

An Improved LinkNet Based Approach For Retinal Image Segmentation

A PROJECT REPORT

Submitted by

**Srijarko Roy [RA1911026010087]
Ankit Mathur [RA1911026010088]**

Under the guidance of
DR. VELLIANGIRI S
ASSISTANT PROFESSOR,
DEPARTMENT OF COMPUTATIONAL
INTELLIGENCE, SRM IST, KTR - 603203

in partial fulfillment for the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

**COMPUTER SCIENCE & ENGINEERING with specialization in
Artificial Intelligence and Machine Learning**



**DEPARTMENT OF COMPUTATIONAL INTELLIGENCE
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR- 603 203**

MAY 2023



Department of Computational Intelligence
SRM Institute of Science & Technology
Own Work Declaration Form

Degree/ Course : B.Tech in Computer Science Engineering with specialization in Artificial Intelligence and Machine Learning

Student Name : Srijarko Roy, Ankit Mathur

Registration Number : RA1911026010087, RA1911026010088

Title of Work : An Improved LinkNet Based Approach For Retinal Image Segmentation

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

(Under Section3 of UGC Act,1956)

BONAFIDE CERTIFICATE

Certified that 18CSP109L minor project report titled “An improved LinkNet based approach for Retinal Image Segmentation” is the bonafide work of “**SRIJARKO ROY [RA1911026010087], ANKIT MATHUR [R1911026010088]**” who carried out the minor project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

DR. VELLIANGIRI S

SUPERVISOR
Assistant Professor
COMPUTATIONAL
INTELLIGENCE

SIGNATURE

DR. KRISHNAVENI S

PANEL HEAD
Associate Professor
COMPUTATIONAL
INTELLIGENCE

SIGNATURE

DR. R. ANNIE UTHRA

HEAD OF THE DEPARTMENT
COMPUTATIONAL INTELLIGENCE

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. R Annie Uthra**, Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinator, **Dr. T.S. Shiny Angel**, Associate Professor, Panel Head, **Dr. S. Krishnaveni**, Associate Professor and members, **Dr. Velliangiri S**, Assistant Professor and **Dr. A. Robert Singh**, Assistant Professor Department of Computational Intelligence, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. Athilakshmi R**, Assistant Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. Velliangiri S**, Asssistant Professor, Department of Computational Intelligence. SRM Institute of Science and Technology, for

providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Computational Intelligence Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

Srijarko Roy [RA1811026010087]

Ankit Mathur [RA1811026010088]

TABLE OF CONTENTS

Chapter No.	Title	Page No.
	ABSTRACT	iii
	TABLE OF CONTENTS	iv
	LIST OF FIGURES	v
	LIST OF TABLES	vi
	ABBREVIATIONS	vii
1	INTRODUCTION	1
1.1	Objective	1
1.2	Motivation	2
1.3	Software Requirements Specification	3
1.4	Hardware Requirements Specification	3
2	LITERATURE SURVEY	4
2.2	Limitations	4
3	SYSTEM ARCHITECTURE AND DESIGN	9
3.1	Model Architecture	9
3.1.1	Module Descriptions	10
3.1.1.1	Initial Block	10
3.1.1.2	Feature Extractor	10
3.1.1.3	Linked Architecture	10
3.1.1.4	Decoder Block	10
3.1.1.5	Segmentation Block	11
4	METHODOLOGY	12
4.1	Workflow	12
4.2	Dataset	13

APPENDIX

A

CONFERENCE SUBMISSION PLAGIARISM REPORT

ABSTRACT

The characteristics of Retinal Blood Vessels helps diagnose various eye ailments. The proper localization, extraction and segmentation of the blood vessels is very essential to the cause of treatment of the eye. Manual segmentation of blood vessels may be error prone and inaccurate, leading to difficulty in further treatment. We present a novel approach of semantic segmentation of Retinal Blood Vessels using Linked Networks to account for lost spatial information during feature extraction. The implementation of the segmentation technique involves using Residual Networks as a feature extractor and Transpose Convolution for image-to-image translation thereby giving a segmentation mask as an output. The main feature of the architecture is the links between the Feature Extractor and the Decoder networks that enhance the performance of the network by helping in the recovery of lost spatial information. Training and Validation using the Pytorch framework has been performed on the Digital Retinal Images for Vessel Extraction (DRIVE) Dataset to establish quality results.

LIST OF FIGURES

Figure No.	Figure Name	Page No.
3.1	LinkNet Architecture	8
4.1	Workflow Diagram	12
4.2	Original, Horizontal Flip, and Vertical Flip Images	13

LIST OF TABLES

Table No.	Table Name	Page No.
2.1	Literature Survey	5

ABBREVIATIONS

CNN	Convolutional Neural Networks
ReLU	Rectified Linear Unit
SVM	Support Vector Machine

CHAPTER 1

INTRODUCTION

1.1 General

Eye diseases refer to a wide range of conditions that affect the eye, including its structures and vision. Some of the most common eye diseases include:

- Cataracts: A condition where the lens of the eye becomes cloudy, leading to blurry vision.
- Glaucoma: A group of eye diseases that cause damage to the optic nerve, resulting in vision loss and blindness.
- Age-related macular degeneration: A condition where the macula, the central part of the retina responsible for sharp, detailed vision, begins to deteriorate, leading to loss of central vision.
- Diabetic retinopathy: A complication of diabetes that causes damage to the blood vessels in the retina, leading to vision loss and blindness.
- Retinal detachment: A condition where the retina pulls away from its underlying tissue, resulting in vision loss.

Retinal image segmentation plays an important role in the diagnosis, monitoring, and treatment of many of these eye diseases. By accurately segmenting retinal images, doctors and ophthalmologists can detect early signs of disease, monitor progression, and make informed decisions about treatment options. For example, in diabetic retinopathy, retinal image segmentation can help detect and quantify the extent of damage to the blood vessels in the retina, allowing doctors to make informed decisions about treatment, such as laser therapy or surgery. Overall, retinal image segmentation is a critical tool in the diagnosis and management of many eye diseases.

Segmentation of retinal images is an essential aspect of the diagnosis, monitoring, and treatment of various eye diseases. The retinal blood vessels and other structures contain crucial information about the overall health of the eye and the patient. The accurate segmentation of these

structures helps healthcare professionals obtain critical information about the location, severity, and presence of eye diseases such as glaucoma, diabetic retinopathy, and macular degeneration.

The early detection and diagnosis of these conditions is crucial for effective treatment and management of the diseases. With accurate segmentation of the retinal blood vessels and other structures, medical professionals can identify the early signs of eye diseases, monitor their progression over time, and develop more targeted and effective treatments. This approach reduces the risk of vision loss and improves the patient's quality of life.

Furthermore, retinal image segmentation plays an important role in the development and evaluation of new treatments for eye diseases. Precise segmentation of the retinal structures can provide valuable data for clinical trials and research studies, aiding in the identification of potential biomarkers for the disease. Thus, retinal image segmentation is an indispensable tool in the field of ophthalmology and medical imaging, and its significance in the diagnosis and treatment of various eye diseases cannot be overstated.

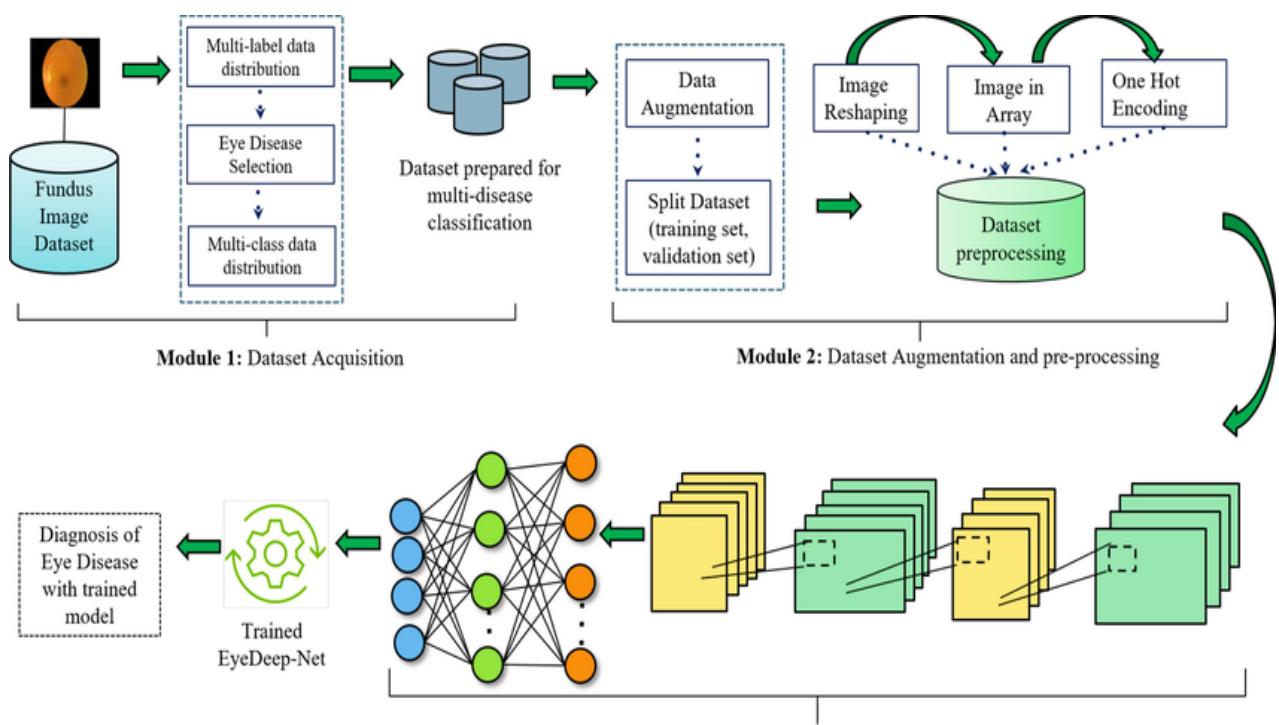


Figure 1.1: Training of a typical ML Model for feature extraction and classification

1.2 Objective

The retinal vascular system plays a vital role in detecting a wide range of disorders and providing essential information about the health of the eye. Retinal scans are frequently utilized to identify early indicators of systemic vascular disease, which makes accurate segmentation of the arteries an essential component of the diagnostic process. In this context, the automated segmentation of retinal blood vessels from fundus images has become increasingly popular in the field of medical imaging.

However, existing systems for Retinal Blood Vessel Segmentation such as UNETs, CNNs, and SVMs have a common issue of losing spatial information during feature extraction, which results in reduced accuracy. This loss of information is caused by multiple downsampling, which compresses the data and makes it more difficult to detect subtle differences.

To address this problem, our paper proposes an improved LinkNet-based architecture for semantic segmentation of Retinal Blood Vessels. Our network uses skip connections to link each Encoder Block to its corresponding Decoder Block, thus preserving the lost spatial information and concatenating it during Upsampling. Additionally, we use Upsample layers instead of Transpose Convolution to counteract the issue of noisy outputs, which is often observed in networks that use Transpose Convolutions for the Up-sampling task.

Our goal is to accurately segment the Blood Vessels present in the retina to provide proper treatment and reduce operator fatigue. By retaining the spatial information that is lost in the feature extraction stage and using Upsample layers, we aim to achieve greater accuracy than existing systems for Retinal Blood Vessel Segmentation. This would have significant implications for the diagnosis and treatment of systemic vascular disease and other related disorders. In our paper, we have tried an implementation of a Semantic Segmentation Network to accurately segment the Blood Vessels present in the retina for proper treatment and reduction of operator fatigue. In order to counter the problem of lost spatial information, we propose using an improved LinkNet based architecture for semantic segmentation of Retinal Blood Vessels. The network makes use of skip connections to link each Encoder Block to its corresponding Decoder Block, passing on the lost spatial information to be concatenated during Upsampling while using Upsample layers as our approach, instead of Transpose Conv counters the problem of noisy output.

1.3 Motivation

Manual segmentation of retinal blood vessels can be a time-consuming and labor-intensive process, requiring expert knowledge and significant training. Moreover, manual segmentation is prone to human error and subjectivity, leading to incorrect segmentation that can make subsequent treatment challenging.

Automatic segmentation of retinal blood vessels can overcome the limitations of manual segmentation by providing accurate and consistent results. The retinal blood vessel system is unique as it provides extensive information about the eye's health, and it is the only non-invasive imaging tool that can get visible blood vessels from the human body. Retinal vascular segmentation is crucial for identifying many diseases, including diabetic retinopathy, age-related macular degeneration, and glaucoma.

Early diagnosis of systemic vascular diseases is essential for better treatment outcomes. Retinal scans are frequently used to detect early signs of systemic vascular disease, as changes in the retinal vasculature are often an early indicator of these conditions. Accurate segmentation of the arteries is critical for the diagnosis of systemic vascular diseases.

Automatic segmentation of retinal blood vessels from fundus images has gained favor in the field of medical imaging due to the development of advanced computer vision techniques such as deep learning-based methods. These techniques use large datasets to train the network to automatically learn the features necessary for segmentation. The automatic segmentation of retinal blood vessels can provide accurate and reproducible results, enabling better diagnosis, treatment, and monitoring of various eye diseases.

In conclusion, automatic segmentation of retinal blood vessels from fundus images has several advantages over manual segmentation, including accuracy, consistency, and efficiency. The segmentation of retinal blood vessels plays a crucial role in the identification and monitoring of many diseases, and automatic segmentation has become a widely used technique in the field of medical imaging.

1.4 Definitions

The study of **artificial intelligence (AI)**, a subfield of computer science, aims to create robots that are capable of voice recognition, decision-making, and language translation—tasks that generally call for human intellect. It entails developing smart algorithms capable of learning from data and making forecasts or judgements based on such data.

Machine learning, deep learning, natural language processing, and computer vision are just a few of the many subfields that make up AI. A branch of artificial intelligence called "machine learning" focuses on creating algorithms that can automatically get better at a particular activity over time. Neural networks, a sort of machine learning that can learn from massive quantities of data and is modelled after the structure of the human brain, are used in deep learning.

While computer vision focuses on creating algorithms that can interpret and analyse visual data, such as images and videos, natural language processing involves teaching computers to understand and produce human language.

Predictive analytics, fraud detection, autonomous cars, and picture and speech recognition are just a few of the useful uses of AI. From healthcare to banking to transportation, it has the potential to revolutionize a wide range of industries.

However, there are other worries about how AI will affect society, like the possibility of job loss and the moral ramifications of autonomous decision-making. As a result, it is critical that ethical standards and rigorous analysis of its possible effects serve as the foundation for the development of AI.

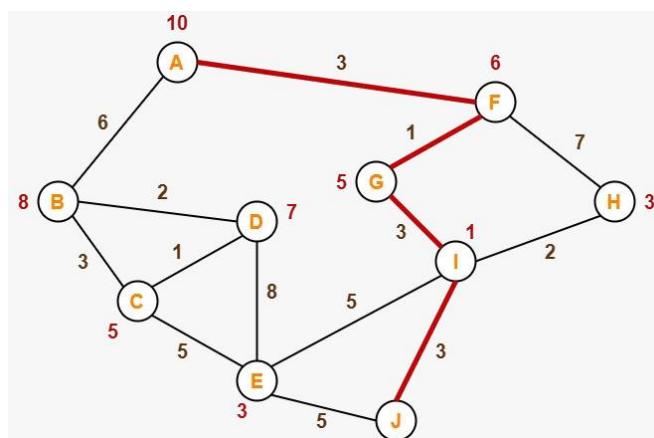


Figure 1.2: A* algorithm using AI

Machine learning (ML) is a kind of artificial intelligence that uses algorithms to give computers the ability to learn from data and get better over time. It is a branch of artificial intelligence that focuses on creating systems that can autonomously learn from experience and get better over time without explicit programming.

The fundamental principle underlying machine learning is to build a model using supervised learning or unsupervised learning on a huge collection of instances. In supervised learning, each example is connected to a predetermined output or label, and the model is trained on labelled data. By minimizing the gap between its projected output and the actual output for each example in the training set, the model subsequently learns to map inputs to outputs.

Unsupervised learning involves training the model on data that has not yet been labelled and has no known output. By grouping related samples together or making the data less dimensional, the model subsequently learns to recognize patterns and structure in the data.

Machine learning algorithms come in a wide variety, each with specific advantages and disadvantages. Typical types include:

- Decision Trees: Using the attributes of the input data, a decision tree-like model produces a series of decisions that result in a projected output or label.
- Naive Bayes: a probabilistic model that, given input data and a set of prior probabilities, determines the likelihood of an output or label.
- Support Vector Machines: A model that determines the hyperplane that optimally separates the input is called a support vector machine.
- Neural networks: A collection of linked nodes that have been taught to carry out a particular activity, such image recognition or natural language processing.

Predictive analytics, fraud detection, natural language processing, speech recognition, picture identification, and recommender systems are just a few of the many useful uses of machine learning. It has the potential to change a wide range of sectors, including healthcare, banking, and transportation.

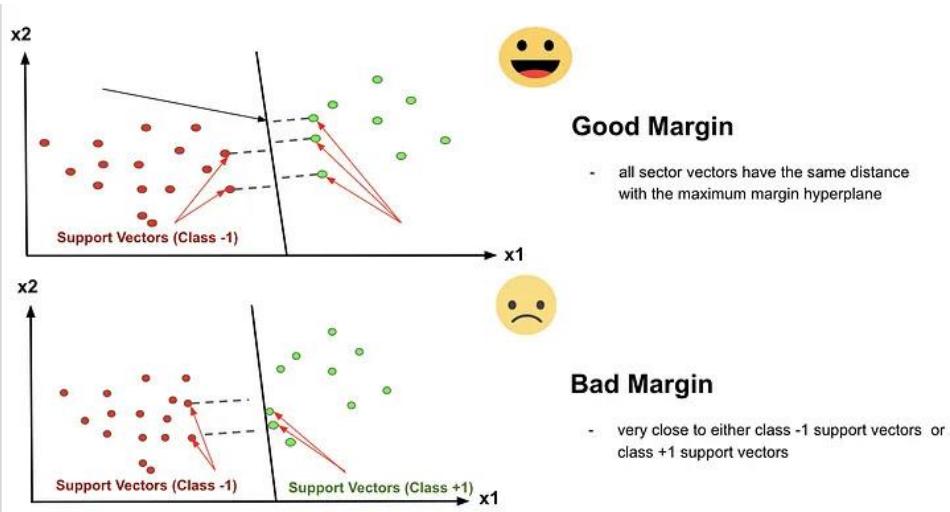


Figure 1.3: Support Vector Machines in ML

Deep learning (DL) is a kind of machine learning that uses neural networks to give computers the ability to learn from data and carry out tasks automatically. It is a type of artificial intelligence (AI) that attempts to replicate how the human brain functions by processing and interpreting data using numerous layers of linked nodes.

Deep learning's fundamental goal is to develop a neural network that can recognize patterns in data on its own, without being explicitly instructed what such patterns are. To do this, a huge collection of instances is used to train the network, and backpropagation is used to modify the network's connections between nodes in reaction to predictions that are incorrect.

Deep learning's primary benefit is its capacity to automatically extract features from unprocessed data, as opposed to relying on labor-intensive manual feature engineering. As a result, it excels at activities like speech and picture recognition, natural language processing, and even gaming.

A deep neural network's design generally consists of numerous layers, each of which processes the incoming data in a unique way. Typically, the input layer is the top layer and is where the raw data is sent. The majority of the processing is done by the intermediate layers, sometimes referred to as hidden layers, which use non-linear activation functions to convert the input data into a higher-level representation. The output layer, the last layer, creates the network's prediction using the processed data.

Convolutional neural networks (CNNs) for image and video processing, recurrent neural networks (RNNs) for sequential data like speech and text, and generative adversarial networks (GANs) for creating new content like images or music are just a few examples of the many different types of neural network architectures used in deep learning.

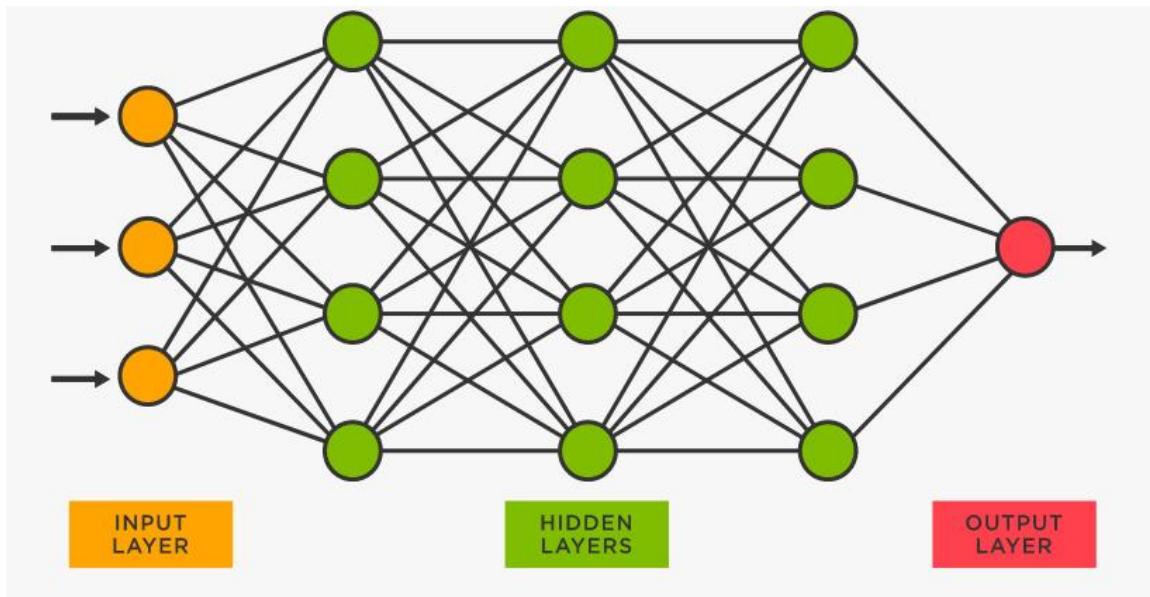


Figure 1.4: A typical 5-layered Neural Network

While deep learning has demonstrated impressive results across a variety of fields, it also necessitates a significant investment in training data and computational power. Deep learning models' interpretability, accountability, possible biases, and ethical ramifications are further issues that need to be addressed. As a result, it's crucial to proceed with caution and give all potential effects of deep learning models serious thought while developing and deploying them.

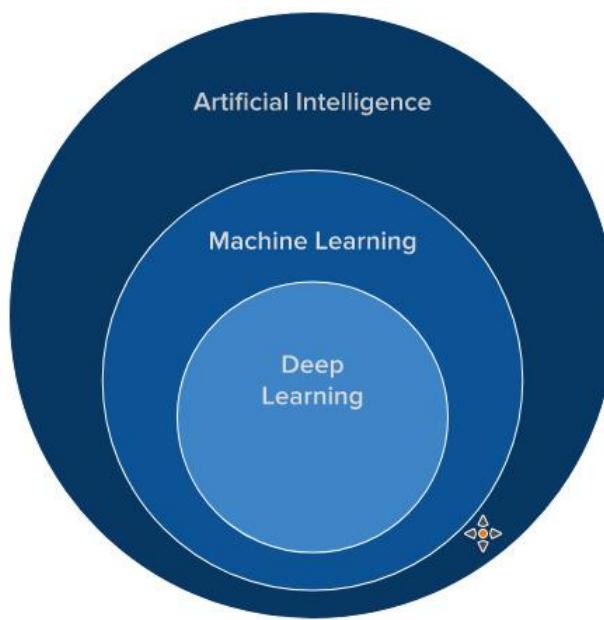


Figure 1.5: Relation between AI, ML and DL

The goal of the artificial intelligence (AI) discipline of **computer vision** is to give computers the ability to comprehend and analyse visual data from the environment. Computer vision aims to make it possible for machines to carry out operations that would often be limited to human vision, such as identifying objects and faces, comprehending and interpreting situations, and detecting and tracking motion.

A variety of methods and tools are used in computer vision, including image processing, pattern recognition, machine learning, and deep learning. These methods are employed to decode, analyse, and extract useful data from picture and video streams.

Computer vision is frequently used for the following purposes:

- Object Recognition and Detection: Identifying and categorising things inside photos or video streams, as well as seeing when they emerge or vanish, are all examples of object identification and detection.
- Face Recognition: Identifying and recognizing human faces within photos or video streams, as well as identifying facial characteristics like the mouth, nose, and eyes.
- Scene Understanding: Comprehension of a scene's context and content, including the things present, how they relate to one another, and the actions taking place, is known as scene comprehension.
- Motion analysis and tracking: Tracking and analyzing object motion in video or picture streams, as well as identifying patterns of object movement across time.
- Image and video enhancement: Boosting specific characteristics or details while boosting the clarity and quality of image and video streams, as well as lowering noise and distortion.

Numerous practical uses for computer vision can be found in a variety of sectors, including healthcare, manufacturing, transportation, and entertainment. It has the potential to revolutionise a variety of facets of daily life, including robots, autonomous cars, and medical diagnosis and care.

