 16, 16, 32)	0
conv3d_8 (Conv3D)	(None, 2, 16, 16, 32)	27680
max_pooling3d_4 (MaxPooling3D)	(None, 1, 8, 8, 32)	0
batch_normalization_1 (BatchNormalization)	(None, 1, 8, 8, 32)	128
conv3d_9 (Conv3D)	(None, 1, 8, 8, 64)	55360
max_pooling3d_5 (MaxPooling3D)	(None, 1, 4, 4, 64)	0
conv3d_10 (Conv3D)	(None, 1, 4, 4, 64)	110656
max_pooling3d_6 (MaxPooling3D)	(None, 1, 2, 2, 64)	0
conv3d_11 (Conv3D)	(None, 1, 2, 2, 64)	110656
max_pooling3d_7 (MaxPooling3D)	(None, 1, 1, 1, 64)	0
batch_normalization_2 (BatchNormalization)	(None, 1, 1, 1, 64)	256
conv3d_12 (Conv3D)	(None, 1, 1, 1, 128)	221312
max_pooling3d_8 (MaxPooling3D)	(None, 1, 1, 1, 128)	0
conv3d_13 (Conv3D)	(None, 1, 1, 1, 128)	442496
max_pooling3d_9 (MaxPooling3D)	(None, 1, 1, 1, 128)	0
conv3d_14 (Conv3D)	(None, 1, 1, 1, 128)	442496
max_pooling3d_10 (MaxPooling3D)	(None, 1, 1, 1, 128)	0
batch_normalization_3 (BatchNormalization)	(None, 1, 1, 1, 128)	512
conv3d_15 (Conv3D)	(None, 1, 1, 1, 256)	884992
max_pooling3d_11 (MaxPooling3D)	(None, 1, 1, 1, 256)	0
conv3d_16 (Conv3D)	(None, 1, 1, 1, 256)	1769728
max_pooling3d_12 (MaxPooling3D)	(None, 1, 1, 1, 256)	0
conv3d_17 (Conv3D)	(None, 1, 1, 1, 256)	1769728
max_pooling3d_13 (MaxPooling3D)	(None, 1, 1, 1, 256)	0
dropout (Dropout)	(None, 1, 1, 1, 256)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 512)	131584
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

Figure 49 : Model6 for MRNet and KneeMRI

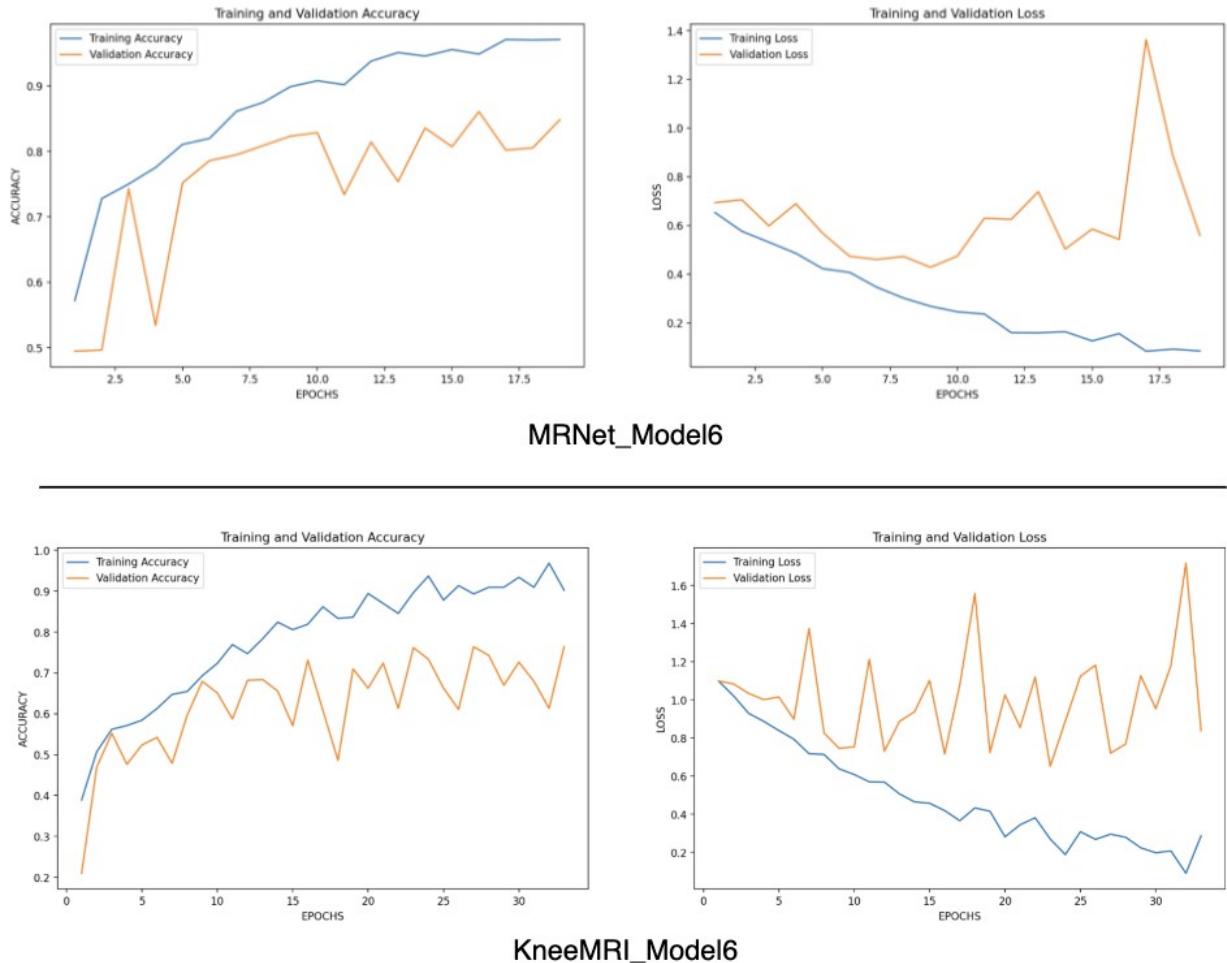


Figure 50 : Model6 Training for MRNet and KneeMRI

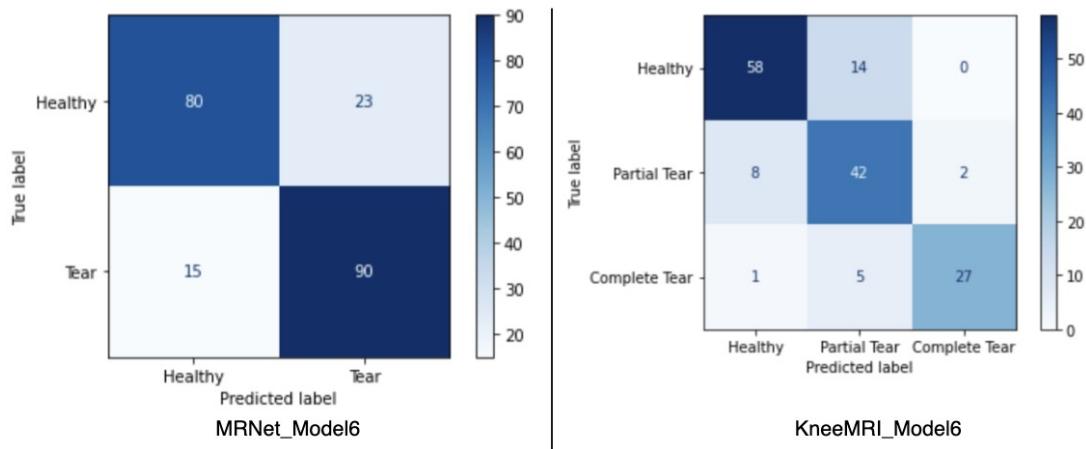


Figure 51 : Model6 Confusion Matrices for MRNet and KneeMRI

Model6 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 9 for MRNet and epoch 23 for KneeMRI.

b) Training models with Transfer Learning (three channels)

- Transfer learning refers to the practice of utilizing a previously developed model for one task as a basis for another task. Specifically, within deep learning and neural networks, this technique involves leveraging pre-trained models as an initial framework and then refining them for a related objective through fine-tuning. (Wang & Chen, 2023)
- We developed three conventional transfer learning models using the TensorFlow Keras library, VGG-16 (Simonyan & Zisserman, 2015), Xception (Chollet, 2017), and ResNet50 (He, et al., 2016).
- These models are trained originally on the ImageNet (Stanford Vision Lab, 2020), a dataset with millions of images used for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which serves as a standard for measuring the performance of deep neural networks to classify 1000 classes.
- The middle three slices of each MRI sample were extracted to comply with a three-channel image as inputs to these models.

▪ Model1_TF_3_VGG16

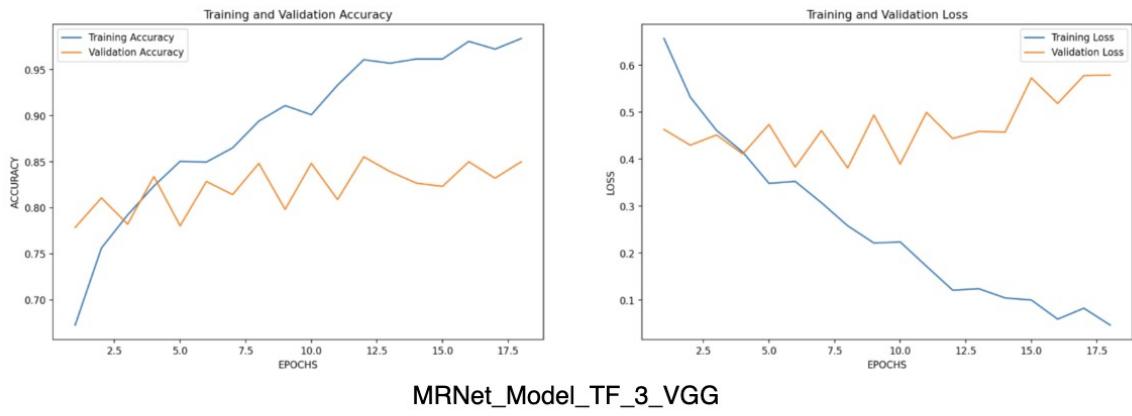
Model: "MRNet_Model_TF_3_VGG"		
Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14714688
flatten_3 (Flatten)	(None, 32768)	0
dense_12 (Dense)	(None, 512)	16777728
dropout_9 (Dropout)	(None, 512)	0
dense_13 (Dense)	(None, 256)	131328
dropout_10 (Dropout)	(None, 256)	0
dense_14 (Dense)	(None, 128)	32896
dropout_11 (Dropout)	(None, 128)	0
dense_15 (Dense)	(None, 1)	129

Total params: 31,656,769
Trainable params: 16,942,081
Non-trainable params: 14,714,688

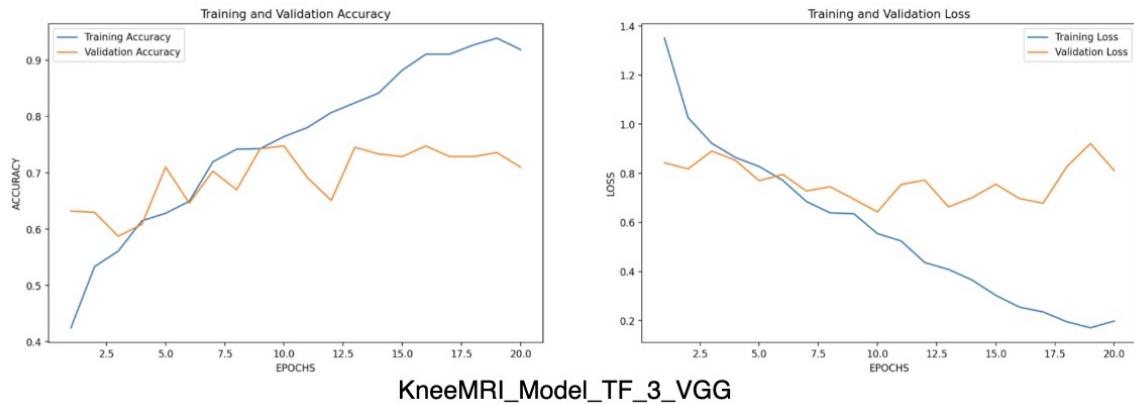
Model: "kneeMRI_Model_TF_3_VGG"		
Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14714688
flatten_1 (Flatten)	(None, 32768)	0
dense_4 (Dense)	(None, 512)	16777728
dropout_3 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 256)	131328
dropout_4 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32896
dropout_5 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 3)	387

Total params: 31,657,027
Trainable params: 16,942,339
Non-trainable params: 14,714,688

Figure 52 : Model_TF_3_VGG16 for MRNet and KneeMRI



MRNet_Model_TF_3_VGG



KneeMRI_Model_TF_3_VGG

Figure 53 : Model_TF_3_VGG16 Training for MRNet and KneeMRI

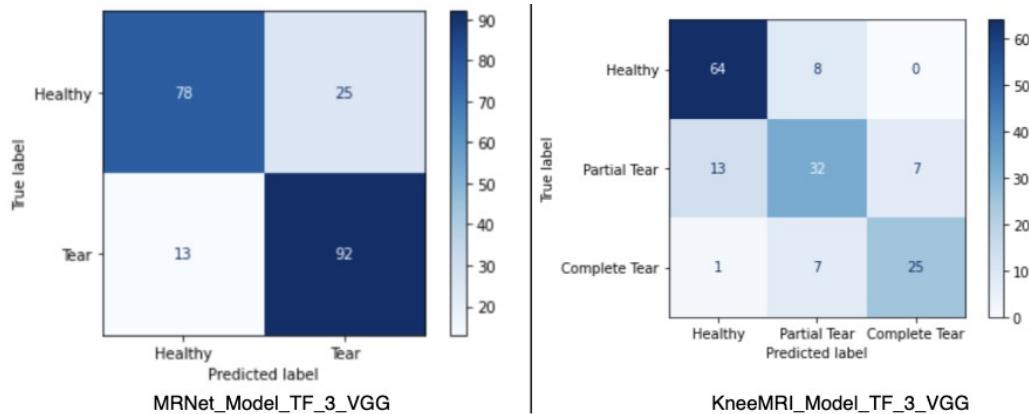


Figure 54 : Model_TF_3_VGG16 Confusion Matrices for MRNet and KneeMRI

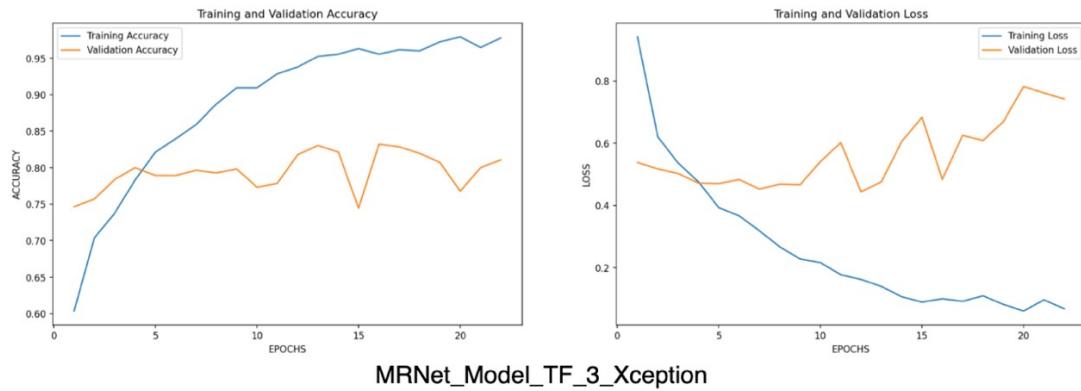
Model_TF_3_VGG16 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 8 for MRNet and epoch 10 for KneeMRI.

- Model2_TF_3_Xception

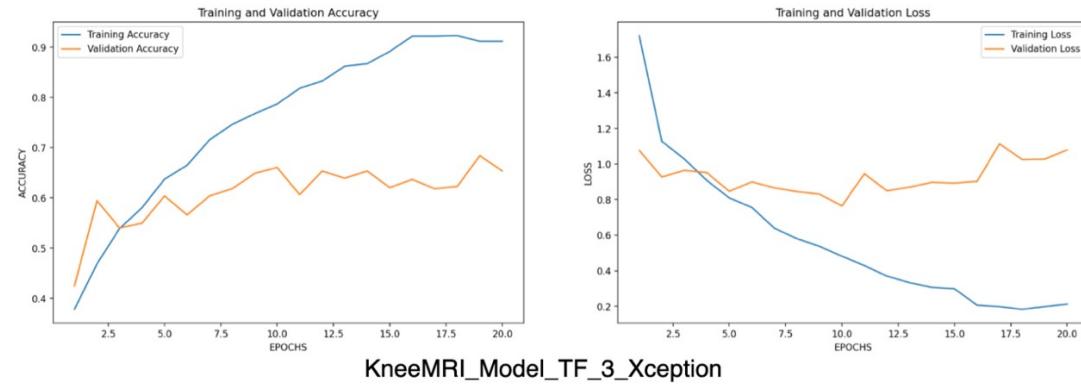
Model: "MRNet_Model_TF_3_Xception"		
Layer (type)	Output Shape	Param #
xception (Functional)	(None, 8, 8, 2048)	20861480
flatten_4 (Flatten)	(None, 131072)	0
dense_16 (Dense)	(None, 512)	67109376
dropout_12 (Dropout)	(None, 512)	0
dense_17 (Dense)	(None, 256)	131328
dropout_13 (Dropout)	(None, 256)	0
dense_18 (Dense)	(None, 128)	32896
dropout_14 (Dropout)	(None, 128)	0
dense_19 (Dense)	(None, 1)	129

Model: "kneeMRI_Model_TF_3_Xception"		
Layer (type)	Output Shape	Param #
xception (Functional)	(None, 8, 8, 2048)	20861480
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67109376
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

Figure 55 : Model_TF_3_Xception for MRNet and KneeMRI



MRNet_Model_TF_3_Xception



KneeMRI_Model_TF_3_Xception

Figure 56 : Model_TF_3_Xception Training for MRNet and KneeMRI

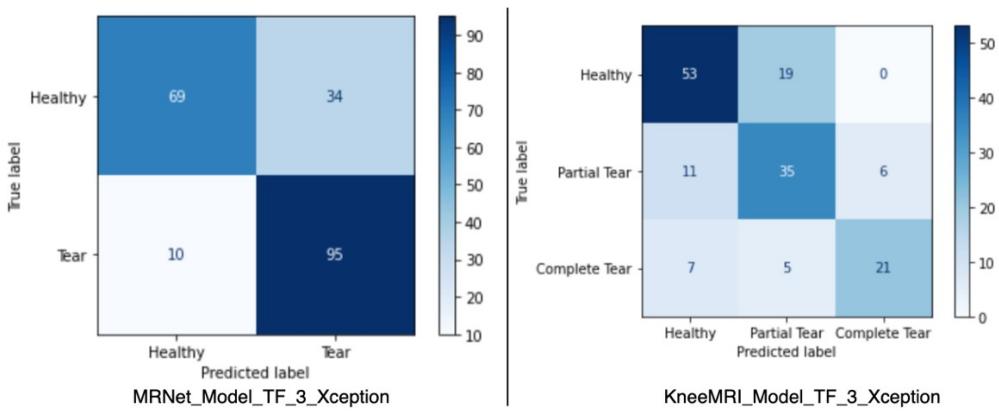


Figure 57 : Model_TF_3_Xception Confusion Matrices for MRNet and KneeMRI

Model_TF_3_Xception restored the weights from the respective best epochs at the end of training based on validation loss, epoch 12 for MRNet and epoch 10 for KneeMRI.

■ Model3_TF_3_ResNet50

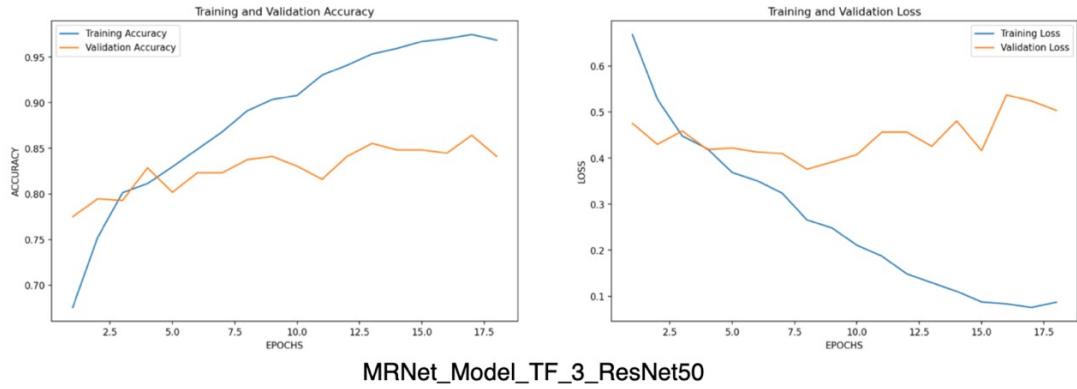
Model: "MRNet_Model_TF_3_ResNet"		
Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14714688
flatten_6 (Flatten)	(None, 32768)	0
dense_24 (Dense)	(None, 512)	16777728
dropout_18 (Dropout)	(None, 512)	0
dense_25 (Dense)	(None, 256)	131328
dropout_19 (Dropout)	(None, 256)	0
dense_26 (Dense)	(None, 128)	32896
dropout_20 (Dropout)	(None, 128)	0
dense_27 (Dense)	(None, 1)	129

Total params: 31,656,769
Trainable params: 16,942,081
Non-trainable params: 14,714,688

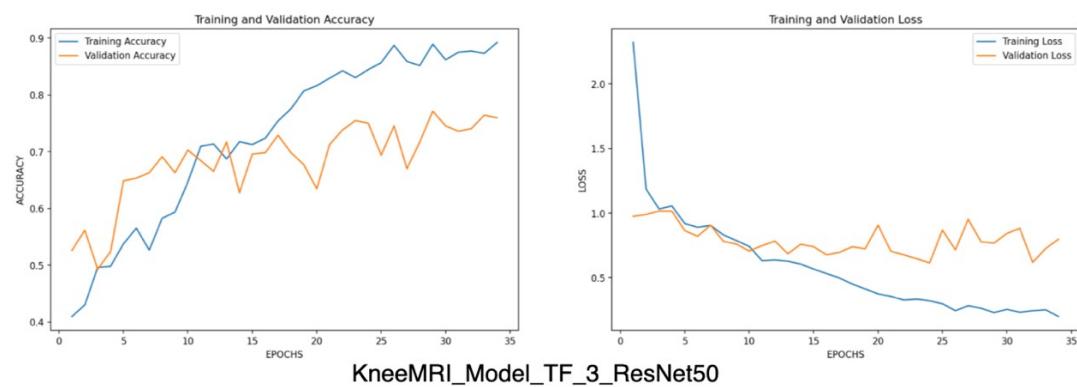
Model: "KneeMRI_Model_TF_3_ResNet"		
Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 8, 8, 2048)	23587712
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67109376
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

Total params: 90,861,699
Trainable params: 67,273,987
Non-trainable params: 23,587,712

Figure 58 : Model_TF_3_ResNet50 for MRNet and KneeMRI



MRNet_Model_TF_3_ResNet50



KneeMRI_Model_TF_3_ResNet50

Figure 59 : Model_TF_3_ResNet50 Training for MRNet and KneeMRI

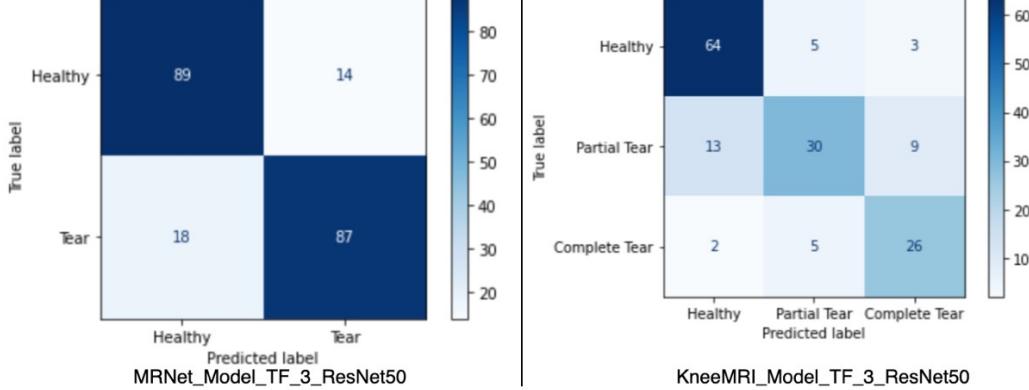


Figure 60 : Model_TF_3_ResNet50 Confusion Matrices for MRNet and KneeMRI

Model_TF_3_ResNet50 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 8 for MRNet and epoch 24 for KneeMRI.

- c) Training models with a hybrid multi-channel approach to Transfer Learning (five channels)
- Two models were developed based on transfer learning with a hybrid multi-channel input. We manipulated the first convolutional blocks of VGG-16 and ResNet-50 by taking an average of the weights in the first three channels and applying them to the extra two channels.
 - The middle five slices were extracted from the MRI samples to train these models to incorporate additional information in the adjoining slices of the middle portion.

■ Model1_TF_5_VGG

Model: "MRNet_Model_TF_5_VGG"			Model: "kneeMRI_Model_TF_5_VGG"		
Layer (type)	Output Shape	Param #	Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14715840	vgg16 (Functional)	(None, 8, 8, 512)	14715840
flatten_5 (Flatten)	(None, 32768)	0	flatten_2 (Flatten)	(None, 32768)	0
dense_20 (Dense)	(None, 512)	16777728	dense_8 (Dense)	(None, 512)	16777728
dropout_15 (Dropout)	(None, 512)	0	dropout_6 (Dropout)	(None, 512)	0
dense_21 (Dense)	(None, 256)	131328	dense_9 (Dense)	(None, 256)	131328
dropout_16 (Dropout)	(None, 256)	0	dropout_7 (Dropout)	(None, 256)	0
dense_22 (Dense)	(None, 128)	32896	dense_10 (Dense)	(None, 128)	32896
dropout_17 (Dropout)	(None, 128)	0	dropout_8 (Dropout)	(None, 128)	0
dense_23 (Dense)	(None, 1)	129	dense_11 (Dense)	(None, 3)	387

Total params: 31,657,921
Trainable params: 16,942,081
Non-trainable params: 14,715,840

Total params: 31,658,179
Trainable params: 16,942,339
Non-trainable params: 14,715,840

Figure 61 : Model_TF_5_VGG16 for MRNet and KneeMRI

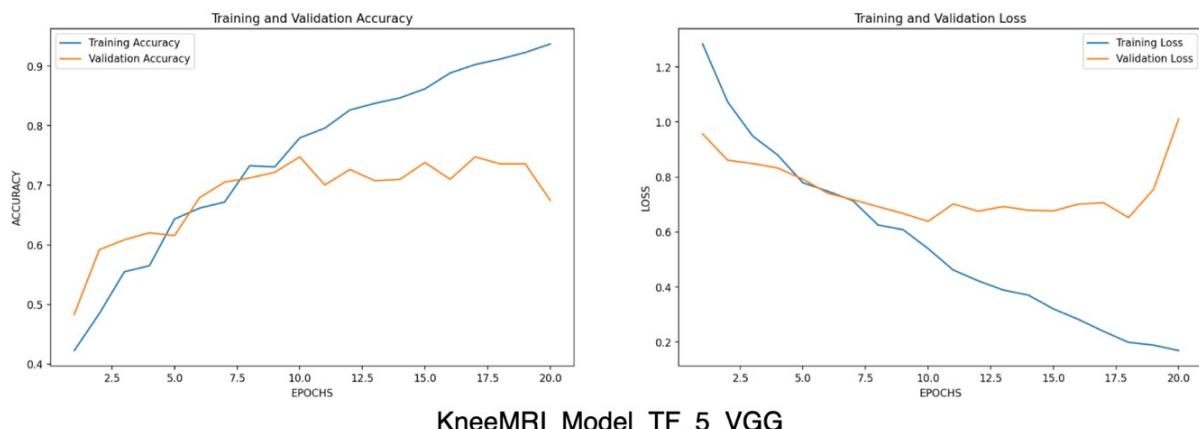
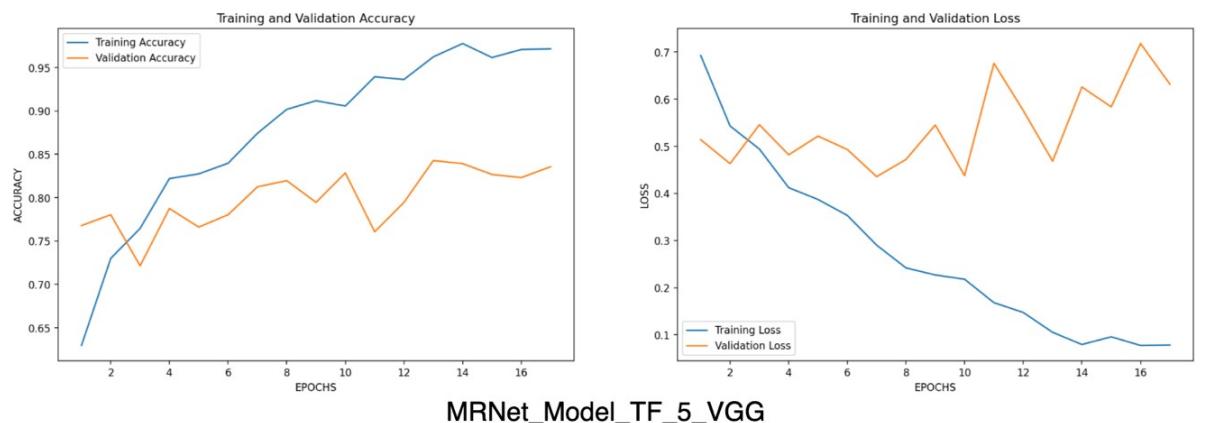


Figure 62 : Model_TF_5_VGG16 Training for MRNet and KneeMRI

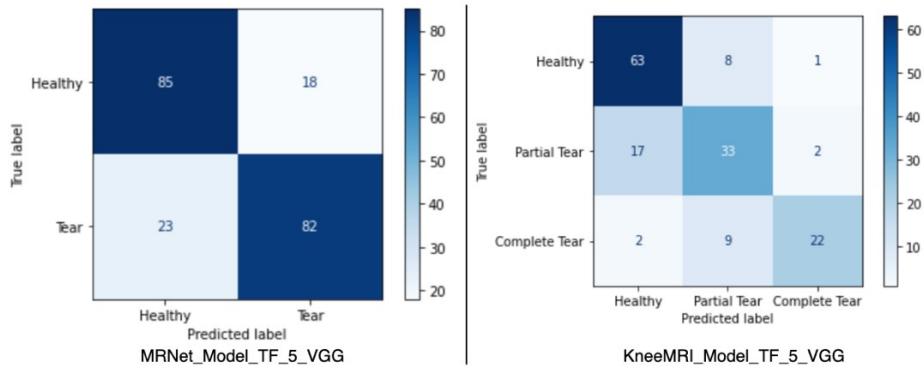


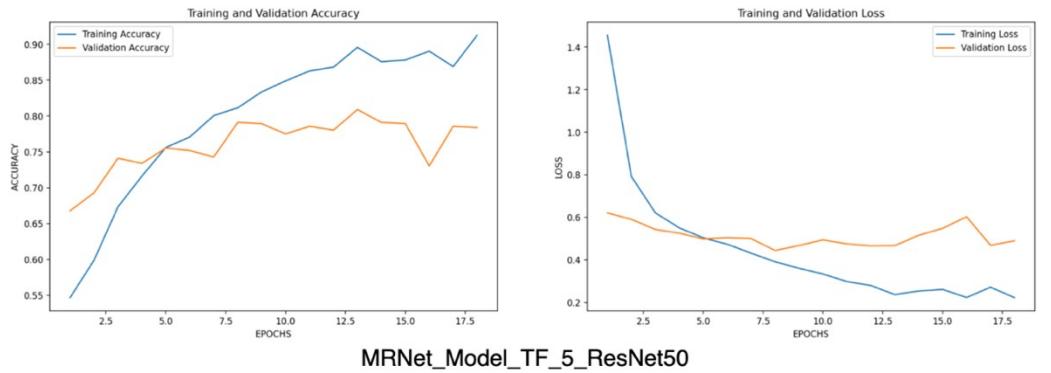
Figure 63 : Model_TF_5_VGG16 Confusion Matrices for MRNet and KneeMRI

Model_TF_5_VGG16 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 7 for MRNet and epoch 10 for KneeMRI.

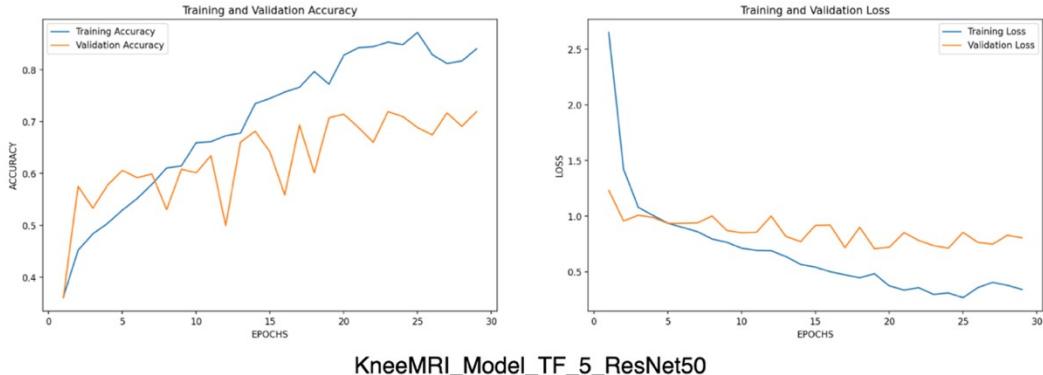
■ Model2_TF_5_ResNet50

Model: "MRNet_Model_TF_5_ResNet"			Model: "kneeMRI_Model_TF_5_ResNet"		
Layer (type)	Output Shape	Param #	Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 8, 8, 2048)	23593984	resnet50 (Functional)	(None, 8, 8, 2048)	23593984
flatten (Flatten)	(None, 131072)	0	flatten_1 (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67109376	dense_4 (Dense)	(None, 512)	67109376
dropout (Dropout)	(None, 512)	0	dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328	dense_5 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0	dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896	dense_6 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0	dropout_5 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129	dense_7 (Dense)	(None, 3)	387
<hr/>					
Total params: 90,867,713					
Trainable params: 67,273,729					
Non-trainable params: 23,593,984					
<hr/>					
Total params: 90,867,971					
Trainable params: 67,273,987					
Non-trainable params: 23,593,984					

Figure 64 : Model_TF_5_ResNet50 for MRNet and KneeMRI



MRNet_Model_TF_5_ResNet50



KneeMRI_Model_TF_5_ResNet50

Figure 65 : Model_TF_5_ResNet50 Training for MRNet and KneeMRI

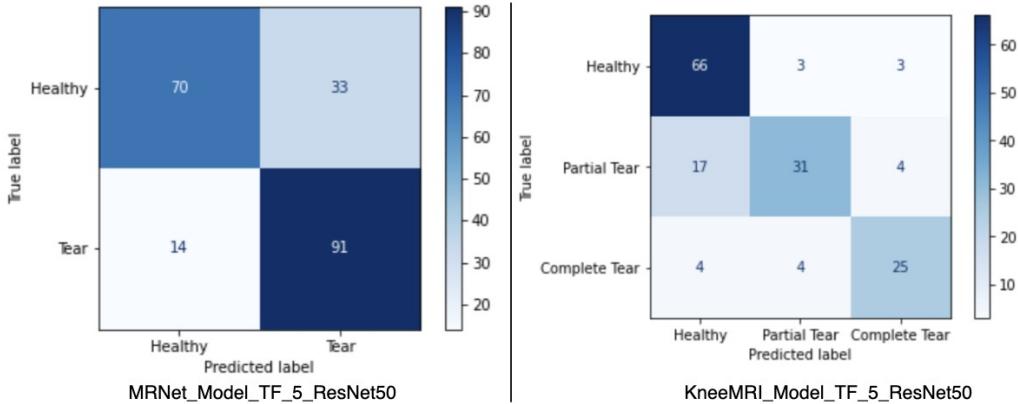


Figure 66 : Model_TF_5_ResNet50 Confusion Matrices for MRNet and KneeMRI

Model_TF_5_ResNet50 restored the weights from the respective best epochs at the end of training based on validation loss, epoch 8 for MRNet and epoch 19 for KneeMRI.

To facilitate the comparison of all trained models, we have compiled a table summarizing the evaluation metrics. Subsequently, the most optimal models for each specific task will be selected and integrated into our product interface:

Table 6 : Comparing Evaluation Metrics of the Models trained on MRNet Dataset

Model	Balanced Accuracy	Precision	Recall	F1-Score	ROC AUC Score
MRNet_Model1	0.83	0.82	0.85	0.84	0.88
MRNet_Model2	0.86	0.83	0.91	0.87	0.92

MRNet_Model3	0.88	0.89	0.86	0.87	0.93
MRNet_Model4	0.85	0.81	0.91	0.86	0.93
MRNet_Model5	0.81	0.76	0.92	0.83	0.90
MRNet_Model6	0.82	0.80	0.86	0.83	0.88
MRNet_Model_TF_3_VGG16	0.82	0.79	0.88	0.83	0.89
MRNet_Model_TF_3_Xception	0.79	0.74	0.90	0.81	0.85
MRNet_Model_TF_3_ResNet50	0.85	0.86	0.83	0.84	0.90
MRNet_Model_TF_5_VGG16	0.80	0.82	0.78	0.80	0.87
MRNet_Model_TF_5_ResNet50	0.77	0.73	0.87	0.79	0.87

Table 7 : Comparing Evaluation Metrics of the Models trained on KneeMRI Dataset

Model	Balanced Accuracy	Precision	Recall	F1-Score	ROC AUC Score
KneeMRI_Model1	0.75	0.78	0.75	0.75	0.90
KneeMRI_Model2	0.70	0.72	0.72	0.72	0.87
KneeMRI_Model3	0.67	0.69	0.68	0.69	0.85
KneeMRI_Model4	0.71	0.74	0.75	0.74	0.91
KneeMRI_Model5	0.69	0.73	0.73	0.73	0.86
KneeMRI_Model6	0.81	0.82	0.81	0.81	0.92
KneeMRI_Model_TF_3_VGG16	0.75	0.77	0.77	0.77	0.90
KneeMRI_Model_TF_3_Xception	0.68	0.70	0.69	0.70	0.84
KneeMRI_Model_TF_3_ResNet50	0.75	0.76	0.76	0.76	0.90
KneeMRI_Model_TF_5_VGG16	0.73	0.76	0.75	0.75	0.91
KneeMRI_Model_TF_5_ResNet50	0.76	0.78	0.78	0.77	0.89

After evaluation of the performance of all the models on the test datasets, we identified MRNet_Model3 (Balanced accuracy=0.88, Precision=0.89, Recall=0.86, F1-Score=0.87, ROC AUC=0.93) for our binary classification task and KneeMRI_Model6 (Balanced accuracy=0.81, Precision=0.82, Recall=0.81, F1-Score=0.81, ROC AUC=0.92) for our multi-class classification task. Using a combination of these model predictions we can identify whether a tear is present or not, moreover, we can identify the grade of tear as well if present.

3.12 Graphical User Interface

After determining the two best models, we integrate them to a Streamlit Application (Streamlit Inc., 2023). This is a quick and easy GUI building library for machine learning and deep learning projects. The user need to upload his/her MRI scan file in the user interface and click on ‘Predict’ button to get results. Some of the screenshots of the application built are shown below:

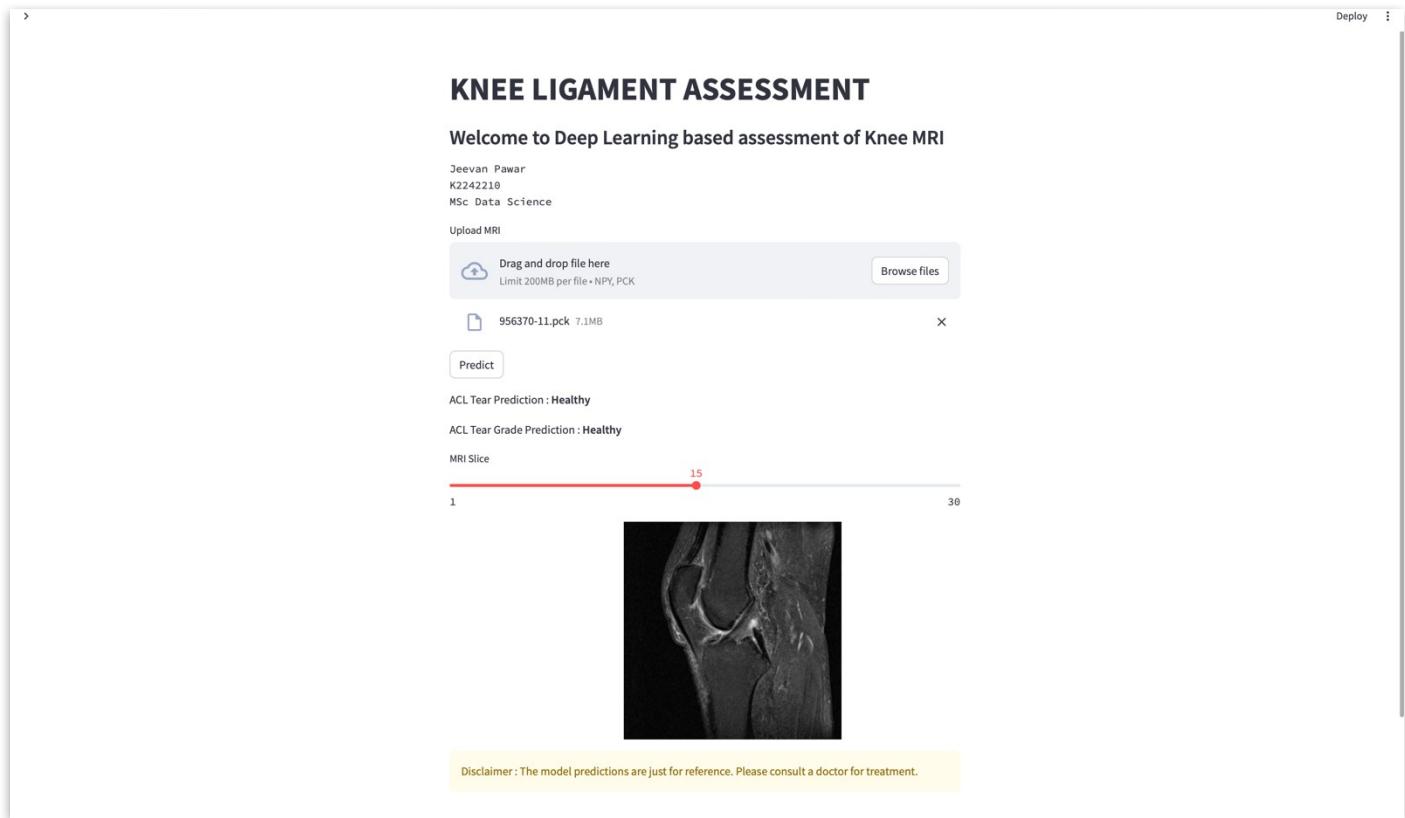


Figure 67 : User Interface with Healthy ACL Prediction from an uploaded MRI

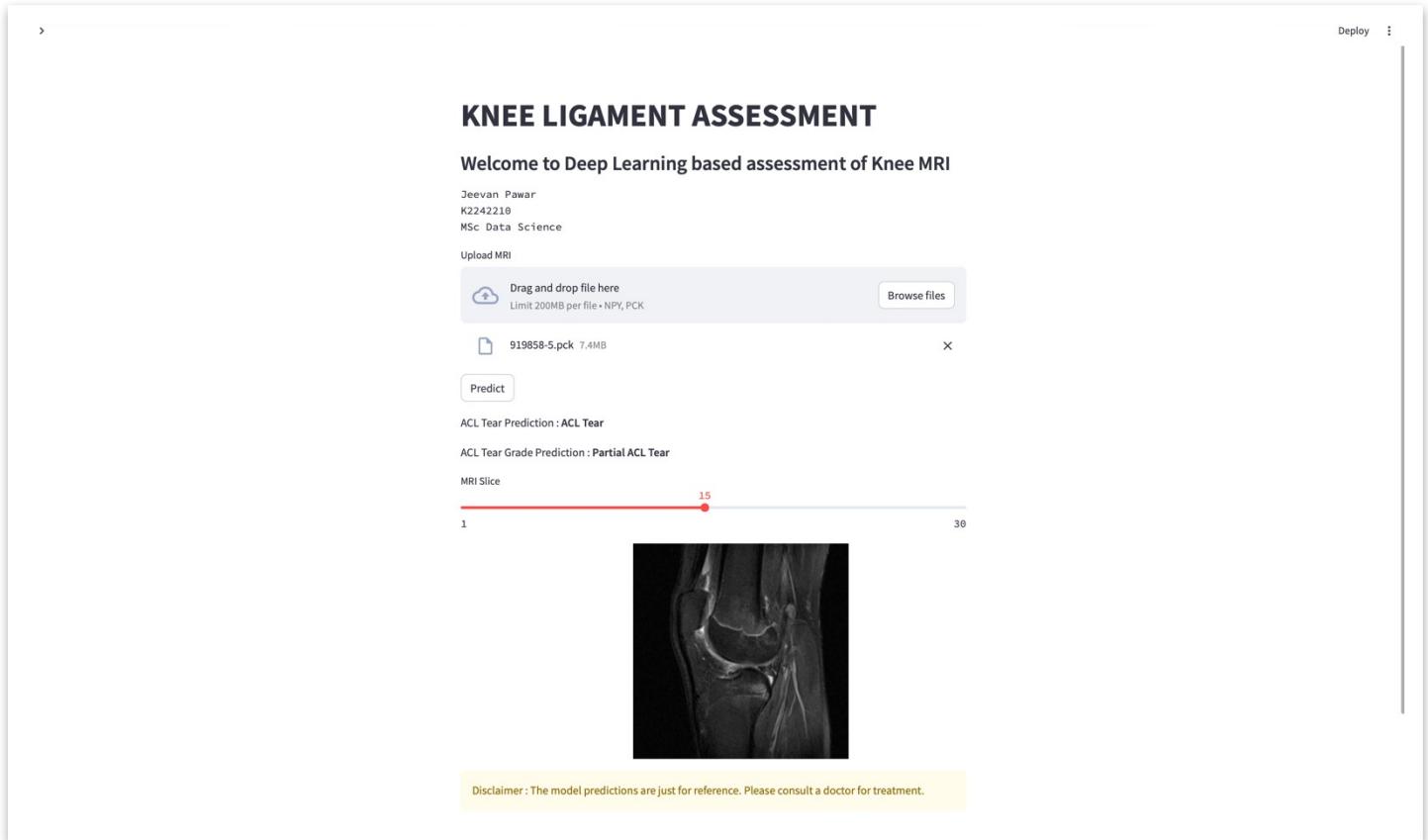


Figure 68 : User Interface with Partial ACL Tear Prediction from an uploaded MRI

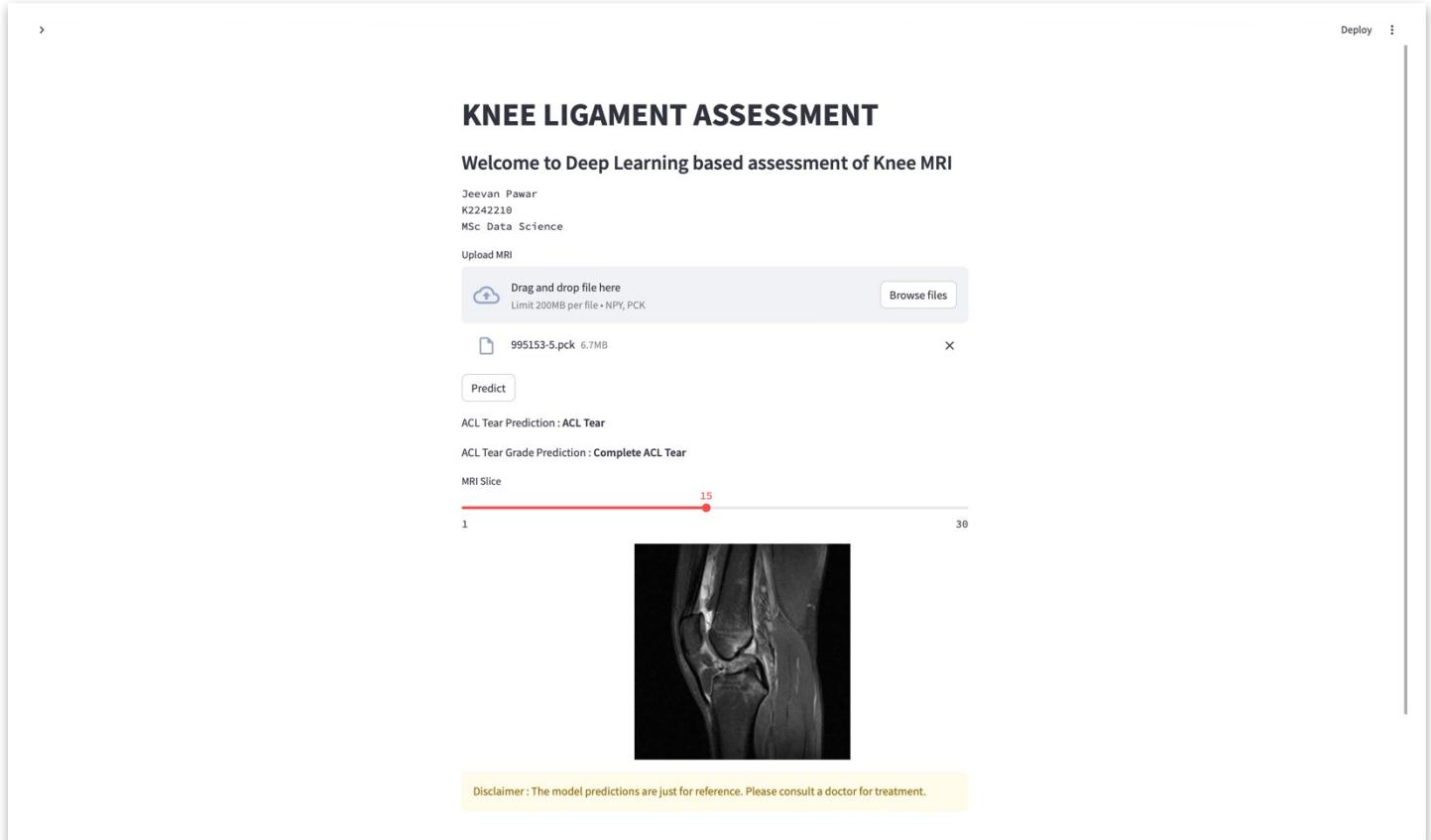


Figure 69 : User Interface with Complete ACL Tear Prediction from an uploaded MRI

Chapter 4 : CONCLUSION

4.1 Summary of Work

In this research study, we developed deep-learning models that can identify ACL tears from MRI scans. We utilised two different datasets of Knee MRIs, namely MRNet and KneeMRI to have a different variety of samples as well as the objective were not only just to identify if a tear was present or not but to identify the grade of the tear as well.

The literature survey of the previously done work in this research topic helped to identify the challenges of working with MRI data, it helped to assess issues present in the current publicly available datasets. We employed ways to overcome the imbalance of samples in the datasets, applied the needed pre-processing techniques to the MRIs, and explored different neural network architectures.

We developed in total 11 convolutional networks for each of the datasets and applied evaluation metrics to identify the best models. In designing the different model architectures, we planned to train 6 different 3D convolutional neural network models that were trained on the entire volumetric slices in an MRI sample.

To enhance our approach, we utilized transfer learning to develop three distinct models based on pre-existing models trained on the 2D coloured ImageNet images. Specifically, we extracted the middle three slices of the MRI samples as these typically contain crucial information about the ligaments within a human knee. This enabled us to simulate a three-channel input in order to leverage transfer learning effectively.

To investigate further, we used transfer learning in a hybrid fashion for multi-channel input from MRI samples. To execute transfer learning on a five-channel input, we created two more models using VGG16 and ResNet50. The first convolutional block of these designs was changed to consume a five-channel MRI derived from the slices in the middle of an MRI volume.

We trained the models on the pre-processed dataset samples after splitting them into test, valid, and train. We used GPU to train the models due to the heavy computations required by a 3D convolutional network. We also applied batch training as the dataset size was large and incorporating the entire dataset into the memory was impossible.

Lastly, we chose the two best models based on evaluation metrics score to integrate with the user interface and finished the knee ligament assessment project. A user can upload his or her MRI scan, and the predictions from the two models show whether a tear is present or not. Additionally, it will also mention the grade of the ligament tear observed in the uploaded MRI.

4.2 Challenges

We will list out some of the major challenges faced along the way in the development of this project:

- Processing 3D volumetric data prior to analysis can present challenges, particularly in terms of time and computational requirements. To address this, the implementation of the N4 bias field correction algorithm was employed as a pre-processing technique. However, due to the extensive dataset size, executing this step would have entailed several days' worth of processing time. Fortunately, through optimization techniques and code enhancements, we successfully improved the efficiency of our approach for implementing this crucial pre-processing stage.
- Class imbalance is one of the major issues in the datasets used, in order to overcome this problem, we applied different data augmentation on the samples from the classes lower in numbers.
- Another challenge arose when configuring TensorFlow Keras to utilize GPU for training, as it turned out to be incompatible with Apple M1 chip-powered MacBook. To overcome this issue, we established the necessary environment on a Kingston University machine to proceed with the training of models. Training these volumetric datasets using the CPU alone would require an extensive amount of time, spanning days or even weeks.

4.3 Critical Review and Reflection on Work

Deep learning has opened a whole new perspective or dimension of solving problems using machines. It can identify essential features and learn to solve a task automatically, however, for a trained model to be acceptable in certain areas of work needs approval on how it works. This presents the challenge of

interpreting these black boxes and explaining the reason behind their predictions. We need to check what our models are picking from an MRI scan to identify the interpretations; this is important especially in medical fields to conclude if a model is making correct prediction by observing the correct features. This can also assist the medical experts or radiologists to check areas of an MRI scan that might have been missed by them.

We could also apply different localisation techniques or pick the slices from an MRI that contain information about the ligaments instead of processing the entire sample or extracting a fixed number of slices from the middle parts. We also rely just on the sagittal planes, if other planes are available, we can develop models to use them for predictions.

We could apply more variety of augmentations applicable to MRI scans such as rescaling, changing brightness or contrast, and not just random horizontal flips and rotations, this will not only increase the number of samples for labels with lesser count but also introduce a much wider variety in the dataset.

Since there are two separate models trained for identifying the tears, sometimes the predictions might not match each other, one model may predict it as healthy whereas the other might interpret it as a partial tear. We need to adopt a better way of combining the results of predicted probabilities from the models and convey them appropriately to the end-user.

We were able to train models with good performance, for detecting a tear we chose MRNet_Model3 (Balanced accuracy=0.88, Precision=0.89, Recall=0.86, F1-Score=0.87, ROC AUC=0.93) and to identify the grade of tear we selected KneeMRI_Model6 (Balanced accuracy=0.81, Precision=0.82, Recall=0.81, F1-Score=0.81, ROC AUC=0.92). However, medical science needs a deep learning model to be highly accurate. We could explore the above-mentioned points to try and improve our models further.

The user interface limits a user to upload only NumPy files or Python pickle object files, this can be made dynamic by accepting other file format such as PNG, JPEG or even DICOM files which is the raw format of an MRI scan.

4.4 Future Scope

This study can be further advanced by utilizing methods such as class activation maps to produce thermal maps of MRI scans analysed by our predictive models. Additionally, the performance of the MRNet model could be enhanced through the integration of specific features from axial and coronal planes.

Due to the lack of availability of datasets for knee MRIs, collecting samples and reports from various sources under appropriate permissions can help to obtain a richer dataset of samples.

Additional methods for hybrid transfer learning can be employed, such as incorporating a larger quantity of MRI slices or utilizing models that have been trained on 3D datasets. These approaches enable the exploration of diverse model options, with some hyperparameter tuning alongside can improve the models' performance further.

REFERENCES

Bhattiprolu, S., 2021. *Tips Tricks 20 - Understanding transfer learning for different size and channel inputs*. [Online] Available at: https://www.youtube.com/watch?v=5kbpoIQUB4Q&ab_channel=DigitalSreeni [Accessed 2 Sep 2023].

Bien, N. et al., 2018. Deep-learning-assisted diagnosis for knee magnetic resonance imaging: Development and retrospective validation of MRNet. *PLOS Medicine*, 15(11).

Bien, N. et al., 2018. *MRNet: Deep-learning-assisted diagnosis for knee magnetic resonance imaging*. [Online] Available at: <https://stanfordmlgroup.github.io/projects/mrnet/>

[Accessed 23 Aug 2023].

Bronstein, R. D. & Schaffer, J. C., 2017. Physical Examination of Knee Ligament Injuries. *Journal of the American Academy of Orthopaedic Surgeons*, 25(4), pp. 280-287.

Brownlee, J., 2019. *How to Accelerate Learning of Deep Neural Networks With Batch Normalization*. [Online] Available at: <https://machinelearningmastery.com/how-to-accelerate-learning-of-deep-neural-networks-with-batch-normalization/> [Accessed 20 Aug 2023].

Brownlee, J., 2019. *How to Develop a Deep Learning Photo Caption Generator from Scratch*. [Online] Available at: <https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/> [Accessed 13 Aug 2023].

Bryan, S., Buxton, M., Sheldon, R. & Grant, A., 1998. Magnetic resonance imaging for the investigation of knee injuries: an investigation of preferences. *Health Economics*, 7(7), p. 595–603.

CS231n, 2021. *Convolutional Neural Networks (CNNs / ConvNets)*. [Online] Available at: <https://cs231n.github.io/convolutional-networks/#layers> [Accessed 12 Aug 2023].

Chamchoun, Y., 2022. *Should Data Science Be Considered As Its Own Discipline?*. [Online] Available at: <https://thedataScientist.com/data-science-considered-own-discipline/> [Accessed 18 Aug 2023].

Chang, P. D., Wong, T. T. & Rasiej, M. J., 2019. Deep Learning for Detection of Complete Anterior Cruciate Ligament Tear. *Journal of Digital Imaging*, Volume 32, pp. 980-986.

Chollet, F., 2017. Xception: Deep Learning with Depthwise Separable Convolutions. *IEEE Conference on Computer Vision and Pattern Recognition*, Volume 2017.

Croft, C., 2012. *Gantt Chart Excel Demo*. [Online] Available at: https://www.youtube.com/watch?v=dDeftOCCMQo&ab_channel=ChrisCroft [Accessed 1 May 2023].

Farshad-Amacker, N. A. & Potter, H. G., 2013. MRI of knee ligament injury and reconstruction. *Journal of Magnetic Resonance Imaging*, 38(4), pp. 757-773.

Gianotti, S. M., Marshall, S. W., Hume, P. A. & Bunt, L., 2009. Incidence of anterior cruciate ligament injury and other knee ligament injuries: A national population-based study. *Journal of Science and Medicine in Sport*, 12(6), pp. 622-627.

He, K., Zhang, X., Ren, S. & Sun, J., 2016. Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, Volume 2016.

Hoeser, T. & Kuenzer, C., 2020. Object Detection and Image Segmentation with Deep Learning on Earth Observation Data: A Review-Part I: Evolution and Recent Trends. *Remote Sensing*, Volume 12.

Jeon, Y. S. et al., 2021. Interpretable and Lightweight 3-D Deep Learning Model for Automated ACL Diagnosis. *IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS*, 25(7), pp. 2388-2397.

Joshi, K. & Suganthi, K., 2022. Anterior Cruciate Ligament Tear Detection Based on Deep Convolutional Neural Network. *Diagnostics*, 12(2314).

Kam, C., Chee, D. W. & Peh, W. C., 2010. Magnetic Resonance Imaging of Cruciate Ligament Injuries of the Knee. *Canadian Association of Radiologists Journal*, 61(2), pp. 80-89.

Kim, J., Hong, J. & Park, H., 2018. Prospects of deep learning for medical imaging. *Precision and Future Medicine*, 2(2), pp. 37-52.

Kohir, V., 2022. *Essential TensorFlow and Keras Callbacks for your Neural Networks*. [Online] Available at: <https://kvirajdatt.medium.com/essential-tensorflow-and-keras-callbacks-for-your-neural-networks-54539244db39> [Accessed 25 Aug 2023].

Lee, J.-G. et al., 2017. Deep Learning in Medical Imaging: General Overview. *Korean Journal of Radiology*, 18(4), pp. 570-584.

Li, Z. et al., 2021. Deep Learning-Based Magnetic Resonance Imaging Image Features for Diagnosis of Anterior Cruciate Ligament Injury. *Journal of Healthcare Engineering*, Volume 2021.

Liu, F. et al., 2019. Fully Automated Diagnosis of Anterior Cruciate Ligament Tears on Knee MR Images by Using Deep Learning. *Radiology: Artificial Intelligence*, 1(3).

Liu, W. et al., 2017. A survey of deep neural network architectures and their applications. *Neurocomputing*, Volume 234, pp. 11-26.

Liu, X. et al., 2021. Advances in Deep Learning-Based Medical Image Analysis. *Health Data Science*, Volume 2021.

Macmillan, C., 2020. *Are ACL Tears Really More Common in Women?*. [Online] Available at: <https://www.yalemedicine.org/news/sports-injuries-gender> [Accessed 28 Aug 2023].

Madeleine, S., 2022. *Normalization, zero centering and standardization of CT images*. [Online] Available at: <https://www.imaios.com/en/resources/blog/ct-images-normalization-zero-centering-and-standardization> [Accessed 17 Aug 2023].

Malato, G., 2021. *Are you still using 0.5 as a threshold?*. [Online] Available at: <https://www.yourdatateacher.com/2021/06/14/are-you-still-using-0-5-as-a-threshold/> [Accessed 23 Aug 2023].

