Francisco João Gonçalves Santana

OPHTHALMOLOGY APPLICATIONS OF FEDERATED LEARNING

Dissertação no âmbito do Ramo de Computadores do Mestrado de Engenharia Eletrotécnica e de Computadores orientada pelo Professor Luís Alberto da Silva Cruz, co-orientada pelo Dr. Pedro Pereira (Retmarker) e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2023

· U C ·

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Ophthalmology Applications of Federated Learning

Francisco João Gonçalves Santana

Coimbra, September 2023

Ophthalmology Applications of Federated Learning

Supervisor:

Luís Alberto da Silva Cruz

Co-Supervisor:

Pedro Pereira

Jury:

Jorge Miguel Sá Silva

Luís Alberto da Silva Cruz

Nuno Miguel Mendonça da Silva Gonçalves

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, September 2023

Acknowledgements

I would like to thank my supervisor, Dr. Luís Cruz, for his continuous support and for pushing me in the right direction during the times when I was a little lost. I would also like to thank the members of Retmarker for taking the time to evaluate my work and for the tips they gave me to improve.

I would also like to thank all the friends I made at the University of Coimbra and those I knew before this journey. It was a pleasure growing up next to all of you. You all have made this part of my life pleasant and much easier to bear.

And last but not least, I would like to thank my family for their unwavering love and support. My mother, who always cared about my welfare and academic success. My sister, who always made sure that I gave my all while keeping it fun. All my uncles and aunts for always showing interest in how well I was doing and for giving me courage to keep going.

Abstract

Diabetic Retinopathy, a common diabetic consequence, requires early detection and treatment to avoid vision damage. Manual detection of Diabetic Retinopathy from medical photos is a time consuming and resource intensive technique. Machine Learning, particularly Deep Learning, appears to be a promising option for automating Diabetic Retinopathy detection in order to overcome this difficulty.

The collection of big medical picture datasets for model training, on the other hand, raises serious concerns about patient data privacy. Federated Learning emerges as a viable approach for model training over decentralized datasets while maintaining patient privacy.

This study begins an in-depth investigation of Federated Learning for Diabetic Retinopathy detection. It includes training local reference models on two distinct public datasets, as well as the employment of three aggregation algorithms: Federated Averaging, Federated Proximal, and SCAFFOLD. Two scenarios are simulated: one in which clients use their whole dataset in each communication round, and another in which datasets are partitioned into subsets to reflect real-world resource limits.

The implemented aggregation algorithms, notably SCAFFOLD, demonstrate enhanced convergence and accuracy. The scenario where datasets are divided into subsets exhibits the feasibility of dynamic data updates without compromising model performance.

This study combines Deep Learning, privacy preservation, and decentralized training to advance automated Diabetic Retinopathy detection while protecting patient privacy. It discusses all of the obtained results and lays the groundwork for future research into Federated Learning techniques and their larger applications in medical diagnosis.

Keywords: Diabetic Retinopathy, Deep Learning, Federated Learning, Privacy-Preserving

Resumo

A Retinopatia Diabética, uma consequência comum da diabetes, requer deteção e tratamento precoces para evitar danos na visão. A deteção manual da Retinopatia Diabética a partir de fotografias médicas é uma técnica que consome muito tempo e recursos. A Aprendizagem Computacional, em particular a Aprendizagem Profunda, aparenta ser uma opção promissora para automatizar a deteção da Retinopatia Diabética, de forma a ultrapassar esta dificuldade.

Por outro lado, a recolha de grandes conjuntos de dados de imagens médicas para o treino de modelos cria sérias preocupações quanto à privacidade dos dados dos pacientes. A Aprendizagem Federada surge como uma abordagem viável para o treino de modelos em conjuntos de dados descentralizados, enquanto mantém a privacidade dos pacientes.

Este estudo realiza uma investigação aprofundada da Aprendizagem Federada para a deteção da Retinopatia Diabética. Inclui o treino de modelos de referência locais em dois conjuntos de dados públicos distintos, assim como o uso de três algoritmos de agregação: Federated Averaging, Federated Proximal e SCAFFOLD. São simulados dois cenários: um em que os clientes utilizam todo o seu conjunto de dados em cada ronda de comunicação e outro em que os conjuntos de dados são divididos em subconjuntos para refletir os limites de recursos do mundo real.

Os algoritmos de agregação implementados, nomeadamente o SCAFFOLD, demonstram uma maior convergência e precisão. O cenário em que os conjuntos de dados são divididos em subconjuntos exibe a viabilidade de atualizações dinâmicas de dados sem comprometer o desempenho do modelo.

Este estudo combina Aprendizagem Profunda, preservação de privacidade e treino descentralizado para avançar na deteção automática de Retinopatia Diabética, enquanto garante a privacidade do paciente. São discutidos todos os resultados obtidos e são estabelecidas as bases para trabalho futuro sobre técnicas de Aprendizagem Federada e as suas aplicações mais no diagnóstico médico.

Palavras chave: Retinopatia Diabética, Aprendizagem Profunda, Aprendizagem Federada, Preservação da Privacidade

“Our greatest glory is not in never falling, but in rising every time we fall."

— Confucius

Contents

Acknowledgements **ii**

Abstract **iv**

Resumo **vi**

List of Acronyms **xiii**

List of Figures **xiv**

List of Tables **xviii**

1 Introduction **1**

1.1 Context and Motivation 1

1.2 Objectives 3

1.3 Dissertation Outline 3

2 Background Information **5**

2.1 Diabetic Retinopathy Detection 6

2.2 Machine Learning 7

2.2.1 Supervised Learning 8

2.2.2 Convolutional Neural Networks 9

2.3 Federated Learning 13

2.3.1 Problem Formulation 14

2.3.2 Core Challenges 15

2.3.3 Types of Federated Learning 15

2.3.4 Aggregation Algorithms 17

2.4 Related Work 18

2.5 Software and Datasets 22

3 Federated Learning Applied to Diabetic Retinopathy Detection **24**

3.1 Reference Models Trained Locally 24

3.1.1 Model Selection and Implementation 24
3.1.2 Improving the Model Classification 26
3.1.3 Solutions for Imbalanced Datasets 28
3.1.4 Binary Classification and Training the model on the Kaggle DR Dataset . . . 30
3.2 Federated Learning Trained Models 32
3.2.1 Datasets Partition 34
3.2.2 Aggregation Algorithms 37

4 Experimental Results **40**
4.1 Reference Models Results 40
4.2 Federated Learning Results 42
4.3 Discussion 48

5 Conclusion and Future Work **50**

A Results Obtained Improving the Local Model **57**

B Other Federated Learning Results **65**

List of Acronyms

Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
AMD	Age-related Macular Degeneration
APFL	Adaptive Personalized Federated Learning
BN	Batch Normalization
BS	Batch Size
CE	Cross Entropy
CLAHE	Contrast Limited Adaptive Histogram Equalization
CMI	Conditional Mutual Information
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
DA	Domain Alignment
DL	Deep Learning
DR	Diabetic Retinopathy
FCL	Fully Connected Layer
FedAvg	Federated Averaging
FedMA	Federated Matched Averaging
FedProx	Federated Proximal
FedSGD	Federated Stochastic Gradient Descent
FL	Federated Learning

FTL Federated Transfer Learning

GA Genetics Algorithm

HFL Horizontal Federated Learning

HIPAA Health Insurance Portability and Accountability Act of 1996

IDF International Diabetes Federation

LR Learning Rate

MAE Mean Absolute Error

ML Machine Learning

MSE Mean Square Error

NPDR Non-Proliferative Diabetic Retinopathy

NRDR Non-Referable Diabetic Retinopathy

OCT Optical Coherence Tomography

OCTA Optical Coherence Tomography Angiography

PDR Proliferative Diabetic Retinopathy

RDR Referable Diabetic Retinopathy

ReLU Rectified Linear Unit

ROP Retinopathy of Prematurity

SCAFFOLD Stochastic Controlled Averaging for Federated Learning

SGD Stochastic Gradient Descent

SL Supervised Learning

Tanh Hyperbolic Tangent

TL Transfer Learning

VFL Vertical Federated Learning

VRAM Video Random Access Memory

List of Figures

1.1 Worldwide prevalence of Diabetes by IDF. [2] 1

2.1 Fundus photograph of the retina showing some anatomical structures. [12] 5

2.2 Lesions caused by DR. [18] 7

2.3 Architecture of a popular CNN - ResNet50. [19] 8

2.4 Convolution operation. [21] 9

2.5 Results from Max Pooling and Average Pooling using a 2×2 filter with stride 2. [22] 10

2.6 Sigmoid, Tanh and ReLU activation functions. [23] 11

2.7 Example of a FCL with the input layer, two dense layers and the output layer. [24] . 12

2.8 FCL with and without dropout. [25] 13

2.9 Illustration of how Centralized Learning works. [26] 14

2.10 Illustration of how FL works. [27] 14

2.11 Types of FL. [32] 16

2.12 Modified AlexNet architecture. [39] 19

2.13 Best architectures for each of the datasets, a) Messidor and b) Kaggle DR. [41] . . . 20

2.14 Model transferring between server and client through cloud-based drop-off folders. [43] 21

3.1 Comparison of the same retina image before and after being cropped and resized. . . 25

3.2 Same retina image before and after applying CLAHE. 28

3.3 Oversampling illustrated. 30

3.4 Class conversion for MESSIDOR dataset. 31

3.5 Class conversion for Kaggle DR and APTOS datasets. 31

3.6 Implemented FL framework diagram. 34

3.7 MESSIDOR divided between clients. 35

3.8 MESSIDOR divided by subsets. 36

3.9 Kaggle DR divided between clients. 36

3.10 Kaggle DR divided by subsets. 37

4.1 ROC curve for four classes classification. 40

4.2 ROC curve for three classes classification. 41

4.3 ROC curve for two classes classification 41
4.4 ROC curve for training on the MESSIDOR dataset. 42
4.5 ROC curve for training on the Kaggle DR dataset. 42
4.6 Accuracy obtained at every communication round using the FedAvg algorithm on the MESSIDOR dataset. 43
4.7 Accuracy obtained at every communication round using the FedProx algorithm on the MESSIDOR dataset. 43
4.8 Accuracy obtained at every communication round using the SCAFFOLD algorithm on the MESSIDOR dataset. 44
4.9 Accuracy obtained at every communication round using the FedAvg algorithm on the Kaggle Diabetic Retinopathy (DR) dataset. 44
4.10 Accuracy obtained at every communication round using the FedProx algorithm on the Kaggle DR dataset. 45
4.11 Accuracy obtained at every communication round using the SCAFFOLD algorithm on the Kaggle DR dataset. 45
4.12 Accuracy obtained at every communication round using the FedAvg algorithm on the MESSIDOR dataset divided in subsets. 46
4.13 Accuracy obtained at every communication round using the FedProx algorithm on the MESSIDOR dataset divided in subsets. 46
4.14 Accuracy obtained at every communication round using the SCAFFOLD algorithm on the MESSIDOR dataset divided in subsets. 47
4.15 Accuracy obtained at every communication round using the FedAvg algorithm on the Kaggle DR dataset divided in subsets. 47
4.16 Accuracy obtained at every communication round using the FedProx algorithm on the Kaggle DR dataset divided in subsets. 48
4.17 Accuracy obtained at every communication round using the SCAFFOLD algorithm on the Kaggle DR dataset divided in subsets. 48

A.1 ROC curve for four classes classification after changing the loss function to CE. . . . 57
A.2 ROC curve for four classes classification with BS = 2. 58
A.3 ROC curve for four classes classification with BS = 8. 58
A.4 ROC curve for four classes classification training with the full architecture. 59
A.5 ROC curve for four classes classification with Adam optimizer and LR = 0.0001. . . 59
A.6 ROC curve for four classes classification with increased data augmentation. 60
A.7 ROC curve for four classes classification with an added dropout layer. 60
A.8 ROC curve for four classes classification with CLAHE added to the preprocessing stage. 61

A.9 ROC curve for four classes classification with Class Weighting. 61
A.10 ROC curve for four classes classification with Oversampling. 62
A.11 ROC curve for four classes classification with Focal Loss $\gamma = 2$. 62
A.12 ROC curve for four classes classification with Focal Loss $\gamma = 2.5$. 63
A.13 ROC curve for four classes classification with Focal Loss $\gamma = 3$. 63
A.14 ROC curve for four classes classification with Focal Loss $\gamma = 3.5$. 64
A.15 ROC curve for four classes classification with Focal Loss $\gamma = 4$. 64

B.1 Accuracy obtained at every communication round using the FedAvg algorithm on the MESSIDOR dataset and testing on the Kaggle DR dataset. 65
B.2 Accuracy obtained at every communication round using the FedProx algorithm on the MESSIDOR dataset and testing on the Kaggle DR dataset. 66
B.3 Accuracy obtained at every communication round using the SCAFFOLD algorithm on the MESSIDOR dataset and testing on the Kaggle DR dataset. 66
B.4 Accuracy obtained at every communication round using the FedAvg algorithm on the Kaggle DR dataset and testing on the MESSIDOR dataset. 67
B.5 Accuracy obtained at every communication round using the FedProx algorithm on the Kaggle DR dataset and testing on the MESSIDOR dataset. 67
B.6 Accuracy obtained at every communication round using the SCAFFOLD algorithm on the Kaggle DR dataset and testing on the MESSIDOR dataset. 68
B.7 Accuracy obtained at every communication round using the FedAvg algorithm on the MESSIDOR dataset divided in subsets and testing on the Kaggle DR dataset 68
B.8 Accuracy obtained at every communication round using the FedProx algorithm on the MESSIDOR dataset divided in subsets and testing on the Kaggle DR dataset. . 69
B.9 Accuracy obtained at every communication round using the SCAFFOLD algorithm on the MESSIDOR dataset divided in subsets and testing on the Kaggle DR dataset. 69
B.10 Accuracy obtained at every communication round using the FedAvg algorithm on the Kaggle DR dataset divided in subsets and testing on the MESSIDOR dataset. 70
B.11 Accuracy obtained at every communication round using the FedProx algorithm on the Kaggle DR dataset divided in subsets and testing on the MESSIDOR dataset. . 70
B.12 Accuracy obtained at every communication round using the SCAFFOLD algorithm on the Kaggle DR dataset divided in subsets and testing on the MESSIDOR dataset. 71

List of Tables

2.1 Datasets used. 22

2.2 Description of the MESSIDOR Dataset. 23

2.3 Description of the Kaggle DR Dataset. 23

2.4 Description of the APTOS Dataset. 23

3.1 MESSIDOR's image distribution for binary classification. 31

3.2 Kaggle DR's image distribution for binary classification. 31

3.3 APTOS's image distribution for binary classification. 32

4.1 Accuracy obtained for each of the four classes. 40

4.2 Accuracy obtained for each of the three classes. 41

4.3 Accuracy obtained for class 0 and 3. 41

4.4 Specificity and sensitivity obtained when training on the MESSIDOR dataset. 42

4.5 Specificity and sensitivity obtained when training on the Kaggle DR dataset. 42

4.6 Specificity and sensitivity obtained when training on the MESSIDOR dataset using the FedAvg algorithm. 43

4.7 Specificity and sensitivity obtained when training on the MESSIDOR dataset using the FedProx algorithm. 43

4.8 Specificity and sensitivity obtained when training on the MESSIDOR dataset using the SCAFFOLD algorithm. 44

4.9 Specificity and sensitivity obtained when training on the Kaggle DR dataset using the FedAvg algorithm. 44

4.10 Specificity and sensitivity obtained when training on the Kaggle DR dataset using the FedProx algorithm. 45

4.11 Specificity and sensitivity obtained when training on the Kaggle DR dataset using the SCAFFOLD algorithm. 45

4.12 Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the FedAvg algorithm. 46

4.13 Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the FedProx algorithm. 46
4.14 Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the SCAFFOLD algorithm. 47
4.15 Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the FedAvg algorithm. 47
4.16 Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the FedProx algorithm. 48
4.17 Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the SCAFFOLD algorithm. 48

A.1 Accuracy obtained for each of the four classes after changing the loss function to CE. 57
A.2 Accuracy obtained for each of the four classes with BS = 2. 58
A.3 Accuracy obtained for each of the four classes with BS = 8. 58
A.4 Accuracy obtained for each of the four classes training with the full architecture. . . 59
A.5 Accuracy obtained for each of the four classes with Adam optimizer and LR = 0.0001. 59
A.6 Accuracy obtained for each of the four classes with increased data augmentation. . . 60
A.7 Accuracy obtained for each of the four classes with an added dropout layer. 60
A.8 Accuracy obtained for each of the four classes with CLAHE added to the preprocessing stage. 61
A.9 Accuracy obtained for each of the four classes with Class Weighting. 61
A.10 Accuracy obtained for each of the four classes with Oversampling. 62
A.11 Accuracy obtained for each of the four classes with Focal Loss $\gamma = 2$. 62
A.12 Accuracy obtained for each of the four classes with Focal Loss $\gamma = 2.5$. 63
A.13 Accuracy obtained for each of the four classes with Focal Loss $\gamma = 3$. 63
A.14 Accuracy obtained for each of the four classes with Focal Loss $\gamma = 3.5$. 64
A.15 Accuracy obtained for each of the four classes with Focal Loss $\gamma = 4$. 64

B.1 Specificity and sensitivity obtained when training on the MESSIDOR dataset using the FedAvg algorithm and testing on the Kaggle DR dataset. 65
B.2 Specificity and sensitivity obtained when training on the MESSIDOR dataset using the FedProx algorithm and testing on the Kaggle DR dataset. 66
B.3 Specificity and sensitivity obtained when training on the MESSIDOR dataset using the SCAFFOLD algorithm and testing on the Kaggle DR dataset. 66
B.4 Specificity and sensitivity obtained when training on the Kaggle DR dataset using the FedAvg algorithm and testing on the MESSIDOR dataset. 67

B.5 Specificity and sensitivity obtained when training on the Kaggle DR dataset using the FedProx algorithm and testing on the MESSIDOR dataset. 67
B.6 Specificity and sensitivity obtained when training on the Kaggle DR dataset using the SCAFFOLD algorithm and testing on the MESSIDOR dataset. 68
B.7 Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the FedAvg algorithm and testing on the Kaggle DR dataset. 68
B.8 Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the FedProx algorithm and testing on the Kaggle DR dataset. 69
B.9 Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the SCAFFOLD algorithm and testing on the Kaggle DR dataset. . 69
B.10 Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the FedAvg algorithm and testing on the MESSIDOR dataset. . . . 70
B.11 Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the FedProx algorithm and testing on the MESSIDOR dataset. . . . 70
B.12 Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the SCAFFOLD algorithm and testing on the MESSIDOR dataset. . 71

1 Introduction

1.1 Context and Motivation

Diabetes mellitus is a disease characterised by high glucose (sugar) levels in the human body. Diabetes occurs when the pancreas does not produce enough or any insulin at all, or when the body does not respond properly to the effects of insulin. It affects people of all ages, and although most forms are chronic, all forms can be managed with medication and/or lifestyle changes [1].

The International Diabetes Federation (IDF) estimates that 537 million people were living with diabetes in 2021 (10.5% of the world's adult population), and projects that there will be 783 million by 2045, a 46% increase from 2021. The IDF also estimates that one in two adults affected by diabetes is unaware of this fact [2].

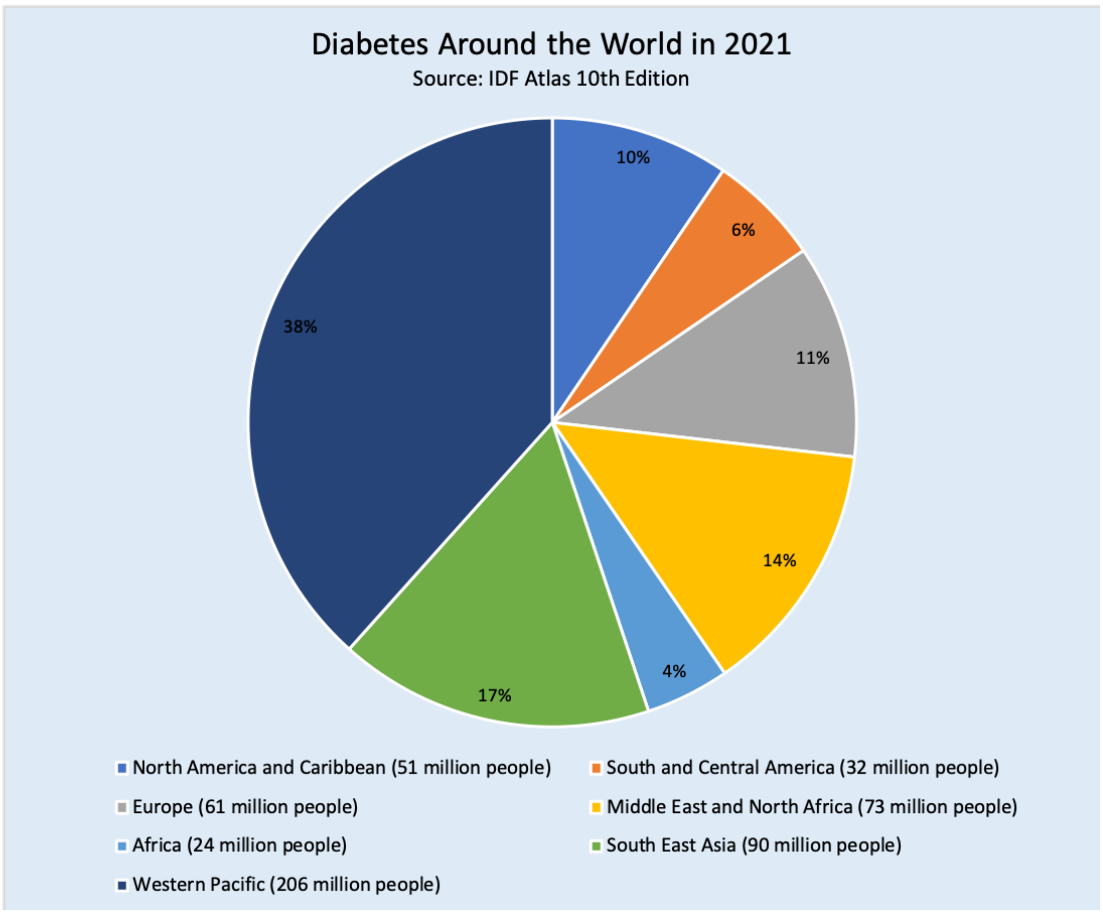


Figure 1.1: Worldwide prevalence of Diabetes by IDF. [2]

Diabetes can lead to numerous complications that can cause serious damage if left uncontrolled and untreated. One of these complications is DR, which affects vision. DR is the most common complication of diabetes mellitus [3] and affects the blood vessels in the retina, the light-sensitive tissue at the back of the eye. This disease can lead to vision loss and even blindness if not detected and treated early. Anyone with type 1 or type 2 diabetes is potentially at risk for developing DR. This risk is greatly increased if the person has had diabetes for a long time and has persistently high blood glucose, blood pressure and cholesterol [4].

As the most common and specific complication of diabetes mellitus, DR is one of the leading causes of preventable blindness in the adult population [5]. According to the Global Burden of Disease Study, DR is the fifth leading cause of blindness and moderate and severe visual impairment in adults 50 years and older. By 2020, DR is estimated to affect 22.27% of people with diabetes mellitus worldwide. This corresponds to approximately 103.13 million adults diagnosed with this eye-related complication [6].

Traditional methods of diagnosing ocular disease rely on clinical assessment and the use of imaging equipment. However, these methods are time-consuming and expensive. Ophthalmology is well suited for the application of Deep Learning (DL) techniques that can analyze eye images such as digital fundus photographs and visual fields. DL has shown promise in automating the screening and diagnosis of common vision-threatening diseases such as DR. Therefore, DL could be a valuable addition to current diagnostic procedures by supporting or even replacing trained human image graders [7, 8].

Although DL models show great promise for high-performance models in ophthalmology, there are still some clinical and technical challenges to be solved. One of them is the limited availability of data for both rare and common ocular diseases. Obtaining a large and diverse data set is critical for training DL models. Moreover, the problem is exacerbated by the homogeneity of the data. To ensure generalization of DL models, it is important to include diversified data, such as images acquired with different image widths, fields of view, and magnifications. In addition, it is important to have a dataset that represents a population with different ethnicities [9].