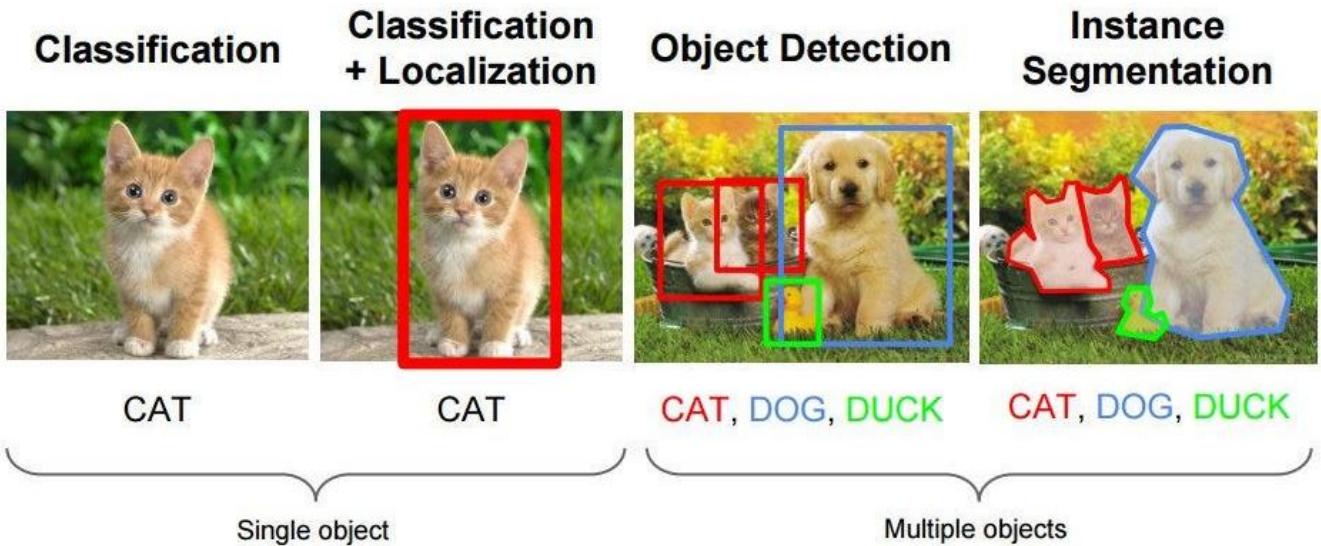


Figure 1.6: Computer Vision Tasks

Segmentation in computer vision is the process of splitting an image into different parts or segments according to predetermined criteria. Segmentation is the process of separating pixels that are part of one item or region from those that are part of other objects or regions.

In computer vision, segmentation may be accomplished using a wide range of methods, including:

- Thresholding: Setting a threshold value and categorising all pixels as belonging to a certain region or item are two steps in the thresholding process.
- Edge detection: It is a technique for segmenting images by locating the edges or borders between various objects or regions in the picture.
- Region Growing: A technique known as "region growing" is starting with a seed pixel or region and spreading outward to include nearby pixels or regions that satisfy particular requirements, such as having the same colour or texture.
- Clustering: It is a method that involves assembling pixels or areas that share characteristics, such as texture or colour.

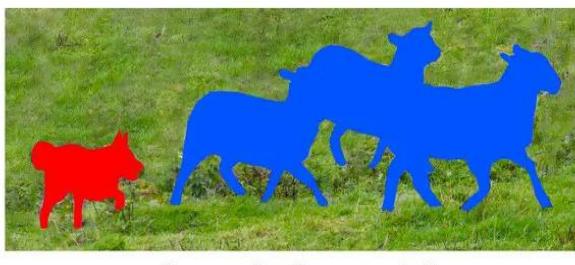
Many computer vision tasks, including object detection, image recognition, and image segmentation, depend on segmentation. Once an image has been segmented, various aspects of the image can be examined independently, enabling more thorough and precise processing.

Segmentation, for instance, may be used to distinguish between several items in a picture, enabling more precise object recognition and tracking. In order to more accurately classify a picture, segmentation in image recognition can be used to separate various elements of the image, such as the foreground and background. Additionally, the objective of image segmentation is to divide the entire image into various areas or objects, enabling a more in-depth examination and processing of the image.

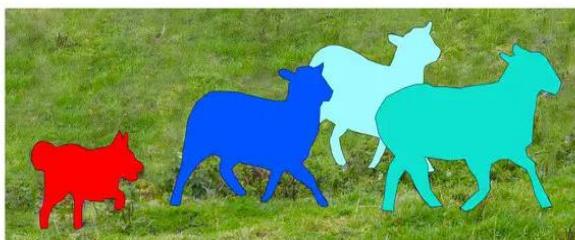
Semantic segmentation associates every pixel of an image with a class label such as a person, flower, car and so on. It treats multiple objects of the same class as a single entity.

In contrast, instance segmentation treats multiple objects of the same class as distinct individual instances.

For example, in the image shown below, Semantic Segmentation identifies all sheep as the same entity and gives them the same mask. However, Instance Segmentation treats each of the three sheep as different entities and gives them different colored masks.



Semantic Segmentation



Instance Segmentation

Figure 1.7: Semantic vs Instance Segmentation

The process of removing significant features or patterns from an image or collection of pictures is referred to as **feature extraction** in computer vision. The objective of feature extraction is to keep the most crucial and pertinent information while reducing the dimensionality of the visual data.

For feature extraction in computer vision, a variety of methods can be utilised, including:

- Histogram of Oriented Gradients (HOG): A method that groups the gradients of an image into histograms according to their orientations after computing the gradients of the picture.
- Scale Invariant Feature Transform: Using the size-Invariant Feature Transform (SIFT) approach, key points and local features are located in an image, and their size, direction, and intensity are used to describe them.
- Convolutional Neural Networks (CNNs): A class of neural network that is specifically created to analyze pictures and that, using the convolution and pooling techniques, can automatically learn to extract information from images.
- Local Binary Patterns (LBP): A method that compares the brightness of each pixel in an image to each of its neighbors before grouping the comparisons into patterns that may be used to identify specific local characteristics in the picture.

Once features have been extracted from an image or set of images, they can be applied to a number of tasks, including object recognition, image retrieval, and image classification. For instance, the objective of picture classification is to place an image into one of many already established groups depending on its features.

The objective of object recognition is to recognize particular things within a picture based on their attributes. In image retrieval, the objective is to find photos in a database that, based on their properties, are comparable to a query image.

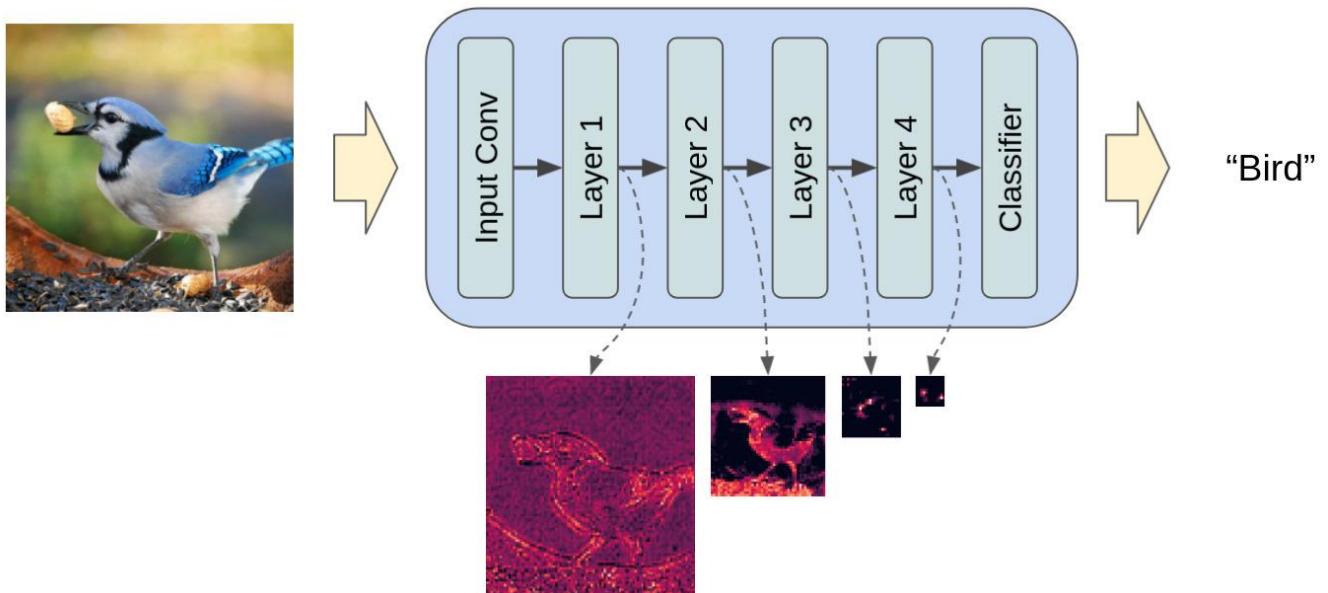


Figure 1.8: Feature Extraction

Upsampling is a technique used in computer vision to boost an image's resolution or size. To produce a more accurate and detailed image, it is frequently used with other image processing techniques like downsampling or pooling.

In computer vision, there are various ways to upsample a picture, including:

- Nearest-neighbor interpolation: With nearest-neighbor interpolation, each pixel from the original image is simply duplicated to produce a larger image. Although quick and simple to use, this method may cause a loss of sharpness and detail.
- Bilinear interpolation: This technique generates a new pixel in the expanded picture by computing a weighted average of the four closest pixels in the original image. Compared to nearest-neighbor interpolation, this approach maintains more detail and results in a smoother image.
- Lanczos Interpolation: The Lanczos interpolation technique takes a weighted average of more pixels from the source picture than other interpolation techniques. Although it can be slower and require more computational power than other methods, it still yields results of high quality.

There are several applications for upsampling, including picture super-resolution, which converts a low-resolution image into a high-resolution image, and object detection, which uses upsampling to expand the size of feature maps in a convolutional neural network to enhance the recognition of objects.

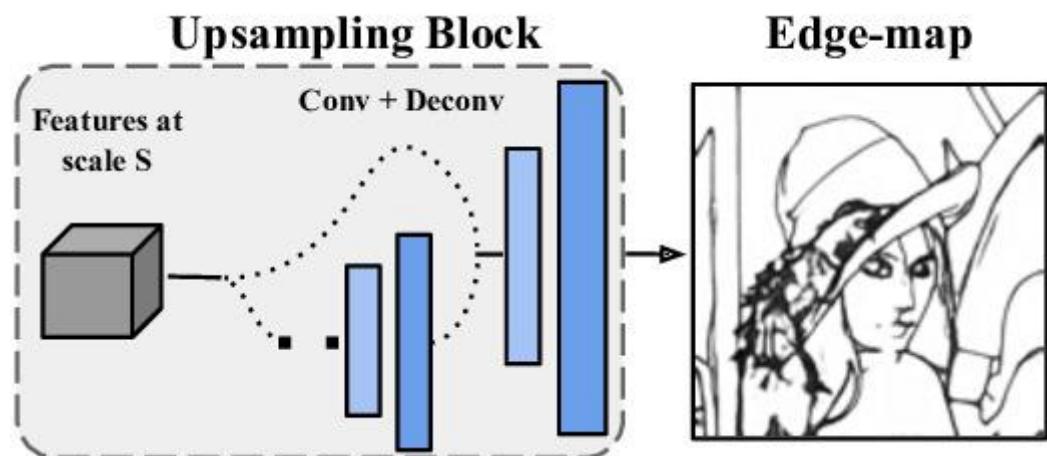


Figure 1.9: Upsample Operation

Skip connections are a method used in computer vision to increase the accuracy of image processing tasks including object detection, segmentation, and picture synthesis, notably in deep neural networks.

Instead of processing data through a number of intermediary levels, a skip link enables data from an earlier layer of the network to be transferred straight to a later tier. This aids in resolving the issue of vanishing gradients, which makes it challenging for a network to learn since the gradients used in backpropagation become extremely tiny as they are propagated back through several layers of the network.

The output of one layer is mixed with the output of an earlier layer in a neural network with skip connections before being transmitted to the following layer.

This helps to retain the spatial information of the picture across the network and enables the network to learn both low-level and high-level characteristics at the same time.

The residual connection, which was first used in the ResNet design, is a typical sort of skip connection. In a residual connection, an earlier layer's output is added to its input rather than concatenated with it. Instead of attempting to learn the transformation from start, the network may learn residual functions, which record the difference between an input and an output of a layer.

Overall, it has been demonstrated that skip connections enhance the performance of deep neural networks on a range of computer vision applications, and they are now a common element of many state-of-the-art architectures.

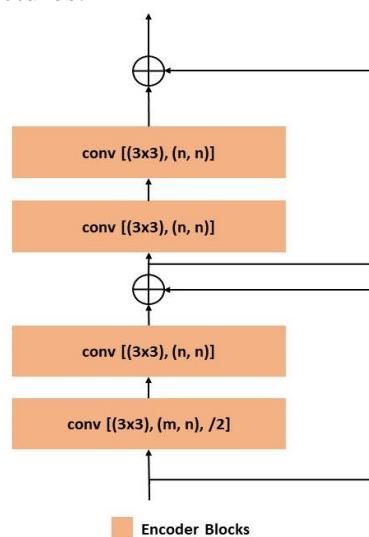


Figure 1.9: Skip Connections

1.5 Software Requirements Specification

- ❖ Google Colab for Training
- ❖ VS Code
- ❖ Linux Terminal for local Testing
- ❖ Python
- ❖ PyTorch

1.6 Hardware Requirements Specification

- ❖ NVIDIA P100
- ❖ cuDNN VERSION 8005

CHAPTER 2

LITERATURE SURVEY

2.1 Key Points Identified

Existing systems for Retinal Blood Vessel Segmentation such as UNETs [1], CNNs [2], and SVMs [3] have a common problem of the loss of important spatial information during the feature extraction stage due to multiple downsampling. Hence, the existing models are not able to achieve the accuracy that they could have if the spatial information has been retained. Retention of this spatial information can lead to increased accuracy of segmentation thereby improving the diagnosis of various eye ailments. Certain networks such as UNETs use Transpose Convolutions [4] for the Upsampling task. The usage of Transpose Convolution often leads to noisy outputs. In our paper, we have tried an implementation of a Semantic Segmentation Network to segment blood vessels present in the retina for proper treatment and reduction of operator fatigue. As a counter to the problem of lost spatial information, we propose using an improved LinkNet [5] based architecture for semantic segmentation of the Retinal Blood Vessels. The network makes use of skip connections [6] to link each and every Encoder Block to its corresponding Decoder Block, concatenating the lost spatial information during Upsampling while using Upsample Blocks as our approach, instead of Transpose Conv counters the problem of noisy output.

Significant advancements have been achieved over the years in the field of Computer Vision with respect to biomedical image segmentation. Implementations of various architectures for Retinal Image Segmentation have been made with increasing accuracy with each implementation. [7] has been used by many architectures for their development over the years. A major breakthrough started when the ridge-based segmentation method was used by Joes Staal et. al. in [8], achieving an accuracy of 0.9441. Centerline detection and morphological reconstruction were used a few years later by Mendonca et. al. [9], to improve the accuracy of segmentation to 0.9452. An attempt to further improve morphological reconstruction was made in [10] by using mathematical morphological theory and intensity transformation algorithm. The method used True Positive Fraction as the evaluation metric achieving a value of 0.8214 for the same. Further research was done on the DRIVE Dataset during the time with Morlet Transforms [11] by Ghaderi et. al. which achieved an AUC ROC of 0.90 to 0.96. Research shifted from DRIVE Dataset to other custom datasets when OctaveUNets [12] were used for Retinal Image Segmentation. Trained and validated on a custom dataset, OctaveUNets achieved an accuracy of 0.9524, thus leading to further improvements in this field. With OctaveUnets achieving high

accuracy, neural networks scheme for pixel classification and moment invariant-based features for pixel representation-based methods were experimented with by Marin et. al. on the DRIVE Dataset. Training and Validation with this architecture led to an accuracy of 0.9452.

New datasets like STARE [13] started to be used alongside DRIVE for model improvement. Xinge et. al. used both DRIVE and STARE datasets for their research with semi-supervised learning and radial projection [14], and achieved a better accuracy on STARE Dataset, 0.9434 on DRIVE and 0.9497 on STARE. With better results on STARE, new datasets such as CHASE-DB [15] started to be used for training. [16] achieved an accuracy of 0.9480, 0.9534 and 0.9469 on DRIVE, STARE and CHASE -DB datasets respectively using ensemble methods and Gabor Filters. Multiscale line-detection [17] was used by Nguyen et. al. on STARE Dataset achieving an accuracy of 0.9326. CLAHE followed by a 2D Gabor wavelet was used in [18] on DRIVE and STARE datasets to get an accuracy of 0.9477 and 0.9509 respectively. Further research we're conducted using a Drive dataset with a B-COSFIRE filter [19], unsupervised iterative segmentation with tophat reconstruction and global thresholding [20] and Holistically Nested Edge Detection [21]. They achieved an accuracy of 0.9442, 0.9494 and 0.9435 respectively. Multilevel CNN with Conditional Random Fields [22] was used on DRIVE, STARE and CHASE-DB datasets with an average accuracy of 0.9523. SVMs were used on DRIVE Dataset, using accuracy and specificity as evaluation metrics, achieving values of 0.9538 and 0.9773 respectively. Moving to a more data-centric approach, UNETs with data-specific augmentations [23] were used on DRIVE, achieving an AUC ROC of 0.9790. UNETs were further improved in [24], achieving an accuracy of 0.9650 on DRIVE Dataset. Modified SUSAN edge detector and shape analysis was used in [25] on DRIVE and STARE datasets achieving an accuracy of 0.9633 and 0.9610 on them respectively. This was improved upon by Deformable ConvNets [26] achieving 0.9628 and 0.9690 accuracy on DRIVE and STARE datasets.

In our paper, we have tried to employ skip connections to concatenate lost spatial information during feature extraction with their corresponding decoder blocks to convert the lost information into useful information for better segmentation.

2.2 Limitations

UNETs [1], CNNs [2], and SVMs [3] are some of the current Retinal Blood Vessel Segmentation algorithms that share the problem of losing important spatial information during the feature extraction stage as a consequence of recurrent downsampling. The accuracy of the existing models is thus worse than it may be if the spatial information had been preserved.

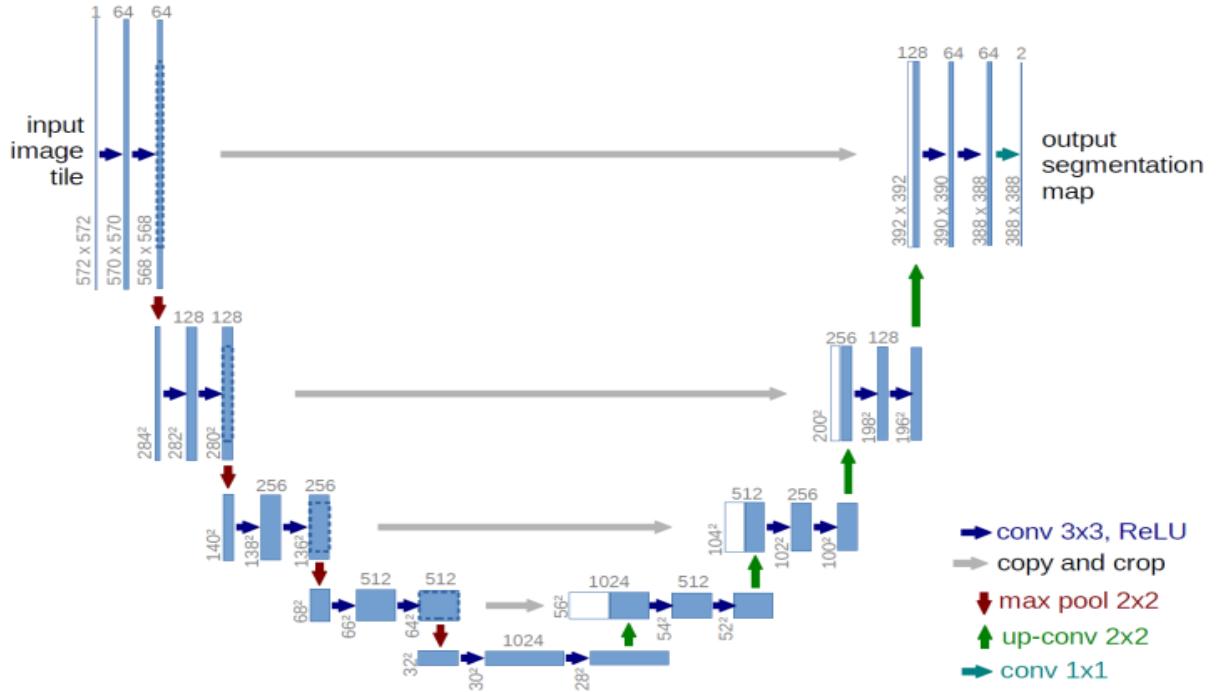


Figure 2.1: U-Net Architecture

It will be easier to diagnose various eye problems by improving segmentation accuracy by keeping this spatial information. Some networks, including UNETs, use Transpose Convolutions [4] to perform upsampling. Transpose Convolution, which generates noisy outputs, is commonly utilised for correct handling and to lessen operator fatigue.

As a result, a major issue with current techniques for retinal blood vessel segmentation is the loss of crucial spatial information during successive downsampling, the feature extraction step. As a result, the accuracy of the current models is worse than it may be if the spatial information had been maintained.

Transpose Convolutions are used in some networks, including UNETs, for upsampling. Transpose Convolution use frequently results in noisy outputs.

Table 2.1: Literature Survey

Paper	Author(s)	Dataset	Method	Metrics
Retina Blood Vessel Segmentation Using A U-Net Based Convolutional Neural Network	Wang Xiancheng et. al.	DRIVE Dataset	U-Net with specific data augmentations	0.9790 (AUC ROC)
Blood Vessel Segmentation in Retinal Images Using Lattice Neural Networks	Gildardo Sanchez-Ante et. al.	STARE Database	Lattice Neural Networks with Dendritic Processing	0.81 (F1 Score)
Methods for the detection of blood vessels in retinal fundus images and reduction of false-positive pixels around the optic nerve head	Ashish Kumar Dhara et. al.	DRIVE Dataset	Multiscale vesselness measures and Gabor filters	0.9616 (F1 Score)
Recurrent Convolutional Neural Networks for Continuous Sign Language Recognition by Staged Optimization	<u>Runpeng Cui, Hu Liu, Changshui Zhang</u>	RWTH-PHOENIX-Weather multi-signer 2014	CNN with temporal convolution and RNN with LSTM	38.7 % (WER)
A Fundus Retinal Vessels Segmentation Scheme Based on the Improved Deep Learning U-Net Model	Xiujin Pan et. al.	DRIVE Dataset	Improved U-Net	96.50% (Segmentation Accuracy)
DeepVessel: Retinal Vessel Segmentation via Deep Learning and Conditional Random Field	Huazhu Fu et. al.	DRIVE, STARE and CHASE_DB1	Multi-level CNN with Conditional Random Fields	0.9523 (Accuracy)
Segmentation of retinal blood vessels by combining the detection of centerlines and morphological reconstruction	Mendonca et. al.	DRIVE Dataset	Morphological Processing	0.9452 (Accuracy)

An effective retinal blood vessel segmentation method using multi-scale line detection	Nguyen et. al.	STARE Dataset Recognition Dataset	Multi-Scale Line Detection	0.9326 (Accuracy) 0.789 (Jaccard Index)
Retinal blood vessel segmentation based on heuristic image analysis	Maja Braovic et. al.	DRIVE and STARE Dataset	Modified SUSAN edge detector and shape analysis	96.33% and 96.10% (Accuracy) respectively
Retinal vessels segmentation based on level set and region growing	Yu Qian Zhao et. al.	DRIVE and STARE databases	Contrast-limited adaptive histogram equalization (CLAHE), followed by a 2D Gabor wavelet to enhance the contrast of the retinal image.	94.77% on the DRIVE database and 95.09% on the STARE database (Accuracy)
An Ensemble Classification-Based Approach Applied to Retinal Blood Vessel Segmentation	Fraz et. al.	DRIVE, STARE, CHASE	Ensemble system of bagged and boosted decision trees and utilizes a feature vector based on the orientation analysis of gradient vector field, morphological transformation, line strength measures, and Gabor filter	0.9480, 0.9534, 0.9469 respectively (Accuracy)
A New Supervised Method for Blood Vessel Segmentation in Retinal Images by Using Gray-Level and Moment Invariants-Based Features	Diego Marin et. al.	DRIVE	neural network (NN) scheme for pixel classification and moment invariants-based features for pixel representation	0.9452 (Accuracy)
Trainable COSFIRE filters for vessel delineation with application to retinal images	G Azzopardi et. al.	DRIVE	B-COSFIRE filter	0.9442 (Accuracy)

Iterative vessel segmentation of fundus images	Roychowdhury et. al.	DRIVE	Unsupervised Iterative Segmentation with tophat reconstruction and global thresholding	0.9494 (Accuracy)
Ridge-Based Vessel Segmentation in Color Images of the Retina	Joes Staal et. al.	DRIVE	Ridge-Based	0.9441 (Accuracy)
Holistically-Nested Edge Detection	Saining Xie, Zhuowen Tu	DRIVE	HED	0.9435 (Accuracy)
Comparative Analysis of Vessel Segmentation Techniques in Retinal Images	AZHAR IMRAN et. al.	DRIVE, STARE, CHASE_DB1, HRF, MESSIDOR, REVIEW	Vector machine-based methods, neural network-based methods, miscellaneous methods, matched filter methods.	67.6% On CHASE_DB1 (Accuracy)
A retinal vessel boundary tracking method based on Bayesian theory and multi-scale line detection	JiaZhang et. al.	REVIEW (KPIs, CLRIS, HRIS, VDIS)	A Bayesian method with the Maximum a posterior (MAP)	100%, 98.23%, 94.2%, 100% (Width Error)
Accurate Retinal Vessel Segmentation via Octave Convolution Neural Network	Zhun Fan et. al.	DRIVE, STARE, CHASE DB1, and HRF	Octave UNet	0.9663 (Accuracy), on DRIVE Dataset
Retinal Blood Vessel Segmentation by Support Vector Machine Classification	Eva Tuba et. al.	DRIVE	Overlapping-block-based supervised machine learning approach for blood vessel segmentation	0.9538 (Accuracy), 0.9773 (Specificity)
AUTOMATIC BLOOD VESSEL SEGMENTATION IN COLOR IMAGES OF RETINA	A. OSAREH and B. SHADGAR	Custom Dataset	Octave UNet	95.24% (Accuracy)

Segmentation of retinal blood vessels using the radial projection and semi-supervised approach	Xinge You et. al.	DRIVE and STARE	Vessel segmentation	0.9434 on DRIVE And 0.9497 on STARE
Morphology Approach for Features Extraction in Retinal Images for Diabetic Retionopathy Diagnosis	Ibrahim Abdurrazaq et. al.	DRIVE	Mathematical morphology theory and intensity transformation algorithm	TPF value of 0.8214 (True Positive Fraction)
Retinal Vessel Segmentation Using Deep Neural Networks	Martina Melinscak, Pavle Prentasic and Sven Loncaric	DRIVE	DNNs or CNNs which instead of subsampling or down-sampling layers have a maxpooling layer (MPCNNs).	0.9466 with 0.7276 and 0.0215 TPR and FPR
Construction of Retinal Vessel Segmentation Models Based on Convolutional Neural Network	Qiangguo Jin et. al.	DRIVE and STARE	Deformable-ConvNet	Accuracy of 0.9628/0.9690
Retinal Vessel Segmentation Using the 2-D Morlet Wavelet and Neural Network	R.Ghaderi et. al.	DRIVE	Morlet transform	0.90% to 0.9668% of the area under the ROC curve

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Model Architecture

The LinkNet architecture as depicted in Figure 3.1 and makes use of Residual Blocks based on the ResNet18 architecture in its Feature Extractor. To enable the linking operation, strided convolutions have been used in the encoder.