Marx, R. G., Yin, K. & Kaeding, C. C., 2016. AAOS Appropriate Use Criteria: Treatment of Anterior Cruciate Ligament Injuries. *Journal of the American Academy of Orthopaedic Surgeons*, 24(8), pp. e84-e86.

Massobrio, A., 2023. *3D Convolutional Neural Network — A Guide for Engineers*. [Online] Available at: <https://www.neuralconcept.com/post/3d-convolutional-neural-network-a-guide-for-engineers> [Accessed 20 Aug 2023].

Mena, R., 2023. *MRI Preprocessing Techniques*. [Online] Available at: <https://github.com/Angeluz-07/MRI-preprocessing-techniques/tree/main> [Accessed 10 Aug 2023].

Mulcahey, M. K., 2022. *Common Knee Injuries*. [Online] Available at: <https://orthoinfo.aaos.org/en/diseases--conditions/common-knee-injuries/> [Accessed 20 Aug 2023].

Namiri, N. K. et al., 2020. Deep Learning for Hierarchical Severity Staging of Anterior Cruciate Ligament Injuries from MRI. *Radiology: Artificial Intelligence*, 2(4).

NumFOCUS Revision, 2020. *N4 Bias Field Correction*. [Online] Available at: https://simpleitk.readthedocs.io/en/master/link_N4BiasFieldCorrection_docs.html [Accessed 20 Aug 2023].

Ogura, M., 2019. *MRNet*. [Online] Available at: <https://github.com/MisaOgura/MRNet/tree/master> [Accessed 15 Aug 2023].

Potter, H. G. et al., 2002. Magnetic Resonance Imaging of the Multiple-Ligament Injured Knee. *Journal of Orthopaedic Trauma*, 16(5), pp. 330-339.

Remer, E. M. et al., 1992. Anterior cruciate ligament injury: MR imaging diagnosis and patterns of injury. *RadioGraphics*, 12(5), pp. 901-915.

Simonyan, K. & Zisserman, A., 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*, Volume 2015.

Singh, S. P. et al., 2020. 3D Deep Learning on Medical Images: A Review. *Sensors*, 20(18).

Siouras, A. et al., 2022. Knee Injury Detection Using Deep Learning on MRI Studies: A Systematic Review. *Diagnostics*, 12(537).

Spindler, K. P., 2007. *Multicenter Orthopaedics Outcomes Network for ACL Reconstructions*. [Online] Available at: <https://clinicaltrials.gov/study/NCT00463099> [Accessed 24 August 2023].

Stanford Vision Lab, 2020. *Imagenet*. [Online] Available at: <https://www.image-net.org/index.php>

Stephenson, D. R., Rueff, D. & Johnson, D. L., 2010. MRI and Arthroscopic Analysis of Collateral Knee Ligament Injuries In Combined Knee Ligament Injuries. *Orthopedics (Online)*, 33(3), pp. 187-189.

Streamlit Inc., 2023. *Main Concepts*. [Online] Available at: <https://docs.streamlit.io/library/get-started/main-concepts> [Accessed 5 September 2023].

Suzuki, K., 2017. Overview of deep learning in medical imaging. *Radiological Physics and Technology*, Volume 10, pp. 257-273.

Tensorflow Installation Guide, 2023. *Tensorflow Installation Guide*. [Online] Available at: <https://www.tensorflow.org/install/pip#windows-native> [Accessed 10 Aug 2023].

Thakur, A., 2020. *Intuitive understanding of 1D, 2D, and 3D convolutions in convolutional neural networks..* [Online] Available at: <https://wandb.ai/ayush-thakur/dl-question-bank/reports/Intuitive-understanding-of-1D-2D-and-3D-convolutions-in-convolutional-neural-networks--VmlldzoxOTk2MDA> [Accessed 17 Aug 2023].

The SciPy community, n.d. *Multidimensional image processing*. [Online] Available at: <https://docs.scipy.org/doc/scipy/reference/ndimage.html> [Accessed 19 Aug 2023].

The female ACL: Why is it more prone to injury?, 2016. The female ACL: Why is it more prone to injury?. *Journal of Orthopaedics*, 13(2), pp. A1-A4.

Tran, A. et al., 2022. Deep learning to detect anterior cruciate ligament tear on knee MRI: multi-continental external validation. *European Radiology*, Volume 32, pp. 8394-8403.

Tran, D. et al., 2015. Learning Spatiotemporal Features with 3D Convolutional Networks. *IEEE International Conference on Computer Vision*.

Tsai, C.-H. et al., 2020. Knee Injury Detection using MRI with Efficiently-LayeredNetwork (ELNet). *Medical Imaging with Deep Learning*, Volume 121.

Voulodimos, A., Doulamis, N., Doulamis, A. & Protopapadakis, E., 2018. Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*, Volume 2018.

Wang, J. & Chen, Y., 2023. *Introduction to Transfer Learning Algorithms and Practice*. 1st Edition ed. Singapore: Springer Singapore.

Zheng, A., 2015. Evaluating Machine Learning Models. In: *Evaluation Metrics*. 1st ed. California: O'Riley Media, Inc., pp. 7-18.

Zunair, H., 2020. *3D image classification from CT scans*. [Online]
Available at: https://keras.io/examples/vision/3D_image_classification/
[Accessed 18 Aug 2023].

scikit-learn developers, n.d. *sklearn.model_selection.train_test_split*. [Online]
Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
[Accessed 16 Aug 2023].

Štajduhar, I., Mamula, M., Miletić, D. & Ünal, G., 2017. Semi-automated detection of anterior cruciate ligament injury from MRI. *Computer Methods and Programs in Biomedicine*, Volume 140, pp. 151-164.

Štajduhar, I., Mamula, M., Miletić, D. & Ünal, G., 2017. *kneeMRI dataset*. [Online]
Available at: <http://www.riteh.uniri.hr/~istajduh/projects/kneeMRI/>
[Accessed 26 Aug 2023].

APPENDIX

A) Utility

```
import os
import pickle
import platform

import matplotlib.pyplot as plt
import numpy as np
import SimpleITK as sitk
from ipywidgets import interact
from scipy import ndimage
from sklearn.metrics import (ConfusionMatrixDisplay, auc,
                             balanced_accuracy_score, classification_report,
                             confusion_matrix, f1_score,
                             precision_recall_curve, precision_score,
                             recall_score, roc_auc_score, roc_curve)
from sklearn.preprocessing import label_binarize
from sklearn.utils import class_weight, shuffle
from tensorflow import keras
from tensorflow.keras.models import load_model

def explore_3D_volume(vol: np.ndarray, cmap: str = 'gray'):
    """
        Given a 3D volumetric array with shape (Z,X,Y). This function will create
        an interactive
        widget to check out all the 2D arrays with shape (X,Y) inside the 3D
        array.
        The purpose of this function to visually inspect the 2D arrays in the
        image.

    Args:
        vol (np.ndarray): 3D array with shape (Z,X,Y) that represents the
            volume of a MRI image
        cmap (str, optional): color map use to plot the slices in
            matplotlib.pyplot
    """
    def fn(SLICE):
        """
            This function plots MRI slice.

        Args:
            SLICE (NumPy array): MRI Slice
```

```

"""
plt.figure(figsize=(5, 5))
plt.imshow(vol[SLICE, :, :], cmap=cmap)

interact(fn, SLICE=(0, vol.shape[0] - 1))

def compare_3D_volume(vol_before: np.ndarray, vol_after: np.ndarray, cmap: str = 'gray'):
    """
    Given two 3D volumetric arrays with shape (Z,X,Y). This function will
    create an interactive
        widget to check out all the 2D arrays with shape (X,Y) inside the 3D
    arrays.
    The purpose of this function to visual compare the 2D arrays after some
    transformations.

    Args:
        vol_before (np.ndarray): 3D array with shape (Z,X,Y) that represents
            the volume of a MRI image, before any transform
        vol_after (np.ndarray): 3D array with shape (Z,X,Y) that represents
            the volume of a MRI image, after some transform
        cmap (str, optional): Which color map use to plot the slices in
            matplotlib.pyplot
    """
    assert vol_after.shape == vol_before.shape

    def fn(SLICE):
        """
        This function plots before and after MRI slices.

        Args:
            SLICE (NumPy array): MRI Slice
        """
        fig, (ax1, ax2) = plt.subplots(1, 2, sharex='col', sharey='row',
                                      figsize=(10, 10))

        ax1.set_title('Before', fontsize=15)
        ax1.imshow(vol_before[SLICE, :, :], cmap=cmap)

        ax2.set_title('After', fontsize=15)
        ax2.imshow(vol_after[SLICE, :, :], cmap=cmap)

        plt.tight_layout()

    interact(fn, SLICE=(0, vol_before.shape[0] - 1))

def resize_3D_volume(vol, target_size=(30, 256, 256)):
    """
    Given a 3D volumetric array with shape (Z,X,Y). This function will resize
    the image across z-axis.
    The purpose of this function to standardise the depth of MRI image.

```

```

Args:
    vol: 3D array with shape (Z,X,Y) that represents the volume of a MRI
image
        target_size: target size to shape into the volumetric data

Returns:
    np.ndarray: Returns the resized MRI volume
"""
# Set the desired depth
desired_depth, desired_width, desired_height = target_size
# Get current depth
current_depth = vol.shape[0]
current_width = vol.shape[1]
current_height = vol.shape[2]
# Compute depth factor
depth = current_depth / desired_depth
width = current_width / desired_width
height = current_height / desired_height
depth_factor = 1 / depth
width_factor = 1 / width
height_factor = 1 / height
# Resize across z-axis
resized_vol = ndimage.zoom(vol, (depth_factor, width_factor,
height_factor), order=1)
return resized_vol

def denoise_3D_volume(vol):
    """Summary

Args:
    vol (np.ndarray): MRI volume to denoise

Returns:
    np.ndarray: Returns denoised MRI volume
"""
    vol_sitk = sitk.GetImageFromArray(vol)
    denoised_vol_sitk = sitk.CurvatureFlow(vol_sitk, timeStep=0.01,
numberOfIterations=7)
    denoised_vol = sitk.GetArrayFromImage(denoised_vol_sitk)
    return denoised_vol

def bias_field_correction_volume(vol):
    """Summary

Args:
    vol (np.ndarray): MRI volume to perform bias field correction

Returns:
    np.ndarray: Returns bias field corrected MRI volume
"""
    # Convert the NumPy array to SimpleITK image
    vol_sitk = sitk.GetImageFromArray(vol)

```

```

# Perform bias field correction using N4BiasFieldCorrection
corrector = sitk.N4BiasFieldCorrectionImageFilter()
bias_corrected_vol_sitk = corrector.Execute(vol_sitk)

# Get the NumPy array representation of the bias-corrected volume
bias_corrected_vol = sitk.GetArrayFromImage(bias_corrected_vol_sitk)

return bias_corrected_vol

def efficient_bias_field_correction_volume(vol):
    """Summary

    Args:
        vol (np.ndarray): MRI volume to perform efficient bias field
    correction

    Returns:
        np.ndarray: Returns bias field corrected MRI volume
    """
    # Ref: https://medium.com/@alexandro.ramr777/how-to-do-bias-field-correction-with-python-156b9d51dd79
    # Ref:
https://simpleitk.readthedocs.io/en/master/link\_N4BiasFieldCorrection\_docs.html
    # Convert the NumPy array to SimpleITK image
    vol_sitk = sitk.GetImageFromArray(vol)

    vol_sitk = sitk.Cast(vol_sitk, sitk.sitkFloat64)

    vol_sitk_transformed = sitk.RescaleIntensity(vol_sitk, 0, 255)

    vol_sitk_transformed = sitk.LiThreshold(vol_sitk_transformed, 0, 1)

    head_mask = vol_sitk_transformed

    shrink_factor = 4

    input_img = vol_sitk

    input_img = sitk.Shrink(vol_sitk, [shrink_factor] *
input_img.GetDimension())
    mask_img = sitk.Shrink(head_mask, [shrink_factor] *
input_img.GetDimension())

    # Perform bias field correction using N4BiasFieldCorrection
    bias_corrector = sitk.N4BiasFieldCorrectionImageFilter()
    corrected = bias_corrector.Execute(input_img, mask_img)

    log_bias_field = bias_corrector.GetLogBiasFieldAsImage(vol_sitk)

    log_bias_field = sitk.Cast(log_bias_field, sitk.sitkFloat64)

```

```
corrected_image_full_resolution = vol_sitk / sitk.Exp(log_bias_field)

# Get the NumPy array representation of the bias-corrected volume
bias_corrected_vol =
sitk.GetArrayFromImage(corrected_image_full_resolution)

return bias_corrected_vol

def standardise_volume_pixels(vol):
    """Summary

Args:
    vol (np.ndarray): MRI volume

Returns:
    np.ndarray: Standardised MRI volume
"""

# Calculate the mean and standard deviation
mean_value = np.mean(vol)
std_value = np.std(vol)

# Standardise the data
standardised_vol = (vol - mean_value) / std_value

return standardised_vol

def center_volume_pixels(vol):
    """Summary

Args:
    vol (np.ndarray): MRI volume

Returns:
    np.ndarray: Zero centered MRI volume
"""

# Calculate the mean value
mean_value = np.mean(vol)

# Center the data
centered_vol = vol - mean_value

return centered_vol

def normalise_volume_pixels(vol):
    """Summary

Args:
    vol (np.ndarray): MRI volume

Returns:
    np.ndarray: Normalised MRI volume
```

```

"""
# Normalise the volume pixels to the range [0, 1]
min_value = np.min(vol)
max_value = np.max(vol)
normalised_vol = (vol - min_value) / (max_value - min_value)

return normalised_vol

def random_rotation(vol, rotation_angles=[-2.0, -1.5, -1.0, 1.0, 1.5, 2.0, ]):
    """Summary

Args:
    vol (np.ndarray): MRI volume
    rotation_angles (list, optional): List angles for random rotations

Returns:
    np.ndarray: Returns randomly rotated MRI volume
"""
    rotation_angle = np.random.choice(rotation_angles)
    # print(f"Rotation by {rotation_angle} degrees.")
    rotated_vol = ndimage.rotate(vol, rotation_angle, reshape=False,
mode='nearest')
    # print(rotated_vol.shape)
    return rotated_vol

def random_horizontal_flip(vol):
    """Summary

Args:
    vol (np.ndarray): MRI volume

Returns:
    np.ndarray: Returns horizontally flipped MRI volume
"""
    flipped_vol = np.flip(vol, axis=2)
    return flipped_vol

def preprocess_mri(mri_vol):
    """Summary

Args:
    mri_vol (np.ndarray): MRI volume

Returns:
    np.ndarray: Returns preprocessed MRI volume
"""
    mri_vol = resize_3D_volume(mri_vol)
    mri_vol = denoise_3D_volume(mri_vol)
    mri_vol = efficient_bias_field_correction_volume(mri_vol)
    mri_vol = normalise_volume_pixels(mri_vol)
    mri_vol = center_volume_pixels(mri_vol)

```

```

mri_vol = standardise_volume_pixels(mri_vol)
return mri_vol

def get_correct_labels_mrnet(filenames, labels_dataframe):
    """Summary

Args:
    filenames (list): List of filenames of the MRI scans
    labels_dataframe (pd.DataFrame): Dataframe with all MRNet cases and
labels

Returns:
    list: List of corresponding labels for given MRNet MRI filenames
"""
    labels = []
    for file in filenames:
        name = os.path.normpath(file).split(os.sep)[-1]
        case_name = name.split('.')[0]
        label = labels_dataframe.loc[labels_dataframe['Case'] == case_name,
'ACL'].tolist()[0]
        labels.append(label)
    return labels

def get_correct_labels_kneemri(filenames, labels_dataframe):
    """Summary

Args:
    filenames (list): List of filenames of the MRI scans
    labels_dataframe (pd.DataFrame): Dataframe with all KneeMRI metadata
and labels

Returns:
    list: List of corresponding labels for given KneeMRI case filenames
"""
    labels = []
    for file in filenames:
        name = os.path.normpath(file).split(os.sep)[-1]
        vol_file_name = name.split('.')[0] + '.pck'
        label = labels_dataframe.loc[labels_dataframe['volumeFilename'] ==
vol_file_name, 'aclDiagnosis'].tolist()[0]
        labels.append(label)
    return labels

def batch_generator(filenames, labels, batch_size):
    """
    This function loads the respective filenames and labels in the memory
    based on the parameter batch size. It helps to control the amount of
    RAM being consumed as the datasets are large.

Args:
    filenames (list): List of file paths to the MRI
    labels (list): List of corresponding labels of the MRI
    """

```

```

batch_size (int): Batch size

Yields:
...
tuple: Tuple of list of loaded MRI files and corresponding labels
...

N = len(filenames)
i = 0
random_state_counter = 610
filenames, labels = shuffle(filenames, labels,
random_state=random_state_counter + 69) # Shuffle at the start
while True:
    batch_images = []
    batch_filenames = filenames[i:i + batch_size]
    for file in batch_filenames:
        mri_vol = np.load(file)
        mri_vol = np.expand_dims(mri_vol, axis=3) # Adding extra axis for
making it compatible for 3D Convolutions
        batch_images.append(mri_vol)
    batch_labels = labels[i:i + batch_size]
    batch_images = np.array(batch_images)
    batch_labels = np.array(batch_labels)
    yield (batch_images, batch_labels)
    i = i + batch_size
    if i + batch_size > N:
        i = 0
        random_state_counter += 1
        filenames, labels = shuffle(filenames, labels,
random_state=random_state_counter + 69) # Shuffle at the end of each epoch

def predict_batch_generator(filenames, batch_size):
    ...

This function loads the respective filenames and labels in the memory
based on the parameter batch size. It helps to control the amount of
RAM being consumed as the datasets are large.

Args:
    filenames (list): List of filenames of MRI
    batch_size (int): Batch size

Yields:
...
list: List of loaded MRIs
...

N = len(filenames)
i = 0
while i < N:
    batch_images = []
    batch_filenames = filenames[i:i + batch_size]
    for file in batch_filenames:
        mri_vol = np.load(file)
        mri_vol = np.expand_dims(mri_vol, axis=3) # Adding extra axis for
making it compatible for 3D Convolutions
        batch_images.append(mri_vol)

```

```

batch_images = np.array(batch_images)
yield batch_images
i = i + batch_size

def extract_middle_three(mri_vol):
    """Summary

Args:
    mri_vol (np.ndarray): MRI volume

Returns:
    np.ndarray: Returns the extracted middle three slices reshaped to
(256,256,3)
"""
    middle_index = int(len(mri_vol) / 2) - 1
    extracted_portion = mri_vol[middle_index - 1:middle_index + 2]
    extracted_portion = extracted_portion.reshape(256, 256, 3)
    return extracted_portion

def batch_generator_tf_3(filenames, labels, batch_size):
    ...
    This function loads the respective filenames and labels in the memory
    based on the parameter batch size. It helps to control the amount of
    RAM being consumed as the datasets are large.

Args:
    filenames (list): List of file paths to MRI
    labels (list): List of corresponding labels
    batch_size (int): Batch Size

Yields:
    tuple: Tuple of list of loaded MRIs with middle three slices extracted
and corresponding labels
    ...
    N = len(filenames)
    i = 0
    random_state_counter = 610
    filenames, labels = shuffle(filenames, labels,
random_state=random_state_counter + 69) # Shuffle at the start
    while True:
        batch_images = []
        batch_filenames = filenames[i:i + batch_size]
        for file in batch_filenames:
            mri_vol = np.load(file)
            extracted = extract_middle_three(mri_vol)
            # mri_vol = np.expand_dims(mri_vol, axis=3) # Adding extra axis
for making it compatible for 3D Convolutions
            batch_images.append(extracted)
        batch_labels = labels[i:i + batch_size]
        batch_images = np.array(batch_images)
        batch_labels = np.array(batch_labels)
        yield (batch_images, batch_labels)

```

```

    i = i + batch_size
    if i + batch_size > N:
        i = 0
        random_state_counter += 1
        filenames, labels = shuffle(filenames, labels,
random_state=random_state_counter + 69) # Shuffle at the end of each epoch
def predict_batch_generator_tf_3(filenames, batch_size):
    ...
    This function loads the respective filenames and labels in the memory
    based on the parameter batch size. It helps to control the amount of
    RAM being consumed as the datasets are large.

Args:
    filenames (list): List of file paths to MRI
    batch_size (int): Batch Size

Yields:
    list: List of loaded MRI with middle three slices extracted
    ...
N = len(filenames)
i = 0
while i < N:
    batch_images = []
    batch_filenames = filenames[i:i + batch_size]
    for file in batch_filenames:
        mri_vol = np.load(file)
        extracted = extract_middle_three(mri_vol)
        # mri_vol = np.expand_dims(mri_vol, axis=3) # Adding extra axis
    for making it compatible for 3D Convolutions
        batch_images.append(extracted)
    batch_images = np.array(batch_images)
    yield batch_images
    i = i + batch_size

def extract_middle_five(mri_vol):
    """Summary

Args:
    mri_vol (np.ndarray): MRI volume

Returns:
    np.ndarray: Returns the extracted middle five slices reshaped to
(256,256,5)
    """
    middle_index = int(len(mri_vol) / 2) - 1
    extracted_portion = mri_vol[middle_index - 2:middle_index + 3]
    extracted_portion = extracted_portion.reshape(256, 256, 5)
    return extracted_portion

def batch_generator_tf_5(filenames, labels, batch_size):
    ...

```

This function loads the respective filenames and labels in the memory based on the parameter batch size. It helps to control the amount of RAM being consumed as the datasets are large.

Args:

```
filenames (list): List of file paths to MRI  
labels (list): List of corresponding labels  
batch_size (int): Batch Size
```

Yields:

```
tuple: Tuple of list of loaded MRIs with middle five slices extracted  
and corresponding labels  
'''
```

```
N = len(filenames)  
i = 0  
random_state_counter = 610  
filenames, labels = shuffle(filenames, labels,  
random_state=random_state_counter + 69) # Shuffle at the start  
while True:  
    batch_images = []  
    batch_filenames = filenames[i:i + batch_size]  
    for file in batch_filenames:  
        mri_vol = np.load(file)  
        extracted = extract_middle_five(mri_vol)  
        # mri_vol = np.expand_dims(mri_vol, axis=3) # Adding extra axis  
    for making it compatible for 3D Convolutions  
        batch_images.append(extracted)  
    batch_labels = labels[i:i + batch_size]  
    batch_images = np.array(batch_images)  
    batch_labels = np.array(batch_labels)  
    yield (batch_images, batch_labels)  
    i = i + batch_size  
    if i + batch_size > N:  
        i = 0  
        random_state_counter += 1  
        filenames, labels = shuffle(filenames, labels,  
random_state=random_state_counter + 69) # Shuffle at the end of each epoch
```

```
def predict_batch_generator_tf_5(filenames, batch_size):  
    '''
```

This function loads the respective filenames and labels in the memory based on the parameter batch size. It helps to control the amount of RAM being consumed as the datasets are large.

Args:

```
filenames (list): List of file paths to MRI  
batch_size (int): Batch Size
```

Yields:

```
list: List of loaded MRI with middle five slices extracted  
'''
```

```
N = len(filenames)
```

```

i = 0
while i < N:
    batch_images = []
    batch_filenames = filenames[i:i + batch_size]
    for file in batch_filenames:
        mri_vol = np.load(file)
        extracted = extract_middle_five(mri_vol)
        # mri_vol = np.expand_dims(mri_vol, axis=3) # Adding extra axis
    for making it compatible for 3D Convolutions
        batch_images.append(extracted)
    batch_images = np.array(batch_images)
    yield batch_images
    i = i + batch_size

def compute_class_weights(y_train):
    """Summary

    Args:
        y_train (list): List of labels

    Returns:
        dict: A dictionary of labels and their corresponding class weights
    """
    class_weights = dict(zip(np.unique(y_train),
        class_weight.compute_class_weight(class_weight='balanced',
            classes=np.unique(y_train),
            y=y_train)))
    return class_weights

# MODEL CALLBACK FUNCTIONS

def model_lr_schedule(initial_lr=0.0001):
    """Summary

    Args:
        initial_lr (float, optional): Initial learning rate to be set

    Returns:
        TYPE: Learning rate scheduler for Keras
    """
    lr_schedule = keras.optimizers.schedules.ExponentialDecay(initial_lr,
        decay_steps=100000,
        decay_rate=0.96,
        staircase=True)
    return lr_schedule

def model_callback_checkpoint(model_name, model_store_path='Models'):
    """Summary

```

```

Args:
    model_name (str): Name of the model
    model_store_path (str, optional): Path to store the models

Returns:
    TYPE: Keras checkpoint callback to store the best model
"""
file_name = f"{model_store_path}/{model_name}/{model_name}.h5"

# For running code on Windows
if platform.system() == "Windows":
    file_name = file_name.replace('/', '\\')

checkpoint_callback = keras.callbacks.ModelCheckpoint(file_name,
                                                      save_best_only=True)
return checkpoint_callback

def model_callback_earlystopping():
    """Summary

Returns:
    TYPE: Keras earlystopping callback for monitoring Validation Loss
"""
    earlystopping_callback = keras.callbacks.EarlyStopping(monitor="val_loss",
                                                          patience=10,
                                                          verbose=1,
                                                          restore_best_weights=True)
    return earlystopping_callback

def store_model_history(model_name, model_history,
model_history_path='Models'):
    """Summary

Args:
    model_name (str): Model name
    model_history (TYPE): Keras model history
    model_history_path (str, optional): Path to store model history
"""
    file_name = f"{model_history_path}/{model_name}/{model_name}-history.pck"

    # For running code on Windows
    if platform.system() == "Windows":
        file_name = file_name.replace('/', '\\')

    parent_directory = os.path.dirname(file_name)
    if not os.path.exists(parent_directory):
        os.makedirs(parent_directory, exist_ok=True)

```

```

with open(file_name, 'wb') as fh:
    pickle.dump(model_history, fh)

def load_model_history(model_name, model_history_path='Models'):
    """Summary

Args:
    model_name (str): Model name
    model_history_path (str, optional): Path where model history is stored

Returns:
    TYPE: Keras model history
"""
file_name = f"{model_history_path}/{model_name}/{model_name}-history.pck"

# For running code on Windows
if platform.system() == "Windows":
    file_name = file_name.replace('/', '\\')

if os.path.exists(file_name):
    with open(file_name, 'rb') as fh:
        history = pickle.load(fh)
else:
    print(f"ERROR: History file {file_name} not found.")
    return None

return history

def load_model_from_disk(model_name, model_store_path='Models'):
    """Summary

Args:
    model_name (str): Model name
    model_store_path (str, optional): Path to load a model from

Returns:
    TYPE: Description
"""
file_name = f"{model_store_path}/{model_name}/{model_name}.h5"

# For running code on Windows
if platform.system() == "Windows":
    file_name = file_name.replace('/', '\\')

if os.path.exists(file_name):
    model = load_model(file_name)
else:
    print(f"ERROR: Model file {file_name} not found.")
    return None

return model

```

```

# Function for Model Evaluation

def evaluate_model(true_labels, predicted_labels, predicted_probs,
label_names):
    """Summary

Args:
    true_labels (list): List of true labels
    predicted_labels (list): List of predicted labels
    predicted_probs (list): List of predicted probabilities
    label_names (list): List of labels
"""
    print('\nEvaluation Metrics:\n')
    # Check for multi-class, else proceed for binary
    if len(label_names) > 2:
        print(f"Balanced Accuracy : {round(balanced_accuracy_score(true_labels, predicted_labels), 2)}")
        print(f"Precision : {round(precision_score(true_labels, predicted_labels, average='weighted'), 2)}")
        print(f"Recall : {round(recall_score(true_labels, predicted_labels, average='weighted'), 2)}")
        print(f"F1 Score: {round(f1_score(true_labels, predicted_labels, average='weighted'), 2)}")
        print(f"ROC AUC Score : {round(roc_auc_score(true_labels, predicted_probs, multi_class='ovr'), 2)}")
    else:
        print(f"Balanced Accuracy : {round(balanced_accuracy_score(true_labels, predicted_labels), 2)}")
        print(f"Precision : {round(precision_score(true_labels, predicted_labels), 2)}")
        print(f"Recall : {round(recall_score(true_labels, predicted_labels), 2)}")
        print(f"F1 Score: {round(f1_score(true_labels, predicted_labels), 2)}")
        print(f"ROC AUC Score : {round(roc_auc_score(true_labels, predicted_probs), 2)}")

    print("\nClassification report : ")
    print(classification_report(true_labels, predicted_labels,
target_names=label_names))

    matrix = confusion_matrix(true_labels, predicted_labels)
    print("\nConfusion Matrix : ")
    # print(matrix)
    ConfusionMatrixDisplay(matrix,
display_labels=label_names).plot(cmap=plt.cm.Blues)

# Function to plot Accuracy and Loss of a model

def plot_acc_loss(model_history):
    """Summary

```

```

Args:
    model_history (TYPE): Model history
"""
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6), dpi=160)

acc = model_history['accuracy']
val_acc = model_history['val_accuracy']

# Get number of epochs
epochs = range(1, len(acc) + 1)

# Plot training and validation accuracy per epoch
ax1.plot(epochs, acc, label="Training Accuracy")
ax1.plot(epochs, val_acc, label="Validation Accuracy")
ax1.set_title('Training and Validation Accuracy')
ax1.set_xlabel('EPOCHS')
ax1.set_ylabel('ACCURACY')
ax1.legend()

loss = model_history['loss']
val_loss = model_history['val_loss']

# Plot training and validation loss per epoch
ax2.plot(epochs, loss, label="Training Loss")
ax2.plot(epochs, val_loss, label="Validation Loss")
ax2.set_title('Training and Validation Loss')
ax2.set_xlabel('EPOCHS')
ax2.set_ylabel('LOSS')
ax2.legend()
# plt.show()

def calculate_best_cutoff_threshold(labels, predicted_probs):
    """Summary

Args:
    labels (list): List of true labels
    predicted_probs (list): List of predicted probabilities

Returns:
    TYPE: Description
"""
    # Calculate PR Curve
    precision, recall, thresholds = precision_recall_curve(labels,
predicted_probs)

    # Convert to f score
    fscore = (2 * precision * recall) / (precision + recall)

    # Locate the index of the largest f score
    ix = np.argmax(fscore)
    print('\nBest cutoff Threshold = %f, F-Score = %.3f' % (thresholds[ix],

```

```

    fscore[ix]))
    return thresholds[ix]

```

B) MRNet Dataset (Pre-processing of MRNet and Data Augmentation)

```

import os
import platform
from glob import glob

import numpy as np
import pandas as pd
import utils

mrnet_dataset_dir = 'Data/MRNet-v1.0'
mrnet_train_path = os.path.join(mrnet_dataset_dir, 'train')
mrnet_valid_path = os.path.join(mrnet_dataset_dir, 'valid')
mrnet_planes = ['axial', 'coronal', 'sagittal']

# For running code on Windows
if platform.system() == "Windows":
    mrnet_dataset_dir = mrnet_dataset_dir.replace('/', '\\')
    mrnet_train_path = mrnet_train_path.replace('/', '\\')
    mrnet_valid_path = mrnet_valid_path.replace('/', '\\')

mrnet_datasets = {'train': mrnet_train_path, 'valid': mrnet_valid_path}
mrnet_labels = ['abnormal', 'acl', 'meniscus']

# TRAIN DATASET
for label in mrnet_labels:
    if platform.system() == "Windows":
        if label == 'abnormal':
            train_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}\\train-{label}.csv",
                                            header=None,
                                            names=['Case', 'Abnormal'],
                                            dtype={'Case': str, 'Abnormal': np.int64})
        elif label == 'acl':
            train_acl_df = pd.read_csv(f"{mrnet_dataset_dir}\\train-{label}.csv",
                                       header=None,
                                       names=['Case', 'ACL'],
                                       dtype={'Case': str, 'ACL': np.int64})
        if label == 'meniscus':
            train_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}\\train-{label}.csv",
                                            header=None,
                                            names=['Case', 'Meniscus'],
                                            dtype={'Case': str, 'Meniscus': np.int64})
    else:

```

```

if label == 'abnormal':
    train_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                    header=None,
                                    names=['Case', 'Abnormal'],
                                    dtype={'Case': str, 'Abnormal':
np.int64})
elif label == 'acl':
    train_acl_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                header=None,
                                names=['Case', 'ACL'],
                                dtype={'Case': str, 'ACL': np.int64})
if label == 'meniscus':
    train_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                    header=None,
                                    names=['Case', 'Meniscus'],
                                    dtype={'Case': str, 'Meniscus':
np.int64})

train_df = pd.merge(train_abnormal_df, train_acl_df,
on='Case').merge(train_meniscus_df, on='Case')

# VALID DATASET
for label in mrnet_labels:
    if platform.system() == "Windows":
        if label == 'abnormal':
            valid_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}||valid-
{label}.csv",
                                            header=None,
                                            names=['Case', 'Abnormal'],
                                            dtype={'Case': str, 'Abnormal':
np.int64})
        elif label == 'acl':
            valid_acl_df = pd.read_csv(f"{mrnet_dataset_dir}||valid-
{label}.csv",
                                        header=None,
                                        names=['Case', 'ACL'],
                                        dtype={'Case': str, 'ACL': np.int64})
        if label == 'meniscus':
            valid_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}||valid-
{label}.csv",
                                            header=None,
                                            names=['Case', 'Meniscus'],
                                            dtype={'Case': str, 'Meniscus':
np.int64})
    else:
        if label == 'abnormal':
            valid_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                            header=None,
                                            names=['Case', 'Abnormal'],
                                            dtype={'Case': str, 'Abnormal':
np.int64})

```

```

                names=['Case', 'Abnormal'],
                dtype={'Case': str, 'Abnormal':
np.int64})
            elif label == 'acl':
                valid_acl_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                            header=None,
                                            names=['Case', 'ACL'],
                                            dtype={'Case': str, 'ACL': np.int64})
            if label == 'meniscus':
                valid_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                                header=None,
                                                names=['Case', 'Meniscus'],
                                                dtype={'Case': str, 'Meniscus':
np.int64})

valid_df = pd.merge(valid_abnormal_df, valid_acl_df,
on='Case').merge(valid_meniscus_df, on='Case')

def preprocess_mri_vols(cases, overwrite=False):
    """
    This function preprocesses all the MRI volumes in MRNet
    and stores them under 'Preprocessed_Data' directory.

    Args:
        cases (list): List of files in MRNet dataset
        overwrite (bool, optional): Option to overwrite already preprocessed
MRI
    """
    cases.sort()
    for case in cases:
        mri_vol = np.load(case)
        mri_vol = mri_vol.astype(np.float64) # Change the dtype to float64

        case_path = os.path.normpath(case).split(os.sep)
        case_path[0] = 'Preprocessed_Data'
        preprocessed_case_path = os.path.join(*case_path)

        if overwrite or not os.path.exists(preprocessed_case_path):
            preprocessed_mri_vol = utils.preprocess_mri(mri_vol)
            os.makedirs(os.path.join(*case_path[:-1]), exist_ok=True)
            np.save(preprocessed_case_path, preprocessed_mri_vol)

def augment_mri_vols(dataset, labels, aug_flip_prob=0.95, overwrite=False):
    """
    This function augments MRI volumes in MRNet dataset to create more samples
    for labels that have lower number of cases.

    Args:
        dataset (str): Path to either train or valid MRNet dataset
        labels (Pandas dataframe): Labels dataframe for the exams
    """

```

```

    aug_flip_prob (float, optional): Augmentation flip probability
    overwrite (bool, optional): Option to overwrite already preprocessed
MRI
"""
aug_labels_list = []
plane = 'sagittal'
if platform.system() == "Windows":
    cases = glob(f"{dataset}\{plane}\*.npy")
else:
    cases = glob(f"{dataset}/{plane}/*.npy")
cases.sort()
for case in cases:
    # We will create a new path file for augmented images by adding '_aug'
in file names
    # and we store them under the folder <plane>/aug

    case_path = os.path.normpath(case).split(os.sep)
    file_name = case_path[-1]

    orig_sagittal = os.path.join(*case_path)

    case_path[0] = 'Preprocessed_Data'
    case_path.insert(-1, 'aug')

    # SAGITTAL
    sa_temp = file_name
    dot_index = sa_temp.index('.')

    # Do this only once as the label of augmented MRIs will be the same
for all three planes and tasks
    temp_aug_labels = labels.loc[labels['Case'] == sa_temp[:dot_index]]
[['Abnormal', 'ACL', 'Meniscus']].values.tolist()[0]

    # If acl_diagnosis is 1, only 5% chance of augmentation as majority
samples are without tear
    # Increase probability of augmentation in case of ACL tears
    if np.random.rand() >= aug_flip_prob or temp_aug_labels[1] == 1:

        case_path[-1] = f"{sa_temp[:dot_index]}-aug-
0{sa_temp[dot_index:]}"
        aug_sagittal = os.path.join(*case_path)

        if temp_aug_labels[1] == 0:
            if overwrite or not os.path.exists(aug_sagittal):
                mri_vol = np.load(orig_sagittal)
                mri_vol = mri_vol.astype(np.float64) # Change the dtype
to float64

                aug_mri_vol = utils.random_horizontal_flip(mri_vol)
                aug_mri_vol = utils.random_rotation(aug_mri_vol)

                preprocessed_aug_mri_vol =

```

```

utils.preprocess_mri(aug_mri_vol)
    os.makedirs(os.path.join(*case_path[:-1]), exist_ok=True)
    np.save(aug_sagittal, preprocessed_aug_mri_vol)
    aug_labels_list.append([f"{sa_temp[:dot_index]}-aug-{aug-0}"] +
temp_aug_labels)

    elif temp_aug_labels[1] == 1:
        for aug_ind in range(3): # We will augment sample three times
            if aug_ind >= 1:
                case_path[-1] = f"{sa_temp[:dot_index]}-aug-{aug_ind}"
{sa_temp[dot_index:]}"
                aug_sagittal = os.path.join(*case_path)

            if overwrite or not os.path.exists(aug_sagittal):
                mri_vol = np.load(orig_sagittal)
                mri_vol = mri_vol.astype(np.float64) # Change the
dtype to float64

                if aug_ind == 0:
                    aug_mri_vol =
utils.random_horizontal_flip(mri_vol)
                elif aug_ind == 1:
                    aug_mri_vol = utils.random_rotation(mri_vol)
                elif aug_ind == 2:
                    aug_mri_vol =
utils.random_horizontal_flip(mri_vol)
                    aug_mri_vol = utils.random_rotation(aug_mri_vol)
                preprocessed_aug_mri_vol =
utils.preprocess_mri(aug_mri_vol)
                os.makedirs(os.path.join(*case_path[:-1]),
exist_ok=True)
                np.save(aug_sagittal, preprocessed_aug_mri_vol)
                aug_labels_list.append([f"{sa_temp[:dot_index]}-aug-
{aug_ind}"] + temp_aug_labels)

aug_train_df = pd.DataFrame(aug_labels_list, columns=labels.columns)
# print(aug_train_df)
csv_file_path = os.path.normpath(dataset).split(os.sep)
if csv_file_path[-1] == 'train':
    if platform.system() == "Windows":
        aug_train_df.to_csv(os.path.join(*csv_file_path[:-1]) + "\\train-
aug.csv")
    else:
        aug_train_df.to_csv(os.path.join(*csv_file_path[:-1]) + "/train-
aug.csv")
elif csv_file_path[-1] == 'valid':
    if platform.system() == "Windows":
        aug_train_df.to_csv(os.path.join(*csv_file_path[:-1]) + "\\valid-
aug.csv")
    else:
        aug_train_df.to_csv(os.path.join(*csv_file_path[:-1]) + "/valid-
aug.csv")

```

```

print(f"For {dataset.upper()} dataset we have {len(aug_labels_list)} augmented samples.")

def preprocess_mri_vols_for_plane(dataset, plane):
    """
    This function calls preprocessing on given dataset of MRNet and plane.

    Args:
        dataset (str): Path to either train or valid MRNet dataset
        plane (str): MRNet dataset plane axial, coronal or sagittal
    """
    if platform.system() == "Windows":
        cases = glob(f"{dataset}\{plane}\*.npy")
    else:
        cases = glob(f"{dataset}/{plane}/*.npy")
    preprocess_mri_vols(cases)
    print(f"For {dataset.upper()} {plane} we have {len(cases)} samples.")

# Preprocess only sagittal plane
preprocess_mri_vols_for_plane(mrnet_datasets['train'], 'sagittal')

For DATA\MRNET-V1.0\TRAIN sagittal plane we have 1130 samples.

# Preprocess only sagittal plane
preprocess_mri_vols_for_plane(mrnet_datasets['valid'], 'sagittal')

For DATA\MRNET-V1.0\VALID sagittal plane we have 120 samples.

augment_mri_vols(mrnet_datasets['train'], train_df)

For DATA\MRNET-V1.0\TRAIN datset we have 664 augmented samples.

augment_mri_vols(mrnet_datasets['valid'], valid_df)

For DATA\MRNET-V1.0\VALID datset we have 166 augmented samples.

```

C) KneeMRI Dataset (Pre-processing of KneeMRI and Data Augmentation)

```

import os
import pickle
import platform
from glob import glob

import numpy as np
import pandas as pd
import utils

# Directory where the volumetric data is located
kneemri_data_dir = 'Data/KneeMRI'

# Path to metadata csv file
kneemri_metadata_csv_path = 'Data/KneeMRI/metadata.csv'

```

```

# For running code on Windows
if platform.system() == "Windows":
    kneemri_data_dir = kneemri_data_dir.replace('/', '\\')
    kneemri_metadata_csv_path = kneemri_metadata_csv_path.replace('/', '\\')

# Dataset label meanings
kneemri_labels = {0: 'healthy', 1: 'partially ruptured', 2: 'completely ruptured'}

if platform.system() == "Windows":
    mri_vol_paths = glob(kneemri_data_dir + "\\vol*")
else:
    mri_vol_paths = glob(kneemri_data_dir + "/vol*")
mri_vol_paths.sort()

mri_vol_paths

['Data/KneeMRI/vol01',
 'Data/KneeMRI/vol02',
 'Data/KneeMRI/vol03',
 'Data/KneeMRI/vol04',
 'Data/KneeMRI/vol05',
 'Data/KneeMRI/vol06',
 'Data/KneeMRI/vol07',
 'Data/KneeMRI/vol08',
 'Data/KneeMRI/vol09',
 'Data/KneeMRI/vol10']

# names=True loads the interprets the first row of csv file as column names
# 'i4' = 4 byte signed integer, 'U20' = unicode max 20 char string
metadata = np.genfromtxt(kneemri_metadata_csv_path, delimiter=',', names=True,
                         dtype='i4,i4,i4,i4,i4,i4,i4,i4,i4,U20')

metadata_df = pd.DataFrame(metadata)

metadata_df

   examId  seriesNo  aclDiagnosis  kneeLR  roiX  roiY  roiZ  roiHeight \
0    329637         8            0       1    139    184    14        74
1    390116         9            0       0    113    105    10        83
2    404663         8            1       1    120    117    15       101
3    406320         9            0       0    117    124    12        91
4    412857         8            0       1    122    105    14        83
..     ...
912   1027212        5            1       1    113    127    16       101
913   1028019        5            1       1    105    102    14        95
914   1028028        5            0       0    118     84    15       100
915   1028069        5            0       0    105     97    15       103
916   1028670        5            1       0    113    108    14       103

   roiWidth  roiDepth volumeFilename
0        72          3   329637-8.pck
1        98          6   390116-9.pck

```

```

2      115      2  404663-8.pck
3      80       3  406320-9.pck
4      98       4  412857-8.pck
...
912     99       3  1027212-5.pck
913    100      3  1028019-5.pck
914    100      2  1028028-5.pck
915    106      4  1028069-5.pck
916    110      4  1028670-5.pck

[917 rows x 11 columns]

def preprocess_mri_vols(kneemri_data_paths, overwrite=False):
    """
    This function preprocesses all the MRI volumes in KneeMRI
    and stores them under 'Preprocessed_Data' directory.

    Args:
        kneemri_data_paths (list): List of the directories in KneeMRI dataset
        overwrite (bool, optional): Option to overwrite already preprocessed
    """
    for mri_data_path in kneemri_data_paths:
        if platform.system() == "Windows":
            all_exams = glob(mri_data_path + "\\*.pck")
        else:
            all_exams = glob(mri_data_path + "/*.pck")

        all_exams.sort()

        for exam in all_exams:
            exam_path = os.path.normpath(exam).split(os.sep)
            exam_path[0] = 'Preprocessed_Data'

            file_temp = exam_path[-1]
            dot_index = file_temp.index('.')
            exam_path[-1] = file_temp[:dot_index] + '.npy'
            preprocessed_exam_path = os.path.join(*exam_path)

            if overwrite or not os.path.exists(preprocessed_exam_path):
                with open(exam, 'rb') as file_handler: # Must use 'rb' as the
data is binary
                    mri_vol = pickle.load(file_handler)
                    mri_vol = mri_vol.astype(np.float64) # Change the dtype
to float64
                    preprocessed_mri_vol = utils.preprocess_mri(mri_vol)
                    os.makedirs(os.path.join(*exam_path[:-1]), exist_ok=True)
                    np.save(preprocessed_exam_path, preprocessed_mri_vol)

def augment_mri_vols(kneemri_data_paths, metadata_df, aug_flip_prob=0.95,
overwrite=False):
    """
    """

```

This function augments MRI volumes in KneeMRI dataset to create more samples for labels that have lower number of cases.

Args:

```
kneemri_data_paths (list): List of the directories in KneeMRI dataset
metadata_df (Pandas dataframe): Metadata dataframe for the cases
aug_flip_prob (float, optional): Augmentation flip probability
overwrite (bool, optional): Option to overwrite already preprocessed
MRI
"""
aug_labels_list = []
for mri_data_path in kneemri_data_paths:
    if platform.system() == "Windows":
        all_exams = glob(mri_data_path+"\\"*.pck")
    else:
        all_exams = glob(mri_data_path+"/*.pck")
    all_exams.sort()
    for exam in all_exams:
        exam_path = os.path.normpath(exam).split(os.sep)
        exam_vol_name = exam_path[-1]
        exam_labels = metadata_df[metadata_df['volumeFilename'] ==
exam_vol_name].copy()
        acl_diagnosis = exam_labels['aclDiagnosis'].tolist()[0]
        kneeLR_val = exam_labels['kneeLR'].tolist()[0]

            # If acl_diagnosis is 1, only 5% chance of augmentation as
majority samples are healthy
            if np.random.rand() >= aug_flip_prob or acl_diagnosis == 1 or
acl_diagnosis == 2:

                if acl_diagnosis == 0: # Augment into only one sample
                    # Flip kneeLR value as we do a horizontal flip
                    exam_labels.loc[exam_labels['volumeFilename'] ==
exam_vol_name, 'kneeLR'] = 1 - kneeLR_val

                # Make the new volumeFilename value by adding '-aug' in the
original
                new_vol_name = exam_vol_name.split('.')
                new_vol_name[0] = new_vol_name[0] + '-aug-0' #We will
augment healthy samples only once

                # Update the df volumeFilename accordingly
                new_vol_name = '.'.join(new_vol_name)
                exam_labels.loc[exam_labels['volumeFilename'] ==
exam_vol_name, 'volumeFilename'] = new_vol_name

                # Make changes to path where it will be stored
                exam_path[0] = 'Preprocessed_Data'
                exam_path.insert(-1, 'aug')

                # Add changes to filename to be stored
```

```

file_temp = exam_path[-1]
dot_index = file_temp.index('.')
exam_path[-1] = file_temp[:dot_index] + '-aug-0.npy'
preprocessed_exam_path = os.path.join(*exam_path)

if overwrite or not
os.path.exists(preprocessed_exam_path):
    with open(exam, 'rb') as file_handler: # Must use 'rb'
as the data is binary
        mri_vol = pickle.load(file_handler)
        mri_vol = mri_vol.astype(np.float64) # Change the
dtype to float64
utils.random_horizontal_flip(mri_vol)
        aug_mri_vol =
        aug_mri_vol = utils.random_rotation(aug_mri_vol)
        preprocessed_aug_mri_vol =
utils.preprocess_mri(aug_mri_vol)
        os.makedirs(os.path.join(*exam_path[:-1]),
exist_ok=True)
        np.save(preprocessed_exam_path,
preprocessed_aug_mri_vol)
        # Add labels to the augmented samples list
        aug_labels_list.append(exam_labels.values.tolist()
[0])

elif acl_diagnosis == 1: # Augment into multiple samples
sample
    for aug_ind in range(2): # Two augmentations for each
sample
        if aug_ind >= 1:
            exam_labels =
metadata_df[metadata_df['volumeFilename'] == exam_vol_name].copy()
            kneeLR_val = exam_labels['kneeLR'].tolist()[0]

            # Make the new volumeFilename value by adding '-aug' in
the original
            new_vol_name = exam_vol_name.split('.')
            new_vol_name[0] = f"{new_vol_name[0]}-aug-{aug_ind}" #
We will augment partial tear samples two times

            # Update the df volumeFilename accordingly
            new_vol_name = '...'.join(new_vol_name)
            exam_labels.loc[exam_labels['volumeFilename'] ==
exam_vol_name, 'volumeFilename'] = new_vol_name

            # Make changes to path where it will be stored
            # And changes to filename to be stored
            if aug_ind == 0:
                exam_path[0] = 'Preprocessed_Data'
                exam_path.insert(-1, 'aug')
                file_temp = exam_path[-1]

```

```

dot_index = file_temp.index('.')

exam_path[-1] = f'{file_temp[:dot_index]}-aug-'
preprocessed_exam_path = os.path.join(*exam_path)

if overwrite or not
os.path.exists(preprocessed_exam_path):
    with open(exam, 'rb') as file_handler: # Must use
'rb' as the data is binary
        mri_vol = pickle.load(file_handler)
        mri_vol = mri_vol.astype(np.float64) # Change
the dtype to float64
    if aug_ind == 0:
        # Flip kneeLR value as we do a horizontal
flip

exam_labels.loc[exam_labels['volumeFilename'] == exam_vol_name, 'kneeLR'] = 1
- kneelr_val
    aug_mri_vol =
utils.random_horizontal_flip(mri_vol)
    elif aug_ind == 1:
        aug_mri_vol =
utils.random_rotation(mri_vol)
    elif aug_ind == 2:
        # Flip kneeLR value as we do a horizontal
flip

exam_labels.loc[exam_labels['volumeFilename'] == exam_vol_name, 'kneeLR'] = 1
- kneelr_val
    aug_mri_vol =
utils.random_horizontal_flip(mri_vol)
    aug_mri_vol =
utils.random_rotation(aug_mri_vol)
    preprocessed_aug_mri_vol =
utils.preprocess_mri(aug_mri_vol)
        os.makedirs(os.path.join(*exam_path[:-1]),
exist_ok=True)
        np.save(preprocessed_exam_path,
preprocessed_aug_mri_vol)

        # Add labels to the augmented samples list
aug_labels_list.append(exam_labels.values.tolist()[0])

    elif acl_diagnosis == 2: # Augment into even more samples
        for aug_ind in range(5): # Five augmentations for each
sample
            if aug_ind >= 1:
                exam_labels =
metadata_df[metadata_df['volumeFilename'] == exam_vol_name].copy()

```

```

kneelr_val = exam_labels['kneeLR'].tolist()[0]

# Make the new volumeFilename value by adding '-aug' in
the original
new_vol_name = exam_vol_name.split('.')
new_vol_name[0] = f"{new_vol_name[0]}-aug-{aug_ind}" #
We will augment complete tear samples five times

# Update the df volumeFilename accordingly
new_vol_name = '.'.join(new_vol_name)
exam_labels.loc[exam_labels['volumeFilename'] ==
exam_vol_name, 'volumeFilename'] = new_vol_name

# Make changes to path where it will be stored
# And changes to filename to be stored
if aug_ind == 0:
    exam_path[0] = 'Preprocessed_Data'
    exam_path.insert(-1, 'aug')
    file_temp = exam_path[-1]
    dot_index = file_temp.index('.')

exam_path[-1] = f"{file_temp[:dot_index]}-aug-
{aug_ind}.npy"
preprocessed_exam_path = os.path.join(*exam_path)

if overwrite or not
os.path.exists(preprocessed_exam_path):
    with open(exam, 'rb') as file_handler: # Must use
'rb' as the data is binary
        mri_vol = pickle.load(file_handler)
        mri_vol = mri_vol.astype(np.float64) # Change
the dtype to float64
    if aug_ind == 0:
        # Flip kneeLR value as we do a horizontal
flip

exam_labels.loc[exam_labels['volumeFilename'] == exam_vol_name, 'kneeLR'] = 1
- kneelr_val
        aug_mri_vol =
utils.random_horizontal_flip(mri_vol)
    elif aug_ind == 1:
        # Flip kneeLR value as we do a horizontal
flip

exam_labels.loc[exam_labels['volumeFilename'] == exam_vol_name, 'kneeLR'] = 1
- kneelr_val
        aug_mri_vol =
utils.random_horizontal_flip(mri_vol)
        aug_mri_vol =
utils.random_rotation(aug_mri_vol)
    elif aug_ind == 2:
        # Flip kneeLR value as we do a horizontal

```

```

flip

exam_labels.loc[exam_labels['volumeFilename'] == exam_vol_name, 'kneeLR'] = 1
- kneelr_val
                aug_mri_vol =
utils.random_horizontal_flip(mri_vol)
                aug_mri_vol =
utils.random_rotation(aug_mri_vol)
                elif aug_ind == 3:
                    aug_mri_vol =
utils.random_rotation(mri_vol)
                elif aug_ind == 4:
                    aug_mri_vol =
utils.random_rotation(mri_vol)
                    preprocessed_aug_mri_vol =
utils.preprocess_mri(aug_mri_vol)
                        os.makedirs(os.path.join(*exam_path[:-1]),
exist_ok=True)
                        np.save(preprocessed_exam_path,
preprocessed_aug_mri_vol)

# Add labels to the augmented samples list

aug_labels_list.append(exam_labels.values.tolist()[0])

aug_labels_df = pd.DataFrame(aug_labels_list, columns=metadata_df.columns)

csv_file_path = os.path.normpath(kneemri_data_paths[0]).split(os.sep)
if platform.system() == "Windows":
    aug_labels_df.to_csv(os.path.join(*csv_file_path[:-1])+"\\metadata-
aug.csv")
else:
    aug_labels_df.to_csv(os.path.join(*csv_file_path[:-1])+"/metadata-
aug.csv")
print(f"For KneeMRI dataset we have {len(aug_labels_list)} augmented
samples.")

preprocess_mri_vols(mri_vol_paths)
augment_mri_vols(mri_vol_paths, metadata_df)

For KneeMRI dataset we have 650 augmented samples.