One possible solution to the problems outlined could be a method called centralized learning, in which data are collected and processed in a single central location [10]. However, this method is a less desirable approach in many situations, particularly due to the fact that centralizing data means that data, which may contain sensitive information, is transferred from multiple sources to be stored in a central location. This raises privacy and security concerns because centralizing all data increases the risk of unauthorized access, data breach, or misuse of the data. A single point of failure or a security breach in the central server can result in a massive compromise of privacy. The Health Insurance Portability and Accountability Act of 1996 (HIPAA) is an instance of regulations which aims to protect the sensitive information of patient by allowing the use of data only with the consent of the patient.

To address these privacy concerns, alternative approaches such as Federated Learning (FL) [11] have been proposed. FL allows models to be trained on data distributed across multiple nodes, such as smartphones or computers, rather than centralized in a single location. This eliminates the need to share data with a central server. In a FL scenario, each node contributes to model training with its own data. First, a central server sends a Machine Learning (ML) model to participating devices, also known as clients. Each device then uses its local data to train the model and update the model parameters. These updated model parameters are then sent back to the central server, where they are aggregated and used to update the global model shared by all clients. This process is usually repeated several times, with each iteration referred to as a round of communication. The goal is to find a model that has stable performance and generalizes well to new data from different nodes, even if the data is not identically distributed.

1.2 Objectives

FL has gained prominence in the health domain in recent years, with recent works showing that models trained using FL can achieve comparable performance to locally trained models. Aligning with Retmarker's work, the goal of this dissertation is to develop a FL framework for the detection of DR in eye fundus images capable of achieving comparable performance to the already available DL solutions. Following this, the objectives of this dissertation are as follows:

- Investigate the state of the art in DL solutions for DR detection.
- Investigate the state of the art in FL applications in ophthalmology.
- Investigate the different model aggregation algorithms reported in the literature.
- Develop a novel FL-based DR detection algorithm that achieves comparable performance to locally trained models.

1.3 Dissertation Outline

This Dissertation is organized as follows:

- **Chapter 1** presents the context, motivation, and objectives of this dissertation.
- **Chapter 2** presents several core topics necessary for the understanding of the structure and performance of the solutions proposed in the following chapter.
- **Chapter 3** proposes a local reference model as well as different FL implementations.

- **Chapter 4** studies the performance with the different implementations.
- **Chapter 5** presents conclusions based on the obtained results and proposes ideas for future work.
- **Annex A** presents the results obtained while improving the local reference model.
- **Annex B** presents other results obtained in a different setting with the FL framework.

2 Background Information

DR is a disease that affects the eyes of people with diabetes. It is caused by damage to the blood vessels in the retina. It usually goes undetected until the later stages of the disease. Early detection is essential to prevent vision damage or even loss. Various imaging techniques are available to detect this disease, including fundus photography, fluorescein angiography, and Optical Coherence Tomography (OCT).

In this work, fundus photography is used as an imaging technique to detect DR. Fundus photography involves taking images of the retina with a special camera called a fundus camera by pointing the camera through the pupil at the back of the eye. These cameras are equipped with a complicated microscope and a flash to illuminate the retina. To get the most out of this imaging technique, knowledge of retinal anatomy and recognition of normal and abnormal features is required.

Figure 2.1 shows an example of a fundus photograph of the retina, highlighting some of the anatomical structures of the retina: the optic disc, a round section at the back of the eye where the retina and optic nerve connect and into which the main retinal artery and vein enter; the optic cup, a small indentation in the centre of the optic disc that connects the optic nerve to the retina; the macula, the most sensitive point in the centre of the retina responsible for visual acuity, central vision, and colour vision; and the fovea, a tiny depression in the macula whose function is to detect other image details, such as distinguishing colours and perceiving three-dimensional depth.

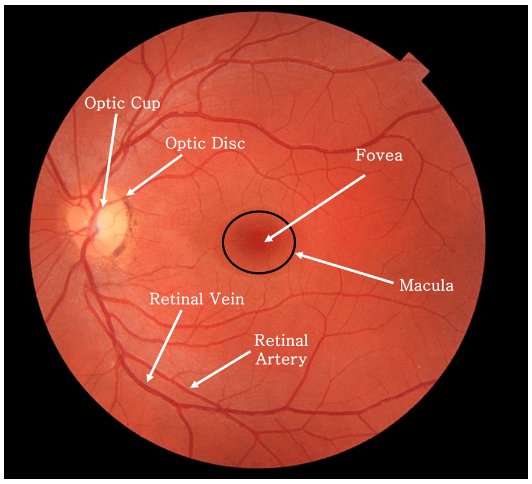


Figure 2.1: Fundus photograph of the retina showing some anatomical structures. [12]

2.1 Diabetic Retinopathy Detection

The likelihood of DR in a diabetic patient increases with time. Regular retinal exams are necessary for diabetics to diagnose and treat DR at an early stage of the disease to prevent vision problems. DR can be detected by the appearance of various lesions on a retinal image, as shown in Figure 2.2. These lesions include microaneurysms, which appear as small red round spots of less than 125μm on the side of the blood vessels, hemorrhages, which are caused by the leakage of blood from the blood vessels after the capillary wall is weakened and eventually ruptures due to the microaneurysms [13], appear as larger spots of more than 125μm. In addition, there are hard exudates, which look like bright yellow spots and are caused by plasma leakage, and soft exudates (cotton wool spots), which appear as white spots on the retina and are caused by swelling of nerve fibers. The lesions described above define Non-Proliferative Diabetic Retinopathy (NPDR), the early stage of DR. The other, more severe stage is called Proliferative Diabetic Retinopathy (PDR). In this latter stage, new blood vessels form, a phenomenon called neovascularization. These new, atypical blood vessels that grow inside the retina are vulnerable and can lead to fluid leakage or cause scar tissue to form, leading to problems with vision.

As mentioned earlier in Chapter 1, the process of manually diagnosing DR is a costly and time-consuming task that requires specialized personnel with extensive experience in detecting this disease based on medical imaging and other factors. In addition, there may be times when the number of patients processed is less due to human limitations. The diagnosis itself is also prone to error. The limitations described are the reason for the need for an automated procedure to diagnose this disease. Although automating the process does not mean that skilled personnel are no longer needed, it does help these professionals by reducing the cost and time required for the diagnosis.

One potential solution for automation lies in the realm of ML. Among the various ML techniques, DL, powered by deep neural networks, stands out due to its demonstrated superiority in image classification tasks compared to traditional ML methods [14, 15, 16, 17]. However, there are challenges that DL alone may not adequately address in the field of ophthalmology. These challenges include maintaining data privacy, dealing with limited local data availability, and fostering collaborative knowledge sharing.

In light of these considerations, the adoption of FL emerges as a viable solution. This approach not only addresses the shortcomings of DL but also tackles the issues of data privacy preservation, localized data scarcity, and the necessity for collaborative expertise sharing.

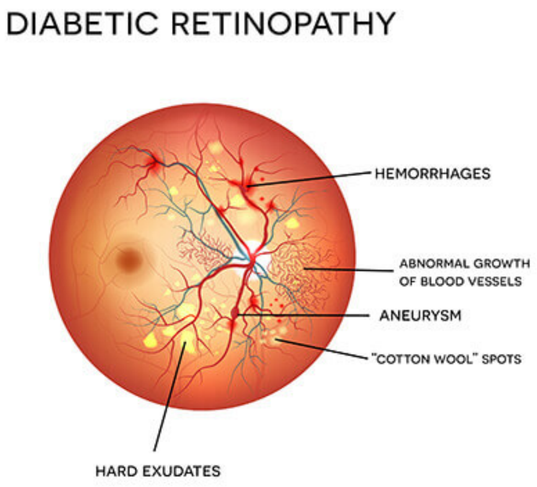


Figure 2.2: Lesions caused by DR. [18]

2.2 Machine Learning

ML is a branch of Artificial Intelligence (AI) that through the use of data and algorithms aims to imitate the way that humans learn. ML is a growing and important part in the field of data science. ML algorithms can be trained to make classifications or predictions based on input data, which can be labeled or unlabeled, giving the algorithm the ability to produce estimates about patterns in the data. Instead of being explicitly programmed, these algorithms learn patterns and relationships within the data to improve their performance over time. ML is broadly divided into three main types: Supervised Learning (SL)-where a model is trained on a labeled dataset, with each input associated with a corresponding label; unsupervised learning-this type of ML deals with unlabeled data and forces the model to discover hidden patterns and structures in the data; and reinforcement learning-where models are trained based on a sequence of decisions to maximize reward, learning by trial and error and adjusting its actions based on the results.

Within the real of ML, there exists a sub-field named DL that specifically focuses on using deep neural networks to perform complex tasks. It can be seen as a specialized and advanced form of ML able to tackle tasks that involve high-dimensional and intricate data.

Deep Learning

DL is a branch of ML based on artificial neural networks that involves hierarchical layers of nonlinear processing stages to extract progressively higher-level features from a given input. Each neural network consists of nodes that mimic the behavior of neurons in the human brain. The neurons that make up a neural network are usually grouped in independent layers. An example is feed-forward neural networks, where data fed into the network travels from the input layer through a series of hidden layers until it reaches the output layer. In computer vision tasks, Convolutional Neural

Networks (CNNs) are the most commonly used architectures. CNNs mainly consist of an input layer, one or more hidden layers, and an output layer. An example of a popular CNN architecture is shown in Figure 2.3. These networks use a mathematical operation known as convolution and can be used for image processing and recognition. When it comes to image classification/recognition, the input is usually a three-dimensional matrix with width, height (corresponding to the dimensions of the image), and depth determined by the number of color channels.

At the end of the network is the Fully Connected Layer (FCL), whose input is the output of the previous layer and whose output is N neurons, where N is the number of different classes into which the images in a given data set can fall.

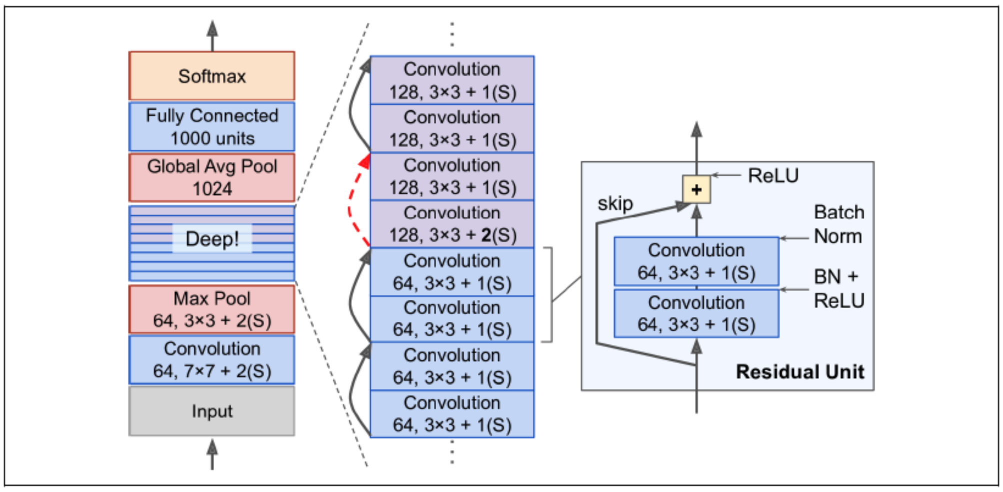

Softmax
Fully Connected 1000 units
Global Avg Pool 1024
Deep!
Max Pool 64, 3×3 + 2(S)
Convolution 64, 7×7 + 2(S)
Input
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 2(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
ReLU
skip
+
Batch Norm
Convolution 64, 3×3 + 1(S)
BN + ReLU
Convolution 64, 3×3 + 1(S)
Residual Unit

Figure 2.3: Architecture of a popular CNN - ResNet50. [19]

2.2.1 Supervised Learning

SL is a type of ML where machines use labeled training data to predict outputs based on that data. An optimal scenario will allow for the algorithm to correctly determine output values for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations. The quality of an algorithm is measured through the so-called generalization error. When testing new unseen data on a trained supervised model, the goal is to estimate generalization error on the unlabeled data [20].

The process of training a model involves choosing the optimal hyperparameters. Hyperparameters are variables whose values are set before starting the model training process with the objective of controlling it. These values are typically tuned by conducting a series of experiments. Some of the most important hyperparameters to tune are:

1. **Learning Rate (LR)** - The LR is an important hyperparameter. It controls how much the model must change in response to the estimated error when the weights are updated. Choosing the optimal LR is difficult because if the chosen value is too large, it will result in too large

weight updates and the model's performance (e.g., its loss) will fluctuate during training. On the other hand, if the value is too small, the model takes too long to reach the minimum and sometimes cannot even reach it.

2. **Batch Size (BS)** - The BS determines the number of samples to process before updating the model's weights and biases are updated. At the end of each batch, the predictions are compared to the ground truth to calculate the loss value, which is then used to improve the model. The BS is usually limited by the available GPU memory, which may mean that the optimal BS cannot be used for the problem at hand.

3. **Number of Epochs** - The number of epochs determines how many times the learning algorithm trains on the dataset. At each epoch, the algorithm uses each sample in the training dataset to update its internal parameters. It is common to define a large number of training epochs to ensure that the error of the model is sufficiently minimized. However, when defining a large number of training epochs, it is common to define an early stopping mechanism. This mechanism has the task of stopping the training of the model when the loss function is no longer improving for a certain number of epochs to avoid overfitting.

2.2.2 Convolutional Neural Networks

Convolutional Layer

As mentioned earlier, the basis of a CNN is the convolution operation, as shown in Figure 2.4. Convolutions act like filters in the form of small squares that glide through an entire image, capturing the most important details. The depth of the convolution result is equal to the number of filters applied. Convolution layers convolve the input and pass the result to the next layer.

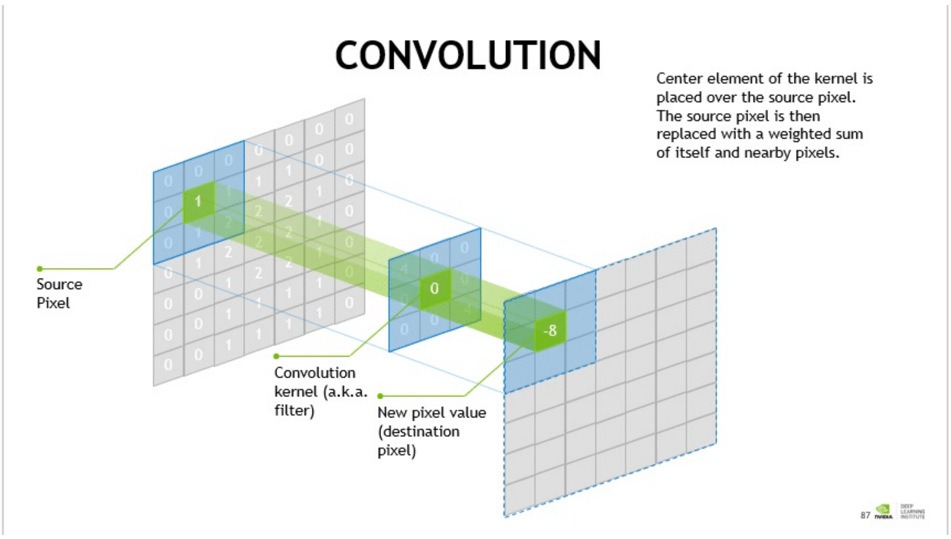

CONVOLUTION
Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
Source Pixel
Convolution kernel (a.k.a. filter)
New pixel value (destination pixel)
-8

Figure 2.4: Convolution operation. [21]

The filter, also known as kernel, is a $N \times N$ matrix usually made of randomly initialized weights, that are updated at every entry during a process named backpropagation. Different-sized filters detect different-sized features in an image, resulting in different feature maps.

Pooling Layer

Pooling layers usually come between convolutional layers. Their main function is to reduce the size of the representations obtained from a previous convolutional layer as well as reduce computational complexity. This layer's procedure is similar to the convolutional layer. After defining an $M \times M$-sized filter, it slides through the feature map with a stride S and produces an output dependent on the pooling operation chosen. The two most common pooling operations are max pooling, which selects the maximum pixel value from the current pixels the filter is sliding through, and average pooling, which calculates the average value of the selected pixels. Figure 2.5 showcases the different results obtained with the pooling operations described above.

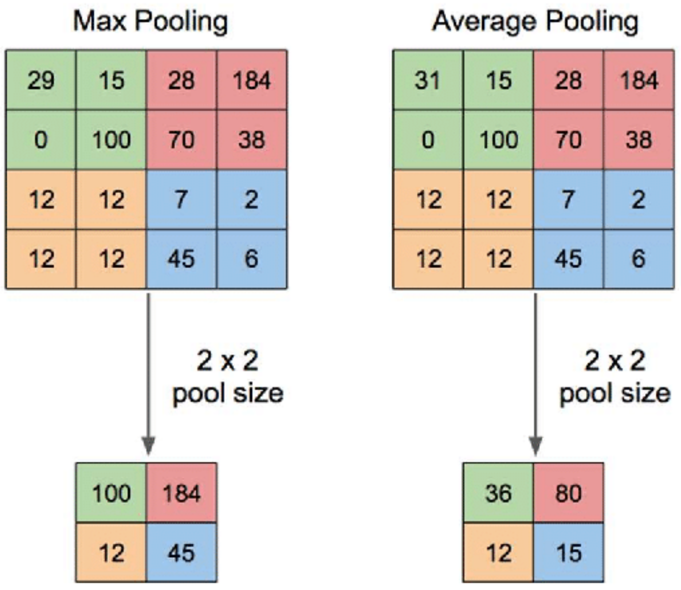


Figure 2.5: Results from Max Pooling and Average Pooling using a 2×2 filter with stride 2. [22]

Activation Layer

Activation layers come immediately after each convolution layer. Activation layers are technically not layers, since no parameters or weights are learned in them; they simply apply a nonlinear activation function to the output feature map of convolutional layers. The Rectified Linear Unit (ReLU) shown in Figure 2.6 is the most commonly used activation function. Two other popular activation functions, also shown in Figure 2.6, are Sigmoid and Hyperbolic Tangent (Tanh).

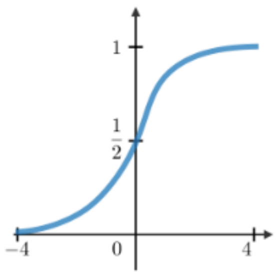
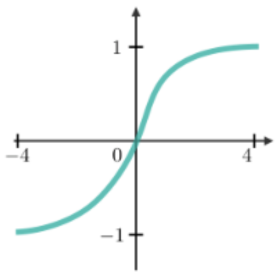
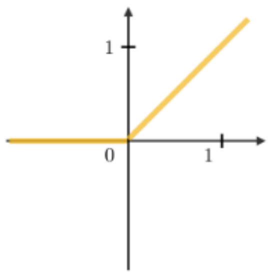
Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

Figure 2.6: Sigmoid, Tanh and ReLU activation functions. [23]

Batch Normalization Layer

The data in an image are its pixels, which are usually between 0 and 255. For this reason, it is common practice to normalize the data to a manageable range before feeding it into the network. This normalization is usually done by setting the mean and standard deviation of a data distribution to 0 and 1, respectively, as in the equation 2.1:

$$x' = \frac{x - \mu}{\sigma} \tag{2.1}$$

where x' is normalized value, x is the value to be normalized, μ is the mean of the data distribution and σ is the standard deviation of the data distribution.

Nevertheless, when training a CNN, the weights in the convolutional layers may become too large, resulting in feature maps with a large scatter of pixel values, making the previously performed normalization of the data a bit pointless. Moreover, the training process itself could become slower or even leading to unstable gradients, affecting the convergence of the network.

To solve this problem, normalization is introduced in each layer of the CNN. Batch Normalization (BN) is another layer that is usually inserted between the convolutional layers. Essentially, it sets all feature maps obtained from a mini-batch fed to the convolutional layer to a new mean and standard deviation. The BN formula is based on the normalization formula shown in equation 2.1, and is written as shown in equation 2.2.

$$x' = \frac{x - \mu}{\sigma} \times \alpha + \beta \tag{2.2}$$

where α and β are learnable arbitrary parameters which the CNN will then use to ensure that pixel values in feature maps are within a manageable range, soothing the gradient instability.

Fully Connected Layer

The FCL, illustrated in Figure 2.7, is the last layer inserted into a CNN. It takes as input the output of the last pooling or convolutional layer, which is *flattened* (the values of the output of the previous layer are rolled up into a vector) before being passed to the FCL. The FCL may consist of one or more layers, with each layer performing the following calculation

$$g(Wx + b) \tag{2.3}$$

where x is the input vector, W is the weight matrix, b is the bias vector, and g is the activation function.

After passing through the first FCLs, which take the inputs from the feature analysis and apply weights to predict the correct label, the final layer typically uses the softmax function to determine the probabilities of the image belonging to a particular class.

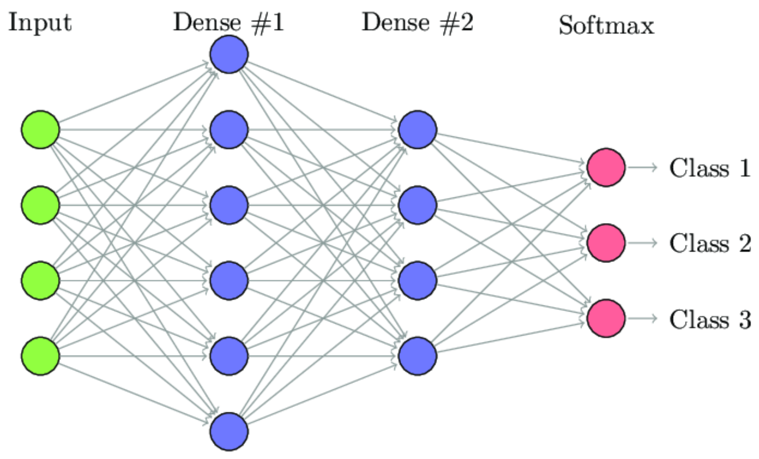


Figure 2.7: Example of a FCL with the input layer, two dense layers and the output layer. [24]

Dropout Layer

The dropout layer is typically implemented in many deep learning architectures to reduce model overfitting. A model becomes overfitted when it learns the statistical noise in the training data, resulting in poor performance when the model is tested with new data. The dropout layer is so named because it drops out nodes and essentially removes the forward and backward connections of those dropped nodes, creating a new network architecture from the original network. An example comparing a FCL with and without a dropout layer can be seen in Figure 2.8

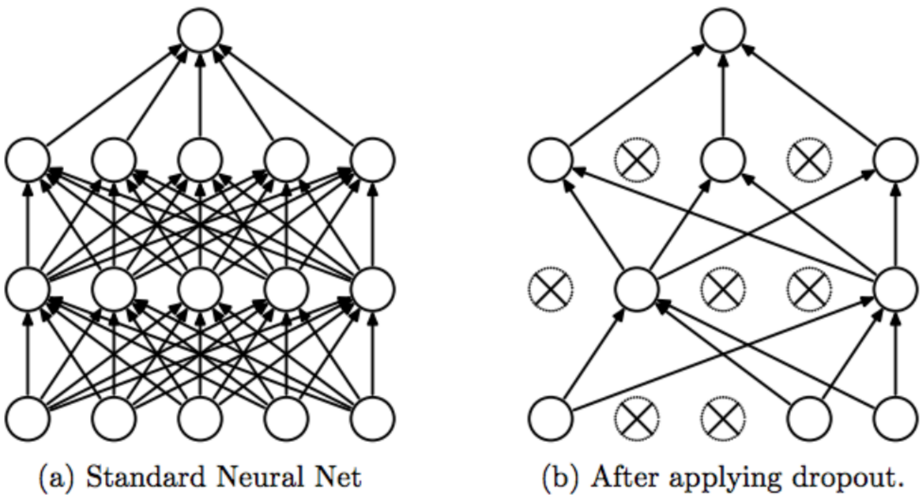


Figure 2.8: FCL with and without dropout. [25]

2.3 Federated Learning

FL (also known as collaborative learning) is a ML technique in which an algorithm is trained across multiple independent devices, usually called clients, each using its own dataset. These datasets remain on the local devices without ever being transferred to a local server. This contrasts FL with centralized learning (illustrated in Figure 2.9), where all local datasets are sent to a server and combined for a single training session. Therefore, ML algorithms, such as deep neural networks, are trained on multiple local datasets contained in local edge nodes, mitigating many of the systemic privacy risks and costs associated with traditional centralized approaches to ML. Training is performed locally on the client, using the global model (i.e., the server's original model) as a starting point. During local training, the client device updates the weights of the model based on its local data and sends the updated model to the central server. On the server, the updated weights from multiple clients are aggregated to obtain an updated global model. The updated global model is then sent back to the clients, and the process repeats. This iterative process continues until the global model reaches the desired level of accuracy. The key idea is that the model updates are computed locally, preserving privacy, while the aggregated updates improve the performance of the global model for all clients. Through collaborative learning from decentralized data sources, FL enables the development of robust and privacy-preserving ML models. An example of how FL works is shown in Figure 2.10.