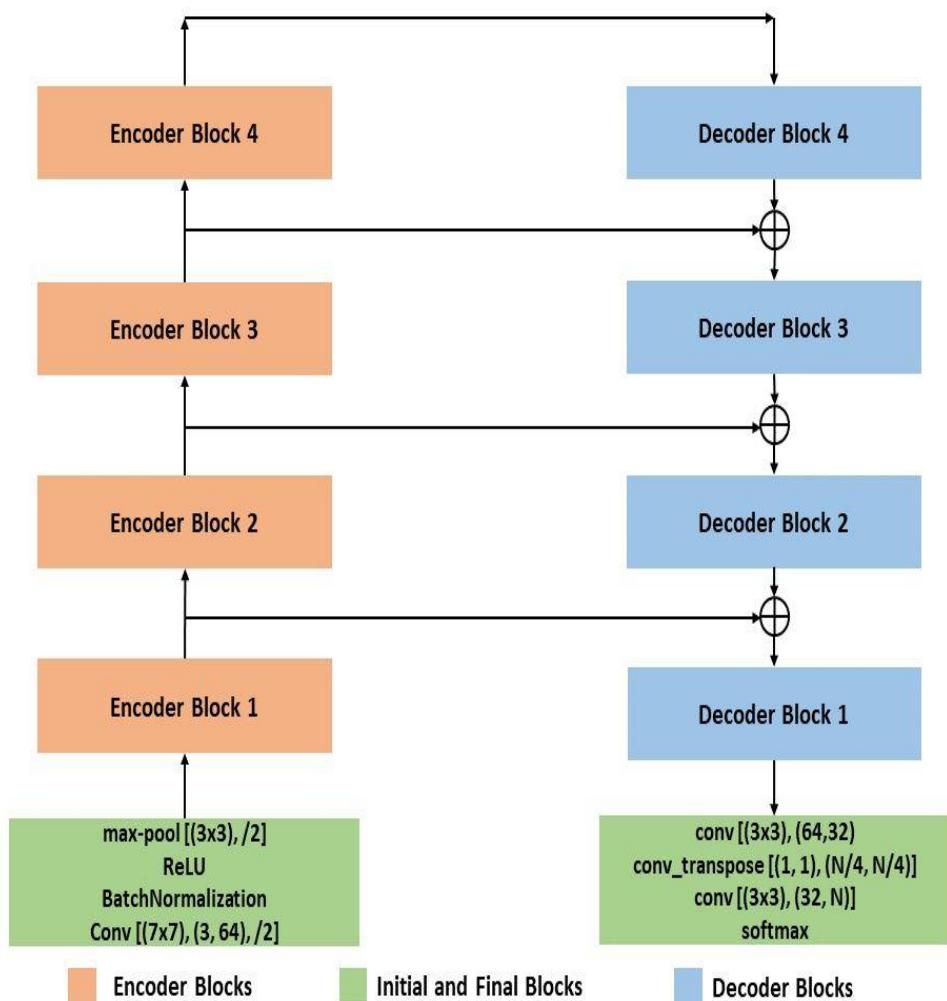


Figure 3.1: LinkNet Architecture

3.1.1 Module Descriptions

3.1.1.1 Initial Block

The Initial Block of the Feature Extractor performs a convolutional operation on the input image with a (7x7) kernel and a stride of 2. This is followed by Batch Normalization and ReLU activation and a spatial max-pooling operation with a kernel size of (3x3) and a stride of 2.

3.1.1.2 Feature Extractor (Residual Blocks)

Following the Initial Block of the Feature Extractor are the Residual Blocks for the main feature extraction. They are represented in Figure 3.1 as Encoder Block. Each layer within the Residual Block is shown in Figure 3.1 in detail. Each Residual Block has a strided convolution operation with a (3x3) kernel accompanied by skip connections. The skip connections are responsible for countering the issue of Vanishing Gradients that affects the performance of previously proposed models. With the use of skip connections, the network introduces alternate shortcut paths such that the loss calculated at the output layer can find an alternate way in case they are not backpropagated to the previous layers.

3.1.1.3 Linked Architecture

The linking of each encoder to each decoder has been performed exactly as mentioned in, in order to recover the lost spatial information as a result of multiple downsampling operations in its encoder. To enable the linking operation, strided convolutions have been used in the encoder.

3.1.1.4 Decoder Blocks

Each decoder block has 2 convolutional operations with a (1x1) kernel and an Upsample operation in trilinear mode with scale factor 2, between them. The Upsample operation counters the noisy output produced by transpose convolution. The decoder blocks are followed by the final segmentation block.

3.1.1.5 Segmentation Block

The final segmentation block of the decoder performs the main segmentation task on the output received from Decoder Block 1. A convolution operation with kernel size (3x3) is performed on the decoder output, followed by an Upsample operation with scale factor 2 in the trilinear mode for a noise-reduced output. Finally, another convolution operation with kernel size (3x3) is performed before passing it through a softmax layer to get the final segmentation mask with pixel probabilities.

Table 3.1: LinkNet Modules and Blocks

Module	Blocks	Action	Out Channels
Feature Extractor	Initial Input Block	Convolution and max-pooling	64
	Residual Blocks	4 blocks following the initial block for Feature Extraction	64, 128, 256, 512
Decoder	Decoder Blocks (Upsample)	Convert concatenated channels (512, 256, 128, 64) to required channel size	256, 128, 64, 64
	Final Block	Produces Final Segmentation Mask	1

CHAPTER 4

METHODOLOGY

4.1 Workflow

The first step of workflow for our model implementation is dataset collection. Once the data has been collected, each image in the dataset was preprocessed. The preprocessing includes augmentations, standardization and normalization. The preprocessed images are passed to the LinkNet Model for training and validation purposes. The weights have been saved and hosted for real-time usage.

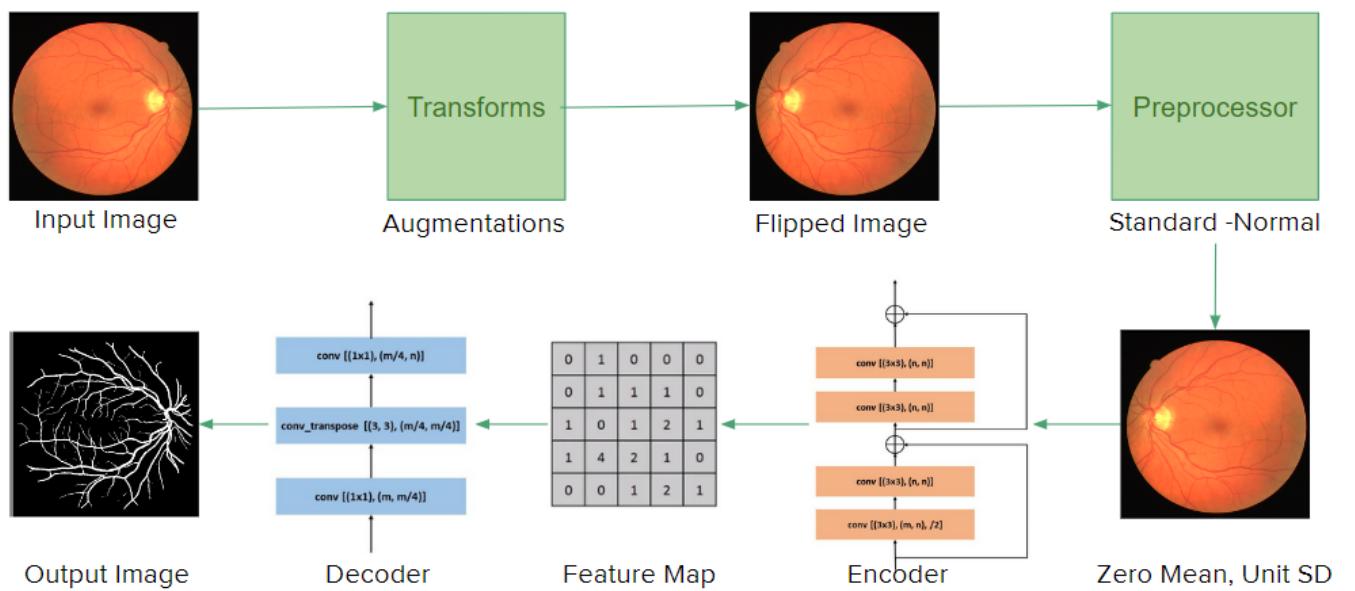


Figure 4.1: Workflow Diagram

4.2 Dataset

A dataset for segmenting retinal vessels is called Digital Retinal Images for Vessel Extraction (DRIVE). It contains of 40 JPEG colour fundus photos in total, including 7 occurrences of aberrant disease. The photos came from a screening programme for diabetic retinopathy in the Netherlands. A Canon CR5 non-mydriatic 3CCD camera with a 45-degree field of view was used to capture the photographs. Each picture has a 584 by 565 pixel resolution and eight bits per colour channel (3 channels).

Twenty photos were used for the training set and twenty images were used for the testing set out of the total 40 images. A circular field of view (FOV) mask with a diameter of around 540 pixels is present in both sets for each picture. For each image in the practise set, an ophthalmological specialist has used manual segmentation. In the testing set, each image has undergone two manual segmentations performed by two distinct observers, with the first observer's segmentation serving as the baseline for performance assessment. Augmentation techniques namely, horizontal flip and vertical flip have been applied for further improvement of the training data.

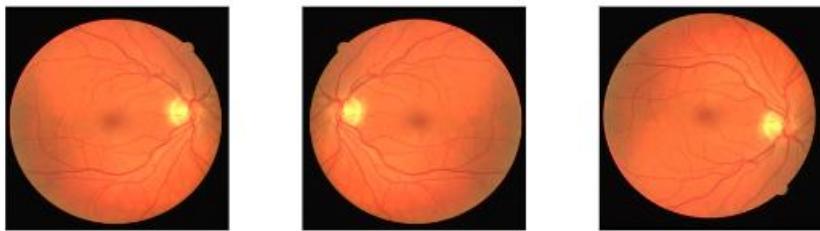


Figure 4.2: Original, Horizontal Flip, and Vertical Flip Images

Pre-processing techniques like normalization and standardization have been applied along with the augmentations; all images were processed to contain zero mean and unit standard deviation, i.e. convert the images to a standard Gaussian distribution as a part of the procedure.

4.3 Metrics

4.3.1 Dice Loss

We have adopted Dice Loss as a metric for evaluating the performance of our model. The overlap or resemblance between two sets is calculated using the dice coefficient. This metric is particularly popular with class-imbalance problems. The predicted label and the ground truth can be considered as two sets for the purposes of semantic segmentation. The Dice coefficient ranges in value from 0 to 1, where 0 indicates no overlap and 1 indicates total overlap. The dice coefficient is calculated using the following formula.

$$\text{Dice Coefficient} = 2 | T \cap P | / |T| + |P|$$

The generalised loss function is calculated from the dice coefficient by subtracting it from 1. This dice loss is minimised which maximises the dice coefficient as higher value implies a better overlap.

4.3.2 IoU Loss

We have used Intersection over Union as another metric for evaluating the performance of our model. IoU, as the name suggests, is calculated as a ratio of the overlap of the predicted label with the ground truth to their union, i.e. the total area they cover. Similar to the dice loss, it ranges from 0 to 1 , with 0 indicating no overlap and 1 indicating total overlap. The Intersection over Union value is calculated as:

$$\text{Intersection over Union} = | T \cap P | / |T \cup P|$$

The IoU loss function is calculated by subtracting the IoU Score from 1, and similar to the Dice Loss, a lower IoU loss gives a better overlap, hence is minimised.

4.4 Training Details

The PyTorch [16] framework was used to code the architecture, and the NVIDIA Tesla T4 GPU and CUDA integration were combined to train it. With a batch size of 4, 80 images were used to train the model. Before being loaded into a dataloader object for training, each picture underwent pre-processing, was enhanced, normalised, and standardised. To reduce the dice and IoU loss, the model was trained using the Adam optimizer for 100 iterations at a learning rate 0.0001.

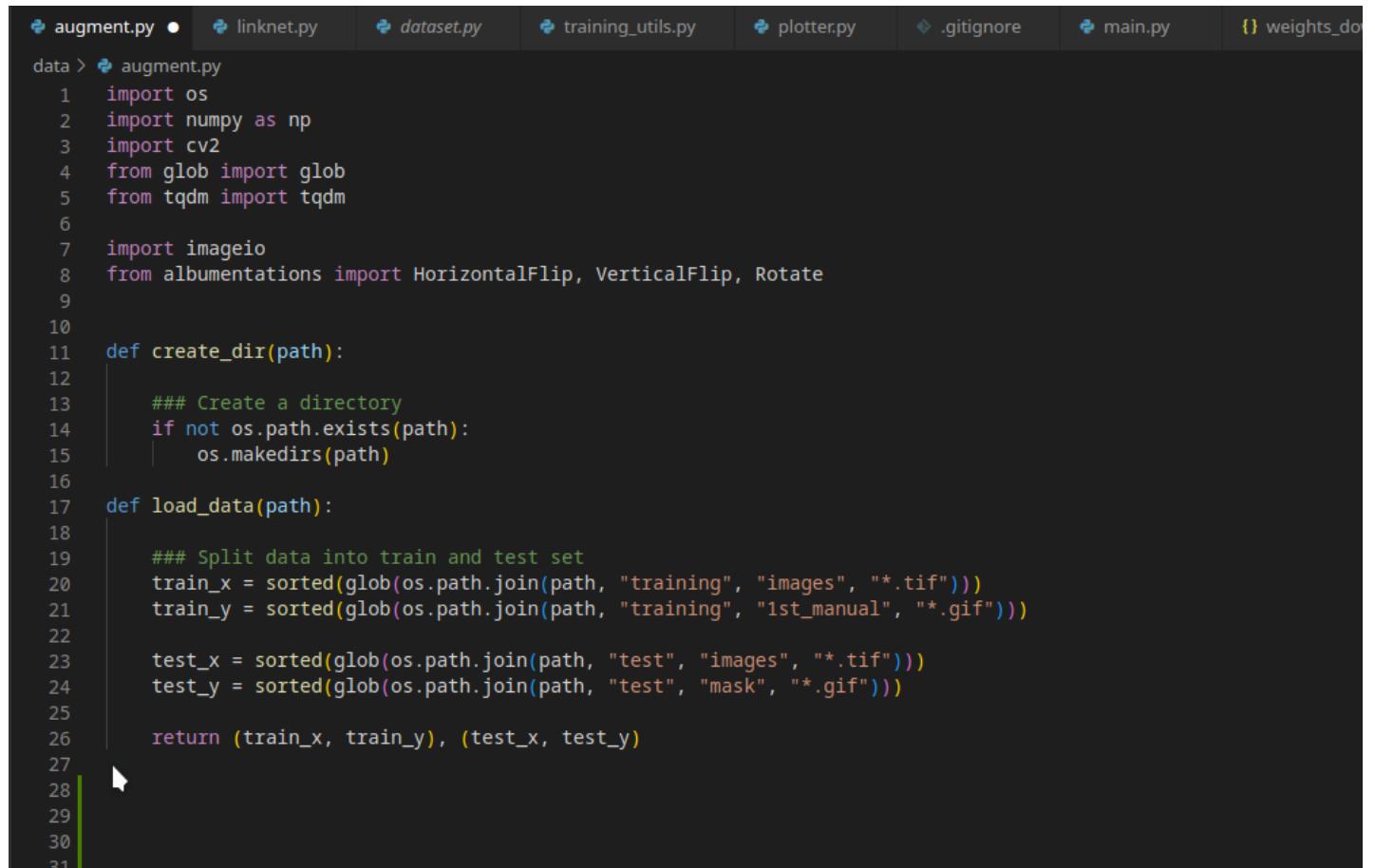
Table 4.1: Hyperparameters

Hyperparameter	Value
Batch Size	4
Epochs	100
Learning Rate	0.0001
Optimizer	Adam
Loss Function	Dice Loss, Intersection over Union

CHAPTER 5

CODING AND TESTING

5.1 Data



```
augment.py ● linknet.py dataset.py training_utils.py plotter.py .gitignore main.py weights_d...  
data > augment.py  
1 import os  
2 import numpy as np  
3 import cv2  
4 from glob import glob  
5 from tqdm import tqdm  
6  
7 import imageio  
8 from albumentations import HorizontalFlip, VerticalFlip, Rotate  
9  
10 def create_dir(path):  
11     """ Create a directory  
12     if not os.path.exists(path):  
13         os.makedirs(path)  
14  
15 def load_data(path):  
16     """ Split data into train and test set  
17     train_x = sorted(glob(os.path.join(path, "training", "images", "*tif")))  
18     train_y = sorted(glob(os.path.join(path, "training", "1st_manual", "*gif")))  
19  
20     test_x = sorted(glob(os.path.join(path, "test", "images", "*tif")))  
21     test_y = sorted(glob(os.path.join(path, "test", "mask", "*gif")))  
22  
23     return (train_x, train_y), (test_x, test_y)  
24  
25  
26  
27  
28  
29  
30  
31
```

Fig 5.1: Load Data Function

```

augment.py X linknet.py dataset.py training_utils.py plotter.py .gitignore main.py { weights_down
data > augment.py
27
28     def augment_data(images, masks, save_path, augment=True):
29
30         """ Augmenting data into Horizontal,Vertical & Rotate
31
32         size = (512, 512)
33
34         for idx, (x, y) in tqdm(enumerate(zip(images, masks)), total=len(images)):
35             """ Extracting the name """
36             name = x.split("/")[-1].split(".")[0]
37
38             """ Reading image and mask """
39             x = cv2.imread(x, cv2.IMREAD_COLOR)
40             y = imageio.imread(y)[0]
41
42             if augment == True:
43                 aug = HorizontalFlip(p=1.0)
44                 augmented = aug(image=x, mask=y)
45                 x1 = augmented["image"]
46                 y1 = augmented["mask"]
47
48                 aug = VerticalFlip(p=1.0)
49                 augmented = aug(image=x, mask=y)
50                 x2 = augmented["image"]
51                 y2 = augmented["mask"]
52
53                 aug = Rotate(limit=45, p=1.0)
54                 augmented = aug(image=x, mask=y)
55                 x3 = augmented["image"]
56                 y3 = augmented["mask"]
57

```

Fig 5.2: Data Augmentation Function

```

augment.py X linknet.py dataset.py training_utils.py plotter.py .gitignore main.py
data > augment.py
57
58         X = [x, x1, x2, x3]
59         Y = [y, y1, y2, y3]
60
61     else:
62         X = [x]
63         Y = [y]
64
65     index = 0
66     for i, m in zip(X, Y):
67         i = cv2.resize(i, size)
68         m = cv2.resize(m, size)
69
70         tmp_image_name = f"{name}_{index}.png"
71         tmp_mask_name = f"{name}_{index}.png"
72
73         image_path = os.path.join(save_path, "image", tmp_image_name)
74         mask_path = os.path.join(save_path, "mask", tmp_mask_name)
75
76         cv2.imwrite(image_path, i)
77         cv2.imwrite(mask_path, m)
78
79     index += 1
80
81 ## Usage ##
82
83 # if __name__ == "__main__":
84 #     """ Seeding """
85 #     np.random.seed(42)
86
87 #     """ Load the data """

```

Fig 5.3: Data Augmentation Function

```

augment.py X linknet.py dataset.py training_utils.py plotter.py .gitignore main.py
data > augment.py
80
81 ## Usage ##
82
83 # if __name__ == "__main__":
84 #     """ Seeding """
85 #     np.random.seed(42)
86
87 #     """ Load the data """
88 #     data_path = "<path/to/dataset>"
89 #     (train_x, train_y), (test_x, test_y) = load_data(data_path)
90
91 #     print(f"Train: {len(train_x)} - {len(train_y)}")
92 #     print(f"Test: {len(test_x)} - {len(test_y)}")
93
94 #     """ Create directories to save the augmented data """
95 #     create_dir("<path/to/train/image>")
96 #     create_dir("<path/to/train/mask>")
97 #     create_dir("<path/to/test/image>")
98 #     create_dir("<path/to/test/mask>")
99
100 #     """ Data augmentation """
101 #     augment_data(train_x, train_y, "<path/to/aug_data/train>", augment=True)
102 #     augment_data(test_x, test_y, "<path/to/aug_data/test>", augment=False)

```

Fig 5.4: Data Augmentation Function Call

```

linknet.py X dataset.py X training_utils.py plotter.py .gitignore main.py { weights_download
data > dataset.py
19
20         self.images_path = images_path
21         self.masks_path = masks_path
22         self.n_samples = len(images_path)
23
24     def __getitem__(self, index):
25
26         """ function overriden for extraction of items (images) from the dataset """
27
28         """ Reading image """
29         image = cv2.imread(self.images_path[index], cv2.IMREAD_COLOR)
30         image = image/255.0 ## (512, 512, 3)
31         image = np.transpose(image, (2, 0, 1)) ## (3, 512, 512)
32         image = image.astype(np.float32)
33         image = torch.from_numpy(image)
34
35         """ Reading mask """
36         mask = cv2.imread(self.masks_path[index], cv2.IMREAD_GRAYSCALE)
37         mask = mask/255.0 ## (512, 512)
38         mask = np.expand_dims(mask, axis=0) ## (1, 512, 512)
39         mask = mask.astype(np.float32)
40         mask = torch.from_numpy(mask)
41
42         return image, mask
43
44     def __len__(self):
45
46         """ function overriden for no. of images """
47         return self.n_samples
48
49

```

Fig 5.5: PyTorch Data Loader

5.2 Model Architecture

The screenshot shows a code editor with multiple tabs at the top: linknet.py (active), training_utils.py, plotter.py, .gitignore, main.py, weights_download.json, inference.py, and test.py. The code in linknet.py is as follows:

```
linkseg > linknet.py
39     self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size, stride, padding, groups=groups, bias=bias)
40     self.bn1 = nn.BatchNorm2d(out_planes)
41     self.relu = nn.ReLU(inplace=True)
42     self.conv2 = nn.Conv2d(out_planes, out_planes, kernel_size, 1, padding, groups=groups, bias=bias)
43     self.bn2 = nn.BatchNorm2d(out_planes)
44     self.downsample = None
45
46     if stride > 1:
47         self.downsample = nn.Sequential(nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, bias=False),
48                                         nn.BatchNorm2d(out_planes),)
49
50 def forward(self, x):
51
52     residual = x
53
54     out = self.conv1(x)
55     out = self.bn1(out)
56     out = self.relu(out)
57
58     out = self.conv2(out)
59     out = self.bn2(out)
60
61     if self.downsample is not None:
62         residual = self.downsample(x)
63
64     out += residual
65     out = self.relu(out)
66
67     return out
68
69
```