```

D) MRNet EDA (MRNet Exploratory Data Analysis)

```
# Explore the directory structure and files in the dataset MRNet
!tree -L 2 Data/MRNet-v1.0/
```

```
Data/MRNet-v1.0/
└── train
    ├── axial
    └── coronal
```

```
└── sagittal
├── train-abnormal.csv
├── train-acl.csv
├── train-meniscus.csv
└── valid
    ├── axial
    ├── coronal
    └── sagittal
├── valid-abnormal.csv
├── valid-acl.csv
└── valid-meniscus.csv
```

9 directories, 6 files

```
import os
import platform
from glob import glob

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

data_dir = 'Data/MRNet-v1.0'
train_data_path = os.path.join(data_dir, 'train')
valid_data_path = os.path.join(data_dir, 'valid')

planes = ['axial', 'coronal', 'sagittal']
datasets = {'train': train_data_path, 'valid': valid_data_path}

for dataset, path in datasets.items():
    print(f"\nTotal exams in {dataset.upper()}")
    for plane in planes:
        print(f"{plane:8} plane : {len(glob(f'{os.path.join(path, plane)}/*.npy'))}")
```

```
Total exams in TRAIN
axial   plane : 1130
coronal plane : 1130
sagittal plane : 1130
```

```
Total exams in VALID
axial   plane : 120
coronal plane : 120
sagittal plane : 120
```

```
def get_slices_per_exam(exams):
    """
    This function gets number of slices found in MRIs
    of the MRNet dataset.
```

Args:

```

exams (list): List of files in MRNet dataset

Returns:
    NumPy array: Array of slices per exam
"""
num_slices_per_exam = []
for exam in exams:
    mri_vol = np.load(exam)
    num_slices_per_exam.append(mri_vol.shape[0])
return np.asarray(num_slices_per_exam)

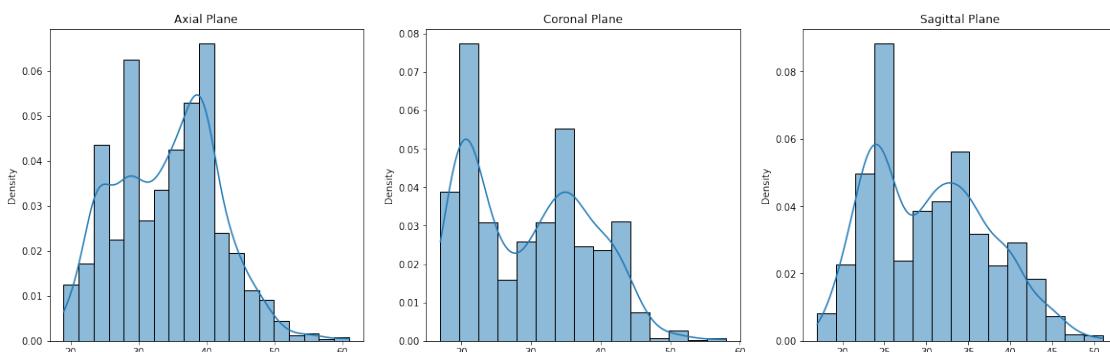
def plot_slices_per_exam(dataset):
    """
    This function plots the distribution of slices found in MRIs
    of the MRNet dataset.

    Args:
        dataset (str): Path to either train or valid MRNet dataset
    """
    fig, axes = plt.subplots(1, 3, figsize=(20, 6))
    for i, plane in enumerate(planes):
        num_slices = get_slices_per_exam(glob(f"{dataset}/{plane}/*.npy"))
        print(f"For {dataset.upper()} {plane} plane min : {num_slices.min()}", 
max : {num_slices.max()}, avg : {num_slices.mean()}")
        sns.histplot(num_slices, stat='density', ax=axes[i], kde=True)
        axes[i].set_title(f"{plane.title()} Plane")

# TRAIN DATASET
plot_slices_per_exam(datasets['train'])

For DATA/MRNET-V1.0\TRAIN axial plane min : 19, max : 61, avg :
34.316814159292036
For DATA/MRNET-V1.0\TRAIN coronal plane min : 17, max : 58, avg :
29.77787610619469
For DATA/MRNET-V1.0\TRAIN sagittal plane min : 17, max : 51, avg :
30.41592920353982

```



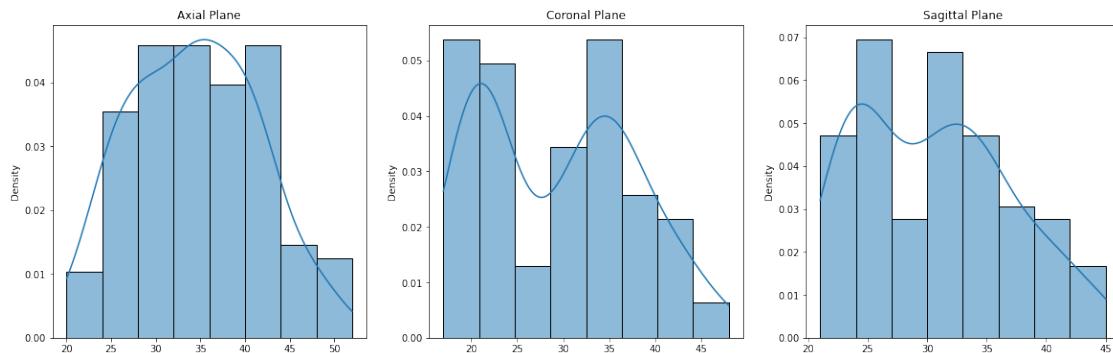
```

# VALID DATASET
plot_slices_per_exam(datasets['valid'])

For DATA/MRNET-V1.0\VALID axial plane min : 20, max : 52, avg :
34.31666666666667

```

For DATA/MRNET-V1.0\VALID coronal plane min : 17, max : 48, avg : 29.425
 For DATA/MRNET-V1.0\VALID sagittal plane min : 21, max : 45, avg : 30.525



```
# TRAIN DATASET
label_categories = ['abnormal', 'acl', 'meniscus']

for label in label_categories:
    if label == 'abnormal':
        train_abnormal_df = pd.read_csv(f"{data_dir}/train-{label}.csv",
                                         header=None,
                                         names=['Case', 'Abnormal'],
                                         dtype={'Case': str, 'Abnormal':
                                                np.int64})
    elif label == 'acl':
        train_acl_df = pd.read_csv(f"{data_dir}/train-{label}.csv",
                                    header=None,
                                    names=['Case', 'ACL'],
                                    dtype={'Case': str, 'ACL': np.int64})
    if label == 'meniscus':
        train_meniscus_df = pd.read_csv(f"{data_dir}/train-{label}.csv",
                                         header=None,
                                         names=['Case', 'Meniscus'],
                                         dtype={'Case': str, 'Meniscus':
                                                np.int64})

train_abnormal_df['Abnormal'].value_counts()
1    913
0    217
Name: Abnormal, dtype: int64

train_acl_df['ACL'].value_counts()
0    922
1    208
Name: ACL, dtype: int64

train_meniscus_df['Meniscus'].value_counts()
0    733
1    397
Name: Meniscus, dtype: int64
```

```

train_df = pd.merge(train_abnormal_df, train_acl_df,
on='Case').merge(train_meniscus_df, on='Case')

train_df

   Case  Abnormal  ACL  Meniscus
0    0000        1    0        0
1    0001        1    1        1
2    0002        1    0        0
3    0003        1    0        1
4    0004        1    0        0
...
1125  1125        1    0        1
1126  1126        1    0        1
1127  1127        0    0        0
1128  1128        1    0        0
1129  1129        1    1        0

[1130 rows x 4 columns]

fig, ax = plt.subplots(1, 3, figsize=(15, 5), dpi=80)
fig.suptitle('MRNet Train : Total Samples in each Class')

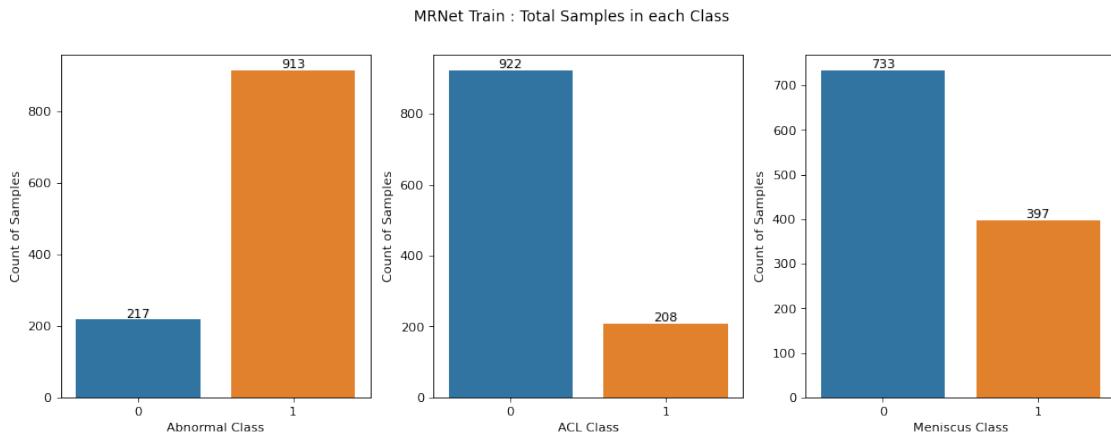
# First graph
sns.countplot(data = train_df, x='Abnormal', ax=ax[0])
ax[0].bar_label(ax[0].containers[0])
ax[0].set_xlabel('Abnormal Class')
ax[0].set_ylabel('Count of Samples')

# Second graph
sns.countplot(data = train_df, x='ACL', ax=ax[1])
ax[1].bar_label(ax[1].containers[0])
ax[1].set_xlabel('ACL Class')
ax[1].set_ylabel('Count of Samples')

# Third graph
sns.countplot(data = train_df, x='Meniscus', ax=ax[2])
ax[2].bar_label(ax[2].containers[0])
ax[2].set_xlabel('Meniscus Class')
ax[2].set_ylabel('Count of Samples')

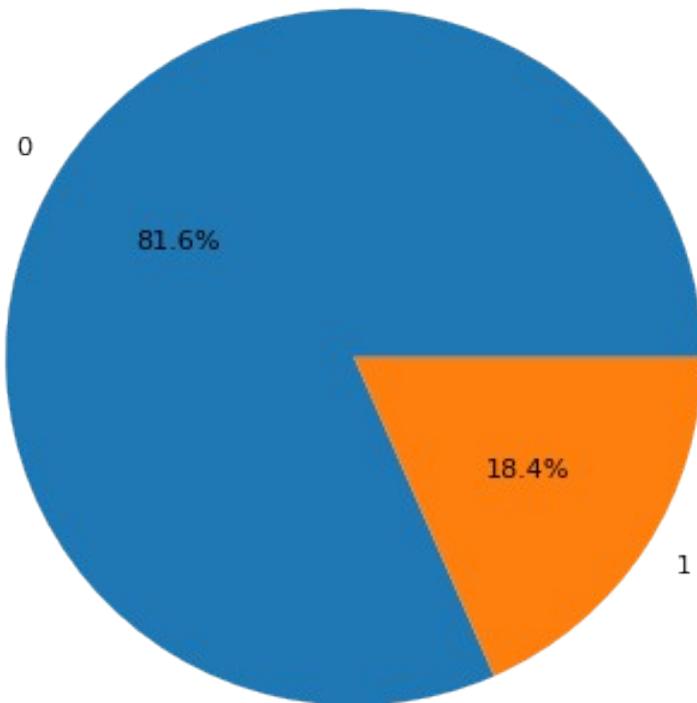
plt.show()

```



```
fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
ax.pie(x=train_df['ACL'].value_counts(),
       labels=train_df['ACL'].value_counts().index,
       autopct='%.1f%%')
ax.set_title('MRNet Train : Pie Chart of ACL Class Imbalance')
plt.show()
```

MRNet: Pie Chart of ACL Class Imbalance



```
# VALID DATASET
for label in label_categories:
    if label == 'abnormal':
```

```

valid_abnormal_df = pd.read_csv(f"{data_dir}/valid-{label}.csv",
                                header=None,
                                names=['Case', 'Abnormal'],
                                dtype={'Case': str, 'Abnormal':
np.int64})
elif label == 'acl':
    valid_acl_df = pd.read_csv(f"{data_dir}/valid-{label}.csv",
                               header=None,
                               names=['Case', 'ACL'],
                               dtype={'Case': str, 'ACL': np.int64})
if label == 'meniscus':
    valid_meniscus_df = pd.read_csv(f"{data_dir}/valid-{label}.csv",
                                    header=None,
                                    names=['Case', 'Meniscus'],
                                    dtype={'Case': str, 'Meniscus':
np.int64})

valid_abnormal_df['Abnormal'].value_counts()
1    95
0    25
Name: Abnormal, dtype: int64

valid_acl_df['ACL'].value_counts()
0    66
1    54
Name: ACL, dtype: int64

valid_meniscus_df['Meniscus'].value_counts()
0    68
1    52
Name: Meniscus, dtype: int64

valid_df = pd.merge(valid_abnormal_df, valid_acl_df,
on='Case').merge(valid_meniscus_df, on='Case')

valid_df
   Case  Abnormal  ACL  Meniscus
0    1130       0     0       0
1    1131       0     0       0
2    1132       0     0       0
3    1133       0     0       0
4    1134       0     0       0
..   ...
115   1245       1     1       1
116   1246       1     1       1
117   1247       1     0       1
118   1248       1     1       1
119   1249       1     0       1

[120 rows x 4 columns]

```

```

fig, ax = plt.subplots(1, 3, figsize=(15, 5), dpi=80)
fig.suptitle('MRNet Valid : Total Samples in each Class')

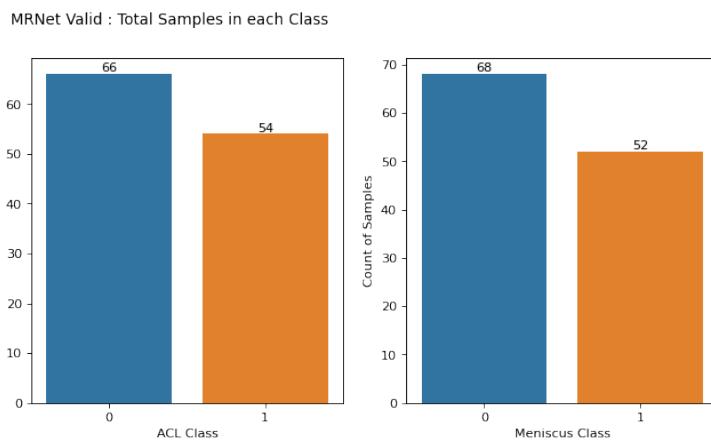
# First graph
sns.countplot(data = valid_df, x='Abnormal', ax=ax[0])
ax[0].bar_label(ax[0].containers[0])
ax[0].set_xlabel('Abnormal Class')
ax[0].set_ylabel('Count of Samples')

# Second graph
sns.countplot(data = valid_df, x='ACL', ax=ax[1])
ax[1].bar_label(ax[1].containers[0])
ax[1].set_xlabel('ACL Class')
ax[1].set_ylabel('Count of Samples')

# Third graph
sns.countplot(data = valid_df, x='Meniscus', ax=ax[2])
ax[2].bar_label(ax[2].containers[0])
ax[2].set_xlabel('Meniscus Class')
ax[2].set_ylabel('Count of Samples')

plt.show()

```

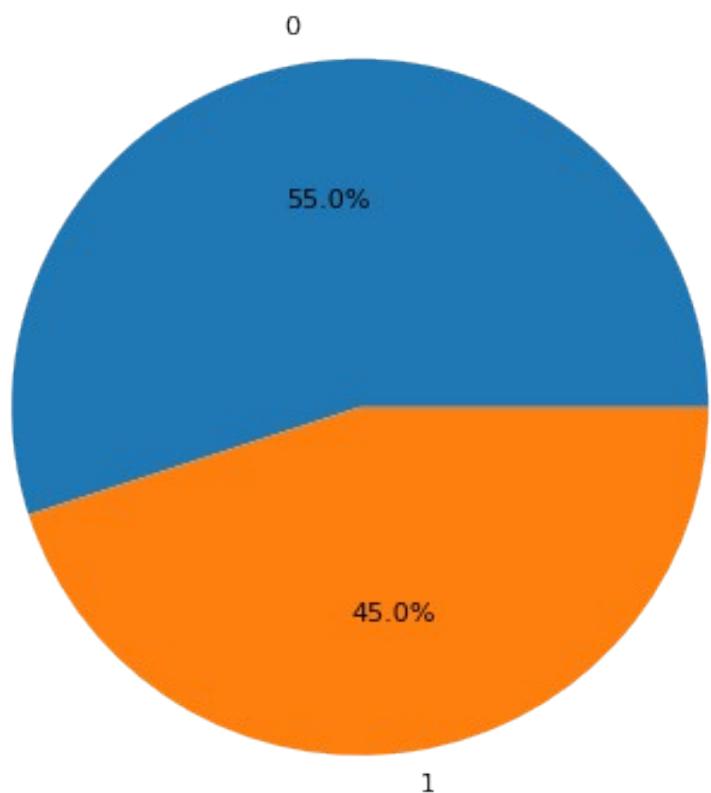


```

fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
ax.pie(x=valid_df['ACL'].value_counts(),
       labels=valid_df['ACL'].value_counts().index,
       autopct='%.1f%%')
ax.set_title('MRNet Valid : Pie Chart of ACL Class Imbalance')
plt.show()

```

MRNet Valid : Pie Chart of ACL Class Imbalance



full_df

```
      Case  Abnormal  ACL  Meniscus
0     0000        1    0        0
1     0001        1    1        1
2     0002        1    0        0
3     0003        1    0        1
4     0004        1    0        0
...
1245   1245        1    1        1
1246   1246        1    1        1
1247   1247        1    0        1
1248   1248        1    1        1
1249   1249        1    0        1
```

[1250 rows x 4 columns]

```
full_df = pd.concat([train_df, valid_df], ignore_index=True)

fig, ax = plt.subplots(1, 3, figsize=(15, 5), dpi=80)
fig.suptitle('MRNet \n\n Total Samples in each Class', fontsize=14)

# First graph
sns.countplot(data = full_df, x='Abnormal', ax=ax[0])
```

```

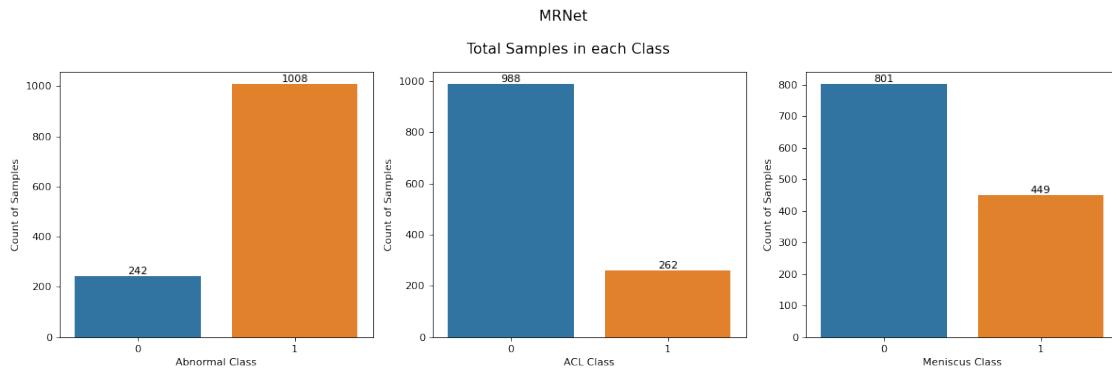
ax[0].bar_label(ax[0].containers[0])
ax[0].set_xlabel('Abnormal Class')
ax[0].set_ylabel('Count of Samples')

# Second graph
sns.countplot(data = full_df, x='ACL', ax=ax[1])
ax[1].bar_label(ax[1].containers[0])
ax[1].set_xlabel('ACL Class')
ax[1].set_ylabel('Count of Samples')

# Third graph
sns.countplot(data = full_df, x='Meniscus', ax=ax[2])
ax[2].bar_label(ax[2].containers[0])
ax[2].set_xlabel('Meniscus Class')
ax[2].set_ylabel('Count of Samples')

fig.tight_layout()
plt.show()

```



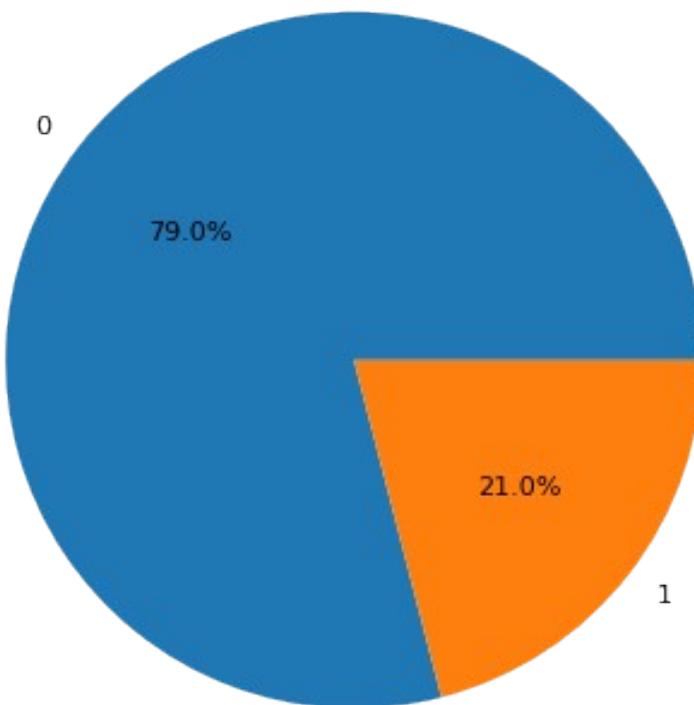
```

fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
fig.suptitle('MRNet', fontsize=14)
ax.pie(x=full_df['ACL'].value_counts(),
       labels=full_df['ACL'].value_counts().index,
       autopct='%.1f%%')
ax.set_title('Pie Chart of ACL Class Imbalance', pad=10)
plt.show()

```

MRNet

Pie Chart of ACL Class Imbalance



```
# Assuming you have a Pandas DataFrame called train_df with the columns
# Abnormal, ACL, Meniscus, etc.

# Group by the three columns and calculate the total cases for each
# combination
grouped = train_df.groupby(['Abnormal', 'ACL',
                           'Meniscus']).size().reset_index(name='Total Cases')

# Calculate the total number of cases in the dataset
total_cases = grouped['Total Cases'].sum()

# Calculate the percentage of total cases for each combination
grouped['Percentage'] = np.round((grouped['Total Cases'] / total_cases) * 100,
                                  2)

# Display the grouped DataFrame
print(grouped)
```

Abnormal	ACL	Meniscus	Total Cases	Percentage
0	0	0	217	19.20
1	1	0	433	38.32

```

2      1   0       1      272     24.07
3      1   1       0      83      7.35
4      1   1       1     125    11.06

train_occurrence_df = train_df.groupby(['Abnormal', 'ACL', 'Meniscus']).count()

train_occurrence_df['Percent'] =
np.round((train_occurrence_df['Case']/train_occurrence_df['Case'].sum())*100, 2)

train_occurrence_df

          Case  Percent
Abnormal ACL Meniscus
0         0   0      217    19.20
1         0   0      433    38.32
           1      272    24.07
           1   0      83     7.35
           1      125   11.06

# Assuming you have a Pandas DataFrame called train_df with the columns
# Abnormal, ACL, Meniscus, etc.

# Group by the three columns and calculate the total cases for each
# combination
grouped = valid_df.groupby(['Abnormal', 'ACL',
'Meniscus']).size().reset_index(name='Total Cases')

# Calculate the total number of cases in the dataset
total_cases = grouped['Total Cases'].sum()

# Calculate the percentage of total cases for each combination
grouped['Percentage'] = np.round((grouped['Total Cases'] / total_cases) * 100,
2)

# Display the grouped DataFrame
print(grouped)

      Abnormal  ACL  Meniscus  Total Cases  Percentage
0         0   0        0        25     20.83
1         1   0        0        20     16.67
2         1   0        1        21     17.50
3         1   1        0        23     19.17
4         1   1        1        31     25.83

valid_occurrence_df = valid_df.groupby(['Abnormal', 'ACL', 'Meniscus']).count()

valid_occurrence_df['Percent'] = np.round((valid_occurrence_df['Case'] /
valid_occurrence_df['Case'].sum()) * 100, 2)

valid_occurrence_df

          Case  Percent
Abnormal ACL Meniscus
0         0   0      25    20.83

```

```

1      0      20    16.67
      1      21    17.50
1      0      23    19.17
      1      31    25.83

# Assuming you have a Pandas DataFrame called train_df with the columns
# Abnormal, ACL, Meniscus, etc.

# Group by the three columns and calculate the total cases for each
# combination
grouped = full_df.groupby(['Abnormal', 'ACL',
                           'Meniscus']).size().reset_index(name='Total Cases')

# Calculate the total number of cases in the dataset
total_cases = grouped['Total Cases'].sum()

# Calculate the percentage of total cases for each combination
grouped['Percentage'] = np.round((grouped['Total Cases'] / total_cases) * 100,
                                  2)

# Display the grouped DataFrame
print(grouped)

   Abnormal  ACL  Meniscus  Total Cases  Percentage
0          0     0         0        242      19.36
1          1     0         0        453      36.24
2          1     0         1        293      23.44
3          1     1         0        106      8.48
4          1     1         1        156     12.48

mrnet_dataset_dir = 'Data/MRNet-v1.0'
mrnet_train_path = os.path.join(mrnet_dataset_dir, 'train')
mrnet_valid_path = os.path.join(mrnet_dataset_dir, 'valid')

preprocessed_mrnet_dataset_dir = 'Preprocessed_Data/MRNet-v1.0'
preprocessed_mrnet_train_path = os.path.join(preprocessed_mrnet_dataset_dir,
                                             'train')
preprocessed_mrnet_valid_path = os.path.join(preprocessed_mrnet_dataset_dir,
                                             'valid')

mrnet_planes = ['axial', 'coronal', 'sagittal']

# For running code on Windows
if platform.system() == "Windows":
    mrnet_dataset_dir = mrnet_dataset_dir.replace('/', '\\')
    mrnet_train_path = mrnet_train_path.replace('/', '\\')
    mrnet_valid_path = mrnet_valid_path.replace('/', '\\')

    preprocessed_mrnet_dataset_dir =
    preprocessed_mrnet_dataset_dir.replace('/', '\\')
        preprocessed_mrnet_train_path = preprocessed_mrnet_train_path.replace('/',
        '\\')

```

```

    preprocessed_mrnet_valid_path = preprocessed_mrnet_valid_path.replace('/', 
'\\')

mrnet_datasets = {'train': mrnet_train_path, 'valid': mrnet_valid_path}
mrnet_labels = ['abnormal', 'acl', 'meniscus']

# TRAIN DATASET
for label in mrnet_labels:
    if platform.system() == "Windows":
        if label == 'abnormal':
            train_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}\\train-
{label}.csv",
                                            header=None,
                                            names=['Case', 'Abnormal'],
                                            dtype={'Case': str, 'Abnormal': 
np.int64})
        elif label == 'acl':
            train_acl_df = pd.read_csv(f"{mrnet_dataset_dir}\\train-
{label}.csv",
                                            header=None,
                                            names=['Case', 'ACL'],
                                            dtype={'Case': str, 'ACL': np.int64})
        if label == 'meniscus':
            train_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}\\train-
{label}.csv",
                                            header=None,
                                            names=['Case', 'Meniscus'],
                                            dtype={'Case': str, 'Meniscus': 
np.int64})
    else:
        if label == 'abnormal':
            train_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                            header=None,
                                            names=['Case', 'Abnormal'],
                                            dtype={'Case': str, 'Abnormal': 
np.int64})
        elif label == 'acl':
            train_acl_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                            header=None,
                                            names=['Case', 'ACL'],
                                            dtype={'Case': str, 'ACL': np.int64})
        if label == 'meniscus':
            train_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                            header=None,
                                            names=['Case', 'Meniscus'],
                                            dtype={'Case': str, 'Meniscus': 
np.int64})

```

```

train_df = pd.merge(train_abnormal_df, train_acl_df,
on='Case').merge(train_meniscus_df, on='Case')

# VALID DATASET
for label in mrnet_labels:
    if platform.system() == "Windows":
        if label == 'abnormal':
            valid_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}\valid-
{label}.csv",
                                            header=None,
                                            names=['Case', 'Abnormal'],
                                            dtype={'Case': str, 'Abnormal':
np.int64})
        elif label == 'acl':
            valid_acl_df = pd.read_csv(f"{mrnet_dataset_dir}\valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'ACL'],
                                         dtype={'Case': str, 'ACL': np.int64})
        if label == 'meniscus':
            valid_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}\valid-
{label}.csv",
                                             header=None,
                                             names=['Case', 'Meniscus'],
                                             dtype={'Case': str, 'Meniscus':
np.int64})
    else:
        if label == 'abnormal':
            valid_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                              header=None,
                                              names=['Case', 'Abnormal'],
                                              dtype={'Case': str, 'Abnormal':
np.int64})
        elif label == 'acl':
            valid_acl_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'ACL'],
                                         dtype={'Case': str, 'ACL': np.int64})
        if label == 'meniscus':
            valid_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                             header=None,
                                             names=['Case', 'Meniscus'],
                                             dtype={'Case': str, 'Meniscus':
np.int64})
    valid_df = pd.merge(valid_abnormal_df, valid_acl_df,
on='Case').merge(valid_meniscus_df, on='Case')

```

```

# AUGMENTED TRAIN LABELS
if platform.system() == "Windows":
    train_aug_df = pd.read_csv(f"{mrnet_dataset_dir}\train-aug.csv",
                               index_col=0,
                               dtype={'Case': str, 'Abnormal': np.int64,
                                      'ACL': np.int64, 'Meniscus': np.int64})
else:
    train_aug_df = pd.read_csv(f"{mrnet_dataset_dir}/train-aug.csv",
                               index_col=0,
                               dtype={'Case': str, 'Abnormal': np.int64,
                                      'ACL': np.int64, 'Meniscus': np.int64})

# AUGMENTED VALID LABELS
if platform.system() == "Windows":
    valid_aug_df = pd.read_csv(f"{mrnet_dataset_dir}\valid-aug.csv",
                               index_col=0,
                               dtype={'Case': str, 'Abnormal': np.int64,
                                      'ACL': np.int64, 'Meniscus': np.int64})
else:
    valid_aug_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-aug.csv",
                               index_col=0,
                               dtype={'Case': str, 'Abnormal': np.int64,
                                      'ACL': np.int64, 'Meniscus': np.int64})

full_mrnet_df = pd.concat([train_df, valid_df, train_aug_df, valid_aug_df],
                          ignore_index=True)

len(full_mrnet_df)
2080

fig, ax = plt.subplots(1, 3, figsize=(15, 5), dpi=80)
fig.suptitle('MRNet \n\n Total Samples in each Class', fontsize=14)

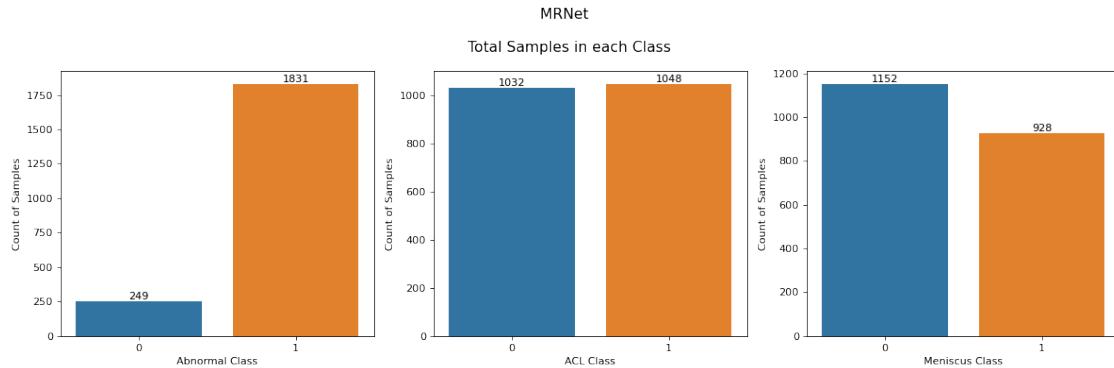
# First graph
sns.countplot(data = full_mrnet_df, x='Abnormal', ax=ax[0])
ax[0].bar_label(ax[0].containers[0])
ax[0].set_xlabel('Abnormal Class')
ax[0].set_ylabel('Count of Samples')

# Second graph
sns.countplot(data = full_mrnet_df, x='ACL', ax=ax[1])
ax[1].bar_label(ax[1].containers[0])
ax[1].set_xlabel('ACL Class')
ax[1].set_ylabel('Count of Samples')

# Third graph
sns.countplot(data = full_mrnet_df, x='Meniscus', ax=ax[2])
ax[2].bar_label(ax[2].containers[0])
ax[2].set_xlabel('Meniscus Class')
ax[2].set_ylabel('Count of Samples')

```

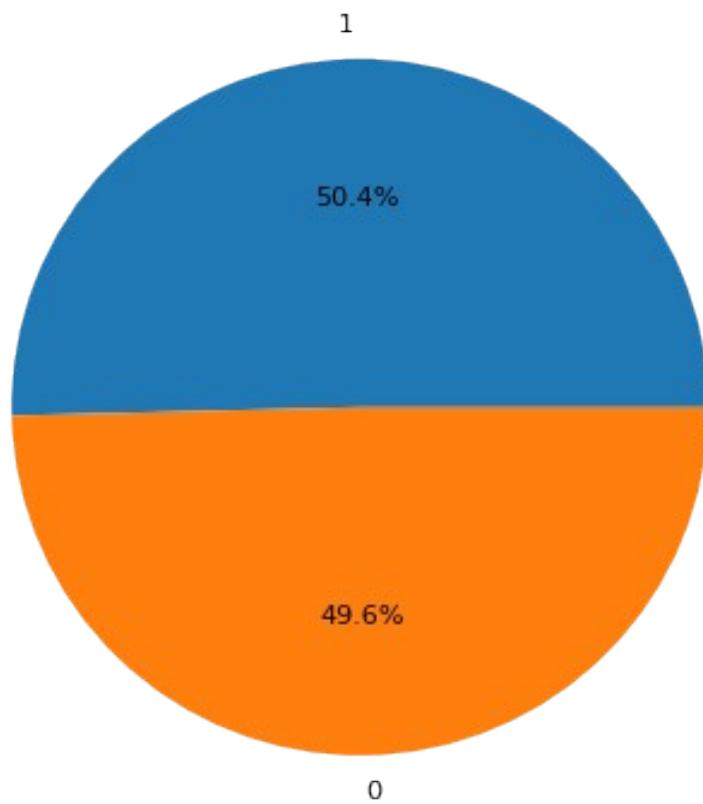
```
fig.tight_layout()  
plt.show()
```



```
fig, ax = plt.subplots(figsize=(10, 6), dpi=80)  
fig.suptitle('MRNet', fontsize=14)  
ax.pie(x=full_mrnet_df['ACL'].value_counts(),  
       labels=full_mrnet_df['ACL'].value_counts().index,  
       autopct='%.1f%%')  
ax.set_title('Pie Chart of reduced ACL Class Imbalance', pad=10)  
plt.show()
```

MRNet

Pie Chart of reduced ACL Class Imbalance



E) KneeMRI EDA (KneeMRI Exploratory Data Analysis)

```
# Explore the directory structure and files in the dataset MRNet
!tree -L 1 Data/KneeMRI/
```

```
Data/KneeMRI/
├── metadata-aug.csv
├── metadata.csv
└── vol01
└── vol02
└── vol03
└── vol04
└── vol05
└── vol06
└── vol07
└── vol08
└── vol09
└── vol10
```

11 directories, 2 files

```

import pickle
import platform
from glob import glob

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# Directory where the volumetric data is located
volumetric_data_dir = 'Data/KneeMRI'

# Path to metadata csv file
metadata_csv_path = 'Data/KneeMRI/metadata.csv'

kneemri_labels = {0: 'healthy', 1: 'partially ruptured', 2: 'completely ruptured'}

def plot_slices_per_exam(data_paths):
    """
    This function plots the distribution of slices found in MRIs of the KneeMRI dataset.

    Args:
        data_paths (list): List of the directories in KneeMRI dataset
    """
    num_slices_per_exam = []
    for path in data_paths:
        for exam in glob(path + "/*.pck"):
            with open(exam, 'rb') as file_handler: # Must use 'rb' as the data is binary
                mri_vol = pickle.load(file_handler)
                num_slices_per_exam.append(mri_vol.shape[0])
    fig, ax = plt.subplots(figsize=(8, 6), dpi=80)
    fig.suptitle('KneeMRI', fontsize=14)
    num_slices = np.asarray(num_slices_per_exam)
    print(f"For KneeMRI Sagittal plane min : {num_slices.min()}, max : {num_slices.max()}, avg : {num_slices.mean()}")
    sns.histplot(num_slices, stat='density', ax=ax, kde=True)
    ax.set_title("Sagittal Plane Slices Distribution", pad=15)
    plt.show()

mri_vol_paths = glob(volumetric_data_dir + "/vol*")
mri_vol_paths.sort()
mri_vol_paths

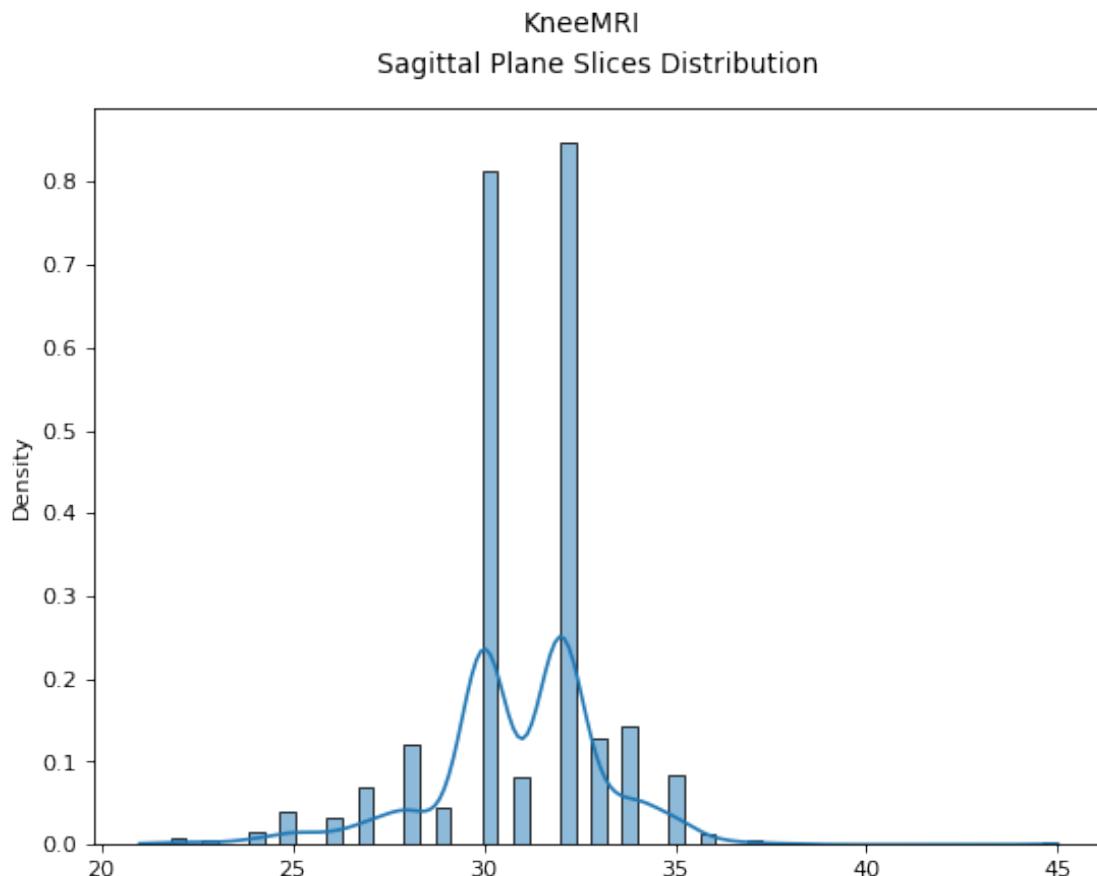
['Data/KneeMRI\\vol01',
 'Data/KneeMRI\\vol02',
 'Data/KneeMRI\\vol03',
 'Data/KneeMRI\\vol04',
 'Data/KneeMRI\\vol05',

```

```
'Data/KneeMRI\\vol06',  
'Data/KneeMRI\\vol07',  
'Data/KneeMRI\\vol08',  
'Data/KneeMRI\\vol09',  
'Data/KneeMRI\\vol10']
```

plot_slices_per_exam(mri_vol_paths)

For KneeMRI Sagittal plane min : 21, max : 45, avg : 30.925845147219192



```
# names=True loads the interprets the first row of csv file as column names
# 'i4' = 4 byte signed integer, 'U20' = unicode max 20 char string
metadata = np.genfromtxt(metadata_csv_path, delimiter=',', names=True,
                        dtype='i4,i4,i4,i4,i4,i4,i4,i4,i4,U20')
```

```
metadata_df = pd.DataFrame(metadata)
```

metadata df

```

912 1027212      5           1           1   113   127   16   101
913 1028019      5           1           1   105   102   14    95
914 1028028      5           0           0   118   84    15   100
915 1028069      5           0           0   105   97    15   103
916 1028670      5           1           0   113   108   14   103

  roiWidth  roiDepth volumeFilename
0       72          3   329637-8.pck
1       98          6   390116-9.pck
2      115          2   404663-8.pck
3       80          3   406320-9.pck
4       98          4   412857-8.pck
..     ...
912      99          3 1027212-5.pck
913     100          3 1028019-5.pck
914     100          2 1028028-5.pck
915     106          4 1028069-5.pck
916     110          4 1028670-5.pck

```

[917 rows x 11 columns]

```
metadata_df[metadata['examId'] == 329637]
```

	examId	seriesNo	aclDiagnosis	kneeLR	roiX	roiY	roiZ	roiHeight	\
0	329637	8	0	1	139	184	14	74	
0	roiWidth	roiDepth	volumeFilename						
0	72	3	329637-8.pck						

```
metadata_df['aclDiagnosis'].value_counts()
```

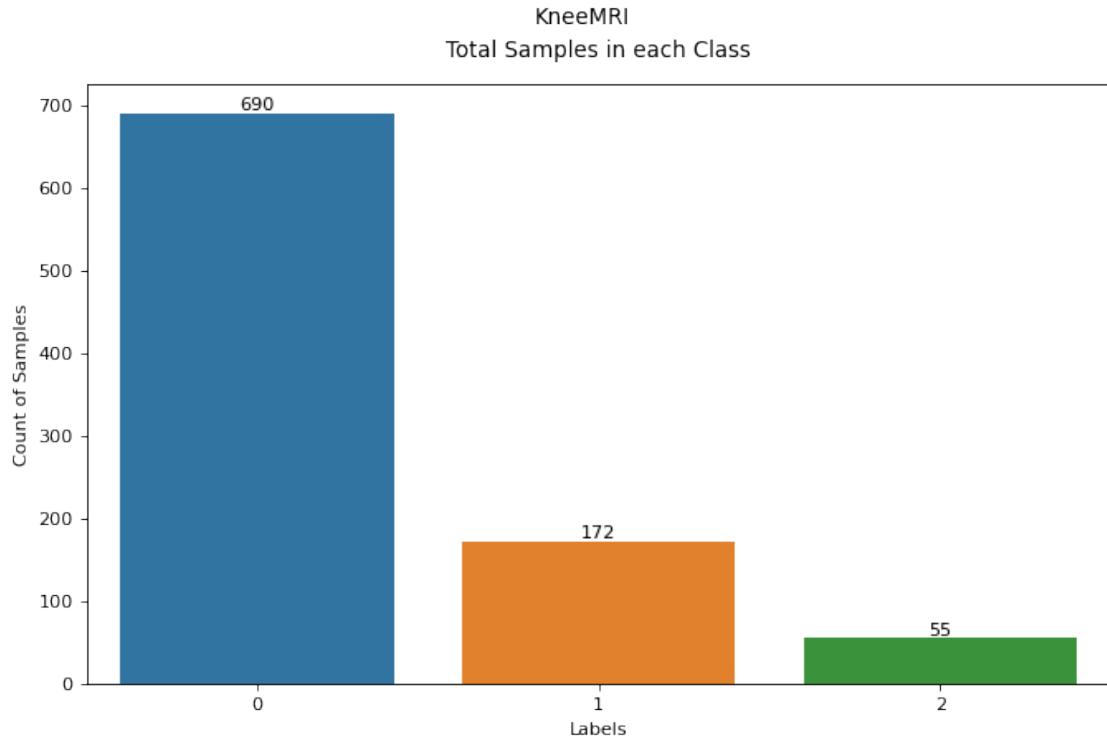
0	690
1	172
2	55

Name: aclDiagnosis, dtype: int64

```

fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
fig.suptitle('KneeMRI', fontsize=14)
ax = sns.barplot(x=metadata_df['aclDiagnosis'].value_counts().index,
                  y=metadata_df['aclDiagnosis'].value_counts(),
                  ax=ax)
ax.bar_label(ax.containers[0])
ax.set_xlabel('Labels')
ax.set_ylabel('Count of Samples')
ax.set_title('Total Samples in each Class', pad=15)
plt.show()

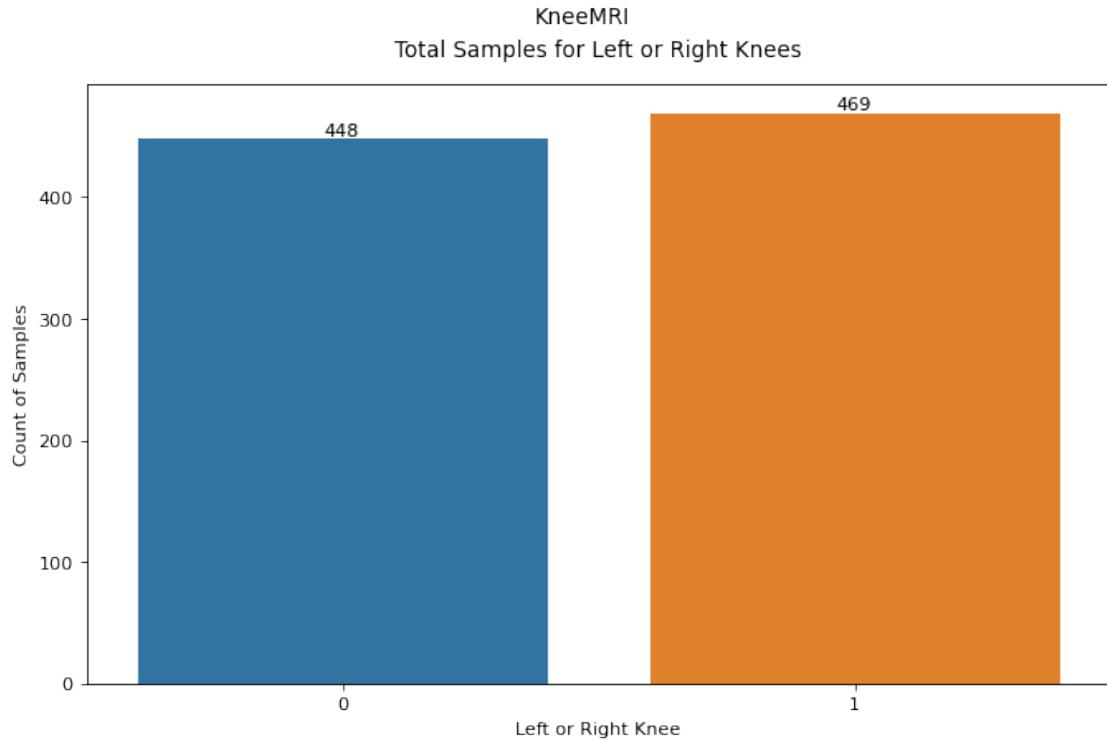
```



```
metadata_df['kneeLR'].value_counts()
```

```
1    469
0    448
Name: kneeLR, dtype: int64

fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
fig.suptitle('KneeMRI', fontsize=14)
ax = sns.barplot(x=metadata_df['kneeLR'].value_counts().index,
                  y=metadata_df['kneeLR'].value_counts(),
                  ax=ax)
ax.bar_label(ax.containers[0])
ax.set_xlabel('Left or Right Knee')
ax.set_ylabel('Count of Samples')
ax.set_title('Total Samples for Left or Right Knees', pad=15)
plt.show()
```



```
metadata_df['aclDiagnosis'].value_counts(normalize=True)*100
```

```
0    75.245365
1    18.756816
2     5.997819
Name: aclDiagnosis, dtype: float64
```

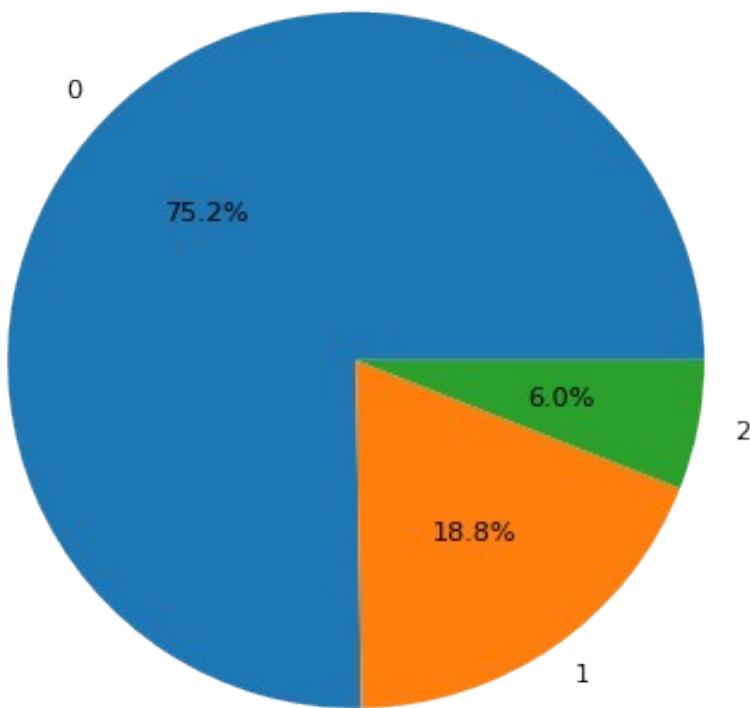
```
metadata_df['kneeLR'].value_counts(normalize=True)
```

```
1    0.51145
0    0.48855
Name: kneeLR, dtype: float64
```

```
fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
fig.suptitle('KneeMRI', fontsize=14)
ax.pie(x=metadata_df['aclDiagnosis'].value_counts(),
       labels=metadata_df['aclDiagnosis'].value_counts().index,
       autopct='%.1f%%')
ax.set_title('Pie Chart of Class Imbalance', pad=10)
plt.show()
```

KneeMRI

Pie Chart of Class Imbalance



```
# Directory where the preprocessed volumetric data is located
preprocessed_kneemri_data_dir = 'Preprocessed_Data/KneeMRI'
# path to metadata csv file
kneemri_metadata_csv_path = 'Data/KneeMRI/metadata.csv'
aug_metadata_csv_path = 'Data/KneeMRI/metadata-aug.csv'

# For running code on Windows
if platform.system() == "Windows":
    preprocessed_kneemri_data_dir = preprocessed_kneemri_data_dir.replace('/', '\\')
    kneemri_metadata_csv_path = kneemri_metadata_csv_path.replace('/', '\\')
    aug_metadata_csv_path = aug_metadata_csv_path.replace('/', '\\')

if platform.system() == "Windows":
    mri_vol_paths = glob(preprocessed_kneemri_data_dir + "\\vol*")
else:
    mri_vol_paths = glob(preprocessed_kneemri_data_dir + "/vol*")
mri_vol_paths.sort()

cases = []
for mri_data_path in mri_vol_paths:
```

```

if platform.system() == "Windows":
    all_exams = glob(mri_data_path + "\\\*.npy")
else:
    all_exams = glob(mri_data_path + "/*.npy")
all_exams.sort()
cases.extend(all_exams)

aug_cases = []
for mri_data_path in mri_vol_paths:
    if platform.system() == "Windows":
        all_exams = glob(mri_data_path + "\\\\aug\\\\*.npy")
    else:
        all_exams = glob(mri_data_path + "/aug/*.npy")
    all_exams.sort()
    aug_cases.extend(all_exams)

filenames = []
filenames.extend(cases)
filenames.extend(aug_cases)
filenames.sort()

len(filenames)
1567

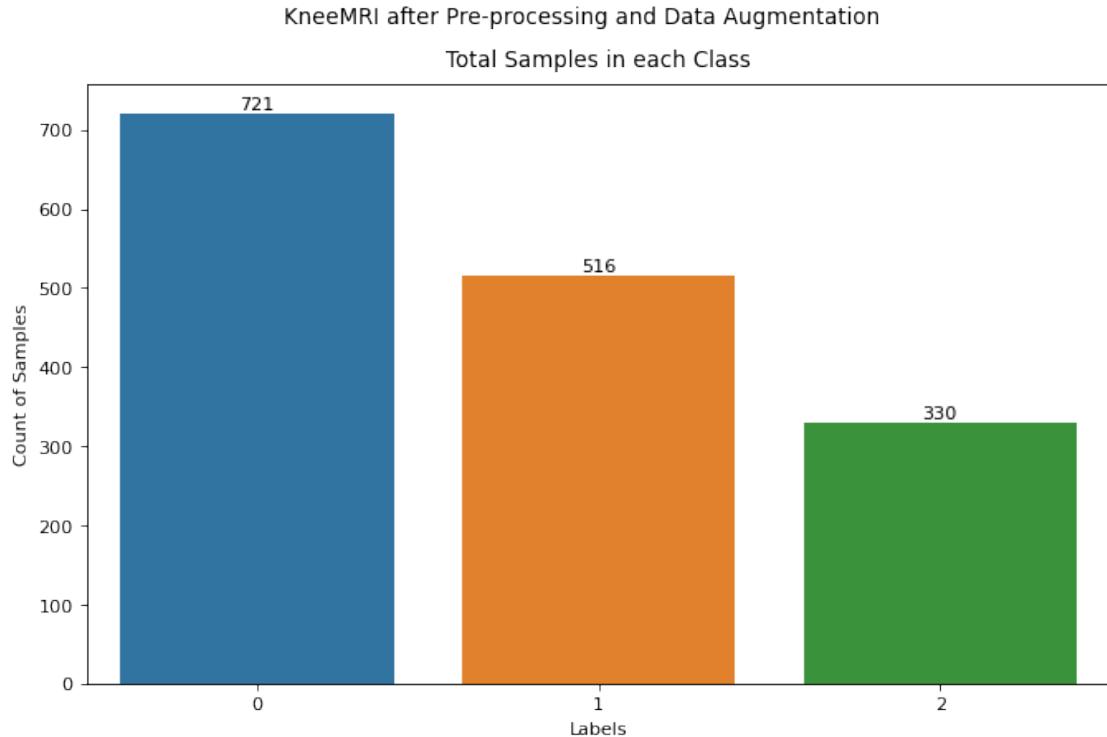
# names=True loads the interprets the first row of csv file as column names
# 'i4' = 4 byte signed integer, 'U20' = unicode max 20 char string
metadata = np.genfromtxt(kneemri_metadata_csv_path, delimiter=',', names=True,
                        dtype='i4,i4,i4,i4,i4,i4,i4,i4,i4,i4,U20')

metadata_df = pd.DataFrame(metadata)
aug_metadata_df = pd.read_csv(aug_metadata_csv_path, index_col=0)
full_metadata_df = pd.concat([metadata_df, aug_metadata_df],
                             ignore_index=True)

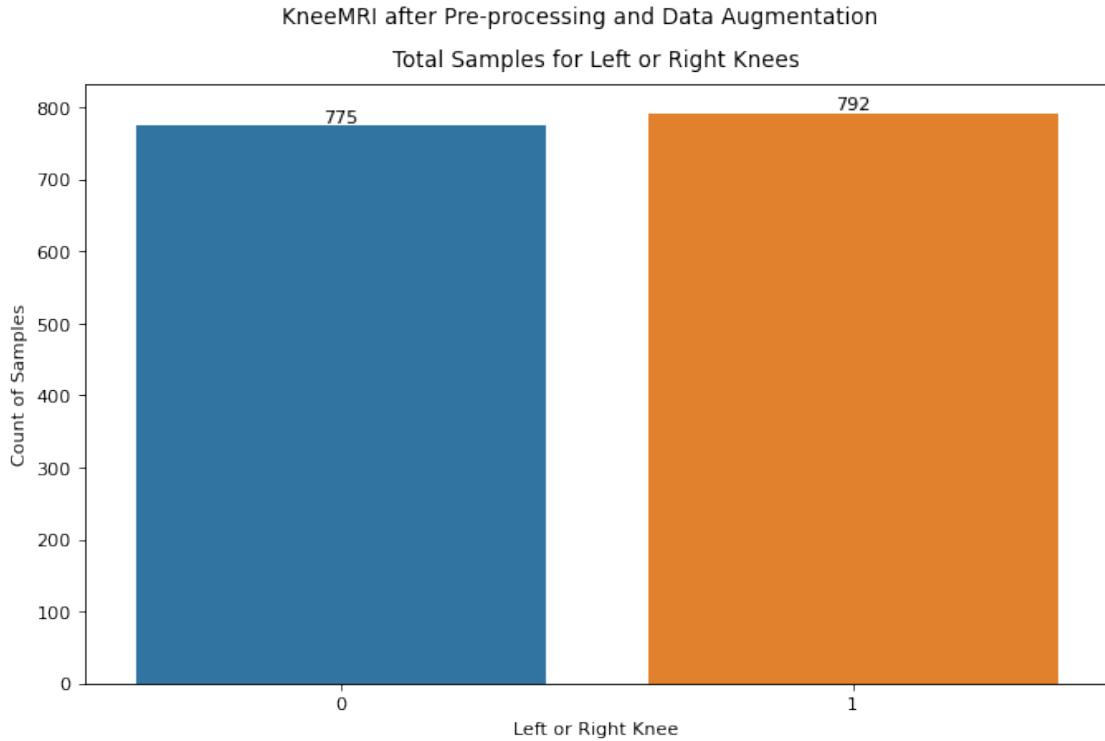
len(full_metadata_df)
1567

fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
fig.suptitle('KneeMRI after Pre-processing and Data Augmentation',
             fontsize=14)
ax = sns.barplot(x=full_metadata_df['aclDiagnosis'].value_counts().index,
                  y=full_metadata_df['aclDiagnosis'].value_counts(),
                  ax=ax)
ax.bar_label(ax.containers[0])
ax.set_xlabel('Labels')
ax.set_ylabel('Count of Samples')
ax.set_title('Total Samples in each Class', pad=10)
plt.show()

```



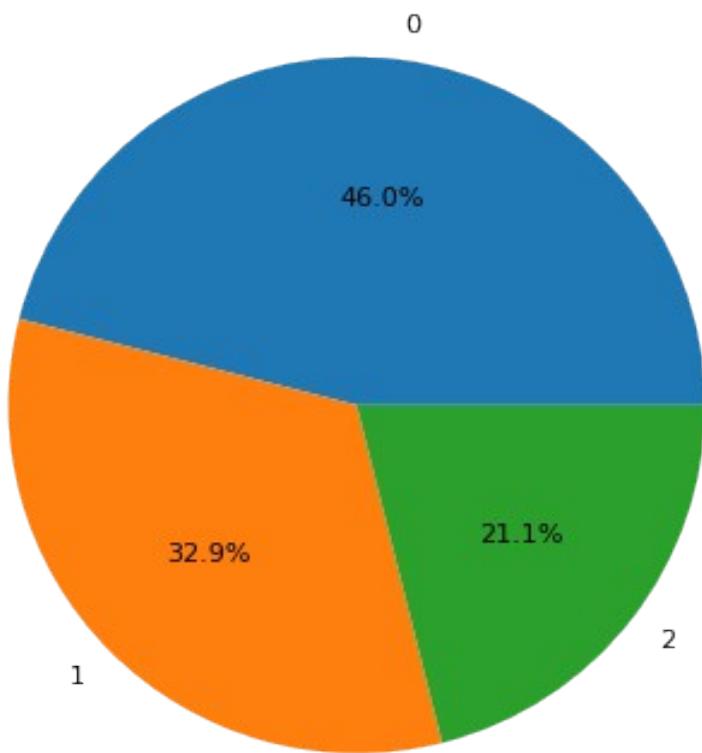
```
fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
fig.suptitle('KneeMRI after Pre-processing and Data Augmentation',
             fontsize=14)
ax = sns.barplot(x=full_metadata_df['kneeLR'].value_counts().index,
                  y=full_metadata_df['kneeLR'].value_counts(),
                  ax=ax)
ax.bar_label(ax.containers[0])
ax.set_xlabel('Left or Right Knee')
ax.set_ylabel('Count of Samples')
ax.set_title('Total Samples for Left or Right Knees', pad=10)
plt.show()
```



```
fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
fig.suptitle('KneeMRI after Pre-processing and Data Augmentation',
             fontsize=14)
ax.pie(x=full_metadata_df['aclDiagnosis'].value_counts(),
        labels=full_metadata_df['aclDiagnosis'].value_counts().index,
        autopct='%.1f%%')
ax.set_title('Pie Chart of reduced Class Imbalance', pad=10)
plt.show()
```

KneeMRI after Pre-processing and Data Augmentation

Pie Chart of reduced Class Imbalance



F) Keras Models

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.layers import (BatchNormalization, Conv3D, Dense,
                                     Dropout, Flatten, GlobalAveragePooling3D,
                                     MaxPooling3D)
from tensorflow.keras.models import Model, Sequential
from tensorflow.python.client import device_lib

print(f"\nTensorflow version : {tf.__version__}")

Tensorflow version : 2.10.1

print(f"\nTensorflow devices available : \n{device_lib.list_local_devices()}")
```

```

Tensorflow devices available :
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 7255090747541413501
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 22402342912
locality {
  bus_id: 1
  links {
  }
}
incarnation: 7704950245629368983
physical_device_desc: "device: 0, name: NVIDIA RTX A5000, pci bus id: 0000:61:00.0, compute capability: 8.6"
xla_global_id: 416903419
]

print(f"\nTensorflow physical devices available : \n{tf.config.list_physical_devices()}\n")

Tensorflow physical devices available :
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]

def mri_model_1(model_name, num_classes, depth=30, width=256, height=256):
    """Summary

Args:
    model_name (str): Model name
    num_classes (int): Number of classes
    depth (int, optional): Depth of MRI volume
    width (int, optional): Width of MRI volume
    height (int, optional): Height of MRI volume

Returns:
    TYPE: Keras Model
    """
    # Determine the number of units and activation function of the last layer
    # based on the input number of classes
    if num_classes == 2:
        last_layer_units = 1
        last_layer_activation = 'sigmoid'
    elif num_classes > 2:
        last_layer_units = num_classes
        last_layer_activation = 'softmax'

    model = Sequential([

```

```

        Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu',
input_shape=(depth, width, height, 1)),
        MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
        BatchNormalization(),
        Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
        MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
        BatchNormalization(),
        Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
        MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
        BatchNormalization(),
        Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
        MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
        BatchNormalization(),
        Conv3D(256, kernel_size=(3, 3, 3), padding="same", activation='relu'),
        MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
        BatchNormalization(),
        GlobalAveragePooling3D(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(last_layer_units, activation=last_layer_activation)
    ], name=model_name)

    return model

```

```

def mri_model_2(model_name, num_classes, depth=30, width=256, height=256):
    """Summary

```

Args:

```

    model_name (str): Model name
    num_classes (int): Number of classes
    depth (int, optional): Depth of MRI volume
    width (int, optional): Width of MRI volume
    height (int, optional): Height of MRI volume

```

Returns:

```

    TYPE: Keras Model
"""

```

```

# Determine the number of units and activation function of the last layer
# based on the input number of classes

```

```

if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes
    last_layer_activation = 'softmax'

```

```

model = Sequential([
    Conv3D(16, kernel_size=(3, 3, 3), padding="same", activation='relu',
input_shape=(depth, width, height, 1)),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),

```

```

    BatchNormalization(),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(256, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    GlobalAveragePooling3D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model

```

```
def mri_model_3(model_name, num_classes, depth=30, width=256, height=256):
    """Summary
```

Args:

```

    model_name (str): Model name
    num_classes (int): Number of classes
    depth (int, optional): Depth of MRI volume
    width (int, optional): Width of MRI volume
    height (int, optional): Height of MRI volume

```

Returns:

```
    TYPE: Keras Model
```

```
"""
```

```

# Determine the number of units and activation function of the last layer
# based on the input number of classes
if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes

```

```

last_layer_activation = 'softmax'

model = Sequential([
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu',
input_shape=(depth, width, height, 1)),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(256, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(256, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    GlobalAveragePooling3D(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(512, activation='relu'),
    Dropout(0.2),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model

def mri_model_4(model_name, num_classes, depth=30, width=256, height=256):
    """Summary

```

Args:

```
model_name (str): Model name
num_classes (int): Number of classes
depth (int, optional): Depth of MRI volume
width (int, optional): Width of MRI volume
height (int, optional): Height of MRI volume
```

Returns:

```
TYPE: Keras Model
```

```
"""
```

```
# Determine the number of units and activation function of the last layer
# based on the input number of classes
if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes
    last_layer_activation = 'softmax'

model = Sequential([
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu',
input_shape=(depth, width, height, 1)),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Dropout(0.6),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model
```

```
def mri_model_5(model_name, num_classes, depth=30, width=256, height=256):
    """Summary
```

Args:

```
model_name (str): Model name
num_classes (int): Number of classes
depth (int, optional): Depth of MRI volume
width (int, optional): Width of MRI volume
height (int, optional): Height of MRI volume
```

Returns:

```

    TYPE: Keras Model
"""

# Determine the number of units and activation function of the last layer
# based on the input number of classes
if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes
    last_layer_activation = 'softmax'

model = Sequential([
    Conv3D(16, kernel_size=(3, 3, 3), padding="same", activation='relu',
input_shape=(depth, width, height, 1)),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(16, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(256, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Dropout(0.5),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.4),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model

```

```

def mri_model_6(model_name, num_classes, depth=30, width=256, height=256):
    """Summary

```

Args:

```

    model_name (str): Model name
    num_classes (int): Number of classes

```

```

depth (int, optional): Depth of MRI volume
width (int, optional): Width of MRI volume
height (int, optional): Height of MRI volume

Returns:
    TYPE: Keras Model
"""

# Determine the number of units and activation function of the last layer
# based on the input number of classes
if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes
    last_layer_activation = 'softmax'

model = Sequential([
    Conv3D(8, kernel_size=(3, 3, 3), padding="same", activation='relu',
input_shape=(depth, width, height, 1)),
    Conv3D(16, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(16, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    Conv3D(16, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(16, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    Conv3D(16, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(32, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(64, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(128, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    BatchNormalization(),
    Conv3D(256, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Conv3D(256, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),

```

```

    Conv3D(256, kernel_size=(3, 3, 3), padding="same", activation='relu'),
    MaxPooling3D(pool_size=(2, 2, 2), padding="same"),
    Dropout(0.5),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.3),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model

def mri_model_tf_3_vgg(model_name, num_classes, depth=30, width=256,
height=256):
    """Summary

Args:
    model_name (str): Model name
    num_classes (int): Number of classes
    depth (int, optional): Depth of MRI volume
    width (int, optional): Width of MRI volume
    height (int, optional): Height of MRI volume

Returns:
    TYPE: Keras VGG16 Model for Transfer Learning with 3 channels
"""
# Determine the number of units and activation function of the last layer
# based on the input number of classes
if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes
    last_layer_activation = 'softmax'

# Import VGG16 model with ImageNet weights and specified input shape with
three channels
vgg_model = VGG16(include_top=False, weights='imagenet',
input_shape=(width, height, 3))

# Set VGG16 model as non-trainable
vgg_model.trainable = False

model = Sequential([
    vgg_model,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.4),
    Dense(256, activation='relu'),

```

```

        Dropout(0.2),
        Dense(128, activation='relu'),
        Dropout(0.1),
        Dense(last_layer_units, activation=last_layer_activation)
    ], name=model_name)

return model

def mri_model_tf_3_xception(model_name, num_classes, depth=30, width=256,
height=256):
    """Summary

Args:
    model_name (str): Model name
    num_classes (int): Number of classes
    depth (int, optional): Depth of MRI volume
    width (int, optional): Width of MRI volume
    height (int, optional): Height of MRI volume

Returns:
    TYPE: Keras Xception Model for Transfer Learning with 3 channels
"""
# Determine the number of units and activation function of the last layer
# based on the input number of classes
if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes
    last_layer_activation = 'softmax'

# Import Xception model with ImageNet weights and specified input shape
with three channels
xception_model = Xception(include_top=False, weights='imagenet',
input_shape=(width, height, 3))

# Set Xception model as non-trainable
xception_model.trainable = False

model = Sequential([
    xception_model,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.4),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model

```

```

def mri_model_tf_3_resnet(model_name, num_classes, depth=30, width=256,
height=256):
    """Summary

Args:
    model_name (str): Model name
    num_classes (int): Number of classes
    depth (int, optional): Depth of MRI volume
    width (int, optional): Width of MRI volume
    height (int, optional): Height of MRI volume

Returns:
    TYPE: Keras ResNet50 Model for Transfer Learning with 3 channels
"""
# Determine the number of units and activation function of the last layer
# based on the input number of classes
if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes
    last_layer_activation = 'softmax'