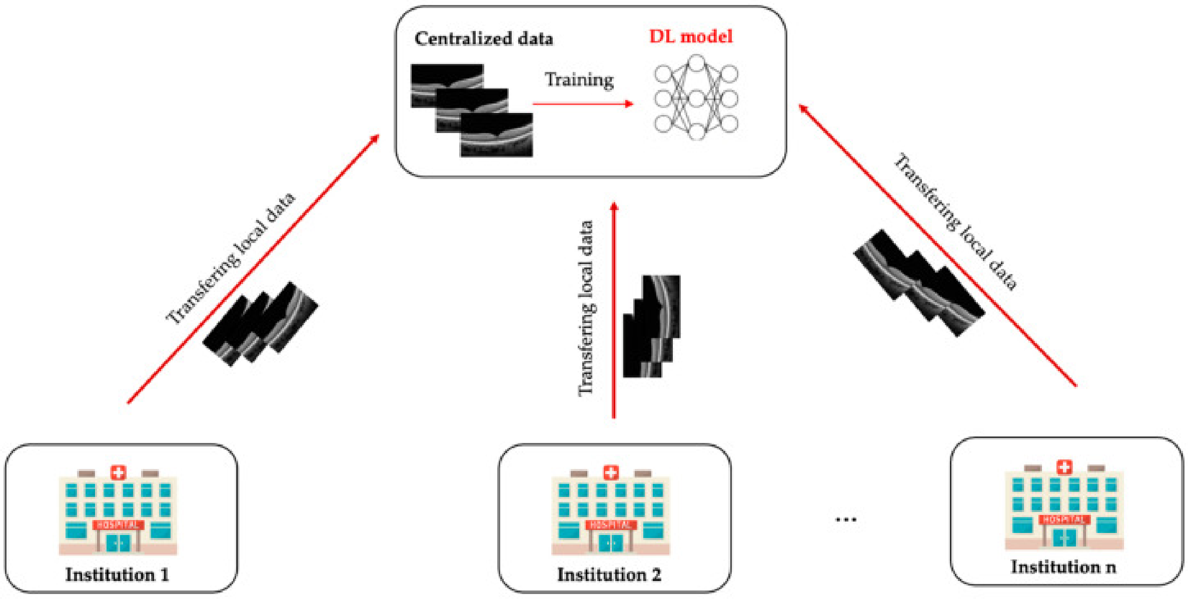


Figure 2.9: Illustration of how Centralized Learning works. [26]

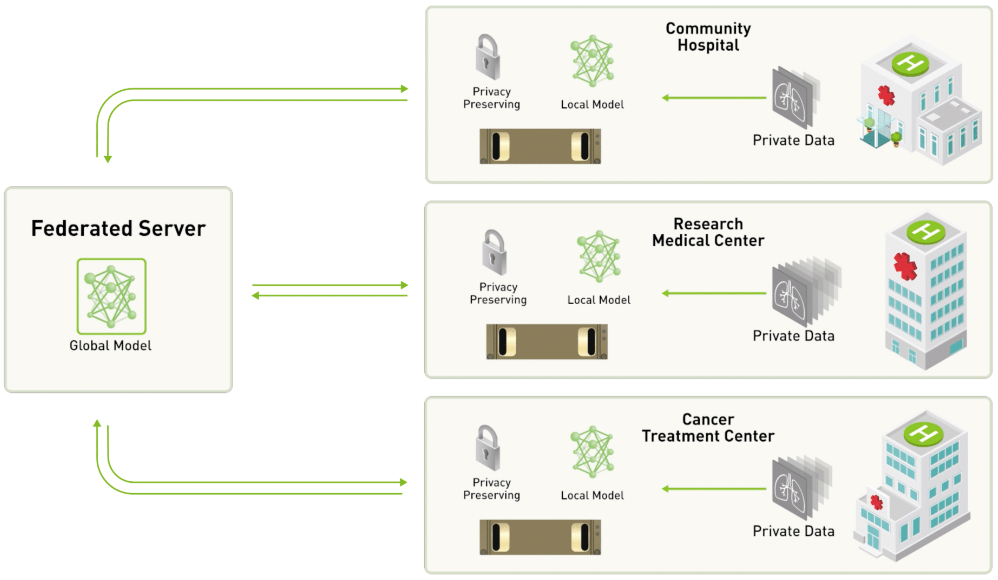


Figure 2.10: Illustration of how FL works. [27]

2.3.1 Problem Formulation

The FL problem involves learning a single, global statistical model from data stored across multiple remote devices. The objective is to learn the model under the constraint that device-generated data is stores and processed locally, with only intermediate updates being communicated periodically. The end goal is typically to minimize the following function:

$$\underset{w}{min} F(w), \quad where F(w) := \sum_{k=1}^{m} p_k F_k(w). \tag{2.4}$$

where m is the total number of devices, $p_k \geq 0$ and $\sum_k p_k = 1$, and F_k is the local objective function for the kth device. The local objective function is usually defined as the empirical risk over local data, i.e, $F_k(w) = \frac{1}{n_k} \sum_{j_k=1}^{n_k} f_{j_k}(w; x_{j_k}, y_{j_k})$, where n_k is the number of samples available locally. The user-defined p_k term specifies weight of the data of each device, with two usual settings being $p_k = \frac{1}{n}$ or $p_k = \frac{n_k}{n}$, where $n = \sum_k n_k$ is the total number of samples [28].

2.3.2 Core Challenges

In order to successfully implement a FL environment, it's necessary to address a set of key challenges:

1. **Communication costs**: federated networks sometimes include a huge number of client devices, which can cause the network to be slower than local computation [29]. There are two main ways to reduce communication - reducing the total number of communication rounds by increasing local updates on each client, or reducing the size of messages in each round using model compression techniques;

2. **Systems Heterogeneity**: the hardware of each client in a federated network may differ to some degree compared to the others. Some methods to address this problem include selecting clients with similar hardware or weighting each client's contribution to the model based on their hardware;

3. **Statistical Heterogeneity**: devices often generate and collect data in a non-identically distributed manner across the network [28]. To address this issue, one approach is to use a weighted average to update the model parameters, with weights determined by the quality or similarity of the data on each device.

4. **Privacy**: this is the main reason for using FL. FL on itself already pushes towards data privacy by sharing model updates instead of raw data. To further protect the information from other clients, one of the most commonly used privacy-preserving techniques is differential privacy, a technique that adds some noise to the trained parameters of the model before uploading to the aggregation server to make it impossible for third parties to distinguish individuals [30]. However, these methods may reduce model performance or system efficiency.

2.3.3 Types of Federated Learning

Regarding different studies, FL can be placed in different categories relating to the participants in the model training and the data used. Figure 2.11 helps with understanding the types of FL.

Types of Participants

1. Cross-device FL: This implementation usually requires a large number of participants for the federation to work with small amount of data. These devices usually range from smartphones, to wearables and edge devices that may not always be online, which raises the need to be careful when choosing to compute as to avoid impacting the user experience;

2. Cross-silo FL: This approach is about training a shared model using data from a few participants, where the participants have a lot of data each. The success of this approach depends on how well the model performs despite the differences in the data, and how efficiently the model is trained with minimal computational and communication costs [31].

Types of Data

1. Horizontal Federated Learning (HFL): HFL means that different datasets share a similar feature space but differ in samples;

2. Vertical Federated Learning (VFL): VFL is applied when different datasets share the same samples but differ in feature space;

3. Federated Transfer Learning (FTL): FTL is applied in scenarios where different datasets differ both in sample space and in feature space.

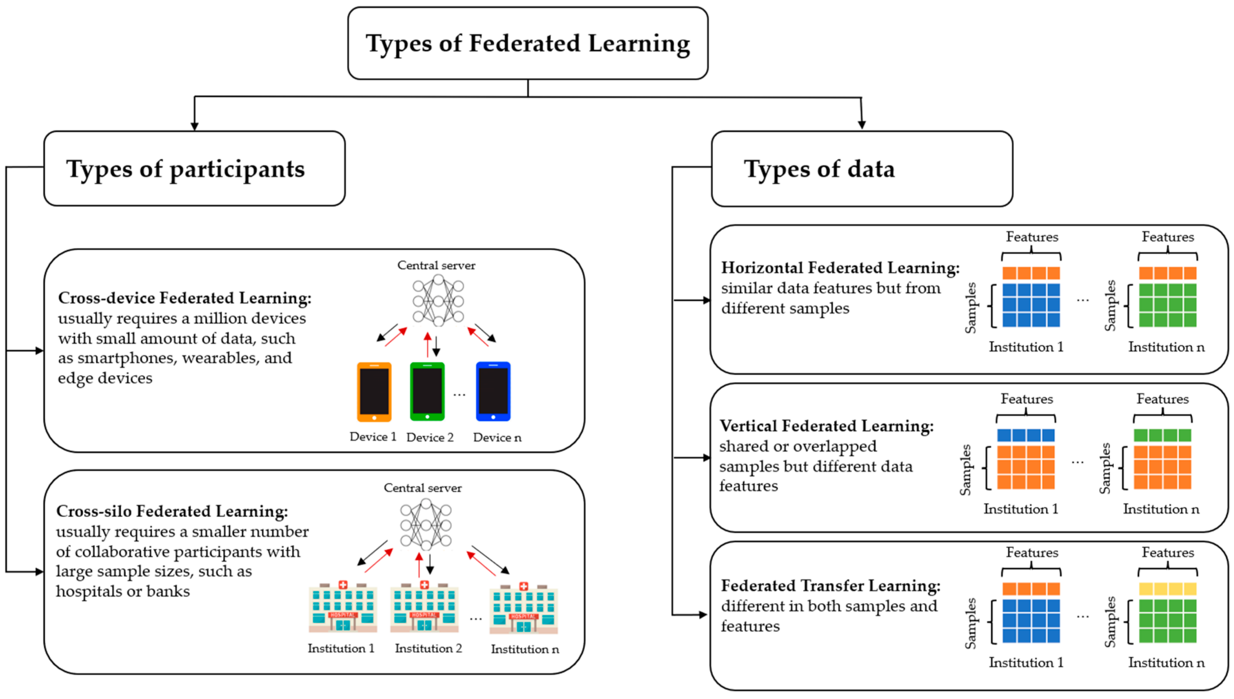

Types of Federated Learning
Types of participants
Types of data
Cross-device Federated Learning: usually requires a million devices with small amount of data, such as smartphones, wearables, and edge devices
Central server
Device 1
Device 2
Device n
Cross-silo Federated Learning: usually requires a smaller number of collaborative participants with large sample sizes, such as hospitals or banks
Central server
Institution 1
Institution 2
Institution n
Horizontal Federated Learning: similar data features but from different samples
Vertical Federated Learning: shared or overlapped samples but different data features
Federated Transfer Learning: different in both samples and features
Features
Samples
Institution 1
Institution n

Figure 2.11: Types of FL. [32]

2.3.4 Aggregation Algorithms

In FL, both clients and server collaborate towards the training of a model. A crucial aspect of FL is model aggregation, a process that brings together the knowledge acquired from the clients to create a global model. Several implementations for FL aggregation algorithms are already reported in literature, and the ones that focus primarily in improving model aggregation will be detailed in this section:

- **Federated Averaging (FedAvg)** was the first implementation of a FL framework, proposed by [11]. The process begins by initializing a global model on a central server, which typically has the same architecture as the models on individual clients and is often pre-trained on a large, centralized dataset. The next step is to randomly, or by a more sophisticated method, select a subset of devices from the available participating devices for each training round. Local model training is performed on each local device using the device's local data. After local training, each device generates a model update consisting of either the updated model only or the difference between the updated model and the previous state. It is then the server's responsibility to collect the model updates from the selected devices and aggregate them into an updated global model. This aggregation process often involves averaging the received local model updates and can be modified to use weighted averaging based on, for example, the amount of data available on a particular client compared to the total amount of data available on all clients.

- **Stochastic Controlled Averaging for Federated Learning (SCAFFOLD)** [33] tries to correct *drifts* in the updates of each client that result from data heterogeneity and can lead to a slow and unstable convergence of the global model. To accomplish this, SCAFFOLD uses control variates in the server and in each client to control the 'client-drift' in the local updates. A control variate is a random variable that is correlated with the error in the local updates. In this context, the control variate is related to the difference between the local and global updates. By adding the control variate to the local updates, SCAFFOLD can reduce the variance of the updates, which makes the updates more likely to converge to the same value. This, in turn, allows SCAFFOLD to converge faster than FedAvg and to be more robust to data heterogeneity.

- **Federated Proximal (FedProx)** [34] also tries to address the challenges of data heterogeneity in order to provide improved convergence. This is done by adding a proximal term defined as:

$$\frac{\mu}{2}\|w - w^t\|^2 \tag{2.5}$$

where μ is a penalty variable, w is the updated local model and w^t is the global model. This proximal term is added to the local loss function to limit the impact of local updates by keeping them close to the global model.

- **Federated Matched Averaging (FedMA)** [35] works by matching and averaging the hidden elements of the local models from different clients, ensuring that the global update is a good representation of the local models from different clients. This, in turn, helps to reduce communication overhead, because the server only needs to communicate the matched hidden elements.

- **Adaptive Personalized Federated Learning (APFL)** [36] allows clients to personalize their models while contributing to the global model by using an adaptive mixing parameter to weigh the contribution of the local and global models. The mixing parameter is dynamically updated based on the client's data distribution and the performance of the local and global models. This helps to improve the accuracy of the local models, because the local models are more likely to be tailored to the individual data distributions of the clients. It also helps to improve the generalization performance of the global model, because the global model is more likely to generalize well to the data on all clients.

- **FedSR** [37] works by encouraging the global model to learn a simple representation of the data. This is done by enforcing an L2-norm regularizer on the representation of the data and a Conditional Mutual Information (CMI) (between the representation and the data given the label) regularizer. The L2-norm regularizer encourages the representation to be small, while the CMI regularizer encourages the representation to be informative.

2.4 Related Work

The authors of [38] propose a method for classifying DR images using a customized CNN architecture with 2 convolution layers and 3 FCLs. The authors first evaluated three pre-trained CNN models, AlexNet, VGG-16, and SqueezeNet, on the MESSIDOR dataset. Before feeding the images to any of the mentioned models, the authors perform a pre-processing stage where the images are cropped with the intent of centralizing the fundus image and evening out the black portions of the image on each side. After cropping, the images are resized to 244×244 pixels to optimize execution time, and histogram equalization is applied to enhance contrast between different parts of the retina. Having trained every proposed model, the authors found that their customized CNN was able to outperform the other models in terms of accuracy, sensitivity and specificity.

In [39], the authors use modified version of the AlexNet CNN architecture (shown in Figure 2.12) to classify DR images from the MESSIDOR dataset based on severity. Before feeding the images to the architecture, the authors extract the green channel of each image due to the fact that this channel provides better details of the retina's structure, and all images were resized to a fixed resolution. The authors then reported that the modified AlexNet architecture achieved a classification greater than the one obtained with the original AlexNet architecture.

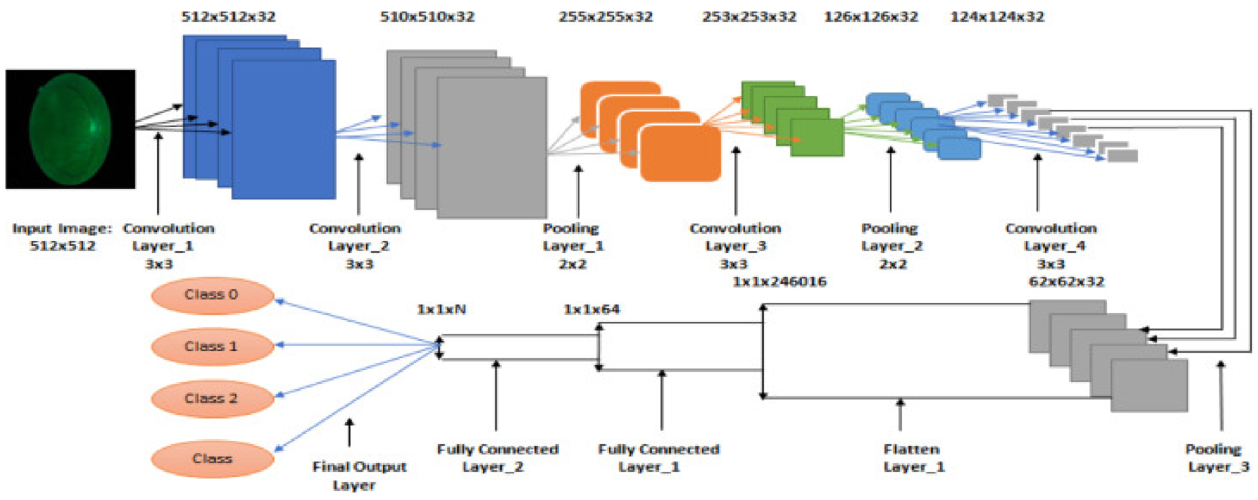


Figure 2.12: Modified AlexNet architecture. [39]

The authors of [40] used Transfer Learning (TL) for automatic classification of DR based on the ResNet architecture. They also analyzed other popular architectures within a 5-fold cross-validation for comparison. The models were trained and tested on the MESSIDOR dataset, with images cropped to a square containing the retina and resized to 900×900 pixels, and data augmentation via horizontal flipping was applied when feeding the images to the model. The authors tested the model on three different scenarios: The first was the multiclass scenario, where all labels were considered; the second was an aggregation of labels 0 and 1, reducing the classification to three classes instead of four; and the last considered only labels 0 and 3, making it a binary classification problem.

In [41], the authors evaluated the VGG basic architecture and used Genetics Algorithm (GA) to evolve the CNN architecture with the goal of improving DR severity classification. The work was evaluated on the MESSIDOR and Kaggle DR datasets. In the pre-processing stage, the images were rotated $40^{\underline{a}}$ and resize to a resolution of 512×512. The best performing architectures in each of the mentioned datasets are depicted in Figure 2.13.

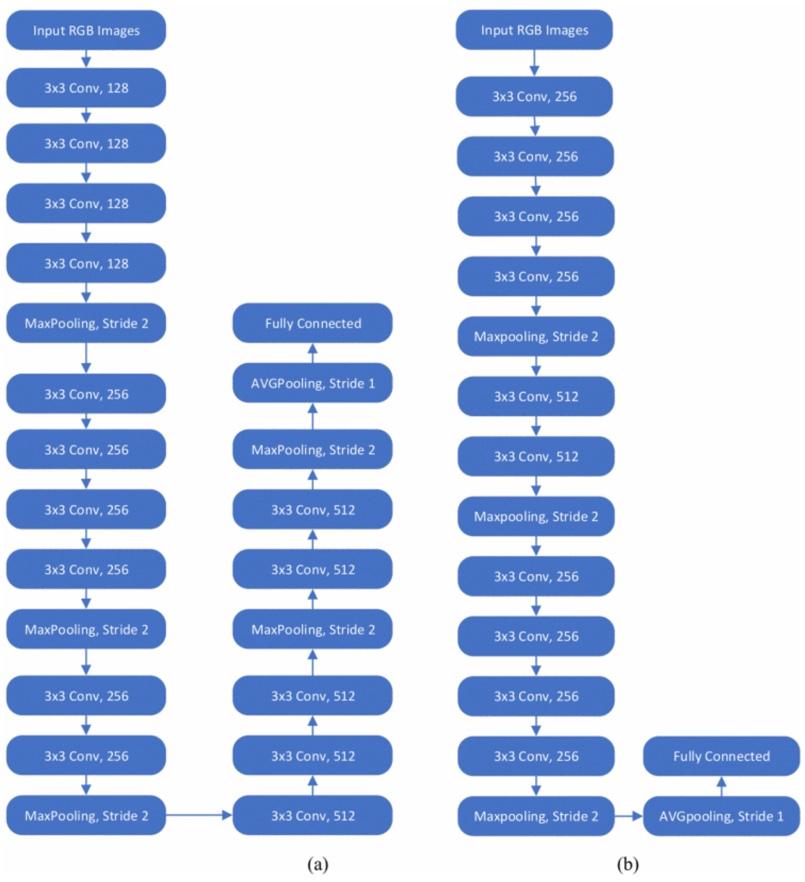


Figure 2.13: Best architectures for each of the datasets, a) Messidor and b) Kaggle DR. [41]

The authors in [42] proposed three main models for classifying retina images based on two classes of non-DR and DR. Non-DR stands for patients whose eyes are in a healthy form, and DR means that a patient needs treatment. The standard model is a non-FL approach that utilizes TL with a modified AlexNet architecture, which is a pre-trained CNN using weights from the ImageNet dataset, but only the last three FCLs have been altered. The other two models employed different FL aggregation algorithms, while also utilizing the same TL technique as the standard model. The first of these models used the FedAvg algorithm, a widely used method to combine the parameters of local models from multiple clients. This algorithm calculates the average of the parameters received from each client to update the global model. The final model employed the FedProx algorithm, a variation of the widely-used FedAvg method.

Lo *et al.* [43] proposed a FL framework for deep neural network-based retinal microvasculature segmentation and Referable Diabetic Retinopathy (RDR) or Non-Referable Diabetic Retinopathy (NRDR) classification using OCT and Optical Coherence Tomography Angiography (OCTA). The main components of the framework are the central server, which is responsible for training co-ordination, defining the hyperparameters of each client's hyperparameters, and aggregating each client's updates, and the individual clients, which use their local data to train the global model. Upon receiving the model, each client performed data augmentation and trained on its training set for a full epoch before validating on its validation set. Secure model transfer between each client and the central server was accomplished via cloud-based drop-off folders, as illustrated in Figure 2.14. Upon completion of training, each client sends its resulting model to the central server along with a Comma-Separated Values (CSV) file containing the loss and accuracy for both training and validation. After receiving the model and CSV file from each client, the server further validates each client model on a small validation set to choose the best performing models and rejecting those that could disrupt the training process. The authors first evaluated the performance of the FL framework for microvasculature segmentation on a simulated dataset of 153 OCTA en face images. The authors then used a real-world dataset of 700 eyes for RDR classification.

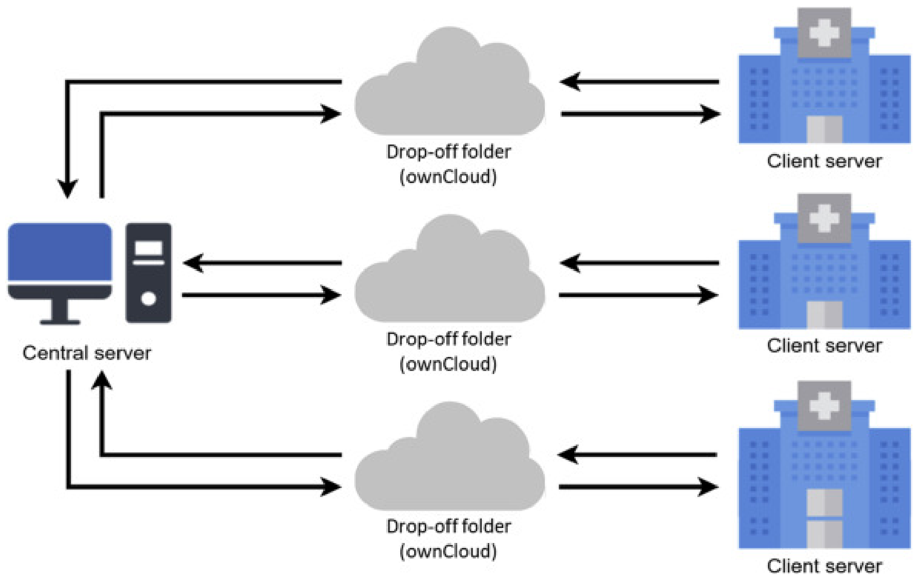


Figure 2.14: Model transferring between server and client through cloud-based drop-off folders. [43]

In [44], the authors developed a FL approach for classifying Age-related Macular Degeneration (AMD) using OCT image data from three distinct datasets to simulate three different institutions contributing to the training of a global model for binary image classification. The primary FL algorithms are FedAvg and Federated Stochastic Gradient Descent (FedSGD), first presented in [11]. The authors highlight the challenges of FL, especially the heterogeneity of the data, and recommend the use of Domain Alignment (DA) as a solution. DA is a process that modifies a ML model trained in one domain to perform proficiently in a related domain. The implemented framework uses four different DA strategies: FedProx, FedSR, FedMRI [45], and APFL. FedMRI divides the MR reconstruction model into two parts: a globally shared encoder to obtain a generalized representation at the global level, and a client-specific decoder to preserve the domain-specific properties of each client.