Fig 5.6: Basic Block

The screenshot shows a code editor with multiple tabs at the top: linknet.py (active), training_utils.py, plotter.py, .gitignore, main.py, weights_download.json, inference.py, and test.py. The code in linknet.py is as follows:

```
linkseg > linknet.py
75     """
76     Parameters:
77
78     - in_planes: no. of input channels
79
80     - out_planes: no. of output channels
81
82     - kernel_size: kernel size for conv in each block
83
84     - stride(default: 1): stride to be assigned to each block
85
86     - padding(default: 0): amount of padding to be assigned to each block
87
88     - groups(default: 1): groups to be assigned to each block
89
90     - bias(default: False): boolean bias to be assigned to each block
91
92     """
93
94     super(Encoder, self).__init__()
95
96     self.block1 = BasicBlock(in_planes, out_planes, kernel_size, stride, padding, groups, bias)
97     self.block2 = BasicBlock(out_planes, out_planes, kernel_size, 1, padding, groups, bias)
98
99 def forward(self, x):
100
101     x = self.block1(x)
102     x = self.block2(x)
103
104     return x
105
```

Fig. 5.7: Residual Blocks

```

super(Decoder, self).__init__()

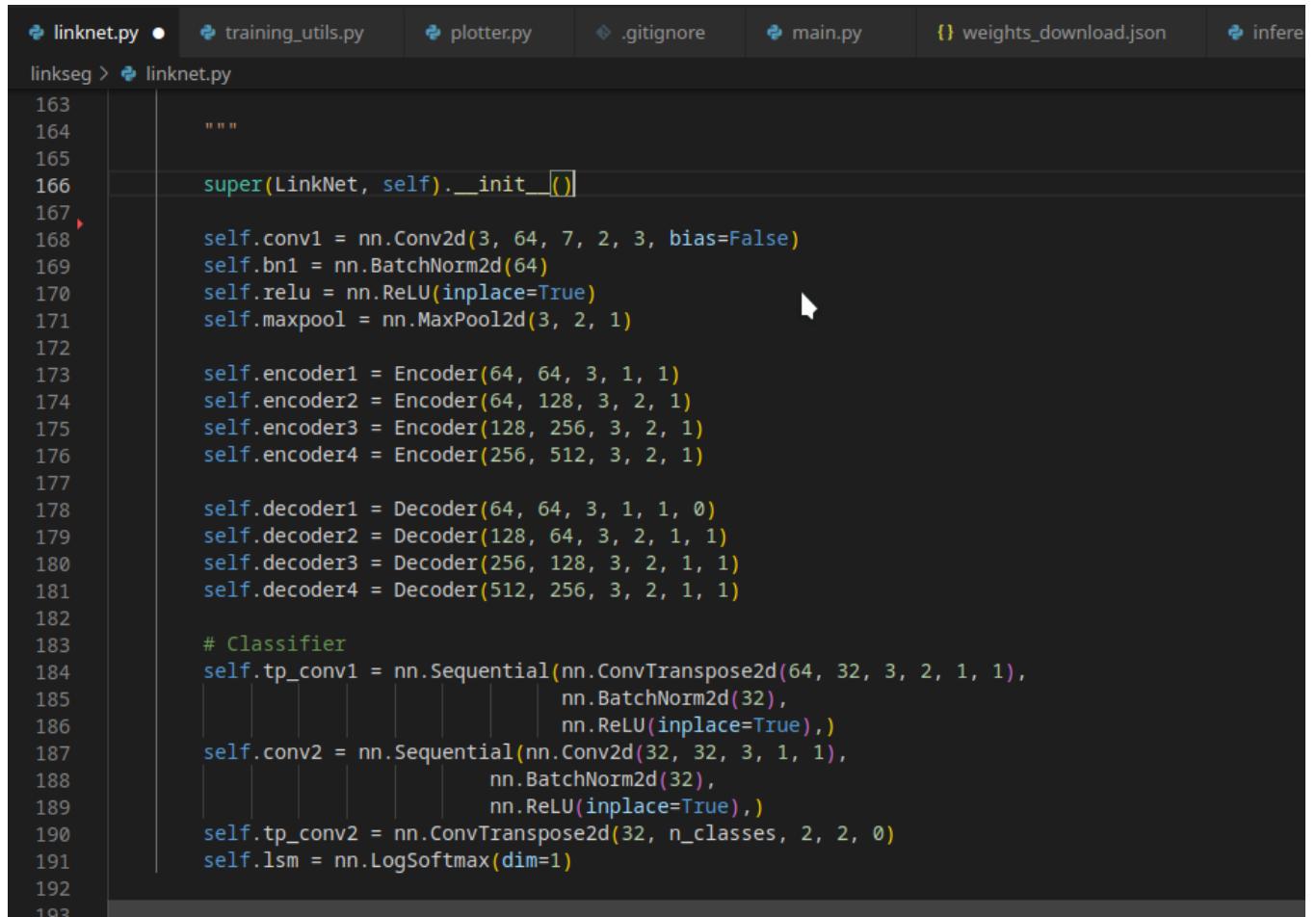
self.conv1 = nn.Sequential(nn.Conv2d(in_planes, in_planes//4, 1, 1, 0, bias=bias),
                         nn.BatchNorm2d(in_planes//4),
                         nn.ReLU(inplace=True))
self.tp_conv = nn.Sequential(nn.ConvTranspose2d(in_planes//4, in_planes//4, kernel_size, stride, padding, output_padding, bias=bias),
                            nn.BatchNorm2d(in_planes//4),
                            nn.ReLU(inplace=True))
self.conv2 = nn.Sequential(nn.Conv2d(in_planes//4, out_planes, 1, 1, 0, bias=bias),
                         nn.BatchNorm2d(out_planes),
                         nn.ReLU(inplace=True))

def forward(self, x):
    x = self.conv1(x)
    x = self.tp_conv(x)
    x = self.conv2(x)

    return x

```

Fig 5.8: Decoder Blocks



```

linkseg > linknet.py
163
164
165
166     super(LinkNet, self).__init__()
167
168     self.conv1 = nn.Conv2d(3, 64, 7, 2, 3, bias=False)
169     self.bn1 = nn.BatchNorm2d(64)
170     self.relu = nn.ReLU(inplace=True)
171     self.maxpool = nn.MaxPool2d(3, 2, 1)
172
173     self.encoder1 = Encoder(64, 64, 3, 1, 1)
174     self.encoder2 = Encoder(64, 128, 3, 2, 1)
175     self.encoder3 = Encoder(128, 256, 3, 2, 1)
176     self.encoder4 = Encoder(256, 512, 3, 2, 1)
177
178     self.decoder1 = Decoder(64, 64, 3, 1, 1, 0)
179     self.decoder2 = Decoder(128, 64, 3, 2, 1, 1)
180     self.decoder3 = Decoder(256, 128, 3, 2, 1, 1)
181     self.decoder4 = Decoder(512, 256, 3, 2, 1, 1)
182
183     # Classifier
184     self.tp_conv1 = nn.Sequential(nn.ConvTranspose2d(64, 32, 3, 2, 1, 1),
185                                 nn.BatchNorm2d(32),
186                                 nn.ReLU(inplace=True))
187     self.conv2 = nn.Sequential(nn.Conv2d(32, 32, 3, 1, 1),
188                             nn.BatchNorm2d(32),
189                             nn.ReLU(inplace=True))
190     self.tp_conv2 = nn.ConvTranspose2d(32, n_classes, 2, 2, 0)
191     self.lsm = nn.LogSoftmax(dim=1)
192
193

```

Fig. 5.9: LinkNet Blocks

The screenshot shows a code editor with multiple tabs at the top: linknet.py, training_utils.py, plotter.py, .gitignore, main.py, and weight. The active tab is linknet.py, which contains the following code:

```
207     def forward(self, x):
208         # Initial block
209         x = self.conv1(x)
210         x = self.bn1(x)
211         x = self.relu(x)
212         x = self.maxpool(x)
213
214         # Encoder blocks
215         e1 = self.encoder1(x)
216         e2 = self.encoder2(e1)
217         e3 = self.encoder3(e2)
218         e4 = self.encoder4(e3)
219
220         # Decoder blocks
221         d4 = e3 + self.decoder4(e4)
222         d3 = e2 + self.decoder3(d4)
223         d2 = e1 + self.decoder2(d3)
224         d1 = x + self.decoder1(d2)
225
226         # Classifier
227         y = self.tp_conv1(d1)
228         y = self.conv2(y)
229         y = self.tp_conv2(y)
230
231         y = self.lsm(y)
232
233         return y
234
235
236
```

Fig. 5.10: LinkNet Architecture

5.3 Training

```
linkseg > loss.py
 1  import torch
 2  import torch.nn as nn
 3
 4  import warnings
 5  warnings.filterwarnings('ignore')
 6
 7  class DiceLoss(nn.Module):
 8
 9      def __init__(self, weight=None, size_average=True):
10          super(DiceLoss, self).__init__()
11
12      def forward(self, inputs, targets, smooth=1e-6):
13
14          #comment out if your model contains a sigmoid or equivalent activation layer
15          inputs = torch.sigmoid(inputs)
16
17          #flatten label and prediction tensors
18          inputs = inputs.view(-1)
19          targets = targets.view(-1)
20
21          intersection = (inputs * targets).sum()
22          dice = (2.*intersection + smooth)/(inputs.sum() + targets.sum() + smooth)
23
24          return dice, 1 - dice
25
```

Fig. 5.11: Dice Loss

```
26  class IoU(nn.Module):
27
28      def __init__(self, weight=None, size_average=True):
29          super(IoU, self).__init__()
30
31      def forward(self, inputs, targets, smooth=1e-6):
32
33          #comment out if your model contains a sigmoid or equivalent activation layer
34          inputs = torch.sigmoid(inputs)    →
35
36          #flatten label and prediction tensors
37          inputs = inputs.view(-1)
38          targets = targets.view(-1)
39
40          intersection = (inputs * targets).sum()
41          iou = (intersection + smooth)/(inputs.sum() + targets.sum() - intersection + smooth)
42
43          return iou, 1 - iou
```

Fig 5.12: IoU Loss

```

model.train()

if torch.cuda.is_available():
    print("Shifting the model to cuda!")
    model.cuda()

epoch_loss1 = 0.0
epoch_loss2 = 0.0

for x,y in loader:

    x = x.to(self.device, dtype=torch.float32)
    y = y.to(self.device, dtype=torch.float32)

    optimizer.zero_grad()
    y_pred = model(x)

    score1,loss1 = self.loss_fn1(y_pred, y)
    score2,loss2 = self.loss_fn2(y_pred, y)

    loss1.backward(retain_graph = True)
    loss2.backward(retain_graph = True)

    optimizer.step()

    epoch_loss1 += loss1.item()
    epoch_loss2 += loss2.item()

epoch_loss1 = epoch_loss1/len(loader)
epoch_loss2 = epoch_loss2/len(loader)

```

Fig 5.13: Training Function

```

model.eval()

if torch.cuda.is_available():
    print("Shifting the model to cuda!")
    model.cuda()

epoch_loss1 = 0.0
epoch_loss2 = 0.0

with torch.no_grad():
    for x,y in loader:

        x = x.to(self.device, dtype=torch.float32)
        y = y.to(self.device, dtype=torch.float32)

        # optimizer.zero_grad()
        y_pred = model(x)

        score1,loss1 = self.loss_fn1(y_pred, y)
        score2,loss2 = self.loss_fn2(y_pred, y)

        epoch_loss1 += loss1.item()
        epoch_loss2 += loss2.item()

    epoch_loss1 = epoch_loss1/len(loader)
    epoch_loss2 = epoch_loss2/len(loader)

    print("\nValidation Dice Loss: {}, ".format(epoch_loss1),"Validation IoU Loss: {}, ".format(epoch_loss2))

```

Fig 5.14: Validation Function

```
Thu Mar  9 15:47:44 2023
+-----+
| NVIDIA-SMI 525.85.12      Driver Version: 525.85.12      CUDA Version: 12.0 |
+-----+
| GPU  Name      Persistence-M | Bus-Id     Disp.A  Volatile Uncorr. ECC | | | |
| Fan  Temp     Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|          |          |          |             | GPU-Util  Memory M. |
|-----+
|    0  Tesla T4           Off  | 00000000:00:04.0 Off   |                0 |
|    N/A   58C     P0    28W /  70W |          0MiB / 15360MiB |      0%     Default |
|                               |                  N/A      |
+-----+
+-----+
| Processes:
| GPU  GI CI      PID   Type  Process name          GPU Memory |
|          ID  ID
|-----|
|        No running processes found
+-----+
```

```
[ ] 1 from google.colab import drive
```

Fig 5.15: Nvidia Tesla T4

```
[ ] 1 import os
2 from getpass import getpass
3
4 user = getpass("Github User:")
5 password = getpass("Github Password:")
6 os.environ['GITHUB_AUTH'] = user + ':' + password

Github User:.....
Github Password:.....
```

```
[ ] 1 REPO_DIR = '/demo'
2 %mkdir -p "$REPO_DIR"
3 %cd "$REPO_DIR"
4 !git clone https://$GITHUB_AUTH@github.com/srijarkoroy/LinkSeg
5 %cd "$REPO_DIR/LinkSeg"

/demo
Cloning into 'LinkSeg'...
remote: Enumerating objects: 116, done.
remote: Counting objects: 100% (116/116), done.
remote: Compressing objects: 100% (86/86), done.
remote: Total 116 (delta 51), reused 84 (delta 23), pack-reused 0
Receiving objects: 100% (116/116), 654.19 KiB | 17.68 MiB/s, done.
Resolving deltas: 100% (51/51), done.
```

Fig 5.15: Cloning

```

[ ] 1 !unzip /content/drive/MyDrive/datasets.zip -d /demo/LinkSeg
Archive: /content/drive/MyDrive/datasets.zip
warning: stripped absolute path spec from /
mapname: conversion of failed
extracting: /demo/LinkSeg/test.zip
extracting: /demo/LinkSeg/training.zip

[ ] 1 %mkdir dataset/

[ ] 1 !unzip /demo/LinkSeg/test.zip -d /demo/LinkSeg/dataset/
Archive: /demo/LinkSeg/test.zip
creating: /demo/LinkSeg/dataset/test/
creating: /demo/LinkSeg/dataset/test/images/
inflating: /demo/LinkSeg/dataset/test/images/01_test.tif
inflating: /demo/LinkSeg/dataset/test/images/02_test.tif
inflating: /demo/LinkSeg/dataset/test/images/03_test.tif
inflating: /demo/LinkSeg/dataset/test/images/04_test.tif
inflating: /demo/LinkSeg/dataset/test/images/05_test.tif
inflating: /demo/LinkSeg/dataset/test/images/06_test.tif
inflating: /demo/LinkSeg/dataset/test/images/07_test.tif
inflating: /demo/LinkSeg/dataset/test/images/08_test.tif
inflating: /demo/LinkSeg/dataset/test/images/09_test.tif

```

Fig 5.17: Unzipping the Data

```

[ ] 1 data_path = "/demo/LinkSeg/dataset"
2 print(data_path)

/demo/LinkSeg/dataset

[ ] 1 !cat data/augment.py
import os
import numpy as np
import cv2
from glob import glob
from tqdm import tqdm

import imageio
from albumentations import HorizontalFlip, VerticalFlip, Rotate

def create_dir(path):
    """ Create a directory
    if not os.path.exists(path):
        os.makedirs(path)

def load_data(path):
    """

```

Fig 5.18: Checking the Data Directory

```

[ ] 1 from data.augment import load_data
[ ] 2 (train_x, train_y), (test_x, test_y) = load_data(data_path)

[ ] 1 print(f"Train: {len(train_x)} - {len(train_y)}")
2 print(f"Test: {len(test_x)} - {len(test_y)}")

Train: 20 - 20
Test: 20 - 20

[ ] 1 from data.augment import create_dir
2
3 create_dir("/demo/LinkSeg/new_data/train/image")
4 create_dir("/demo/LinkSeg/new_data/train/mask")
5 create_dir("/demo/LinkSeg/new_data/test/image/")
6 create_dir("/demo/LinkSeg/new_data/test/mask/")

[ ] 1 from data.augment import augment_data
2
3 augment_data(train_x, train_y, "/demo/LinkSeg/new_data/train", augment=True)
4 augment_data(test_x, test_y, "/demo/LinkSeg/new_data/test", augment=False)

100%|██████████| 20/20 [00:01<00:00, 11.26it/s]
100%|██████████| 20/20 [00:00<00:00, 36.20it/s]

```

Fig 5.19: Setup for Augmentation

```

[ ] 100%|██████████| 20/20 [00:01<00:00, 11.26it/s]
100%|██████████| 20/20 [00:00<00:00, 36.20it/s]

[ ] 1 import torch
2 from torch.utils.data import DataLoader
3
4 import os
5 from glob import glob
6
7 from data.augment import load_data
8 from data.dataset import DriveDataset
9
10
11 data_path = "/demo/LinkSeg/new_data"
12
13 train_x = sorted(glob(os.path.join(data_path, "train", "image", "*.*")))
14 train_y = sorted(glob(os.path.join(data_path, "train", "mask", "*.*")))
15
16 valid_x = sorted(glob(os.path.join(data_path, "test", "image", "*.*")))
17 valid_y = sorted(glob(os.path.join(data_path, "test", "mask", "*.*")))
18
19 print(f"Train: {len(train_x)} - {len(train_y)}")
20 print(f"Test: {len(valid_x)} - {len(valid_y)}")
21

```

Fig 5.20: Augmented Data

```

18
[ ] 19 print(f"Train: {len(train_x)} - {len(train_y)}")
20 print(f"Test: {len(valid_x)} - {len(valid_y)}")
21
22 train_dataset = DriveDataset(train_x, train_y)
23
24 train_loader = DataLoader(
25     dataset=train_dataset,
26     batch_size=4,
27     shuffle=True,
28     num_workers=2
29 )
30
31 val_dataset = DriveDataset(valid_x, valid_y)
32
33 val_loader = DataLoader(
34     dataset=val_dataset,
35     batch_size=4,
36     shuffle=False,
37     num_workers=2
38 )

```

```

Train: 80 - 80
Test: 20 - 20

```

Fig 5.21: Training Data Setup

```

__all__ = [
    DiceLoss,
    IoU,
    LinkNet,
    Train,
    Evaluate,
]

from linkseg import LinkNet, DiceLoss, IoU, Train, Evaluate
from linkseg import plotter
from tqdm import tqdm
# Training and Evaluate object
train = Train(dice=DiceLoss(), iou=IoU())
eval = Evaluate(dice=DiceLoss(), iou=IoU())
# Model Initialization and setting up hyperparameters
model = LinkNet()
print(model)
# Hyperparameters
lr = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
epochs = 100

```

Fig 5.22: Hyperparameters

```

21
22 # Training
23 for epoch in tqdm(range(epochs)):
24     print("Epoch: ", epoch)
25
26     train_dice, train_iou = train.forward(model=model, loader=train_loader, optimizer=optimizer)
27     val_dice, val_iou = eval.forward(model=model, loader=val_loader)
28
29     dice_loss.append(train_dice)
30     iou_loss.append(train_iou)
31
32 plotter.plot(dice_loss, iou_loss)
33
34 torch.save(model.state_dict(), 'linknet.pth')

LinkNet(
    (encoder1): EncoderBlock(
        (conv): ConvBlock(
            (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU()
        )
        (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
    )
)

```

Fig 5.23: Training Function Call

```

0%|          | 0/100 [00:00<?, ?it/s]Epoch:  0
Shift the model to cuda!
Train Dice Loss: 0.7625233799219131, Train IoU Loss: 0.8645183622837067,
Shift the model to cuda!
1%|          | 1/100 [00:36<1:00:29, 36.67s/it]
Validation Dice Loss: 0.40112115144729615, Validation IoU Loss: 0.5725687265396118,
Epoch:  1
Shift the model to cuda!
Train Dice Loss: 0.6659743875265122, Train IoU Loss: 0.7993300944566727,
Shift the model to cuda!
2%||         | 2/100 [01:10<57:24, 35.15s/it]
Validation Dice Loss: 0.6646595597267151, Validation IoU Loss: 0.7985406279563904,
Epoch:  2
Shift the model to cuda!
Train Dice Loss: 0.6241493344306945, Train IoU Loss: 0.7683533012866974,
Shift the model to cuda!
3%||         | 3/100 [01:44<55:47, 34.51s/it]
Validation Dice Loss: 0.5649534702301026, Validation IoU Loss: 0.7219907999038696,
Epoch:  3
Shift the model to cuda!
Train Dice Loss: 0.6021922618150711, Train IoU Loss: 0.7515868365764617,
Shift the model to cuda!
4%||         | 4/100 [02:19<55:37, 34.77s/it]
Validation Dice Loss: 0.5023146510124207, Validation IoU Loss: 0.6687108755111695,
Epoch:  4
Shift the model to cuda!
Train Dice Loss: 0.5860089540481568, Train IoU Loss: 0.7388259142637252,