# Import ResNet50 model with ImageNet weights and specified input shape
# with three channels
resnet_model = ResNet50(include_top=False, weights='imagenet',
input_shape=(width, height, 3))

# Set ResNet50 model as non-trainable
resnet_model.trainable = False

model = Sequential([
    resnet_model,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.4),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model

def avg_and_copy_wts(weights, num_channels_to_fill):
    """
Function to calculate average weights along axes channels and copy to
remaning n-channels that need to be filled

Args:
    weights (TYPE): Keras model weights
"""

```

```

    num_channels_to_fill (TYPE): Number of channels to fill with the
average weights

Returns:
    TYPE: Returns the new weights after filling n-channels with average of
weights
"""
average_weights = np.mean(weights, axis=-2).reshape(weights[:, :, -1:, :].shape) # Average along the second to last channel axis channel
wts_copied_to_mult_channels = np.tile(average_weights,
(num_channels_to_fill, 1)) # Repeat the array n-times
return(wts_copied_to_mult_channels)

def mri_model_tf_5_vgg(model_name, num_classes, depth=30, width=256,
height=256):
    """Summary

Args:
    model_name (str): Model name
    num_classes (int): Number of classes
    depth (int, optional): Depth of MRI volume
    width (int, optional): Width of MRI volume
    height (int, optional): Height of MRI volume

Returns:
    TYPE: Keras VGG16 Model for Transfer Learning with 5 channels
"""
# Determine the number of units and activation function of the last layer
# based on the input number of classes
if num_classes == 2:
    last_layer_units = 1
    last_layer_activation = 'sigmoid'
elif num_classes > 2:
    last_layer_units = num_classes
    last_layer_activation = 'softmax'

# Import VGG16 model without input shape
vgg_model = VGG16(include_top=False, weights='imagenet')

# Get config dictionary for VGG16
vgg_config = vgg_model.get_config()

# Set input shape to new desired shape
h, w, c = height, width, 5
vgg_config["layers"][0]["config"]["batch_input_shape"] = (None, h, w, c)

# Get new model with the updated configuration
vgg_updated = Model.from_config(vgg_config)

# Get config. for the updated model
vgg_updated_config = vgg_updated.get_config()
# Extract layer names

```

```

    vgg_updated_layer_names = [vgg_updated_config['layers'][x]['name'] for x
in range(len(vgg_updated_config['layers']))]

# Grab the first conv block name
first_conv_name = vgg_updated_layer_names[1]

# Update weights for all layers, for the first conv layer average first
three channel weights to the
# remaning number out of total channels
for layer in vgg_model.layers:
    if layer.name in vgg_updated_layer_names:

        if layer.get_weights() != []: # Check if the layer has weights
            target_layer = vgg_updated.get_layer(layer.name)

            if layer.name in first_conv_name: # First conv block
                weights = layer.get_weights()[0]
                biases = layer.get_weights()[1]

                # Adjust weights of the extra channels
                weights_extra_channels = np.concatenate((weights,
avg_and_copy_wts(weights, c - 3)),
                                            axis=-2)
                # Set weights for the first conv block
                target_layer.set_weights([weights_extra_channels, biases])
                target_layer.trainable = False # Set the layer as non-
trainable
            else:
                # Set weights for other layers
                target_layer.set_weights(layer.get_weights())
                target_layer.trainable = False # Set the layer as non-
trainable

model = Sequential([
    vgg_updated,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.4),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model

def mri_model_tf_5_resnet(model_name, num_classes, depth=30, width=256,
height=256):
    """Summary

```

Args:

```
    model_name (str): Model name  
    num_classes (int): Number of classes  
    depth (int, optional): Depth of MRI volume  
    width (int, optional): Width of MRI volume  
    height (int, optional): Height of MRI volume
```

Returns:

```
    TYPE: Keras ResNet50 Model for Transfer Learning with 5 channels
```

```
"""
```

```
# Determine the number of units and activation function of the last layer  
# based on the input number of classes
```

```
if num_classes == 2:  
    last_layer_units = 1  
    last_layer_activation = 'sigmoid'  
elif num_classes > 2:  
    last_layer_units = num_classes  
    last_layer_activation = 'softmax'
```

```
# Import ResNet50 model without input shape
```

```
resnet_model = ResNet50(include_top=False, weights='imagenet')
```

```
# Get config dictionary for ResNet50
```

```
resnet_config = resnet_model.get_config()
```

```
# Set input shape to new desired shape
```

```
h, w, c = height, width, 5
```

```
resnet_config["layers"][0]["config"]["batch_input_shape"] = (None, h, w,
```

```
c)
```

```
# Get new model with the updated configuration
```

```
resnet_updated = Model.from_config(resnet_config)
```

```
# Get config. for the updated model
```

```
resnet_updated_config = resnet_updated.get_config()
```

```
# Extract layer names
```

```
resnet_updated_layer_names = [resnet_updated_config['layers'][x]['name']  
for x in range(len(resnet_updated_config['layers']))]
```

```
# Grab the first conv block name
```

```
first_conv_name = resnet_updated_layer_names[2]
```

```
# Update weights for all layers, for the first conv layer average first  
three channel weights to the
```

```
# remaning number out of total channels
```

```
for layer in resnet_model.layers:
```

```
    if layer.name in resnet_updated_layer_names:
```

```
        if layer.get_weights() != []: # Check if the layer has weights  
            target_layer = resnet_updated.get_layer(layer.name)
```

```
        if layer.name in first_conv_name: # First conv block
```

```

weights = layer.get_weights()[0]
biases = layer.get_weights()[1]

# Adjust weights of the extra channels
weights_extra_channels = np.concatenate((weights,
avg_and_copy_wts(weights, c - 3)),
axis=-2)

# Set weights for the first conv block
target_layer.set_weights([weights_extra_channels, biases])
target_layer.trainable = False # Set the layer as non-
trainable
else:
    # Set weights for other layers
    target_layer.set_weights(layer.get_weights())
    target_layer.trainable = False # Set the layer as non-
trainable

model = Sequential([
    resnet_updated,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.4),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(last_layer_units, activation=last_layer_activation)
], name=model_name)

return model

```

G) MRNet Model Training

```

import os
import platform
import pandas as pd
import numpy as np
from glob import glob

from tensorflow import keras
from sklearn.model_selection import train_test_split
import utils
import models

```

Tensorflow version : 2.10.1

Tensorflow devices available :
[name: "/device:CPU:0"

```

device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 15771189179446765976
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 22439854080
locality {
  bus_id: 1
  links {
  }
}
incarnation: 8292459354870456221
physical_device_desc: "device: 0, name: NVIDIA RTX A5000, pci bus id: 0000:61:00.0, compute capability: 8.6"
xla_global_id: 416903419
]

Tensorflow physical devices available :
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]

mrnet_dataset_dir = 'Data/MRNet-v1.0'
mrnet_train_path = os.path.join(mrnet_dataset_dir, 'train')
mrnet_valid_path = os.path.join(mrnet_dataset_dir, 'valid')

mrnet_preprocessed_dataset_dir = 'Preprocessed_Data/MRNet-v1.0'
mrnet_preprocessed_train_path = os.path.join(mrnet_preprocessed_dataset_dir,
'train')
mrnet_preprocessed_valid_path = os.path.join(mrnet_preprocessed_dataset_dir,
'valid')

mrnet_planes = ['axial', 'coronal', 'sagittal']

# For running code on Windows
if platform.system() == "Windows":
    mrnet_dataset_dir = mrnet_dataset_dir.replace('/', '\\')
    mrnet_train_path = mrnet_train_path.replace('/', '\\')
    mrnet_valid_path = mrnet_valid_path.replace('/', '\\')

    mrnet_preprocessed_dataset_dir =
    mrnet_preprocessed_dataset_dir.replace('/', '\\')
    mrnet_preprocessed_train_path = mrnet_preprocessed_train_path.replace('/',
'\\')
    mrnet_preprocessed_valid_path = mrnet_preprocessed_valid_path.replace('/',
'\\')

mrnet_datasets = { 'train' : mrnet_train_path, 'valid' : mrnet_valid_path}
mrnet_classes = ['abnormal', 'acl', 'meniscus']

```

```

# TRAIN DATASET
for label in mrnet_classes:
    if platform.system() == "Windows":
        if label == 'abnormal':
            train_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}\train-{label}.csv",
                                            header=None,
                                            names=['Case', 'Abnormal'],
                                            dtype={'Case':str,
                                                   'Abnormal':np.int64})
        elif label == 'acl':
            train_acl_df = pd.read_csv(f"{mrnet_dataset_dir}\train-{label}.csv",
                                       header=None,
                                       names=['Case', 'ACL'],
                                       dtype={'Case':str,
                                              'ACL':np.int64})
        if label == 'meniscus':
            train_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}\train-{label}.csv",
                                             header=None,
                                             names=['Case', 'Meniscus'],
                                             dtype={'Case':str,
                                                    'Meniscus':np.int64})
    else:
        if label == 'abnormal':
            train_abnormal_df = pd.read_csv(f'{mrnet_dataset_dir}/train-{label}.csv',
                                            header=None,
                                            names=['Case', 'Abnormal'],
                                            dtype={'Case':str,
                                                   'Abnormal':np.int64})
        elif label == 'acl':
            train_acl_df = pd.read_csv(f'{mrnet_dataset_dir}/train-{label}.csv',
                                       header=None,
                                       names=['Case', 'ACL'],
                                       dtype={'Case':str,
                                              'ACL':np.int64})
        if label == 'meniscus':
            train_meniscus_df = pd.read_csv(f'{mrnet_dataset_dir}/train-{label}.csv',
                                             header=None,
                                             names=['Case', 'Meniscus'],
                                             dtype={'Case':str,
                                                    'Meniscus':np.int64})
    'Meniscus':np.int64)
'meniscus':np.int64)

mrnet_train_df = pd.merge(train_abnormal_df, train_acl_df,
on='Case').merge(train_meniscus_df, on='Case')

mrnet_train_df

```



```

                    dtype={'Case':str,
'ACL':np.int64})
    if label == 'meniscus':
        valid_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Meniscus'],
                                         dtype={'Case':str,
'Meniscus':np.int64})

mrnet_valid_df = pd.merge(valid_abnormal_df, valid_acl_df,
on='Case').merge(valid_meniscus_df, on='Case')

mrnet_valid_df

```

	Case	Abnormal	ACL	Meniscus
0	1130	0	0	0
1	1131	0	0	0
2	1132	0	0	0
3	1133	0	0	0
4	1134	0	0	0
..
115	1245	1	1	1
116	1246	1	1	1
117	1247	1	0	1
118	1248	1	1	1
119	1249	1	0	1

[120 rows x 4 columns]

```

# AUGMENTED TRAIN LABELS
if platform.system() == "Windows":
    mrnet_train_aug_df = pd.read_csv(f"{mrnet_dataset_dir}\train-aug.csv",
                                      index_col=0,
                                      dtype={'Case':str, 'Abnormal':np.int64,
'ACL':np.int64, 'Meniscus':np.int64})
else:
    mrnet_train_aug_df = pd.read_csv(f"{mrnet_dataset_dir}/train-aug.csv",
                                      index_col=0,
                                      dtype={'Case':str, 'Abnormal':np.int64,
'ACL':np.int64, 'Meniscus':np.int64})

mrnet_train_aug_df

```

	Case	Abnormal	ACL	Meniscus
0	0001-aug-0	1	1	1
1	0001-aug-1	1	1	1
2	0001-aug-2	1	1	1
3	0018-aug-0	1	1	1
4	0018-aug-1	1	1	1
..
659	1117-aug-1	1	1	1
660	1117-aug-2	1	1	1

```

661 1129-aug-0      1    1      0
662 1129-aug-1      1    1      0
663 1129-aug-2      1    1      0

[664 rows x 4 columns]

# AUGMENTED VALID LABELS
if platform.system() == "Windows":
    mrnet_valid_aug_df = pd.read_csv(f"{mrnet_dataset_dir}\valid-aug.csv",
                                      index_col=0,
                                      dtype={'Case':str, 'Abnormal':np.int64,
'ACL':np.int64, 'Meniscus':np.int64})
else:
    mrnet_valid_aug_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-aug.csv",
                                      index_col=0,
                                      dtype={'Case':str, 'Abnormal':np.int64,
'ACL':np.int64, 'Meniscus':np.int64})

mrnet_valid_aug_df

   Case  Abnormal  ACL  Meniscus
0  1158-aug-0      1    0      0
1  1172-aug-0      1    1      1
2  1172-aug-1      1    1      1
3  1172-aug-2      1    1      1
4  1173-aug-0      1    1      1
..   ...
161 1246-aug-1     1    1      1
162 1246-aug-2     1    1      1
163 1248-aug-0     1    1      1
164 1248-aug-1     1    1      1
165 1248-aug-2     1    1      1

[166 rows x 4 columns]

# We are working only with Sagittal plane

# TRAIN
if platform.system() == "Windows":
    mrnet_sagittal_train_files = glob(mrnet_preprocessed_train_path+"\\"+
sagittal"\*.npy")
else:
    mrnet_sagittal_train_files =
glob(mrnet_preprocessed_train_path+"/sagittal/*.npy")
mrnet_sagittal_train_files.sort()

# VALID
if platform.system() == "Windows":
    mrnet_sagittal_valid_files = glob(mrnet_preprocessed_valid_path+"\\"+
sagittal"\*.npy")
else:
    mrnet_sagittal_valid_files =

```

```

glob(mrnet_preprocessed_valid_path+"/sagittal/*.npy")
mrnet_sagittal_valid_files.sort()

# AUGMENTED TRAIN
if platform.system() == "Windows":
    mrnet_sagittal_train_aug_files = glob(mrnet_preprocessed_train_path+"\\" sagittal\\aug\\*.npy")
else:
    mrnet_sagittal_train_aug_files =
glob(mrnet_preprocessed_train_path+"/sagittal/aug/*.npy")
mrnet_sagittal_train_aug_files.sort()

# AUGMENTED VALID
if platform.system() == "Windows":
    mrnet_sagittal_valid_aug_files = glob(mrnet_preprocessed_valid_path+"\\" sagittal\\aug\\*.npy")
else:
    mrnet_sagittal_valid_aug_files =
glob(mrnet_preprocessed_valid_path+"/sagittal/aug/*.npy")
mrnet_sagittal_valid_aug_files.sort()

print(len(mrnet_sagittal_train_files))
print(len(mrnet_sagittal_valid_files))
print(len(mrnet_sagittal_train_aug_files))
print(len(mrnet_sagittal_valid_aug_files))

1130
120
664
166

mrnet_filenames = []
mrnet_filenames.extend(mrnet_sagittal_train_files)
mrnet_filenames.extend(mrnet_sagittal_valid_files)
mrnet_filenames.extend(mrnet_sagittal_train_aug_files)
mrnet_filenames.extend(mrnet_sagittal_valid_aug_files)
mrnet_filenames.sort()

len(mrnet_filenames)

2080

len(mrnet_train_df)+len(mrnet_valid_df)+len(mrnet_train_aug_df)
+len(mrnet_valid_aug_df)

2080

mrnet_full_df = pd.concat([mrnet_train_df, mrnet_valid_df, mrnet_train_aug_df,
mrnet_valid_aug_df], ignore_index=True)

mrnet_full_df

```

	Case	Abnormal	ACL	Meniscus
0	0000	1	0	0

```

1      0001      1      1      1
2      0002      1      0      0
3      0003      1      0      1
4      0004      1      0      0
...
2075  1246-aug-1  1      1      1
2076  1246-aug-2  1      1      1
2077  1248-aug-0  1      1      1
2078  1248-aug-1  1      1      1
2079  1248-aug-2  1      1      1

[2080 rows x 4 columns]

mrnet_labels = utils.get_correct_labels_mrnet(mrnet_filenames, mrnet_full_df)

mrnet_filenames[:5]

['Preprocessed_Data\\MRNet-v1.0\\train\\sagittal\\0000.npy',
 'Preprocessed_Data\\MRNet-v1.0\\train\\sagittal\\0001.npy',
 'Preprocessed_Data\\MRNet-v1.0\\train\\sagittal\\0002.npy',
 'Preprocessed_Data\\MRNet-v1.0\\train\\sagittal\\0003.npy',
 'Preprocessed_Data\\MRNet-v1.0\\train\\sagittal\\0004.npy']

mrnet_labels[:5]

[0, 1, 0, 0, 0]

# Quick check of counts of samples for each case
[[x, mrnet_labels.count(x)] for x in set(mrnet_labels)]

[[0, 1032], [1, 1048]]

BATCH_SIZE = 8
EPOCHS = 100

# Splitting into train, test and validation

X, X_test, y, y_test = train_test_split(mrnet_filenames,
                                         mrnet_labels,
                                         test_size=0.1,
                                         random_state=610,
                                         shuffle=True,
                                         stratify=mrnet_labels)

X_train, X_valid, y_train, y_valid = train_test_split(X,
                                                       y,
                                                       train_size=0.7,
                                                       random_state=610,
                                                       shuffle=True,
                                                       stratify=y)

[[x, y_train.count(x)] for x in set(y_train)]

[[0, 650], [1, 660]]

```

```

[[x, y_valid.count(x)] for x in set(y_valid)]
[[0, 279], [1, 283]]
[[x, y_test.count(x)] for x in set(y_test)]
[[0, 103], [1, 105]]
mrnet_class_weights = utils.compute_class_weights(y_train)
mrnet_class_weights
{0: 1.0076923076923077, 1: 0.9924242424242424}

model_name = 'MRNet_Model1'
MRNet_Model1 = models.mri_model_1(model_name, 2)
MRNet_Model1.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
MRNet_Model1.summary()
Model: "MRNet_Model1"



| Layer (type)                                 | Output Shape             | Param # |
|----------------------------------------------|--------------------------|---------|
| conv3d (Conv3D)                              | (None, 30, 256, 256, 64) | 1792    |
| max_pooling3d (MaxPooling3D )                | (None, 15, 128, 128, 64) | 0       |
| batch_normalization (BatchN ormalization)    | (None, 15, 128, 128, 64) | 256     |
| conv3d_1 (Conv3D)                            | (None, 15, 128, 128, 64) | 110656  |
| max_pooling3d_1 (MaxPooling 3D)              | (None, 8, 64, 64, 64)    | 0       |
| batch_normalization_1 (Bathc hNormalization) | (None, 8, 64, 64, 64)    | 256     |
| conv3d_2 (Conv3D)                            | (None, 8, 64, 64, 128)   | 221312  |
| max_pooling3d_2 (MaxPooling 3D)              | (None, 4, 32, 32, 128)   | 0       |
| batch_normalization_2 (Bathc hNormalization) | (None, 4, 32, 32, 128)   | 512     |
| conv3d_3 (Conv3D)                            | (None, 4, 32, 32, 128)   | 442496  |
| max_pooling3d_3 (MaxPooling 3D)              | (None, 2, 16, 16, 128)   | 0       |


```

3D)

<i>batch_normalization_3</i> (Batch Normalization)	(None, 2, 16, 16, 128)	512
<i>conv3d_4</i> (Conv3D)	(None, 2, 16, 16, 256)	884992
<i>max_pooling3d_4</i> (MaxPooling 3D)	(None, 1, 8, 8, 256)	0
<i>batch_normalization_4</i> (Batch Normalization)	(None, 1, 8, 8, 256)	1024
<i>global_average_pooling3d</i> (Global Average Pooling 3D)	(None, 256)	0
<i>dense</i> (Dense)	(None, 512)	131584
<i>dropout</i> (Dropout)	(None, 512)	0
<i>dense_1</i> (Dense)	(None, 256)	131328
<i>dropout_1</i> (Dropout)	(None, 256)	0
<i>dense_2</i> (Dense)	(None, 1)	257

```
=====
Total params: 1,926,977
Trainable params: 1,925,697
Non-trainable params: 1,280
```

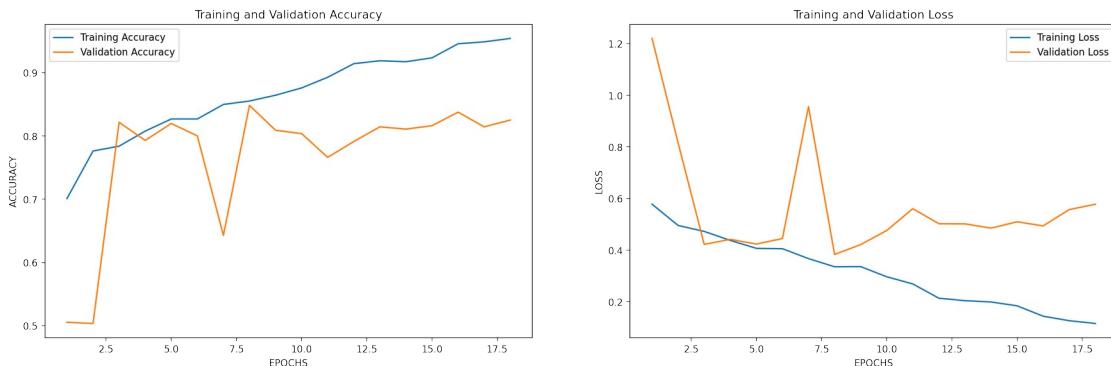
```
%%time
with tf.device('/device:GPU:0'):
    history = MRNet_Model1.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EP0CHS,
                                validation_data=utils.batch_generator(X_valid,
y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=mrnet_class_weights,
                                verbose=1,
                                callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()])

Epoch 1/100
163/163 [=====] - 205s 893ms/step - loss: 0.5778 - accuracy: 0.7009 - val_loss: 1.2201 - val_accuracy: 0.5054
Epoch 2/100
```

163/163 [=====] - 144s 882ms/step - loss: 0.4954 -
accuracy: 0.7761 - val_loss: 0.8152 - val_accuracy: 0.5036
Epoch 3/100
163/163 [=====] - 144s 884ms/step - loss: 0.4724 -
accuracy: 0.7837 - val_loss: 0.4222 - val_accuracy: 0.8214
Epoch 4/100
163/163 [=====] - 143s 880ms/step - loss: 0.4375 -
accuracy: 0.8075 - val_loss: 0.4416 - val_accuracy: 0.7929
Epoch 5/100
163/163 [=====] - 144s 883ms/step - loss: 0.4068 -
accuracy: 0.8267 - val_loss: 0.4234 - val_accuracy: 0.8196
Epoch 6/100
163/163 [=====] - 145s 887ms/step - loss: 0.4054 -
accuracy: 0.8267 - val_loss: 0.4447 - val_accuracy: 0.8000
Epoch 7/100
163/163 [=====] - 143s 879ms/step - loss: 0.3670 -
accuracy: 0.8497 - val_loss: 0.9559 - val_accuracy: 0.6429
Epoch 8/100
163/163 [=====] - 143s 880ms/step - loss: 0.3354 -
accuracy: 0.8551 - val_loss: 0.3831 - val_accuracy: 0.8482
Epoch 9/100
163/163 [=====] - 143s 879ms/step - loss: 0.3360 -
accuracy: 0.8643 - val_loss: 0.4213 - val_accuracy: 0.8089
Epoch 10/100
163/163 [=====] - 144s 882ms/step - loss: 0.2967 -
accuracy: 0.8758 - val_loss: 0.4760 - val_accuracy: 0.8036
Epoch 11/100
163/163 [=====] - 145s 888ms/step - loss: 0.2689 -
accuracy: 0.8926 - val_loss: 0.5605 - val_accuracy: 0.7661
Epoch 12/100
163/163 [=====] - 143s 880ms/step - loss: 0.2136 -
accuracy: 0.9141 - val_loss: 0.5022 - val_accuracy: 0.7911
Epoch 13/100
163/163 [=====] - 144s 882ms/step - loss: 0.2040 -
accuracy: 0.9187 - val_loss: 0.5017 - val_accuracy: 0.8143
Epoch 14/100
163/163 [=====] - 143s 879ms/step - loss: 0.1990 -
accuracy: 0.9172 - val_loss: 0.4851 - val_accuracy: 0.8107
Epoch 15/100
163/163 [=====] - 143s 879ms/step - loss: 0.1841 -
accuracy: 0.9233 - val_loss: 0.5096 - val_accuracy: 0.8161
Epoch 16/100
163/163 [=====] - 143s 880ms/step - loss: 0.1439 -
accuracy: 0.9456 - val_loss: 0.4938 - val_accuracy: 0.8375
Epoch 17/100
163/163 [=====] - 143s 880ms/step - loss: 0.1264 -
accuracy: 0.9486 - val_loss: 0.5570 - val_accuracy: 0.8143
Epoch 18/100
163/163 [=====] - ETA: 0s - loss: 0.1158 - accuracy:
0.9540Restoring model weights from the end of the best epoch: 8.
163/163 [=====] - 144s 882ms/step - loss: 0.1158 -
accuracy: 0.9540 - val_loss: 0.5777 - val_accuracy: 0.8250

```
Epoch 18: early stopping  
Wall time: 44min 6s
```

```
# Store history  
utils.store_model_history(model_name, history.history)  
  
# Plot training graphs  
utils.plot_acc_loss(history.history)
```



```
# Evaluate model  
X_test_prob = MRNet_Model1.predict(utils.predict_batch_generator(X_test,  
BATCH_SIZE))  
  
# Now get the correct labels based on the optimal threshold  
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)  
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')  
  
utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),  
['Healthy', 'Tear'])
```

26/26 [=====] - 12s 487ms/step

Best cutoff Threshold = 0.315281, F-Score = 0.836

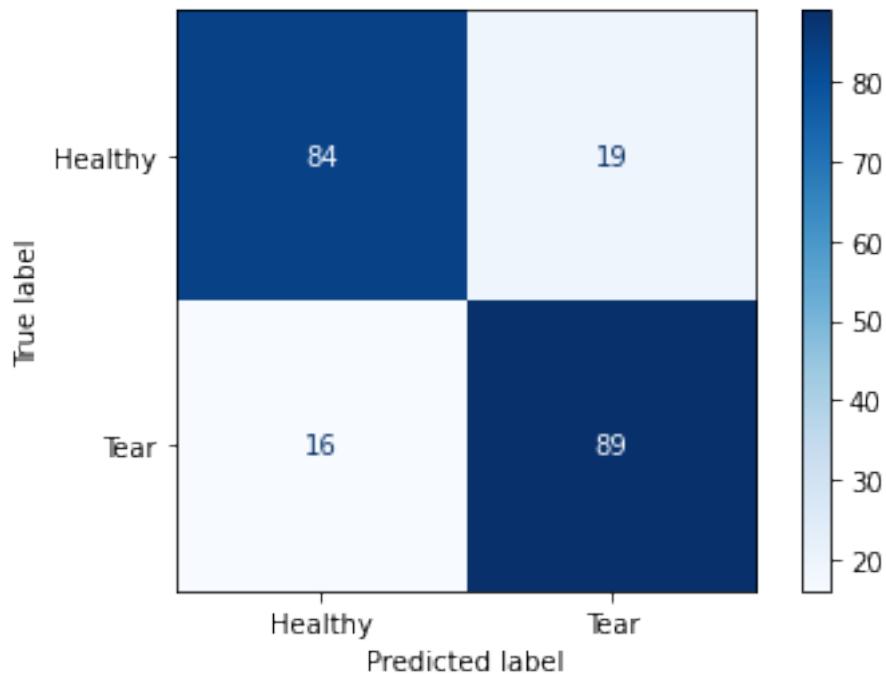
Evaluation Metrics:

Balanced Accuracy : 0.83
Precision : 0.82
Recall : 0.85
F1 Score: 0.84
ROC AUC Score : 0.88

Classification report :

	precision	recall	f1-score	support
Healthy	0.84	0.82	0.83	103
Tear	0.82	0.85	0.84	105
accuracy			0.83	208
macro avg	0.83	0.83	0.83	208
weighted avg	0.83	0.83	0.83	208

Confusion Matrix :



```
model_name = 'MRNet_Model2'  
MRNet_Model2 = models.mri_model_2(model_name, 2)  
MRNet_Model2.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),  
                      loss='binary_crossentropy',  
                      metrics=['accuracy'])
```

```
MRNet_Model2.summary()
```

```
Model: "MRNet_Model2"
```

Layer (type)	Output Shape	Param #
conv3d_5 (Conv3D)	(None, 30, 256, 256, 16)	448
max_pooling3d_5 (MaxPooling 3D)	(None, 15, 128, 128, 16)	0
batch_normalization_5 (BatchNormalization)	(None, 15, 128, 128, 16)	64
conv3d_6 (Conv3D)	(None, 15, 128, 128, 32)	13856
max_pooling3d_6 (MaxPooling 3D)	(None, 8, 64, 64, 32)	0
batch_normalization_6 (BatchNormalization)	(None, 8, 64, 64, 32)	128

<i>conv3d_7</i> (<i>Conv3D</i>)	(None, 8, 64, 64, 32)	27680
<i>max_pooling3d_7</i> (<i>MaxPooling3D</i>)	(None, 4, 32, 32, 32)	0
<i>batch_normalization_7</i> (<i>BatchNormalization</i>)	(None, 4, 32, 32, 32)	128
<i>conv3d_8</i> (<i>Conv3D</i>)	(None, 4, 32, 32, 64)	55360
<i>max_pooling3d_8</i> (<i>MaxPooling3D</i>)	(None, 2, 16, 16, 64)	0
<i>batch_normalization_8</i> (<i>BatchNormalization</i>)	(None, 2, 16, 16, 64)	256
<i>conv3d_9</i> (<i>Conv3D</i>)	(None, 2, 16, 16, 64)	110656
<i>max_pooling3d_9</i> (<i>MaxPooling3D</i>)	(None, 1, 8, 8, 64)	0
<i>batch_normalization_9</i> (<i>BatchNormalization</i>)	(None, 1, 8, 8, 64)	256
<i>conv3d_10</i> (<i>Conv3D</i>)	(None, 1, 8, 8, 128)	221312
<i>max_pooling3d_10</i> (<i>MaxPooling3D</i>)	(None, 1, 4, 4, 128)	0
<i>batch_normalization_10</i> (<i>BatchNormalization</i>)	(None, 1, 4, 4, 128)	512
<i>conv3d_11</i> (<i>Conv3D</i>)	(None, 1, 4, 4, 128)	442496
<i>max_pooling3d_11</i> (<i>MaxPooling3D</i>)	(None, 1, 2, 2, 128)	0
<i>batch_normalization_11</i> (<i>BatchNormalization</i>)	(None, 1, 2, 2, 128)	512
<i>conv3d_12</i> (<i>Conv3D</i>)	(None, 1, 2, 2, 256)	884992
<i>max_pooling3d_12</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>batch_normalization_12</i> (<i>BatchNormalization</i>)	(None, 1, 1, 1, 256)	1024
<i>global_average_pooling3d_1</i> (<i>GlobalAveragePooling3D</i>)	(None, 256)	0

<i>dense_3</i> (<i>Dense</i>)	(None, 512)	131584
<i>dropout_2</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_4</i> (<i>Dense</i>)	(None, 256)	131328
<i>dropout_3</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_5</i> (<i>Dense</i>)	(None, 1)	257

```
=====
Total params: 2,022,849
Trainable params: 2,021,409
Non-trainable params: 1,440
```

```
%%time
with tf.device('/device:GPU:0'):
    history = MRNet_Model2.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EP0CHS,
                                validation_data=utils.batch_generator(X_valid,
y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=mrnet_class_weights,
                                verbose=1,
                                callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()])

Epoch 1/100
163/163 [=====] - 73s 421ms/step - loss: 0.6596 - accuracy: 0.6572 - val_loss: 0.7482 - val_accuracy: 0.5054
Epoch 2/100
163/163 [=====] - 68s 414ms/step - loss: 0.5460 - accuracy: 0.7462 - val_loss: 0.6914 - val_accuracy: 0.5429
Epoch 3/100
163/163 [=====] - 68s 415ms/step - loss: 0.4946 - accuracy: 0.7799 - val_loss: 0.4758 - val_accuracy: 0.7857
Epoch 4/100
163/163 [=====] - 68s 415ms/step - loss: 0.4436 - accuracy: 0.7975 - val_loss: 0.4520 - val_accuracy: 0.7821
Epoch 5/100
163/163 [=====] - 67s 414ms/step - loss: 0.4107 - accuracy: 0.8183 - val_loss: 0.3953 - val_accuracy: 0.8429
Epoch 6/100
163/163 [=====] - 67s 414ms/step - loss: 0.3447 - accuracy: 0.8551 - val_loss: 0.3899 - val_accuracy: 0.8232
Epoch 7/100
163/163 [=====] - 68s 414ms/step - loss: 0.2803 -
```

```

accuracy: 0.8880 - val_loss: 0.5396 - val_accuracy: 0.7625
Epoch 8/100
163/163 [=====] - 67s 412ms/step - loss: 0.2169 -
accuracy: 0.9195 - val_loss: 0.4425 - val_accuracy: 0.8304
Epoch 9/100
163/163 [=====] - 67s 413ms/step - loss: 0.1849 -
accuracy: 0.9317 - val_loss: 0.4553 - val_accuracy: 0.8054
Epoch 10/100
163/163 [=====] - 67s 413ms/step - loss: 0.1667 -
accuracy: 0.9340 - val_loss: 0.6973 - val_accuracy: 0.7518
Epoch 11/100
163/163 [=====] - 67s 412ms/step - loss: 0.2044 -
accuracy: 0.9141 - val_loss: 0.4275 - val_accuracy: 0.8446
Epoch 12/100
163/163 [=====] - 68s 418ms/step - loss: 0.1514 -
accuracy: 0.9517 - val_loss: 0.5820 - val_accuracy: 0.8232
Epoch 13/100
163/163 [=====] - 67s 413ms/step - loss: 0.1495 -
accuracy: 0.9479 - val_loss: 0.4801 - val_accuracy: 0.8161
Epoch 14/100
163/163 [=====] - 67s 411ms/step - loss: 0.1649 -
accuracy: 0.9356 - val_loss: 0.4173 - val_accuracy: 0.8661
Epoch 15/100
163/163 [=====] - 67s 411ms/step - loss: 0.1528 -
accuracy: 0.9379 - val_loss: 0.4423 - val_accuracy: 0.8357
Epoch 16/100
163/163 [=====] - ETA: 0s - loss: 0.1051 - accuracy:
0.9601Restoring model weights from the end of the best epoch: 6.
163/163 [=====] - 67s 411ms/step - loss: 0.1051 -
accuracy: 0.9601 - val_loss: 0.5047 - val_accuracy: 0.8589
Epoch 16: early stopping
Wall time: 18min 3s

```

```

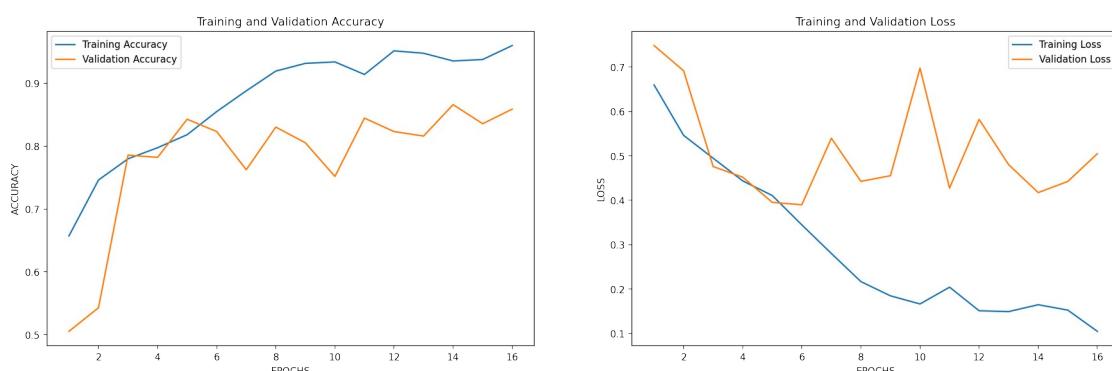
# Store history
utils.store_model_history(model_name, history.history)

```

```

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob = MRNet_Model2.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])

```

26/26 [=====] - 5s 183ms/step

Best cutoff Threshold = 0.442159, F-Score = 0.869

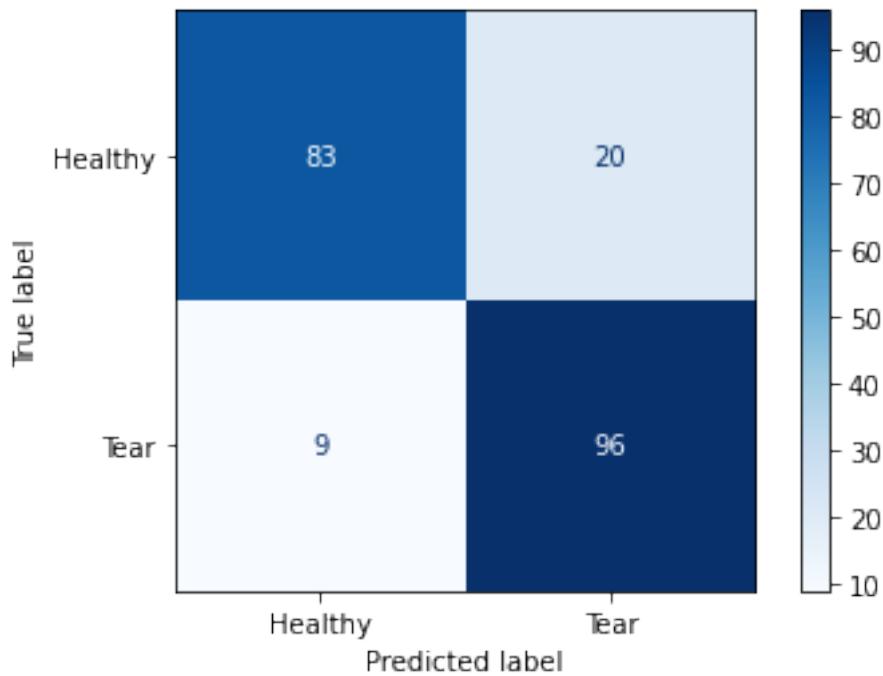
Evaluation Metrics:

*Balanced Accuracy : 0.86
Precision : 0.83
Recall : 0.91
F1 Score: 0.87
ROC AUC Score : 0.92*

Classification report :

	<i>precision</i>	<i>recall</i>	<i>f1-score</i>	<i>support</i>
<i>Healthy</i>	<i>0.90</i>	<i>0.81</i>	<i>0.85</i>	<i>103</i>
<i>Tear</i>	<i>0.83</i>	<i>0.91</i>	<i>0.87</i>	<i>105</i>
<i>accuracy</i>			<i>0.86</i>	<i>208</i>
<i>macro avg</i>	<i>0.86</i>	<i>0.86</i>	<i>0.86</i>	<i>208</i>
<i>weighted avg</i>	<i>0.86</i>	<i>0.86</i>	<i>0.86</i>	<i>208</i>

Confusion Matrix :



```

model_name = 'MRNet_Model3'
MRNet_Model3 = models.mri_model_3(model_name, 2)
MRNet_Model3.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
MRNet_Model3.summary()

```

Model: "MRNet_Model3"

Layer (type)	Output Shape	Param #
conv3d_13 (Conv3D)	(None, 30, 256, 256, 32)	896
max_pooling3d_13 (MaxPooling3D)	(None, 15, 128, 128, 32)	0
batch_normalization_13 (BatchNormalization)	(None, 15, 128, 128, 32)	128
conv3d_14 (Conv3D)	(None, 15, 128, 128, 32)	27680
max_pooling3d_14 (MaxPooling3D)	(None, 8, 64, 64, 32)	0
batch_normalization_14 (BatchNormalization)	(None, 8, 64, 64, 32)	128
conv3d_15 (Conv3D)	(None, 8, 64, 64, 32)	27680
max_pooling3d_15 (MaxPooling3D)	(None, 4, 32, 32, 32)	0

g3D)

<i>batch_normalization_15 (BatchNormalization)</i>	(None, 4, 32, 32, 32)	128
<i>conv3d_16 (Conv3D)</i>	(None, 4, 32, 32, 64)	55360
<i>max_pooling3d_16 (MaxPooling3D)</i>	(None, 2, 16, 16, 64)	0
<i>batch_normalization_16 (BatchNormalization)</i>	(None, 2, 16, 16, 64)	256
<i>conv3d_17 (Conv3D)</i>	(None, 2, 16, 16, 64)	110656
<i>max_pooling3d_17 (MaxPooling3D)</i>	(None, 1, 8, 8, 64)	0
<i>batch_normalization_17 (BatchNormalization)</i>	(None, 1, 8, 8, 64)	256
<i>conv3d_18 (Conv3D)</i>	(None, 1, 8, 8, 64)	110656
<i>max_pooling3d_18 (MaxPooling3D)</i>	(None, 1, 4, 4, 64)	0
<i>batch_normalization_18 (BatchNormalization)</i>	(None, 1, 4, 4, 64)	256
<i>conv3d_19 (Conv3D)</i>	(None, 1, 4, 4, 128)	221312
<i>max_pooling3d_19 (MaxPooling3D)</i>	(None, 1, 2, 2, 128)	0
<i>batch_normalization_19 (BatchNormalization)</i>	(None, 1, 2, 2, 128)	512
<i>conv3d_20 (Conv3D)</i>	(None, 1, 2, 2, 128)	442496
<i>max_pooling3d_20 (MaxPooling3D)</i>	(None, 1, 1, 1, 128)	0
<i>batch_normalization_20 (BatchNormalization)</i>	(None, 1, 1, 1, 128)	512
<i>conv3d_21 (Conv3D)</i>	(None, 1, 1, 1, 128)	442496
<i>max_pooling3d_21 (MaxPooling3D)</i>	(None, 1, 1, 1, 128)	0
<i>batch_normalization_21 (BatchNormalization)</i>	(None, 1, 1, 1, 128)	512

<i>conv3d_22</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	884992
<i>max_pooling3d_22</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>batch_normalization_22</i> (<i>BatchNormalization</i>)	(None, 1, 1, 1, 256)	1024
<i>conv3d_23</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	1769728
<i>max_pooling3d_23</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>batch_normalization_23</i> (<i>BatchNormalization</i>)	(None, 1, 1, 1, 256)	1024
<i>global_average_pooling3d_2</i> (<i>GlobalAveragePooling3D</i>)	(None, 256)	0
<i>dense_6</i> (<i>Dense</i>)	(None, 512)	131584
<i>dropout_4</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_7</i> (<i>Dense</i>)	(None, 512)	262656
<i>dropout_5</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_8</i> (<i>Dense</i>)	(None, 256)	131328
<i>dropout_6</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_9</i> (<i>Dense</i>)	(None, 1)	257

Total params: 4,624,513
 Trainable params: 4,622,145
 Non-trainable params: 2,368

```
%%time
with tf.device('/device:GPU:0'):
    history = MRNet_Model3.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EPOCHS,
                                validation_data=utils.batch_generator(X_valid,
y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=mrnet_class_weights,
                                verbose=1,
```

```
callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()]

Epoch 1/100
163/163 [=====] - 90s 507ms/step - loss: 0.6681 - accuracy: 0.6495 - val_loss: 0.6929 - val_accuracy: 0.5179
Epoch 2/100
163/163 [=====] - 82s 503ms/step - loss: 0.5744 - accuracy: 0.7109 - val_loss: 0.6920 - val_accuracy: 0.5036
Epoch 3/100
163/163 [=====] - 82s 503ms/step - loss: 0.5567 - accuracy: 0.7377 - val_loss: 0.5895 - val_accuracy: 0.7268
Epoch 4/100
163/163 [=====] - 82s 503ms/step - loss: 0.5330 - accuracy: 0.7477 - val_loss: 0.4849 - val_accuracy: 0.7607
Epoch 5/100
163/163 [=====] - 82s 503ms/step - loss: 0.4796 - accuracy: 0.7768 - val_loss: 0.4824 - val_accuracy: 0.7893
Epoch 6/100
163/163 [=====] - 82s 503ms/step - loss: 0.4763 - accuracy: 0.7776 - val_loss: 0.4778 - val_accuracy: 0.7875
Epoch 7/100
163/163 [=====] - 82s 501ms/step - loss: 0.4089 - accuracy: 0.8359 - val_loss: 0.5345 - val_accuracy: 0.7661
Epoch 8/100
163/163 [=====] - 82s 502ms/step - loss: 0.3586 - accuracy: 0.8535 - val_loss: 0.4330 - val_accuracy: 0.8036
Epoch 9/100
163/163 [=====] - 81s 500ms/step - loss: 0.3226 - accuracy: 0.8581 - val_loss: 0.5422 - val_accuracy: 0.7482
Epoch 10/100
163/163 [=====] - 82s 501ms/step - loss: 0.3245 - accuracy: 0.8604 - val_loss: 0.4314 - val_accuracy: 0.7911
Epoch 11/100
163/163 [=====] - 81s 500ms/step - loss: 0.3160 - accuracy: 0.8712 - val_loss: 0.4522 - val_accuracy: 0.7821
Epoch 12/100
163/163 [=====] - 81s 500ms/step - loss: 0.2528 - accuracy: 0.9026 - val_loss: 0.5166 - val_accuracy: 0.7982
Epoch 13/100
163/163 [=====] - 81s 501ms/step - loss: 0.2430 - accuracy: 0.9026 - val_loss: 0.3958 - val_accuracy: 0.8357
Epoch 14/100
163/163 [=====] - 81s 500ms/step - loss: 0.2608 - accuracy: 0.8988 - val_loss: 0.3863 - val_accuracy: 0.8482
Epoch 15/100
163/163 [=====] - 81s 499ms/step - loss: 0.2367 - accuracy: 0.9034 - val_loss: 0.6619 - val_accuracy: 0.6893
Epoch 16/100
163/163 [=====] - 81s 500ms/step - loss: 0.2243 - accuracy: 0.9172 - val_loss: 0.3862 - val_accuracy: 0.8214
```

Epoch 17/100
163/163 [=====] - 81s 498ms/step - loss: 0.1954 -
accuracy: 0.9233 - val_loss: 0.4920 - val_accuracy: 0.7804
Epoch 18/100
163/163 [=====] - 81s 499ms/step - loss: 0.1804 -
accuracy: 0.9379 - val_loss: 0.6024 - val_accuracy: 0.7661
Epoch 19/100
163/163 [=====] - 81s 498ms/step - loss: 0.1937 -
accuracy: 0.9317 - val_loss: 0.4864 - val_accuracy: 0.8071
Epoch 20/100
163/163 [=====] - 81s 498ms/step - loss: 0.1948 -
accuracy: 0.9325 - val_loss: 0.4513 - val_accuracy: 0.8232
Epoch 21/100
163/163 [=====] - 81s 500ms/step - loss: 0.1629 -
accuracy: 0.9410 - val_loss: 0.3527 - val_accuracy: 0.8571
Epoch 22/100
163/163 [=====] - 81s 498ms/step - loss: 0.1489 -
accuracy: 0.9471 - val_loss: 0.4925 - val_accuracy: 0.7875
Epoch 23/100
163/163 [=====] - 81s 498ms/step - loss: 0.1526 -
accuracy: 0.9348 - val_loss: 0.5527 - val_accuracy: 0.7357
Epoch 24/100
163/163 [=====] - 81s 498ms/step - loss: 0.1596 -
accuracy: 0.9433 - val_loss: 0.4357 - val_accuracy: 0.8143
Epoch 25/100
163/163 [=====] - 81s 499ms/step - loss: 0.1685 -
accuracy: 0.9394 - val_loss: 0.3512 - val_accuracy: 0.8750
Epoch 26/100
163/163 [=====] - 81s 497ms/step - loss: 0.1408 -
accuracy: 0.9486 - val_loss: 0.4034 - val_accuracy: 0.8375
Epoch 27/100
163/163 [=====] - 81s 498ms/step - loss: 0.1245 -
accuracy: 0.9540 - val_loss: 0.5447 - val_accuracy: 0.7857
Epoch 28/100
163/163 [=====] - 81s 497ms/step - loss: 0.1809 -
accuracy: 0.9363 - val_loss: 0.3949 - val_accuracy: 0.8321
Epoch 29/100
163/163 [=====] - 81s 498ms/step - loss: 0.1099 -
accuracy: 0.9617 - val_loss: 0.3832 - val_accuracy: 0.8714
Epoch 30/100
163/163 [=====] - 81s 499ms/step - loss: 0.1473 -
accuracy: 0.9548 - val_loss: 0.2845 - val_accuracy: 0.8857
Epoch 31/100
163/163 [=====] - 81s 498ms/step - loss: 0.1071 -
accuracy: 0.9693 - val_loss: 0.4773 - val_accuracy: 0.8304
Epoch 32/100
163/163 [=====] - 81s 498ms/step - loss: 0.1034 -
accuracy: 0.9678 - val_loss: 0.3785 - val_accuracy: 0.8679
Epoch 33/100
163/163 [=====] - 81s 498ms/step - loss: 0.1381 -
accuracy: 0.9540 - val_loss: 0.3684 - val_accuracy: 0.8446
Epoch 34/100

```

163/163 [=====] - 81s 498ms/step - loss: 0.1322 -
accuracy: 0.9517 - val_loss: 0.4043 - val_accuracy: 0.8661
Epoch 35/100
163/163 [=====] - 81s 496ms/step - loss: 0.1036 -
accuracy: 0.9663 - val_loss: 0.3833 - val_accuracy: 0.8571
Epoch 36/100
163/163 [=====] - 81s 497ms/step - loss: 0.0965 -
accuracy: 0.9655 - val_loss: 0.3592 - val_accuracy: 0.8643
Epoch 37/100
163/163 [=====] - 81s 496ms/step - loss: 0.0931 -
accuracy: 0.9663 - val_loss: 0.3576 - val_accuracy: 0.8643
Epoch 38/100
163/163 [=====] - 81s 497ms/step - loss: 0.0998 -
accuracy: 0.9670 - val_loss: 0.8350 - val_accuracy: 0.8000
Epoch 39/100
163/163 [=====] - 81s 497ms/step - loss: 0.1174 -
accuracy: 0.9578 - val_loss: 0.4556 - val_accuracy: 0.8107
Epoch 40/100
163/163 [=====] - ETA: 0s - loss: 0.0973 - accuracy:
0.9678Restoring model weights from the end of the best epoch: 30.
163/163 [=====] - 81s 497ms/step - loss: 0.0973 -
accuracy: 0.9678 - val_loss: 0.4780 - val_accuracy: 0.8018
Epoch 40: early stopping
Wall time: 54min 18s

```

```

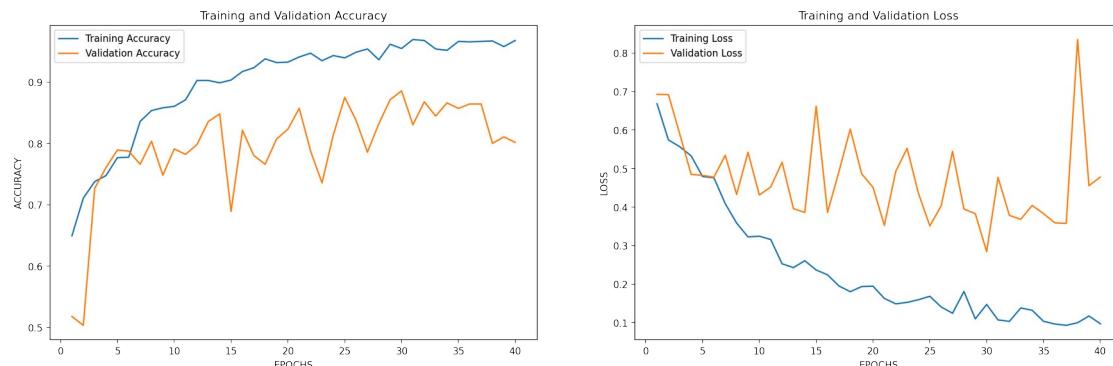
# Store history
utils.store_model_history(model_name, history.history)

```

```

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob = MRNet_Model3.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))

```

```

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

```

```
utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])
```

26/26 [=====] - 5s 185ms/step

Best cutoff Threshold = 0.429860, F-Score = 0.874

Evaluation Metrics:

Balanced Accuracy : 0.88

Precision : 0.89

Recall : 0.86

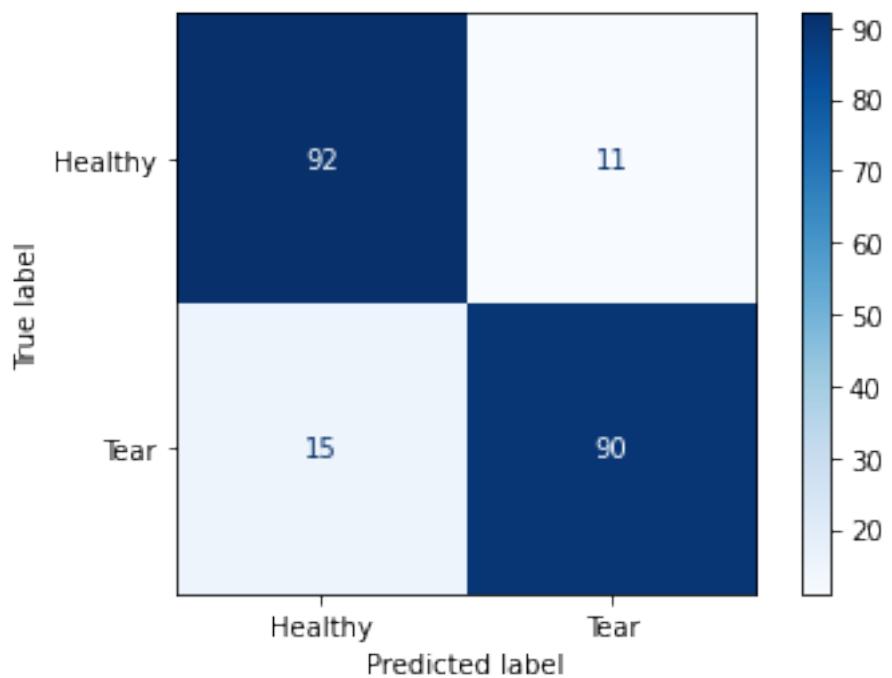
F1 Score: 0.87

ROC AUC Score : 0.93

Classification report :

	precision	recall	f1-score	support
Healthy	0.86	0.89	0.88	103
Tear	0.89	0.86	0.87	105
accuracy			0.88	208
macro avg	0.88	0.88	0.87	208
weighted avg	0.88	0.88	0.87	208

Confusion Matrix :



```
model_name = 'MRNet_Model4'
```

```
MRNet_Model4 = models.mri_model_4(model_name, 2)
```

```
MRNet_Model4.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model
```

```
_lr_schedule(),
      loss='binary_crossentropy',
      metrics=['accuracy'])
MRNet_Model4.summary()
```

Model: "MRNet_Model4"

Layer (type)	Output Shape	Param #
conv3d_24 (Conv3D)	(None, 30, 256, 256, 32)	896
conv3d_25 (Conv3D)	(None, 30, 256, 256, 32)	27680
max_pooling3d_24 (MaxPooling3D)	(None, 15, 128, 128, 32)	0
conv3d_26 (Conv3D)	(None, 15, 128, 128, 32)	27680
conv3d_27 (Conv3D)	(None, 15, 128, 128, 64)	55360
max_pooling3d_25 (MaxPooling3D)	(None, 8, 64, 64, 64)	0
conv3d_28 (Conv3D)	(None, 8, 64, 64, 64)	110656
conv3d_29 (Conv3D)	(None, 8, 64, 64, 128)	221312
max_pooling3d_26 (MaxPooling3D)	(None, 4, 32, 32, 128)	0
dropout_7 (Dropout)	(None, 4, 32, 32, 128)	0
flatten (Flatten)	(None, 524288)	0
dense_10 (Dense)	(None, 256)	134217984
dropout_8 (Dropout)	(None, 256)	0
dense_11 (Dense)	(None, 128)	32896
dropout_9 (Dropout)	(None, 128)	0
dense_12 (Dense)	(None, 1)	129
<hr/>		
Total params: 134,694,593		
Trainable params: 134,694,593		
Non-trainable params: 0		

```
%%time
with tf.device('/device:GPU:0'):
```

```
history = MRNet_Model4.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                           steps_per_epoch=len(X_train)//BATCH_SIZE,
                           epochs=EP0CHS,
                           validation_data=utils.batch_generator(X_valid,
y_valid, BATCH_SIZE),
                           validation_steps=len(X_valid)//BATCH_SIZE,
                           shuffle=True,
                           class_weight=mrnet_class_weights,
                           verbose=1,
                           callbacks=[utils.model_callback_checkpoint(model_name),
                           utils.model_callback_earlystopping()])

Epoch 1/100
163/163 [=====] - 284s 1s/step - loss: 0.6381 - accuracy: 0.6549 - val_loss: 0.5337 - val_accuracy: 0.7196
Epoch 2/100
163/163 [=====] - 212s 1s/step - loss: 0.5608 - accuracy: 0.7385 - val_loss: 0.4988 - val_accuracy: 0.7446
Epoch 3/100
163/163 [=====] - 209s 1s/step - loss: 0.5307 - accuracy: 0.7546 - val_loss: 0.5483 - val_accuracy: 0.7268
Epoch 4/100
163/163 [=====] - 212s 1s/step - loss: 0.4823 - accuracy: 0.7830 - val_loss: 0.4652 - val_accuracy: 0.7893
Epoch 5/100
163/163 [=====] - 212s 1s/step - loss: 0.4312 - accuracy: 0.8198 - val_loss: 0.4373 - val_accuracy: 0.8304
Epoch 6/100
163/163 [=====] - 209s 1s/step - loss: 0.3741 - accuracy: 0.8482 - val_loss: 0.5070 - val_accuracy: 0.7964
Epoch 7/100
163/163 [=====] - 212s 1s/step - loss: 0.3096 - accuracy: 0.8758 - val_loss: 0.4229 - val_accuracy: 0.8393
Epoch 8/100
163/163 [=====] - 209s 1s/step - loss: 0.2224 - accuracy: 0.9041 - val_loss: 0.4744 - val_accuracy: 0.8357
Epoch 9/100
163/163 [=====] - 212s 1s/step - loss: 0.1835 - accuracy: 0.9302 - val_loss: 0.3768 - val_accuracy: 0.8607
Epoch 10/100
163/163 [=====] - 209s 1s/step - loss: 0.1306 - accuracy: 0.9479 - val_loss: 0.5577 - val_accuracy: 0.8429
Epoch 11/100
163/163 [=====] - 209s 1s/step - loss: 0.0912 - accuracy: 0.9647 - val_loss: 0.5942 - val_accuracy: 0.8518
Epoch 12/100
163/163 [=====] - 209s 1s/step - loss: 0.0517 - accuracy: 0.9816 - val_loss: 0.6471 - val_accuracy: 0.8589
Epoch 13/100
163/163 [=====] - 209s 1s/step - loss: 0.0390 -
```

```

accuracy: 0.9877 - val_loss: 0.6118 - val_accuracy: 0.8714
Epoch 14/100
163/163 [=====] - 209s 1s/step - loss: 0.0488 -
accuracy: 0.9831 - val_loss: 0.6714 - val_accuracy: 0.8804
Epoch 15/100
163/163 [=====] - 209s 1s/step - loss: 0.0403 -
accuracy: 0.9847 - val_loss: 0.5570 - val_accuracy: 0.8714
Epoch 16/100
163/163 [=====] - 209s 1s/step - loss: 0.0199 -
accuracy: 0.9946 - val_loss: 1.1492 - val_accuracy: 0.8357
Epoch 17/100
163/163 [=====] - 209s 1s/step - loss: 0.0416 -
accuracy: 0.9862 - val_loss: 0.7561 - val_accuracy: 0.8607
Epoch 18/100
163/163 [=====] - 209s 1s/step - loss: 0.0170 -
accuracy: 0.9939 - val_loss: 0.8616 - val_accuracy: 0.8518
Epoch 19/100
163/163 [=====] - ETA: 0s - loss: 0.0150 - accuracy:
0.9962Restoring model weights from the end of the best epoch: 9.
163/163 [=====] - 209s 1s/step - loss: 0.0150 -
accuracy: 0.9962 - val_loss: 0.9347 - val_accuracy: 0.8821
Epoch 19: early stopping
Wall time: 1h 7min 41s

```

```

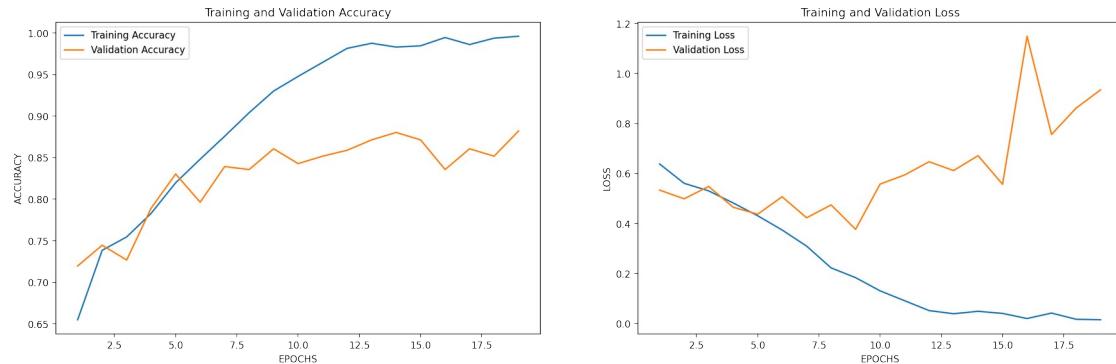
# Store history
utils.store_model_history(model_name, history.history)

```