2.5 Software and Datasets

This work used Python 3.7.6 and Pytorch 1.13.1, an open source ML library used for developing and training neural network based DL models. All the experiments done in this work used public eye fundus images datasets (Table 2.1). The training phase was done using two datasets (1 and 2), while the testing was performed on a single dataset (3).

Nº	**Dataset**	**Nº of Images**
1	MESSIDOR [46]	1200
2	Kaggle DR [47]	35126
3	APTOS [48]	3662

Table 2.1: Datasets used.

All the aforementioned datasets have different characteristics:

- **MESSIDOR** is composed by a total of 1200 colored eye fundus images with a resolution of either 1440 × 960, 2240 × 1488 or 2304 × 1536 pixels with 8 bits per color plane. The images were acquired by three different ophthalmologic departments with a a color video 3CCD camera mounted on a Topcon TRC NW6 non-mydriatic retinograph with a 45 degree field of view. The DR samples present in the dataset indicate various levels of severity, graded from 0 to 3. A brief explanation regarding the levels of severity is shown in Table 2.2.

- **Kaggle DR** is composed by 35126 high-resolution retina images taken with different models and types of cameras under a variety of imaging conditions. A left and right field is provided for every subject. The images found in this dataset may contain artifacts, be out of focus, underexposed, or overexposed. Here, the DR grading ranges from 0 to 4, unlike in the MESSIDOR dataset. The description of the dataset can be found in Table 2.3.

- **APTOS** is a set of 3662 retina images taken using fundus photography under a variety of imaging conditions. Just like in the Kaggle DR dataset, the images may contain artifacts, be out of focus, underexposed, or overexposed. The images were gathered from multiple clinics using a variety of cameras over an extended period of time, which will introduce further variation. The grading used in this dataset also ranges from 0 to 4, with a description given in Table 2.4.

DR Stage	**Label**	**Description**	**Nº of Images**
Healthy	0	Zero symptoms	548
Mild NPDR	1	A few mycroaneurisms	152
Moderate and Severe NPDR	2	More mycroaneurisms and a few hemorrhages	246
PDR	3	More hemorrhages and beginning of neovascularization	254

Table 2.2: Description of the MESSIDOR Dataset.

DR Stage	**Label**	**Nº of Images**
Healthy	0	25810
Mild NPDR	1	2443
Moderate NPDR	2	5292
Severe NPDR	3	708
PDR	4	873

Table 2.3: Description of the Kaggle DR Dataset.

DR Stage	**Label**	**Nº of Images**
Healthy	0	1805
Mild NPDR	1	370
Moderate NPDR	2	999
Severe NPDR	3	193
PDR	4	295

Table 2.4: Description of the APTOS Dataset.

3 Federated Learning Applied to Diabetic Retinopathy Detection

This chapter provides a detailed description of both local and FL methods for DR detection. The section regarding local implementation includes several aspects, including the chosen model and the fundamental reason for its adoption. It goes on to explain the improvements made to the chosen model to improve its performance. This section also dives into the novel tactics used to address class imbalance issues inherent in the datasets.

The section on the FL approach covers the definition of the participating clients, how each client trains its personal model, how the training datasets are divided among the clients, and the implemented averaging algorithms.

The FL approach is used to simulate two different scenarios: the first is a scenario in which clients have all of their local data available from the start, without ever adding new data or removing old data; the second is a scenario in which the client's dataset is divided into subsets, with each subset being used to train the model for a single communication round.

It should be noted that the local implementation serves as the basic standard against which the performance of FL trained models is measured. The goal of this comparison analysis is to determine and explain the disparate outcomes obtained by the two separate methodologies.

3.1 Reference Models Trained Locally

3.1.1 Model Selection and Implementation

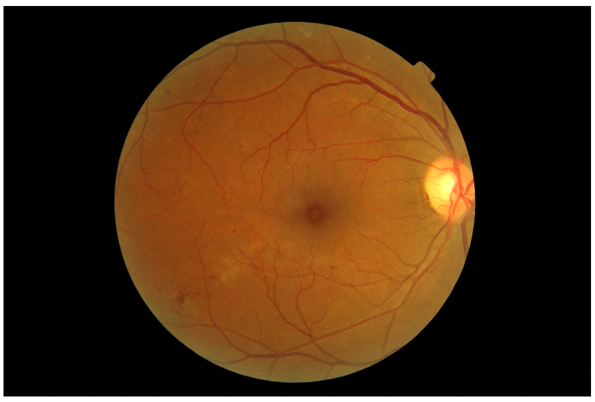
The first step in creating the locally trained reference model was a thorough review of relevant literature on DL implementations for the detection of DR, which is presented in 2.4. Among the abundance of published research, four publications stood out as particularly promising and significant in this domain: [38], [41], [39], and [40]. The use of the MESSIDOR dataset, which is thoroughly described in Section 2.5, was a commonality throughout these investigations, which was also one of the reasons for choosing these works in particular.

Following a thorough investigation of the implementation steps outlined in the aforementioned works, it became clear that the study providing the most similar results to those described in their publications was that conducted by [40]. This effective alignment of outcomes highlights the robustness of the deployed approach and instills confidence in its ability to detect DR accurately. Next, a thorough explanation regarding the preprocessing of the data and the implementation of the chosen model is given.

The images in the dataset were preprocessed to extract the region of interest, specifically the retina. Each image was initially cropped and shrunk to a constant resolution of 900×900. The OpenCV library was used to help in this cropping technique.

A grayscale duplicate was created upon reading an image. Following that, a thresholding technique based on Otsu's method was used. To compute an optimal threshold value, Otsu's approach uses the statistical distribution of pixel intensities inside the grayscale image. This threshold value worked as a differentiator for pixel intensities, resulting in the grayscale image being converted to a binary representation. Pixels with intensities higher than the threshold were turned to white, while those with intensities lower than the threshold were set to black.

This binary image was then used to find contours inside it. The bounding box of the greatest exterior contour — basically an encompassing frame around the object of interest — was extracted by analyzing these contours. To finish, the original RGB image (Figure 3.1a) was cropped using the coordinates provided by this bounding box. The result was a square picture segment containing the retina (Figure 3.1b).


(a) Original image.

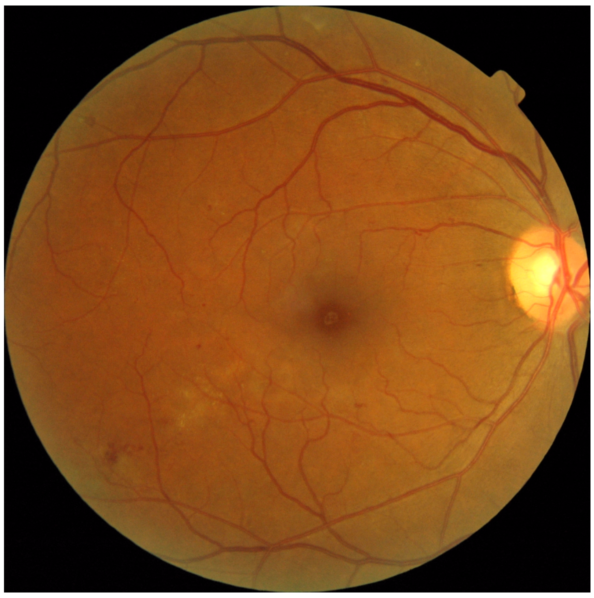

(b) Cropped and shrunk image.

Figure 3.1: Comparison of the same retina image before and after being cropped and resized.

The local implementation process began after the image processing phase. Initially, a strategy was devised for loading the dataset, which involved creating a CSV file containing the relative paths to the newly processed images, along with their corresponding labels. PyTorch's customizable

Dataset class was used to make dataset handling easier. Once the dataset was loaded, it was handed to PyTorch's Dataloader, which was in charge of retrieving instances from the dataset, grouping them into batches (with a batch size of $BS = 4$ in this case), and delivering them to the model for processing. Data augmentation in the form of random horizontal flips (with $p = 0.5$) is used to increase the dataset's diversity even further. Image intensity is automatically scaled to the range $[0, 1]$ by PyTorch when converting an image to a tensor though the use of the ToTensor class.

It is now necessary to define the model that will be trained on the dataset. The one that gave the best results in the work being followed was ResNet50 along with TL (ResNet50 was trained on the Imagenet dataset [49], which contains more than 14 million images belonging to 1000 different classes) by retraining only the last residual block and the FCL. Since the original architecture was made to classify 1000 different classes, it is necessary to modify the FCL to classify only the necessary number of classes.

The initial defined loss function was actually a combination of two loss functions, Mean Absolute Error (MAE) and Mean Square Error (MSE) to penalize non-adjacent mistakes. The optimizer used is the well known Stochastic Gradient Descent (SGD) with $LR = 0.001$ and $momentum = 0.9$. With everything in place, it is now possible to start training and testing the model. The training and testing phases of the model used a subset of the MESSIDOR dataset. Specifically, 80% of the dataset was set aside for training, with the remaining 20% set aside for rigorous testing.

It should be noted that this work also tested the three distinct scenarios tested by the authors, i.e., considering all of the classes; aggregating class 0 and 1; and considering only class 0 and 3. To meet Retmarker's goal of building a model capable of identifying RDR and NRDR effectively, the case in which a binary classification problem is considered will be the main focus of this work, even though the fine-tuning of the model, i.e., improving the model classification (Section 3.1.2) and implementing solutions to solve the inherent issues of imbalanced data (Section 3.1.3), was done considering all of the available classes on the dataset.

3.1.2 Improving the Model Classification

While the initial results obtained with the base model were adequate, there was room for improvement. The results obtained with every change made are present in Appendix A and the best results obtained are in Section 4.1.

The first change was to use an alternate loss function known as Cross Entropy (CE). This loss function quantifies the difference between expected probability and actual labels in classification problems. Lower CE loss values suggest that the model's predictions and factual labels are more closely aligned. The distinguishing feature of CE loss is its propensity to penalize incorrect predictions more harshly than correct ones, hence emphasizing precision.

Following that, a series of experiments were carried out to determine the effect of different

BS values on the model's performance. Due to Video Random Access Memory (VRAM) capacity constraints, the range of exploration was limited to a maximum BS value of 8. Following careful examination, the best BS value was determined to be 4. This decision was reached after considering computing constraints and their consequences for efficiency.

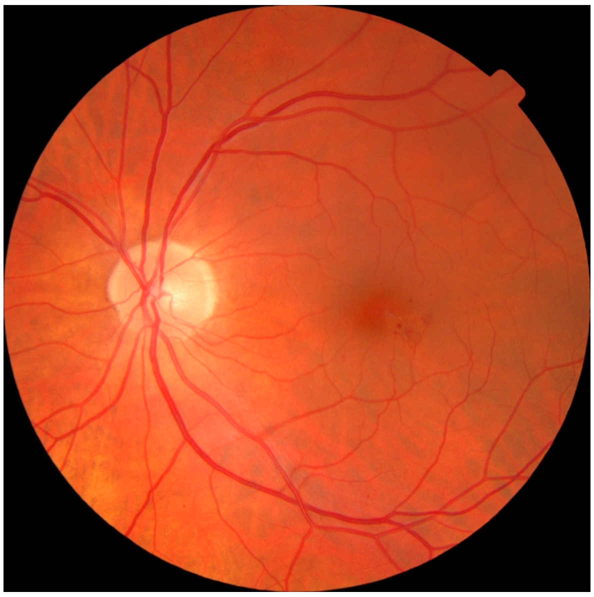
Additional improvements were made by broadening the scope of model training. Previously, only the final residual block and the FCL of the model architecture were trained; however, the paradigm was altered to include the full model architecture. This approach was used in order to fully utilize the potential of all the layers that form the ResNet50 architecture.

The investigation then moved on to optimizers, with a particular emphasis on the Adaptive Moment Estimation (Adam) optimizer. Although Adam exhibits a faster convergence rate compared to SGD, it has been noted to potentially underperform in terms of generalization performance [50]. This led to a careful consideration of trade-offs when selecting the most suitable optimizer, taking into account the model's convergence speed and its ability to generalize effectively.

Given the MESSIDOR dataset's small size, efforts were focused on improving the already implemented data augmentation techniques. This included introducing random vertical flips (with $p = 0.3$) as well as random rotations of up to 40 degrees. These additions attempted to introduce unpredictability into the dataset, enhancing the model's adaptability to various settings.

Given the fact that ResNet50 is a deep and complex architecture with numerous layers, it tend to have a high capacity to fit the training data perfectly, which can lead to overfitting if not controlled. Because of this, a dropout layer (with $p = 0.4$) was added between the last convolutional layer and the FCL to regulate this complexity and balance the model's expressiveness.

The final experiment involved using histogram equalization techniques to the images in order to improve both image quality and contrast. The approach utilized was Contrast Limited Adaptive Histogram Equalization (CLAHE). CLAHE, unlike global histogram equalization approaches, works on smaller image portions known as "tiles." CLAHE distinguishes itself by focusing on these localized tiles and boosting their contrast individually. To avoid the problem of artificial boundaries, CLAHE uses bilinear interpolation to seamlessly blend adjacent tiles, resulting in a more natural and coherent result. In the context of this work, CLAHE was applied selectively to the green channel of the images. This channel was chosen because of it encapsulated the most critical information, particularly blood vessels and exudates. To achieve this channel-specific augmentation, the RGB image (Figure 3.2a) was first decomposed into three independent matrices, each representing a different color channel. Following that, CLAHE was only applied to the matrix corresponding to the green channel. The final stage involved the combining of the three matrices, which resulted in an improved image (Figure 3.2b). There are two hyperparameters than need to be defined when using CLAHE: "clipLimit" and "tileGridSize." The former controls the threshold for restricting contrast, while the latter regulates how tiles are arranged in rows and columns. The specific parameter values utilized for this experiment were 2.0 for "clipLimit" and $(8, 8)$ for "tileGridSize."


(a) Image without CLAHE applied.

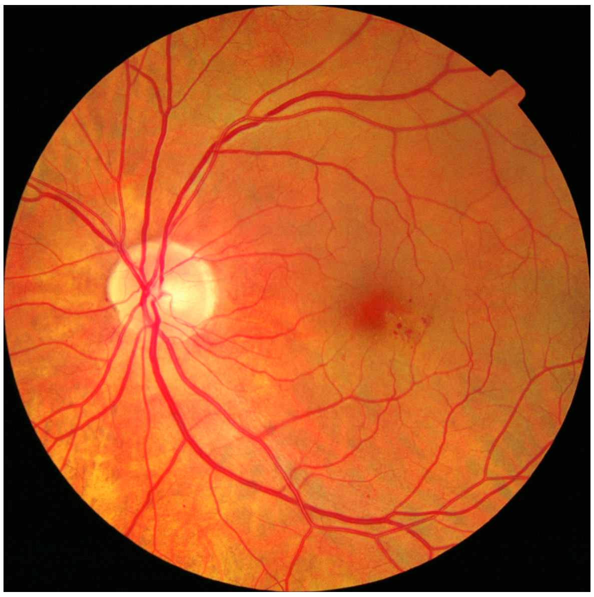

(b) Image with CLAHE applied.

Figure 3.2: Same retina image before and after applying CLAHE.

3.1.3 Solutions for Imbalanced Datasets

By analyzing Tables 2.2, 2.3 and 2.4, it is possible to conclude that one class significantly outnumbers one or more other classes. In all the used datasets, class 0 outnumbers all the other classes. This situation is more apparent in the Kaggle DR dataset, where in a total of five different classes, 73.48% of its data belongs to class 0. This is a common problem in real world scenarios, and thus, there are already a few solutions for this problem. The solutions that were implemented to try to solve this problem are:

- **Class Weighting**: in this technique, each class is assigned a different weight during training so that the contribution of each class is balanced. The weights assigned to each class are usually inversely proportional to their quantity in a given dataset. For example, in the Kaggle DR there are 25810 images belonging to class 0 and 2443 images belonging to class 1, which means that the weight assigned to class 0 is lower than that assigned to class 1.

  To implement Class Weighting, the respective weight of each class was calculated using the following equation:

$$w_i = \frac{1 - n\_samples(i)}{n\_total} \tag{3.1}$$

  where w_i is the resulting weight for class i, $n\_samples(i)$ is the number of samples belonging to class i, and n_total is the total number of samples in the dataset. After calculating the weights for each class, the resulting values are divided by the sum of all class weights to normalize the values.

- **Oversampling**: this technique alters the dataset in a way that increases the amount of minority observations that are shown to the model at each epoch. In PyTorch, this is done through the use of *samplers*, more specifically, *WeightedRandomSampler* which needs to be passed to the Dataloader. This sampler requires class weights for each class, which can be calculated following Equation 3.1. The sampler will then sample the data to be shown to the model based on the passed class weights, which will basically duplicate samples from the minority classes (Figure 3.3) in order to balance the classes that the Dataloader feeds the model at each iteration.

- **Focal Loss** [51]: the focal loss function has two major advantages over the cross-entropy loss function: it focuses on difficult examples, i.e., the examples that are most difficult to classify correctly, and improves the performance of the model on these examples; it reduces the weighting of the easy examples, i.e., the examples that are easiest to classify correctly. This prevents the model from overfitting to these examples. The focal loss function is defined by :

$$FocalLoss = -\alpha(1 - p_t)^{\gamma} \log p_t \tag{3.2}$$

where α is a hyperparameter that controls the weighting of easy examples (a higher value gives more weight to easy examples, while a lower value gives more weight to hard examples), γ controls the degree of focus on hard examples (a higher value puts more focus on hard examples, while a lower value puts less focus on hard examples), and p_t is the predicted probability of the positive class.

In this work, the value assigned to α was the class weight relative to the class of the image to be classified, calculated according to equation 3.1, and the optimal γ value was found by experimentation, with $\gamma = 1.5$ giving the best results.

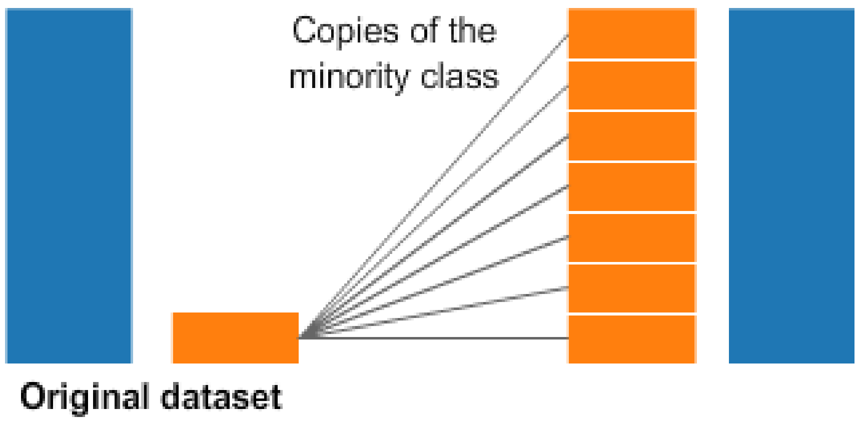


Figure 3.3: Oversampling illustrated.

After testing all the mentioned solutions to deal with data imbalance, the one that gave the best results was Focal Loss. Since this was the better performing approach, which overall obtained the best results, it was tested in the three different scenarios mentioned in the end of Section 3.1.1. The results are shown in Section 4.1.

3.1.4 Binary Classification and Training the model on the Kaggle DR Dataset

After conducting all the experiments described in the sections above with the initial local reference model, the problem's approach was altered to fit Retmarker's goal. The primary goal was to train a model capable of differentiating between cases of RDR and NRDR. To accomplish this, a change was made in the way classes were classified.

The technique employed involved combining classes 0 and 1 into a unitary category called class 0, designating NRDR. Similarly, the remaining classes were merged into a new single category designated class 1, which represented RDR. Figures 3.4 and 3.5 depict this reclassification procedure.

By implementing this classification transformation, the resulting distribution of images took on a different structure, now categorized into the newly defined classes. An overview of these adjusted class distributions can be found in Tables 3.1, 3.2, and 3.3.

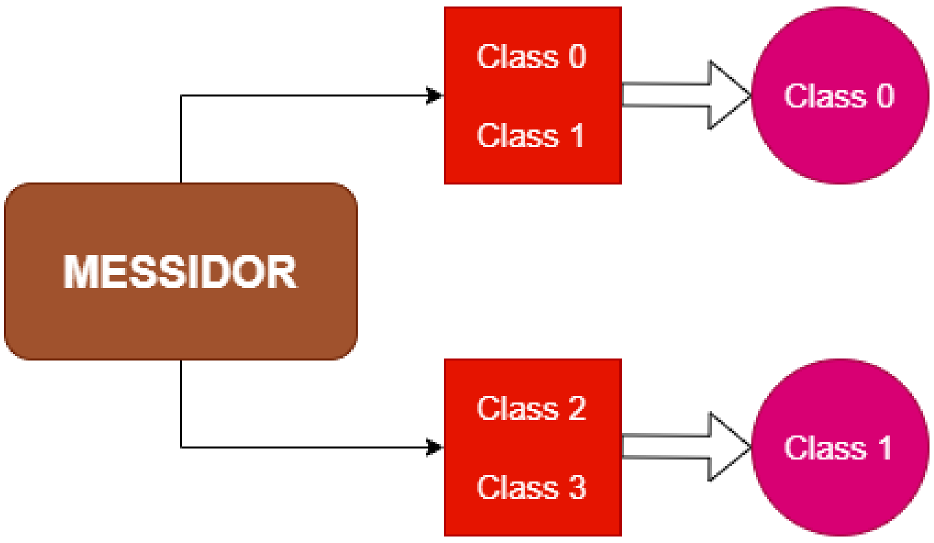


Figure 3.4: Class conversion for MESSIDOR dataset.

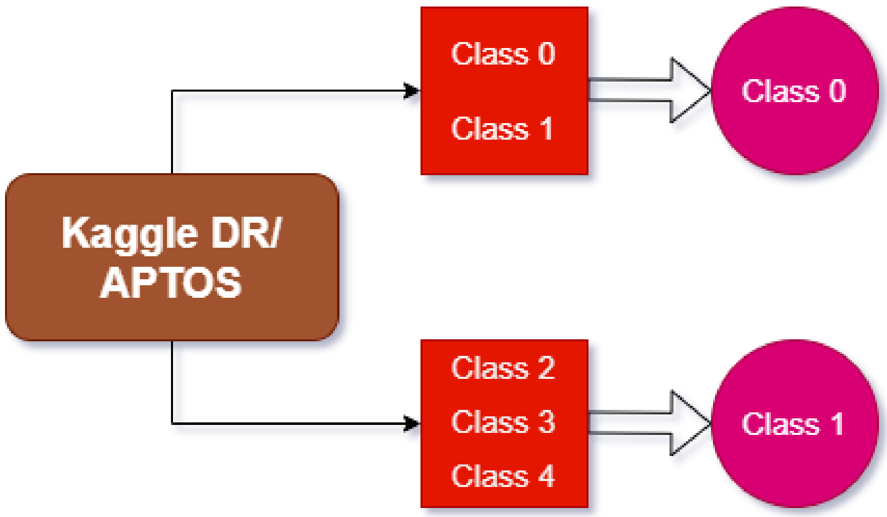


Figure 3.5: Class conversion for Kaggle DR and APTOS datasets.

Label	**Nº of Images**
0	700
1	500

Table 3.1: MESSIDOR's image distribution for binary classification.

Label	**Nº of Images**
0	28253
1	6873

Table 3.2: Kaggle DR's image distribution for binary classification.

Label	Nº of Images
0	2175
1	1487

Table 3.3: APTOS's image distribution for binary classification.