```

Fig 5.24: Training

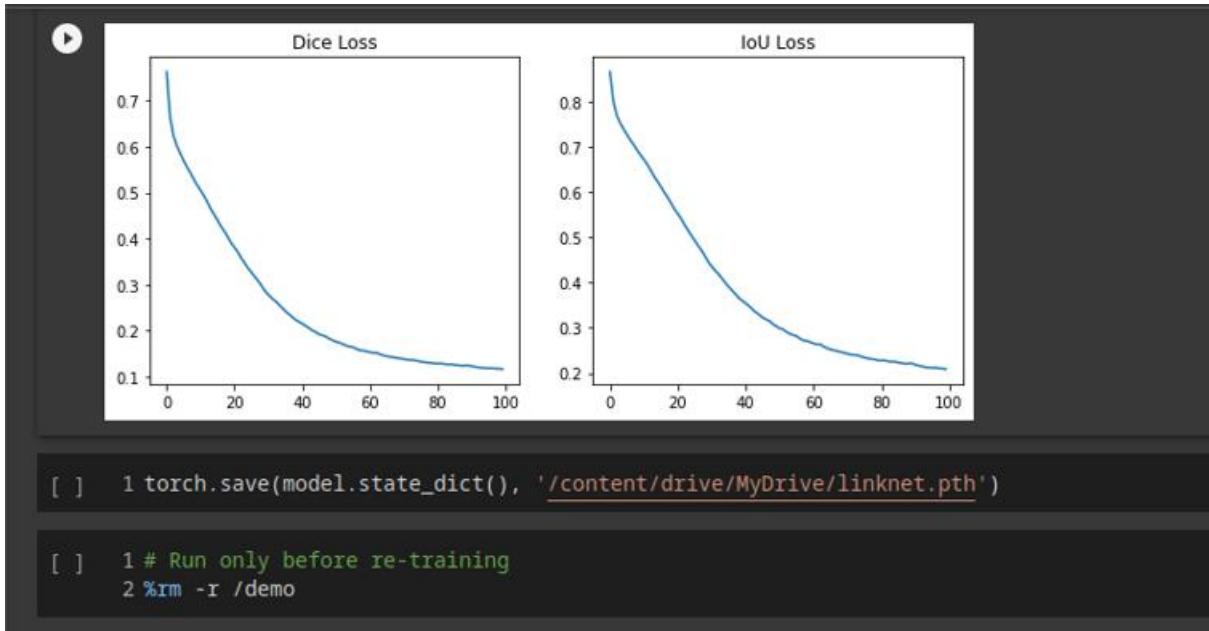


Fig 5.25: Training Progression

5.4 Testing

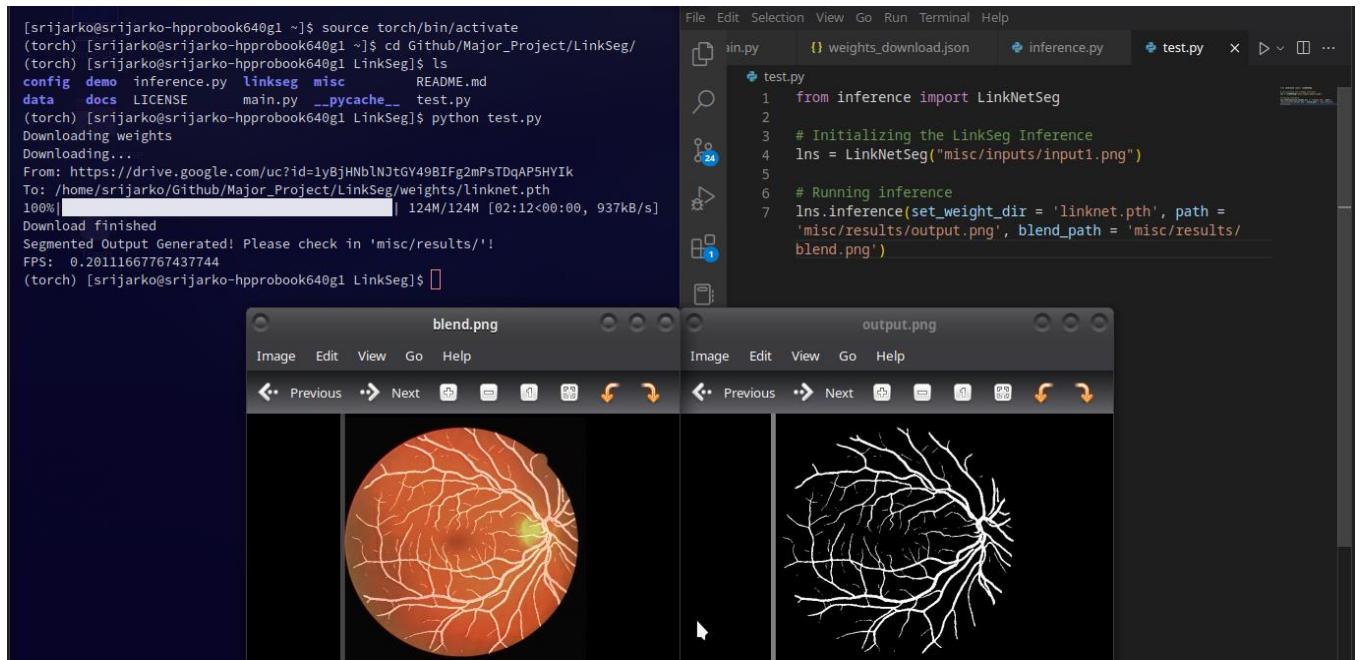


Fig 5.26: Testing

5.5 User Interface

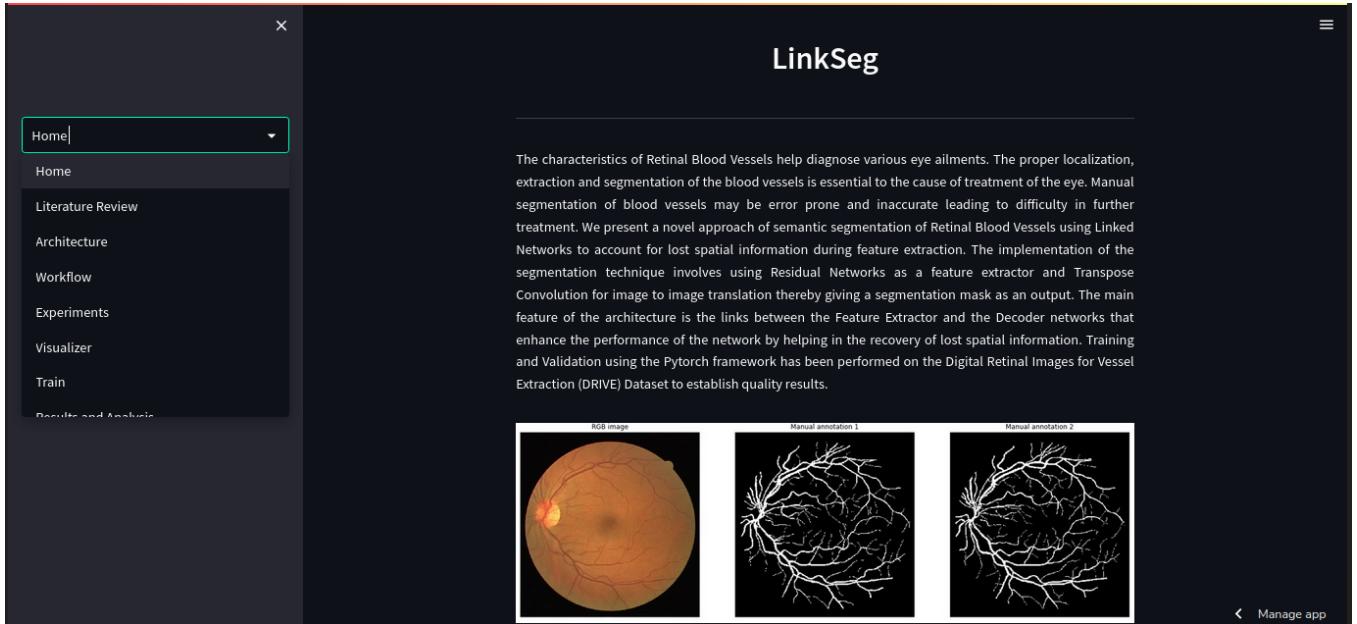


Fig 5.27: Home Page

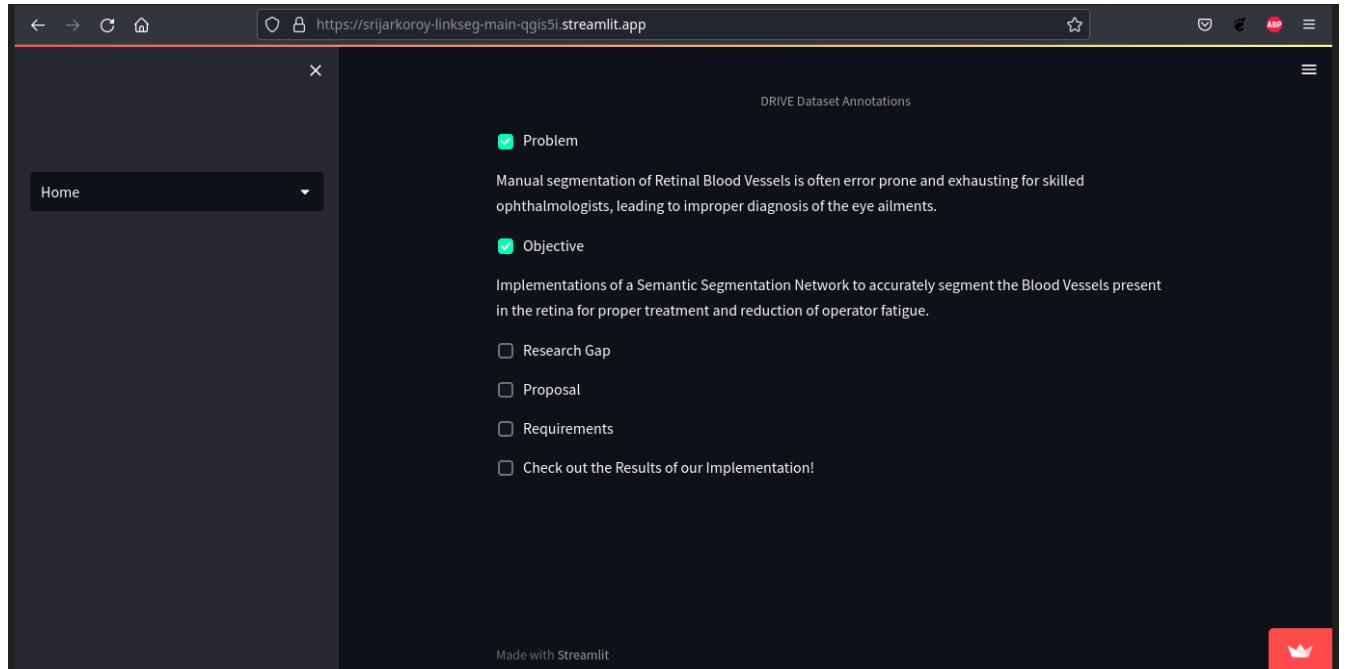


Fig 5.28: Problem and Objective

The screenshot shows a Streamlit application window with a dark theme. At the top, the URL is https://srijarkoroy-linkseg-main-qgjls5i.streamlit.app. The left sidebar has a 'Home' button. The main content area has a title 'X'. Below it are several sections with checkboxes:

- Problem
- Objective
- Research Gap

The 'Research Gap' section contains text about existing systems for Retinal Blood Vessel Segmentation and their limitations.

- Proposal

The 'Proposal' section describes the proposed improved LinkNet architecture for semantic segmentation of Retinal Blood Vessels, listing two bullet points:

- The network makes use of skip connections to link each Encoder Block to its corresponding Decoder Block, passing on the lost spatial information to be concatenated during Upsampling.
- Using Upsample instead of Transpose Conv counters the problem of noisy output.

- Requirements
- Check out the Results of our Implementation!

Fig 5.29: Research Gap and Proposal

The screenshot shows a Streamlit application window with a dark theme. At the top, the URL is https://srijarkoroy-linkseg-main-qgjls5i.streamlit.app. The left sidebar has a 'Home' button. The main content area has a title '—'. Below it are several sections with checkboxes:

- Requirements

The 'Requirements' section is divided into Software Requirements and Hardware Requirements:

Software Requirements:

- Google Colab for Training
- Visual Studio Code
- Linux Terminal for local Testing
- Python
- PyTorch

Hardware Requirements:

- NVIDIA Tesla T4 GPU
- cuDNN VERSION 8005

- Check out the Results of our Implementation!

Fig 5.30: Requirements

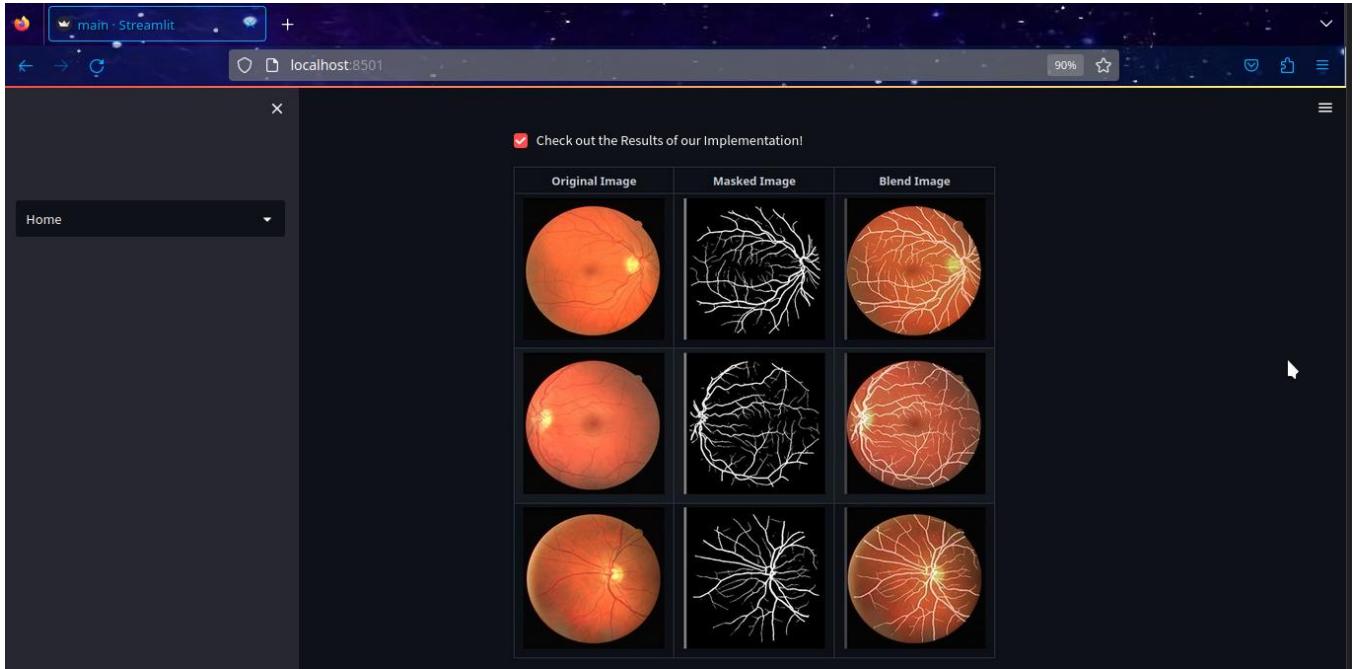


Fig 5.31: Results

The screenshot shows a Streamlit application interface. On the left is a sidebar with a dropdown menu currently set to "Literature Review". The main area has a title "Literature Review". Below the title is a table comparing two academic papers:

Paper	Author(s)	Dataset	Method	Metrics	Year
Segmentation of retinal blood vessels by combining the detection of centerlines and morphological reconstruction	Mendonca et. al.	DRIVE	Morphological Processing	0.9452 (Accuracy)	2006
Morphology Approach for Features Extraction in Retinal Images for Diabetic Retionopathy Diagnosis	Ibrahim Abdurrazaq et. al.	DRIVE	Mathematical morphology theory and intensity transformation algorithm	TPF value of 0.8214 (True Positive Fraction)	2008

Fig 5.32: Literature Review - 1

Literature Review

Segmentation Scheme Based on the Improved Deep Learning U-Net Model	Xiuqin Pan et. al.	DRIVE	Improved U-Net	96.50% (Segmentation Accuracy)	2019
Retinal blood vessel segmentation based on heuristic image analysis	Maja Braovic et. al.	DRIVE and STARE	Modified SUSAN edge detector and shape analysis	96.33% and 96.10% (Accuracy) respectively	2019
Accurate Retinal Vessel Segmentation via Octave Convolution Neural Network	Zhun Fan et. al.	DRIVE and STARE and CHASE DB1 and and HRF	Octave UNet	0.9663 (Accuracy) on DRIVE Dataset	2019
Construction of Retinal Vessel Segmentation Models Based on Convolutional Neural Network	Qiangguo Jin et. al.	DRIVE and STARE	Deformable-ConvNet	Accuracy of 0.9628/0.9690	2020

Fig 5.33: Literature Review - 2

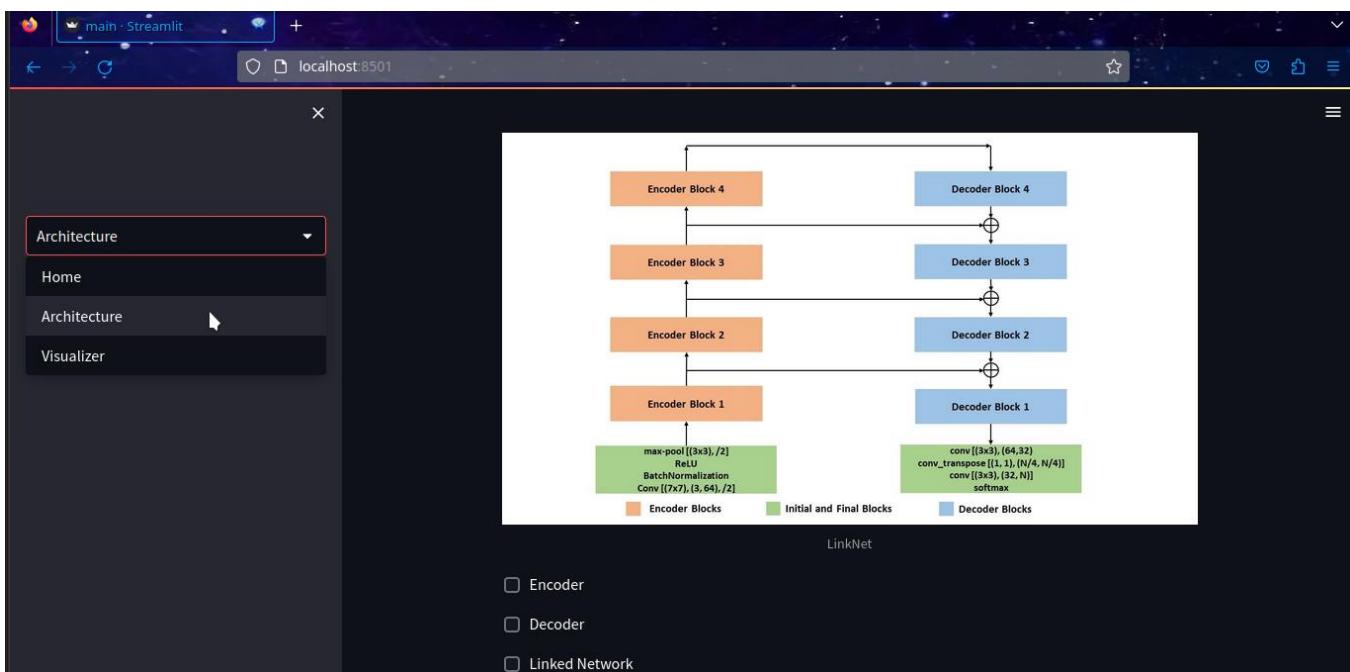


Fig 5.34: Architecture

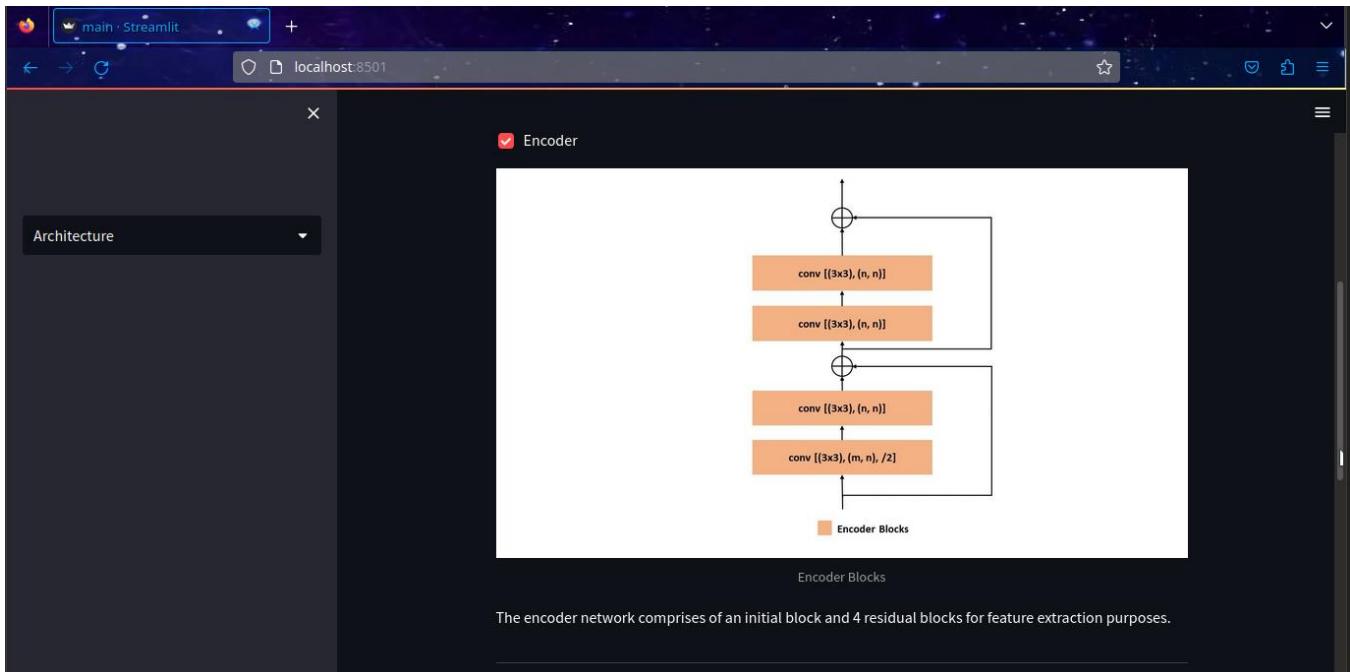


Fig 5.35: Encoder

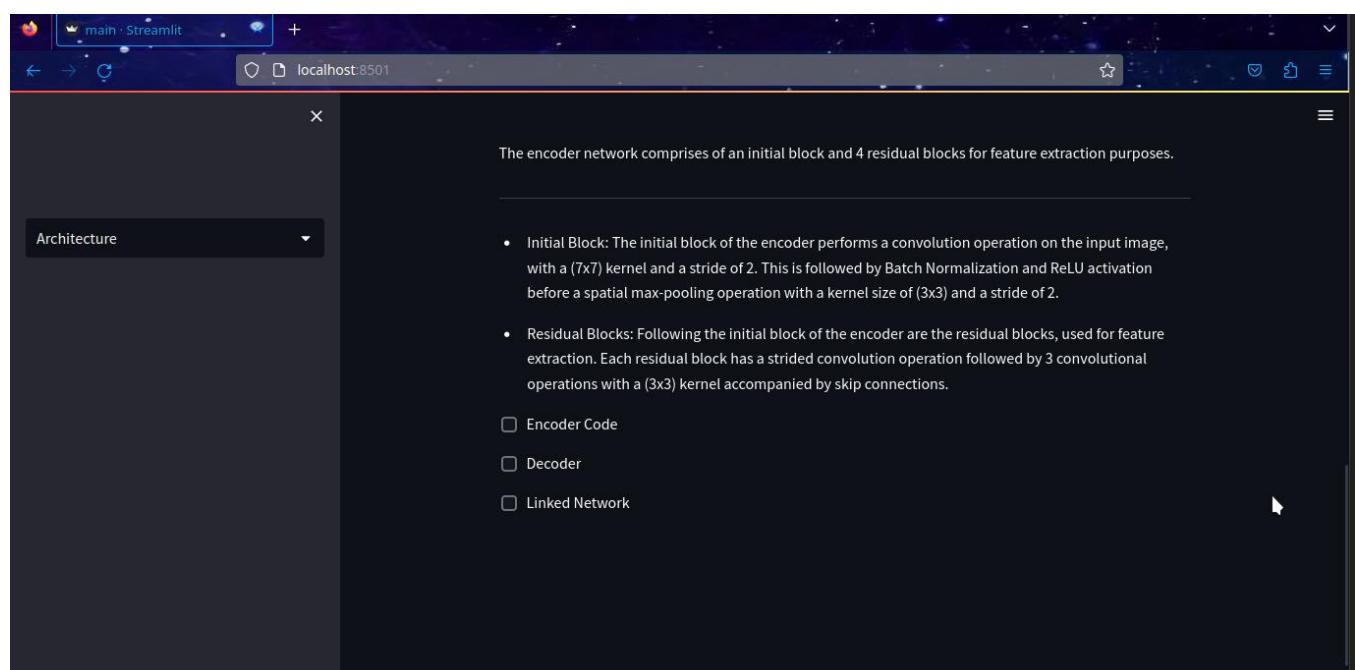


Fig 5.36: Encoder Theory

The screenshot shows a Streamlit application window titled "main - Streamlit". The URL "localhost:8501" is visible in the address bar. On the left, a sidebar menu has "Architecture" selected. In the main content area, a checkbox labeled "Encoder Code" is checked, revealing a block of Python code for an "Encoder" module. Below the code, there are two other options: "Decoder" and "Linked Network", each with an unchecked checkbox.

```

class Encoder(nn.Module):
    def __init__(self, in_planes, out_planes, kernel_size, stride=1, padding=0, groups=1):
        super(Encoder, self).__init__()
        self.block1 = BasicBlock(in_planes, out_planes, kernel_size, stride, padding, groups)
        self.block2 = BasicBlock(out_planes, out_planes, kernel_size, 1, padding, groups)

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)

        return x