```

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob = MRNet_Model4.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])

```

26/26 [=====] - 14s 488ms/step

Best cutoff Threshold = 0.249402, F-Score = 0.857

Evaluation Metrics:

Balanced Accuracy : 0.85

Precision : 0.81

Recall : 0.91

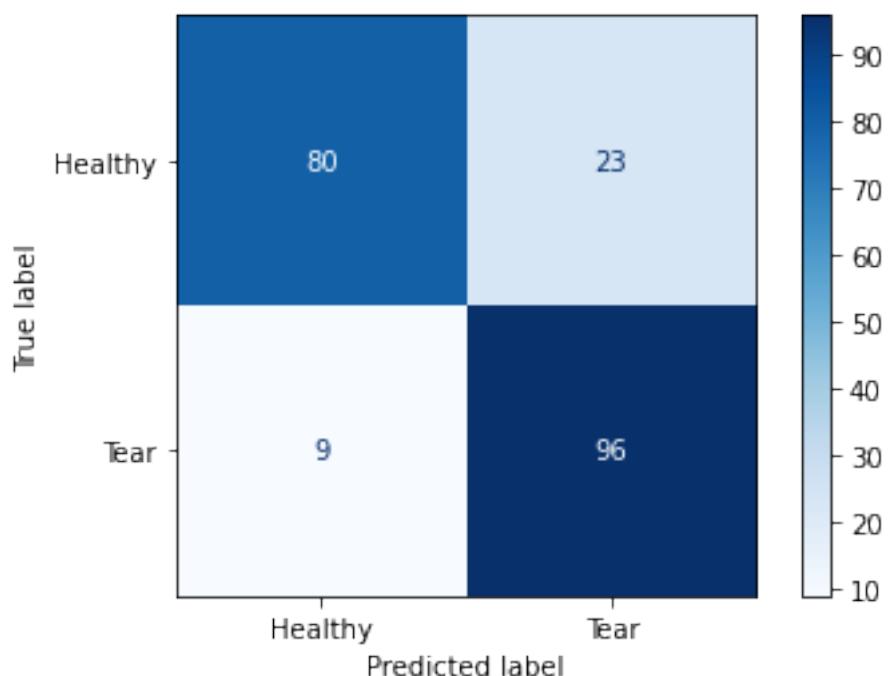
F1 Score: 0.86

ROC AUC Score : 0.93

Classification report :

	precision	recall	f1-score	support
Healthy	0.90	0.78	0.83	103
Tear	0.81	0.91	0.86	105
accuracy			0.85	208
macro avg	0.85	0.85	0.85	208
weighted avg	0.85	0.85	0.85	208

Confusion Matrix :



```
model_name = 'MRNet_Model5'
MRNet_Model5 = models.mri_model_5(model_name, 2)
MRNet_Model5.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                      loss='binary_crossentropy',
```

<i>Layer (type)</i>	<i>Output Shape</i>	<i>Param #</i>
<i>conv3d_30 (Conv3D)</i>	<i>(None, 30, 256, 256, 16)</i>	448
<i>max_pooling3d_27 (MaxPooling3D)</i>	<i>(None, 15, 128, 128, 16)</i>	0
<i>conv3d_31 (Conv3D)</i>	<i>(None, 15, 128, 128, 16)</i>	6928
<i>max_pooling3d_28 (MaxPooling3D)</i>	<i>(None, 8, 64, 64, 16)</i>	0
<i>conv3d_32 (Conv3D)</i>	<i>(None, 8, 64, 64, 32)</i>	13856
<i>max_pooling3d_29 (MaxPooling3D)</i>	<i>(None, 4, 32, 32, 32)</i>	0
<i>conv3d_33 (Conv3D)</i>	<i>(None, 4, 32, 32, 32)</i>	27680
<i>max_pooling3d_30 (MaxPooling3D)</i>	<i>(None, 2, 16, 16, 32)</i>	0
<i>conv3d_34 (Conv3D)</i>	<i>(None, 2, 16, 16, 32)</i>	27680
<i>max_pooling3d_31 (MaxPooling3D)</i>	<i>(None, 1, 8, 8, 32)</i>	0
<i>conv3d_35 (Conv3D)</i>	<i>(None, 1, 8, 8, 64)</i>	55360
<i>max_pooling3d_32 (MaxPooling3D)</i>	<i>(None, 1, 4, 4, 64)</i>	0
<i>conv3d_36 (Conv3D)</i>	<i>(None, 1, 4, 4, 64)</i>	110656
<i>max_pooling3d_33 (MaxPooling3D)</i>	<i>(None, 1, 2, 2, 64)</i>	0
<i>conv3d_37 (Conv3D)</i>	<i>(None, 1, 2, 2, 128)</i>	221312
<i>max_pooling3d_34 (MaxPooling3D)</i>	<i>(None, 1, 1, 1, 128)</i>	0
<i>conv3d_38 (Conv3D)</i>	<i>(None, 1, 1, 1, 128)</i>	442496
<i>max_pooling3d_35 (MaxPooling3D)</i>	<i>(None, 1, 1, 1, 128)</i>	0

<i>conv3d_39</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	884992
<i>max_pooling3d_36</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>dropout_10</i> (<i>Dropout</i>)	(None, 1, 1, 1, 256)	0
<i>flatten_1</i> (<i>Flatten</i>)	(None, 256)	0
<i>dense_13</i> (<i>Dense</i>)	(None, 512)	131584
<i>dropout_11</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_14</i> (<i>Dense</i>)	(None, 256)	131328
<i>dropout_12</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_15</i> (<i>Dense</i>)	(None, 128)	32896
<i>dropout_13</i> (<i>Dropout</i>)	(None, 128)	0
<i>dense_16</i> (<i>Dense</i>)	(None, 1)	129

Total params: 2,087,345
 Trainable params: 2,087,345
 Non-trainable params: 0

```
%%time
with tf.device('/device:GPU:0'):
    history = MRNet_Model5.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EP0CHS,
                                validation_data=utils.batch_generator(X_valid,
y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=mrnet_class_weights,
                                verbose=1,
                                callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()])
Epoch 1/100
163/163 [=====] - 67s 403ms/step - loss: 0.6934 - accuracy: 0.4962 - val_loss: 0.6932 - val_accuracy: 0.4946
Epoch 2/100
163/163 [=====] - 65s 397ms/step - loss: 0.6933 - accuracy: 0.4716 - val_loss: 0.6931 - val_accuracy: 0.4964
Epoch 3/100
```

163/163 [=====] - 64s 395ms/step - loss: 0.6135 -
accuracy: 0.6526 - val_loss: 0.5917 - val_accuracy: 0.7107
Epoch 4/100
163/163 [=====] - 65s 396ms/step - loss: 0.5354 -
accuracy: 0.7431 - val_loss: 0.5510 - val_accuracy: 0.7054
Epoch 5/100
163/163 [=====] - 64s 395ms/step - loss: 0.4960 -
accuracy: 0.7715 - val_loss: 0.5025 - val_accuracy: 0.7696
Epoch 6/100
163/163 [=====] - 64s 394ms/step - loss: 0.4682 -
accuracy: 0.7952 - val_loss: 0.4819 - val_accuracy: 0.7661
Epoch 7/100
163/163 [=====] - 64s 395ms/step - loss: 0.4390 -
accuracy: 0.8090 - val_loss: 0.4899 - val_accuracy: 0.7875
Epoch 8/100
163/163 [=====] - 64s 394ms/step - loss: 0.4203 -
accuracy: 0.8090 - val_loss: 0.4557 - val_accuracy: 0.7768
Epoch 9/100
163/163 [=====] - 64s 394ms/step - loss: 0.4041 -
accuracy: 0.8344 - val_loss: 0.4756 - val_accuracy: 0.7661
Epoch 10/100
163/163 [=====] - 64s 393ms/step - loss: 0.3860 -
accuracy: 0.8298 - val_loss: 0.4897 - val_accuracy: 0.7750
Epoch 11/100
163/163 [=====] - 64s 393ms/step - loss: 0.3713 -
accuracy: 0.8451 - val_loss: 0.4563 - val_accuracy: 0.7964
Epoch 12/100
163/163 [=====] - 64s 392ms/step - loss: 0.3216 -
accuracy: 0.8742 - val_loss: 0.5097 - val_accuracy: 0.8089
Epoch 13/100
163/163 [=====] - 64s 392ms/step - loss: 0.2965 -
accuracy: 0.8888 - val_loss: 0.4839 - val_accuracy: 0.8179
Epoch 14/100
163/163 [=====] - 64s 393ms/step - loss: 0.2788 -
accuracy: 0.8926 - val_loss: 0.4358 - val_accuracy: 0.8268
Epoch 15/100
163/163 [=====] - 64s 392ms/step - loss: 0.2586 -
accuracy: 0.9041 - val_loss: 0.5009 - val_accuracy: 0.7946
Epoch 16/100
163/163 [=====] - 64s 391ms/step - loss: 0.2105 -
accuracy: 0.9210 - val_loss: 0.5702 - val_accuracy: 0.8411
Epoch 17/100
163/163 [=====] - 64s 392ms/step - loss: 0.2021 -
accuracy: 0.9195 - val_loss: 0.4864 - val_accuracy: 0.8411
Epoch 18/100
163/163 [=====] - 64s 392ms/step - loss: 0.1694 -
accuracy: 0.9410 - val_loss: 0.5537 - val_accuracy: 0.8393
Epoch 19/100
163/163 [=====] - 64s 391ms/step - loss: 0.1338 -
accuracy: 0.9494 - val_loss: 0.5010 - val_accuracy: 0.8161
Epoch 20/100
163/163 [=====] - 64s 392ms/step - loss: 0.1166 -

```

accuracy: 0.9609 - val_loss: 0.6598 - val_accuracy: 0.8232
Epoch 21/100
163/163 [=====] - 64s 391ms/step - loss: 0.0853 -
accuracy: 0.9747 - val_loss: 0.6354 - val_accuracy: 0.8339
Epoch 22/100
163/163 [=====] - 64s 391ms/step - loss: 0.0969 -
accuracy: 0.9632 - val_loss: 0.4981 - val_accuracy: 0.8625
Epoch 23/100
163/163 [=====] - 64s 390ms/step - loss: 0.0525 -
accuracy: 0.9862 - val_loss: 0.7641 - val_accuracy: 0.8071
Epoch 24/100
163/163 [=====] - ETA: 0s - loss: 0.0820 - accuracy:
0.9709Restoring model weights from the end of the best epoch: 14.
163/163 [=====] - 64s 391ms/step - loss: 0.0820 -
accuracy: 0.9709 - val_loss: 0.6962 - val_accuracy: 0.8375
Epoch 24: early stopping
Wall time: 25min 40s

```

```

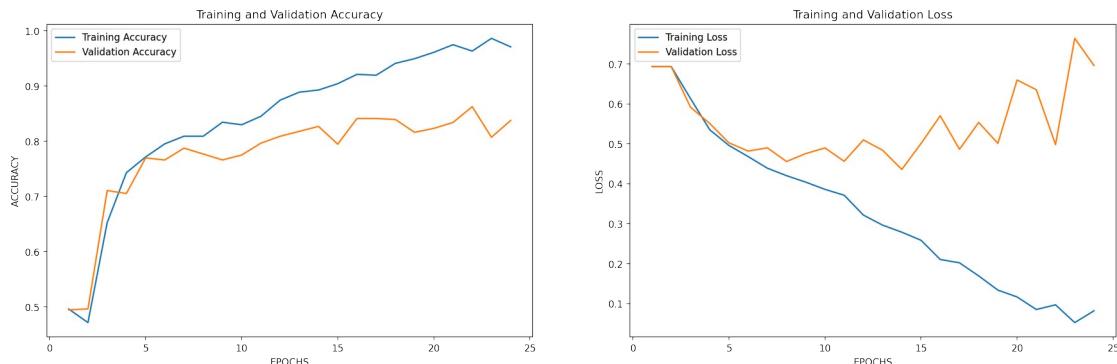
# Store history
utils.store_model_history(model_name, history.history)

```

```

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob = MRNet_Model5.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])

```

```

26/26 [=====] - 5s 199ms/step

```

Best cutoff Threshold = 0.246566, F-Score = 0.833

Evaluation Metrics:

Balanced Accuracy : 0.81

Precision : 0.76

Recall : 0.92

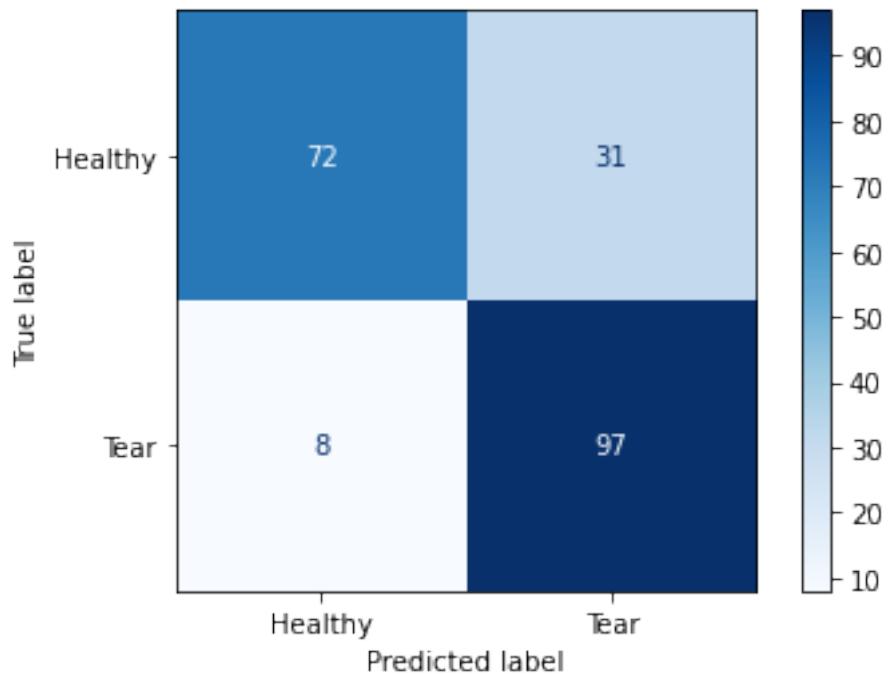
F1 Score: 0.83

ROC AUC Score : 0.9

Classification report :

	precision	recall	f1-score	support
Healthy	0.90	0.70	0.79	103
Tear	0.76	0.92	0.83	105
accuracy			0.81	208
macro avg	0.83	0.81	0.81	208
weighted avg	0.83	0.81	0.81	208

Confusion Matrix :



```
model_name = 'MRNet_Model6'
MRNet_Model6 = models.mri_model_6(model_name, 2)
MRNet_Model6.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
MRNet_Model6.summary()
```

Model: "MRNet_Model6"

Layer (type)	Output Shape	Param #

<i>conv3d</i> (<i>Conv3D</i>)	(None, 30, 256, 256, 8)	224
<i>conv3d_1</i> (<i>Conv3D</i>)	(None, 30, 256, 256, 16)	3472
<i>max_pooling3d</i> (<i>MaxPooling3D</i>)	(None, 15, 128, 128, 16)	0
<i>conv3d_2</i> (<i>Conv3D</i>)	(None, 15, 128, 128, 16)	6928
<i>conv3d_3</i> (<i>Conv3D</i>)	(None, 15, 128, 128, 16)	6928
<i>max_pooling3d_1</i> (<i>MaxPooling3D</i>)	(None, 8, 64, 64, 16)	0
<i>conv3d_4</i> (<i>Conv3D</i>)	(None, 8, 64, 64, 16)	6928
<i>conv3d_5</i> (<i>Conv3D</i>)	(None, 8, 64, 64, 16)	6928
<i>max_pooling3d_2</i> (<i>MaxPooling3D</i>)	(None, 4, 32, 32, 16)	0
<i>batch_normalization</i> (<i>BatchNormalization</i>)	(None, 4, 32, 32, 16)	64
<i>conv3d_6</i> (<i>Conv3D</i>)	(None, 4, 32, 32, 32)	13856
<i>conv3d_7</i> (<i>Conv3D</i>)	(None, 4, 32, 32, 32)	27680
<i>max_pooling3d_3</i> (<i>MaxPooling3D</i>)	(None, 2, 16, 16, 32)	0
<i>conv3d_8</i> (<i>Conv3D</i>)	(None, 2, 16, 16, 32)	27680
<i>max_pooling3d_4</i> (<i>MaxPooling3D</i>)	(None, 1, 8, 8, 32)	0
<i>batch_normalization_1</i> (<i>BatchNormalization</i>)	(None, 1, 8, 8, 32)	128
<i>conv3d_9</i> (<i>Conv3D</i>)	(None, 1, 8, 8, 64)	55360
<i>max_pooling3d_5</i> (<i>MaxPooling3D</i>)	(None, 1, 4, 4, 64)	0
<i>conv3d_10</i> (<i>Conv3D</i>)	(None, 1, 4, 4, 64)	110656
<i>max_pooling3d_6</i> (<i>MaxPooling3D</i>)	(None, 1, 2, 2, 64)	0
<i>conv3d_11</i> (<i>Conv3D</i>)	(None, 1, 2, 2, 64)	110656
<i>max_pooling3d_7</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 64)	0

3D)

<i>batch_normalization_2</i> (<i>BatchNormalization</i>)	(None, 1, 1, 1, 64)	256
<i>conv3d_12</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 128)	221312
<i>max_pooling3d_8</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 128)	0
<i>conv3d_13</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 128)	442496
<i>max_pooling3d_9</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 128)	0
<i>conv3d_14</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 128)	442496
<i>max_pooling3d_10</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 128)	0
<i>batch_normalization_3</i> (<i>BatchNormalization</i>)	(None, 1, 1, 1, 128)	512
<i>conv3d_15</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	884992
<i>max_pooling3d_11</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>conv3d_16</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	1769728
<i>max_pooling3d_12</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>conv3d_17</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	1769728
<i>max_pooling3d_13</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>dropout</i> (<i>Dropout</i>)	(None, 1, 1, 1, 256)	0
<i>flatten</i> (<i>Flatten</i>)	(None, 256)	0
<i>dense</i> (<i>Dense</i>)	(None, 512)	131584
<i>dropout_1</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_1</i> (<i>Dense</i>)	(None, 256)	131328
<i>dropout_2</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_2</i> (<i>Dense</i>)	(None, 128)	32896

<code>dropout_3 (Dropout)</code>	<code>(None, 128)</code>	<code>0</code>
<code>dense_3 (Dense)</code>	<code>(None, 1)</code>	<code>129</code>

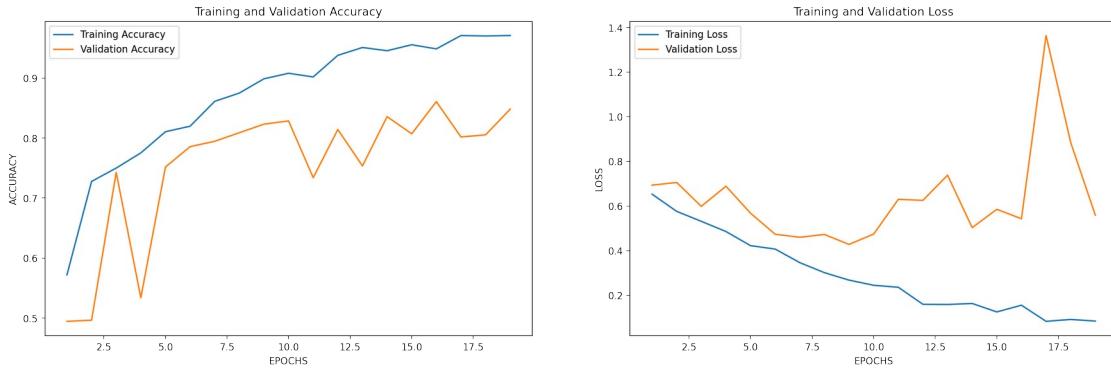
`Total params: 6,204,945`
`Trainable params: 6,204,465`
`Non-trainable params: 480`

```
%%time
with tf.device('/device:GPU:0'):
    history = MRNet_Model6.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EPOCHS,
                                validation_data=utils.batch_generator(X_valid,
y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=mrnet_class_weights,
                                verbose=1,
                                callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()])
Epoch 1/100
163/163 [=====] - 123s 573ms/step - loss: 0.6529 - accuracy: 0.5721 - val_loss: 0.6934 - val_accuracy: 0.4946
Epoch 2/100
163/163 [=====] - 90s 553ms/step - loss: 0.5759 - accuracy: 0.7278 - val_loss: 0.7052 - val_accuracy: 0.4964
Epoch 3/100
163/163 [=====] - 94s 576ms/step - loss: 0.5312 - accuracy: 0.7500 - val_loss: 0.5983 - val_accuracy: 0.7429
Epoch 4/100
163/163 [=====] - 100s 616ms/step - loss: 0.4856 - accuracy: 0.7753 - val_loss: 0.6886 - val_accuracy: 0.5339
Epoch 5/100
163/163 [=====] - 92s 562ms/step - loss: 0.4223 - accuracy: 0.8106 - val_loss: 0.5676 - val_accuracy: 0.7518
Epoch 6/100
163/163 [=====] - 90s 554ms/step - loss: 0.4069 - accuracy: 0.8198 - val_loss: 0.4733 - val_accuracy: 0.7857
Epoch 7/100
163/163 [=====] - 90s 554ms/step - loss: 0.3462 - accuracy: 0.8612 - val_loss: 0.4599 - val_accuracy: 0.7946
Epoch 8/100
163/163 [=====] - 90s 550ms/step - loss: 0.3013 - accuracy: 0.8750 - val_loss: 0.4724 - val_accuracy: 0.8089
Epoch 9/100
163/163 [=====] - 90s 551ms/step - loss: 0.2678 -
```

```
accuracy: 0.8988 - val_loss: 0.4279 - val_accuracy: 0.8232
Epoch 10/100
163/163 [=====] - 89s 547ms/step - loss: 0.2448 -
accuracy: 0.9080 - val_loss: 0.4743 - val_accuracy: 0.8286
Epoch 11/100
163/163 [=====] - 89s 547ms/step - loss: 0.2359 -
accuracy: 0.9018 - val_loss: 0.6296 - val_accuracy: 0.7339
Epoch 12/100
163/163 [=====] - 89s 547ms/step - loss: 0.1595 -
accuracy: 0.9379 - val_loss: 0.6253 - val_accuracy: 0.8143
Epoch 13/100
163/163 [=====] - 90s 553ms/step - loss: 0.1587 -
accuracy: 0.9509 - val_loss: 0.7381 - val_accuracy: 0.7536
Epoch 14/100
163/163 [=====] - 89s 549ms/step - loss: 0.1632 -
accuracy: 0.9456 - val_loss: 0.5030 - val_accuracy: 0.8357
Epoch 15/100
163/163 [=====] - 89s 548ms/step - loss: 0.1253 -
accuracy: 0.9555 - val_loss: 0.5851 - val_accuracy: 0.8071
Epoch 16/100
163/163 [=====] - 89s 548ms/step - loss: 0.1555 -
accuracy: 0.9486 - val_loss: 0.5428 - val_accuracy: 0.8607
Epoch 17/100
163/163 [=====] - 89s 547ms/step - loss: 0.0833 -
accuracy: 0.9709 - val_loss: 1.3635 - val_accuracy: 0.8018
Epoch 18/100
163/163 [=====] - 89s 548ms/step - loss: 0.0917 -
accuracy: 0.9701 - val_loss: 0.8846 - val_accuracy: 0.8054
Epoch 19/100
163/163 [=====] - ETA: 0s - loss: 0.0843 - accuracy:
0.9709Restoring model weights from the end of the best epoch: 9.
163/163 [=====] - 89s 547ms/step - loss: 0.0843 -
accuracy: 0.9709 - val_loss: 0.5587 - val_accuracy: 0.8482
Epoch 19: early stopping
Wall time: 29min 11s

# Store history
utils.store_model_history(model_name, history.history)

# Plot training graphs
utils.plot_acc_loss(history.history)
```



Evaluate model

```
X_test_prob = MRNet_Model6.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])
```

26/26 [=====] - 7s 280ms/step

Best cutoff Threshold = 0.523336, F-Score = 0.826

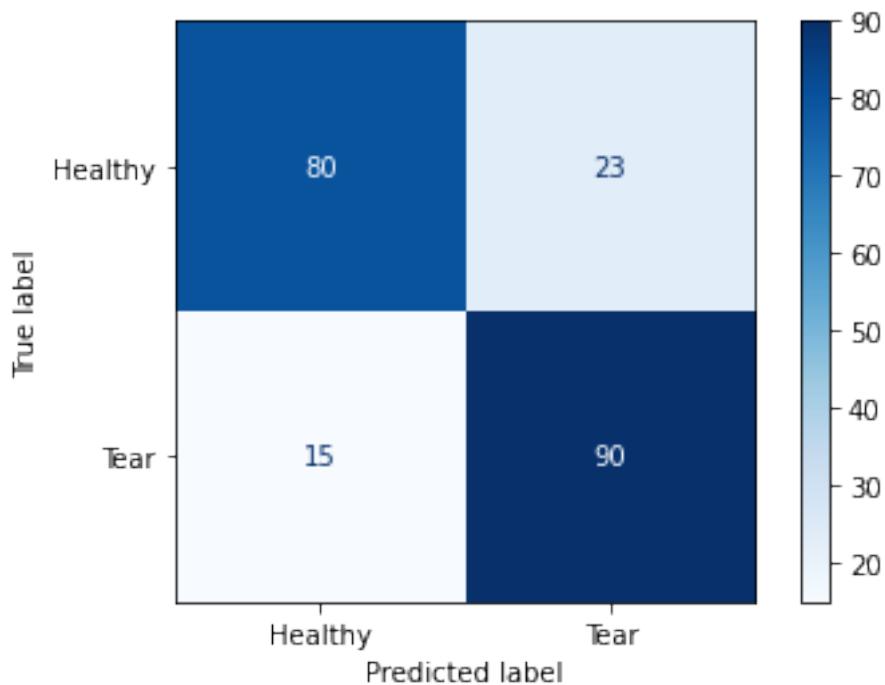
Evaluation Metrics:

Balanced Accuracy : 0.82
Precision : 0.8
Recall : 0.86
F1 Score: 0.83
ROC AUC Score : 0.88

Classification report :

	precision	recall	f1-score	support
Healthy	0.84	0.78	0.81	103
Tear	0.80	0.86	0.83	105
accuracy			0.82	208
macro avg	0.82	0.82	0.82	208
weighted avg	0.82	0.82	0.82	208

Confusion Matrix :



H) KneeMRI Model Training

```

import platform
import pandas as pd
import numpy as np
from glob import glob

import tensorflow as tf
from tensorflow import keras

from sklearn.model_selection import train_test_split
import utils
import models

```

Tensorflow version : 2.10.1

```

Tensorflow devices available :
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {}
incarnation: 3167730454879211143
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 22672310272
locality {}

```

```

bus_id: 1
links {
}
}
incarnation: 8468114776510979279
physical_device_desc: "device: 0, name: NVIDIA RTX A5000, pci bus id: 0000:61:00.0, compute capability: 8.6"
xla_global_id: 416903419
]

Tensorflow physical devices available :
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]

# Directory where the preprocessed volumetric data is located
kneemri_preprocessed_data_dir = 'Preprocessed_Data/KneeMRI'
# path to metadata csv file
kneemri_metadata_csv_path = 'Data/KneeMRI/metadata.csv'
kneemri_aug_metadata_csv_path = 'Data/KneeMRI/metadata-aug.csv'

# For running code on Windows
if platform.system() == "Windows":
    kneemri_preprocessed_data_dir = kneemri_preprocessed_data_dir.replace('/', '\\')
    kneemri_metadata_csv_path = kneemri_metadata_csv_path.replace('/', '\\')
    kneemri_aug_metadata_csv_path = kneemri_aug_metadata_csv_path.replace('/', '\\')

kneemri_classes = { 0:'healthy', 1:'partially ruptured', 2:'completely ruptured' }

if platform.system() == "Windows":
    kneemri_vol_paths = glob(kneemri_preprocessed_data_dir+"\\vol*")
else:
    kneemri_vol_paths = glob(kneemri_preprocessed_data_dir+"/vol*")
kneemri_vol_paths.sort()

kneemri_vol_paths

['Preprocessed_Data\\KneeMRI\\vol01',
'Preprocessed_Data\\KneeMRI\\vol02',
'Preprocessed_Data\\KneeMRI\\vol03',
'Preprocessed_Data\\KneeMRI\\vol04',
'Preprocessed_Data\\KneeMRI\\vol05',
'Preprocessed_Data\\KneeMRI\\vol06',
'Preprocessed_Data\\KneeMRI\\vol07',
'Preprocessed_Data\\KneeMRI\\vol08',
'Preprocessed_Data\\KneeMRI\\vol09',
'Preprocessed_Data\\KneeMRI\\vol10']

kneemri_cases = []
for mri_data_path in kneemri_vol_paths:
    if platform.system() == "Windows":

```

```
    all_exams = glob(mri_data_path+"\\*.npy")
else:
    all_exams = glob(mri_data_path+"/*.npy")
all_exams.sort()
kneemri_cases.extend(all_exams)
```

```
print('Original cases : ', len(kneemri_cases))
```

```
Original cases : 917
```

```
kneemri_cases
```

```
['Preprocessed_Data\\KneeMRI\\vol01\\329637-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\390116-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\404663-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\406320-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\412857-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\412865-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\415102-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\425707-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\425713-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\437474-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\444503-7.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\451760-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\454319-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\455741-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\456691-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\457181-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\457491-6.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\457511-7.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\457519-6.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\457531-6.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\457572-6.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\459462-10.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\459569-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\462792-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\462862-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\463541-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\463773-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\465953-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\466435-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\470658-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\471659-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\471699-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\472130-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\472842-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\473438-7.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\474967-7.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\475047-7.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\476260-7.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\476877-7.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\477303-7.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol01\\477330-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\477376-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\477390-8.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\479150-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\479235-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\479684-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\479697-8.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\480133-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\480892-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\481311-8.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\481331-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\481371-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\481412-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\482432-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\482840-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\482870-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\482876-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\482893-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\482899-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\483367-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\484779-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\484808-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\485381-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\485381-8.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\485761-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\485830-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\485841-7.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\487493-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\487594-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\488561-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\488592-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\490907-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\490928-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\490937-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\491177-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\491387-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\491596-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\492730-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\492792-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\494291-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\494338-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\494350-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\494353-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\496489-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\496580-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\496621-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\497304-8.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\497747-8.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\497764-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\498724-5.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\500142-6.npy',
'Preprocessed_Data\\KneeMRI\\vol01\\500636-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol02\\500804-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\501527-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\502889-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\502889-8.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\502904-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\502951-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\502969-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\504699-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\505151-8.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\505174-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\505176-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\507029-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\507277-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\507834-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\509041-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\510057-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\510970-4.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\511402-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\513064-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\513074-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\513118-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\513145-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\515026-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\515707-7.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\515804-6.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\515837-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\515853-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\515891-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\515999-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\516014-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\517913-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\518221-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\518759-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\518793-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\519259-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\520985-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\521412-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\521464-9.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\524118-6.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\524178-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\524502-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\528728-7.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\528796-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\528814-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\528860-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\531241-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\531267-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\531715-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\532493-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\533741-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\533809-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\536367-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol02\\538076-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\538749-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\540462-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\540503-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\540525-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\542395-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\542846-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\542858-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\544957-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\544984-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\545451-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\547369-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\547373-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\547393-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\547422-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\547450-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\547519-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\549163-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\549844-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\549879-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\549888-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\549897-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\549957-6.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\551570-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\552377-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\552839-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\554377-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\554495-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\554499-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\555003-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\555019-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\555031-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\557202-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\557229-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\557655-7.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\557711-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\557719-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\559079-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\559390-5.npy',
'Preprocessed_Data\\KneeMRI\\vol02\\560649-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\560659-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\560665-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\560718-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\561654-7.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\561659-7.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\561964-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\561983-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\562893-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\562961-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\563359-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\563415-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\563422-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol03\\563429-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\563432-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\563519-10.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\564317-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\566015-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\566022-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\566069-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\566082-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\566089-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\566095-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\566236-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\567480-8.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\568862-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\568871-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\568916-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\568937-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\568943-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\569264-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\569768-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\570373-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\571707-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\571778-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\572059-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\572093-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\574468-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\574851-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\576971-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\579840-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\579883-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\580017-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\580420-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\580449-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\580637-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\580637-6.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\581844-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\582678-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\582708-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\582721-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\582730-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\584099-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\584179-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\584196-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\584216-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\584216-8.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\584216-9.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\584916-6.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\585526-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\587683-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\587708-8.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\587739-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\588303-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\588775-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol03\\588912-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\590262-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\590642-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\591144-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\591444-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\591529-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\593558-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\593613-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\593625-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\594492-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\595176-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\595279-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\595740-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\596077-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\596970-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\597509-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\597514-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\597547-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\598322-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\600136-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\600139-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\600142-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\600582-6.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\600771-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\601065-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\601122-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\601363-5.npy',
'Preprocessed_Data\\KneeMRI\\vol03\\602331-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\602341-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\605630-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\606059-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\606936-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\606960-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\606988-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\607514-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\608173-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\610597-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\610677-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\613253-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\614613-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\615220-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\615440-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\616156-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\616731-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\618174-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\618362-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\619109-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\619118-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\619143-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\619855-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\619863-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\621974-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol04\\621986-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\622040-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\622688-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\623884-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\624751-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\624797-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\626139-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\626422-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\626645-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\626657-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\627639-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\627649-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\629160-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\629729-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\630135-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\630633-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\630664-6.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\630685-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\630694-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\632579-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\633387-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\633759-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\633927-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\634187-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\634511-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\635516-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\638165-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\638190-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\638231-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\638534-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\641111-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\641156-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\641768-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\641907-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\642414-9.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\642940-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\643623-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\644191-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\644883-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\644982-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\645117-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\645849-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\647723-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\647779-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\648536-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\649208-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\649280-8.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\652255-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\656270-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\661531-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\662323-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\663898-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol04\\663927-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\666063-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\666261-8.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\666927-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\666931-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\669290-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\670444-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\670486-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\670591-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\670898-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\671254-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\671274-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\671572-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\673486-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\673588-5.npy',
'Preprocessed_Data\\KneeMRI\\vol04\\673647-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\676034-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\676105-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\676111-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\676592-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\676612-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\678613-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\678660-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\678671-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\682580-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\682602-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\683028-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\683028-8.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\683478-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\683991-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\685814-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\685842-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\685860-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\688218-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\688808-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\688833-6.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\688838-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\688886-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\689147-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\690314-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\700481-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\700785-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\702580-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\703073-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\703130-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\703798-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\704348-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\704469-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\705080-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\705114-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\705138-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\705666-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol05\\706967-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\707171-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\710314-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\710395-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\714569-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\716903-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\717337-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\717994-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\717996-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\718030-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\718175-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\722177-9.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\723427-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\725464-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\725506-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\727832-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\728433-9.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\730378-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\730431-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\730439-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\730451-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\731748-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\732220-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\733146-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\733178-6.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\733198-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\733220-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\733786-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\733828-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\737151-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\737155-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\737223-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\737279-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\739006-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\739458-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\739469-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\739683-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\739740-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\739867-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\740062-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\742102-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\742110-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\742136-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\742140-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\742863-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\743503-7.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\745206-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\748033-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\748489-6.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\749555-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\749611-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\749859-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol05\\749920-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\750313-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\751087-5.npy',
'Preprocessed_Data\\KneeMRI\\vol05\\751529-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\752201-6.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\753060-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\753175-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\753546-6.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\754063-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\754074-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\754081-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\754414-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\755231-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\756618-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\756622-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\759404-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\759420-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\759955-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\760062-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\761207-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\762092-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\762151-6.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\762215-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\762582-6.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\763294-6.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\763299-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\764536-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\764603-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\765512-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\766154-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\766193-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\766315-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\766889-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\766928-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\768351-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\768362-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\768443-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\768954-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\769061-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\769102-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\771150-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\772543-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\773936-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\773950-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\773978-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\774012-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\774080-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\774371-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\775307-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\775451-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\776202-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\776592-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol06\\776645-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\777091-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\777830-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\778445-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\779006-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\779507-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\779575-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\780129-6.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\780859-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\780871-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\781686-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\782329-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\782432-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\782831-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\783758-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\784261-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\785256-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\785277-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\785283-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\785645-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\786061-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\786518-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\788123-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\788573-7.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\790498-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\790941-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\791130-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\791491-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\791552-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\792881-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\793887-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\793894-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\796281-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\796904-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\796909-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\799297-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\799309-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\799331-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\799848-6.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\799918-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\799963-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\800042-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\800047-5.npy',
'Preprocessed_Data\\KneeMRI\\vol06\\800663-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\802346-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\804892-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\805533-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\805593-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\807255-7.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\807444-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\808414-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\808420-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol07\\808452-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\809032-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\811641-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\813960-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\813983-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\813989-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\814006-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\814313-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\814634-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\815949-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\815981-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\816937-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\816960-8.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\817106-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\818787-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\819861-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\819869-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\819924-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\820094-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\820295-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\822166-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\822233-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\822268-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\827588-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\829360-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\829819-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\830332-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\832419-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\832423-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\833353-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\833354-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\834905-15.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\834905-20.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\837349-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\837404-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\837781-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\838219-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\838439-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\838952-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\839006-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\839015-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\842580-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\842584-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\844204-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\845068-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\845090-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\845188-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\845357-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\845391-9.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\847797-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\847907-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\850556-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol07\\850595-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\852918-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\852956-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\852996-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\854579-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\855548-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\855943-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\857262-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\858148-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\858166-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\858210-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\858216-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\859300-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\859575-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\859614-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\860327-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\860375-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\860395-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\860535-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\861053-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\861054-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863119-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863127-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863175-6.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863191-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863205-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863216-6.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863380-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863474-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\863766-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\864453-5.npy',
'Preprocessed_Data\\KneeMRI\\vol07\\865914-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\865989-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\866308-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\867362-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\869584-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\870978-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\871117-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\872088-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\873477-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\873527-8.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\874480-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\877360-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\878496-8.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\878909-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\879816-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\879895-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\879896-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\882446-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\882468-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\883170-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\884521-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol08\\885884-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\885904-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\886009-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\886869-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\887655-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\887721-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\888105-7.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\888167-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\888173-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\888187-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\891610-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\891633-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\891672-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\891705-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\891712-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\891774-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\891784-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\891832-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\893814-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\893933-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\895184-8.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\895196-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\895283-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\895401-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\895625-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\896124-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\896562-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\896573-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\896614-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\899763-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\899797-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\900653-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\901171-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\902667-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\903103-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\903145-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\903167-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\905147-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\905220-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\905717-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\907426-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\907434-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\907920-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\907924-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\907936-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\907942-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\913605-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\913634-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\913644-3.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\913673-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\913701-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\915116-7.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol08\\915369-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\915722-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\916438-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\916445-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\917040-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\917214-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\917810-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919157-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919181-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919194-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919200-6.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919210-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919218-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919772-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919776-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919778-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919780-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919791-5.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919791-8.npy',
'Preprocessed_Data\\KneeMRI\\vol08\\919795-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\919803-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\919809-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\919812-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\919858-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\922655-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\922799-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\922927-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\923490-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\924901-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\924915-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\927081-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\927817-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\928454-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\929002-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\929004-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\929005-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\929011-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\929014-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\929017-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\929038-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\931418-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\931491-6.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\931506-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\931509-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\932249-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\932265-6.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\932294-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\932304-6.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\932305-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\932338-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\933325-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\934291-5.npy',
```

```
'Preprocessed_Data\\KneeMRI\\vol09\\934306-6.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\934316-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\934433-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\935664-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\936150-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\936593-6.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\936646-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\936714-6.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\938663-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\942595-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\944779-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\944819-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\947642-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\947843-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\950624-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\950665-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\950719-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\951154-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\951237-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\953440-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\953493-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\953522-6.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\954247-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\954254-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\956370-11.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\956377-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\957038-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\961627-6.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\961740-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\961791-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\961793-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\962019-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\963158-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\963229-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\963545-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\964580-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\964652-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\967475-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\967499-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\967529-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\967556-7.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\967579-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\969135-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\970147-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\972492-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\972891-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\972912-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\972935-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\972960-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\973377-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\975352-8.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\978118-5.npy',
```

'Preprocessed_Data\\KneeMRI\\vol09\\978128-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\978151-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\978170-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\980285-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\980576-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\980623-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\980967-5.npy',
'Preprocessed_Data\\KneeMRI\\vol09\\980969-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1001866-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1001888-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1001946-8.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1002625-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1004948-6.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1004960-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1005020-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1005096-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1006832-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1007746-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1007800-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1008543-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1009152-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1009387-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1009446-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1009764-7.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1010656-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1010675-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1013324-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1013391-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1013460-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1013500-6.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1016062-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1016187-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1016203-6.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1016907-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1018997-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1020681-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1020734-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1022164-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1022186-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1022274-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1022291-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1022353-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1024141-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1025026-7.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1025078-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1025091-8.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1025100-6.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1026369-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1026388-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1026406-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1026420-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1026473-5.npy',

```
'Preprocessed_Data\\KneeMRI\\vol10\\1026480-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1026521-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1027085-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1027212-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1028019-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1028028-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1028069-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\1028670-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\982731-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\982808-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\982950-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\983216-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\984110-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\984116-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\984130-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\984177-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\984210-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\984275-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\984295-6.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\985215-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\985256-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\985284-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\987812-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\987832-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\987852-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\990617-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\990618-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\990655-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\992575-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\993036-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\993043-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\993422-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\993435-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\994220-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\994275-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\994846-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\994861-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\995153-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\996311-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\996739-6.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\997658-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\997668-6.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\999594-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\999657-5.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\999916-5.npy']
```

```
kneemri_aug_cases = []
for mri_data_path in kneemri_vol_paths:
    if platform.system() == "Windows":
        all_exams = glob(mri_data_path+"\\aug\\*.npy")
    else:
        all_exams = glob(mri_data_path+"/aug/*.npy")
```

```
all_exams.sort()
kneemri_aug_cases.extend(all_exams)

print('Augmented cases : ', len(kneemri_aug_cases))

Augmented cases : 650

kneemri_aug_cases

['Preprocessed_Data\\KneeMRI\\vol01\\aug\\404663-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\404663-8-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\412857-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\412865-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\412865-8-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\451760-9-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\451760-9-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\454319-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\454319-8-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\457511-7-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\457511-7-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\457519-6-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\457519-6-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\457531-6-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\457531-6-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\457572-6-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\457572-6-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\463541-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\463541-8-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\463773-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\463773-8-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\463773-8-aug-2.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\463773-8-aug-3.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\463773-8-aug-4.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\470658-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\472130-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\472130-8-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\472130-8-aug-2.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\472130-8-aug-3.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\472130-8-aug-4.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\475047-7-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\476260-7-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\476260-7-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\476260-7-aug-2.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\476260-7-aug-3.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\476260-7-aug-4.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\477390-8-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\477390-8-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\479150-7-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\479150-7-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\479684-7-aug-0.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\479684-7-aug-1.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\aug\\479697-8-aug-0.npy']
```



```

'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1022353-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1022353-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1025026-7-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1025026-7-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1025091-8-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1025091-8-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1026406-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1026406-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1026420-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1026420-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1026521-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1026521-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1027085-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1027085-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1027212-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1027212-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1028019-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1028019-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1028670-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\1028670-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\982950-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\982950-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\987852-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\987852-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\987852-5-aug-2.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\987852-5-aug-3.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\987852-5-aug-4.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\994275-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\994275-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\995153-5-aug-0.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\995153-5-aug-1.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\995153-5-aug-2.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\995153-5-aug-3.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\995153-5-aug-4.npy',
'Preprocessed_Data\\KneeMRI\\vol10\\aug\\996739-6-aug-0.npy']

# names=True loads the interprets the first row of csv file as column names
# 'i4' = 4 byte signed integer, 'U20' = unicode max 20 char string
kneemri_metadata = np.genfromtxt(kneemri_metadata_csv_path, delimiter=',',
names=True,
dtype='i4,i4,i4,i4,i4,i4,i4,i4,U20')

kneemri_metadata_df = pd.DataFrame(kneemri_metadata)
kneemri_metadata_df

      examId  seriesNo  aclDiagnosis  kneeLR  roiX  roiY  roiZ  roiHeight \
0    329637        8            0       1   139   184    14        74
1    390116        9            0       0   113   105    10        83
2    404663        8            1       1   120   117    15       101
3    406320        9            0       0   117   124    12        91
4    412857        8            0       1   122   105    14        83

```

```

. . . . .
912 1027212      5       1       1   113   127   16   101
913 1028019      5       1       1   105   102   14    95
914 1028028      5       0       0   118   84    15   100
915 1028069      5       0       0   105   97    15   103
916 1028670      5       1       0   113   108   14   103

  roiWidth  roiDepth volumeFilename
0        72          3 329637-8.pck
1        98          6 390116-9.pck
2       115          2 404663-8.pck
3        80          3 406320-9.pck
4        98          4 412857-8.pck
. . .
912       99          3 1027212-5.pck
913      100          3 1028019-5.pck
914      100          2 1028028-5.pck
915      106          4 1028069-5.pck
916      110          4 1028670-5.pck

```

[917 rows x 11 columns]

`kneemri_metadata_df.info()`

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 917 entries, 0 to 916
Data columns (total 11 columns):
 #  Column            Non-Null Count  Dtype  
--- 
 0  examId           917 non-null    int32  
 1  seriesNo         917 non-null    int32  
 2  aclDiagnosis    917 non-null    int32  
 3  kneeLR          917 non-null    int32  
 4  roiX             917 non-null    int32  
 5  roiY             917 non-null    int32  
 6  roiZ             917 non-null    int32  
 7  roiHeight        917 non-null    int32  
 8  roiWidth         917 non-null    int32  
 9  roiDepth         917 non-null    int32  
 10  volumeFilename  917 non-null    object 
dtypes: int32(10), object(1)
memory usage: 43.1+ KB

```

`# metadata_df['examId'].nunique()`

`kneemri_metadata_df['volumeFilename'].nunique()`

917

`# metadata_df['examId'].value_counts()`

```

kneemri_aug_metadata_df = pd.read_csv(kneemri_aug_metadata_csv_path,
index_col=0)

```

```
kneemri_aug_metadata_df
```

	examId	seriesNo	aclDiagnosis	kneeLR	roiX	roiY	roiZ	roiHeight	\
0	404663	8	1	1	120	117	15	101	
1	404663	8	1	1	120	117	15	101	
2	412857	8	0	0	122	105	14	83	
3	412865	8	1	0	111	133	13	78	
4	412865	8	1	0	111	133	13	78	
.
645	995153	5	2	1	113	122	13	93	
646	995153	5	2	1	113	122	13	93	
647	995153	5	2	1	113	122	13	93	
648	995153	5	2	1	113	122	13	93	
649	996739	6	0	0	121	135	14	96	
	roiWidth	roiDepth		volumeFilename					
0	115	2	404663-8-aug-0.pck						
1	115	2	404663-8-aug-1.pck						
2	98	4	412857-8-aug-0.pck						
3	78	4	412865-8-aug-0.pck						
4	78	4	412865-8-aug-1.pck						
.
645	98	3	995153-5-aug-1.pck						
646	98	3	995153-5-aug-2.pck						
647	98	3	995153-5-aug-3.pck						
648	98	3	995153-5-aug-4.pck						
649	92	3	996739-6-aug-0.pck						

```
[650 rows x 11 columns]
```

```
kneemri_metadata_df[kneemri_metadata_df['examId']==584216]
```

	examId	seriesNo	aclDiagnosis	kneeLR	roiX	roiY	roiZ	roiHeight	\
238	584216	5	0	0	135	125	12	74	
239	584216	8	0	0	133	121	11	80	
240	584216	9	0	0	131	137	12	66	
	roiWidth	roiDepth	volumeFilename						
238	72	4	584216-5.pck						
239	88	5	584216-8.pck						
240	64	3	584216-9.pck						

```
kneemri_metadata_df[kneemri_metadata_df['examId']==404663]
```

	examId	seriesNo	aclDiagnosis	kneeLR	roiX	roiY	roiZ	roiHeight	\
2	404663	8	1	1	120	117	15	101	
	roiWidth	roiDepth	volumeFilename						
2	115	2	404663-8.pck						

```
kneemri_aug_metadata_df[kneemri_aug_metadata_df['examId']==404663]
```

```

examId  seriesNo  aclDiagnosis  kneeLR  roiX  roiY  roiZ  roiHeight  \
0  404663          8            1        1   120    117    15      101
1  404663          8            1        1   120    117    15      101

roiWidth  roiDepth  volumeFilename
0         115        2  404663-8-aug-0.pck
1         115        2  404663-8-aug-1.pck

kneemri_full_metadata_df = pd.concat([kneemri_metadata_df,
kneemri_aug_metadata_df], ignore_index=True)

kneemri_full_metadata_df

examId  seriesNo  aclDiagnosis  kneeLR  roiX  roiY  roiZ  roiHeight  \
0  329637          8            0        1   139    184    14      74
1  390116          9            0        0   113    105    10      83
2  404663          8            1        1   120    117    15      101
3  406320          9            0        0   117    124    12      91
4  412857          8            0        1   122    105    14      83
...
1562  995153          5            2        1   113    122    13      93
1563  995153          5            2        1   113    122    13      93
1564  995153          5            2        1   113    122    13      93
1565  995153          5            2        1   113    122    13      93
1566  996739          6            0        0   121    135    14      96

roiWidth  roiDepth  volumeFilename
0         72        3  329637-8.pck
1         98        6  390116-9.pck
2         115       2  404663-8.pck
3         80        3  406320-9.pck
4         98        4  412857-8.pck
...
1562  98        3  995153-5-aug-1.pck
1563  98        3  995153-5-aug-2.pck
1564  98        3  995153-5-aug-3.pck
1565  98        3  995153-5-aug-4.pck
1566  92        3  996739-6-aug-0.pck

[1567 rows x 11 columns]

# metadata_df.loc[metadata_df['volumeFilename']=='584216-5.pck']

kneemri_filenames = []
kneemri_filenames.extend(kneemri_cases)
kneemri_filenames.extend(kneemri_aug_cases)
kneemri_filenames.sort()

print(len(kneemri_filenames))

1567

```

```

# label = full_metadata_df.loc[full_metadata_df['volumeFilename']=='584216-5.pck',]

# full_metadata_df[full_metadata_df['examId']==584216]

# labels = full_metadata_df['aclDiagnosis'].tolist()
kneemri_labels = utils.get_correct_labels_kneemri(kneemri_filenames,
kneemri_full_metadata_df)

print(len(kneemri_labels))

1567

kneemri_filenames[:5]

['Preprocessed_Data\\KneeMRI\\vol01\\329637-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\390116-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\404663-8.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\406320-9.npy',
 'Preprocessed_Data\\KneeMRI\\vol01\\412857-8.npy']

kneemri_labels[:5]

[0, 0, 1, 0, 0]

BATCH_SIZE = 8
EPOCHS = 100

# Quick check of counts of samples for each case
[[x, kneemri_labels.count(x)] for x in set(kneemri_labels)]

[[0, 721], [1, 516], [2, 330]]

# Splitting into train, test and validation

X, X_test, y, y_test = train_test_split(kneemri_filenames,
                                         kneemri_labels,
                                         test_size=0.1,
                                         random_state=610,
                                         shuffle=True,
                                         stratify=kneemri_labels)

X_train, X_valid, y_train, y_valid = train_test_split(X,
                                                       y,
                                                       train_size=0.7,
                                                       random_state=610,
                                                       shuffle=True,
                                                       stratify=y)

print(len(X))
print(len(X_test))
print(len(y))
print(len(y_test))

```

```
1410
157
1410
157

print(len(X_train))
print(len(X_valid))
print(len(y_train))
print(len(y_valid))

986
424
986
424

# CASES
print(len(X_train))
print(len(X_test))
print(len(X_valid))
print(len(X_train) + len(X_test) + len(X_valid))

986
157
424
1567

# LABELS
print(len(y_train))
print(len(y_test))
print(len(y_valid))
print(len(y_train) + len(y_test) + len(y_valid))

986
157
424
1567

tf.executing_eagerly()

True

kneemri_class_weights = utils.compute_class_weights(y_train)
kneemri_class_weights
{0: 0.723935389133627, 1: 1.01440329218107, 2: 1.580128205128205}

model_name = 'kneeMRI_Modell1'
kneeMRI_Modell1 = models.mri_model_1(model_name, len(kneemri_classes))
kneeMRI_Modell1.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                        loss='sparse_categorical_crossentropy',
                        metrics=['accuracy'])
kneeMRI_Modell1.summary()
```

Model: "kneeMRI_Model1"

Layer (type)	Output Shape	Param #
conv3d (Conv3D)	(None, 30, 256, 256, 64)	1792
max_pooling3d (MaxPooling3D)	(None, 15, 128, 128, 64)	0
batch_normalization (BatchNormalization)	(None, 15, 128, 128, 64)	256
conv3d_1 (Conv3D)	(None, 15, 128, 128, 64)	110656
max_pooling3d_1 (MaxPooling3D)	(None, 8, 64, 64, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 8, 64, 64, 64)	256
conv3d_2 (Conv3D)	(None, 8, 64, 64, 128)	221312
max_pooling3d_2 (MaxPooling3D)	(None, 4, 32, 32, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 4, 32, 32, 128)	512
conv3d_3 (Conv3D)	(None, 4, 32, 32, 128)	442496
max_pooling3d_3 (MaxPooling3D)	(None, 2, 16, 16, 128)	0
batch_normalization_3 (BatchNormalization)	(None, 2, 16, 16, 128)	512
conv3d_4 (Conv3D)	(None, 2, 16, 16, 256)	884992
max_pooling3d_4 (MaxPooling3D)	(None, 1, 8, 8, 256)	0
batch_normalization_4 (BatchNormalization)	(None, 1, 8, 8, 256)	1024
global_average_pooling3d (GlobalAveragePooling3D)	(None, 256)	0
dense (Dense)	(None, 512)	131584
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328

<i>dropout_1</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_2</i> (<i>Dense</i>)	(None, 3)	771

Total params: 1,927,491
Trainable params: 1,926,211
Non-trainable params: 1,280

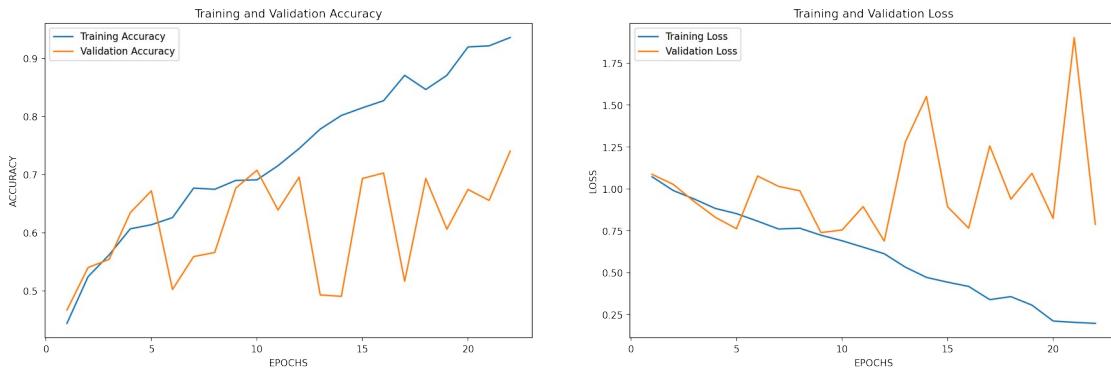
```
%%time
with tf.device('/device:GPU:0'):
    history = kneeMRI_Model1.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EPOCHS,
                                validation_data=utils.batch_generator(X_valid, y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=kneemri_class_weights,
                                verbose=1,
                                callbacks=[utils.model_callback_checkpoint(model_name),
                               utils.model_callback_earlystopping()])

Epoch 1/100
123/123 [=====] - 164s 889ms/step - loss: 1.0721 - accuracy: 0.4441 - val_loss: 1.0866 - val_accuracy: 0.4670
Epoch 2/100
123/123 [=====] - 106s 861ms/step - loss: 0.9903 - accuracy: 0.5244 - val_loss: 1.0270 - val_accuracy: 0.5401
Epoch 3/100
123/123 [=====] - 106s 863ms/step - loss: 0.9391 - accuracy: 0.5620 - val_loss: 0.9238 - val_accuracy: 0.5542
Epoch 4/100
123/123 [=====] - 106s 861ms/step - loss: 0.8830 - accuracy: 0.6067 - val_loss: 0.8299 - val_accuracy: 0.6344
Epoch 5/100
123/123 [=====] - 106s 861ms/step - loss: 0.8521 - accuracy: 0.6138 - val_loss: 0.7622 - val_accuracy: 0.6722
Epoch 6/100
123/123 [=====] - 106s 859ms/step - loss: 0.8072 - accuracy: 0.6260 - val_loss: 1.0764 - val_accuracy: 0.5024
Epoch 7/100
123/123 [=====] - 106s 863ms/step - loss: 0.7602 - accuracy: 0.6768 - val_loss: 1.0144 - val_accuracy: 0.5590
Epoch 8/100
123/123 [=====] - 106s 860ms/step - loss: 0.7649 - accuracy: 0.6748 - val_loss: 0.9885 - val_accuracy: 0.5660
Epoch 9/100
```

```
123/123 [=====] - 106s 864ms/step - loss: 0.7234 -  
accuracy: 0.6900 - val_loss: 0.7389 - val_accuracy: 0.6769  
Epoch 10/100  
123/123 [=====] - 106s 859ms/step - loss: 0.6903 -  
accuracy: 0.6911 - val_loss: 0.7542 - val_accuracy: 0.7075  
Epoch 11/100  
123/123 [=====] - 106s 858ms/step - loss: 0.6521 -  
accuracy: 0.7154 - val_loss: 0.8938 - val_accuracy: 0.6392  
Epoch 12/100  
123/123 [=====] - 106s 864ms/step - loss: 0.6128 -  
accuracy: 0.7449 - val_loss: 0.6882 - val_accuracy: 0.6958  
Epoch 13/100  
123/123 [=====] - 106s 861ms/step - loss: 0.5327 -  
accuracy: 0.7785 - val_loss: 1.2800 - val_accuracy: 0.4929  
Epoch 14/100  
123/123 [=====] - 106s 862ms/step - loss: 0.4717 -  
accuracy: 0.8018 - val_loss: 1.5509 - val_accuracy: 0.4906  
Epoch 15/100  
123/123 [=====] - 106s 865ms/step - loss: 0.4430 -  
accuracy: 0.8150 - val_loss: 0.8932 - val_accuracy: 0.6934  
Epoch 16/100  
123/123 [=====] - 106s 864ms/step - loss: 0.4177 -  
accuracy: 0.8272 - val_loss: 0.7653 - val_accuracy: 0.7028  
Epoch 17/100  
123/123 [=====] - 107s 867ms/step - loss: 0.3389 -  
accuracy: 0.8709 - val_loss: 1.2561 - val_accuracy: 0.5165  
Epoch 18/100  
123/123 [=====] - 106s 864ms/step - loss: 0.3568 -  
accuracy: 0.8465 - val_loss: 0.9385 - val_accuracy: 0.6934  
Epoch 19/100  
123/123 [=====] - 106s 861ms/step - loss: 0.3059 -  
accuracy: 0.8709 - val_loss: 1.0925 - val_accuracy: 0.6061  
Epoch 20/100  
123/123 [=====] - 106s 864ms/step - loss: 0.2115 -  
accuracy: 0.9197 - val_loss: 0.8239 - val_accuracy: 0.6745  
Epoch 21/100  
123/123 [=====] - 106s 862ms/step - loss: 0.2031 -  
accuracy: 0.9217 - val_loss: 1.9029 - val_accuracy: 0.6557  
Epoch 22/100  
123/123 [=====] - ETA: 0s - loss: 0.1975 - accuracy:  
0.9360Restoring model weights from the end of the best epoch: 12.  
123/123 [=====] - 106s 864ms/step - loss: 0.1975 -  
accuracy: 0.9360 - val_loss: 0.7883 - val_accuracy: 0.7406  
Epoch 22: early stopping  
Wall time: 39min 50s
```

Store history
utils.store_model_history(model_name, history.history)

Plot training graphs
utils.plot_acc_loss(history.history)



Evaluate model

```
X_test_prob = kneeMRI_Model1.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))
```

```
X_test_pred = X_test_prob.argmax(axis=-1)
```

```
utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])
```

```
20/20 [=====] - 11s 547ms/step
```

Evaluation Metrics:

Balanced Accuracy : 0.75

Precision : 0.78

Recall : 0.75

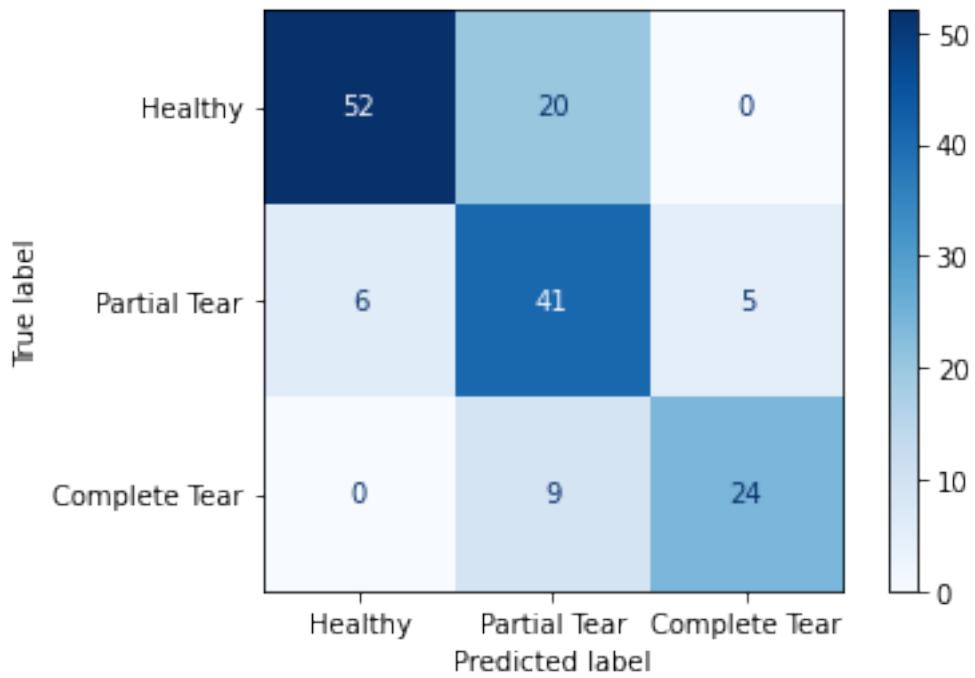
F1 Score: 0.75

ROC AUC Score : 0.9

Classification report :

	precision	recall	f1-score	support
Healthy	0.90	0.72	0.80	72
Partial Tear	0.59	0.79	0.67	52
Complete Tear	0.83	0.73	0.77	33
accuracy			0.75	157
macro avg	0.77	0.75	0.75	157
weighted avg	0.78	0.75	0.75	157