Following multiple experiments on a small dataset, the idea of expanding to a larger dataset for training gained traction. The Kaggle DR dataset stands out among the accessible public datasets due to its large size and popularity, boasting approximately thirty times the data volume of the MESSIDOR dataset. Following this decision, future training attempts focused on using the whole of images from either the MESSIDOR or Kaggle DR datasets.

Simultaneously, a change was made to the testing phase, with the APTOS dataset now serving as the evaluation benchmark. This choice ensured a fair and consistent basis for comparing the outcomes obtained by models trained on the MESSIDOR dataset to those obtained by models trained on the Kaggle dataset.

As a result of these changes, two distinct locally trained reference models emerged. The first model was trained on the MESSIDOR dataset, while the second was trained on the Kaggle DR dataset. These two models now serve as foundational benchmarks for comparison against the FL trained models which will be discussed in the next section.

It is worth noting that these locally trained models were saved in their final phases with the purpose of using them as the initial global models in the FL implementations.

3.2 Federated Learning Trained Models

FL is mainly defined by a central server and the client that participate in the training. FL is implemented in this study on a single computer utilizing a simulated environment. Although all clients' models are trained on the same machine, which also serves as the central server, the architecture mimics the decentralized nature of FL. This method allows the examination of FL concepts while regulating the variables and conditions in a controlled setting.

1. **Central Server**: In this simulated FL setup, the central server is used for training coordination. The central server maintains the global model and orchestrates the communication with virtual clients. It simulates the aggregation of model updates received from clients to produce a refined global model. Below is an overview of the implemented FL architecture to run the simulations:

2. **Virtual Clients**: Instead of genuine distributed devices or servers, virtual clients are emulated on the same computer. Each virtual client represents possesses its own dataset, giving the

impression of data decentralization. These virtual clients train local models autonomously using their unique datasets.

3. **Local Model Training (Virtual Clients)**: Each virtual client follows the FL paradigm by doing local model training on its dataset. This training procedure mimics the behavior of a real-world client updating its model using its personal data. The model updates are generated based on the features of the local data and are sent to the central server for aggregation.

4. **Global Model Aggregation**: The central server aggregates model changes from virtual clients using simulated aggregation techniques. The goal of this method is to imitate the privacy-preserving aggregation of model parameters that would take place in a true FL scenario. The aggregated model update is refined into the global model, which is then shared with the virtual clients.

5. **Iterative Process**: The FL process is performed iteratively, simulating the communication rounds rounds in a real-world FL scenario. Virtual clients continue to train locally, make model updates, and send them to the central server for aggregation. Over numerous communication rounds, the central server simulates the process of refining the global model.

It is possible examine the benefits and challenges of FL in a controlled environment by creating this simulated FL architecture, which is represented in Figure 3.6. Despite the fact that the setting differs from actual dispersed circumstances, this method enables us to explore the impact of parameters such as data distribution and the importance of the chosen aggregation algorithm.

The hyperparameters established within the locally trained reference models will be integrated into this simulation. This uniformity ensures consistency as the same hyperparameters are used across all clients.

It is crucial to emphasize that during the training of a FL model on the MESSIDOR dataset, the global model's initialization involves utilizing the final state of the reference local model that was trained on the Kaggle DR dataset. Conversely, this principle holds true in reverse as well. In other words, when training a FL model on the Kaggle DR dataset, the initial global model is initialized with the final stage of the local reference model trained on the MESSIDOR dataset.

For the scenario where each client has all images available, the experiments ran for a total of fifty communication rounds. With the MESSIDOR dataset, each client performed five local updates before sending the modified model back to the central server. When using the Kaggle DR dataset, on the other hand, each client performed three local updates before sending the model to the central server.

In the alternative case, when each client's dataset is partitioned into subsets, the number of communication rounds is decreased to four, mirroring the number of subsets generated for each client. Regardless of the training dataset, a consistent count of ten local updates was done across all clients.

The results obtained with each of the implemented aggregation algorithms are in Section 4.2. Furthermore, Appendix B shows the results when testing on a different setting. This setting consisted in training a FL model with one of the datasets used specifically for training, while testing on a specific portion of the other.

The following sections will detail the division of the datasets among individual clients and the implementation of the chosen aggregation algorithms.

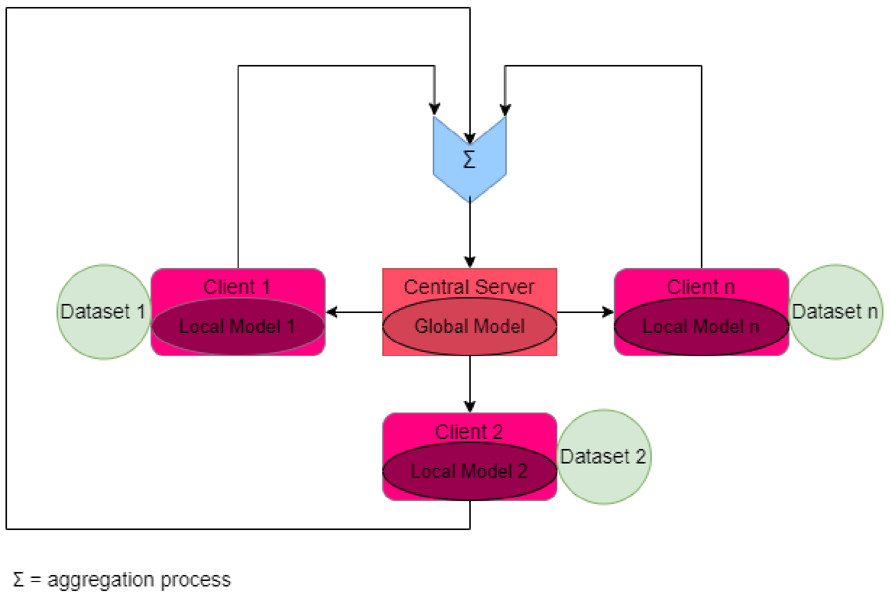


Figure 3.6: Implemented FL framework diagram.

3.2.1 Datasets Partition

This section is dedicated to explain how the training datasets, the MESSIDOR and Kaggle DR, were distributed among the virtual clients, closely resembling the mechanics of a FL scenario. Furthermore, as mentioned in the first section of this chapter, this section will also explain the process of producing subsets of the dataset within each client.

A new column was included into the CSV file used for dataset loading to identify the images owned by each client and simulate the notion of individual clients in a FL framework.

This new column was assigned to indicate each virtual client's unique identifier. As a result, when it became necessary to access the dataset associated with a certain client—say, Client 0—a simple data filtration mechanism was developed. The relevant rows within the dataset identified by the inclusion of Client ID 0 were retrieved, thus constructing an artificial separation akin to genuine scattered clients.

Furthermore, the goal of simulating dataset division into subsets needed another identifier. In this case, another extra column was added to the dataset, indicating the communication round selected for the use of each image.

MESSIDOR

As stated in the MESSIDOR dataset description (Section 2.5), the collection of images was the result of the efforts of three independent ophthalmologic departments. Because of this intrinsic distribution, the idea of sharing the dataset across three virtual clients, each simulating a department's contribution, emerged. Coincidentally, each department contributed with exactly 400 images, effectively equating the contribution of data across the clients.

Through supporting *xls* files, the necessary information about the individual department associated with each image is incorporated within the downloadable dataset. The introduction of departmental information made the process of assigning images to the virtual clients more fluid and easy.

The division of the dataset between clients and the division by subsets are shown in Figures 3.7 and 3.8, respectively.

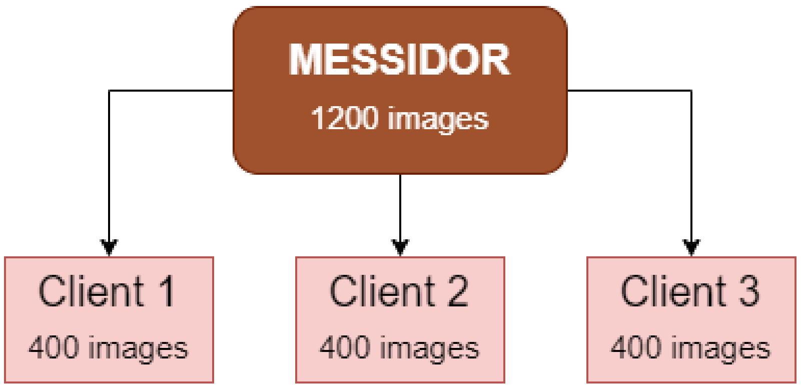


Figure 3.7: MESSIDOR divided between clients.

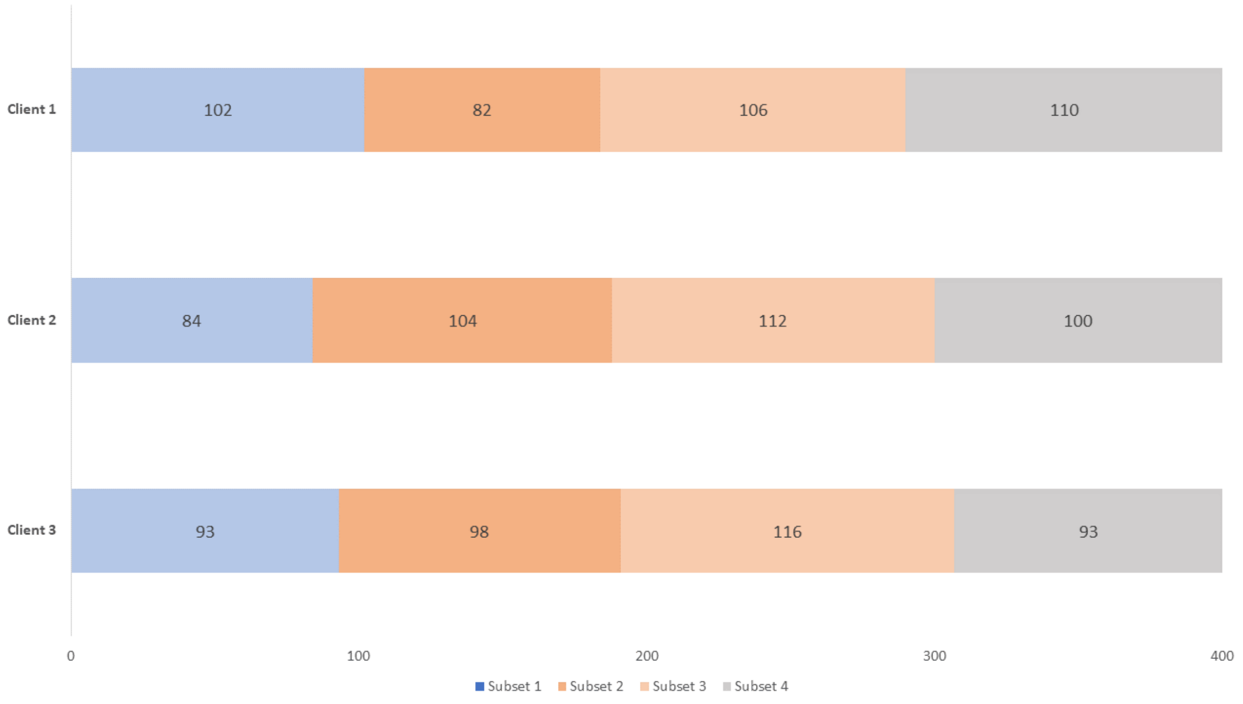


Figure 3.8: MESSIDOR divided by subsets.

Kaggle DR

As stated before, this dataset was chosen specifically due to its huge amount of data. This means that it is possible to create a scenario where some clients have a lot more images than others. Also due to its size, this dataset was distributed among ten virtual clients.

The division of the dataset between clients and the division by subsets are shown in Figures 3.9 and 3.10, respectively.

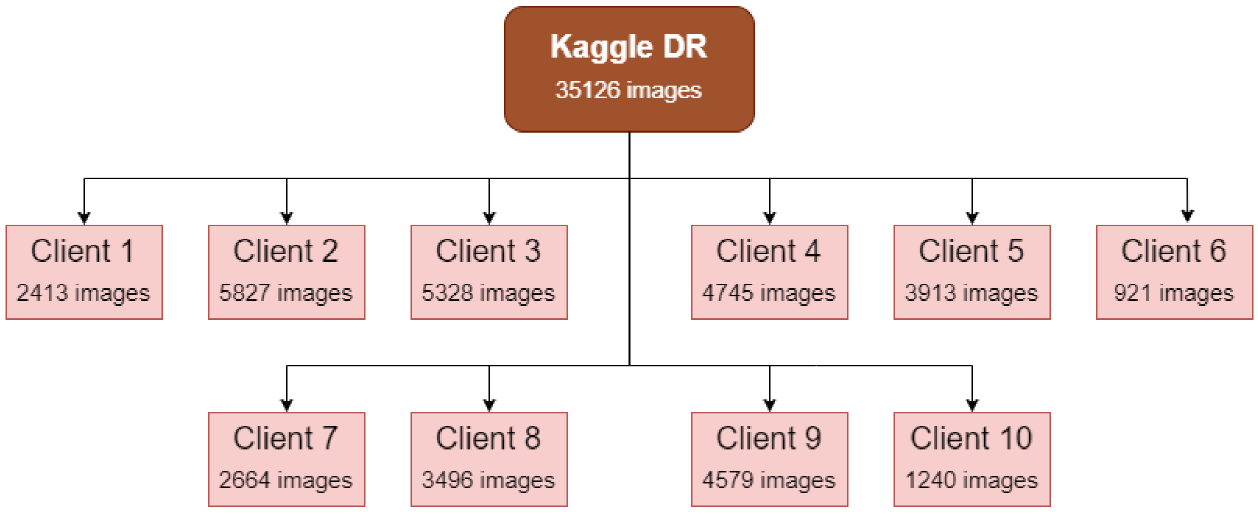


Figure 3.9: Kaggle DR divided between clients.

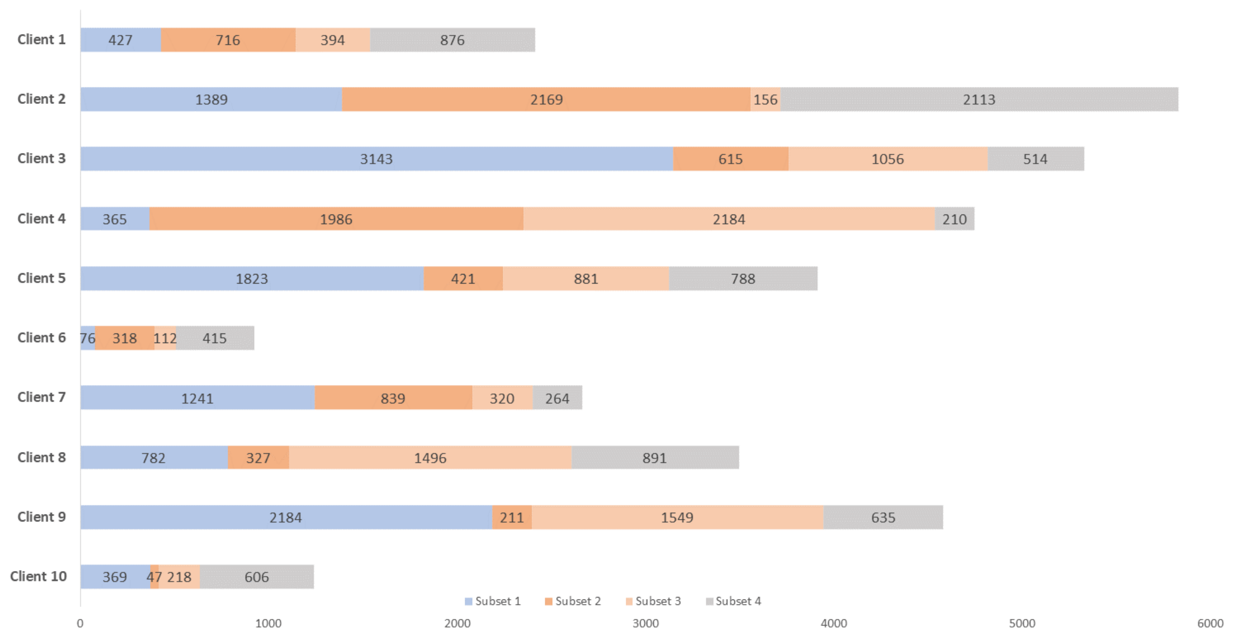


Figure 3.10: Kaggle DR divided by subsets.

3.2.2 Aggregation Algorithms

The success of model aggregation algorithms is critical in establishing a globally refined model in the context of FL. These algorithms dictate how individual clients' local model updates are integrated to build a better global model. Three distinct averaging methods have been implemented in this study: FedAvg, FedProx, and SCAFFOLD. Each method adds novel mechanisms to handle issues, with the main focus of this study being model performance and convergence. Below is a detailed regarding the implementation of this algorithms.

FedAvg

FedAvg is a key averaging algorithm in FL. It focuses on aggregating local model changes by computing an arithmetic average of them. This is done by accessing the client's model parameters and then calculating the mean of model parameters across all clients. This work used a variant of the traditional FedAvg algorithm, assigning a weighting factor to each client's update, based on the size of their dataset. This helps to address the problem of unbalanced datasets, where some clients may have much more data than others. To implement this, the contribution of each client was calculated by dividing the size of their dataset by the total size of all the datasets. Then, these contribution values were used as the weighting factors for the client updates. FedAvg overall is a reliable baseline for comparison with the following algorithms.

FedProx

To address the challenge of optimizing non-convex objectives in a distributed setting, the FedProx algorithm was employed. This technique augments the used loss function with a proximal term in order to incentivize local models to remain close to the global model. The calculation of this proximal term is done in accordance with Equation 2.5.

The variable denoted as μ within the equation can be dynamically adaptive or remain static. In this study, μ starts is initialized at 0.1 and adapted based on fluctuations in the loss. If the loss does not decline for five consecutive rounds, μ is divided by 0.1. otherwise it's multiplied by 0.01. $\|w - w^t\|$ signifies the norm of the difference between model parameters, with w being the local client's model and w^t the global model.

After obtaining the proximal term through these computations, the next step is to combine it with the result of the loss function. This integration serves as a safety, effectively limiting the local model update's divergence from the current global model. The FedProx algorithm harmonizes repetitive local updates while strengthening their affinity with the overall global model by coordinating this delicate interplay.

SCAFFOLD

The SCAFFOLD algorithm is a FL algorithm that is designed to be more robust to heterogeneous data and improve convergence rate. It does so by updating the local models on the clients based on the control variables of the server (*c_ global*) and the client (*c_ local*). The control variables are a set of parameters that are used to control the update process. The server and the client each have their own set of control variables, initially all set to zero.

When training the local model, two extra control variables are initialize: *c_ plus* and *c_ delta*. This variables are used to store the updated control variables. At the conclusion of each local training epoch, adjustments are made to individual local model parameters by adding the difference between the server's and client's respective control variables. This update process ensures that the local model is more aligned with the global model. Upon completing all training epochs, before transmitting the model to the server, *c_ plus* is computed using the formula:

$$c\_plus = c\_local - c\_global + \frac{1}{E \times lr \times (G - L)} \tag{3.3}$$

where E denotes the number of local epochs, lr represents the local optimizer's LR, G is the global model and L stands for the updated local model. Subsequently, *c_ delta* is updated by calculating the difference between *c_ plus* and *c_ local.* A final update is applied to the local model, based on its difference and the global model. Finally, the client's control variable (*c_ local*) is set to the value of *c_ plus.*

Post the training phase across all clients, the clients transmit their updated models, as per usual, as well as their corresponding *c_ delta* values. The server then computes the mean *c_ delta* originating from all clients and updates the global variable *c_ global* by adding both variables.

4 Experimental Results

In this chapter, the obtained experimental results will be presented and discussed. These results are divided into those obtained using the reference models trained locally, and those obtained using the developed FL framework.

4.1 Reference Models Results

In this section, the results obtained with the local reference model after improving it and by using Focal Loss are presented.

The first results to be presented were obtained on the three scenarios mentioned in [16] using the MESSIDOR dataset. The metrics used to evaluate the model are ROC curve along AUC as well as the accuracy for classifying each class.

The second results presented were obtained with the MESSIDOR and Kaggle DR datasets, after changing the problem to a binary classification, which is explained in Section 3.1.4. Here, the metrics presented will be ROC curve along AUC, as well as Specificity and Sensitivity.

Improved Model Results

Class	Accuracy
0	92.05 %
1	64.00 %
2	73.17 %
3	84.78 %

Table 4.1: Accuracy obtained for each of the four classes.

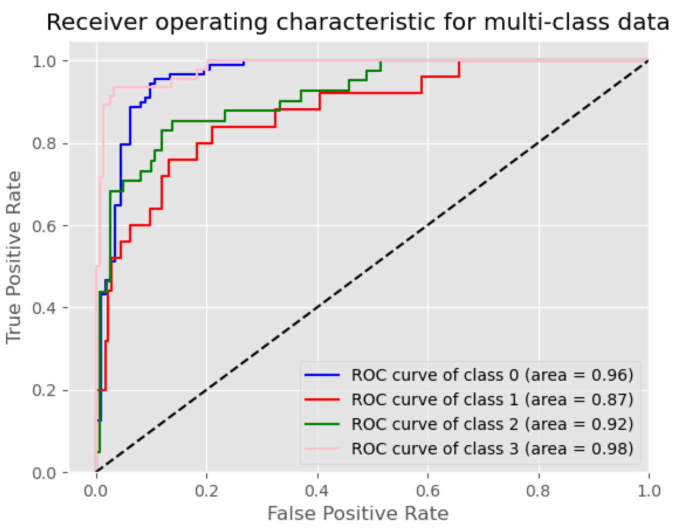


Figure 4.1: ROC curve for four classes classification.

Class	Accuracy
0 and 1	95.73 %
2	65.00 %
3	83.72 %

Table 4.2: Accuracy obtained for each of the three classes.

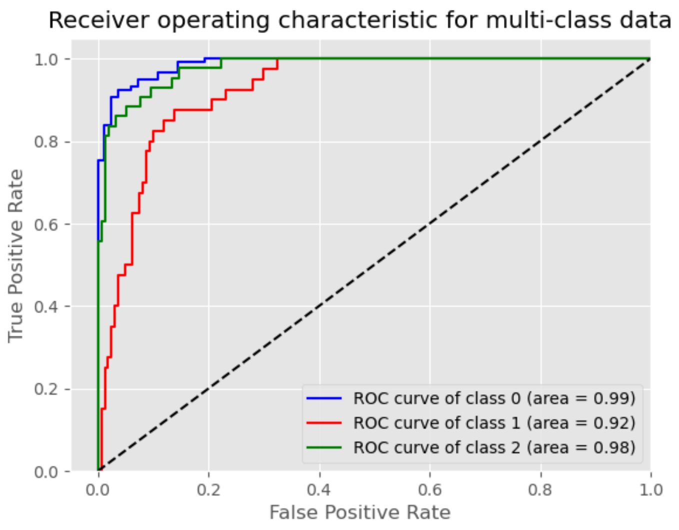


Figure 4.2: ROC curve for three classes classification.

Class	Accuracy
0	98.61 %
3	100.00 %

Table 4.3: Accuracy obtained for class 0 and 3.

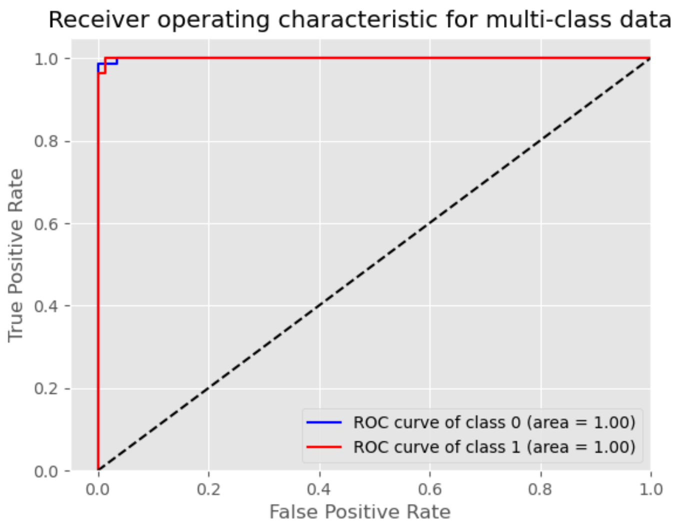


Figure 4.3: ROC curve for two classes classification .

Binary Classification Results

Specificity	Sensitivity
51.74 %	94.28 %

Table 4.4: Specificity and sensitivity obtained when training on the MESSIDOR dataset.