```

Fig 5.37: Encoder Code

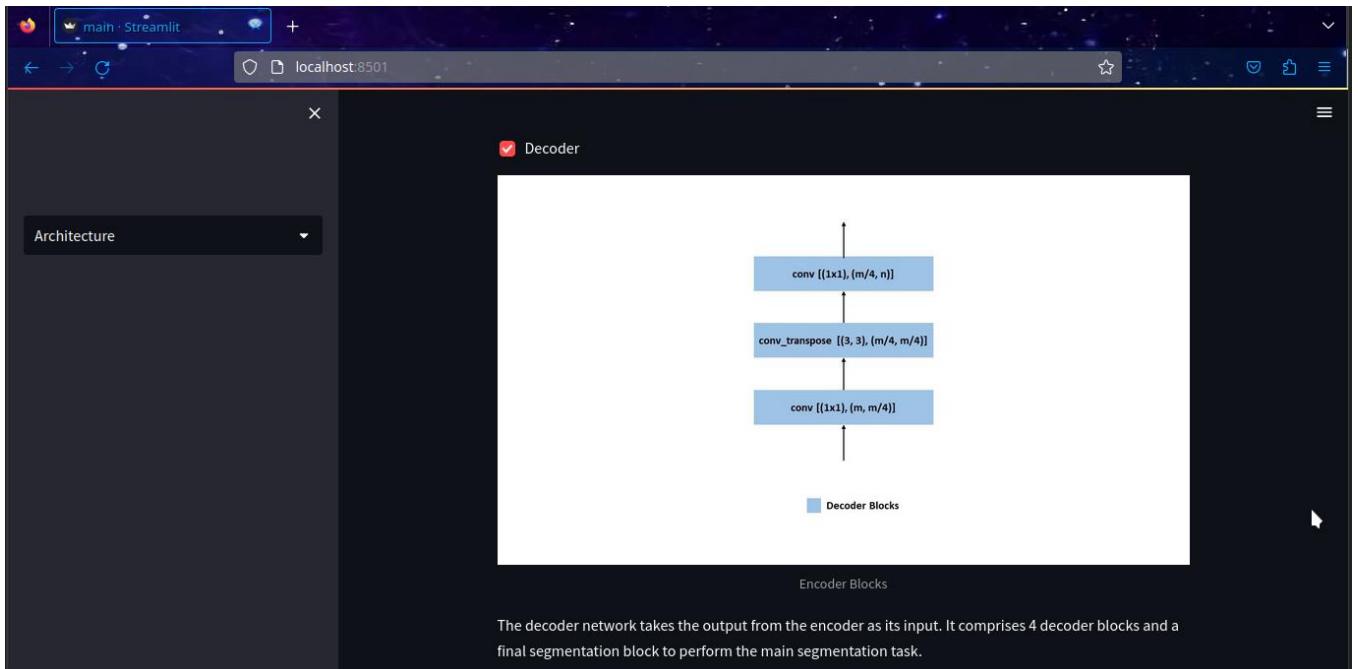


Fig 5.38: Decoder

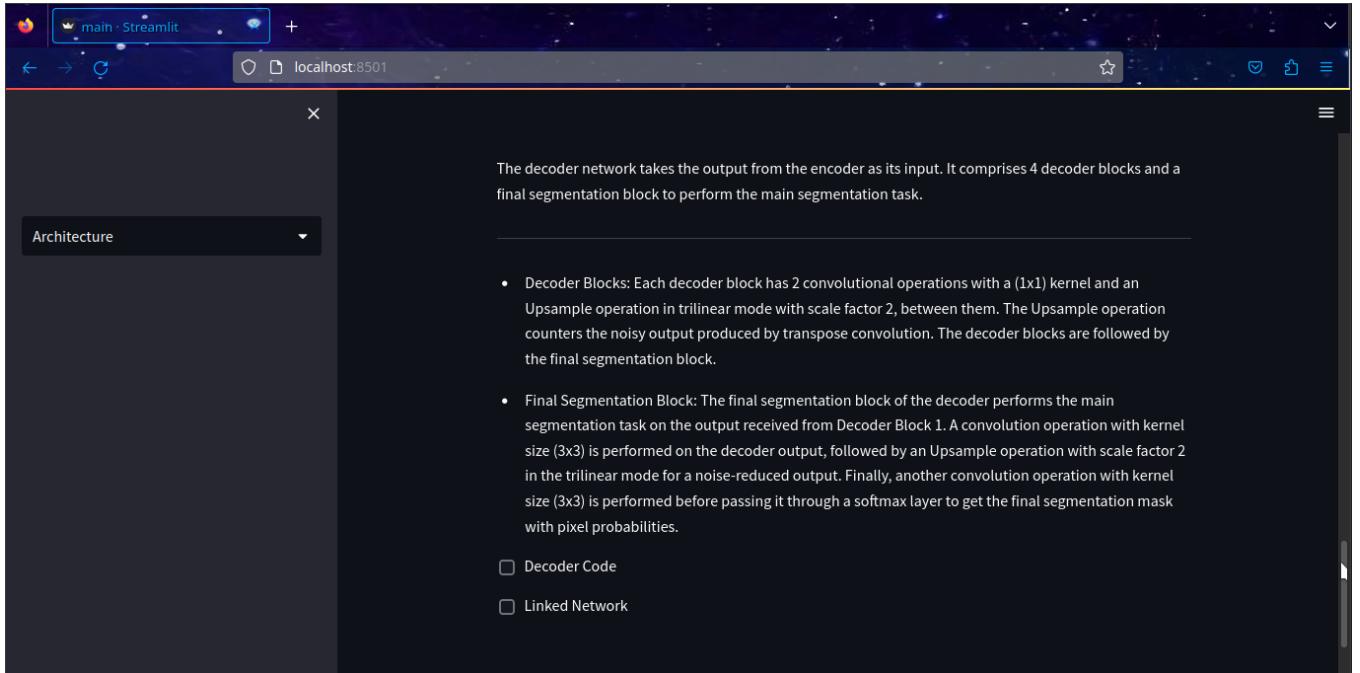


Fig 5.39: Decoder Theory

The screenshot shows a Streamlit application running at localhost:8501. The title bar says "main · Streamlit". The main content area has a dark background with white text. It displays Python code for a "Decoder" module:

```
class Decoder(nn.Module):
    def __init__(self, in_planes, out_planes, kernel_size, stride=1, padding=0, ou
        super(Decoder, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv2d(in_planes, in_planes//4, 1, 1, 0, bias=False),
                                nn.BatchNorm2d(in_planes//4),
                                nn.ReLU(inplace=True))
        self.tp_conv = nn.Sequential(nn.ConvTranspose2d(in_planes//4, in_planes//4,
                                                      kernel_size, stride=2, padding=1, bias=False),
                                    nn.BatchNorm2d(in_planes//4),
                                    nn.ReLU(inplace=True))
        self.conv2 = nn.Sequential(nn.Conv2d(in_planes//4, out_planes, 1, 1, 0, bias=False),
                                nn.BatchNorm2d(out_planes),
                                nn.ReLU(inplace=True))

    def forward(self, x):
        x = self.conv1(x)
        x = self.tp_conv(x)
        x = self.conv2(x)
        return x
```

Fig 5.40: Decoder Code

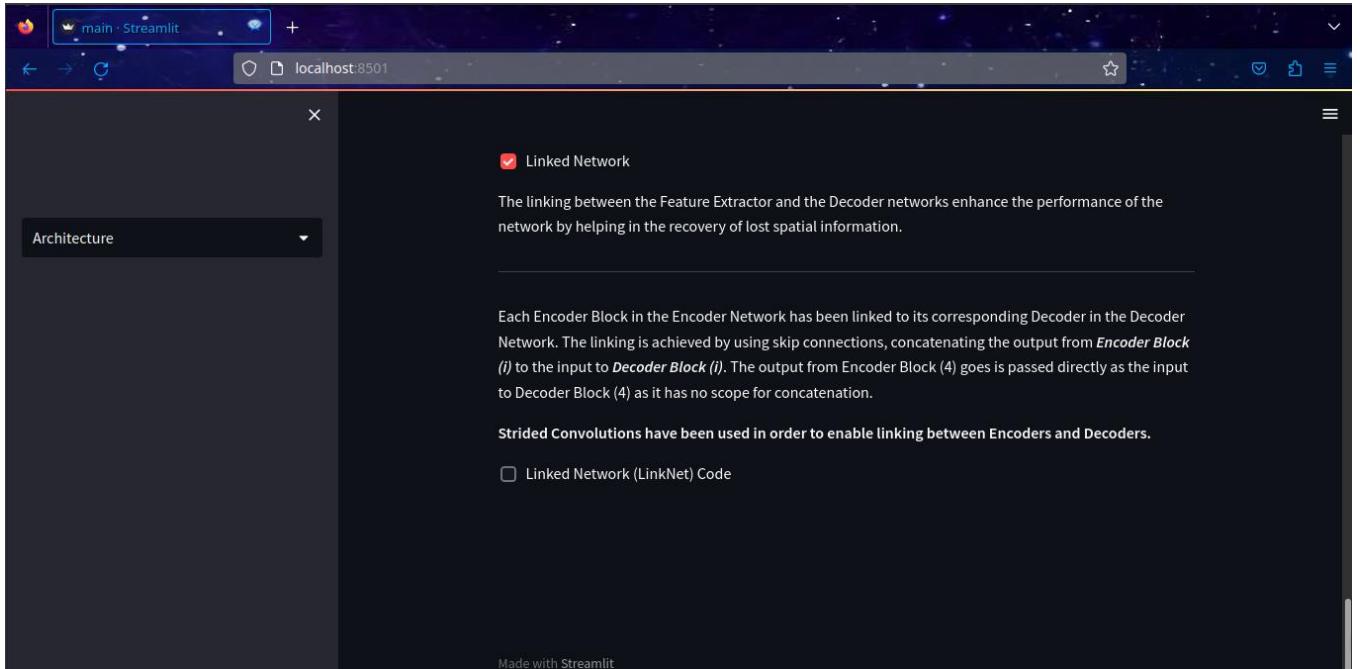


Fig 5.41: Linked Network

The screenshot shows a Streamlit application window titled "main · Streamlit" with the URL "localhost:8501". The main content area displays a large block of Python code for a neural network architecture:

```
self.conv1 = nn.Conv2d(3, 64, 7, 2, 3, bias=False)
self.bn1 = nn.BatchNorm2d(64)
self.relu = nn.ReLU(inplace=True)
self.maxpool = nn.MaxPool2d(3, 2, 1)

self.encoder1 = Encoder(64, 64, 3, 1, 1)
self.encoder2 = Encoder(64, 128, 3, 2, 1)
self.encoder3 = Encoder(128, 256, 3, 2, 1)
self.encoder4 = Encoder(256, 512, 3, 2, 1)

self.decoder1 = Decoder(64, 64, 3, 1, 1, 0)
self.decoder2 = Decoder(128, 64, 3, 2, 1, 1)
self.decoder3 = Decoder(256, 128, 3, 2, 1, 1)
self.decoder4 = Decoder(512, 256, 3, 2, 1, 1)

# Classifier
self.tp_conv1 = nn.Sequential(nn.ConvTranspose2d(64, 32, 3, 2, 1, 1),
                             nn.BatchNorm2d(32),
                             nn.ReLU(inplace=True))
self.conv2 = nn.Sequential(nn.Conv2d(32, 32, 3, 1, 1),
                         nn.BatchNorm2d(32),
                         nn.ReLU(inplace=True))
self.tp_conv2 = nn.ConvTranspose2d(32, n_classes, 2, 2, 0)
self.lsm = nn.LogSoftmax(dim=1)
```

Fig 5.42: Linked Network Code – 1

The screenshot shows a Streamlit application window titled "main · Streamlit" at "localhost:8501". The sidebar on the left has a dropdown menu set to "Architecture". The main area displays a block of Python code representing a neural network architecture:

```

# Initial block
x = self.conv1(x)
x = self.bn1(x)
x = self.relu(x)
x = self.maxpool(x)

# Encoder blocks
e1 = self.encoder1(x)
e2 = self.encoder2(e1)
e3 = self.encoder3(e2)
e4 = self.encoder4(e3)

# Decoder blocks
d4 = e3 + self.decoder4(e4)
d3 = e2 + self.decoder3(d4)
d2 = e1 + self.decoder2(d3)
d1 = x + self.decoder1(d2)

# Classifier
y = self.tp_conv1(d1)
y = self.conv2(y)
y = self.tp_conv2(y)

y = self.lsm(y)

```

Fig 5.43: Linked Network Code – 2

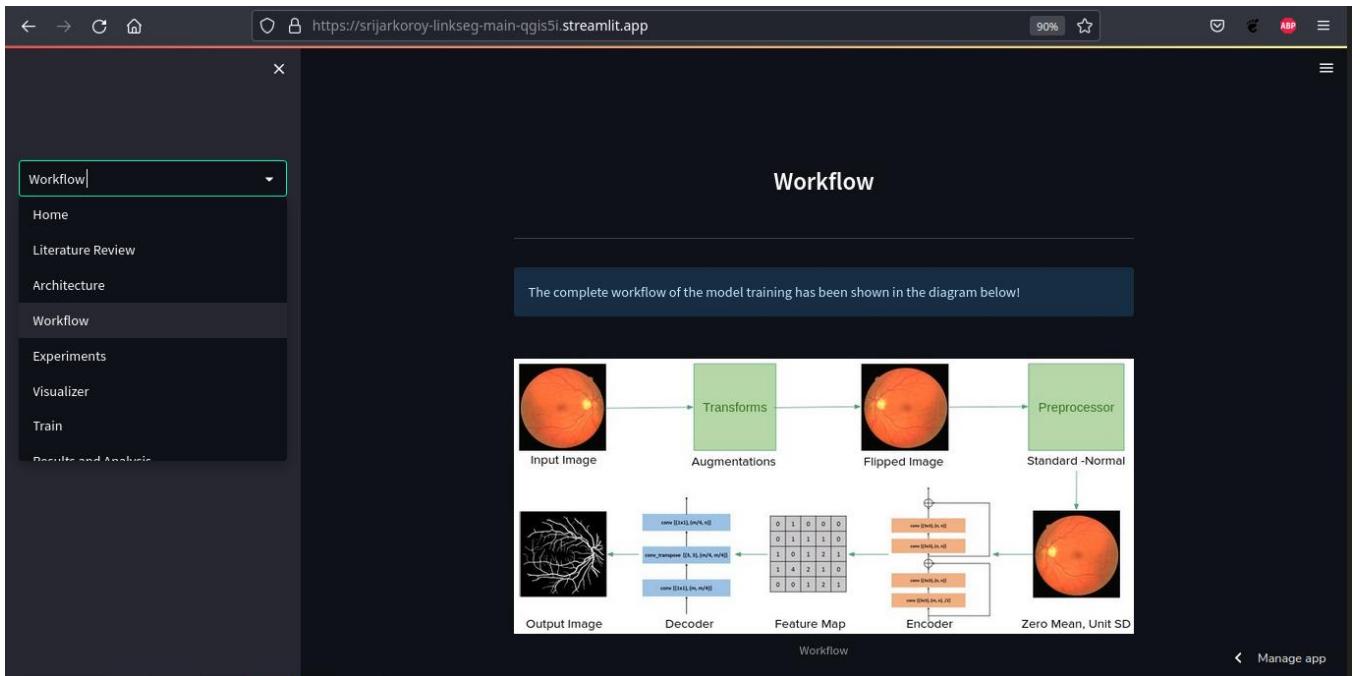


Fig 5.44: Workflow 1

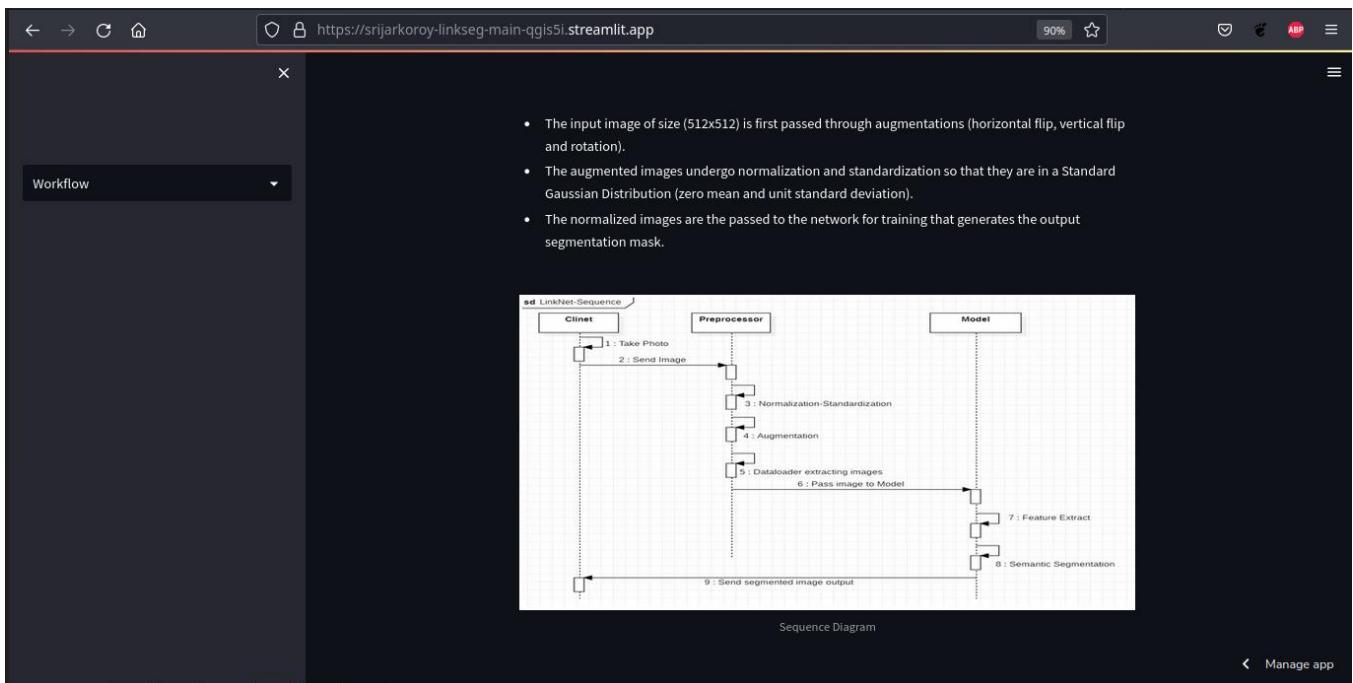


Fig 5.45: Workflow 2

Experiments

Home
Literature Review
Architecture
Workflow
Experiments
Visualizer
Train
Results and Analysis

Experiments

A. Dataset

We have used the Digital Retinal Images for Vessel Extraction (DRIVE) dataset for retinal vessel segmentation. It consists of a total of JPEG 40 color fundus images; including 7 abnormal pathology cases. Each image resolution is 584x565 pixels with eight bits per color channel (3 channels), resized to 512x512 for our model. The data has been augmented using the augmentations module. Data Augmentation has been accomplished by doing HorizontalFlip, VerticalFlip and Rotate, thereby generating 160 images for training.

Original, Horizontal Flip, Vertical Flip

Manage app

Fig 5.46: Dataset

The screenshot shows a Streamlit application window. In the top left, there's a sidebar labeled "Experiments". The main content area has a header "B. Metrics" and a section titled "Dice Loss:". Below it, a text block explains Dice Loss as a metric for evaluating model performance, mentioning its use in semantic segmentation and its formula: $\text{Dice Coefficient} = 2 |T \cap P| / |T| + |P|$. It also notes that the generalised loss function is calculated by subtracting the dice coefficient from 1. A checkbox labeled "Dice Loss Code" is checked, revealing a code block for a PyTorch module named `DiceLoss`:

```
class DiceLoss(nn.Module):
    def __init__(self, weight=None, size_average=True):
        super(DiceLoss, self).__init__()

    def forward(self, inputs, targets, smooth=1e-6):
        #comment out if your model contains a sigmoid or equivalent activation layer
        inputs = torch.sigmoid(inputs)

        #flatten label and prediction tensors
        inputs = inputs.view(-1)
        targets = targets.view(-1)

        intersection = (inputs * targets).sum()
        dice = (2.*intersection + smooth)/(inputs.sum() + targets.sum() + smooth)

        return dice, 1 - dice
```

Fig 5.47: Dice Score Theory

This screenshot shows the same Streamlit application as Fig 5.47, but the "Dice Loss Code" section is now expanded. The code block includes additional logic to handle targets and flatten them before calculating the intersection:

```
class DiceLoss(nn.Module):
    def __init__(self, weight=None, size_average=True):
        super(DiceLoss, self).__init__()

    def forward(self, inputs, targets, smooth=1e-6):
        #comment out if your model contains a sigmoid or equivalent activation layer
        inputs = torch.sigmoid(inputs)

        #flatten label and prediction tensors
        inputs = inputs.view(-1)
        targets = targets.view(-1)

        intersection = (inputs * targets).sum()
        dice = (2.*intersection + smooth)/(inputs.sum() + targets.sum() + smooth)

        return dice, 1 - dice
```

Fig 5.48: Dice Score Code

The screenshot shows a Streamlit application window. At the top, the URL is <https://srrijarkoroy-linkseg-main-qgis5i.streamlit.app>. The main content area has a dark background with white text. It starts with a section titled "IoU Loss:" which contains a detailed explanation of the metric and its formula. Below this, there is a checkbox labeled "IoU Loss Code" which, when checked, reveals a block of Python code for a PyTorch module named "IoU". The code defines the module's initialization and forward pass, including comments explaining the steps of calculating intersection and union tensors.

```
class IoU(nn.Module):
    def __init__(self, weight=None, size_average=True):
        super(IoU, self).__init__()

    def forward(self, inputs, targets, smooth=1e-6):
        #comment out if your model contains a sigmoid or equivalent activation layer
        inputs = torch.sigmoid(inputs)

        #flatten label and prediction tensors
        inputs = inputs.view(-1)
        targets = targets.view(-1)

        intersection = (inputs * targets).sum()
        iou = (intersection + smooth) / (inputs.sum() + targets.sum() - intersection)
        return iou, 1 - iou
```

Fig 5.49: IoU Theory

This screenshot is identical to Fig 5.49, showing the same Streamlit application for "Experiments". It displays the theoretical explanation of IoU Loss and the corresponding Python code for the "IoU" module. The code is identical to the one shown in Fig 5.49.

Fig 5.50: IoU Code

The PyTorch framework was used to code the architecture, and the NVIDIA Tesla T4 GPU and CUDA integration were used to train it on 80 images with a batch size of 4. Before being loaded into a dataloader object for training, each picture underwent pre-processing, was enhanced, normalised, and standardised. To reduce the dice and IoU loss, the model was trained using the Adam optimizer for 100 iterations at a learning rate 0.0001.

```
+-----+-----+-----+-----+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0 |
| GPU  Name      Persistence-M| Bus-Id     Disp.A  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|=====+=====+=====+=====+=====+=====+=====+=====+
| 0   Tesla T4      Off  | 00000000:00:04.0 Off |          0%          0 |
| N/A   58C   P0    28W / 70W |  8MIB / 15360MIB |     0%      Default |
|                               |                  MIG M. |
+-----+-----+-----+-----+
+-----+
| Processes:                   GPU Memory
| GPU  GI CI PID  Type  Process name        Usage
| ID  ID
| =====
| No running processes found
+-----+
```