Confusion Matrix :



```

model_name = 'kneeMRI_Model2'
kneeMRI_Model2 = models.mri_model_2(model_name, len(kneemri_classes))
kneeMRI_Model2.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
kneeMRI_Model2.summary()

```

Model: "kneeMRI_Model2"

Layer (type)	Output Shape	Param #
conv3d_5 (Conv3D)	(None, 30, 256, 256, 16)	448
max_pooling3d_5 (MaxPooling 3D)	(None, 15, 128, 128, 16)	0
batch_normalization_5 (BatchNormalization)	(None, 15, 128, 128, 16)	64
conv3d_6 (Conv3D)	(None, 15, 128, 128, 32)	13856
max_pooling3d_6 (MaxPooling 3D)	(None, 8, 64, 64, 32)	0
batch_normalization_6 (BatchNormalization)	(None, 8, 64, 64, 32)	128
conv3d_7 (Conv3D)	(None, 8, 64, 64, 32)	27680
max_pooling3d_7 (MaxPooling 3D)	(None, 4, 32, 32, 32)	0

3D)

<i>batch_normalization_7</i> (<i>BatchNormalization</i>)	(None, 4, 32, 32, 32)	128
<i>conv3d_8</i> (<i>Conv3D</i>)	(None, 4, 32, 32, 64)	55360
<i>max_pooling3d_8</i> (<i>MaxPooling3D</i>)	(None, 2, 16, 16, 64)	0
<i>batch_normalization_8</i> (<i>BatchNormalization</i>)	(None, 2, 16, 16, 64)	256
<i>conv3d_9</i> (<i>Conv3D</i>)	(None, 2, 16, 16, 64)	110656
<i>max_pooling3d_9</i> (<i>MaxPooling3D</i>)	(None, 1, 8, 8, 64)	0
<i>batch_normalization_9</i> (<i>BatchNormalization</i>)	(None, 1, 8, 8, 64)	256
<i>conv3d_10</i> (<i>Conv3D</i>)	(None, 1, 8, 8, 128)	221312
<i>max_pooling3d_10</i> (<i>MaxPooling3D</i>)	(None, 1, 4, 4, 128)	0
<i>batch_normalization_10</i> (<i>BatchNormalization</i>)	(None, 1, 4, 4, 128)	512
<i>conv3d_11</i> (<i>Conv3D</i>)	(None, 1, 4, 4, 128)	442496
<i>max_pooling3d_11</i> (<i>MaxPooling3D</i>)	(None, 1, 2, 2, 128)	0
<i>batch_normalization_11</i> (<i>BatchNormalization</i>)	(None, 1, 2, 2, 128)	512
<i>conv3d_12</i> (<i>Conv3D</i>)	(None, 1, 2, 2, 256)	884992
<i>max_pooling3d_12</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>batch_normalization_12</i> (<i>BatchNormalization</i>)	(None, 1, 1, 1, 256)	1024
<i>global_average_pooling3d_1</i> (<i>GlobalAveragePooling3D</i>)	(None, 256)	0
<i>dense_3</i> (<i>Dense</i>)	(None, 512)	131584
<i>dropout_2</i> (<i>Dropout</i>)	(None, 512)	0

<i>dense_4</i> (<i>Dense</i>)	(<i>None</i> , 256)	131328
<i>dropout_3</i> (<i>Dropout</i>)	(<i>None</i> , 256)	0
<i>dense_5</i> (<i>Dense</i>)	(<i>None</i> , 3)	771

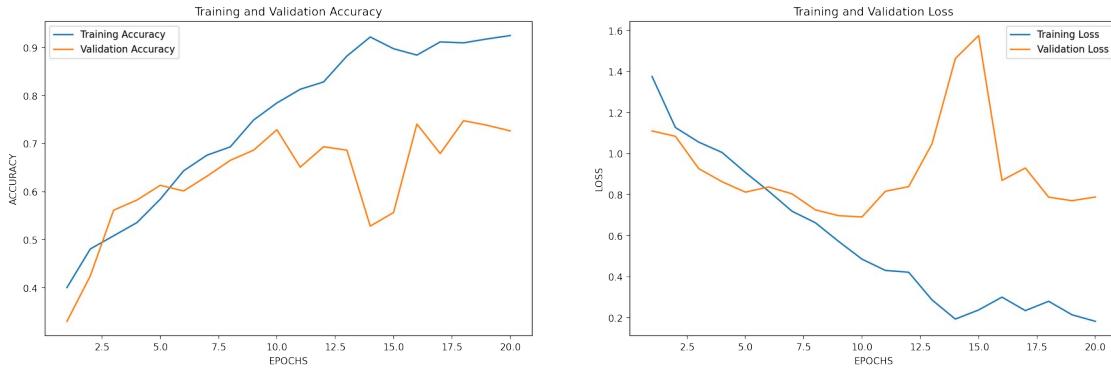
Total params: 2,023,363
Trainable params: 2,021,923
Non-trainable params: 1,440

```
%%time
with tf.device('/device:GPU:0'):
    history = kneeMRI_Model2.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EP0CHS,
validation_data=utils.batch_generator(X_valid, y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=kneemri_class_weights,
                                verbose=1,
callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()])
Epoch 1/100
123/123 [=====] - 56s 420ms/step - loss: 1.3754 - accuracy: 0.4004 - val_loss: 1.1102 - val_accuracy: 0.3302
Epoch 2/100
123/123 [=====] - 48s 391ms/step - loss: 1.1271 - accuracy: 0.4807 - val_loss: 1.0842 - val_accuracy: 0.4245
Epoch 3/100
123/123 [=====] - 48s 392ms/step - loss: 1.0558 - accuracy: 0.5081 - val_loss: 0.9268 - val_accuracy: 0.5613
Epoch 4/100
123/123 [=====] - 48s 392ms/step - loss: 1.0054 - accuracy: 0.5356 - val_loss: 0.8628 - val_accuracy: 0.5825
Epoch 5/100
123/123 [=====] - 48s 392ms/step - loss: 0.9080 - accuracy: 0.5843 - val_loss: 0.8121 - val_accuracy: 0.6132
Epoch 6/100
123/123 [=====] - 48s 392ms/step - loss: 0.8166 - accuracy: 0.6433 - val_loss: 0.8373 - val_accuracy: 0.6014
Epoch 7/100
123/123 [=====] - 48s 394ms/step - loss: 0.7189 - accuracy: 0.6758 - val_loss: 0.8038 - val_accuracy: 0.6321
Epoch 8/100
123/123 [=====] - 48s 394ms/step - loss: 0.6630 - accuracy: 0.6931 - val_loss: 0.7258 - val_accuracy: 0.6651
```

```
Epoch 9/100
123/123 [=====] - 49s 395ms/step - loss: 0.5717 -
accuracy: 0.7490 - val_loss: 0.6973 - val_accuracy: 0.6863
Epoch 10/100
123/123 [=====] - 49s 395ms/step - loss: 0.4854 -
accuracy: 0.7846 - val_loss: 0.6911 - val_accuracy: 0.7288
Epoch 11/100
123/123 [=====] - 48s 394ms/step - loss: 0.4307 -
accuracy: 0.8130 - val_loss: 0.8162 - val_accuracy: 0.6509
Epoch 12/100
123/123 [=====] - 49s 395ms/step - loss: 0.4216 -
accuracy: 0.8283 - val_loss: 0.8389 - val_accuracy: 0.6934
Epoch 13/100
123/123 [=====] - 49s 395ms/step - loss: 0.2868 -
accuracy: 0.8821 - val_loss: 1.0475 - val_accuracy: 0.6863
Epoch 14/100
123/123 [=====] - 49s 395ms/step - loss: 0.1934 -
accuracy: 0.9217 - val_loss: 1.4631 - val_accuracy: 0.5283
Epoch 15/100
123/123 [=====] - 49s 396ms/step - loss: 0.2381 -
accuracy: 0.8974 - val_loss: 1.5751 - val_accuracy: 0.5566
Epoch 16/100
123/123 [=====] - 49s 396ms/step - loss: 0.3005 -
accuracy: 0.8841 - val_loss: 0.8694 - val_accuracy: 0.7406
Epoch 17/100
123/123 [=====] - 49s 396ms/step - loss: 0.2349 -
accuracy: 0.9116 - val_loss: 0.9297 - val_accuracy: 0.6792
Epoch 18/100
123/123 [=====] - 49s 397ms/step - loss: 0.2801 -
accuracy: 0.9096 - val_loss: 0.7876 - val_accuracy: 0.7476
Epoch 19/100
123/123 [=====] - 49s 398ms/step - loss: 0.2144 -
accuracy: 0.9177 - val_loss: 0.7703 - val_accuracy: 0.7382
Epoch 20/100
123/123 [=====] - ETA: 0s - loss: 0.1826 - accuracy:
0.9248Restoring model weights from the end of the best epoch: 10.
123/123 [=====] - 49s 397ms/step - loss: 0.1826 -
accuracy: 0.9248 - val_loss: 0.7884 - val_accuracy: 0.7264
Epoch 20: early stopping
Wall time: 16min 18s
```

```
# Store history
utils.store_model_history(model_name, history.history)
```

```
# Plot training graphs
utils.plot_acc_loss(history.history)
```



Evaluate model

```
X_test_prob = kneeMRI_Model2.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))
```

```
X_test_pred = X_test_prob.argmax(axis=-1)
```

```
utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])
```

20/20 [=====] - 5s 255ms/step

Evaluation Metrics:

Balanced Accuracy : 0.7

Precision : 0.72

Recall : 0.72

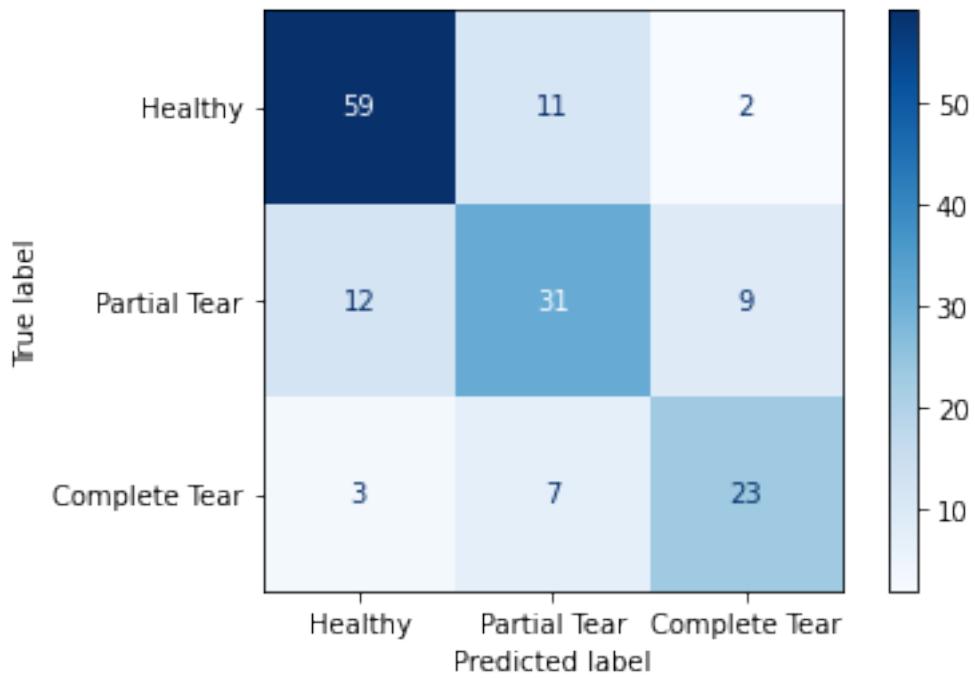
F1 Score: 0.72

ROC AUC Score : 0.87

Classification report :

	precision	recall	f1-score	support
Healthy	0.80	0.82	0.81	72
Partial Tear	0.63	0.60	0.61	52
Complete Tear	0.68	0.70	0.69	33
accuracy			0.72	157
macro avg	0.70	0.70	0.70	157
weighted avg	0.72	0.72	0.72	157

Confusion Matrix :



```

model_name = 'kneeMRI_Model3'
kneeMRI_Model3 = models.mri_model_3(model_name, len(kneemri_classes))
kneeMRI_Model3.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
kneeMRI_Model3.summary()

```

Model: "kneeMRI_Model3"

Layer (type)	Output Shape	Param #
conv3d_13 (Conv3D)	(None, 30, 256, 256, 32)	896
max_pooling3d_13 (MaxPooling3D)	(None, 15, 128, 128, 32)	0
batch_normalization_13 (BatchNormalization)	(None, 15, 128, 128, 32)	128
conv3d_14 (Conv3D)	(None, 15, 128, 128, 32)	27680
max_pooling3d_14 (MaxPooling3D)	(None, 8, 64, 64, 32)	0
batch_normalization_14 (BatchNormalization)	(None, 8, 64, 64, 32)	128
conv3d_15 (Conv3D)	(None, 8, 64, 64, 32)	27680
max_pooling3d_15 (MaxPooling3D)	(None, 4, 32, 32, 32)	0

g3D)

<i>batch_normalization_15 (BatchNormalization)</i>	(None, 4, 32, 32, 32)	128
<i>conv3d_16 (Conv3D)</i>	(None, 4, 32, 32, 64)	55360
<i>max_pooling3d_16 (MaxPooling3D)</i>	(None, 2, 16, 16, 64)	0
<i>batch_normalization_16 (BatchNormalization)</i>	(None, 2, 16, 16, 64)	256
<i>conv3d_17 (Conv3D)</i>	(None, 2, 16, 16, 64)	110656
<i>max_pooling3d_17 (MaxPooling3D)</i>	(None, 1, 8, 8, 64)	0
<i>batch_normalization_17 (BatchNormalization)</i>	(None, 1, 8, 8, 64)	256
<i>conv3d_18 (Conv3D)</i>	(None, 1, 8, 8, 64)	110656
<i>max_pooling3d_18 (MaxPooling3D)</i>	(None, 1, 4, 4, 64)	0
<i>batch_normalization_18 (BatchNormalization)</i>	(None, 1, 4, 4, 64)	256
<i>conv3d_19 (Conv3D)</i>	(None, 1, 4, 4, 128)	221312
<i>max_pooling3d_19 (MaxPooling3D)</i>	(None, 1, 2, 2, 128)	0
<i>batch_normalization_19 (BatchNormalization)</i>	(None, 1, 2, 2, 128)	512
<i>conv3d_20 (Conv3D)</i>	(None, 1, 2, 2, 128)	442496
<i>max_pooling3d_20 (MaxPooling3D)</i>	(None, 1, 1, 1, 128)	0
<i>batch_normalization_20 (BatchNormalization)</i>	(None, 1, 1, 1, 128)	512
<i>conv3d_21 (Conv3D)</i>	(None, 1, 1, 1, 128)	442496
<i>max_pooling3d_21 (MaxPooling3D)</i>	(None, 1, 1, 1, 128)	0
<i>batch_normalization_21 (BatchNormalization)</i>	(None, 1, 1, 1, 128)	512

<i>conv3d_22</i> (<i>Conv3D</i>)	(<i>None</i> , 1, 1, 1, 256)	884992
<i>max_pooling3d_22</i> (<i>MaxPooling3D</i>)	(<i>None</i> , 1, 1, 1, 256)	0
<i>batch_normalization_22</i> (<i>BatchNormalization</i>)	(<i>None</i> , 1, 1, 1, 256)	1024
<i>conv3d_23</i> (<i>Conv3D</i>)	(<i>None</i> , 1, 1, 1, 256)	1769728
<i>max_pooling3d_23</i> (<i>MaxPooling3D</i>)	(<i>None</i> , 1, 1, 1, 256)	0
<i>batch_normalization_23</i> (<i>BatchNormalization</i>)	(<i>None</i> , 1, 1, 1, 256)	1024
<i>global_average_pooling3d_2</i> (<i>GlobalAveragePooling3D</i>)	(<i>None</i> , 256)	0
<i>dense_6</i> (<i>Dense</i>)	(<i>None</i> , 512)	131584
<i>dropout_4</i> (<i>Dropout</i>)	(<i>None</i> , 512)	0
<i>dense_7</i> (<i>Dense</i>)	(<i>None</i> , 512)	262656
<i>dropout_5</i> (<i>Dropout</i>)	(<i>None</i> , 512)	0
<i>dense_8</i> (<i>Dense</i>)	(<i>None</i> , 256)	131328
<i>dropout_6</i> (<i>Dropout</i>)	(<i>None</i> , 256)	0
<i>dense_9</i> (<i>Dense</i>)	(<i>None</i> , 3)	771

Total params: 4,625,027
 Trainable params: 4,622,659
 Non-trainable params: 2,368

```
%%time
with tf.device('/device:GPU:0'):
    history = kneeMRI_Model3.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EP0CHS,
validation_data=utils.batch_generator(X_valid, y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=kneemri_class_weights,
                                verbose=1,
```

```
callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()]

Epoch 1/100
123/123 [=====] - 70s 506ms/step - loss: 1.3134 - accuracy: 0.3648 - val_loss: 1.1139 - val_accuracy: 0.2099
Epoch 2/100
123/123 [=====] - 59s 479ms/step - loss: 1.2042 - accuracy: 0.3791 - val_loss: 1.1148 - val_accuracy: 0.2241
Epoch 3/100
123/123 [=====] - 59s 480ms/step - loss: 1.1494 - accuracy: 0.4228 - val_loss: 1.0978 - val_accuracy: 0.3160
Epoch 4/100
123/123 [=====] - 59s 483ms/step - loss: 1.1123 - accuracy: 0.4278 - val_loss: 1.0534 - val_accuracy: 0.3703
Epoch 5/100
123/123 [=====] - 59s 480ms/step - loss: 1.0407 - accuracy: 0.4583 - val_loss: 0.9555 - val_accuracy: 0.5071
Epoch 6/100
123/123 [=====] - 59s 479ms/step - loss: 0.9900 - accuracy: 0.5041 - val_loss: 0.9588 - val_accuracy: 0.5377
Epoch 7/100
123/123 [=====] - 59s 481ms/step - loss: 0.9775 - accuracy: 0.5132 - val_loss: 0.9359 - val_accuracy: 0.5755
Epoch 8/100
123/123 [=====] - 59s 483ms/step - loss: 0.9591 - accuracy: 0.5346 - val_loss: 0.8901 - val_accuracy: 0.5778
Epoch 9/100
123/123 [=====] - 59s 483ms/step - loss: 0.9008 - accuracy: 0.5762 - val_loss: 0.8480 - val_accuracy: 0.6014
Epoch 10/100
123/123 [=====] - 59s 482ms/step - loss: 0.9175 - accuracy: 0.5356 - val_loss: 0.9178 - val_accuracy: 0.5330
Epoch 11/100
123/123 [=====] - 59s 480ms/step - loss: 0.8605 - accuracy: 0.6077 - val_loss: 0.8905 - val_accuracy: 0.5778
Epoch 12/100
123/123 [=====] - 59s 484ms/step - loss: 0.8357 - accuracy: 0.5915 - val_loss: 0.8322 - val_accuracy: 0.5991
Epoch 13/100
123/123 [=====] - 59s 484ms/step - loss: 0.7905 - accuracy: 0.6230 - val_loss: 0.7847 - val_accuracy: 0.6274
Epoch 14/100
123/123 [=====] - 59s 481ms/step - loss: 0.7203 - accuracy: 0.6474 - val_loss: 0.8740 - val_accuracy: 0.6061
Epoch 15/100
123/123 [=====] - 59s 482ms/step - loss: 0.7246 - accuracy: 0.6575 - val_loss: 0.9144 - val_accuracy: 0.5849
Epoch 16/100
123/123 [=====] - 59s 484ms/step - loss: 0.6960 - accuracy: 0.6504 - val_loss: 0.6910 - val_accuracy: 0.7052
```

```

Epoch 17/100
123/123 [=====] - 59s 483ms/step - loss: 0.6762 -
accuracy: 0.6768 - val_loss: 0.7975 - val_accuracy: 0.6439
Epoch 18/100
123/123 [=====] - 60s 486ms/step - loss: 0.6728 -
accuracy: 0.6900 - val_loss: 0.9928 - val_accuracy: 0.4741
Epoch 19/100
123/123 [=====] - 59s 483ms/step - loss: 0.6462 -
accuracy: 0.6951 - val_loss: 1.0970 - val_accuracy: 0.4458
Epoch 20/100
123/123 [=====] - 62s 502ms/step - loss: 0.6193 -
accuracy: 0.7134 - val_loss: 0.9285 - val_accuracy: 0.5566
Epoch 21/100
123/123 [=====] - 60s 486ms/step - loss: 0.6288 -
accuracy: 0.7175 - val_loss: 0.7445 - val_accuracy: 0.7193
Epoch 22/100
123/123 [=====] - 60s 486ms/step - loss: 0.5884 -
accuracy: 0.7276 - val_loss: 0.7575 - val_accuracy: 0.6981
Epoch 23/100
123/123 [=====] - 60s 485ms/step - loss: 0.5118 -
accuracy: 0.7632 - val_loss: 0.7746 - val_accuracy: 0.6038
Epoch 24/100
123/123 [=====] - 59s 484ms/step - loss: 0.4596 -
accuracy: 0.8059 - val_loss: 0.7074 - val_accuracy: 0.6958
Epoch 25/100
123/123 [=====] - 60s 485ms/step - loss: 0.4846 -
accuracy: 0.7856 - val_loss: 0.7207 - val_accuracy: 0.7028
Epoch 26/100
123/123 [=====] - ETA: 0s - loss: 0.5354 - accuracy:
0.7764Restoring model weights from the end of the best epoch: 16.
123/123 [=====] - 59s 484ms/step - loss: 0.5354 -
accuracy: 0.7764 - val_loss: 0.7261 - val_accuracy: 0.6769
Epoch 26: early stopping
Wall time: 25min 54s

```

```

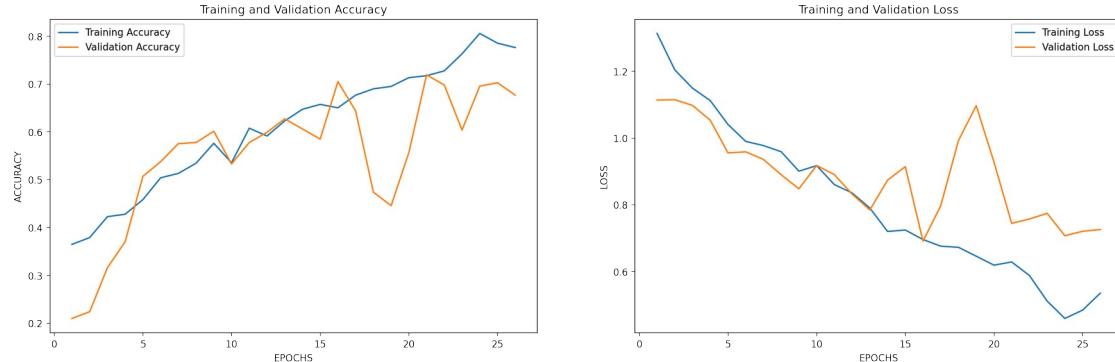
# Store history
utils.store_model_history(model_name, history.history)

```

```

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```
# Evaluate model
X_test_prob = kneeMRI_Model3.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))

X_test_pred = X_test_prob.argmax(axis=-1)

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])

20/20 [=====] - 5s 260ms/step
```

Evaluation Metrics:

Balanced Accuracy : 0.67

Precision : 0.69

Recall : 0.68

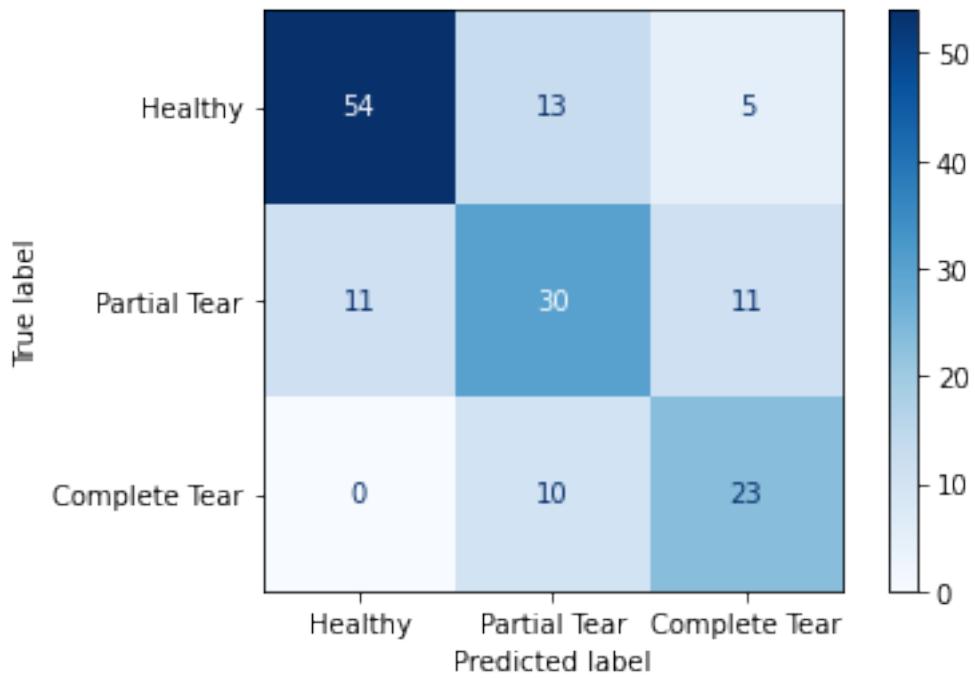
F1 Score: 0.69

ROC AUC Score : 0.85

Classification report :

	precision	recall	f1-score	support
Healthy	0.83	0.75	0.79	72
Partial Tear	0.57	0.58	0.57	52
Complete Tear	0.59	0.70	0.64	33
accuracy			0.68	157
macro avg	0.66	0.67	0.67	157
weighted avg	0.69	0.68	0.69	157

Confusion Matrix :



```

model_name = 'kneeMRI_Model4'
kneeMRI_Model4 = models.mri_model_4(model_name, len(kneemri_classes))
kneeMRI_Model4.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
kneeMRI_Model4.summary()

```

Model: "kneeMRI_Model4"

Layer (type)	Output Shape	Param #
conv3d_24 (Conv3D)	(None, 30, 256, 256, 32)	896
conv3d_25 (Conv3D)	(None, 30, 256, 256, 32)	27680
max_pooling3d_24 (MaxPooling3D)	(None, 15, 128, 128, 32)	0
conv3d_26 (Conv3D)	(None, 15, 128, 128, 32)	27680
conv3d_27 (Conv3D)	(None, 15, 128, 128, 64)	55360
max_pooling3d_25 (MaxPooling3D)	(None, 8, 64, 64, 64)	0
conv3d_28 (Conv3D)	(None, 8, 64, 64, 64)	110656
conv3d_29 (Conv3D)	(None, 8, 64, 64, 128)	221312
max_pooling3d_26 (MaxPooling3D)	(None, 4, 32, 32, 128)	0

g3D)

dropout_7 (Dropout)	(None, 4, 32, 32, 128)	0
flatten (Flatten)	(None, 524288)	0
dense_10 (Dense)	(None, 256)	134217984
dropout_8 (Dropout)	(None, 256)	0
dense_11 (Dense)	(None, 128)	32896
dropout_9 (Dropout)	(None, 128)	0
dense_12 (Dense)	(None, 3)	387

```
=====
Total params: 134,694,851
Trainable params: 134,694,851
Non-trainable params: 0
```

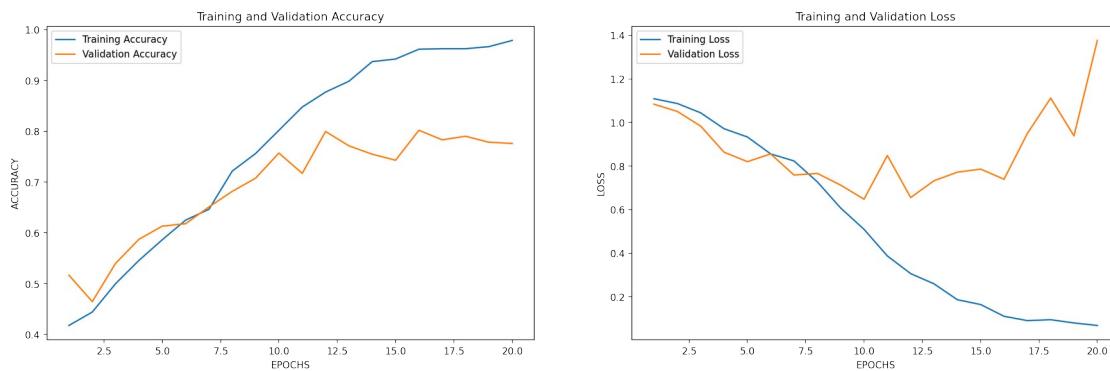
```
%%time
with tf.device('/device:GPU:0'):
    history = kneeMRI_Model4.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EPOCHS,
                                validation_data=utils.batch_generator(X_valid, y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=kneemri_class_weights,
                                verbose=1,
                                callbacks=[utils.model_callback_checkpoint(model_name),
                               utils.model_callback_earlystopping()])

Epoch 1/100
123/123 [=====] - 235s 1s/step - loss: 1.1088 - accuracy: 0.4177 - val_loss: 1.0836 - val_accuracy: 0.5165
Epoch 2/100
123/123 [=====] - 162s 1s/step - loss: 1.0869 - accuracy: 0.4441 - val_loss: 1.0503 - val_accuracy: 0.4646
Epoch 3/100
123/123 [=====] - 161s 1s/step - loss: 1.0441 - accuracy: 0.5000 - val_loss: 0.9830 - val_accuracy: 0.5401
Epoch 4/100
123/123 [=====] - 161s 1s/step - loss: 0.9712 - accuracy: 0.5457 - val_loss: 0.8635 - val_accuracy: 0.5873
Epoch 5/100
123/123 [=====] - 161s 1s/step - loss: 0.9334 -
```

```
accuracy: 0.5864 - val_loss: 0.8201 - val_accuracy: 0.6132
Epoch 6/100
123/123 [=====] - 157s 1s/step - loss: 0.8556 -
accuracy: 0.6250 - val_loss: 0.8555 - val_accuracy: 0.6179
Epoch 7/100
123/123 [=====] - 160s 1s/step - loss: 0.8233 -
accuracy: 0.6463 - val_loss: 0.7586 - val_accuracy: 0.6509
Epoch 8/100
123/123 [=====] - 157s 1s/step - loss: 0.7272 -
accuracy: 0.7215 - val_loss: 0.7662 - val_accuracy: 0.6816
Epoch 9/100
123/123 [=====] - 161s 1s/step - loss: 0.6072 -
accuracy: 0.7561 - val_loss: 0.7123 - val_accuracy: 0.7075
Epoch 10/100
123/123 [=====] - 161s 1s/step - loss: 0.5097 -
accuracy: 0.8018 - val_loss: 0.6478 - val_accuracy: 0.7571
Epoch 11/100
123/123 [=====] - 158s 1s/step - loss: 0.3874 -
accuracy: 0.8476 - val_loss: 0.8484 - val_accuracy: 0.7170
Epoch 12/100
123/123 [=====] - 158s 1s/step - loss: 0.3063 -
accuracy: 0.8770 - val_loss: 0.6554 - val_accuracy: 0.7995
Epoch 13/100
123/123 [=====] - 158s 1s/step - loss: 0.2595 -
accuracy: 0.8984 - val_loss: 0.7326 - val_accuracy: 0.7712
Epoch 14/100
123/123 [=====] - 158s 1s/step - loss: 0.1866 -
accuracy: 0.9370 - val_loss: 0.7718 - val_accuracy: 0.7547
Epoch 15/100
123/123 [=====] - 158s 1s/step - loss: 0.1645 -
accuracy: 0.9421 - val_loss: 0.7859 - val_accuracy: 0.7429
Epoch 16/100
123/123 [=====] - 157s 1s/step - loss: 0.1105 -
accuracy: 0.9614 - val_loss: 0.7393 - val_accuracy: 0.8019
Epoch 17/100
123/123 [=====] - 157s 1s/step - loss: 0.0907 -
accuracy: 0.9624 - val_loss: 0.9491 - val_accuracy: 0.7830
Epoch 18/100
123/123 [=====] - 157s 1s/step - loss: 0.0949 -
accuracy: 0.9624 - val_loss: 1.1117 - val_accuracy: 0.7901
Epoch 19/100
123/123 [=====] - 158s 1s/step - loss: 0.0799 -
accuracy: 0.9665 - val_loss: 0.9385 - val_accuracy: 0.7783
Epoch 20/100
123/123 [=====] - ETA: 0s - loss: 0.0685 - accuracy:
0.9787Restoring model weights from the end of the best epoch: 10.
123/123 [=====] - 159s 1s/step - loss: 0.0685 -
accuracy: 0.9787 - val_loss: 1.3763 - val_accuracy: 0.7759
Epoch 20: early stopping
Wall time: 54min 13s
```

```
# Store history
utils.store_model_history(model_name, history.history)

# Plot training graphs
utils.plot_acc_loss(history.history)
```



```
# Evaluate model
X_test_prob = kneeMRI_Model4.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))

X_test_pred = X_test_prob.argmax(axis=-1)

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])

20/20 [=====] - 10s 533ms/step
```

Evaluation Metrics:

Balanced Accuracy : 0.71

Precision : 0.74

Recall : 0.75

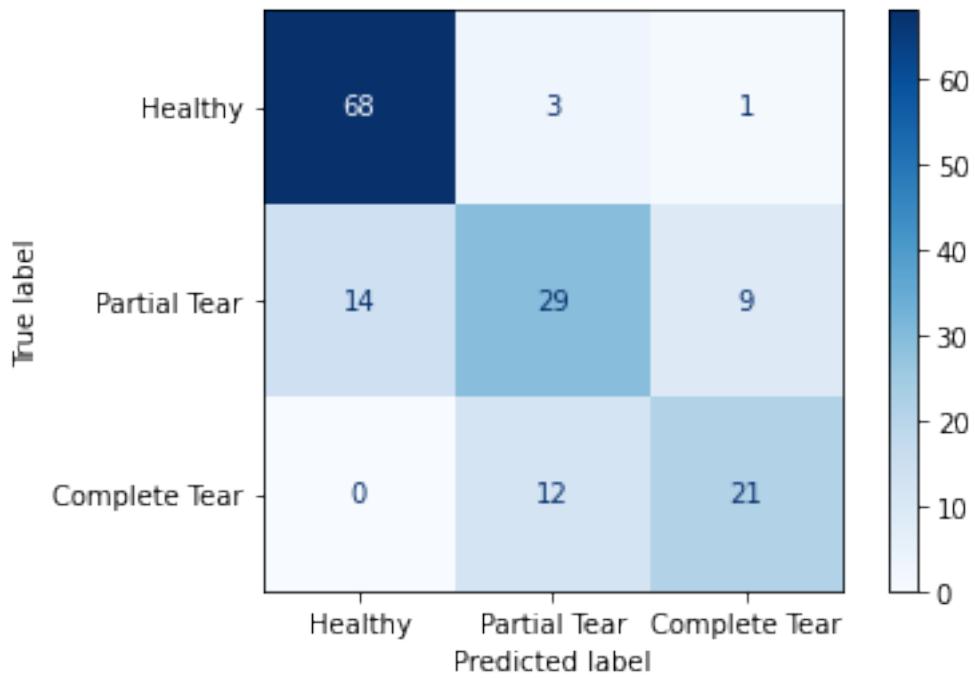
F1 Score: 0.74

ROC AUC Score : 0.91

Classification report :

	precision	recall	f1-score	support
Healthy	0.83	0.94	0.88	72
Partial Tear	0.66	0.56	0.60	52
Complete Tear	0.68	0.64	0.66	33
accuracy			0.75	157
macro avg	0.72	0.71	0.71	157
weighted avg	0.74	0.75	0.74	157

Confusion Matrix :



```

model_name = 'kneeMRI_Model5'
kneeMRI_Model5 = models.mri_model_5(model_name, len(kneemri_classes))
kneeMRI_Model5.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
kneeMRI_Model5.summary()

```

Model: "kneeMRI_Model5"

Layer (type)	Output Shape	Param #
conv3d_30 (Conv3D)	(None, 30, 256, 256, 16)	448
max_pooling3d_27 (MaxPooling3D)	(None, 15, 128, 128, 16)	0
conv3d_31 (Conv3D)	(None, 15, 128, 128, 16)	6928
max_pooling3d_28 (MaxPooling3D)	(None, 8, 64, 64, 16)	0
conv3d_32 (Conv3D)	(None, 8, 64, 64, 32)	13856
max_pooling3d_29 (MaxPooling3D)	(None, 4, 32, 32, 32)	0
conv3d_33 (Conv3D)	(None, 4, 32, 32, 32)	27680
max_pooling3d_30 (MaxPooling3D)	(None, 2, 16, 16, 32)	0

<i>conv3d_34</i> (<i>Conv3D</i>)	(None, 2, 16, 16, 32)	27680
<i>max_pooling3d_31</i> (<i>MaxPooling3D</i>)	(None, 1, 8, 8, 32)	0
<i>conv3d_35</i> (<i>Conv3D</i>)	(None, 1, 8, 8, 64)	55360
<i>max_pooling3d_32</i> (<i>MaxPooling3D</i>)	(None, 1, 4, 4, 64)	0
<i>conv3d_36</i> (<i>Conv3D</i>)	(None, 1, 4, 4, 64)	110656
<i>max_pooling3d_33</i> (<i>MaxPooling3D</i>)	(None, 1, 2, 2, 64)	0
<i>conv3d_37</i> (<i>Conv3D</i>)	(None, 1, 2, 2, 128)	221312
<i>max_pooling3d_34</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 128)	0
<i>conv3d_38</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 128)	442496
<i>max_pooling3d_35</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 128)	0
<i>conv3d_39</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	884992
<i>max_pooling3d_36</i> (<i>MaxPooling3D</i>)	(None, 1, 1, 1, 256)	0
<i>dropout_10</i> (<i>Dropout</i>)	(None, 1, 1, 1, 256)	0
<i>flatten_1</i> (<i>Flatten</i>)	(None, 256)	0
<i>dense_13</i> (<i>Dense</i>)	(None, 512)	131584
<i>dropout_11</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_14</i> (<i>Dense</i>)	(None, 256)	131328
<i>dropout_12</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_15</i> (<i>Dense</i>)	(None, 128)	32896
<i>dropout_13</i> (<i>Dropout</i>)	(None, 128)	0
<i>dense_16</i> (<i>Dense</i>)	(None, 3)	387

=====
Total params: 2,087,603
Trainable params: 2,087,603

Non-trainable params: 0

```
%%time
with tf.device('/device:GPU:0'):
    history = kneeMRI_Model5.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EPOCHS,
                                validation_data=utils.batch_generator(X_valid, y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=kneemri_class_weights,
                                verbose=1,
                                callbacks=[utils.model_callback_checkpoint(model_name),
                               utils.model_callback_earlystopping()])
```

Epoch 1/100
123/123 [=====] - 51s 403ms/step - loss: 1.0995 -
accuracy: 0.4004 - val_loss: 1.0983 - val_accuracy: 0.4599
Epoch 2/100
123/123 [=====] - 47s 385ms/step - loss: 1.0984 -
accuracy: 0.3953 - val_loss: 1.0985 - val_accuracy: 0.4599
Epoch 3/100
123/123 [=====] - 47s 382ms/step - loss: 1.0990 -
accuracy: 0.3648 - val_loss: 1.0987 - val_accuracy: 0.3373
Epoch 4/100
123/123 [=====] - 47s 384ms/step - loss: 1.0982 -
accuracy: 0.4116 - val_loss: 1.0982 - val_accuracy: 0.4646
Epoch 5/100
123/123 [=====] - 47s 380ms/step - loss: 1.0985 -
accuracy: 0.3608 - val_loss: 1.0986 - val_accuracy: 0.3278
Epoch 6/100
123/123 [=====] - 47s 381ms/step - loss: 1.0991 -
accuracy: 0.3608 - val_loss: 1.0818 - val_accuracy: 0.4458
Epoch 7/100
123/123 [=====] - 47s 381ms/step - loss: 1.1001 -
accuracy: 0.3374 - val_loss: 1.0982 - val_accuracy: 0.4693
Epoch 8/100
123/123 [=====] - 47s 383ms/step - loss: 1.0989 -
accuracy: 0.3953 - val_loss: 1.0972 - val_accuracy: 0.4458
Epoch 9/100
123/123 [=====] - 47s 383ms/step - loss: 1.1053 -
accuracy: 0.4289 - val_loss: 1.1008 - val_accuracy: 0.2406
Epoch 10/100
123/123 [=====] - 47s 383ms/step - loss: 1.0882 -
accuracy: 0.2764 - val_loss: 1.0421 - val_accuracy: 0.5330
Epoch 11/100
123/123 [=====] - 47s 383ms/step - loss: 1.0310 -
accuracy: 0.5224 - val_loss: 1.0214 - val_accuracy: 0.5212

Epoch 12/100
123/123 [=====] - 47s 383ms/step - loss: 1.0106 -
accuracy: 0.5315 - val_loss: 1.0011 - val_accuracy: 0.5330
Epoch 13/100
123/123 [=====] - 47s 383ms/step - loss: 1.0088 -
accuracy: 0.5386 - val_loss: 0.9630 - val_accuracy: 0.5401
Epoch 14/100
123/123 [=====] - 47s 381ms/step - loss: 0.9783 -
accuracy: 0.5518 - val_loss: 1.0275 - val_accuracy: 0.5354
Epoch 15/100
123/123 [=====] - 47s 383ms/step - loss: 0.9575 -
accuracy: 0.5417 - val_loss: 1.0522 - val_accuracy: 0.5071
Epoch 16/100
123/123 [=====] - 47s 383ms/step - loss: 0.9331 -
accuracy: 0.5488 - val_loss: 0.8816 - val_accuracy: 0.5778
Epoch 17/100
123/123 [=====] - 47s 381ms/step - loss: 0.9080 -
accuracy: 0.5661 - val_loss: 0.9060 - val_accuracy: 0.5330
Epoch 18/100
123/123 [=====] - 47s 380ms/step - loss: 0.8873 -
accuracy: 0.5711 - val_loss: 0.9161 - val_accuracy: 0.5236
Epoch 19/100
123/123 [=====] - 47s 380ms/step - loss: 0.8627 -
accuracy: 0.5854 - val_loss: 0.8252 - val_accuracy: 0.6085
Epoch 20/100
123/123 [=====] - 47s 381ms/step - loss: 0.8468 -
accuracy: 0.5904 - val_loss: 1.1067 - val_accuracy: 0.4505
Epoch 21/100
123/123 [=====] - 47s 380ms/step - loss: 0.8209 -
accuracy: 0.5935 - val_loss: 0.8838 - val_accuracy: 0.5637
Epoch 22/100
123/123 [=====] - 47s 382ms/step - loss: 0.7980 -
accuracy: 0.6108 - val_loss: 0.8434 - val_accuracy: 0.5920
Epoch 23/100
123/123 [=====] - 47s 382ms/step - loss: 0.7401 -
accuracy: 0.6504 - val_loss: 0.7992 - val_accuracy: 0.6108
Epoch 24/100
123/123 [=====] - 47s 382ms/step - loss: 0.7309 -
accuracy: 0.6758 - val_loss: 0.7807 - val_accuracy: 0.6297
Epoch 25/100
123/123 [=====] - 47s 382ms/step - loss: 0.7099 -
accuracy: 0.6748 - val_loss: 0.8236 - val_accuracy: 0.5967
Epoch 26/100
123/123 [=====] - 46s 378ms/step - loss: 0.6441 -
accuracy: 0.7165 - val_loss: 0.8255 - val_accuracy: 0.6486
Epoch 27/100
123/123 [=====] - 47s 380ms/step - loss: 0.6053 -
accuracy: 0.7388 - val_loss: 0.7546 - val_accuracy: 0.6863
Epoch 28/100
123/123 [=====] - 47s 381ms/step - loss: 0.5582 -
accuracy: 0.7673 - val_loss: 0.8217 - val_accuracy: 0.6651
Epoch 29/100

```

123/123 [=====] - 48s 387ms/step - loss: 0.6050 - accuracy: 0.7398 - val_loss: 0.7668 - val_accuracy: 0.6580
Epoch 30/100
123/123 [=====] - 48s 387ms/step - loss: 0.4885 - accuracy: 0.8069 - val_loss: 0.8230 - val_accuracy: 0.6840
Epoch 31/100
123/123 [=====] - 47s 384ms/step - loss: 0.4050 - accuracy: 0.8404 - val_loss: 1.1375 - val_accuracy: 0.6533
Epoch 32/100
123/123 [=====] - 47s 385ms/step - loss: 0.3590 - accuracy: 0.8506 - val_loss: 0.9462 - val_accuracy: 0.6439
Epoch 33/100
123/123 [=====] - 47s 384ms/step - loss: 0.3955 - accuracy: 0.8476 - val_loss: 0.8789 - val_accuracy: 0.6816
Epoch 34/100
123/123 [=====] - 47s 385ms/step - loss: 0.2810 - accuracy: 0.9004 - val_loss: 1.1882 - val_accuracy: 0.6557
Epoch 35/100
123/123 [=====] - 48s 392ms/step - loss: 0.3204 - accuracy: 0.8933 - val_loss: 1.0743 - val_accuracy: 0.6816
Epoch 36/100
123/123 [=====] - 47s 385ms/step - loss: 0.2347 - accuracy: 0.9075 - val_loss: 1.1559 - val_accuracy: 0.6108
Epoch 37/100
123/123 [=====] - ETA: 0s - loss: 0.2415 - accuracy: 0.9055Restoring model weights from the end of the best epoch: 27.
123/123 [=====] - 47s 384ms/step - loss: 0.2415 - accuracy: 0.9055 - val_loss: 1.2556 - val_accuracy: 0.6462
Epoch 37: early stopping
Wall time: 29min 6s

```

```

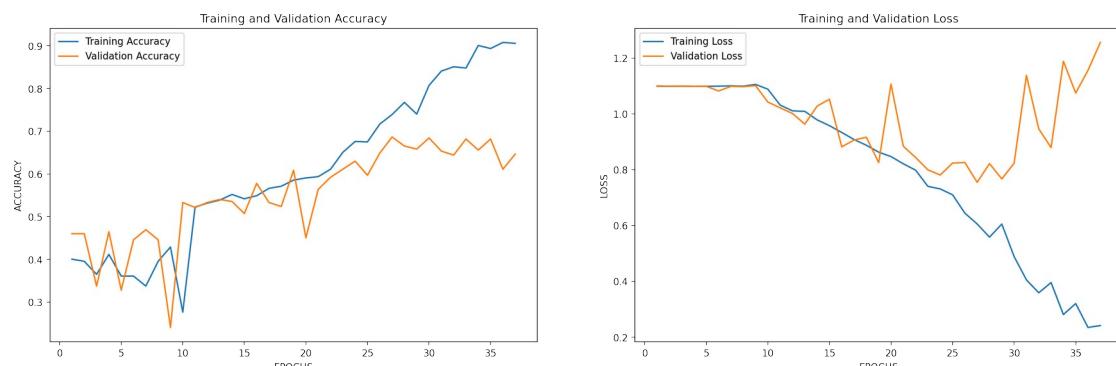
# Store history
utils.store_model_history(model_name, history.history)

```

```

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob = kneeMRI_Model5.predict(utils.predict_batch_generator(X_test, BATCH_SIZE))

```

```

X_test_pred = X_test_prob.argmax(axis=-1)

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])

20/20 [=====] - 5s 269ms/step

```

Evaluation Metrics:

Balanced Accuracy : 0.69

Precision : 0.73

Recall : 0.73

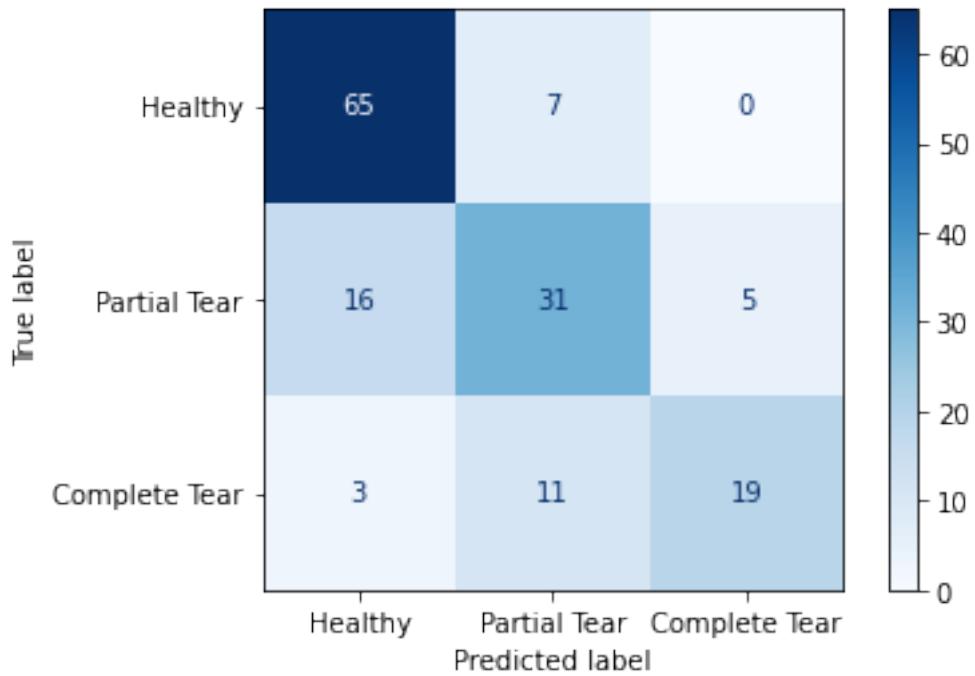
F1 Score: 0.73

ROC AUC Score : 0.86

Classification report :

	precision	recall	f1-score	support
Healthy	0.77	0.90	0.83	72
Partial Tear	0.63	0.60	0.61	52
Complete Tear	0.79	0.58	0.67	33
accuracy			0.73	157
macro avg	0.73	0.69	0.70	157
weighted avg	0.73	0.73	0.73	157

Confusion Matrix :



```

model_name = 'kneeMRI_Model6'
kneeMRI_Model6 = models.mri_model_6(model_name, len(kneemri_classes))
kneeMRI_Model6.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
kneeMRI_Model6.summary()

Model: "kneeMRI_Model6"

```

<i>Layer (type)</i>	<i>Output Shape</i>	<i>Param #</i>
<i>conv3d (Conv3D)</i>	<i>(None, 30, 256, 256, 8)</i>	224
<i>conv3d_1 (Conv3D)</i>	<i>(None, 30, 256, 256, 16)</i>	3472
<i>max_pooling3d (MaxPooling3D)</i>	<i>(None, 15, 128, 128, 16)</i>	0
<i>conv3d_2 (Conv3D)</i>	<i>(None, 15, 128, 128, 16)</i>	6928
<i>conv3d_3 (Conv3D)</i>	<i>(None, 15, 128, 128, 16)</i>	6928
<i>max_pooling3d_1 (MaxPooling3D)</i>	<i>(None, 8, 64, 64, 16)</i>	0
<i>conv3d_4 (Conv3D)</i>	<i>(None, 8, 64, 64, 16)</i>	6928
<i>conv3d_5 (Conv3D)</i>	<i>(None, 8, 64, 64, 16)</i>	6928
<i>max_pooling3d_2 (MaxPooling3D)</i>	<i>(None, 4, 32, 32, 16)</i>	0
<i>batch_normalization (BatchNormalization)</i>	<i>(None, 4, 32, 32, 16)</i>	64
<i>conv3d_6 (Conv3D)</i>	<i>(None, 4, 32, 32, 32)</i>	13856
<i>conv3d_7 (Conv3D)</i>	<i>(None, 4, 32, 32, 32)</i>	27680
<i>max_pooling3d_3 (MaxPooling3D)</i>	<i>(None, 2, 16, 16, 32)</i>	0
<i>conv3d_8 (Conv3D)</i>	<i>(None, 2, 16, 16, 32)</i>	27680
<i>max_pooling3d_4 (MaxPooling3D)</i>	<i>(None, 1, 8, 8, 32)</i>	0
<i>batch_normalization_1 (BatchNormalization)</i>	<i>(None, 1, 8, 8, 32)</i>	128
<i>conv3d_9 (Conv3D)</i>	<i>(None, 1, 8, 8, 64)</i>	55360

<i>max_pooling3d_5</i> (<i>MaxPooling 3D</i>)	(None, 1, 4, 4, 64)	0
<i>conv3d_10</i> (<i>Conv3D</i>)	(None, 1, 4, 4, 64)	110656
<i>max_pooling3d_6</i> (<i>MaxPooling 3D</i>)	(None, 1, 2, 2, 64)	0
<i>conv3d_11</i> (<i>Conv3D</i>)	(None, 1, 2, 2, 64)	110656
<i>max_pooling3d_7</i> (<i>MaxPooling 3D</i>)	(None, 1, 1, 1, 64)	0
<i>batch_normalization_2</i> (<i>BatchNormalization</i>)	(None, 1, 1, 1, 64)	256
<i>conv3d_12</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 128)	221312
<i>max_pooling3d_8</i> (<i>MaxPooling 3D</i>)	(None, 1, 1, 1, 128)	0
<i>conv3d_13</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 128)	442496
<i>max_pooling3d_9</i> (<i>MaxPooling 3D</i>)	(None, 1, 1, 1, 128)	0
<i>conv3d_14</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 128)	442496
<i>max_pooling3d_10</i> (<i>MaxPooling 3D</i>)	(None, 1, 1, 1, 128)	0
<i>batch_normalization_3</i> (<i>BatchNormalization</i>)	(None, 1, 1, 1, 128)	512
<i>conv3d_15</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	884992
<i>max_pooling3d_11</i> (<i>MaxPooling 3D</i>)	(None, 1, 1, 1, 256)	0
<i>conv3d_16</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	1769728
<i>max_pooling3d_12</i> (<i>MaxPooling 3D</i>)	(None, 1, 1, 1, 256)	0
<i>conv3d_17</i> (<i>Conv3D</i>)	(None, 1, 1, 1, 256)	1769728
<i>max_pooling3d_13</i> (<i>MaxPooling 3D</i>)	(None, 1, 1, 1, 256)	0
<i>dropout</i> (<i>Dropout</i>)	(None, 1, 1, 1, 256)	0

<i>flatten</i> (<i>Flatten</i>)	(None, 256)	0
<i>dense</i> (<i>Dense</i>)	(None, 512)	131584
<i>dropout_1</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_1</i> (<i>Dense</i>)	(None, 256)	131328
<i>dropout_2</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_2</i> (<i>Dense</i>)	(None, 128)	32896
<i>dropout_3</i> (<i>Dropout</i>)	(None, 128)	0
<i>dense_3</i> (<i>Dense</i>)	(None, 3)	387

Total params: 6,205,203
 Trainable params: 6,204,723
 Non-trainable params: 480

```
%%time
with tf.device('/device:GPU:0'):
    history = kneeMRI_Model6.fit(utils.batch_generator(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EPOCHS,
                                validation_data=utils.batch_generator(X_valid, y_valid, BATCH_SIZE),
                                validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=kneemri_class_weights,
                                verbose=1,
                                callbacks=[utils.model_callback_checkpoint(model_name),
                               utils.model_callback_earlystopping()])

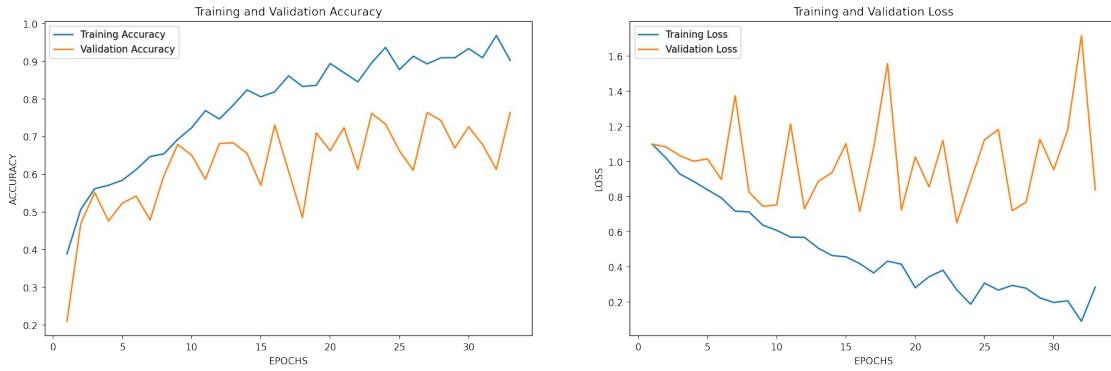
Epoch 1/100
123/123 [=====] - 80s 566ms/step - loss: 1.0993 - accuracy: 0.3892 - val_loss: 1.0990 - val_accuracy: 0.2099
Epoch 2/100
123/123 [=====] - 64s 523ms/step - loss: 1.0206 - accuracy: 0.5071 - val_loss: 1.0834 - val_accuracy: 0.4693
Epoch 3/100
123/123 [=====] - 65s 526ms/step - loss: 0.9292 - accuracy: 0.5620 - val_loss: 1.0338 - val_accuracy: 0.5519
Epoch 4/100
123/123 [=====] - 65s 527ms/step - loss: 0.8873 - accuracy: 0.5711 - val_loss: 1.0012 - val_accuracy: 0.4764
Epoch 5/100
123/123 [=====] - 65s 526ms/step - loss: 0.8395 -
```

```
accuracy: 0.5843 - val_loss: 1.0153 - val_accuracy: 0.5236
Epoch 6/100
123/123 [=====] - 65s 529ms/step - loss: 0.7936 -
accuracy: 0.6128 - val_loss: 0.8983 - val_accuracy: 0.5425
Epoch 7/100
123/123 [=====] - 65s 526ms/step - loss: 0.7179 -
accuracy: 0.6474 - val_loss: 1.3744 - val_accuracy: 0.4788
Epoch 8/100
123/123 [=====] - 65s 528ms/step - loss: 0.7138 -
accuracy: 0.6545 - val_loss: 0.8267 - val_accuracy: 0.5967
Epoch 9/100
123/123 [=====] - 65s 529ms/step - loss: 0.6380 -
accuracy: 0.6931 - val_loss: 0.7458 - val_accuracy: 0.6792
Epoch 10/100
123/123 [=====] - 65s 527ms/step - loss: 0.6085 -
accuracy: 0.7236 - val_loss: 0.7533 - val_accuracy: 0.6509
Epoch 11/100
123/123 [=====] - 65s 527ms/step - loss: 0.5701 -
accuracy: 0.7693 - val_loss: 1.2129 - val_accuracy: 0.5873
Epoch 12/100
123/123 [=====] - 65s 530ms/step - loss: 0.5690 -
accuracy: 0.7470 - val_loss: 0.7308 - val_accuracy: 0.6816
Epoch 13/100
123/123 [=====] - 65s 528ms/step - loss: 0.5071 -
accuracy: 0.7835 - val_loss: 0.8869 - val_accuracy: 0.6840
Epoch 14/100
123/123 [=====] - 65s 528ms/step - loss: 0.4651 -
accuracy: 0.8242 - val_loss: 0.9378 - val_accuracy: 0.6557
Epoch 15/100
123/123 [=====] - 65s 527ms/step - loss: 0.4580 -
accuracy: 0.8059 - val_loss: 1.1020 - val_accuracy: 0.5708
Epoch 16/100
123/123 [=====] - 65s 529ms/step - loss: 0.4192 -
accuracy: 0.8191 - val_loss: 0.7168 - val_accuracy: 0.7311
Epoch 17/100
123/123 [=====] - 65s 527ms/step - loss: 0.3664 -
accuracy: 0.8618 - val_loss: 1.0784 - val_accuracy: 0.6085
Epoch 18/100
123/123 [=====] - 65s 527ms/step - loss: 0.4334 -
accuracy: 0.8333 - val_loss: 1.5576 - val_accuracy: 0.4858
Epoch 19/100
123/123 [=====] - 65s 528ms/step - loss: 0.4163 -
accuracy: 0.8364 - val_loss: 0.7241 - val_accuracy: 0.7099
Epoch 20/100
123/123 [=====] - 65s 528ms/step - loss: 0.2820 -
accuracy: 0.8943 - val_loss: 1.0258 - val_accuracy: 0.6627
Epoch 21/100
123/123 [=====] - 65s 528ms/step - loss: 0.3450 -
accuracy: 0.8699 - val_loss: 0.8553 - val_accuracy: 0.7241
Epoch 22/100
123/123 [=====] - 65s 527ms/step - loss: 0.3820 -
accuracy: 0.8455 - val_loss: 1.1201 - val_accuracy: 0.6132
```

```
Epoch 23/100
123/123 [=====] - 65s 530ms/step - loss: 0.2702 -
accuracy: 0.8963 - val_loss: 0.6520 - val_accuracy: 0.7618
Epoch 24/100
123/123 [=====] - 65s 528ms/step - loss: 0.1886 -
accuracy: 0.9370 - val_loss: 0.8891 - val_accuracy: 0.7335
Epoch 25/100
123/123 [=====] - 65s 528ms/step - loss: 0.3087 -
accuracy: 0.8780 - val_loss: 1.1238 - val_accuracy: 0.6627
Epoch 26/100
123/123 [=====] - 65s 527ms/step - loss: 0.2685 -
accuracy: 0.9136 - val_loss: 1.1823 - val_accuracy: 0.6108
Epoch 27/100
123/123 [=====] - 65s 528ms/step - loss: 0.2956 -
accuracy: 0.8933 - val_loss: 0.7206 - val_accuracy: 0.7642
Epoch 28/100
123/123 [=====] - 65s 528ms/step - loss: 0.2797 -
accuracy: 0.9096 - val_loss: 0.7686 - val_accuracy: 0.7429
Epoch 29/100
123/123 [=====] - 65s 528ms/step - loss: 0.2247 -
accuracy: 0.9096 - val_loss: 1.1264 - val_accuracy: 0.6698
Epoch 30/100
123/123 [=====] - 65s 528ms/step - loss: 0.1982 -
accuracy: 0.9339 - val_loss: 0.9536 - val_accuracy: 0.7264
Epoch 31/100
123/123 [=====] - 65s 528ms/step - loss: 0.2082 -
accuracy: 0.9096 - val_loss: 1.1790 - val_accuracy: 0.6792
Epoch 32/100
123/123 [=====] - 65s 528ms/step - loss: 0.0911 -
accuracy: 0.9685 - val_loss: 1.7167 - val_accuracy: 0.6132
Epoch 33/100
123/123 [=====] - ETA: 0s - loss: 0.2867 - accuracy:
0.9024Restoring model weights from the end of the best epoch: 23.
123/123 [=====] - 65s 527ms/step - loss: 0.2867 -
accuracy: 0.9024 - val_loss: 0.8386 - val_accuracy: 0.7642
Epoch 33: early stopping
Wall time: 35min 53s
```

```
# Store history
utils.store_model_history(model_name, history.history)
```

```
# Plot training graphs
utils.plot_acc_loss(history.history)
```



```
kneeMRI_Model6 = utils.load_model_from_disk('kneeMRI_Model6')

# Evaluate model
X_test_prob = kneeMRI_Model6.predict(utils.predict_batch_generator(X_test,
BATCH_SIZE))

X_test_pred = X_test_prob.argmax(axis=-1)

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])

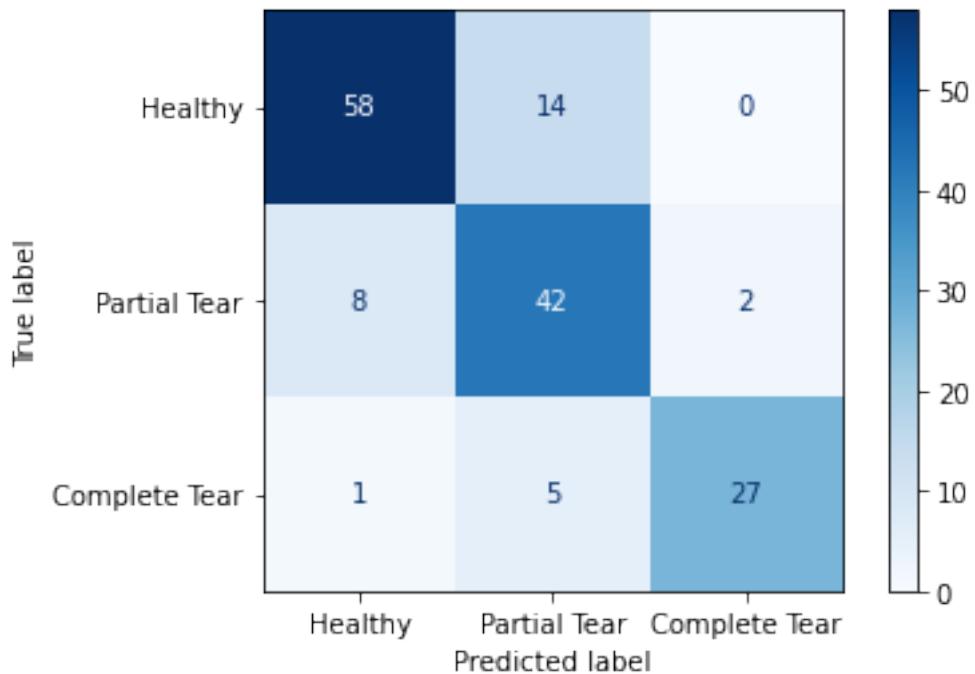
20/20 [=====] - 10s 525ms/step
```