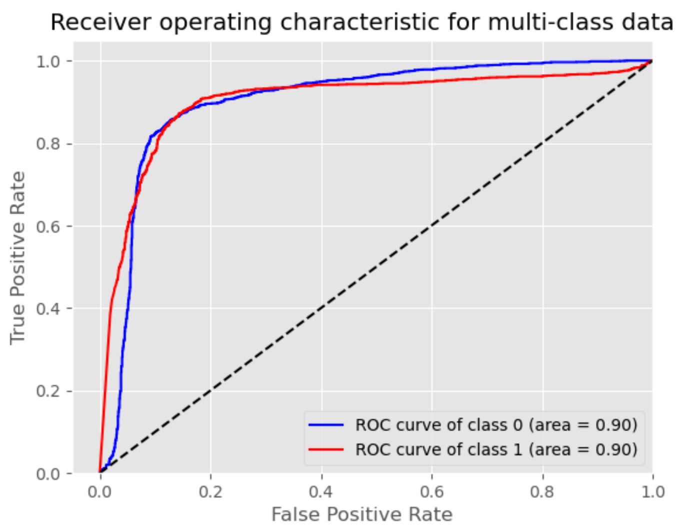


Figure 4.4: ROC curve for training on the MESSIDOR dataset.

Specificity	Sensitivity
85.28 %	95.09 %

Table 4.5: Specificity and sensitivity obtained when training on the Kaggle DR dataset.

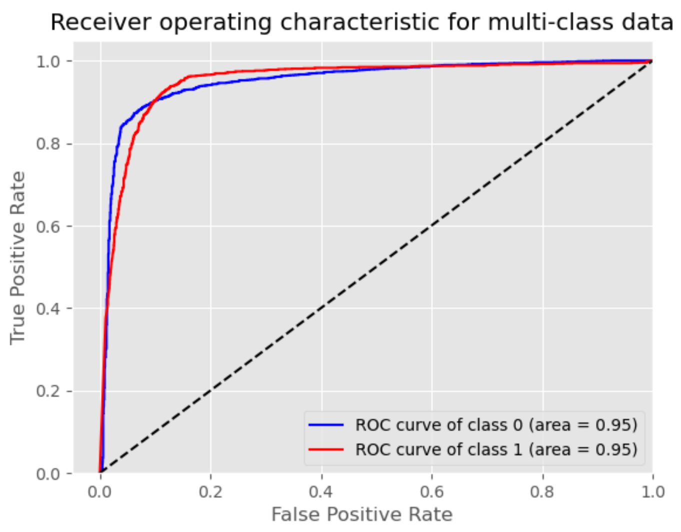


Figure 4.5: ROC curve for training on the Kaggle DR dataset.

4.2 Federated Learning Results

In this section, the results obtained with the developed FL framework are presented. The results shown were obtained training with the MESSIDOR and Kaggle DR dataset, and will include the two scenarios described in Section 3 using the three implemented aggregation algorithms. Here, the metrics taken into consideration are the evolution of the global model's accuracy, in order to see how each aggregation algorithm affects convergence speed, as well as specificity and sensitivity obtained on the final global model.

Training with the entire dataset

Specificity	Sensitivity
84.05 %	95.76 %

Table 4.6: Specificity and sensitivity obtained when training on the MESSIDOR dataset using the FedAvg algorithm.

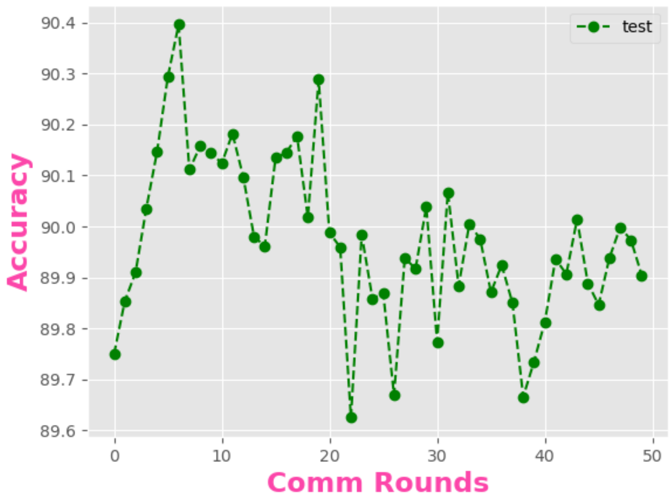


Figure 4.6: Accuracy obtained at every communication round using the FedAvg algorithm on the MESSIDOR dataset.

Specificity	Sensitivity
84.13 %	95.96 %

Table 4.7: Specificity and sensitivity obtained when training on the MESSIDOR dataset using the FedProx algorithm.

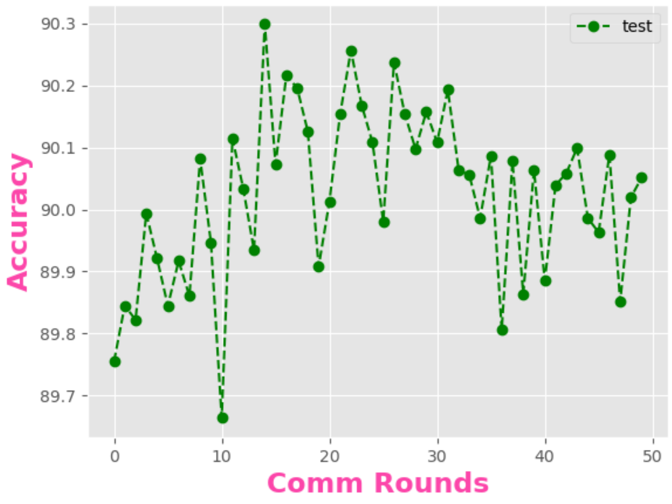


Figure 4.7: Accuracy obtained at every communication round using the FedProx algorithm on the MESSIDOR dataset.

Specificity	Sensitivity
84.41 %	95.36 %

Table 4.8: Specificity and sensitivity obtained when training on the MESSIDOR dataset using the SCAFFOLD algorithm.

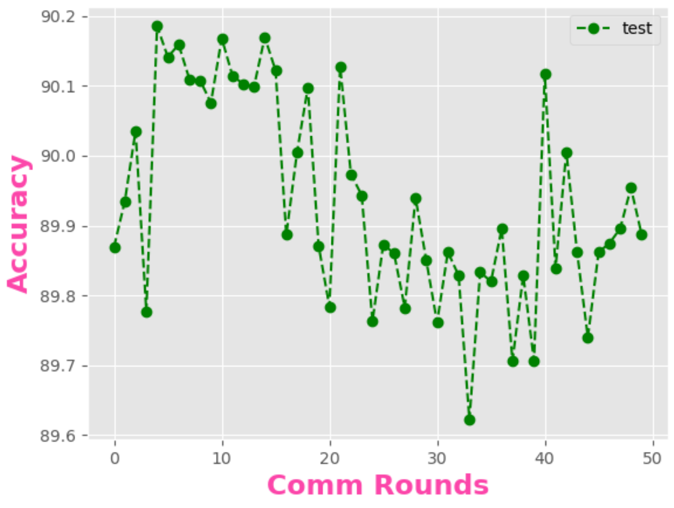


Figure 4.8: Accuracy obtained at every communication round using the SCAFFOLD algorithm on the MESSIDOR dataset.

Specificity	Sensitivity
85.38 %	97.18 %

Table 4.9: Specificity and sensitivity obtained when training on the Kaggle DR dataset using the FedAvg algorithm.

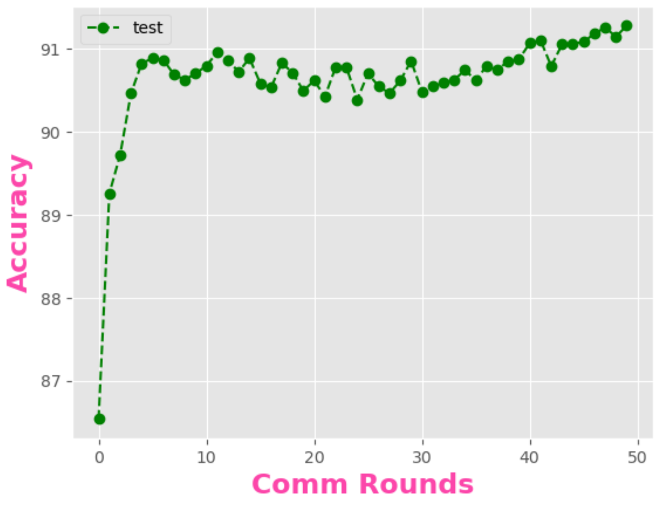


Figure 4.9: Accuracy obtained at every communication round using the FedAvg algorithm on the Kaggle DR dataset.

Specificity	Sensitivity
85.43 %	96.03 %

Table 4.10: Specificity and sensitivity obtained when training on the Kaggle DR dataset using the FedProx algorithm.

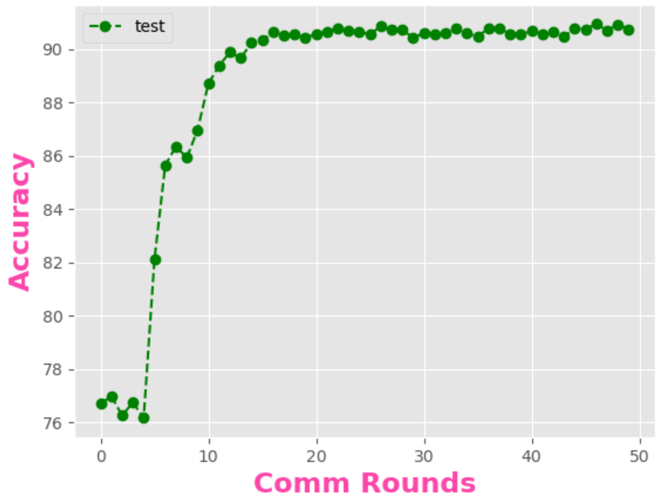


Figure 4.10: Accuracy obtained at every communication round using the FedProx algorithm on the Kaggle DR dataset.

Specificity	Sensitivity
85.74 %	96.44 %

Table 4.11: Specificity and sensitivity obtained when training on the Kaggle DR dataset using the SCAFFOLD algorithm.

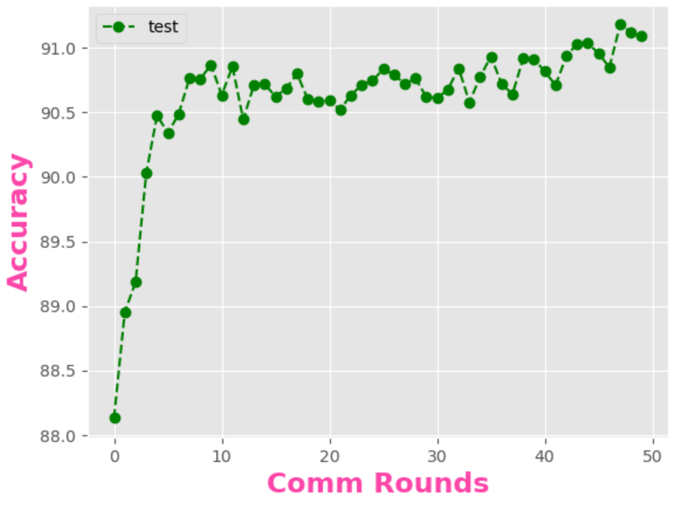


Figure 4.11: Accuracy obtained at every communication round using the SCAFFOLD algorithm on the Kaggle DR dataset.

Training with the dataset divided in subsets

Specificity	Sensitivity
84.36 %	94.96 %

Table 4.12: Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the FedAvg algorithm.

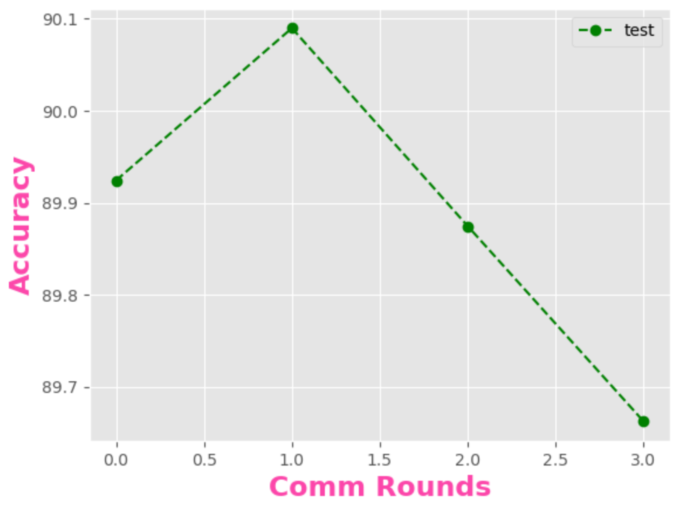


Figure 4.12: Accuracy obtained at every communication round using the FedAvg algorithm on the MESSIDOR dataset divided in subsets.

Specificity	Sensitivity
83.17 %	96.64 %

Table 4.13: Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the FedProx algorithm.

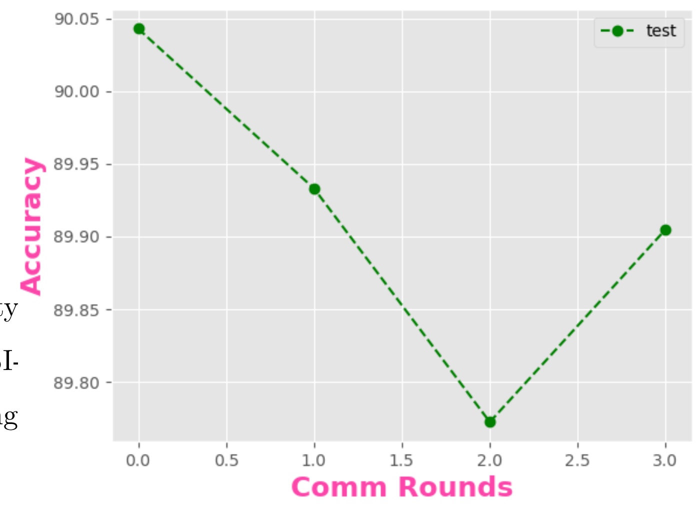


Figure 4.13: Accuracy obtained at every communication round using the FedProx algorithm on the MESSIDOR dataset divided in subsets.

Specificity	Sensitivity
84.83 %	94.55 %

Table 4.14: Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the SCAFFOLD algorithm.

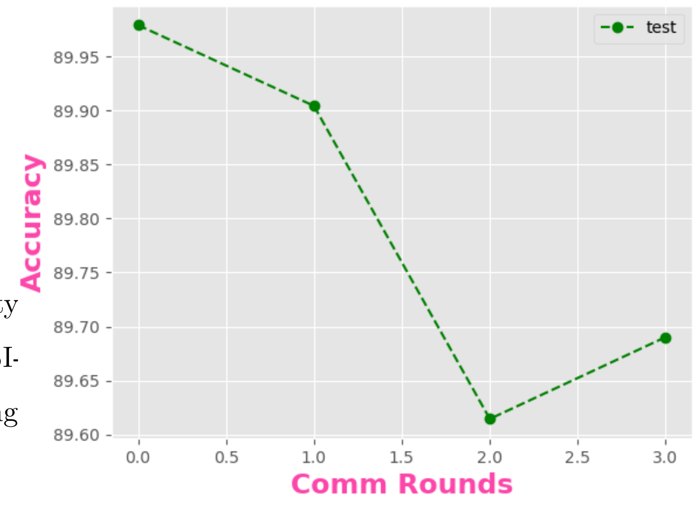


Figure 4.14: Accuracy obtained at every communication round using the SCAFFOLD algorithm on the MESSIDOR dataset divided in subsets.

Specificity	Sensitivity
88.0 %	91.93 %

Table 4.15: Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the FedAvg algorithm.

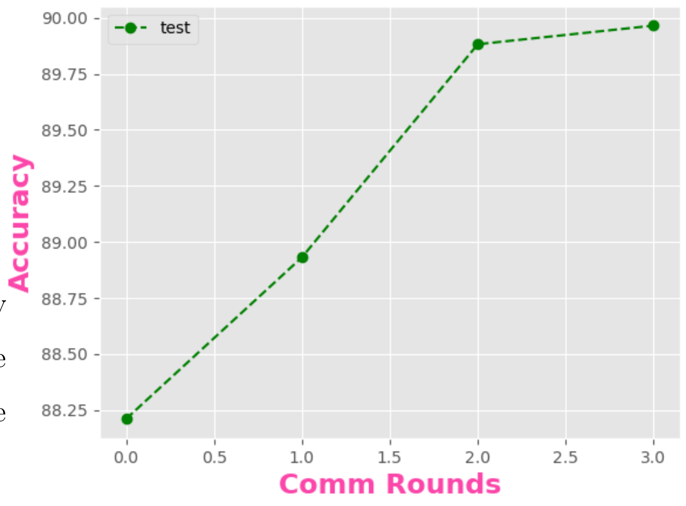


Figure 4.15: Accuracy obtained at every communication round using the FedAvg algorithm on the Kaggle DR dataset divided in subsets.

Specificity	Sensitivity
93.66 %	62.00 %

Table 4.16: Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the FedProx algorithm.

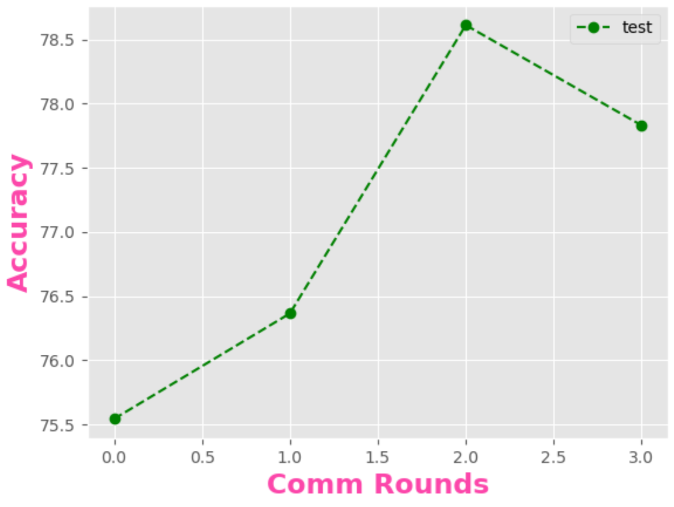


Figure 4.16: Accuracy obtained at every communication round using the FedProx algorithm on the Kaggle DR dataset divided in subsets.

Specificity	Sensitivity
87.08 %	92.87 %

Table 4.17: Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the SCAFFOLD algorithm.

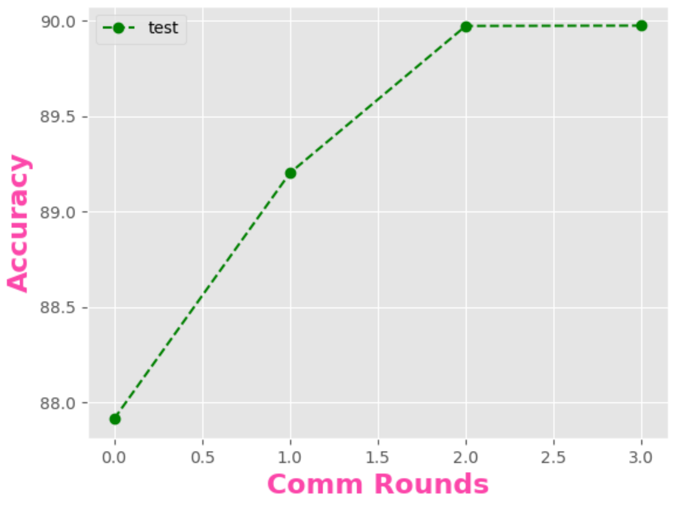


Figure 4.17: Accuracy obtained at every communication round using the SCAFFOLD algorithm on the Kaggle DR dataset divided in subsets.

4.3 Discussion

In terms of the outcomes achieved by the improved model, it is both obvious and expected that its performance improves when tasked with classifying fewer classes. While this result was predictable, it is crucial because one of the key goals of this work was to develop a model capable of distinguishing between RDR and NRDR cases.

From the results of training local reference models on the MESSIDOR and Kaggle DR datasets, it is clear that the final model's performance improves when trained on the Kaggle DR dataset, providing a Specificity of 85.28% and a Sensitivity of 95.09%. Given the significant difference in available data between the two datasets, this discrepancy is to be expected. These results, which serve as a comparative benchmark, provide a thorough foundation for assessing the possible viability of a FL implementation in terms of model performance.

When switching to FL trained models, a recurring pattern emerges: training using the Kaggle DR dataset produces better results than training with the MESSIDOR dataset. Aside from overall enhanced performance, it is clear that training on the Kaggle DR dataset results in smoother convergence. This is supported by the discernible stability shown in the accuracy variation over communication rounds in Figures 4.9, 4.10, and 4.11, as opposed to the fluctuations seen in MESSIDOR dataset training, as shown in Figures 4.6, 4.7, and 4.8.

Within the area of implemented aggregation algorithms, FedAvg primarily acts as a baseline for comparison with the other two techniques. FedProx, meant to improve local model alignment with the global model, has the slowest convergence rate and the least favorable overall performance. SCAFFOLD on the other hand, emerges as a potential alternative to FedAvg, exhibiting a slightly improved convergence rate as well as comparable Specificity and Sensitivity metrics.

An additional scenario was investigated to further expand the examination of the FL technique, in which each client's dataset was partitioned into subsets and used for training just once. This method minimizes the number of communication rounds, hence lowering communication expenses. The results of this approach imply that it is possible to train a FL model in which clients update their data during training without significantly affecting global model performance. This discovery adds another degree of practicality to the previously discussed FL approach.

5 Conclusion and Future Work

This in-depth research into using FL for DR detection has given extensive insights that bridge the gap between distributed learning and medical image analysis. This study has yielded numerous results that have relevance for both research and practical applications after a comprehensive analysis of various approaches. In terms of model performance and adaptability, the direct comparison of locally trained reference models with the FL paradigm illustrates the advantages of collaborative learning.

The implementation and evaluation of several aggregation methods, such as FedProx and SCAFFOLD, highlight the importance of controlled randomness and regularization in guiding the convergence dynamics of distributed models. The effects on convergence speed, stability, and accuracy observed give light on the complex interplay between aggregation approaches and model refining.

FL showcases potential not only in terms of model performance but also in optimizing communication and resource usage. This was demonstrated on the simulated scenario where each client only trains the model with part of its dataset at each communication round without ever using that data again. The results obtained in this scenario proved that is possible to reduce the amount of data that each client uses, therefore reducing computation and communication costs.

Furthermore, this research has highlighted the significance of data properties in shaping model outcomes. The disparities in performance between the MESSIDOR and Kaggle DR datasets highlight the importance of dataset size, diversity, and relevance in FL model success.

To further advance the field of using FL for the detection of DR, the following future work directions are recommended:

- **Enhanced Aggregation Techniques**: By researching hybrid approaches that combine the capabilities of several techniques, it is possible to improve aggregation algorithms. This could result in faster convergence rates and increased model robustness.
- **Privacy-Preserving Mechanisms**: Integrating privacy-preserving techniques such as differential privacy or secure aggregation might alleviate privacy issues, making FL more desirable for healthcare applications.
- **Real-World Clinical Validation**: Collaboration with medical professionals to test the

developed FL models on real-world patient data will bridge the gap between research and clinical practice, confirming the models' robustness and reliability in practical situations.

- **Resource-Efficiency**: Exploring strategies to optimize computational resources during FL training, such as model compression techniques, can reduce resource burdens and make FL more accessible.

Bibliography

[1] Cleveland Clinic medical professional. *Diabetes.* Accessed: 26/06/2023. Feb. 2023. URL: `https://my.clevelandclinic.org/health/diseases/7104-diabetes`.

[2] Natalie Sainz. *2021 diabetes atlas numbers.* Accessed: 26/06/2023. Feb. 2022. URL: `https://diatribechange.org/news/2021-diabetes-atlas-numbers`.

[3] Alan W. Stitt et al. "The progress in understanding and treatment of diabetic retinopathy". In: *Progress in Retinal and Eye Research* 51 (2016), pp. 156–186. ISSN: 1350-9462. DOI: `https://doi.org/10.1016/j.preteyeres.2015.08.001`.

[4] National Health Service. *Diabetic retinopathy.* Accessed: 26/06/2023. 2021. URL: `https://www.nhs.uk/conditions/diabetic-retinopathy/`.