NVIDIA Tesla T4 GPU Instance

Fig 5.51: Training Details

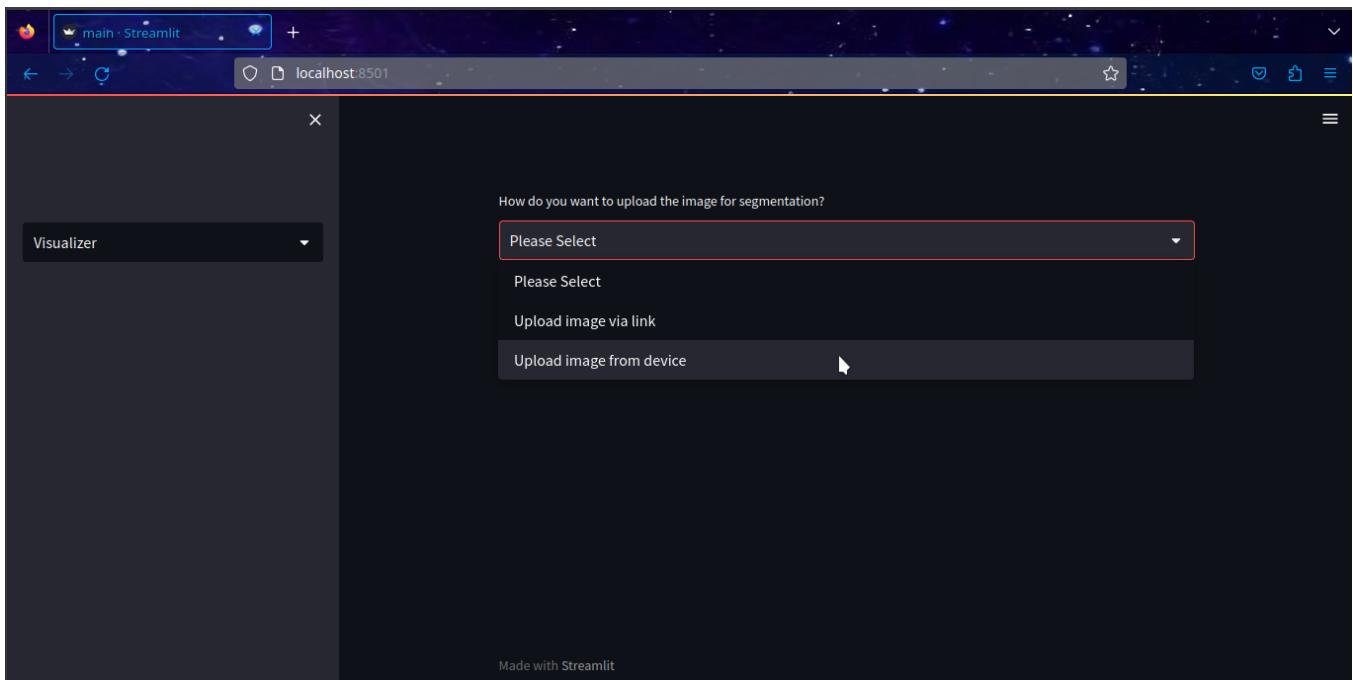


Fig 5.52: Visualizer

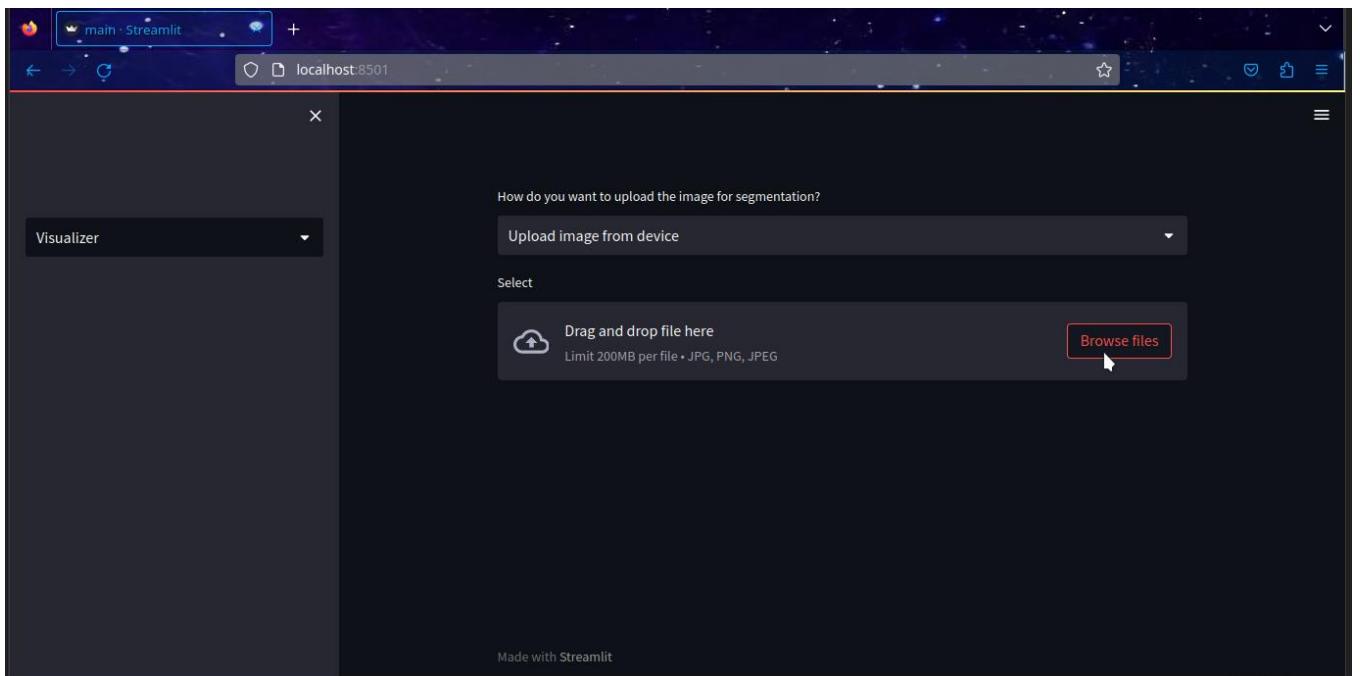


Fig 5.53: Development

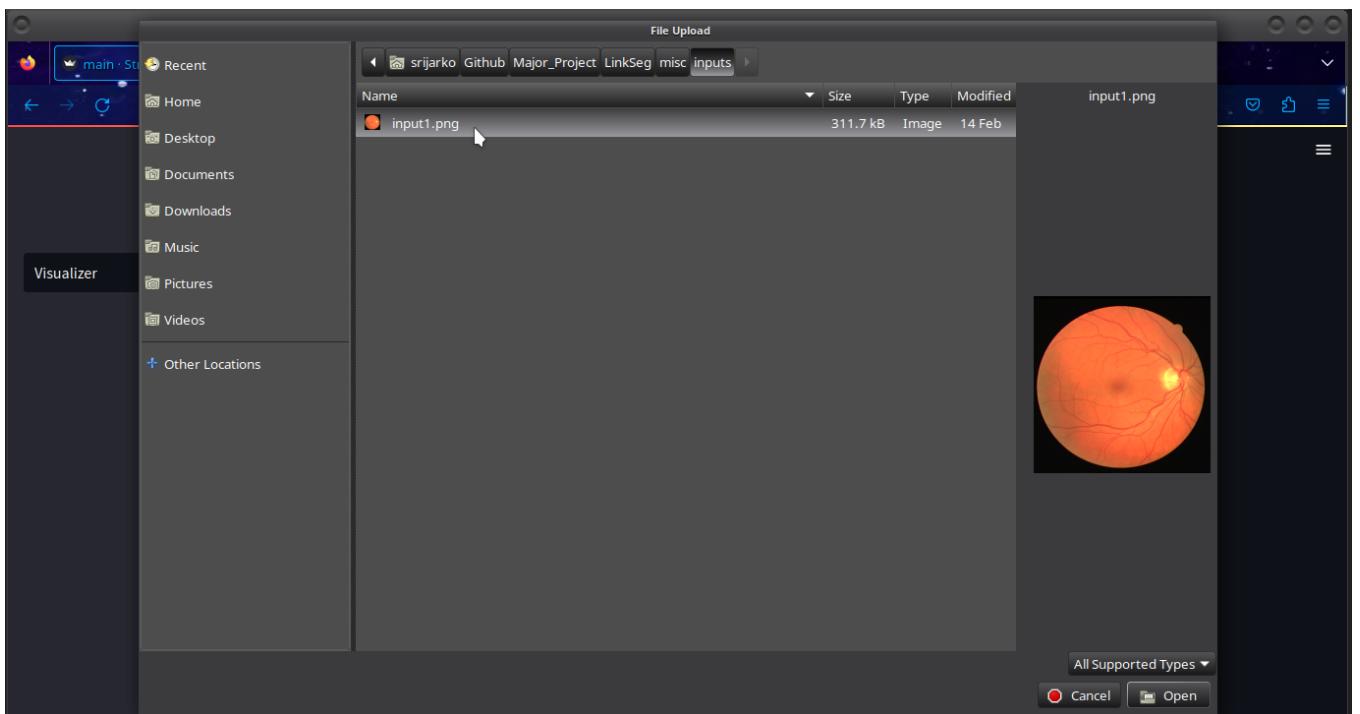


Fig 5.54: Image upload

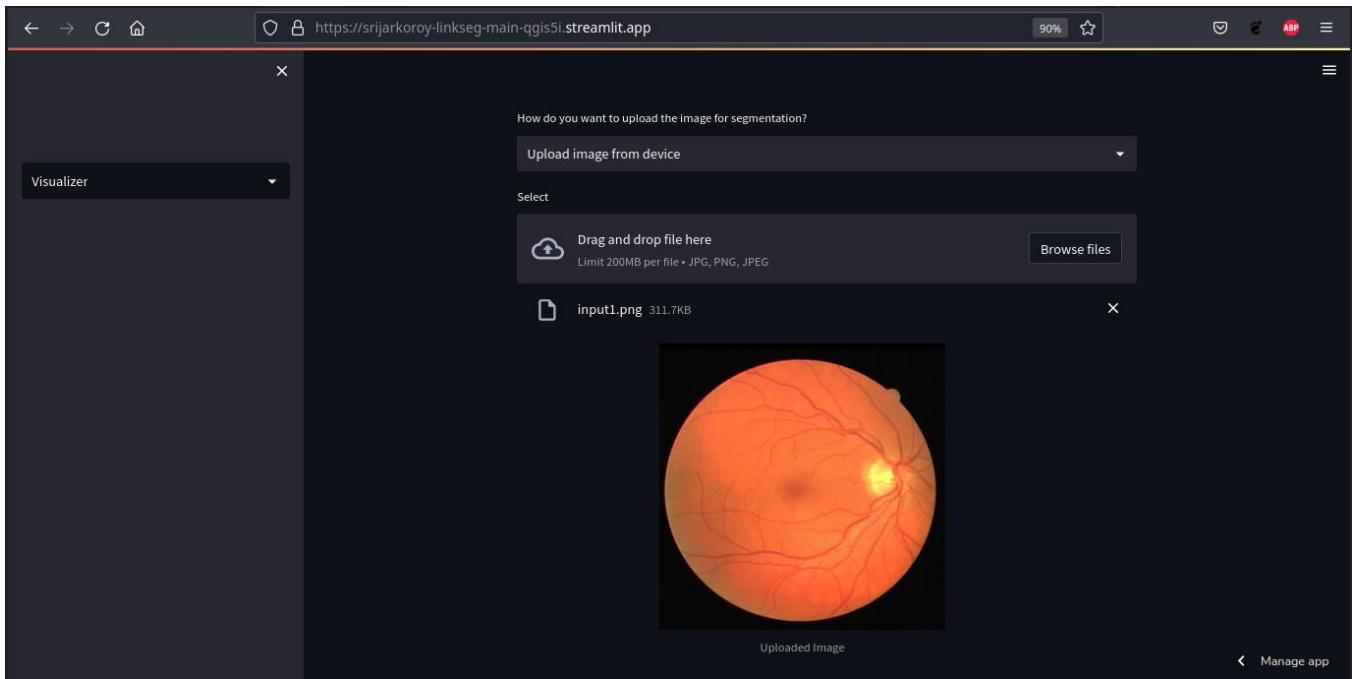


Fig 5.55: Image uploaded.

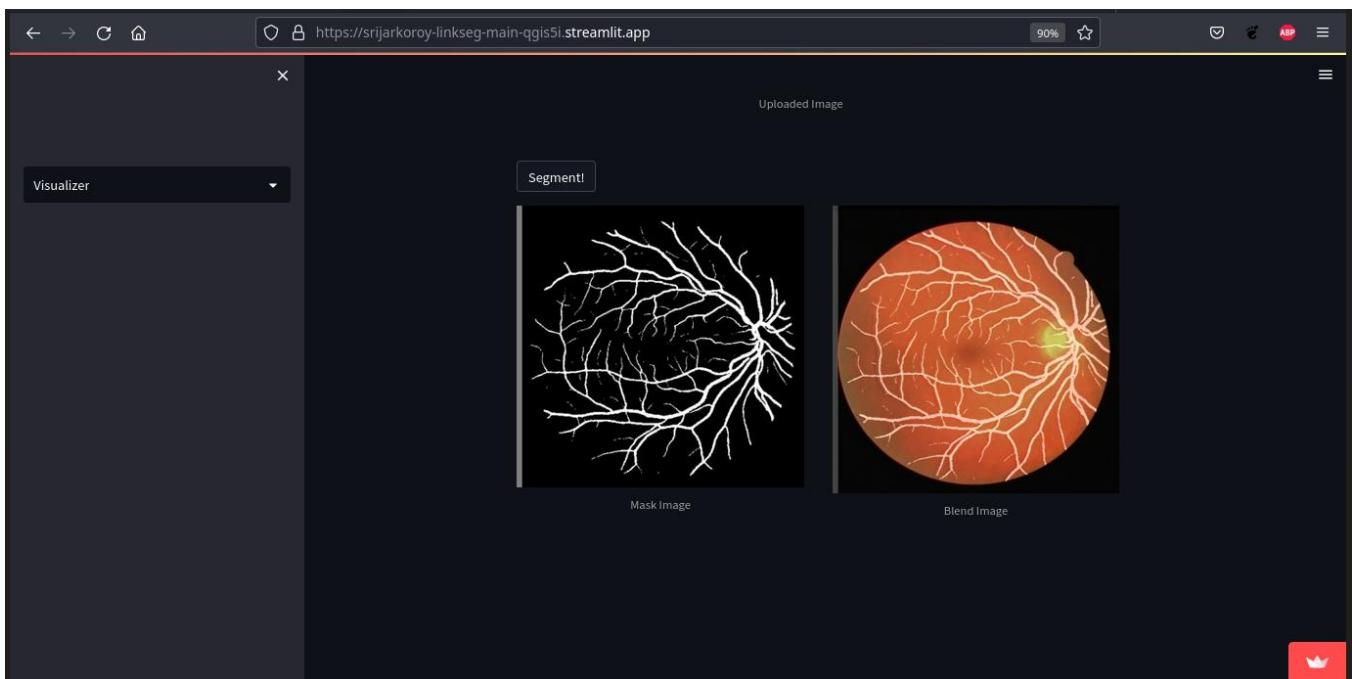


Fig 5.56: Image segmented.