Evaluation Metrics:

Balanced Accuracy : 0.81
Precision : 0.82
Recall : 0.81
F1 Score: 0.81
ROC AUC Score : 0.92

	precision	recall	f1-score	support
Healthy	0.87	0.81	0.83	72
Partial Tear	0.69	0.81	0.74	52
Complete Tear	0.93	0.82	0.87	33
accuracy			0.81	157
macro avg	0.83	0.81	0.82	157
weighted avg	0.82	0.81	0.81	157

Confusion Matrix :



```

from tensorflow import keras
from tensorflow.keras.models import Sequential
# from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Conv3D, MaxPooling3D, Flatten, Dense

model = Sequential([
    Conv3D(32, kernel_size=(3, 3, 3), activation='relu', input_shape=(30, 256,
256, 1)),
    MaxPooling3D(pool_size=(2, 2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(3, activation='softmax') # Three classes: 0, 1, 2
], name='Trial')

model.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

```

Model: "Trial"

Layer (type)	Output Shape	Param #
conv3d_18 (Conv3D)	(None, 28, 254, 254, 32)	896
max_pooling3d_14 (MaxPooling3D)	(None, 14, 127, 127, 32)	0
flatten_1 (Flatten)	(None, 7225792)	0

```
dense_4 (Dense)           (None, 64)          462450752
```

```
dense_5 (Dense)           (None, 3)           195
```

```
Total params: 462,451,843
Trainable params: 462,451,843
Non-trainable params: 0
```

```
X_train_temp = X_train[:128]
y_train_temp = y_train[:128]
X_valid_temp = X_test[:32]
y_valid_temp = y_test[:32]
class_weights_temp = utils.compute_class_weights(y_train_temp)

%%time
with tf.device('/device:GPU:0'):
    history = model.fit(utils.batch_generator(X_train_temp, y_train_temp,
BATCH_SIZE),
                      steps_per_epoch=len(X_train_temp)//BATCH_SIZE,
                      epochs=5,
                      validation_data=utils.batch_generator(X_valid_temp,
y_valid_temp, BATCH_SIZE),
                      validation_steps=len(X_valid_temp)//BATCH_SIZE,
                      shuffle=True,
                      class_weight=class_weights_temp,
                      verbose=1,
callbacks=[utils.model_callback_checkpoint('CPUvsGPU'),
utils.model_callback_earlystopping()])

Epoch 1/5
16/16 [=====] - 21s 1s/step - loss: 51.4816 -
accuracy: 0.4453 - val_loss: 20.8152 - val_accuracy: 0.5625
Epoch 2/5
16/16 [=====] - 18s 1s/step - loss: 16.5783 -
accuracy: 0.6250 - val_loss: 9.4362 - val_accuracy: 0.6562
Epoch 3/5
16/16 [=====] - 18s 1s/step - loss: 7.3881 -
accuracy: 0.7031 - val_loss: 3.6138 - val_accuracy: 0.7188
Epoch 4/5
16/16 [=====] - 6s 361ms/step - loss: 2.4683 -
accuracy: 0.8047 - val_loss: 4.9877 - val_accuracy: 0.6250
Epoch 5/5
16/16 [=====] - 6s 358ms/step - loss: 0.5582 -
accuracy: 0.9219 - val_loss: 4.6749 - val_accuracy: 0.6875
Wall time: 1min 8s

%%time
with tf.device('/device:CPU:0'):
```

```

history = model.fit(utils.batch_generator(X_train_temp, y_train_temp,
BATCH_SIZE),
                     steps_per_epoch=len(X_train_temp)//BATCH_SIZE,
                     epochs=5,
                     validation_data=utils.batch_generator(X_valid_temp,
y_valid_temp, BATCH_SIZE),
                     validation_steps=len(X_valid_temp)//BATCH_SIZE,
                     shuffle=True,
                     class_weight=class_weights_temp,
                     verbose=1,
callbacks=[utils.model_callback_checkpoint('CPUvsGPU'),
utils.model_callback_earlystopping()])

```

Epoch 1/5
16/16 [=====] - 386s 24s/step - loss: 1.0875 -
accuracy: 0.9297 - val_loss: 6.0008 - val_accuracy: 0.7188
Epoch 2/5
16/16 [=====] - 369s 23s/step - loss: 0.3787 -
accuracy: 0.9375 - val_loss: 7.2662 - val_accuracy: 0.5625
Epoch 3/5
16/16 [=====] - 381s 24s/step - loss: 0.1403 -
accuracy: 0.9766 - val_loss: 4.1612 - val_accuracy: 0.7812
Epoch 4/5
16/16 [=====] - 370s 23s/step - loss: 0.4838 -
accuracy: 0.9219 - val_loss: 9.6759 - val_accuracy: 0.5938
Epoch 5/5
16/16 [=====] - 381s 24s/step - loss: 0.4018 -
accuracy: 0.9453 - val_loss: 2.7049 - val_accuracy: 0.7812
Wall time: 31min 28s

I) MRNet Transfer Learning

```

import os
import platform
import pandas as pd
import numpy as np
from glob import glob

from tensorflow import keras
from sklearn.model_selection import train_test_split
import utils
import models

```

Tensorflow version : 2.10.1

Tensorflow devices available :
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456

```

locality {
}
incarnation: 2699538139950674383
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 22672310272
locality {
  bus_id: 1
  links {
  }
}
incarnation: 1285623852169483370
physical_device_desc: "device: 0, name: NVIDIA RTX A5000, pci bus id: 0000:61:00.0, compute capability: 8.6"
xla_global_id: 416903419
]

Tensorflow physical devices available :
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]

mrnet_dataset_dir = 'Data/MRNet-v1.0'
mrnet_train_path = os.path.join(mrnet_dataset_dir, 'train')
mrnet_valid_path = os.path.join(mrnet_dataset_dir, 'valid')

mrnet_preprocessed_dataset_dir = 'Preprocessed_Data/MRNet-v1.0'
mrnet_preprocessed_train_path = os.path.join(mrnet_preprocessed_dataset_dir,
'train')
mrnet_preprocessed_valid_path = os.path.join(mrnet_preprocessed_dataset_dir,
'valid')

mrnet_planes = ['axial', 'coronal', 'sagittal']

# For running code on Windows
if platform.system() == "Windows":
    mrnet_dataset_dir = mrnet_dataset_dir.replace('/', '\\')
    mrnet_train_path = mrnet_train_path.replace('/', '\\')
    mrnet_valid_path = mrnet_valid_path.replace('/', '\\')

    mrnet_preprocessed_dataset_dir =
    mrnet_preprocessed_dataset_dir.replace('/', '\\')
    mrnet_preprocessed_train_path = mrnet_preprocessed_train_path.replace('/', '\\')
    mrnet_preprocessed_valid_path = mrnet_preprocessed_valid_path.replace('/', '\\')

mrnet_datasets = {'train': mrnet_train_path, 'valid': mrnet_valid_path}
mrnet_classes = ['abnormal', 'acl', 'meniscus']

# TRAIN DATASET
for label in mrnet_classes:

```

```

if platform.system() == "Windows":
    if label == 'abnormal':
        train_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}\\"train-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Abnormal'],
                                         dtype={'Case':str,
                                         'Abnormal':np.int64})
    elif label == 'acl':
        train_acl_df = pd.read_csv(f"{mrnet_dataset_dir}\\"train-
{label}.csv",
                                         header=None,
                                         names=['Case', 'ACL'],
                                         dtype={'Case':str,
                                         'ACL':np.int64})
    if label == 'meniscus':
        train_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}\\"train-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Meniscus'],
                                         dtype={'Case':str,
                                         'Meniscus':np.int64})
    else:
        if label == 'abnormal':
            train_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Abnormal'],
                                         dtype={'Case':str,
                                         'Abnormal':np.int64})
        elif label == 'acl':
            train_acl_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                         header=None,
                                         names=['Case', 'ACL'],
                                         dtype={'Case':str,
                                         'ACL':np.int64})
        if label == 'meniscus':
            train_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}/train-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Meniscus'],
                                         dtype={'Case':str,
                                         'Meniscus':np.int64})
mrnet_train_df = pd.merge(train_abnormal_df, train_acl_df,
on='Case').merge(train_meniscus_df, on='Case')

# VALID DATASET
for label in mrnet_classes:
    if platform.system() == "Windows":
        if label == 'abnormal':

```

```

        valid_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}\\"valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Abnormal'],
                                         dtype={'Case':str,
                                                'Abnormal':np.int64})
    elif label == 'acl':
        valid_acl_df = pd.read_csv(f"{mrnet_dataset_dir}\\"valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'ACL'],
                                         dtype={'Case':str,
                                                'ACL':np.int64})
    if label == 'meniscus':
        valid_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}\\"valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Meniscus'],
                                         dtype={'Case':str,
                                                'Meniscus':np.int64})
    else:
        if label == 'abnormal':
            valid_abnormal_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Abnormal'],
                                         dtype={'Case':str,
                                                'Abnormal':np.int64})
        elif label == 'acl':
            valid_acl_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'ACL'],
                                         dtype={'Case':str,
                                                'ACL':np.int64})
        if label == 'meniscus':
            valid_meniscus_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-
{label}.csv",
                                         header=None,
                                         names=['Case', 'Meniscus'],
                                         dtype={'Case':str,
                                                'Meniscus':np.int64})
    'Meniscus':np.int64})

mrnet_valid_df = pd.merge(valid_abnormal_df, valid_acl_df,
on='Case').merge(valid_meniscus_df, on='Case')

# AUGMENTED TRAIN LABELS
if platform.system() == "Windows":
    mrnet_train_aug_df = pd.read_csv(f"{mrnet_dataset_dir}\\"train-aug.csv",
                                         index_col=0,
                                         dtype={'Case':str, 'Abnormal':np.int64,
                                                'ACL':np.int64, 'Meniscus':np.int64})

```

```

else:
    mrnet_train_aug_df = pd.read_csv(f"{mrnet_dataset_dir}/train-aug.csv",
                                      index_col=0,
                                      dtype={'Case':str, 'Abnormal':np.int64,
'ACL':np.int64, 'Meniscus':np.int64})

# AUGMENTED VALID LABELS
if platform.system() == "Windows":
    mrnet_valid_aug_df = pd.read_csv(f"{mrnet_dataset_dir}\valid-aug.csv",
                                      index_col=0,
                                      dtype={'Case':str, 'Abnormal':np.int64,
'ACL':np.int64, 'Meniscus':np.int64})
else:
    mrnet_valid_aug_df = pd.read_csv(f"{mrnet_dataset_dir}/valid-aug.csv",
                                      index_col=0,
                                      dtype={'Case':str, 'Abnormal':np.int64,
'ACL':np.int64, 'Meniscus':np.int64})

# We are working only with Sagittal plane

# TRAIN
if platform.system() == "Windows":
    mrnet_sagittal_train_files = glob(mrnet_preprocessed_train_path+\
sagittal\*.npy")
else:
    mrnet_sagittal_train_files =
glob(mrnet_preprocessed_train_path+/sagittal/*.npy")
mrnet_sagittal_train_files.sort()

# VALID
if platform.system() == "Windows":
    mrnet_sagittal_valid_files = glob(mrnet_preprocessed_valid_path+\
sagittal\*.npy")
else:
    mrnet_sagittal_valid_files =
glob(mrnet_preprocessed_valid_path+/sagittal/*.npy")
mrnet_sagittal_valid_files.sort()

# AUGMENTED TRAIN
if platform.system() == "Windows":
    mrnet_sagittal_train_aug_files = glob(mrnet_preprocessed_train_path+\\
sagittal\aug\*.npy")
else:
    mrnet_sagittal_train_aug_files =
glob(mrnet_preprocessed_train_path+/sagittal/aug/*.npy")
mrnet_sagittal_train_aug_files.sort()

# AUGMENTED VALID
if platform.system() == "Windows":
    mrnet_sagittal_valid_aug_files = glob(mrnet_preprocessed_valid_path+\\
sagittal\aug\*.npy")
else:

```

```

mrnet_sagittal_valid_aug_files =
glob(mrnet_preprocessed_valid_path+"/sagittal/aug/*.npy")
mrnet_sagittal_valid_aug_files.sort()

mrnet_filenames = []
mrnet_filenames.extend(mrnet_sagittal_train_files)
mrnet_filenames.extend(mrnet_sagittal_valid_files)
mrnet_filenames.extend(mrnet_sagittal_train_aug_files)
mrnet_filenames.extend(mrnet_sagittal_valid_aug_files)
mrnet_filenames.sort()

mrnet_full_df = pd.concat([mrnet_train_df, mrnet_valid_df, mrnet_train_aug_df,
mrnet_valid_aug_df], ignore_index=True)

mrnet_labels = utils.get_correct_labels_mrnet(mrnet_filenames, mrnet_full_df)

BATCH_SIZE = 8
EPOCHS = 100

# Splitting into train, test and validation

X, X_test, y, y_test = train_test_split(mrnet_filenames,
                                         mrnet_labels,
                                         test_size=0.1,
                                         random_state=610,
                                         shuffle=True,
                                         stratify=mrnet_labels)

X_train, X_valid, y_train, y_valid = train_test_split(X,
                                                       y,
                                                       train_size=0.7,
                                                       random_state=610,
                                                       shuffle=True,
                                                       stratify=y)

mrnet_class_weights = utils.compute_class_weights(y_train)

mrnet_class_weights

{0: 1.0076923076923077, 1: 0.9924242424242424}

model_name = 'MRNet_Model_TF_3_VGG'
MRNet_Model_TF_3_VGG = models.mri_model_tf_3_vgg(model_name, 2)
MRNet_Model_TF_3_VGG.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                             loss='binary_crossentropy',
                             metrics=['accuracy'])

MRNet_Model_TF_3_VGG.summary()

Model: "MRNet_Model_TF_3_VGG"


```

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14714688

<i>flatten_3</i> (<i>Flatten</i>)	(None, 32768)	0
<i>dense_12</i> (<i>Dense</i>)	(None, 512)	16777728
<i>dropout_9</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_13</i> (<i>Dense</i>)	(None, 256)	131328
<i>dropout_10</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_14</i> (<i>Dense</i>)	(None, 128)	32896
<i>dropout_11</i> (<i>Dropout</i>)	(None, 128)	0
<i>dense_15</i> (<i>Dense</i>)	(None, 1)	129

=====

Total params: 31,656,769

Trainable params: 16,942,081

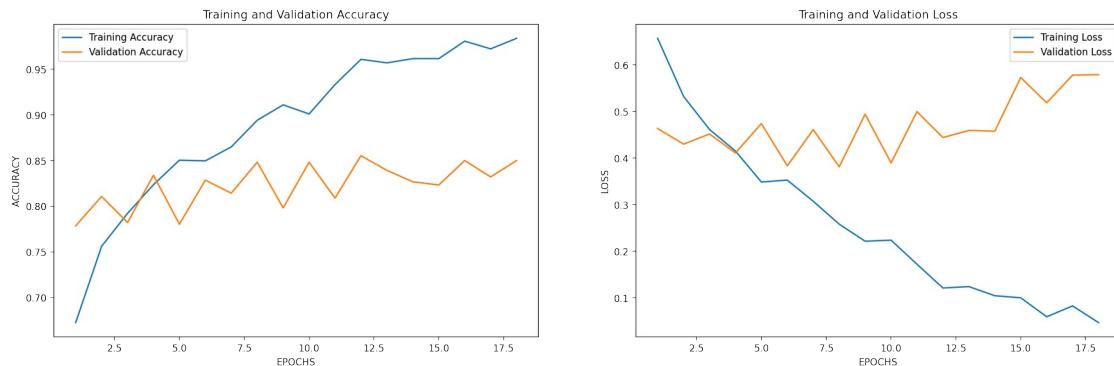
Non-trainable params: 14,714,688

```
%%time
with tf.device('/device:GPU:0'):
    history = MRNet_Model_TF_3_VGG.fit(utils.batch_generator_tf_3(X_train,
y_train, BATCH_SIZE),
steps_per_epoch=len(X_train)//BATCH_SIZE,
                    epochs=EPOCHS,
validation_data=utils.batch_generator_tf_3(X_valid, y_valid, BATCH_SIZE),
validation_steps=len(X_valid)//BATCH_SIZE,
                    shuffle=True,
                    class_weight=mrnet_class_weights,
                    verbose=1,
callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()])
Epoch 1/100
163/163 [=====] - 54s 329ms/step - loss: 0.6572 - accuracy: 0.6725 - val_loss: 0.4635 - val_accuracy: 0.7786
Epoch 2/100
163/163 [=====] - 43s 265ms/step - loss: 0.5324 - accuracy: 0.7561 - val_loss: 0.4300 - val_accuracy: 0.8107
Epoch 3/100
163/163 [=====] - 42s 260ms/step - loss: 0.4615 - accuracy: 0.7922 - val_loss: 0.4518 - val_accuracy: 0.7821
Epoch 4/100
163/163 [=====] - 43s 263ms/step - loss: 0.4152 -
```

```
accuracy: 0.8236 - val_loss: 0.4112 - val_accuracy: 0.8339
Epoch 5/100
163/163 [=====] - 42s 261ms/step - loss: 0.3487 -
accuracy: 0.8505 - val_loss: 0.4740 - val_accuracy: 0.7804
Epoch 6/100
163/163 [=====] - 42s 261ms/step - loss: 0.3527 -
accuracy: 0.8497 - val_loss: 0.3833 - val_accuracy: 0.8286
Epoch 7/100
163/163 [=====] - 42s 259ms/step - loss: 0.3073 -
accuracy: 0.8650 - val_loss: 0.4612 - val_accuracy: 0.8143
Epoch 8/100
163/163 [=====] - 42s 260ms/step - loss: 0.2582 -
accuracy: 0.8942 - val_loss: 0.3813 - val_accuracy: 0.8482
Epoch 9/100
163/163 [=====] - 41s 255ms/step - loss: 0.2215 -
accuracy: 0.9110 - val_loss: 0.4942 - val_accuracy: 0.7982
Epoch 10/100
163/163 [=====] - 41s 254ms/step - loss: 0.2238 -
accuracy: 0.9011 - val_loss: 0.3898 - val_accuracy: 0.8482
Epoch 11/100
163/163 [=====] - 41s 252ms/step - loss: 0.1721 -
accuracy: 0.9333 - val_loss: 0.4998 - val_accuracy: 0.8089
Epoch 12/100
163/163 [=====] - 41s 253ms/step - loss: 0.1210 -
accuracy: 0.9609 - val_loss: 0.4442 - val_accuracy: 0.8554
Epoch 13/100
163/163 [=====] - 41s 250ms/step - loss: 0.1240 -
accuracy: 0.9571 - val_loss: 0.4594 - val_accuracy: 0.8393
Epoch 14/100
163/163 [=====] - 41s 251ms/step - loss: 0.1044 -
accuracy: 0.9617 - val_loss: 0.4578 - val_accuracy: 0.8268
Epoch 15/100
163/163 [=====] - 41s 251ms/step - loss: 0.1000 -
accuracy: 0.9617 - val_loss: 0.5731 - val_accuracy: 0.8232
Epoch 16/100
163/163 [=====] - 41s 250ms/step - loss: 0.0593 -
accuracy: 0.9808 - val_loss: 0.5188 - val_accuracy: 0.8500
Epoch 17/100
163/163 [=====] - 41s 251ms/step - loss: 0.0825 -
accuracy: 0.9724 - val_loss: 0.5783 - val_accuracy: 0.8321
Epoch 18/100
163/163 [=====] - ETA: 0s - loss: 0.0467 - accuracy:
0.9839Restoring model weights from the end of the best epoch: 8.
163/163 [=====] - 40s 249ms/step - loss: 0.0467 -
accuracy: 0.9839 - val_loss: 0.5791 - val_accuracy: 0.8500
Epoch 18: early stopping
Wall time: 12min 39s
```

```
# Store history
utils.store_model_history(model_name, history.history)
```

```
# Plot training graphs
utils.plot_acc_loss(history.history)
```



```
# Evaluate model
X_test_prob =
MRNet_Model_TF_3_VGG.predict(utils.predict_batch_generator_tf_3(X_test,
BATCH_SIZE))

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])

26/26 [=====] - 6s 229ms/step
Best cutoff Threshold=0.452761, F-Score=0.829
```

Evaluation Metrics:

Balanced Accuracy : 0.82

Precision : 0.79

Recall : 0.88

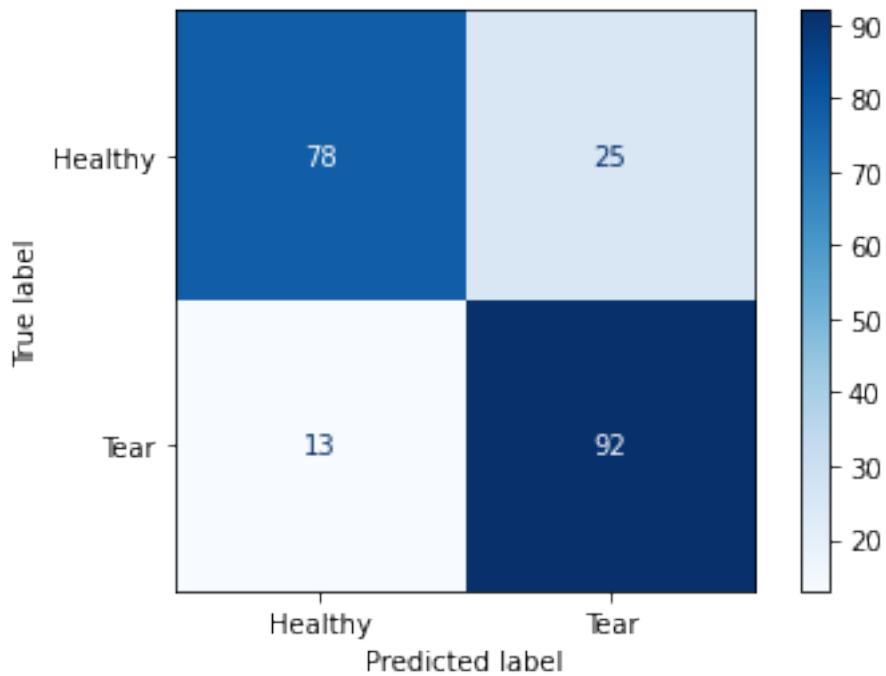
F1 Score: 0.83

ROC AUC Score : 0.89

Classification report :

	precision	recall	f1-score	support
Healthy	0.86	0.76	0.80	103
Tear	0.79	0.88	0.83	105
accuracy			0.82	208
macro avg	0.82	0.82	0.82	208
weighted avg	0.82	0.82	0.82	208

Confusion Matrix :



```

model_name = 'MRNet_Model_TF_3_Xception'
MRNet_Model_TF_3_Xception = models.mri_model_tf_3_xception(model_name, 2)
MRNet_Model_TF_3_Xception.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                                    loss='binary_crossentropy',
                                    metrics=['accuracy'])
MRNet_Model_TF_3_Xception.summary()

```

Model: "MRNet_Model_TF_3_Xception"

Layer (type)	Output Shape	Param #
xception (Functional)	(None, 8, 8, 2048)	20861480
flatten_4 (Flatten)	(None, 131072)	0
dense_16 (Dense)	(None, 512)	67109376
dropout_12 (Dropout)	(None, 512)	0
dense_17 (Dense)	(None, 256)	131328
dropout_13 (Dropout)	(None, 256)	0
dense_18 (Dense)	(None, 128)	32896
dropout_14 (Dropout)	(None, 128)	0
dense_19 (Dense)	(None, 1)	129

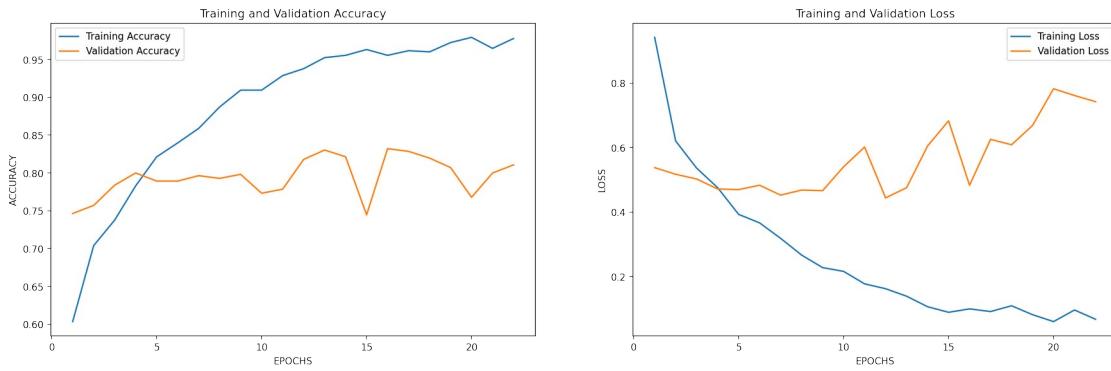
```
Total params: 88,135,209  
Trainable params: 67,273,729  
Non-trainable params: 20,861,480
```

```
%%time  
with tf.device('/device:GPU:0'):  
    history =  
    MRNet_Model_TF_3_Xception.fit(utils.batch_generator_tf_3(X_train, y_train,  
    BATCH_SIZE),  
    steps_per_epoch=len(X_train)//BATCH_SIZE,  
                                epochs=EPOCHS,  
    validation_data=utils.batch_generator_tf_3(X_valid, y_valid, BATCH_SIZE),  
    validation_steps=len(X_valid)//BATCH_SIZE,  
                                shuffle=True,  
                                class_weight=mrnet_class_weights,  
                                verbose=1,  
    callbacks=[utils.model_callback_checkpoint(model_name),  
    utils.model_callback_earlystopping()])  
  
Epoch 1/100  
163/163 [=====] - 52s 306ms/step - loss: 0.9421 -  
accuracy: 0.6035 - val_loss: 0.5380 - val_accuracy: 0.7464  
Epoch 2/100  
163/163 [=====] - 45s 275ms/step - loss: 0.6207 -  
accuracy: 0.7040 - val_loss: 0.5175 - val_accuracy: 0.7571  
Epoch 3/100  
163/163 [=====] - 45s 276ms/step - loss: 0.5364 -  
accuracy: 0.7377 - val_loss: 0.5028 - val_accuracy: 0.7839  
Epoch 4/100  
163/163 [=====] - 45s 275ms/step - loss: 0.4760 -  
accuracy: 0.7830 - val_loss: 0.4718 - val_accuracy: 0.8000  
Epoch 5/100  
163/163 [=====] - 45s 276ms/step - loss: 0.3931 -  
accuracy: 0.8213 - val_loss: 0.4699 - val_accuracy: 0.7893  
Epoch 6/100  
163/163 [=====] - 43s 263ms/step - loss: 0.3666 -  
accuracy: 0.8397 - val_loss: 0.4834 - val_accuracy: 0.7893  
Epoch 7/100  
163/163 [=====] - 45s 277ms/step - loss: 0.3188 -  
accuracy: 0.8589 - val_loss: 0.4527 - val_accuracy: 0.7964  
Epoch 8/100  
163/163 [=====] - 43s 263ms/step - loss: 0.2668 -  
accuracy: 0.8873 - val_loss: 0.4684 - val_accuracy: 0.7929  
Epoch 9/100  
163/163 [=====] - 42s 261ms/step - loss: 0.2279 -  
accuracy: 0.9095 - val_loss: 0.4665 - val_accuracy: 0.7982  
Epoch 10/100
```

```
163/163 [=====] - 42s 260ms/step - loss: 0.2162 -  
accuracy: 0.9095 - val_loss: 0.5407 - val_accuracy: 0.7732  
Epoch 11/100  
163/163 [=====] - 42s 258ms/step - loss: 0.1776 -  
accuracy: 0.9287 - val_loss: 0.6019 - val_accuracy: 0.7786  
Epoch 12/100  
163/163 [=====] - 47s 290ms/step - loss: 0.1624 -  
accuracy: 0.9379 - val_loss: 0.4442 - val_accuracy: 0.8179  
Epoch 13/100  
163/163 [=====] - 42s 260ms/step - loss: 0.1393 -  
accuracy: 0.9525 - val_loss: 0.4757 - val_accuracy: 0.8304  
Epoch 14/100  
163/163 [=====] - 41s 255ms/step - loss: 0.1066 -  
accuracy: 0.9555 - val_loss: 0.6052 - val_accuracy: 0.8214  
Epoch 15/100  
163/163 [=====] - 41s 253ms/step - loss: 0.0893 -  
accuracy: 0.9632 - val_loss: 0.6833 - val_accuracy: 0.7446  
Epoch 16/100  
163/163 [=====] - 41s 254ms/step - loss: 0.0999 -  
accuracy: 0.9555 - val_loss: 0.4834 - val_accuracy: 0.8321  
Epoch 17/100  
163/163 [=====] - 41s 252ms/step - loss: 0.0917 -  
accuracy: 0.9617 - val_loss: 0.6257 - val_accuracy: 0.8286  
Epoch 18/100  
163/163 [=====] - 41s 251ms/step - loss: 0.1096 -  
accuracy: 0.9601 - val_loss: 0.6090 - val_accuracy: 0.8196  
Epoch 19/100  
163/163 [=====] - 41s 251ms/step - loss: 0.0819 -  
accuracy: 0.9724 - val_loss: 0.6687 - val_accuracy: 0.8071  
Epoch 20/100  
163/163 [=====] - 41s 252ms/step - loss: 0.0605 -  
accuracy: 0.9793 - val_loss: 0.7827 - val_accuracy: 0.7679  
Epoch 21/100  
163/163 [=====] - 41s 251ms/step - loss: 0.0964 -  
accuracy: 0.9647 - val_loss: 0.7617 - val_accuracy: 0.8000  
Epoch 22/100  
163/163 [=====] - ETA: 0s - loss: 0.0678 - accuracy:  
0.9778Restoring model weights from the end of the best epoch: 12.  
163/163 [=====] - 41s 252ms/step - loss: 0.0678 -  
accuracy: 0.9778 - val_loss: 0.7427 - val_accuracy: 0.8107  
Epoch 22: early stopping  
Wall time: 15min 44s

# Store history
utils.store_model_history(model_name, history.history)

# Plot training graphs
utils.plot_acc_loss(history.history)
```



```
# Evaluate model
X_test_prob =
MRNet_Model_TF_3_Xception.predict(utils.predict_batch_generator_tf_3(X_test,
BATCH_SIZE))

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])

26/26 [=====] - 9s 307ms/step
Best cutoff Threshold=0.183864, F-Score=0.812
```

Evaluation Metrics:

Balanced Accuracy : 0.79

Precision : 0.74

Recall : 0.9

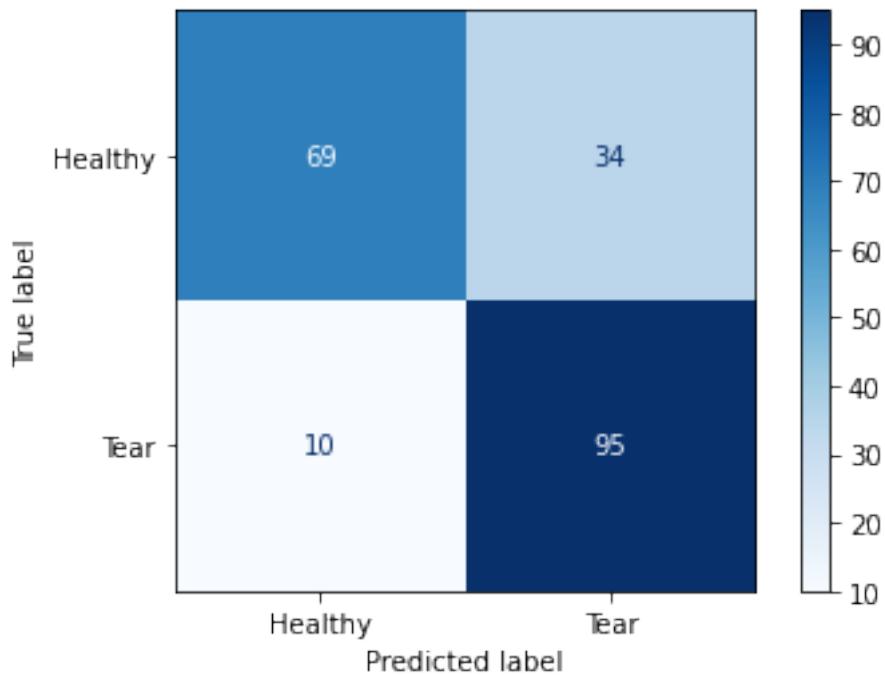
F1 Score: 0.81

ROC AUC Score : 0.85

Classification report :

	precision	recall	f1-score	support
Healthy	0.87	0.67	0.76	103
Tear	0.74	0.90	0.81	105
accuracy			0.79	208
macro avg	0.80	0.79	0.79	208
weighted avg	0.80	0.79	0.79	208

Confusion Matrix :



```

model_name = 'MRNet_Model_TF_3_ResNet'
MRNet_Model_TF_3_ResNet = models.mri_model_tf_3_vgg(model_name, 2)
MRNet_Model_TF_3_ResNet.compile(optimizer=keras.optimizers.Adam(learning_rate=
utils.model_lr_schedule()),
                                loss='binary_crossentropy',
                                metrics=['accuracy'])
MRNet_Model_TF_3_ResNet.summary()

```

Model: "MRNet_Model_TF_3_ResNet"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14714688
flatten_6 (Flatten)	(None, 32768)	0
dense_24 (Dense)	(None, 512)	16777728
dropout_18 (Dropout)	(None, 512)	0
dense_25 (Dense)	(None, 256)	131328
dropout_19 (Dropout)	(None, 256)	0
dense_26 (Dense)	(None, 128)	32896
dropout_20 (Dropout)	(None, 128)	0
dense_27 (Dense)	(None, 1)	129

```
Total params: 31,656,769  
Trainable params: 16,942,081  
Non-trainable params: 14,714,688
```

```
%%time  
with tf.device('/device:GPU:0'):  
    history = MRNet_Model_TF_3_ResNet.fit(utils.batch_generator_tf_3(X_train,  
y_train, BATCH_SIZE),  
  
    steps_per_epoch=len(X_train)//BATCH_SIZE,  
                           epochs=EPOCHS,  
  
    validation_data=utils.batch_generator_tf_3(X_valid, y_valid, BATCH_SIZE),  
  
    validation_steps=len(X_valid)//BATCH_SIZE,  
                           shuffle=True,  
                           class_weight=mrnet_class_weights,  
                           verbose=1,  
  
    callbacks=[utils.model_callback_checkpoint(model_name),  
              utils.model_callback_earlystopping()])  
  
Epoch 1/100  
163/163 [=====] - 54s 327ms/step - loss: 0.6683 -  
accuracy: 0.6756 - val_loss: 0.4754 - val_accuracy: 0.7750  
Epoch 2/100  
163/163 [=====] - 47s 291ms/step - loss: 0.5281 -  
accuracy: 0.7523 - val_loss: 0.4298 - val_accuracy: 0.7946  
Epoch 3/100  
163/163 [=====] - 43s 267ms/step - loss: 0.4479 -  
accuracy: 0.8014 - val_loss: 0.4587 - val_accuracy: 0.7929  
Epoch 4/100  
163/163 [=====] - 43s 268ms/step - loss: 0.4204 -  
accuracy: 0.8113 - val_loss: 0.4187 - val_accuracy: 0.8286  
Epoch 5/100  
163/163 [=====] - 42s 261ms/step - loss: 0.3687 -  
accuracy: 0.8298 - val_loss: 0.4219 - val_accuracy: 0.8018  
Epoch 6/100  
163/163 [=====] - 43s 262ms/step - loss: 0.3508 -  
accuracy: 0.8489 - val_loss: 0.4131 - val_accuracy: 0.8232  
Epoch 7/100  
163/163 [=====] - 42s 262ms/step - loss: 0.3240 -  
accuracy: 0.8681 - val_loss: 0.4097 - val_accuracy: 0.8232  
Epoch 8/100  
163/163 [=====] - 42s 259ms/step - loss: 0.2656 -  
accuracy: 0.8911 - val_loss: 0.3758 - val_accuracy: 0.8375  
Epoch 9/100  
163/163 [=====] - 41s 252ms/step - loss: 0.2484 -  
accuracy: 0.9034 - val_loss: 0.3912 - val_accuracy: 0.8411  
Epoch 10/100  
163/163 [=====] - 41s 250ms/step - loss: 0.2108 -
```

```

accuracy: 0.9080 - val_loss: 0.4074 - val_accuracy: 0.8304
Epoch 11/100
163/163 [=====] - 40s 245ms/step - loss: 0.1872 -
accuracy: 0.9302 - val_loss: 0.4562 - val_accuracy: 0.8161
Epoch 12/100
163/163 [=====] - 39s 242ms/step - loss: 0.1487 -
accuracy: 0.9410 - val_loss: 0.4563 - val_accuracy: 0.8411
Epoch 13/100
163/163 [=====] - 38s 237ms/step - loss: 0.1295 -
accuracy: 0.9532 - val_loss: 0.4256 - val_accuracy: 0.8554
Epoch 14/100
163/163 [=====] - 38s 234ms/step - loss: 0.1108 -
accuracy: 0.9594 - val_loss: 0.4806 - val_accuracy: 0.8482
Epoch 15/100
163/163 [=====] - 37s 231ms/step - loss: 0.0878 -
accuracy: 0.9670 - val_loss: 0.4161 - val_accuracy: 0.8482
Epoch 16/100
163/163 [=====] - 37s 227ms/step - loss: 0.0835 -
accuracy: 0.9701 - val_loss: 0.5371 - val_accuracy: 0.8446
Epoch 17/100
163/163 [=====] - 36s 225ms/step - loss: 0.0757 -
accuracy: 0.9747 - val_loss: 0.5238 - val_accuracy: 0.8643
Epoch 18/100
163/163 [=====] - ETA: 0s - loss: 0.0872 - accuracy:
0.9686Restoring model weights from the end of the best epoch: 8.
163/163 [=====] - 36s 221ms/step - loss: 0.0872 -
accuracy: 0.9686 - val_loss: 0.5035 - val_accuracy: 0.8411
Epoch 18: early stopping
Wall time: 12min 21s

```

```

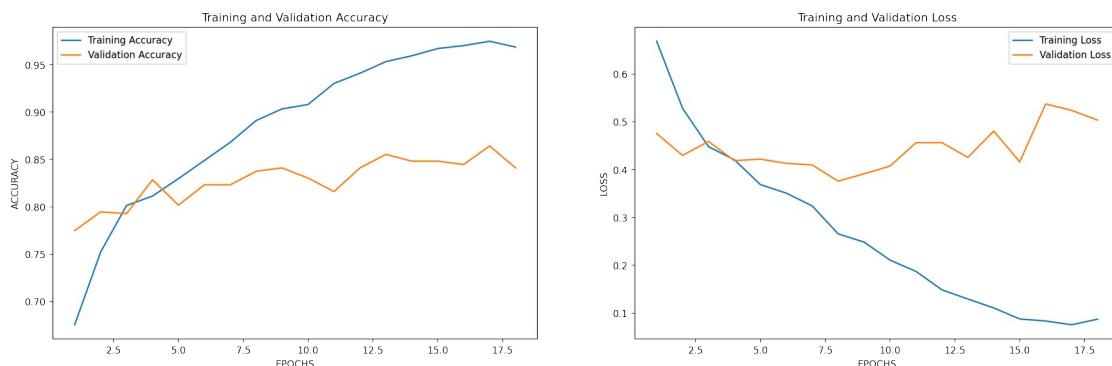
# Store history
utils.store_model_history(model_name, history.history)

```

```

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob =
MRNet_Model_TF_3_ResNet.predict(utils.predict_batch_generator_tf_3(X_test,
BATCH_SIZE))

```

```

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])

```

26/26 [=====] - 6s 223ms/step
Best cutoff Threshold=0.705600, F-Score=0.845

Evaluation Metrics:

Balanced Accuracy : 0.85

Precision : 0.86

Recall : 0.83

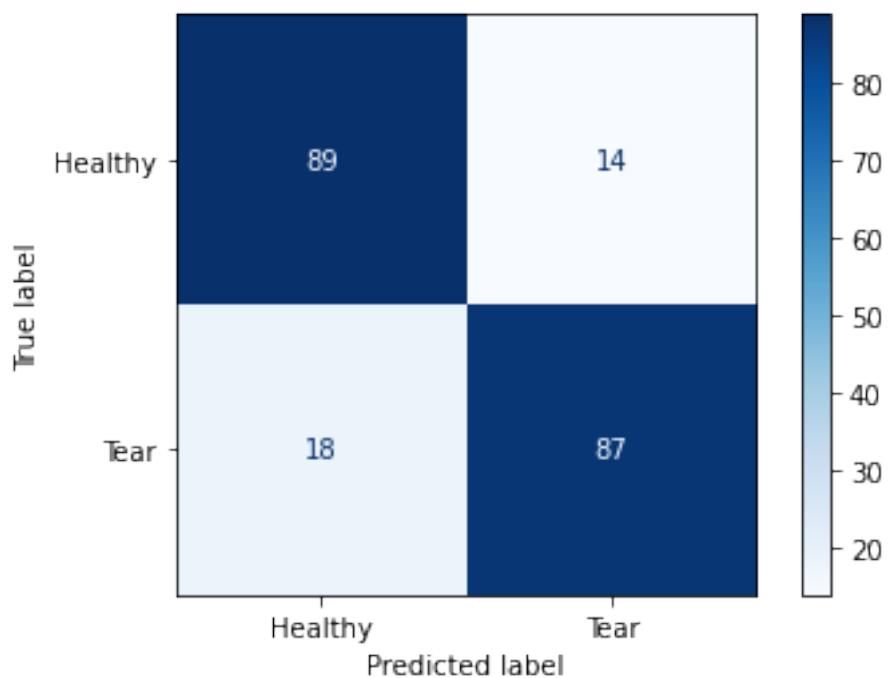
F1 Score: 0.84

ROC AUC Score : 0.9

Classification report :

	precision	recall	f1-score	support
Healthy	0.83	0.86	0.85	103
Tear	0.86	0.83	0.84	105
accuracy			0.85	208
macro avg	0.85	0.85	0.85	208
weighted avg	0.85	0.85	0.85	208

Confusion Matrix :



```

model_name = 'MRNet_Model_TF_5_VGG'
MRNet_Model_TF_5_VGG = models.mri_model_tf_5_vgg(model_name, 2)
MRNet_Model_TF_5_VGG.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                               loss='binary_crossentropy',
                               metrics=['accuracy'])
MRNet_Model_TF_5_VGG.summary()

Model: "MRNet_Model_TF_5_VGG"



---



| Layer (type)         | Output Shape      | Param #  |
|----------------------|-------------------|----------|
| vgg16 (Functional)   | (None, 8, 8, 512) | 14715840 |
| flatten_5 (Flatten)  | (None, 32768)     | 0        |
| dense_20 (Dense)     | (None, 512)       | 16777728 |
| dropout_15 (Dropout) | (None, 512)       | 0        |
| dense_21 (Dense)     | (None, 256)       | 131328   |
| dropout_16 (Dropout) | (None, 256)       | 0        |
| dense_22 (Dense)     | (None, 128)       | 32896    |
| dropout_17 (Dropout) | (None, 128)       | 0        |
| dense_23 (Dense)     | (None, 1)         | 129      |



---



Total params: 31,657,921  

Trainable params: 16,942,081  

Non-trainable params: 14,715,840


```

```

%%time
with tf.device('/device:GPU:0'):
    history = MRNet_Model_TF_5_VGG.fit(utils.batch_generator_tf_5(X_train,
y_train, BATCH_SIZE,
steps_per_epoch=len(X_train)//BATCH_SIZE,
epochs=EPOCHS,
validation_data=utils.batch_generator_tf_5(X_valid, y_valid, BATCH_SIZE),
validation_steps=len(X_valid)//BATCH_SIZE,
shuffle=True,
class_weight=mrnet_class_weights,
verbose=1,

```

```
callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping())]

Epoch 1/100
163/163 [=====] - 54s 326ms/step - loss: 0.6928 - accuracy: 0.6296 - val_loss: 0.5141 - val_accuracy: 0.7679
Epoch 2/100
163/163 [=====] - 48s 297ms/step - loss: 0.5433 - accuracy: 0.7301 - val_loss: 0.4633 - val_accuracy: 0.7804
Epoch 3/100
163/163 [=====] - 44s 274ms/step - loss: 0.4943 - accuracy: 0.7646 - val_loss: 0.5459 - val_accuracy: 0.7214
Epoch 4/100
163/163 [=====] - 45s 279ms/step - loss: 0.4123 - accuracy: 0.8221 - val_loss: 0.4822 - val_accuracy: 0.7875
Epoch 5/100
163/163 [=====] - 68s 421ms/step - loss: 0.3873 - accuracy: 0.8275 - val_loss: 0.5217 - val_accuracy: 0.7661
Epoch 6/100
163/163 [=====] - 46s 283ms/step - loss: 0.3534 - accuracy: 0.8397 - val_loss: 0.4933 - val_accuracy: 0.7804
Epoch 7/100
163/163 [=====] - 46s 286ms/step - loss: 0.2903 - accuracy: 0.8742 - val_loss: 0.4357 - val_accuracy: 0.8125
Epoch 8/100
163/163 [=====] - 47s 284ms/step - loss: 0.2421 - accuracy: 0.9018 - val_loss: 0.4723 - val_accuracy: 0.8196
Epoch 9/100
163/163 [=====] - 45s 280ms/step - loss: 0.2270 - accuracy: 0.9118 - val_loss: 0.5452 - val_accuracy: 0.7946
Epoch 10/100
163/163 [=====] - 44s 273ms/step - loss: 0.2179 - accuracy: 0.9057 - val_loss: 0.4379 - val_accuracy: 0.8286
Epoch 11/100
163/163 [=====] - 45s 274ms/step - loss: 0.1682 - accuracy: 0.9394 - val_loss: 0.6762 - val_accuracy: 0.7607
Epoch 12/100
163/163 [=====] - 44s 270ms/step - loss: 0.1474 - accuracy: 0.9363 - val_loss: 0.5760 - val_accuracy: 0.7946
Epoch 13/100
163/163 [=====] - 44s 270ms/step - loss: 0.1054 - accuracy: 0.9624 - val_loss: 0.4686 - val_accuracy: 0.8429
Epoch 14/100
163/163 [=====] - 68s 421ms/step - loss: 0.0794 - accuracy: 0.9778 - val_loss: 0.6259 - val_accuracy: 0.8393
Epoch 15/100
163/163 [=====] - 45s 280ms/step - loss: 0.0958 - accuracy: 0.9617 - val_loss: 0.5838 - val_accuracy: 0.8268
Epoch 16/100
163/163 [=====] - 45s 278ms/step - loss: 0.0774 - accuracy: 0.9709 - val_loss: 0.7182 - val_accuracy: 0.8232
Epoch 17/100
```

```

163/163 [=====] - ETA: 0s - loss: 0.0783 - accuracy: 0.9716Restoring model weights from the end of the best epoch: 7.
163/163 [=====] - 53s 327ms/step - loss: 0.0783 - accuracy: 0.9716 - val_loss: 0.6321 - val_accuracy: 0.8357
Epoch 17: early stopping
Wall time: 13min 53s

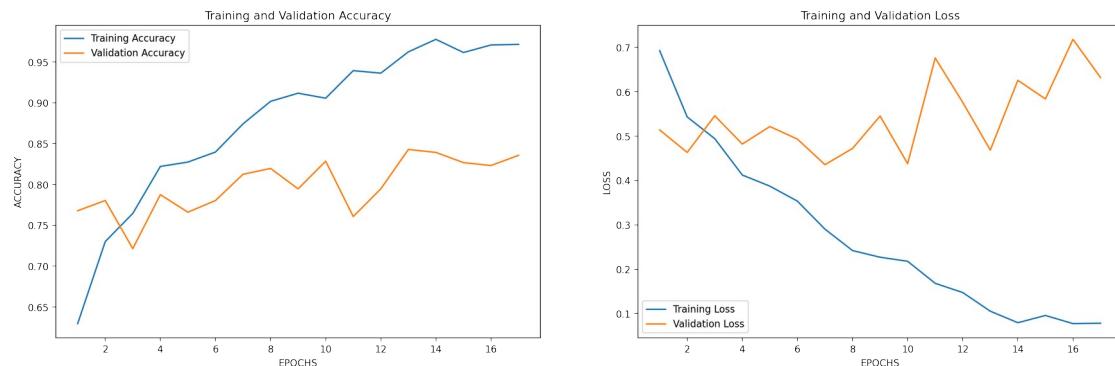
```

```

# Store history
utils.store_model_history(model_name, history.history)

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob =
MRNet_Model_TF_5_VGG.predict(utils.predict_batch_generator_tf_5(X_test,
BATCH_SIZE))

# Now get the correct labels based on the optimal threshold
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Tear'])

26/26 [=====] - 6s 238ms/step
Best cutoff Threshold=0.571411, F-Score=0.800

```

Evaluation Metrics:

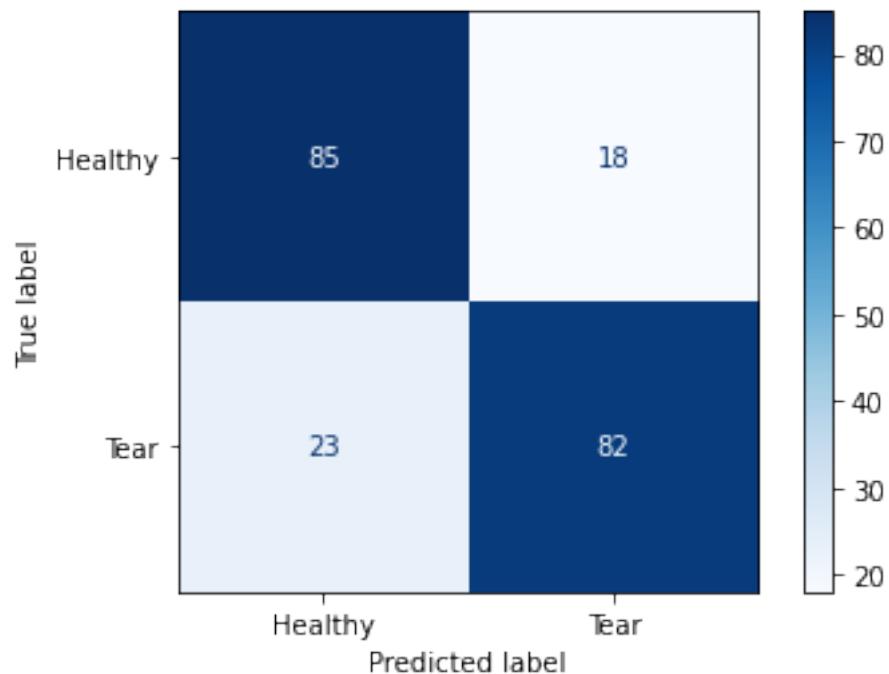
```

Balanced Accuracy : 0.8
Precision : 0.82
Recall : 0.78
F1 Score: 0.8
ROC AUC Score : 0.87
Classification report :
      precision    recall   f1-score   support
Healthy        0.79      0.83      0.81       103

```

Tear	0.82	0.78	0.80	105
accuracy			0.80	208
macro avg	0.80	0.80	0.80	208
weighted avg	0.80	0.80	0.80	208

Confusion Matrix :



```

model_name = 'MRNet_Model_TF_5_ResNet'
MRNet_Model_TF_5_ResNet = models.mri_model_tf_5_resnet(model_name, 2)
MRNet_Model_TF_5_ResNet.compile(optimizer=keras.optimizers.Adam(learning_rate=
utils.model_lr_schedule()),
                                loss='binary_crossentropy',
                                metrics=['accuracy'])
MRNet_Model_TF_5_ResNet.summary()
Model: "MRNet_Model_TF_5_ResNet"

```

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 8, 8, 2048)	23593984
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67109376
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0

<i>dense_2</i> (<i>Dense</i>)	(<i>None</i> , 128)	32896
<i>dropout_2</i> (<i>Dropout</i>)	(<i>None</i> , 128)	0
<i>dense_3</i> (<i>Dense</i>)	(<i>None</i> , 1)	129

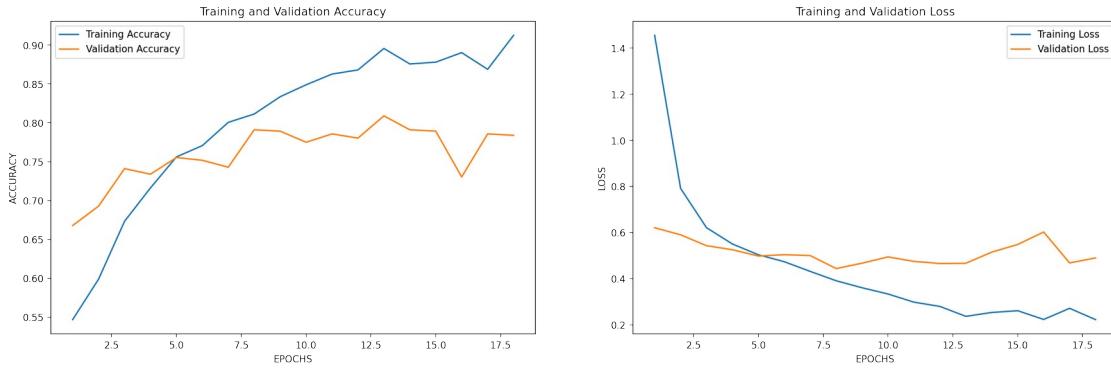
Total params: 90,867,713
Trainable params: 67,273,729
Non-trainable params: 23,593,984

```
%%time
with tf.device('/device:GPU:0'):
    history = MRNet_Model_TF_5_ResNet.fit(utils.batch_generator_tf_5(X_train,
y_train, BATCH_SIZE),
steps_per_epoch=len(X_train)//BATCH_SIZE,
            epochs=EPOCHS,
validation_data=utils.batch_generator_tf_5(X_valid, y_valid, BATCH_SIZE),
validation_steps=len(X_valid)//BATCH_SIZE,
            shuffle=True,
            class_weight=mrnet_class_weights,
            verbose=1,
callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()])
Epoch 1/100
163/163 [=====] - 60s 313ms/step - loss: 1.4550 - accuracy: 0.5468 - val_loss: 0.6207 - val_accuracy: 0.6679
Epoch 2/100
163/163 [=====] - 44s 272ms/step - loss: 0.7916 - accuracy: 0.5989 - val_loss: 0.5901 - val_accuracy: 0.6929
Epoch 3/100
163/163 [=====] - 44s 270ms/step - loss: 0.6209 - accuracy: 0.6733 - val_loss: 0.5428 - val_accuracy: 0.7411
Epoch 4/100
163/163 [=====] - 44s 270ms/step - loss: 0.5502 - accuracy: 0.7163 - val_loss: 0.5253 - val_accuracy: 0.7339
Epoch 5/100
163/163 [=====] - 43s 268ms/step - loss: 0.5035 - accuracy: 0.7561 - val_loss: 0.4981 - val_accuracy: 0.7554
Epoch 6/100
163/163 [=====] - 41s 253ms/step - loss: 0.4734 - accuracy: 0.7707 - val_loss: 0.5038 - val_accuracy: 0.7518
Epoch 7/100
163/163 [=====] - 40s 248ms/step - loss: 0.4314 - accuracy: 0.8006 - val_loss: 0.5001 - val_accuracy: 0.7429
```

```
Epoch 8/100
163/163 [=====] - 42s 261ms/step - loss: 0.3909 -
accuracy: 0.8113 - val_loss: 0.4437 - val_accuracy: 0.7911
Epoch 9/100
163/163 [=====] - 41s 253ms/step - loss: 0.3604 -
accuracy: 0.8336 - val_loss: 0.4671 - val_accuracy: 0.7893
Epoch 10/100
163/163 [=====] - 40s 249ms/step - loss: 0.3334 -
accuracy: 0.8489 - val_loss: 0.4942 - val_accuracy: 0.7750
Epoch 11/100
163/163 [=====] - 39s 243ms/step - loss: 0.2980 -
accuracy: 0.8627 - val_loss: 0.4748 - val_accuracy: 0.7857
Epoch 12/100
163/163 [=====] - 39s 241ms/step - loss: 0.2796 -
accuracy: 0.8681 - val_loss: 0.4656 - val_accuracy: 0.7804
Epoch 13/100
163/163 [=====] - 39s 243ms/step - loss: 0.2365 -
accuracy: 0.8957 - val_loss: 0.4667 - val_accuracy: 0.8089
Epoch 14/100
163/163 [=====] - 39s 242ms/step - loss: 0.2533 -
accuracy: 0.8758 - val_loss: 0.5151 - val_accuracy: 0.7911
Epoch 15/100
163/163 [=====] - 39s 238ms/step - loss: 0.2610 -
accuracy: 0.8781 - val_loss: 0.5482 - val_accuracy: 0.7893
Epoch 16/100
163/163 [=====] - 39s 240ms/step - loss: 0.2231 -
accuracy: 0.8903 - val_loss: 0.6022 - val_accuracy: 0.7304
Epoch 17/100
163/163 [=====] - 39s 239ms/step - loss: 0.2713 -
accuracy: 0.8689 - val_loss: 0.4681 - val_accuracy: 0.7857
Epoch 18/100
163/163 [=====] - ETA: 0s - loss: 0.2224 - accuracy:
0.9126Restoring model weights from the end of the best epoch: 8.
163/163 [=====] - 39s 239ms/step - loss: 0.2224 -
accuracy: 0.9126 - val_loss: 0.4898 - val_accuracy: 0.7839
Epoch 18: early stopping
Wall time: 12min 32s
```

```
# Store history
utils.store_model_history(model_name, history.history)
```

```
# Plot training graphs
utils.plot_acc_loss(history.history)
```



```
# Evaluate model
```

```
X_test_prob =  
MRNet_Model_TF_5_ResNet.predict(utils.predict_batch_generator_tf_5(X_test,  
BATCH_SIZE))
```

```
# Now get the correct labels based on the optimal threshold
```

```
optimal_threshold = utils.calculate_best_cutoff_threshold(y_test, X_test_prob)  
X_test_pred = (X_test_prob >= optimal_threshold).astype('int')
```

```
utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),  
['Healthy', 'Tear'])
```

```
26/26 [=====] - 11s 421ms/step
```

```
Best cutoff Threshold=0.343872, F-Score=0.795
```

Evaluation Metrics:

Balanced Accuracy : 0.77

Precision : 0.73

Recall : 0.87

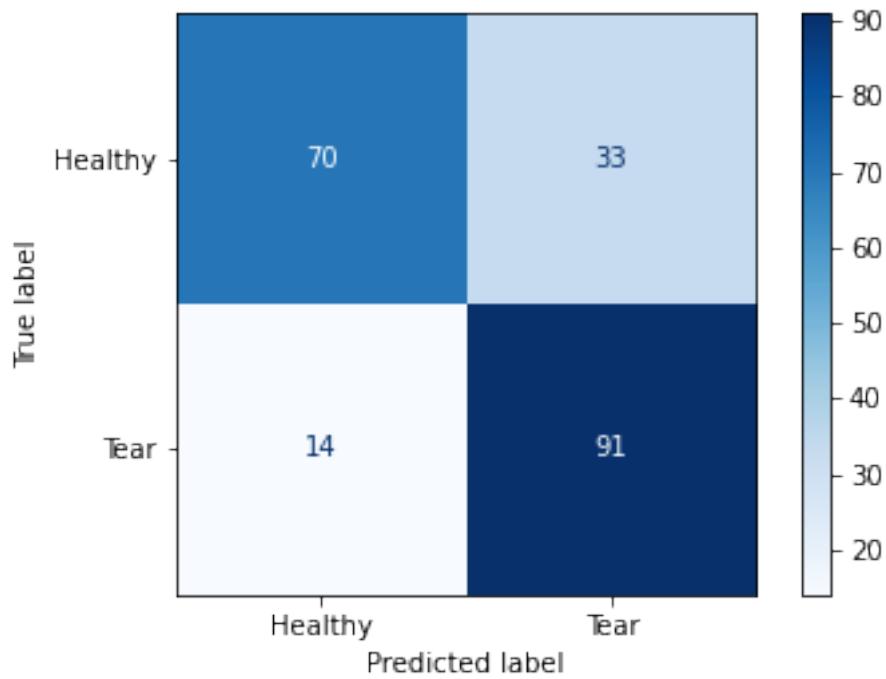
F1 Score: 0.79

ROC AUC Score : 0.87

Classification report :

	precision	recall	f1-score	support
Healthy	0.83	0.68	0.75	103
Tear	0.73	0.87	0.79	105
accuracy			0.77	208
macro avg	0.78	0.77	0.77	208
weighted avg	0.78	0.77	0.77	208

Confusion Matrix :



J) KneeMRI Transfer Learning

```

import platform
import pandas as pd
import numpy as np
from glob import glob
from tensorflow import keras

from sklearn.model_selection import train_test_split
import utils
import models

```

Tensorflow version : 2.10.1

Tensorflow devices available :

```

[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {}
incarnation: 14024334381040161164
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 22672310272
locality {
  bus_id: 1
  links {

```

```

        }
    }
incarnation: 17632151391174911882
physical_device_desc: "device: 0, name: NVIDIA RTX A5000, pci bus id: 0000:61:00.0, compute capability: 8.6"
xla_global_id: 416903419
]

Tensorflow physical devices available :
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]

# Directory where the preprocessed volumetric data is located
kneemri_preprocessed_data_dir = 'Preprocessed_Data/KneeMRI'
# path to metadata csv file
kneemri_metadata_csv_path = 'Data/KneeMRI/metadata.csv'
kneemri_aug_metadata_csv_path = 'Data/KneeMRI/metadata-aug.csv'

# For running code on Windows
if platform.system() == "Windows":
    kneemri_preprocessed_data_dir = kneemri_preprocessed_data_dir.replace('/', '\\')
    kneemri_metadata_csv_path = kneemri_metadata_csv_path.replace('/', '\\')
    kneemri_aug_metadata_csv_path = kneemri_aug_metadata_csv_path.replace('/', '\\')

kneemri_classes = { 0:'healthy', 1:'partially ruptured', 2:'completely ruptured' }

if platform.system() == "Windows":
    kneemri_vol_paths = glob(kneemri_preprocessed_data_dir+"\\vol*")
else:
    kneemri_vol_paths = glob(kneemri_preprocessed_data_dir+"/vol*")
kneemri_vol_paths.sort()

kneemri_vol_paths

['Preprocessed_Data\\KneeMRI\\vol01',
 'Preprocessed_Data\\KneeMRI\\vol02',
 'Preprocessed_Data\\KneeMRI\\vol03',
 'Preprocessed_Data\\KneeMRI\\vol04',
 'Preprocessed_Data\\KneeMRI\\vol05',
 'Preprocessed_Data\\KneeMRI\\vol06',
 'Preprocessed_Data\\KneeMRI\\vol07',
 'Preprocessed_Data\\KneeMRI\\vol08',
 'Preprocessed_Data\\KneeMRI\\vol09',
 'Preprocessed_Data\\KneeMRI\\vol10']

kneemri_cases = []
for mri_data_path in kneemri_vol_paths:
    if platform.system() == "Windows":
        all_exams = glob(mri_data_path+"\\*.npy")
    else:

```

```

    all_exams = glob(mri_data_path+"*.npy")
    all_exams.sort()
    kneemri_cases.extend(all_exams)

print('Original cases : ', len(kneemri_cases))

Original cases : 917

kneemri_aug_cases = []
for mri_data_path in kneemri_vol_paths:
    if platform.system() == "Windows":
        all_exams = glob(mri_data_path+"\aug\*.npy")
    else:
        all_exams = glob(mri_data_path+"/aug/*.npy")
    all_exams.sort()
    kneemri_aug_cases.extend(all_exams)

print('Augmented cases : ', len(kneemri_aug_cases))

Augmented cases : 650

# names=True loads the interprets the first row of csv file as column names
# 'i4' = 4 byte signed integer, 'U20' = unicode max 20 char string
kneemri_metadata = np.genfromtxt(kneemri_metadata_csv_path, delimiter=',',
                                 names=True,
                                 dtype='i4,i4,i4,i4,i4,i4,i4,i4,i4,U20')

kneemri_metadata_df = pd.DataFrame(kneemri_metadata)
kneemri_metadata_df

      examId  seriesNo  aclDiagnosis  kneeLR  roiX  roiY  roiZ  roiHeight \
0     329637         8            0       1   139   184   14        74
1     390116         9            0       0   113   105   10        83
2     404663         8            1       1   120   117   15       101
3     406320         9            0       0   117   124   12        91
4     412857         8            0       1   122   105   14        83
..   ...
912  1027212        5            1       1   113   127   16       101
913  1028019        5            1       1   105   102   14        95
914  1028028        5            0       0   118    84   15       100
915  1028069        5            0       0   105    97   15       103
916  1028670        5            1       0   113   108   14       103

      roiWidth  roiDepth volumeFilename
0          72         3  329637-8.pck
1          98         6  390116-9.pck
2         115         2  404663-8.pck
3          80         3  406320-9.pck
4          98         4  412857-8.pck
..   ...
912         99         3  1027212-5.pck
913         100        3  1028019-5.pck
914         100        2  1028028-5.pck

```

```
915      106      4  1028069-5.pck
916      110      4  1028670-5.pck
```

[917 rows x 11 columns]

```
kneemri_aug_metadata_df = pd.read_csv(kneemri_aug_metadata_csv_path,
index_col=0)
```

kneemri_aug_metadata_df

	examId	seriesNo	aclDiagnosis	kneeLR	roiX	roiY	roiZ	roiHeight	\
0	404663	8		1	120	117	15		101
1	404663	8		1	120	117	15		101
2	412857	8		0	122	105	14		83
3	412865	8		1	0	111	133	13	78
4	412865	8		1	0	111	133	13	78
..
645	995153	5		2	1	113	122	13	93
646	995153	5		2	1	113	122	13	93
647	995153	5		2	1	113	122	13	93
648	995153	5		2	1	113	122	13	93
649	996739	6		0	0	121	135	14	96
	roiWidth	roiDepth		volumeFilename					
0	115	2		404663-8-aug-0.pck					
1	115	2		404663-8-aug-1.pck					
2	98	4		412857-8-aug-0.pck					
3	78	4		412865-8-aug-0.pck					
4	78	4		412865-8-aug-1.pck					
..					
645	98	3		995153-5-aug-1.pck					
646	98	3		995153-5-aug-2.pck					
647	98	3		995153-5-aug-3.pck					
648	98	3		995153-5-aug-4.pck					
649	92	3		996739-6-aug-0.pck					

[650 rows x 11 columns]

```
kneemri_full_metadata_df = pd.concat([kneemri_metadata_df,
kneemri_aug_metadata_df], ignore_index=True)
```

kneemri_full_metadata_df

	examId	seriesNo	aclDiagnosis	kneeLR	roiX	roiY	roiZ	roiHeight	\
0	329637	8	0	1	139	184	14		74
1	390116	9	0	0	113	105	10		83
2	404663	8	1	1	120	117	15		101
3	406320	9	0	0	117	124	12		91
4	412857	8	0	1	122	105	14		83
..
1562	995153	5	2	1	113	122	13		93
1563	995153	5	2	1	113	122	13		93
1564	995153	5	2	1	113	122	13		93

```
1565 995153      5      2      1    113    122    13    93
1566 996739      6      0      0    121    135    14    96
```

```
    roiWidth  roiDepth  volumeFilename
0        72        3      329637-8.pck
1        98        6      390116-9.pck
2       115        2      404663-8.pck
3        80        3      406320-9.pck
4        98        4      412857-8.pck
...
1562     98        3  995153-5-aug-1.pck
1563     98        3  995153-5-aug-2.pck
1564     98        3  995153-5-aug-3.pck
1565     98        3  995153-5-aug-4.pck
1566     92        3  996739-6-aug-0.pck
```

[1567 rows x 11 columns]

```
kneemri_filenames = []
kneemri_filenames.extend(kneemri_cases)
kneemri_filenames.extend(kneemri_aug_cases)
kneemri_filenames.sort()

kneemri_labels = utils.get_correct_labels_kneemri(kneemri_filenames,
kneemri_full_metadata_df)

# Splitting into train, test and validation

X, X_test, y, y_test = train_test_split(kneemri_filenames,
                                         kneemri_labels,
                                         test_size=0.1,
                                         random_state=610,
                                         shuffle=True,
                                         stratify=kneemri_labels)

X_train, X_valid, y_train, y_valid = train_test_split(X,
                                                       y,
                                                       train_size=0.7,
                                                       random_state=610,
                                                       shuffle=True,
                                                       stratify=y)

BATCH_SIZE = 8
EPOCHS = 100
kneemri_class_weights = utils.compute_class_weights(y_train)

kneemri_class_weights
{0: 0.723935389133627, 1: 1.01440329218107, 2: 1.580128205128205}

model_name = 'kneeMRI_Model_TF_3_VGG'
kneeMRI_Model_TF_3_VGG = models.mri_model_tf_3_vgg(model_name,
len(kneemri_classes))
```

```

kneeMRI_Model_TF_3_VGG.compile(optimizer=keras.optimizers.Adam(learning_rate=u
tills.model_lr_schedule()),
                                loss='sparse_categorical_crossentropy',
                                metrics=['accuracy'])
kneeMRI_Model_TF_3_VGG.summary()

```

Model: "kneeMRI_Model_TF_3_VGG"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14714688
flatten_1 (Flatten)	(None, 32768)	0
dense_4 (Dense)	(None, 512)	16777728
dropout_3 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 256)	131328
dropout_4 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32896
dropout_5 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 3)	387

Total params: 31,657,027
Trainable params: 16,942,339
Non-trainable params: 14,714,688

```

%%time
with tf.device('/device:GPU:0'):
    history = kneeMRI_Model_TF_3_VGG.fit(utils.batch_generator_tf_3(X_train,
y_train, BATCH_SIZE),
                                         steps_per_epoch=len(X_train)//BATCH_SIZE,
                                         epochs=EP0CHS,
                                         validation_data=utils.batch_generator_tf_3(X_valid, y_valid, BATCH_SIZE),
                                         validation_steps=len(X_valid)//BATCH_SIZE,
                                         shuffle=True,
                                         class_weight=kneemri_class_weights,
                                         verbose=1,
                                         callbacks=[utils.model_callback_checkpoint(model_name),
                                         utils.model_callback_earlystopping()])

```

Epoch 1/100
123/123 [=====] - 35s 274ms/step - loss: 1.3504 -
accuracy: 0.4248 - val_loss: 0.8425 - val_accuracy: 0.6321
Epoch 2/100
123/123 [=====] - 30s 243ms/step - loss: 1.0263 -
accuracy: 0.5335 - val_loss: 0.8178 - val_accuracy: 0.6297
Epoch 3/100
123/123 [=====] - 29s 235ms/step - loss: 0.9220 -
accuracy: 0.5610 - val_loss: 0.8908 - val_accuracy: 0.5873
Epoch 4/100
123/123 [=====] - 29s 235ms/step - loss: 0.8643 -
accuracy: 0.6148 - val_loss: 0.8526 - val_accuracy: 0.6085
Epoch 5/100
123/123 [=====] - 30s 245ms/step - loss: 0.8277 -
accuracy: 0.6280 - val_loss: 0.7698 - val_accuracy: 0.7099
Epoch 6/100
123/123 [=====] - 29s 234ms/step - loss: 0.7712 -
accuracy: 0.6494 - val_loss: 0.7951 - val_accuracy: 0.6462
Epoch 7/100
123/123 [=====] - 29s 238ms/step - loss: 0.6849 -
accuracy: 0.7195 - val_loss: 0.7279 - val_accuracy: 0.7028
Epoch 8/100
123/123 [=====] - 29s 237ms/step - loss: 0.6388 -
accuracy: 0.7419 - val_loss: 0.7451 - val_accuracy: 0.6698
Epoch 9/100
123/123 [=====] - 29s 240ms/step - loss: 0.6355 -
accuracy: 0.7429 - val_loss: 0.6947 - val_accuracy: 0.7429
Epoch 10/100
123/123 [=====] - 29s 241ms/step - loss: 0.5542 -
accuracy: 0.7642 - val_loss: 0.6428 - val_accuracy: 0.7476
Epoch 11/100
123/123 [=====] - 28s 230ms/step - loss: 0.5245 -
accuracy: 0.7805 - val_loss: 0.7547 - val_accuracy: 0.6910
Epoch 12/100
123/123 [=====] - 29s 234ms/step - loss: 0.4364 -
accuracy: 0.8069 - val_loss: 0.7719 - val_accuracy: 0.6509
Epoch 13/100
123/123 [=====] - 28s 229ms/step - loss: 0.4085 -
accuracy: 0.8242 - val_loss: 0.6633 - val_accuracy: 0.7453
Epoch 14/100
123/123 [=====] - 28s 228ms/step - loss: 0.3653 -
accuracy: 0.8415 - val_loss: 0.7006 - val_accuracy: 0.7335
Epoch 15/100
123/123 [=====] - 28s 226ms/step - loss: 0.3030 -
accuracy: 0.8821 - val_loss: 0.7555 - val_accuracy: 0.7288
Epoch 16/100
123/123 [=====] - 28s 227ms/step - loss: 0.2548 -
accuracy: 0.9106 - val_loss: 0.6971 - val_accuracy: 0.7476
Epoch 17/100
123/123 [=====] - 28s 229ms/step - loss: 0.2354 -
accuracy: 0.9106 - val_loss: 0.6782 - val_accuracy: 0.7288
Epoch 18/100

```

123/123 [=====] - 28s 227ms/step - loss: 0.1953 - accuracy: 0.9268 - val_loss: 0.8268 - val_accuracy: 0.7288
Epoch 19/100
123/123 [=====] - 28s 225ms/step - loss: 0.1706 - accuracy: 0.9390 - val_loss: 0.9212 - val_accuracy: 0.7358
Epoch 20/100
123/123 [=====] - ETA: 0s - loss: 0.1980 - accuracy: 0.9187Restoring model weights from the end of the best epoch: 10.
123/123 [=====] - 28s 226ms/step - loss: 0.1980 - accuracy: 0.9187 - val_loss: 0.8115 - val_accuracy: 0.7099
Epoch 20: early stopping
Wall time: 9min 37s

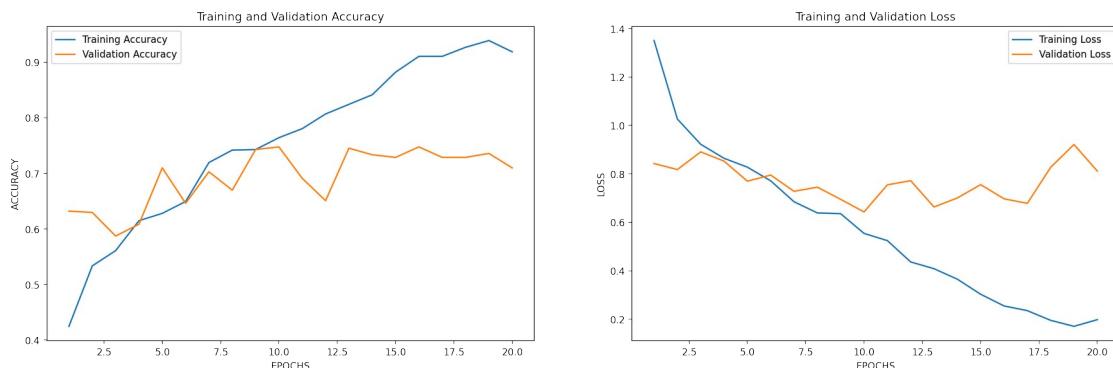
```

```

# Store history
utils.store_model_history(model_name, history.history)

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob =
kneeMRI_Model_TF_3_VGG.predict(utils.predict_batch_generator_tf_3(X_test,
BATCH_SIZE))

X_test_pred = X_test_prob.argmax(axis=-1)

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])

20/20 [=====] - 5s 265ms/step

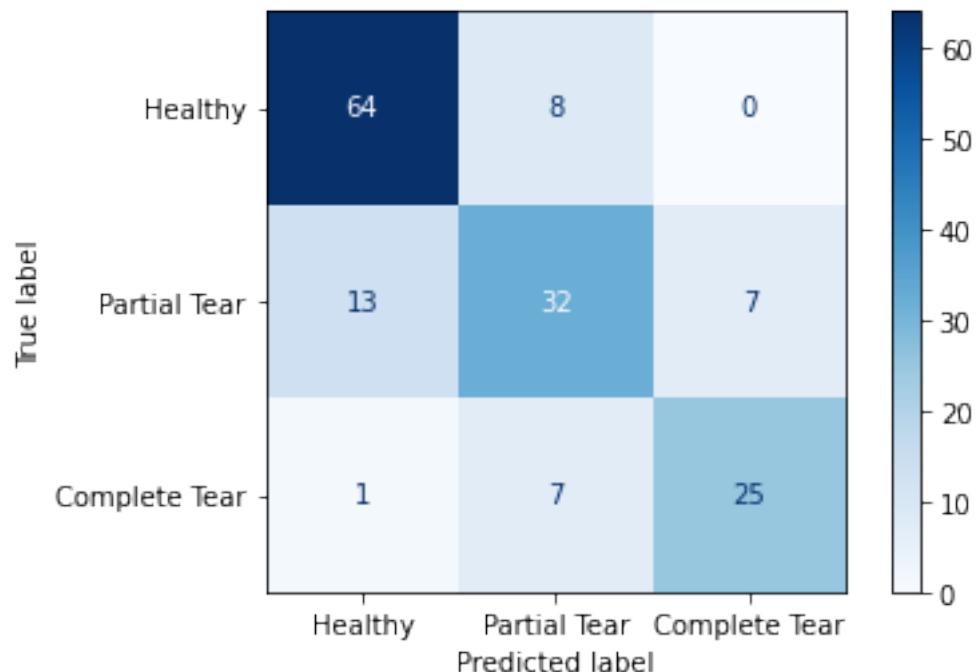
```

Evaluation Metrics:

Balanced Accuracy : 0.75
Precision : 0.77
Recall : 0.77
F1 Score: 0.77
ROC AUC Score : 0.9
Classification report :

	<i>precision</i>	<i>recall</i>	<i>f1-score</i>	<i>support</i>
Healthy	0.82	0.89	0.85	72
Partial Tear	0.68	0.62	0.65	52
Complete Tear	0.78	0.76	0.77	33
<i>accuracy</i>			0.77	157
<i>macro avg</i>	0.76	0.75	0.76	157
<i>weighted avg</i>	0.77	0.77	0.77	157

Confusion Matrix :



```

model_name = 'kneeMRI_Model_TF_3_Xception'
kneeMRI_Model_TF_3_Xception = models.mri_model_tf_3_xception(model_name,
len(kneemri_classes))
kneeMRI_Model_TF_3_Xception.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                                         loss='sparse_categorical_crossentropy',
                                         metrics=['accuracy'])
kneeMRI_Model_TF_3_Xception.summary()

```

Model: "kneeMRI_Model_TF_3_Xception"

Layer (type)	Output Shape	Param #
xception (Functional)	(None, 8, 8, 2048)	20861480
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67109376

<i>dropout</i> (<i>Dropout</i>)	(None, 512)	0
<i>dense_1</i> (<i>Dense</i>)	(None, 256)	131328
<i>dropout_1</i> (<i>Dropout</i>)	(None, 256)	0
<i>dense_2</i> (<i>Dense</i>)	(None, 128)	32896
<i>dropout_2</i> (<i>Dropout</i>)	(None, 128)	0
<i>dense_3</i> (<i>Dense</i>)	(None, 3)	387

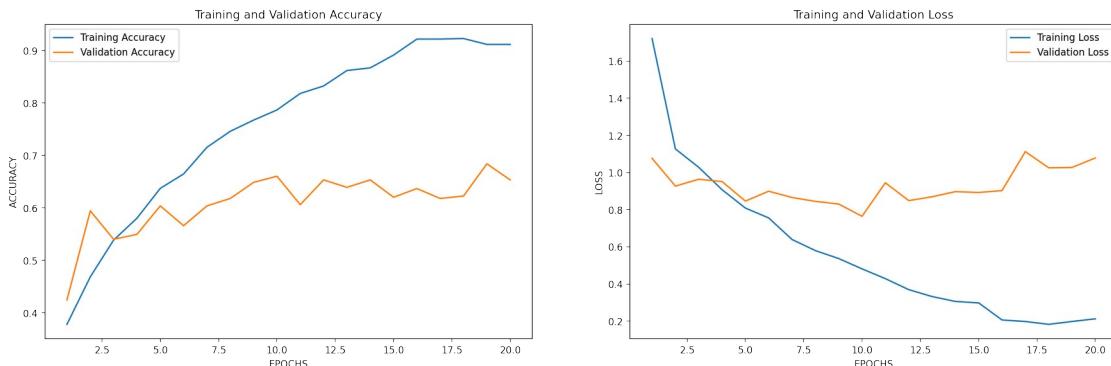
```
=====
Total params: 88,135,467
Trainable params: 67,273,987
Non-trainable params: 20,861,480
```

```
%%time
with tf.device('/device:GPU:0'):
    history =
kneeMRI_Model_TF_3_Xception.fit(utils.batch_generator_tf_3(X_train, y_train,
BATCH_SIZE),
                                    steps_per_epoch=len(X_train)//BATCH_SIZE,
                                    epochs=EP0CHS,
                                    validation_data=utils.batch_generator_tf_3(X_valid, y_valid, BATCH_SIZE),
                                    validation_steps=len(X_valid)//BATCH_SIZE,
                                    shuffle=True,
                                    class_weight=kneemri_class_weights,
                                    verbose=1,
                                    callbacks=[utils.model_callback_checkpoint(model_name),
                                               utils.model_callback_earlystopping()])
Epoch 1/100
123/123 [=====] - 46s 302ms/step - loss: 1.7219 - accuracy: 0.3780 - val_loss: 1.0766 - val_accuracy: 0.4245
Epoch 2/100
123/123 [=====] - 31s 252ms/step - loss: 1.1274 - accuracy: 0.4685 - val_loss: 0.9270 - val_accuracy: 0.5943
Epoch 3/100
123/123 [=====] - 29s 234ms/step - loss: 1.0282 - accuracy: 0.5386 - val_loss: 0.9639 - val_accuracy: 0.5401
Epoch 4/100
123/123 [=====] - 28s 232ms/step - loss: 0.9076 - accuracy: 0.5803 - val_loss: 0.9515 - val_accuracy: 0.5495
Epoch 5/100
123/123 [=====] - 30s 244ms/step - loss: 0.8088 - accuracy: 0.6372 - val_loss: 0.8464 - val_accuracy: 0.6038
Epoch 6/100
```

```
123/123 [=====] - 28s 230ms/step - loss: 0.7556 -  
accuracy: 0.6646 - val_loss: 0.8995 - val_accuracy: 0.5660  
Epoch 7/100  
123/123 [=====] - 28s 226ms/step - loss: 0.6394 -  
accuracy: 0.7154 - val_loss: 0.8657 - val_accuracy: 0.6038  
Epoch 8/100  
123/123 [=====] - 29s 237ms/step - loss: 0.5797 -  
accuracy: 0.7459 - val_loss: 0.8447 - val_accuracy: 0.6179  
Epoch 9/100  
123/123 [=====] - 31s 253ms/step - loss: 0.5369 -  
accuracy: 0.7673 - val_loss: 0.8305 - val_accuracy: 0.6486  
Epoch 10/100  
123/123 [=====] - 31s 251ms/step - loss: 0.4813 -  
accuracy: 0.7866 - val_loss: 0.7643 - val_accuracy: 0.6604  
Epoch 11/100  
123/123 [=====] - 28s 230ms/step - loss: 0.4286 -  
accuracy: 0.8181 - val_loss: 0.9449 - val_accuracy: 0.6061  
Epoch 12/100  
123/123 [=====] - 28s 226ms/step - loss: 0.3695 -  
accuracy: 0.8323 - val_loss: 0.8489 - val_accuracy: 0.6533  
Epoch 13/100  
123/123 [=====] - 27s 222ms/step - loss: 0.3322 -  
accuracy: 0.8618 - val_loss: 0.8694 - val_accuracy: 0.6392  
Epoch 14/100  
123/123 [=====] - 27s 223ms/step - loss: 0.3058 -  
accuracy: 0.8669 - val_loss: 0.8972 - val_accuracy: 0.6533  
Epoch 15/100  
123/123 [=====] - 27s 222ms/step - loss: 0.2979 -  
accuracy: 0.8913 - val_loss: 0.8927 - val_accuracy: 0.6203  
Epoch 16/100  
123/123 [=====] - 27s 220ms/step - loss: 0.2060 -  
accuracy: 0.9217 - val_loss: 0.9029 - val_accuracy: 0.6368  
Epoch 17/100  
123/123 [=====] - 27s 220ms/step - loss: 0.1977 -  
accuracy: 0.9217 - val_loss: 1.1129 - val_accuracy: 0.6179  
Epoch 18/100  
123/123 [=====] - 27s 220ms/step - loss: 0.1825 -  
accuracy: 0.9228 - val_loss: 1.0257 - val_accuracy: 0.6226  
Epoch 19/100  
123/123 [=====] - 27s 221ms/step - loss: 0.1978 -  
accuracy: 0.9116 - val_loss: 1.0274 - val_accuracy: 0.6840  
Epoch 20/100  
123/123 [=====] - ETA: 0s - loss: 0.2122 - accuracy:  
0.9116Restoring model weights from the end of the best epoch: 10.  
123/123 [=====] - 27s 218ms/step - loss: 0.2122 -  
accuracy: 0.9116 - val_loss: 1.0785 - val_accuracy: 0.6533  
Epoch 20: early stopping  
Wall time: 9min 41s
```

```
# Store history  
utils.store_model_history(model_name, history.history)
```

```
# Plot training graphs
utils.plot_acc_loss(history.history)
```



```
# Evaluate model
X_test_prob =
kneeMRI_Model_TF_3_Xception.predict(utils.predict_batch_generator_tf_3(X_test,
BATCH_SIZE))

X_test_pred = X_test_prob.argmax(axis=-1)

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])

20/20 [=====] - 10s 507ms/step
```

Evaluation Metrics:

Balanced Accuracy : 0.68

Precision : 0.7

Recall : 0.69

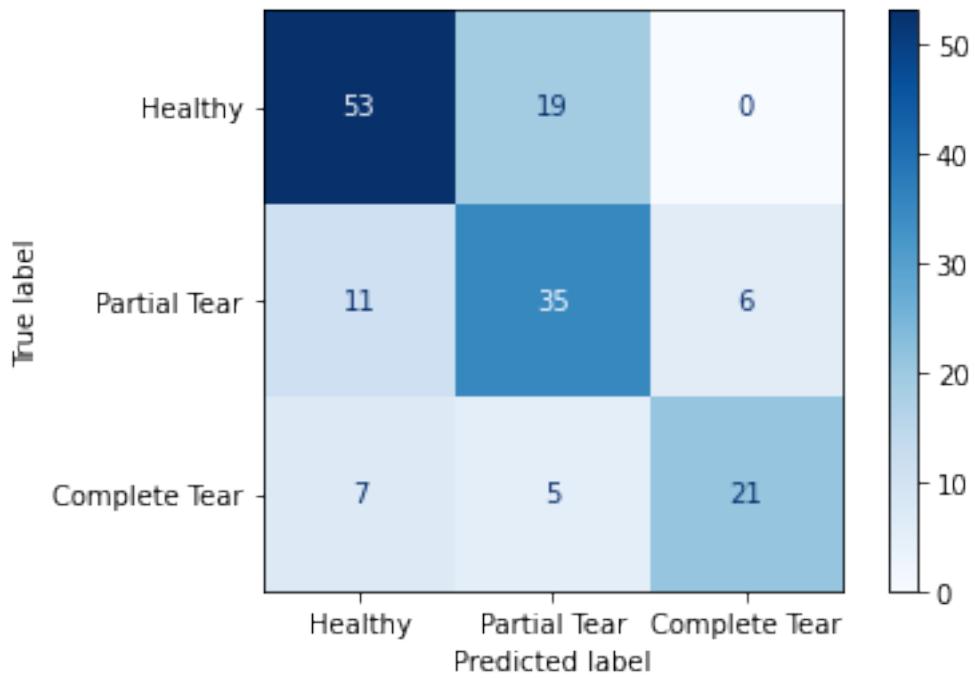
F1 Score: 0.7

ROC AUC Score : 0.84

Classification report :

	precision	recall	f1-score	support
Healthy	0.75	0.74	0.74	72
Partial Tear	0.59	0.67	0.63	52
Complete Tear	0.78	0.64	0.70	33
accuracy			0.69	157
macro avg	0.71	0.68	0.69	157
weighted avg	0.70	0.69	0.70	157

Confusion Matrix :



```

model_name = 'kneeMRI_Model_TF_3_ResNet'
kneeMRI_Model_TF_3_ResNet = models.mri_model_tf_3_resnet(model_name,
len(kneemri_classes))
kneeMRI_Model_TF_3_ResNet.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                                    loss='sparse_categorical_crossentropy',
                                    metrics=['accuracy'])
kneeMRI_Model_TF_3_ResNet.summary()

```

Model: "kneeMRI_Model_TF_3_ResNet"

Layer (type)	Output Shape	Param #
<hr/>		
resnet50 (Functional)	(None, 8, 8, 2048)	23587712
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 512)	67109376
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

```
=====
```

```
Total params: 90,861,699  
Trainable params: 67,273,987  
Non-trainable params: 23,587,712
```

```
%%time  
with tf.device('/device:GPU:0'):  
    history =  
kneeMRI_Model_TF_3_ResNet.fit(utils.batch_generator_tf_3(X_train, y_train,  
BATCH_SIZE),  
                                steps_per_epoch=len(X_train)//BATCH_SIZE,  
                                epochs=EPOCHS,  
  
validation_data=utils.batch_generator_tf_3(X_valid, y_valid, BATCH_SIZE),  
  
validation_steps=len(X_valid)//BATCH_SIZE,  
                    shuffle=True,  
                    class_weight=kneemri_class_weights,  
                    verbose=1,  
  
callbacks=[utils.model_callback_checkpoint(model_name),  
          utils.model_callback_earlystopping()]  
  
Epoch 1/100  
123/123 [=====] - 48s 316ms/step - loss: 2.3201 -  
accuracy: 0.4096 - val_loss: 0.9742 - val_accuracy: 0.5259  
Epoch 2/100  
123/123 [=====] - 32s 262ms/step - loss: 1.1856 -  
accuracy: 0.4299 - val_loss: 0.9893 - val_accuracy: 0.5613  
Epoch 3/100  
123/123 [=====] - 32s 260ms/step - loss: 1.0316 -  
accuracy: 0.4959 - val_loss: 1.0162 - val_accuracy: 0.4929  
Epoch 4/100  
123/123 [=====] - 32s 259ms/step - loss: 1.0550 -  
accuracy: 0.4980 - val_loss: 1.0149 - val_accuracy: 0.5236  
Epoch 5/100  
123/123 [=====] - 34s 277ms/step - loss: 0.9189 -  
accuracy: 0.5376 - val_loss: 0.8644 - val_accuracy: 0.6486  
Epoch 6/100  
123/123 [=====] - 34s 279ms/step - loss: 0.8897 -  
accuracy: 0.5650 - val_loss: 0.8201 - val_accuracy: 0.6533  
Epoch 7/100  
123/123 [=====] - 32s 258ms/step - loss: 0.9046 -  
accuracy: 0.5264 - val_loss: 0.9058 - val_accuracy: 0.6627  
Epoch 8/100  
123/123 [=====] - 34s 278ms/step - loss: 0.8308 -  
accuracy: 0.5823 - val_loss: 0.7815 - val_accuracy: 0.6910  
Epoch 9/100  
123/123 [=====] - 34s 278ms/step - loss: 0.7872 -  
accuracy: 0.5935 - val_loss: 0.7603 - val_accuracy: 0.6627  
Epoch 10/100
```

123/123 [=====] - 34s 276ms/step - loss: 0.7424 -
accuracy: 0.6463 - val_loss: 0.7059 - val_accuracy: 0.7028
Epoch 11/100
123/123 [=====] - 31s 257ms/step - loss: 0.6313 -
accuracy: 0.7093 - val_loss: 0.7483 - val_accuracy: 0.6840
Epoch 12/100
123/123 [=====] - 32s 258ms/step - loss: 0.6367 -
accuracy: 0.7134 - val_loss: 0.7830 - val_accuracy: 0.6651
Epoch 13/100
123/123 [=====] - 34s 281ms/step - loss: 0.6276 -
accuracy: 0.6870 - val_loss: 0.6851 - val_accuracy: 0.7170
Epoch 14/100
123/123 [=====] - 31s 256ms/step - loss: 0.6052 -
accuracy: 0.7175 - val_loss: 0.7590 - val_accuracy: 0.6274
Epoch 15/100
123/123 [=====] - 31s 256ms/step - loss: 0.5667 -
accuracy: 0.7124 - val_loss: 0.7394 - val_accuracy: 0.6958
Epoch 16/100
123/123 [=====] - 34s 276ms/step - loss: 0.5315 -
accuracy: 0.7236 - val_loss: 0.6764 - val_accuracy: 0.6981
Epoch 17/100
123/123 [=====] - 31s 258ms/step - loss: 0.4975 -
accuracy: 0.7541 - val_loss: 0.6954 - val_accuracy: 0.7288
Epoch 18/100
123/123 [=====] - 32s 262ms/step - loss: 0.4504 -
accuracy: 0.7754 - val_loss: 0.7393 - val_accuracy: 0.6981
Epoch 19/100
123/123 [=====] - 31s 257ms/step - loss: 0.4129 -
accuracy: 0.8069 - val_loss: 0.7229 - val_accuracy: 0.6769
Epoch 20/100
123/123 [=====] - 31s 256ms/step - loss: 0.3747 -
accuracy: 0.8161 - val_loss: 0.9067 - val_accuracy: 0.6344
Epoch 21/100
123/123 [=====] - 31s 252ms/step - loss: 0.3550 -
accuracy: 0.8293 - val_loss: 0.7040 - val_accuracy: 0.7123
Epoch 22/100
123/123 [=====] - 30s 243ms/step - loss: 0.3268 -
accuracy: 0.8425 - val_loss: 0.6769 - val_accuracy: 0.7382
Epoch 23/100
123/123 [=====] - 32s 259ms/step - loss: 0.3339 -
accuracy: 0.8303 - val_loss: 0.6471 - val_accuracy: 0.7547
Epoch 24/100
123/123 [=====] - 33s 267ms/step - loss: 0.3210 -
accuracy: 0.8445 - val_loss: 0.6140 - val_accuracy: 0.7500
Epoch 25/100
123/123 [=====] - 30s 245ms/step - loss: 0.2983 -
accuracy: 0.8567 - val_loss: 0.8682 - val_accuracy: 0.6934
Epoch 26/100
123/123 [=====] - 30s 244ms/step - loss: 0.2447 -
accuracy: 0.8872 - val_loss: 0.7160 - val_accuracy: 0.7453
Epoch 27/100
123/123 [=====] - 30s 246ms/step - loss: 0.2827 -

```

accuracy: 0.8587 - val_loss: 0.9516 - val_accuracy: 0.6698
Epoch 28/100
123/123 [=====] - 31s 249ms/step - loss: 0.2633 -
accuracy: 0.8516 - val_loss: 0.7772 - val_accuracy: 0.7170
Epoch 29/100
123/123 [=====] - 28s 230ms/step - loss: 0.2297 -
accuracy: 0.8892 - val_loss: 0.7678 - val_accuracy: 0.7712
Epoch 30/100
123/123 [=====] - 25s 203ms/step - loss: 0.2550 -
accuracy: 0.8618 - val_loss: 0.8423 - val_accuracy: 0.7453
Epoch 31/100
123/123 [=====] - 24s 195ms/step - loss: 0.2323 -
accuracy: 0.8750 - val_loss: 0.8817 - val_accuracy: 0.7358
Epoch 32/100
123/123 [=====] - 24s 197ms/step - loss: 0.2442 -
accuracy: 0.8770 - val_loss: 0.6182 - val_accuracy: 0.7406
Epoch 33/100
123/123 [=====] - 24s 195ms/step - loss: 0.2517 -
accuracy: 0.8730 - val_loss: 0.7250 - val_accuracy: 0.7642
Epoch 34/100
123/123 [=====] - ETA: 0s - loss: 0.1998 - accuracy:
0.8923Restoring model weights from the end of the best epoch: 24.
123/123 [=====] - 24s 200ms/step - loss: 0.1998 -
accuracy: 0.8923 - val_loss: 0.7977 - val_accuracy: 0.7594
Epoch 34: early stopping
Wall time: 17min 39s

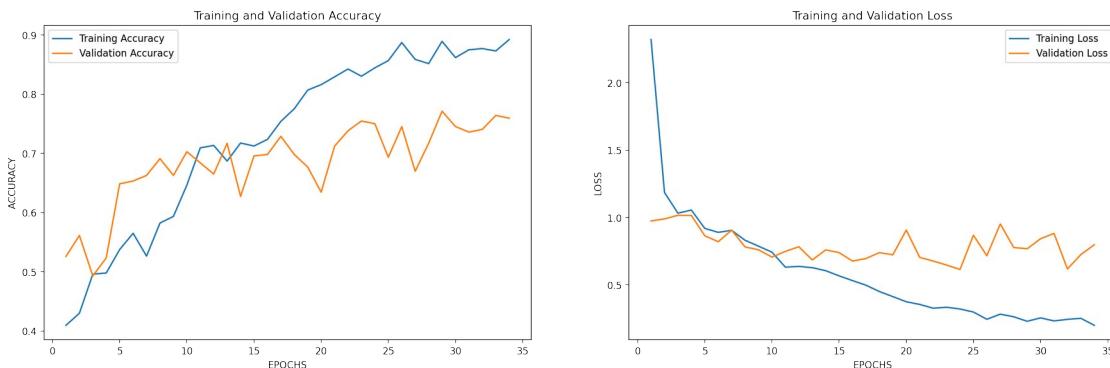
```

```

# Store history
utils.store_model_history(model_name, history.history)

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob =
kneeMRI_Model_TF_3_ResNet.predict(utils.predict_batch_generator_tf_3(X_test,
BATCH_SIZE))

X_test_pred = X_test_prob.argmax(axis=-1)

```

```
utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])
```

```
20/20 [=====] - 9s 456ms/step
```

Evaluation Metrics:

Balanced Accuracy : 0.75

Precision : 0.76

Recall : 0.76

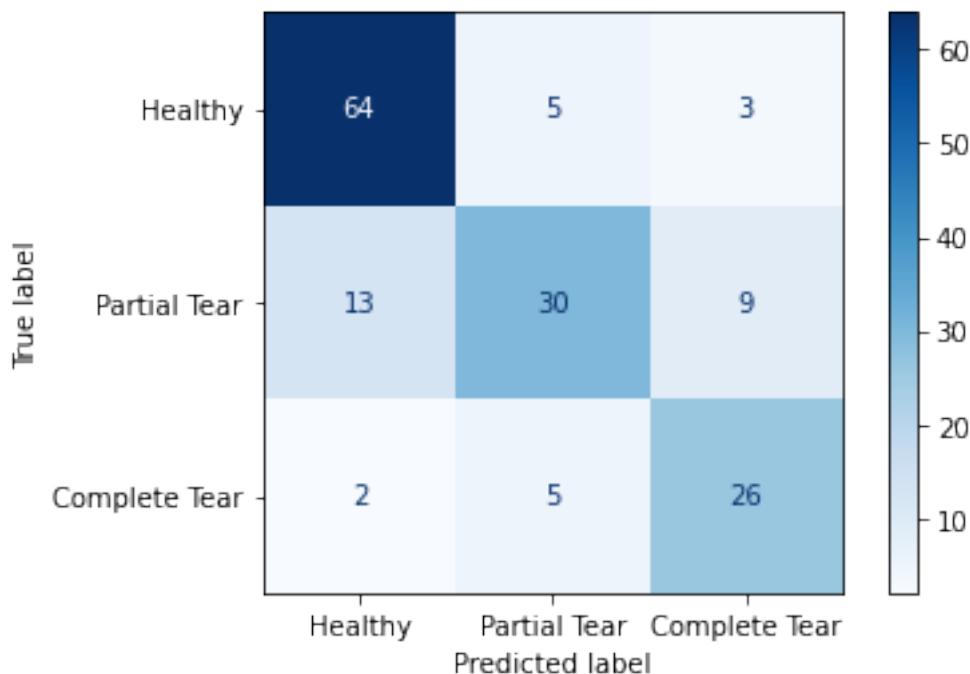
F1 Score: 0.76

ROC AUC Score : 0.9

Classification report :

	precision	recall	f1-score	support
Healthy	0.81	0.89	0.85	72
Partial Tear	0.75	0.58	0.65	52
Complete Tear	0.68	0.79	0.73	33
accuracy			0.76	157
macro avg	0.75	0.75	0.74	157
weighted avg	0.76	0.76	0.76	157

Confusion Matrix :



```
model_name = 'kneeMRI_Model_TF_5_VGG'
kneeMRI_Model_TF_5_VGG = models.mri_model_tf_5_vgg(model_name,
len(kneemri_classes))
kneeMRI_Model_TF_5_VGG.compile(optimizer=keras.optimizers.Adam(learning_rate=u
```

```

    tils.model_lr_schedule(),
                           loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])
kneeMRI_Model_TF_5_VGG.summary()

```

Model: "kneeMRI_Model_TF_5_VGG"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14715840
flatten_2 (Flatten)	(None, 32768)	0
dense_8 (Dense)	(None, 512)	16777728
dropout_6 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 256)	131328
dropout_7 (Dropout)	(None, 256)	0
dense_10 (Dense)	(None, 128)	32896
dropout_8 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 3)	387

Total params: 31,658,179
Trainable params: 16,942,339
Non-trainable params: 14,715,840

```

%%time
with tf.device('/device:GPU:0'):
    history = kneeMRI_Model_TF_5_VGG.fit(utils.batch_generator_tf_5(X_train,
y_train, BATCH_SIZE),
                                         steps_per_epoch=len(X_train)//BATCH_SIZE,
                                         epochs=EP0CHS,
                                         validation_data=utils.batch_generator_tf_5(X_valid, y_valid, BATCH_SIZE),
                                         validation_steps=len(X_valid)//BATCH_SIZE,
                                         shuffle=True,
                                         class_weight=kneemri_class_weights,
                                         verbose=1,
                                         callbacks=[utils.model_callback_checkpoint(model_name),
                                         utils.model_callback_earlystopping()])

```

Epoch 1/100
123/123 [=====] - 40s 319ms/step - loss: 1.2835 -

accuracy: 0.4228 - val_loss: 0.9565 - val_accuracy: 0.4835
Epoch 2/100
123/123 [=====] - 32s 257ms/step - loss: 1.0728 -
accuracy: 0.4848 - val_loss: 0.8608 - val_accuracy: 0.5920
Epoch 3/100
123/123 [=====] - 31s 255ms/step - loss: 0.9487 -
accuracy: 0.5549 - val_loss: 0.8487 - val_accuracy: 0.6085
Epoch 4/100
123/123 [=====] - 32s 260ms/step - loss: 0.8795 -
accuracy: 0.5650 - val_loss: 0.8324 - val_accuracy: 0.6203
Epoch 5/100
123/123 [=====] - 31s 251ms/step - loss: 0.7792 -
accuracy: 0.6433 - val_loss: 0.7920 - val_accuracy: 0.6156
Epoch 6/100
123/123 [=====] - 30s 248ms/step - loss: 0.7481 -
accuracy: 0.6616 - val_loss: 0.7407 - val_accuracy: 0.6792
Epoch 7/100
123/123 [=====] - 30s 249ms/step - loss: 0.7137 -
accuracy: 0.6717 - val_loss: 0.7174 - val_accuracy: 0.7052
Epoch 8/100
123/123 [=====] - 31s 251ms/step - loss: 0.6254 -
accuracy: 0.7327 - val_loss: 0.6921 - val_accuracy: 0.7123
Epoch 9/100
123/123 [=====] - 31s 250ms/step - loss: 0.6085 -
accuracy: 0.7307 - val_loss: 0.6674 - val_accuracy: 0.7217
Epoch 10/100
123/123 [=====] - 30s 247ms/step - loss: 0.5396 -
accuracy: 0.7795 - val_loss: 0.6384 - val_accuracy: 0.7476
Epoch 11/100
123/123 [=====] - 30s 244ms/step - loss: 0.4619 -
accuracy: 0.7957 - val_loss: 0.7020 - val_accuracy: 0.7005
Epoch 12/100
123/123 [=====] - 30s 244ms/step - loss: 0.4226 -
accuracy: 0.8262 - val_loss: 0.6757 - val_accuracy: 0.7264
Epoch 13/100
123/123 [=====] - 29s 241ms/step - loss: 0.3889 -
accuracy: 0.8374 - val_loss: 0.6922 - val_accuracy: 0.7075
Epoch 14/100
123/123 [=====] - 25s 204ms/step - loss: 0.3707 -
accuracy: 0.8465 - val_loss: 0.6792 - val_accuracy: 0.7099
Epoch 15/100
123/123 [=====] - 22s 180ms/step - loss: 0.3204 -
accuracy: 0.8618 - val_loss: 0.6767 - val_accuracy: 0.7382
Epoch 16/100
123/123 [=====] - 22s 177ms/step - loss: 0.2823 -
accuracy: 0.8882 - val_loss: 0.7011 - val_accuracy: 0.7099
Epoch 17/100
123/123 [=====] - 22s 183ms/step - loss: 0.2390 -
accuracy: 0.9024 - val_loss: 0.7061 - val_accuracy: 0.7476
Epoch 18/100
123/123 [=====] - 22s 183ms/step - loss: 0.1988 -
accuracy: 0.9116 - val_loss: 0.6523 - val_accuracy: 0.7358

```

Epoch 19/100
123/123 [=====] - 22s 183ms/step - loss: 0.1887 -
accuracy: 0.9228 - val_loss: 0.7550 - val_accuracy: 0.7358
Epoch 20/100
123/123 [=====] - ETA: 0s - loss: 0.1691 - accuracy:
0.9370Restoring model weights from the end of the best epoch: 10.
123/123 [=====] - 22s 181ms/step - loss: 0.1691 -
accuracy: 0.9370 - val_loss: 1.0105 - val_accuracy: 0.6745
Epoch 20: early stopping
Wall time: 9min 24s

```

```

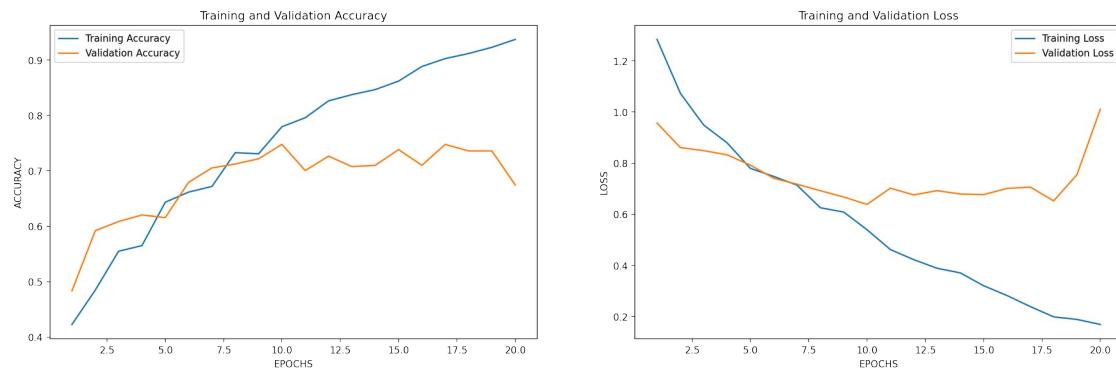
# Store history
utils.store_model_history(model_name, history.history)

```

```

# Plot training graphs
utils.plot_acc_loss(history.history)

```



```

# Evaluate model
X_test_prob =
kneeMRI_Model_TF_5_VGG.predict(utils.predict_batch_generator_tf_5(X_test,
BATCH_SIZE))

X_test_pred = X_test_prob.argmax(axis=-1)

utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])

20/20 [=====] - 3s 143ms/step

```

Evaluation Metrics:

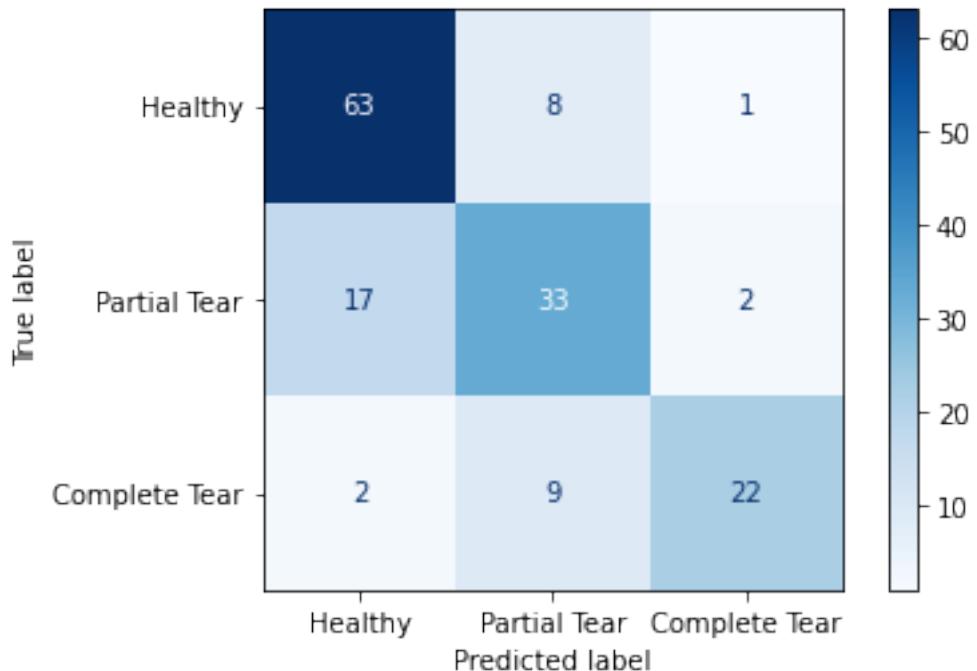
```

Balanced Accuracy : 0.73
Precision : 0.76
Recall : 0.75
F1 Score: 0.75
ROC AUC Score : 0.91
Classification report :
      precision    recall   f1-score   support

```

<i>Healthy</i>	0.77	0.88	0.82	72
<i>Partial Tear</i>	0.66	0.63	0.65	52
<i>Complete Tear</i>	0.88	0.67	0.76	33
<i>accuracy</i>			0.75	157
<i>macro avg</i>	0.77	0.73	0.74	157
<i>weighted avg</i>	0.76	0.75	0.75	157

Confusion Matrix :



```

model_name = 'kneeMRI_Model_TF_5_ResNet'
kneeMRI_Model_TF_5_ResNet = models.mri_model_tf_5_resnet(model_name,
len(kneemri_classes))
kneeMRI_Model_TF_5_ResNet.compile(optimizer=keras.optimizers.Adam(learning_rate=utils.model_lr_schedule()),
                                    loss='sparse_categorical_crossentropy',
                                    metrics=['accuracy'])
kneeMRI_Model_TF_5_ResNet.summary()

```

Model: "kneeMRI_Model_TF_5_ResNet"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 8, 8, 2048)	23593984
flatten_1 (Flatten)	(None, 131072)	0
dense_4 (Dense)	(None, 512)	67109376
dropout_3 (Dropout)	(None, 512)	0

<i>dense_5</i> (<i>Dense</i>)	(<i>None</i> , 256)	131328
<i>dropout_4</i> (<i>Dropout</i>)	(<i>None</i> , 256)	0
<i>dense_6</i> (<i>Dense</i>)	(<i>None</i> , 128)	32896
<i>dropout_5</i> (<i>Dropout</i>)	(<i>None</i> , 128)	0
<i>dense_7</i> (<i>Dense</i>)	(<i>None</i> , 3)	387

=====

Total params: 90,867,971

Trainable params: 67,273,987

Non-trainable params: 23,593,984

```
%%time
with tf.device('/device:GPU:0'):
    history =
kneeMRI_Model_TF_5_ResNet.fit(utils.batch_generator_tf_5(X_train, y_train,
BATCH_SIZE),
                                steps_per_epoch=len(X_train)//BATCH_SIZE,
                                epochs=EPOCHS,
validation_data=utils.batch_generator_tf_5(X_valid, y_valid, BATCH_SIZE),
validation_steps=len(X_valid)//BATCH_SIZE,
                                shuffle=True,
                                class_weight=kneemri_class_weights,
                                verbose=1,
callbacks=[utils.model_callback_checkpoint(model_name),
utils.model_callback_earlystopping()])
Epoch 1/100
123/123 [=====] - 42s 326ms/step - loss: 2.6507 -
accuracy: 0.3618 - val_loss: 1.2343 - val_accuracy: 0.3608
Epoch 2/100
123/123 [=====] - 31s 256ms/step - loss: 1.4225 -
accuracy: 0.4522 - val_loss: 0.9574 - val_accuracy: 0.5755
Epoch 3/100
123/123 [=====] - 29s 240ms/step - loss: 1.0795 -
accuracy: 0.4837 - val_loss: 1.0093 - val_accuracy: 0.5330
Epoch 4/100
123/123 [=====] - 29s 234ms/step - loss: 1.0076 -
accuracy: 0.5041 - val_loss: 0.9891 - val_accuracy: 0.5778
Epoch 5/100
123/123 [=====] - 30s 242ms/step - loss: 0.9375 -
accuracy: 0.5295 - val_loss: 0.9359 - val_accuracy: 0.6061
Epoch 6/100
123/123 [=====] - 28s 230ms/step - loss: 0.9000 -
accuracy: 0.5518 - val_loss: 0.9368 - val_accuracy: 0.5920
```

Epoch 7/100
123/123 [=====] - 28s 227ms/step - loss: 0.8617 -
accuracy: 0.5793 - val_loss: 0.9401 - val_accuracy: 0.5991
Epoch 8/100
123/123 [=====] - 27s 222ms/step - loss: 0.7961 -
accuracy: 0.6108 - val_loss: 1.0023 - val_accuracy: 0.5307
Epoch 9/100
123/123 [=====] - 28s 233ms/step - loss: 0.7660 -
accuracy: 0.6148 - val_loss: 0.8724 - val_accuracy: 0.6085
Epoch 10/100
123/123 [=====] - 30s 248ms/step - loss: 0.7131 -
accuracy: 0.6596 - val_loss: 0.8516 - val_accuracy: 0.6014
Epoch 11/100
123/123 [=====] - 28s 227ms/step - loss: 0.6934 -
accuracy: 0.6616 - val_loss: 0.8559 - val_accuracy: 0.6344
Epoch 12/100
123/123 [=====] - 27s 220ms/step - loss: 0.6909 -
accuracy: 0.6728 - val_loss: 1.0012 - val_accuracy: 0.5000
Epoch 13/100
123/123 [=====] - 29s 235ms/step - loss: 0.6379 -
accuracy: 0.6778 - val_loss: 0.8188 - val_accuracy: 0.6604
Epoch 14/100
123/123 [=====] - 30s 243ms/step - loss: 0.5675 -
accuracy: 0.7348 - val_loss: 0.7728 - val_accuracy: 0.6816
Epoch 15/100
123/123 [=====] - 27s 222ms/step - loss: 0.5438 -
accuracy: 0.7449 - val_loss: 0.9186 - val_accuracy: 0.6415
Epoch 16/100
123/123 [=====] - 26s 216ms/step - loss: 0.5011 -
accuracy: 0.7571 - val_loss: 0.9209 - val_accuracy: 0.5590
Epoch 17/100
123/123 [=====] - 29s 235ms/step - loss: 0.4733 -
accuracy: 0.7663 - val_loss: 0.7175 - val_accuracy: 0.6934
Epoch 18/100
123/123 [=====] - 27s 225ms/step - loss: 0.4474 -
accuracy: 0.7967 - val_loss: 0.9008 - val_accuracy: 0.6014
Epoch 19/100
123/123 [=====] - 29s 234ms/step - loss: 0.4834 -
accuracy: 0.7724 - val_loss: 0.7090 - val_accuracy: 0.7075
Epoch 20/100
123/123 [=====] - 27s 223ms/step - loss: 0.3749 -
accuracy: 0.8283 - val_loss: 0.7214 - val_accuracy: 0.7146
Epoch 21/100
123/123 [=====] - 27s 217ms/step - loss: 0.3368 -
accuracy: 0.8425 - val_loss: 0.8529 - val_accuracy: 0.6887
Epoch 22/100
123/123 [=====] - 26s 211ms/step - loss: 0.3577 -
accuracy: 0.8445 - val_loss: 0.7835 - val_accuracy: 0.6604
Epoch 23/100
123/123 [=====] - 25s 207ms/step - loss: 0.2988 -
accuracy: 0.8537 - val_loss: 0.7362 - val_accuracy: 0.7193
Epoch 24/100

```

123/123 [=====] - 25s 207ms/step - loss: 0.3116 -
accuracy: 0.8486 - val_loss: 0.7129 - val_accuracy: 0.7099
Epoch 25/100
123/123 [=====] - 25s 208ms/step - loss: 0.2695 -
accuracy: 0.8720 - val_loss: 0.8552 - val_accuracy: 0.6887
Epoch 26/100
123/123 [=====] - 26s 210ms/step - loss: 0.3581 -
accuracy: 0.8293 - val_loss: 0.7661 - val_accuracy: 0.6745
Epoch 27/100
123/123 [=====] - 25s 207ms/step - loss: 0.4065 -
accuracy: 0.8120 - val_loss: 0.7491 - val_accuracy: 0.7170
Epoch 28/100
123/123 [=====] - 25s 207ms/step - loss: 0.3796 -
accuracy: 0.8171 - val_loss: 0.8289 - val_accuracy: 0.6910
Epoch 29/100
123/123 [=====] - ETA: 0s - loss: 0.3421 - accuracy:
0.8404Restoring model weights from the end of the best epoch: 19.
123/123 [=====] - 26s 211ms/step - loss: 0.3421 -
accuracy: 0.8404 - val_loss: 0.8063 - val_accuracy: 0.7193
Epoch 29: early stopping
Wall time: 13min 32s

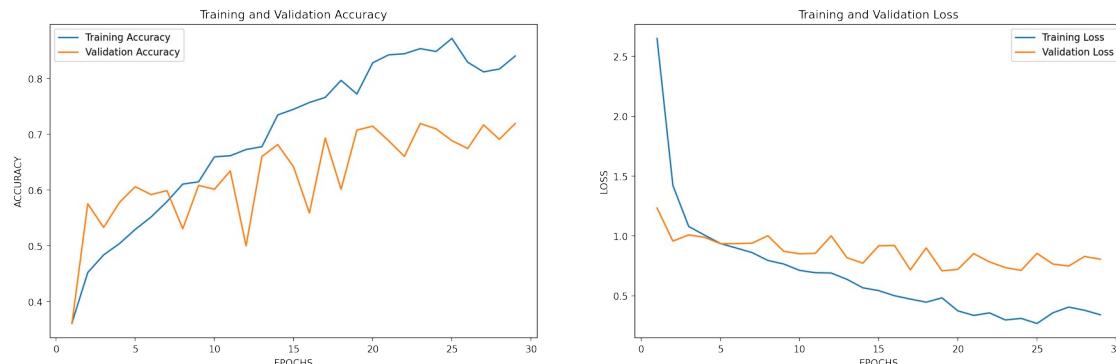
```

```
# Store history
```

```
utils.store_model_history(model_name, history.history)
```

```
# Plot training graphs
```

```
utils.plot_acc_loss(history.history)
```



```
# Evaluate model
```

```
X_test_prob =
kneeMRI_Model_TF_5_ResNet.predict(utils.predict_batch_generator_tf_5(X_test,
BATCH_SIZE))
```

```
X_test_pred = X_test_prob.argmax(axis=-1)
```

```
utils.evaluate_model(y_test, X_test_pred.tolist(), X_test_prob.tolist(),
['Healthy', 'Partial Tear', 'Complete Tear'])
```

```
20/20 [=====] - 5s 208ms/step
```

Evaluation Metrics:

Balanced Accuracy : 0.76

Precision : 0.78

Recall : 0.78

F1 Score: 0.77

ROC AUC Score : 0.89

Classification report :

	precision	recall	f1-score	support
Healthy	0.76	0.92	0.83	72
Partial Tear	0.82	0.60	0.69	52
Complete Tear	0.78	0.76	0.77	33
accuracy			0.78	157
macro avg	0.79	0.76	0.76	157
weighted avg	0.78	0.78	0.77	157

Confusion Matrix :