[5] Ning Cheung and Tien Y Wong. "Diabetic retinopathy and systemic vascular complications". In: *Progress in retinal and eye research* 27.2 (2008), pp. 161–176. DOI: `https://doi.org/10.1016/j.preteyeres.2007.12.001`.

[6] Zhen Ling Teo et al. "Global prevalence of diabetic retinopathy and projection of burden through 2045: systematic review and meta-analysis". In: *Ophthalmology* 128.11 (2021), pp. 1580–1591. DOI: `https://doi.org/10.1016/j.ophtha.2021.04.027`.

[7] Ji-Peng Olivia Li et al. "Digital technology, tele-medicine and artificial intelligence in ophthalmology: A global perspective". In: *Progress in retinal and eye research* 82 (2021), p. 100900. DOI: `https://doi.org/10.1016/j.preteyeres.2020.100900`.

[8] Daniel SW Ting et al. "Deep learning in ophthalmology: the technical and clinical considerations". In: *Progress in retinal and eye research* 72 (2019), p. 100759. DOI: `https://doi.org/10.1016/j.preteyeres.2019.04.003`.

[9] Daniel Shu Wei Ting et al. "Artificial intelligence and deep learning in ophthalmology". In: *British Journal of Ophthalmology* 103.2 (2019), pp. 167–175. DOI: `https://doi.org/10.1136/bjophthalmol-2018-313173`.

[10] Sawsan AbdulRahman et al. "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond". In: *IEEE Internet of Things Journal* 8.7 (2020), pp. 5476–5497. DOI: `https://doi.org/10.1109/JIOT.2020.3030072`.

[11] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR. Apr. 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html.

[12] Mohaimen Al-Zubaidy. *How to perform fundoscopy with a direct ophthalmoscope*. Accessed: 06/07/2023. Nov. 2020. URL: https://jfophth.com/how-to-perform-fundoscopy-with-a-direct-ophthalmoscope/.

[13] MGF Gilliland and Robert Folberg. "Retinal hemorrhages: replicating the clinician's view of the eye". In: *Forensic science international* 56.1 (1992), pp. 77–80. DOI: https://doi.org/10.1016/0379-0738(92)90149-Q.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: 25 (2012). Ed. by F. Pereira et al. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[16] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444. DOI: https://doi.org/10.1038/nature14539.

[18] Star Retina. *Diabetic retinopathy*. Accessed: 10/07/2023. Feb. 2021. URL: https://starretina.com/diabetic-retinopathy/.

[19] Jason Adam. *ResNet-50 API*. Accessed: 11/07/2023. Dec. 2019. URL: https://jason-adam.github.io/resnet50/.

[20] Sergey Aksenov. *What is supervised learning?: Machine learning tasks [updated 2023]*. Accessed: 14/8/2023. May 2023. URL: https://www.superannotate.com/blog/supervised-learning-and-other-machine-learning-tasks#supervised-learning-explained.

[21] Scott Martin. *CNN vs. RNN. what's the difference?* Accessed: 27/06/2023. Sept. 2018. URL: https://blogs.nvidia.com/blog/2018/09/05/whats-the-difference-between-a-cnn-and-an-rnn/.

[22] Muhamad Yani, Budhi Irawan, and Casi Setiningsih. “Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry’s Nail”. In: *Journal of Physics: Conference Series* 1201 (May 2019), p. 012052. DOI: https://doi.org/10.1088/1742-6596/1201/1/012052.

[23] Study Machine Learning. Accessed: 12/07/2023. URL: https://studymachinelearning.com/activation-functions-in-neural-network/.

[24] Charlotte Pelletier, Geoffrey I Webb, and François Petitjean. “Temporal convolutional neural network for the classification of satellite image time series”. In: *Remote Sensing* 11.5 (2019), p. 523. DOI: https://doi.org/10.3390/rs11050523.

[25] Amar Budhiraja. *Dropout in (Deep) Machine learning.* Accessed: 17/07/2023. Dec. 2016. URL: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[26] Truong X Nguyen et al. “Federated learning in ocular imaging: current progress and future direction”. In: *Diagnostics* 12.11 (2022), p. 2835. DOI: https://doi.org/10.3390/diagnostics12112835.

[27] Nicola Rieke. *What is federated learning?* Accessed: 05/07/2023. 2022. URL: https://blogs.nvidia.com/blog/2019/10/13/what-is-federated-learning/.

[28] Tian Li et al. “Federated learning: Challenges, methods, and future directions”. In: *IEEE signal processing magazine* 37.3 (2020), pp. 50–60. DOI: https://doi.org/10.1109/MSP.2020.2975749.

[29] Gaudenz Boesch. *An introduction to federated learning: Challenges and applications.* Accessed: 22/07/2023. 2023. URL: https://viso.ai/deep-learning/federated-learning/.

[30] Z. Iqbal and H.Y. Chan. “Concepts, Key Challenges and Open Problems of Federated Learning”. In: *International Journal of Engineering* 34.7 (2021), pp. 1667–1683. ISSN: 1025-2495. DOI: https://doi.org/10.5829/ije.2021.34.07a.11.

[31] Chao Huang, Jianwei Huang, and Xin Liu. “Cross-silo federated learning: Challenges and opportunities”. In: *arXiv preprint arXiv:2206.12949* (2022). DOI: https://doi.org/10.48550/arXiv.2206.12949.

[32] Truong X. Nguyen et al. “Federated Learning in Ocular Imaging: Current Progress and Future Direction”. In: *Diagnostics* 12.11 (2022). ISSN: 2075-4418. DOI: https://doi.org/10.3390/diagnostics12112835.

[33] Sai Praneeth Karimireddy et al. “SCAFFOLD: Stochastic Controlled Averaging for Federated Learning”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 5132–5143. URL: https://proceedings.mlr.press/v119/karimireddy20a.html.

[34] Tian Li et al. “Federated Optimization in Heterogeneous Networks”. In: *Proceedings of Machine Learning and Systems*. Ed. by I. Dhillon, D. Papailiopoulos, and V. Sze. Vol. 2. 2020, pp. 429–450. URL: https://proceedings.mlsys.org/paper_files/paper/2020/file/1f5fe83998a09396ebe6477d9475ba0c-Paper.pdf.

[35] Hongyi Wang et al. “Federated Learning with Matched Averaging”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=BkluqlSFDS.

[36] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. “Adaptive Personalized Federated Learning”. In: (Mar. 2020). URL: https://openreview.net/forum?id=g0a-XYjpQ7r.

[37] A. Tuan Nguyen, Philip Torr, and Ser-Nam Lim. “FedSR: A Simple and Effective Domain Generalization Method for Federated Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=mrt90D00aQX.

[38] Mobeen-ur-Rehman et al. “Classification of Diabetic Retinopathy Images Based on Customised CNN Architecture”. In: *2019 Amity International Conference on Artificial Intelligence (AICAI)*. 2019, pp. 244–248. DOI: https://doi.org/10.1109/AICAI.2019.8701231.

[39] T. Shanthi and R.S. Sabeenian. “Modified Alexnet architecture for classification of diabetic retinopathy images”. In: *Computers & Electrical Engineering* 76 (2019), pp. 56–64. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2019.03.004.

[40] Francisco J. Martinez-Murcia et al. “Deep residual transfer learning for automatic diagnosis and grading of diabetic retinopathy”. In: *Neurocomputing* 452 (2021), pp. 424–434. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.04.148.

[41] Petrisia Widyasari Sudarmadji, Prisca Deviani Pakan, and Rocky Yefrenes Dillak. “Diabetic Retinopathy Stages Classification using Improved Deep Learning”. In: *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*. 2020, pp. 104–109. DOI: https://doi.org/10.1109/ICIMCIS51567.2020.9354281.

[42] Mohammad Nasajpour et al. “Federated transfer learning for diabetic retinopathy detection using CNN architectures”. In: *SoutheastCon 2022*. IEEE. 2022, pp. 655–660. DOI: https://doi.org/10.1109/SoutheastCon48659.2022.9764031.

[43] Julian Lo et al. “Federated learning for microvasculature segmentation and diabetic retinopathy classification of OCT data”. In: *Ophthalmology Science* 1.4 (2021), p. 100069. DOI: https://doi.org/10.1016/j.xops.2021.100069.

[44] Sina Gholami et al. “Federated Learning for Diagnosis of Age-related Macular Degeneration”. In: *bioRxiv* (2023). DOI: https://doi.org/10.1101/2023.07.06.547937. eprint: https://www.biorxiv.org/content/early/2023/07/15/2023.07.06.547937.full.pdf.

[45] Chun-Mei Feng et al. “Specificity-Preserving Federated Learning for MR Image Reconstruction”. In: *IEEE transactions on medical imaging* PP (Aug. 2022). DOI: https://doi.org/10.1109/TMI.2022.3202106.

[46] Etienne Decencière et al. “FEEDBACK ON A PUBLICLY DISTRIBUTED IMAGE DATABASE: THE MESSIDOR DATABASE”. In: *Image Analysis & Stereology* 33.3 (2014), pp. 231–234. ISSN: 1854-5165. DOI: https://doi.org/10.5566/ias.1155.

[47] Emma Dugas et al. *Diabetic Retinopathy Detection.* 2015. URL: https://kaggle.com/competitions/diabetic-retinopathy-detection.

[48] Sohier Dane, Maggie, and Karthik. *APTOS 2019 Blindness Detection.* 2019. URL: https://kaggle.com/competitions/aptos2019-blindness-detection.

[49] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition.* Ieee. 2009, pp. 248–255.

[50] Nitish Shirish Keskar and Richard Socher. “Improving generalization performance by switching from adam to sgd”. In: *arXiv preprint arXiv:1712.07628* (2017).

[51] Tsung-Yi Lin et al. “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 2980–2988.

Appendix A

Results Obtained Improving the Local Model

Class	Accuracy
0	72.73 %
1	40.00 %
2	51.22 %
3	80.43 %

Table A.1: Accuracy obtained for each of the four classes after changing the loss function to CE.

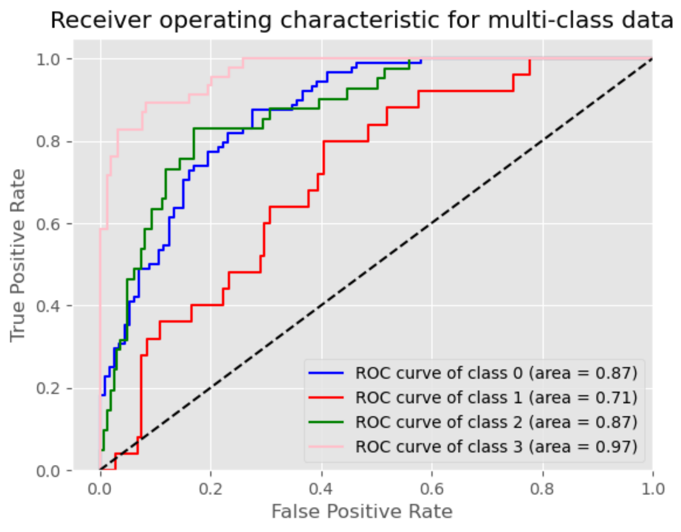


Figure A.1: ROC curve for four classes classification after changing the loss function to CE.

Class	Accuracy
0	81.82 %
1	24.00 %
2	43.90 %
3	78.26 %

Table A.2: Accuracy obtained for each of the four classes with BS = 2.

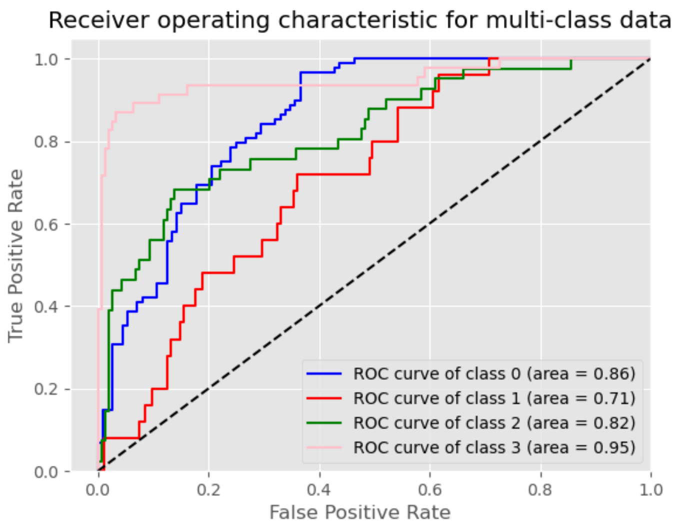


Figure A.2: ROC curve for four classes classification with BS = 2.

Class	Accuracy
0	85.23 %
1	32.00 %
2	56.10 %
3	71.74 %

Table A.3: Accuracy obtained for each of the four classes with BS = 8.

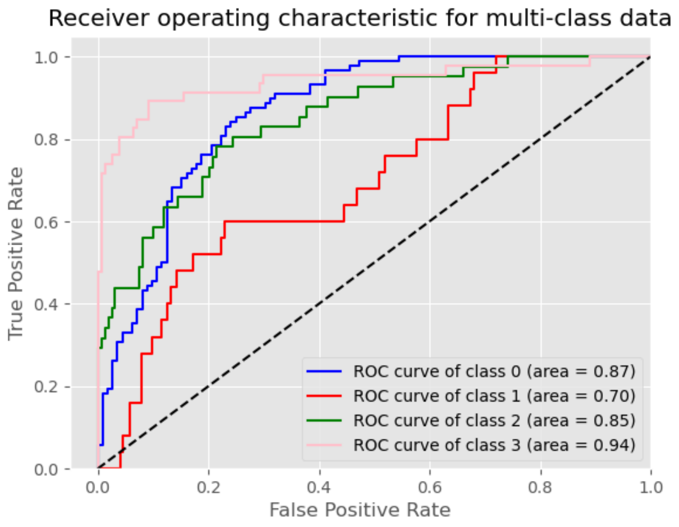


Figure A.3: ROC curve for four classes classification with BS = 8.

Class	Accuracy
0	95.45 %
1	44.00 %
2	75.61 %
3	76.09 %

Table A.4: Accuracy obtained for each of the four classes training with the full architecture.

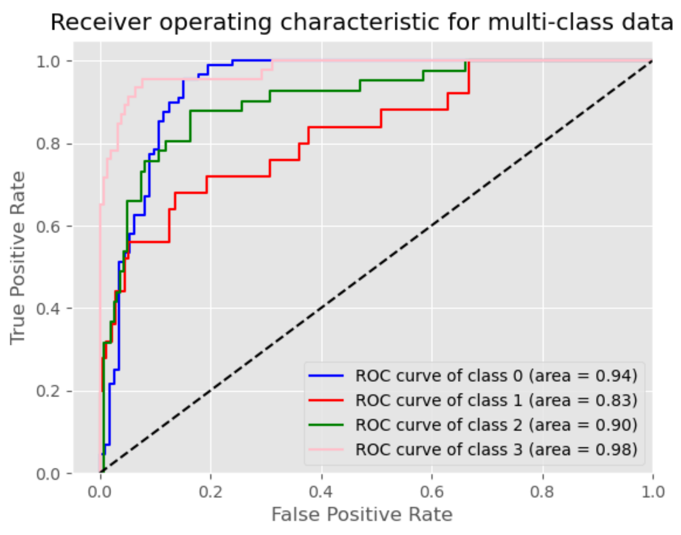


Figure A.4: ROC curve for four classes classification training with the full architecture.

Class	Accuracy
0	98.86 %
1	8.00 %
2	78.05 %
3	73.91 %

Table A.5: Accuracy obtained for each of the four classes with Adam optimizer and LR = 0.0001.

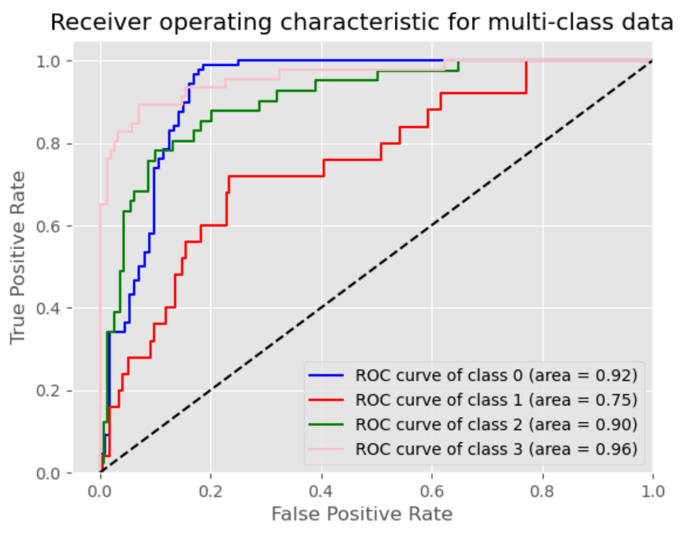


Figure A.5: ROC curve for four classes classification with Adam optimizer and LR = 0.0001.

Class	Accuracy
0	90.91 %
1	28.00 %
2	56.10 %
3	69.57 %

Table A.6: Accuracy obtained for each of the four classes with increased data augmentation.

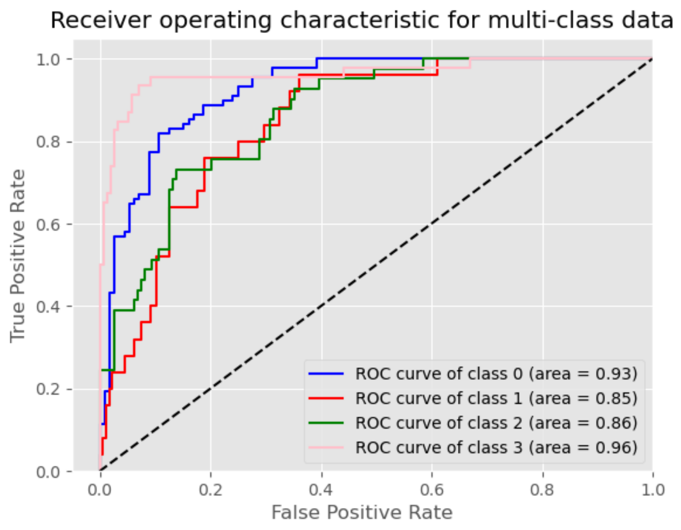


Figure A.6: ROC curve for four classes classification with increased data augmentation.

Class	Accuracy
0	98.86 %
1	48.00 %
2	68.29 %
3	82.61 %

Table A.7: Accuracy obtained for each of the four classes with an added dropout layer.

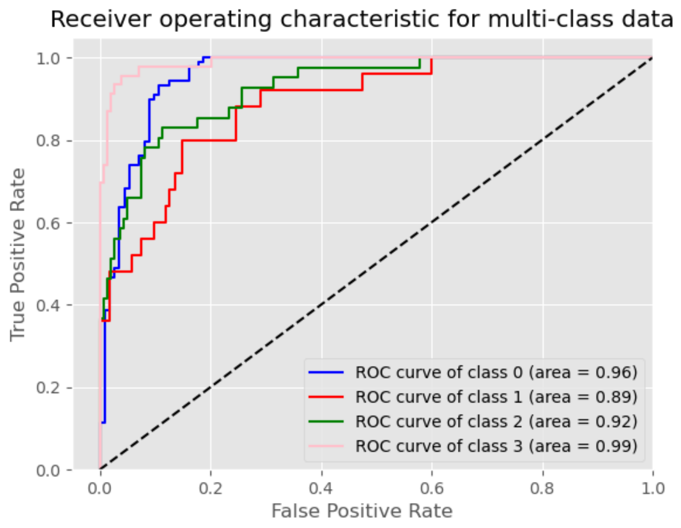


Figure A.7: ROC curve for four classes classification with an added dropout layer.

Class	Accuracy
0	94.32 %
1	20.00 %
2	63.41 %
3	80.43 %

Table A.8: Accuracy obtained for each of the four classes with CLAHE added to the preprocessing stage.

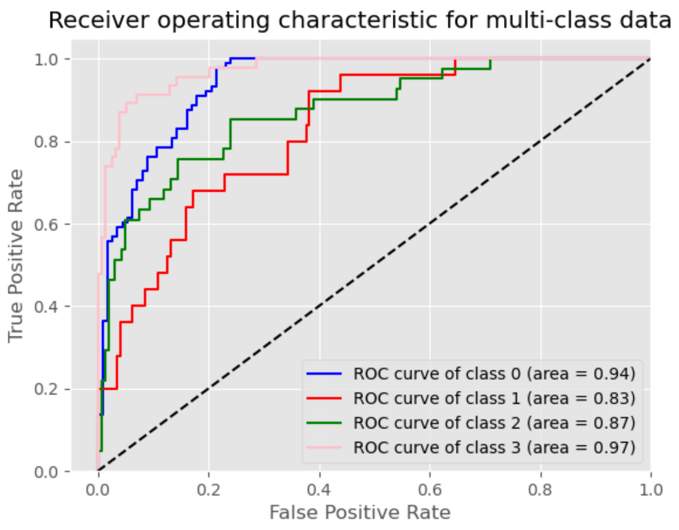


Figure A.8: ROC curve for four classes classification with CLAHE added to the preprocessing stage.

Class	Accuracy
0	86.36 %
1	48.00 %
2	68.29 %
3	78.26 %

Table A.9: Accuracy obtained for each of the four classes with Class Weighting.

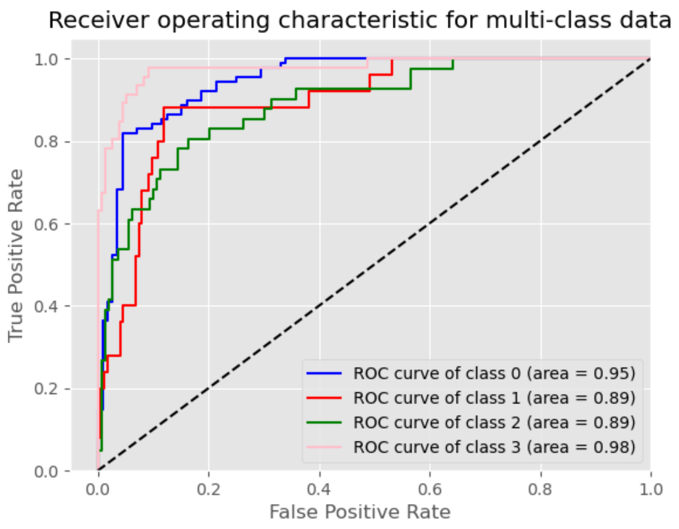


Figure A.9: ROC curve for four classes classification with Class Weighting.

Class	Accuracy
0	95.45 %
1	36.00 %
2	68.29 %
3	82.61 %

Table A.10: Accuracy obtained for each of the four classes with Oversampling.

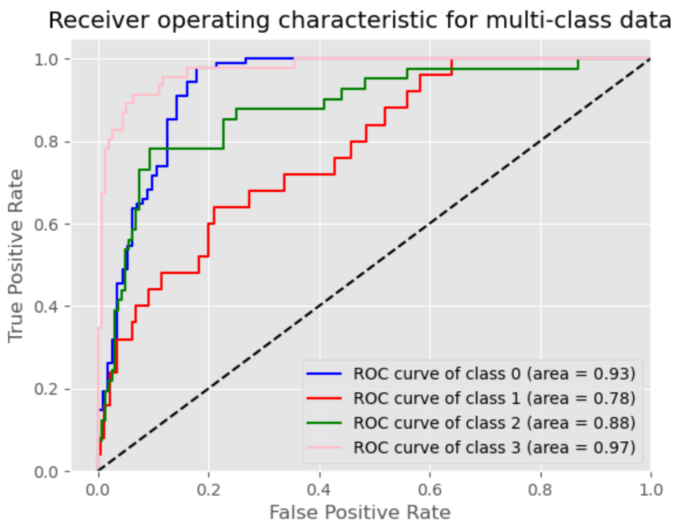


Figure A.10: ROC curve for four classes classification with Oversampling.

Class	Accuracy
0	86.36 %
1	32.00 %
2	78.05 %
3	71.74 %

Table A.11: Accuracy obtained for each of the four classes with Focal Loss $\gamma = 2$.

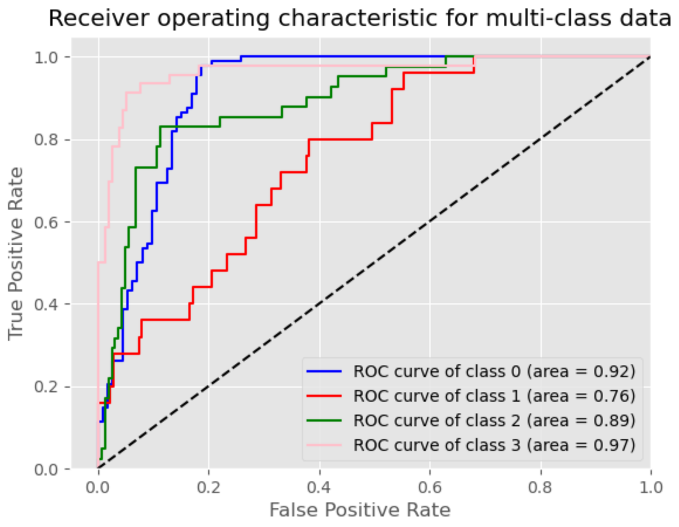


Figure A.11: ROC curve for four classes classification with Focal Loss $\gamma = 2$.

Class	Accuracy
0	82.95 %
1	68.00 %
2	75.61 %
3	86.96 %

Table A.12: Accuracy obtained for each of the four classes with Focal Loss $\gamma = 2.5$.

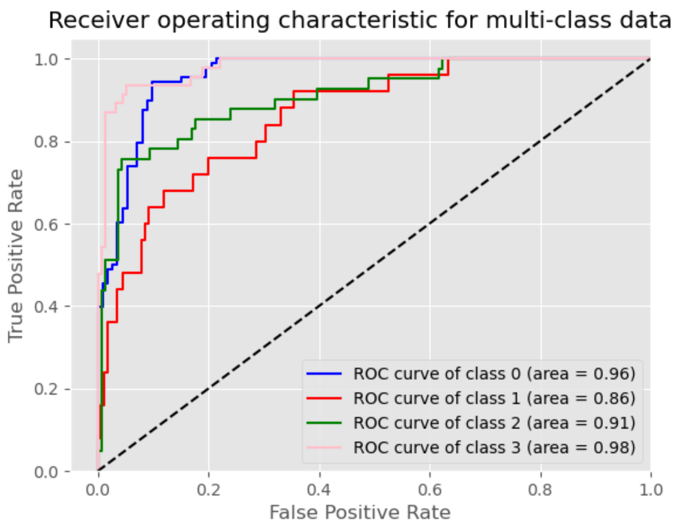


Figure A.12: ROC curve for four classes classification with Focal Loss $\gamma = 2.5$.

Class	Accuracy
0	84.09 %
1	64.00 %
2	68.29 %
3	84.78 %

Table A.13: Accuracy obtained for each of the four classes with Focal Loss $\gamma = 3$.

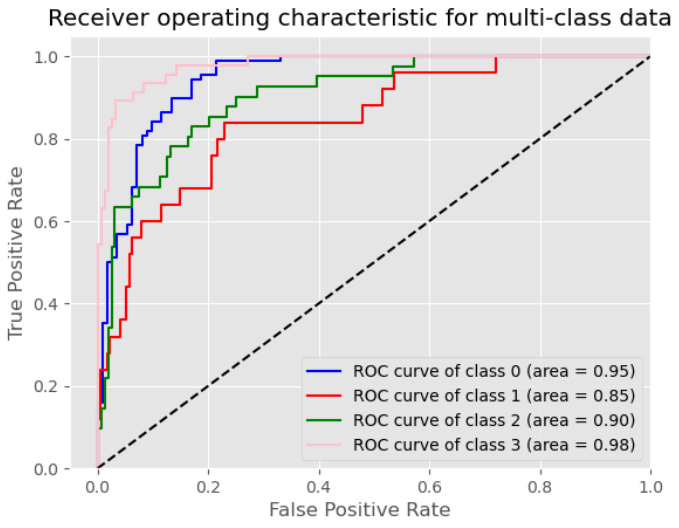


Figure A.13: ROC curve for four classes classification with Focal Loss $\gamma = 3$.

Class	Accuracy
0	85.23 %
1	56.00 %
2	73.17 %
3	84.78 %

Table A.14: Accuracy obtained for each of the four classes with Focal Loss $\gamma = 3.5$.

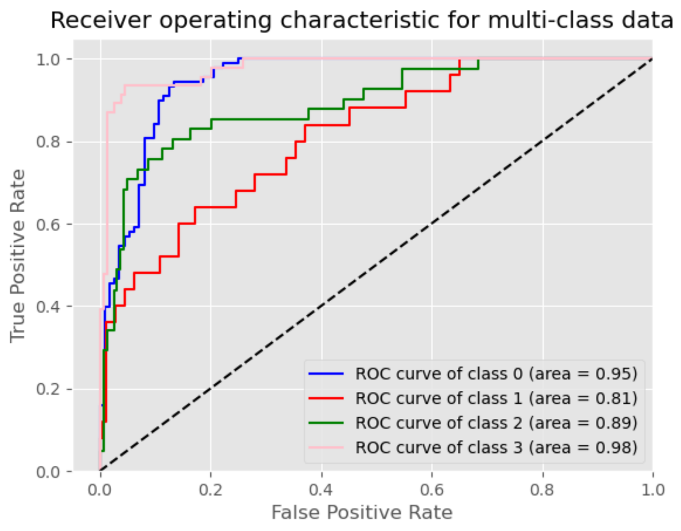


Figure A.14: ROC curve for four classes classification with Focal Loss $\gamma = 3.5$.

Class	Accuracy
0	90.91 %
1	56.00 %
2	65.85 %
3	89.13 %

Table A.15: Accuracy obtained for each of the four classes with Focal Loss $\gamma = 4$.

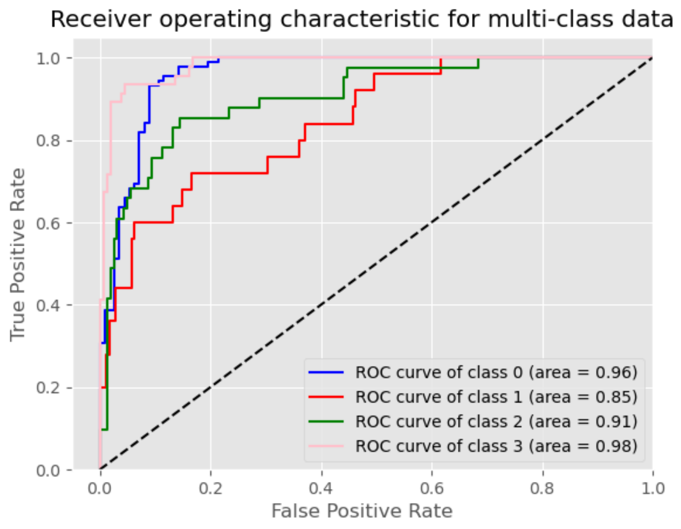


Figure A.15: ROC curve for four classes classification with Focal Loss $\gamma = 4$.

Appendix B

Other Federated Learning Results

Training with the entire dataset

Specificity	Sensitivity
83.62 %	68.54 %

Table B.1: Specificity and sensitivity obtained when training on the MESSIDOR dataset using the FedAvg algorithm and testing on the Kaggle DR dataset.

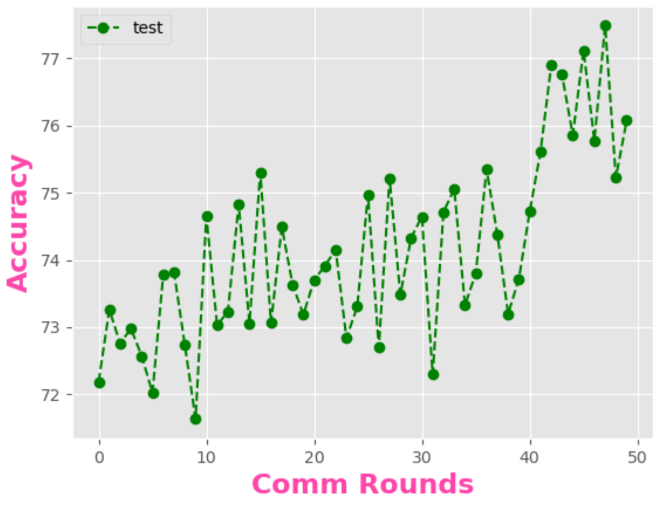


Figure B.1: Accuracy obtained at every communication round using the FedAvg algorithm on the MESSIDOR dataset and testing on the Kaggle DR dataset.

Specificity	Sensitivity
83.76 %	72.01 %

Table B.2: Specificity and sensitivity obtained when training on the MESSIDOR dataset using the FedProx algorithm and testing on the Kaggle DR dataset.

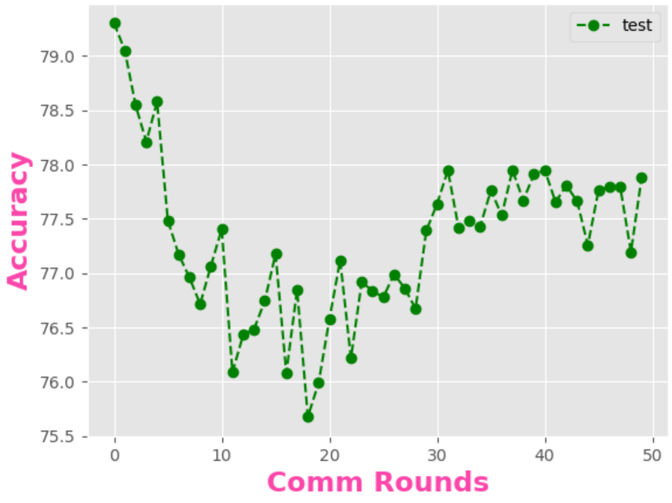


Figure B.2: Accuracy obtained at every communication round using the FedProx algorithm on the MESSIDOR dataset and testing on the Kaggle DR dataset.

Specificity	Sensitivity
77.94 %	72.97 %

Table B.3: Specificity and sensitivity obtained when training on the MESSIDOR dataset using the SCAFFOLD algorithm and testing on the Kaggle DR dataset.

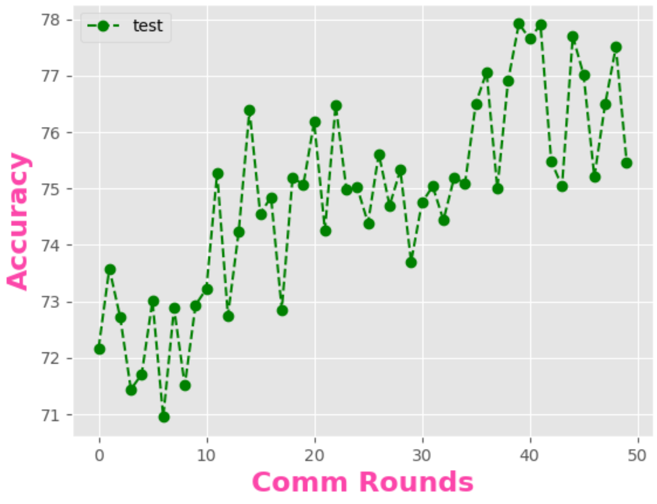


Figure B.3: Accuracy obtained at every communication round using the SCAFFOLD algorithm on the MESSIDOR dataset and testing on the Kaggle DR dataset.

Specificity	Sensitivity
100.00 %	75.65 %

Table B.4: Specificity and sensitivity obtained when training on the Kaggle DR dataset using the FedAvg algorithm and testing on the MESSIDOR dataset.

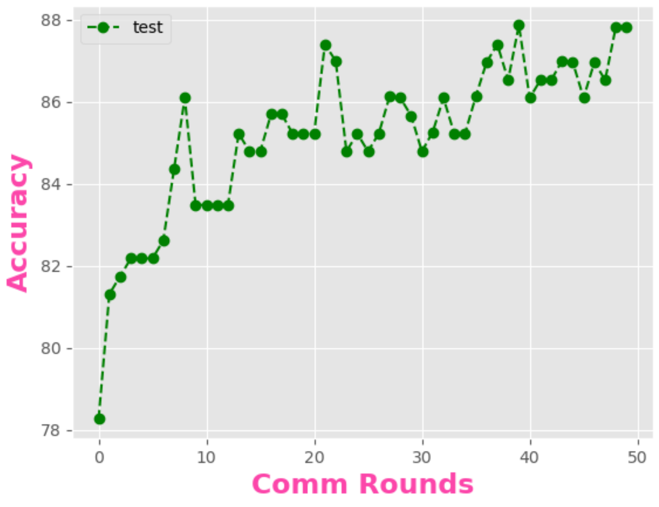


Figure B.4: Accuracy obtained at every communication round using the FedAvg algorithm on the Kaggle DR dataset and testing on the MESSIDOR dataset.

Specificity	Sensitivity
100.00 %	74.78%

Table B.5: Specificity and sensitivity obtained when training on the Kaggle DR dataset using the FedProx algorithm and testing on the MESSIDOR dataset.

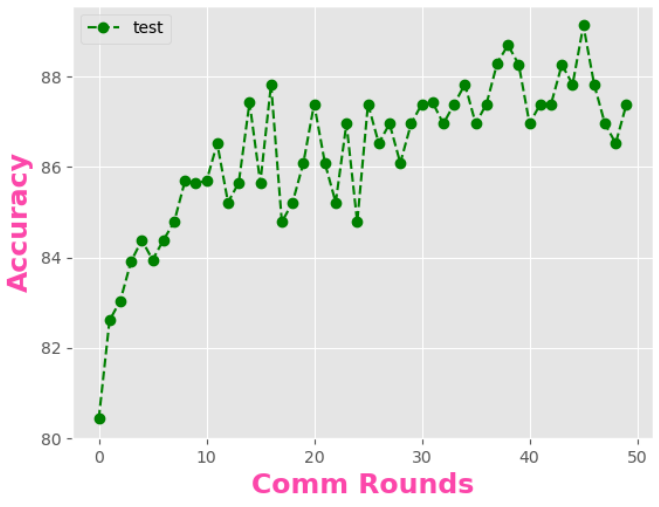


Figure B.5: Accuracy obtained at every communication round using the FedProx algorithm on the Kaggle DR dataset and testing on the MESSIDOR dataset.

Specificity	Sensitivity
100.00 %	75.65 %

Table B.6: Specificity and sensitivity obtained when training on the Kaggle DR dataset using the SCAFFOLD algorithm and testing on the MESSIDOR dataset.

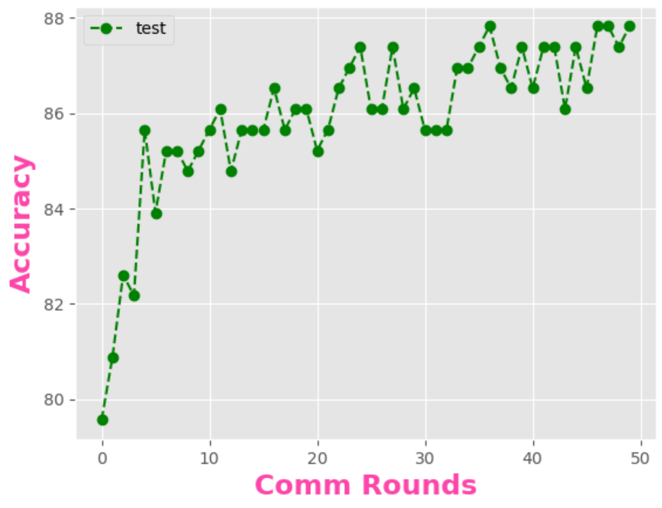


Figure B.6: Accuracy obtained at every communication round using the SCAFFOLD algorithm on the Kaggle DR dataset and testing on the MESSIDOR dataset.

Training with the dataset divided in subsets

Specificity	Sensitivity
82.55 %	70.83 %

Table B.7: Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the FedAvg algorithm and testing on the Kaggle DR dataset.

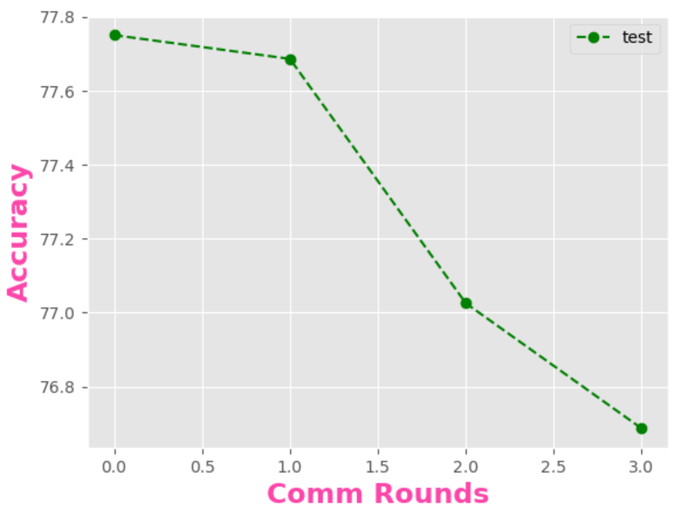


Figure B.7: Accuracy obtained at every communication round using the FedAvg algorithm on the MESSIDOR dataset divided in subsets and testing on the Kaggle DR dataset

Specificity	Sensitivity
83.32 %	73.26 %

Table B.8: Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the FedProx algorithm and testing on the Kaggle DR dataset.

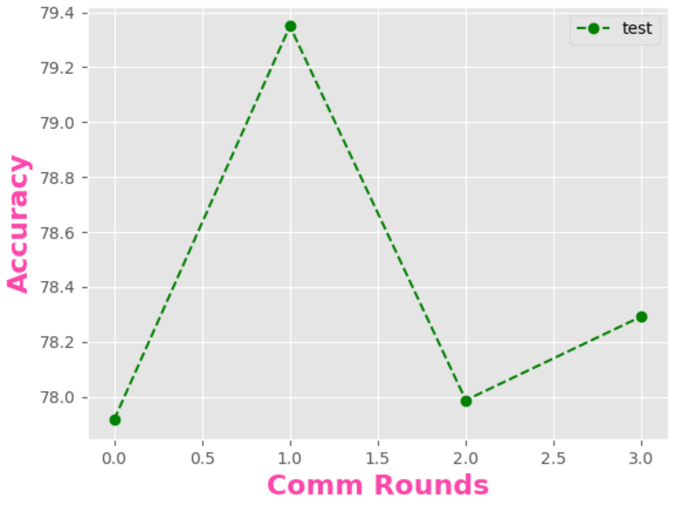


Figure B.8: Accuracy obtained at every communication round using the FedProx algorithm on the MESSIDOR dataset divided in subsets and testing on the Kaggle DR dataset.

Specificity	Sensitivity
83.34 %	68.91 %

Table B.9: Specificity and sensitivity obtained when training on the MESSIDOR dataset divided in subsets using the SCAFFOLD algorithm and testing on the Kaggle DR dataset.

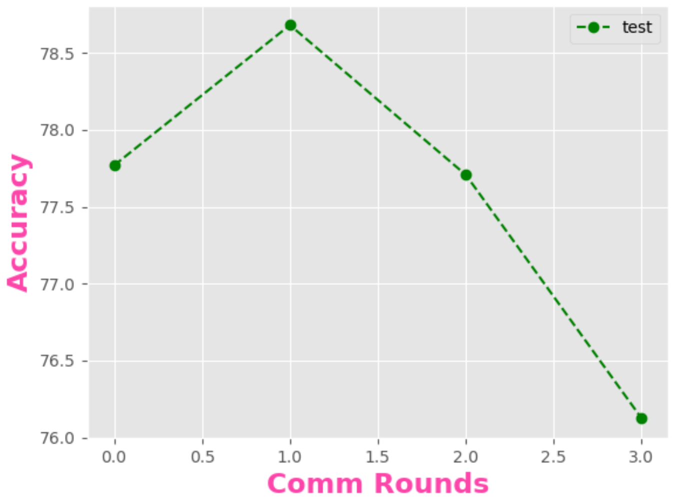


Figure B.9: Accuracy obtained at every communication round using the SCAFFOLD algorithm on the MESSIDOR dataset divided in subsets and testing on the Kaggle DR dataset.

Specificity	Sensitivity
100.00 %	62.61 %

Table B.10: Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the FedAvg algorithm and testing on the MESSIDOR dataset.

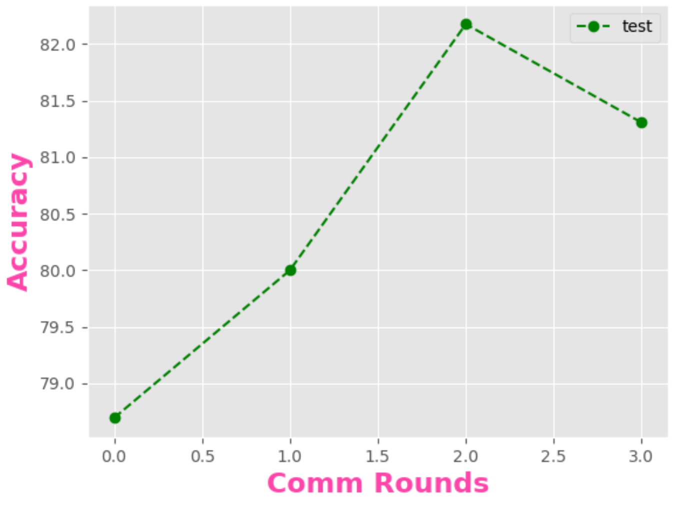


Figure B.10: Accuracy obtained at every communication round using the FedAvg algorithm on the Kaggle DR dataset divided in subsets and testing on the MESSIDOR dataset.

Specificity	Sensitivity
100.00 %	59.13 %

Table B.11: Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the FedProx algorithm and testing on the MESSIDOR dataset.

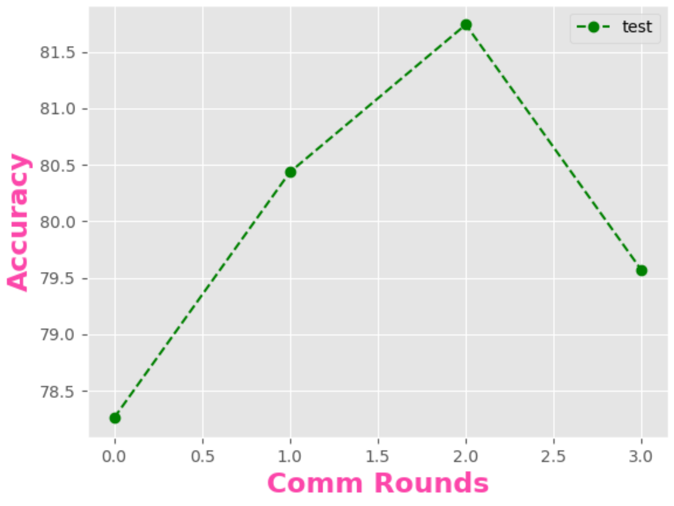


Figure B.11: Accuracy obtained at every communication round using the FedProx algorithm on the Kaggle DR dataset divided in subsets and testing on the MESSIDOR dataset.

Specificity	Sensitivity
100.00 %	65.22 %

Table B.12: Specificity and sensitivity obtained when training on the Kaggle DR dataset divided in subsets using the SCAFFOLD algorithm and testing on the MESSIDOR dataset.

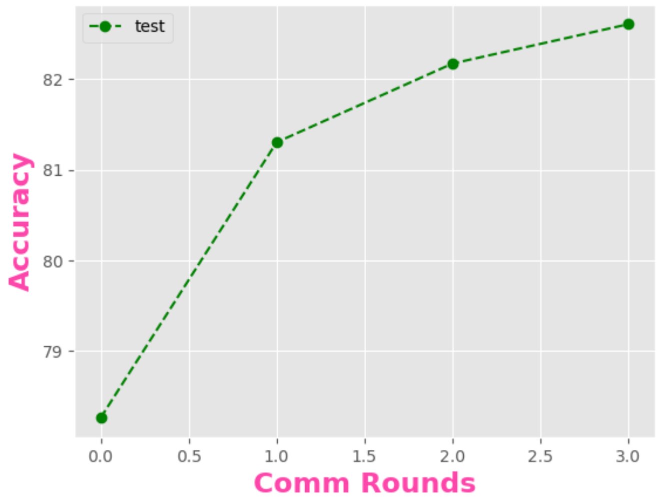


Figure B.12: Accuracy obtained at every communication round using the SCAFFOLD algorithm on the Kaggle DR dataset divided in subsets and testing on the MESSIDOR dataset.