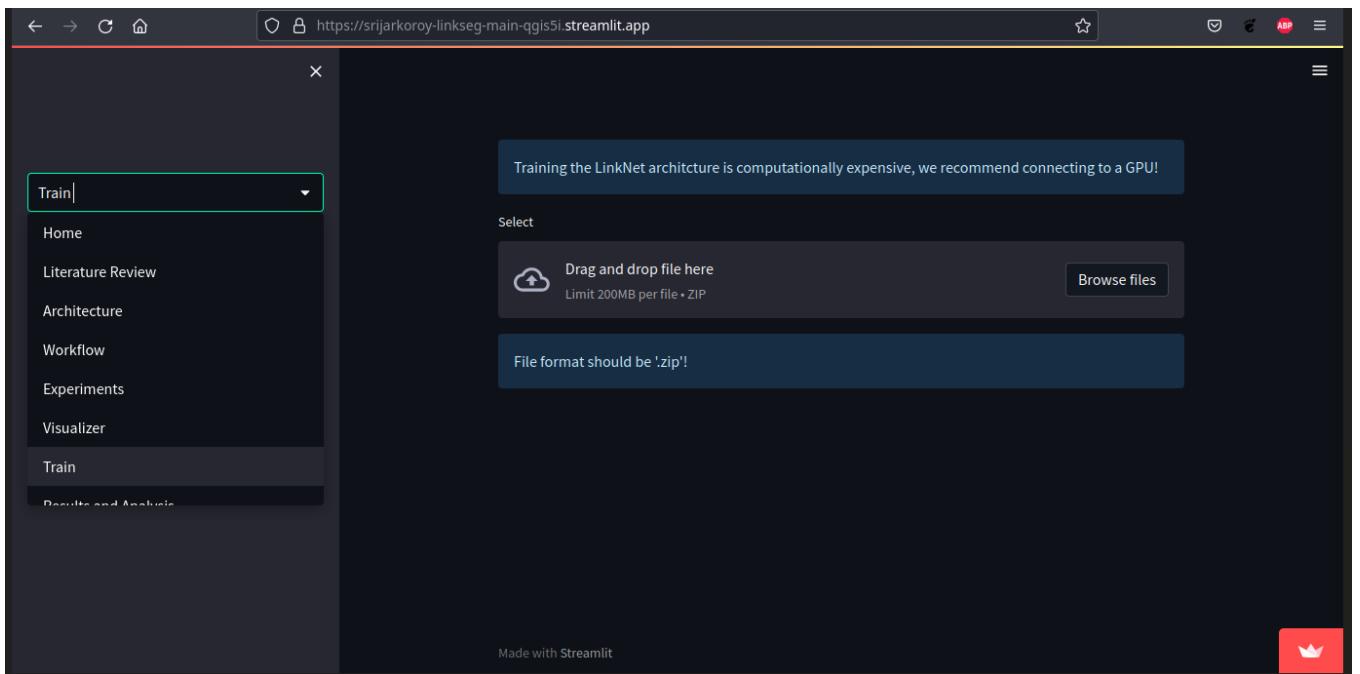


Fig 5.57: Training Module in UI

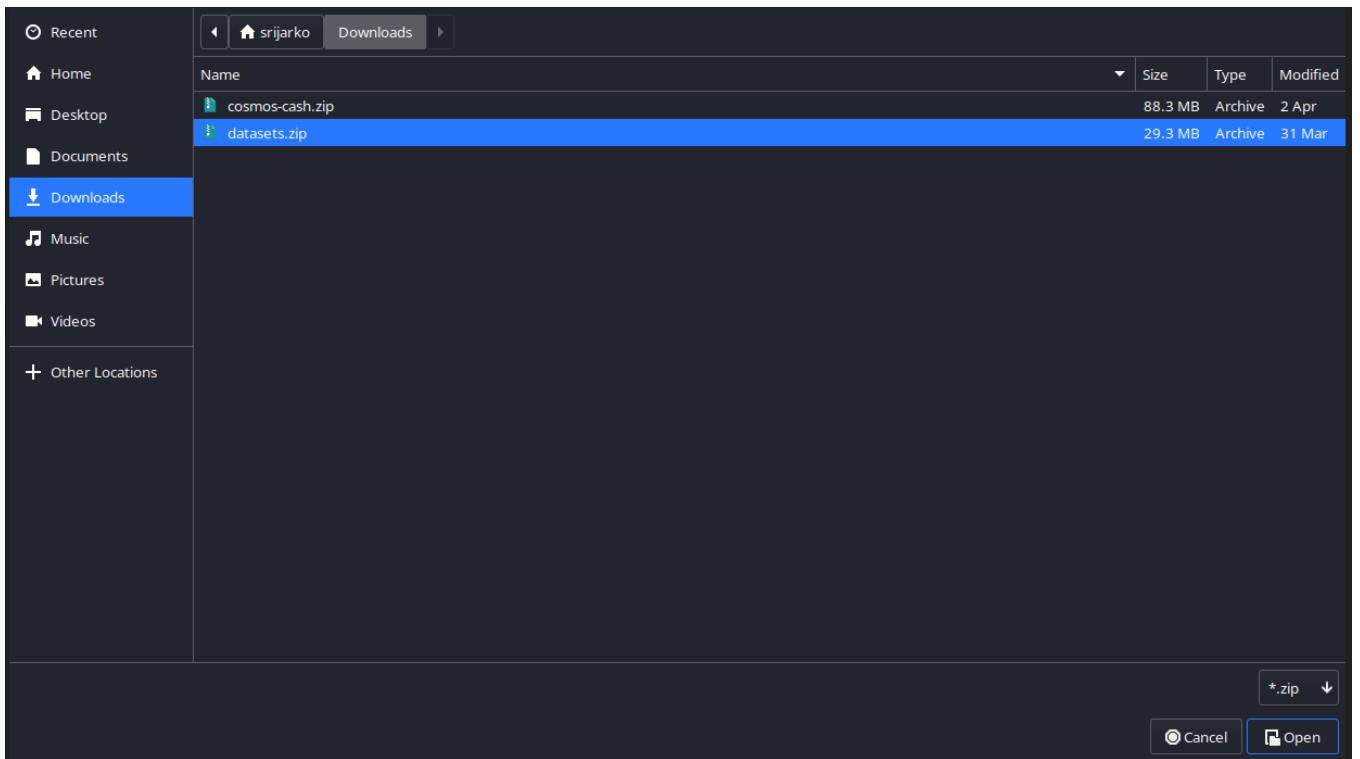


Fig 5.58: Dataset Upload

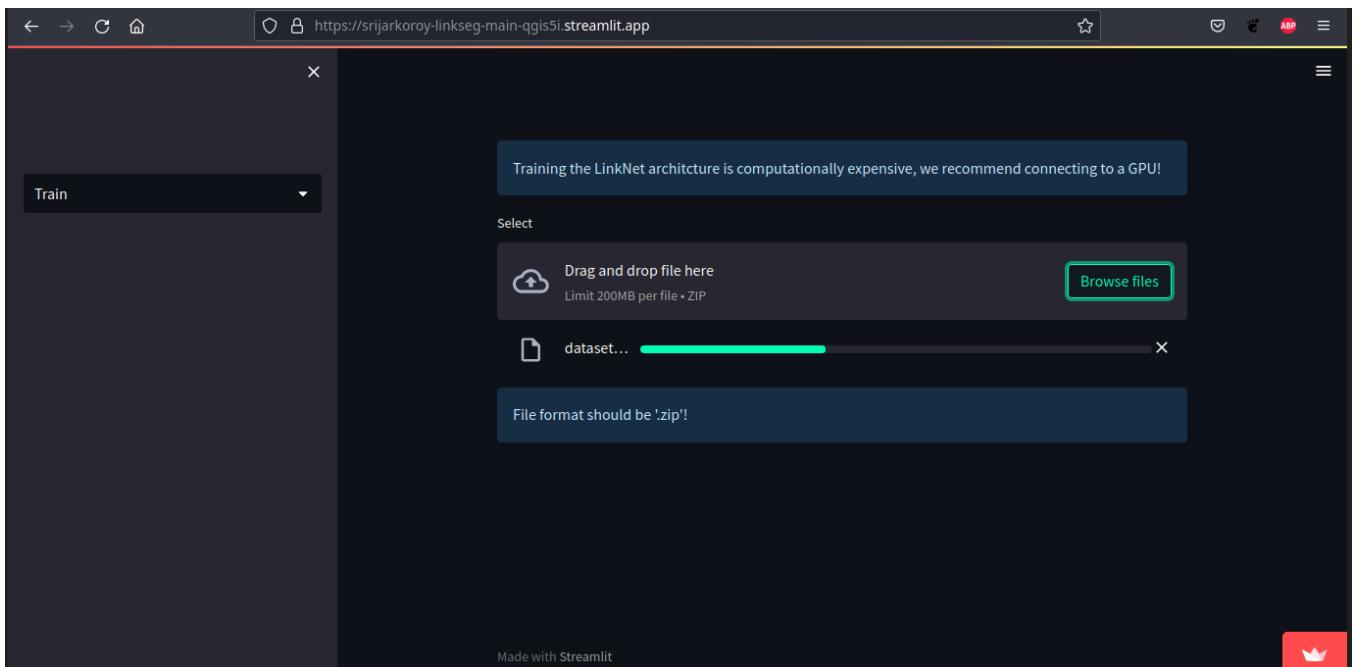


Fig 5.59: Dataset Uploading

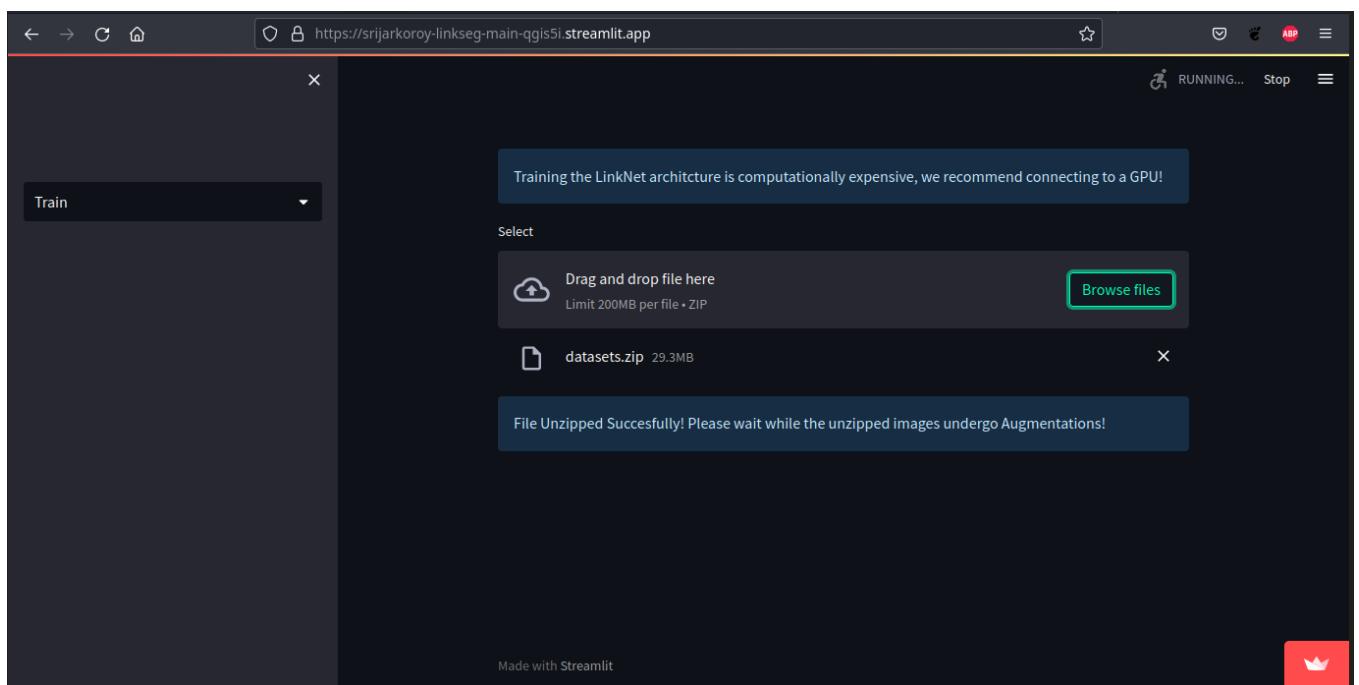


Fig 5.60: Dataset Uploaded

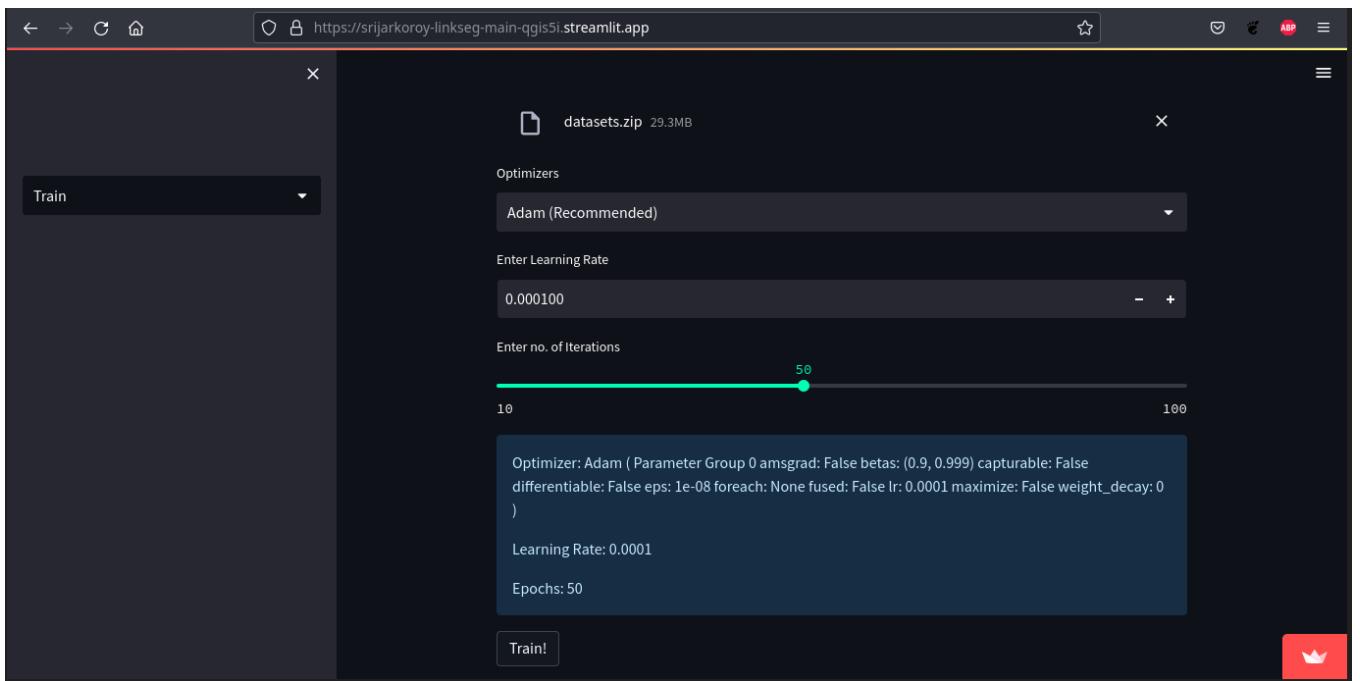


Fig 5.61: Training Hyperparametes

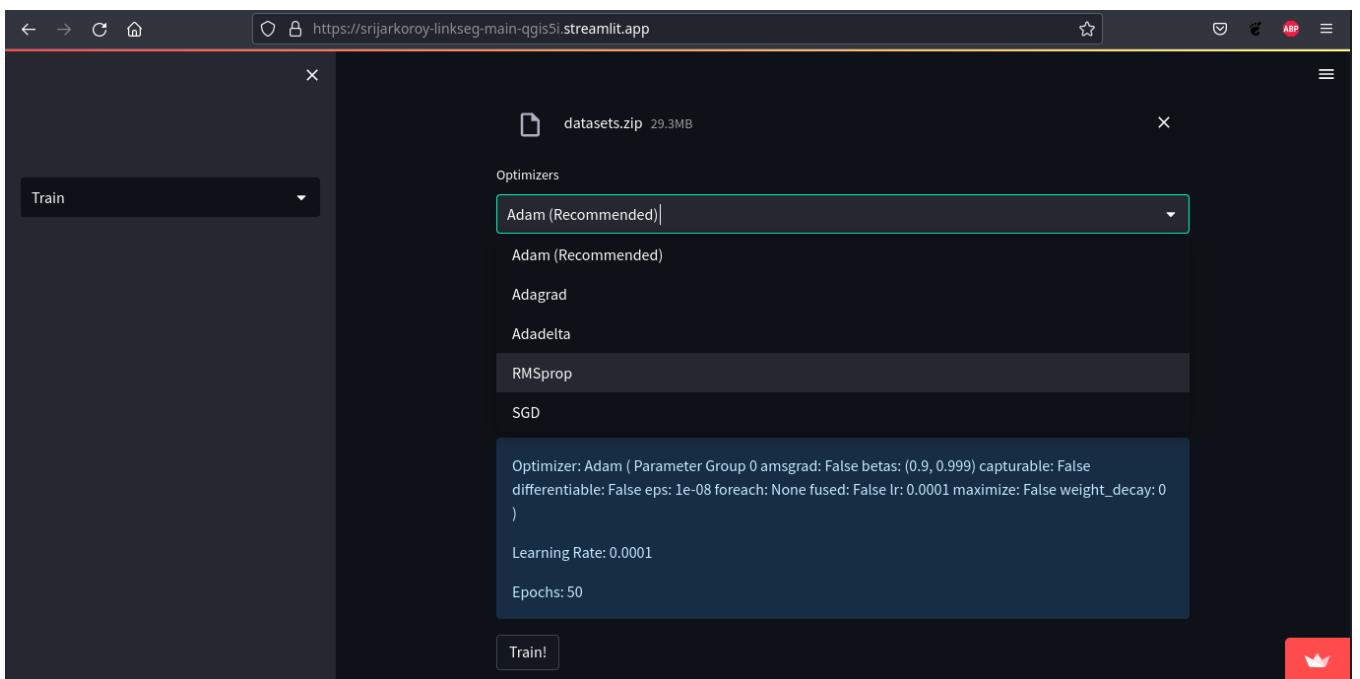


Fig 5.62: Optimizer

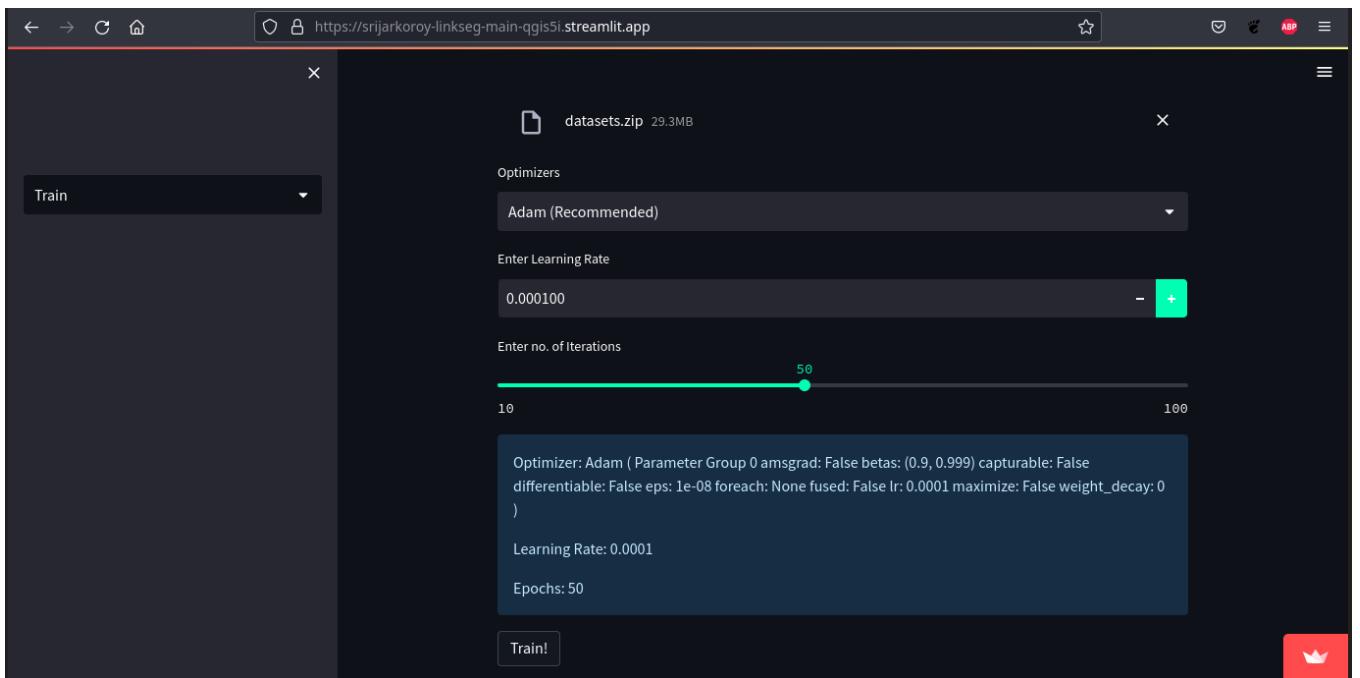


Fig 5.63: Learning Rate

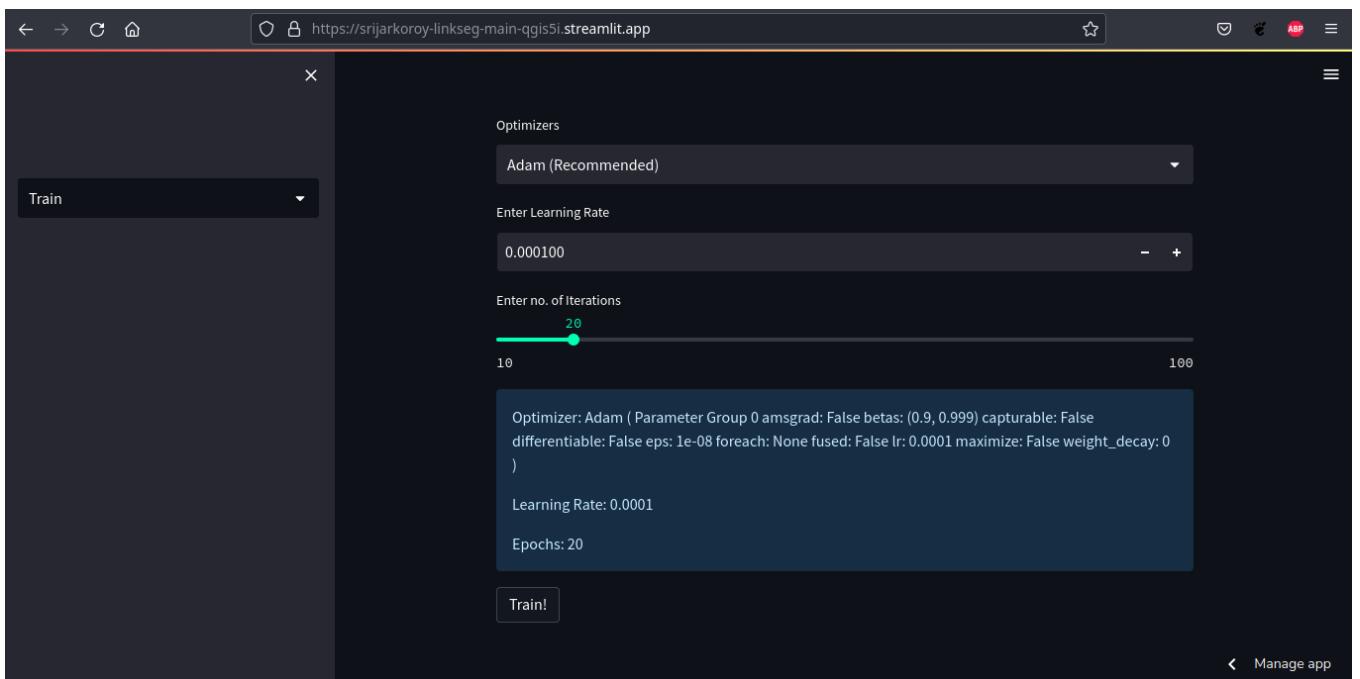


Fig 5.64: Epochs

The screenshot shows a Streamlit application window with a dark theme. The title bar displays the URL <https://srijarkoroy-linkseg-main-qgis5i.streamlit.app>. The main content area is titled "Results and Analysis". A sidebar on the left also has a "Results and Analysis" section. The main content includes a text block stating: "Validation was performed on the DRIVE Dataset and the STARE Dataset using NVIDIA Tesla T4 GPU. The weights were hosted and downloaded via gdown for further tests. The Figure on the next slide shows the test results of the input retinal images from DRIVE Dataset and validation results are mentioned in the Table below." Below this is a table comparing metrics:

Metrics	DRIVE	STARE
Dice Loss	0.1164	0.1277
IoU Loss	0.2086	0.1962

A red banner at the bottom right contains the text "A comparison of our architecture with state-of-the-art architectures has been shown in the Table below" and a small crown icon.

Fig 5.65: Result Analysis

The screenshot shows a Streamlit application window with a dark theme. The title bar displays the URL <https://srijarkoroy-linkseg-main-qgis5i.streamlit.app>. The main content area is titled "Results and Analysis". A sidebar on the left also has a "Results and Analysis" section. The main content includes a table comparing metrics:

Metrics	DRIVE	STARE
Dice Loss	0.1164	0.1277
IoU Loss	0.2086	0.1962

A red banner at the bottom right contains the text "A comparison of our architecture with state-of-the-art architectures has been shown in the Table below" and a small crown icon.

Below the table is another table comparing various architectures:

Architecture	DRIVE
UNETs	0.9790 (AUC ROC)
Lattice NN with Dendrite Processing	0.81 (F1 Score)
Multi-level CNN with Conditional Random Fields	0.9523 (Accuracy)
Multi-scale Line Detection	0.9326 (Accuracy)
CLAHE	0.9477 (Accuracy)
Modified SUSAN edge detector	0.9633 (Accuracy)
Improved LinkNet	0.8836 (Dice Score)

A red banner at the bottom right contains a small crown icon.

Fig 5.66: SOTA

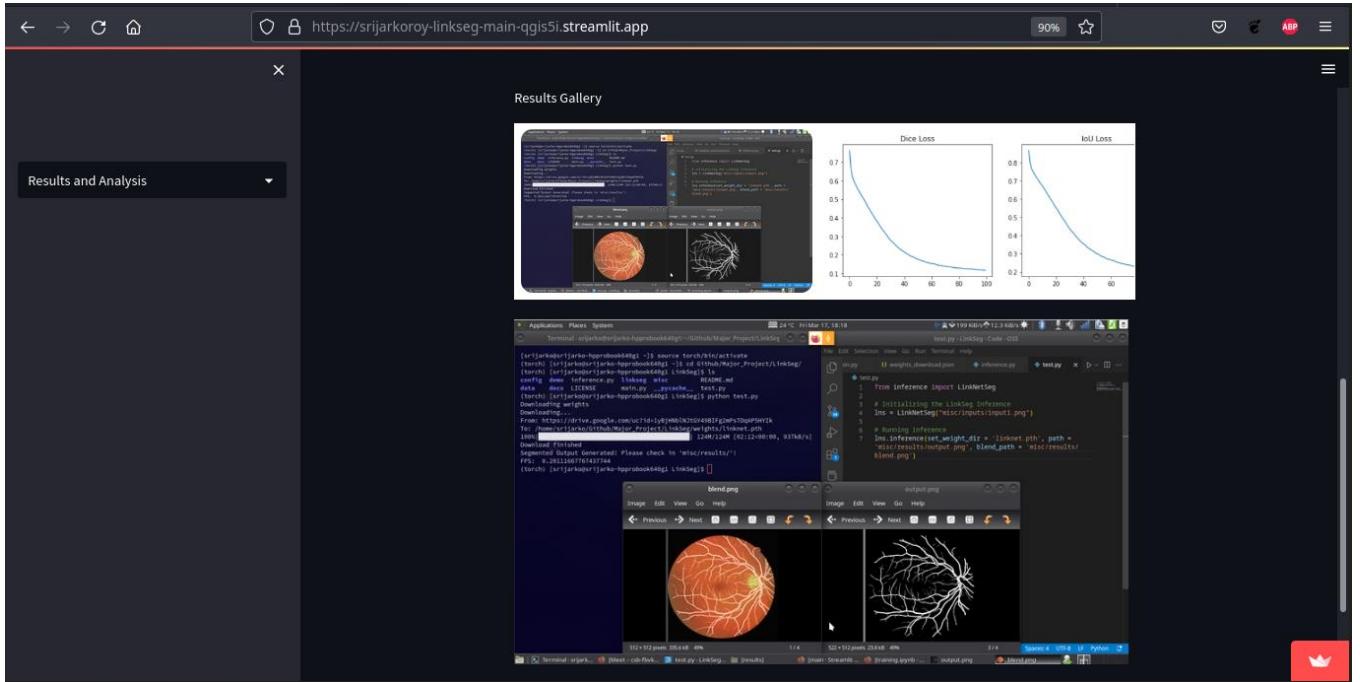


Fig 5.67: Gallery



Fig 5.68: Gallery – 2

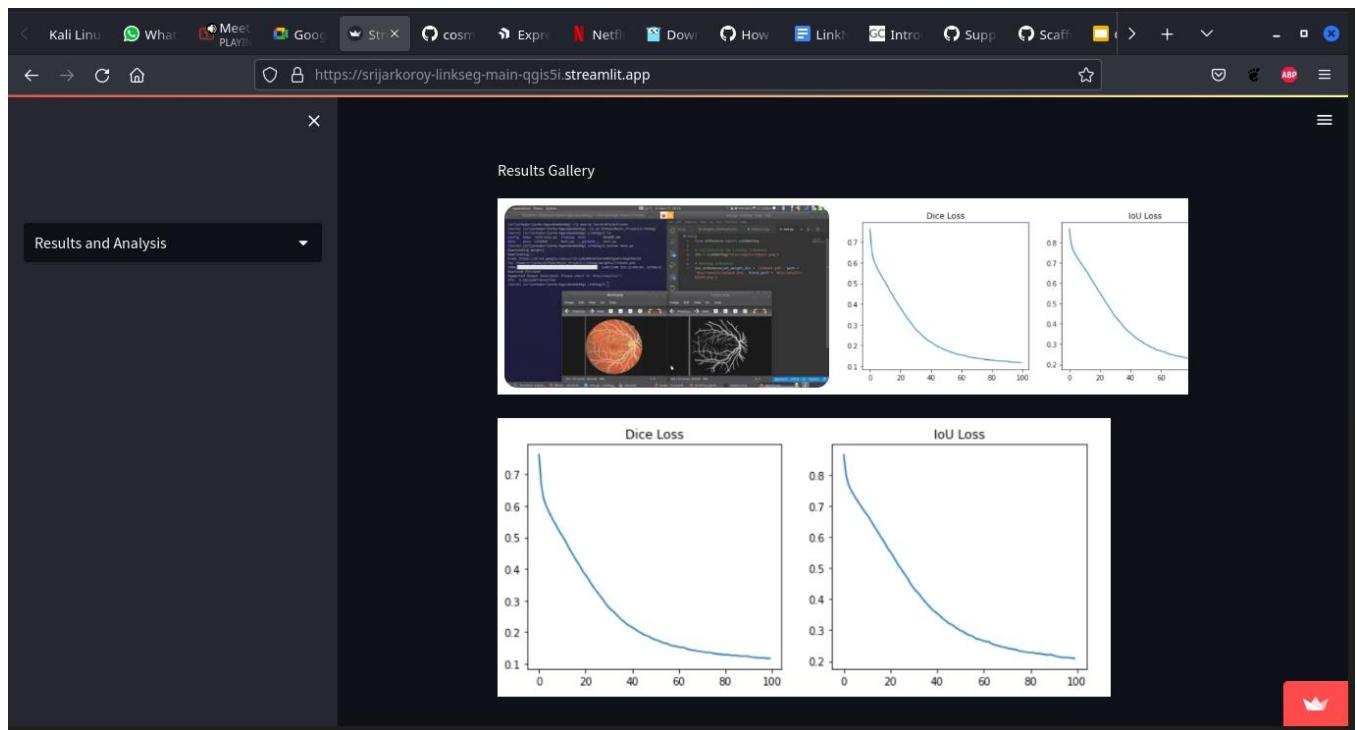


Fig 5.69: Gallery – 2

CHAPTER 6

RESULTS AND DISCUSSIONS

6.1 Test Results

Validation was performed on the DRIVE Dataset and the STARE Dataset using NVIDIA Tesla T4 GPU. The weights were hosted and downloaded via gdown for further tests. Figure 6.1 shows the test results of the input retinal images from DRIVE Dataset and validation results are mentioned in Table 6.1.

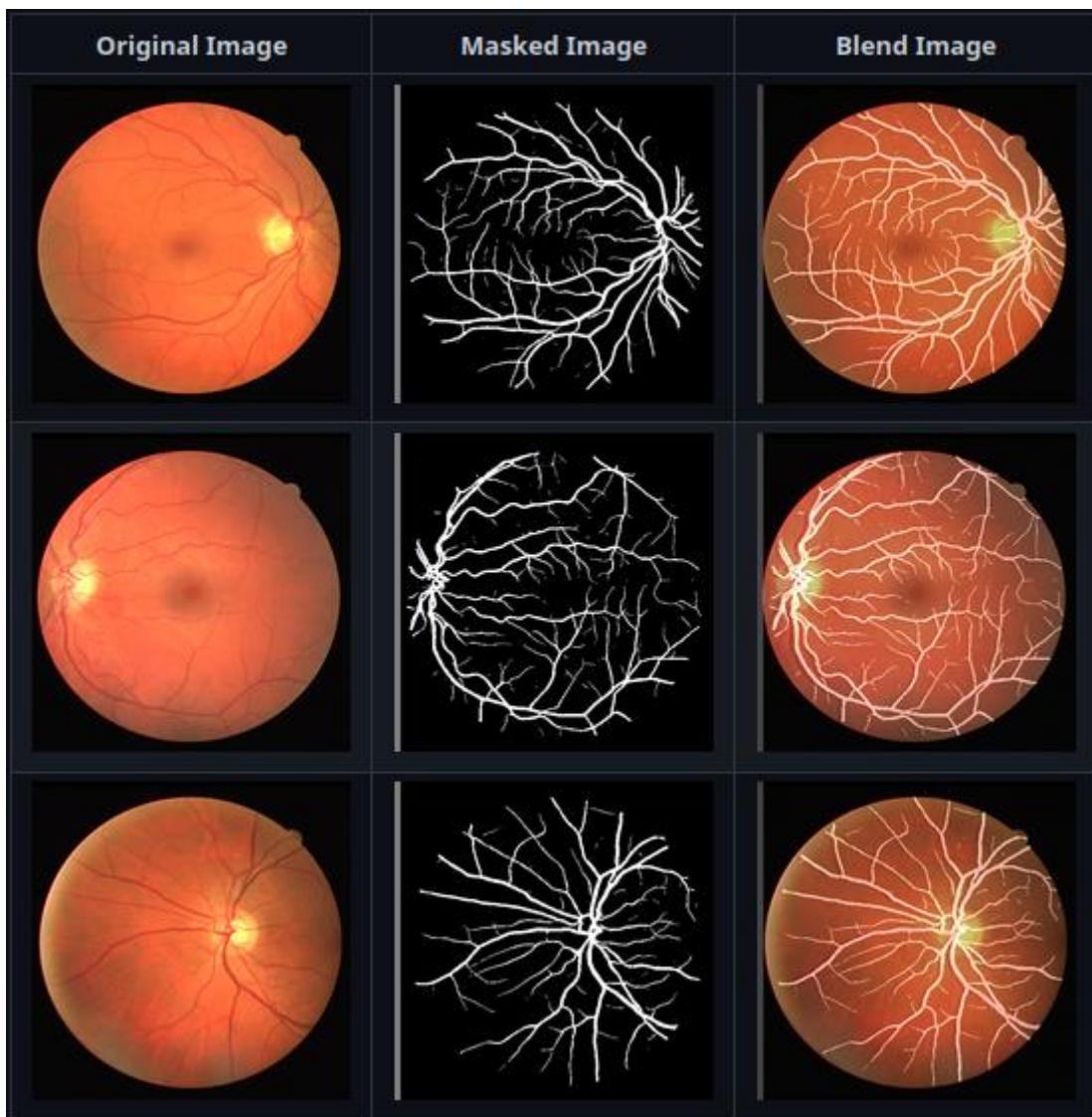


Fig 6.1: Testing Results

6.2 Graphs

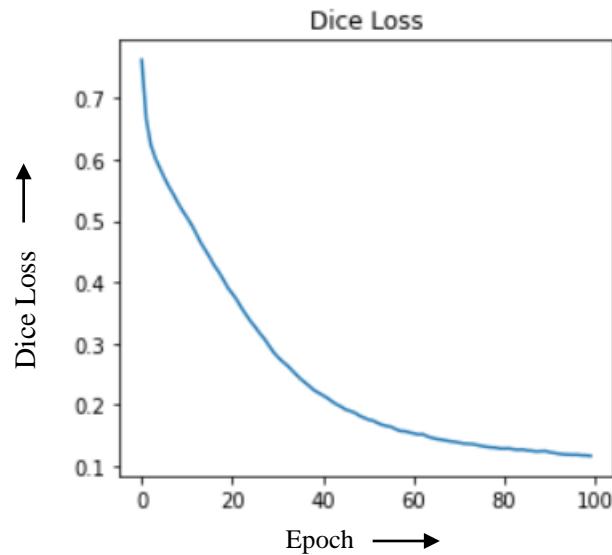


Fig 6.2: Dice Loss

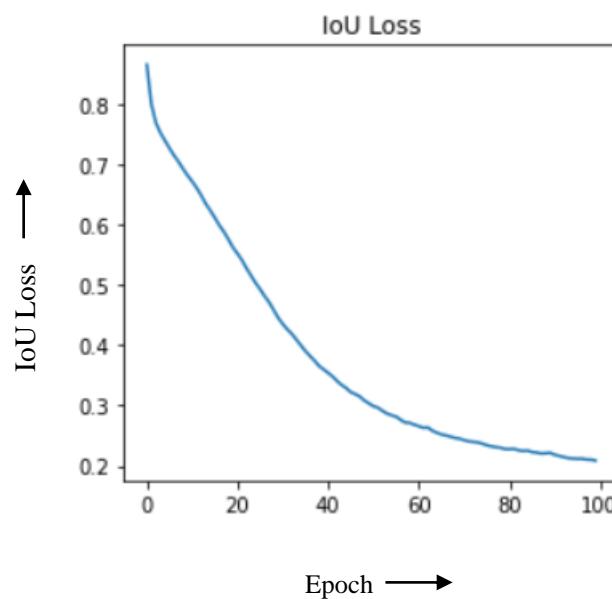


Fig 6.3: IoU Loss

6.3 Tabular Representation of Metrics

Table 6.3.1: Metrics

METRIC	DRIVE	STARE
Dice Loss	0.1164	0.1277
IoU Loss	0.2086	0.1962

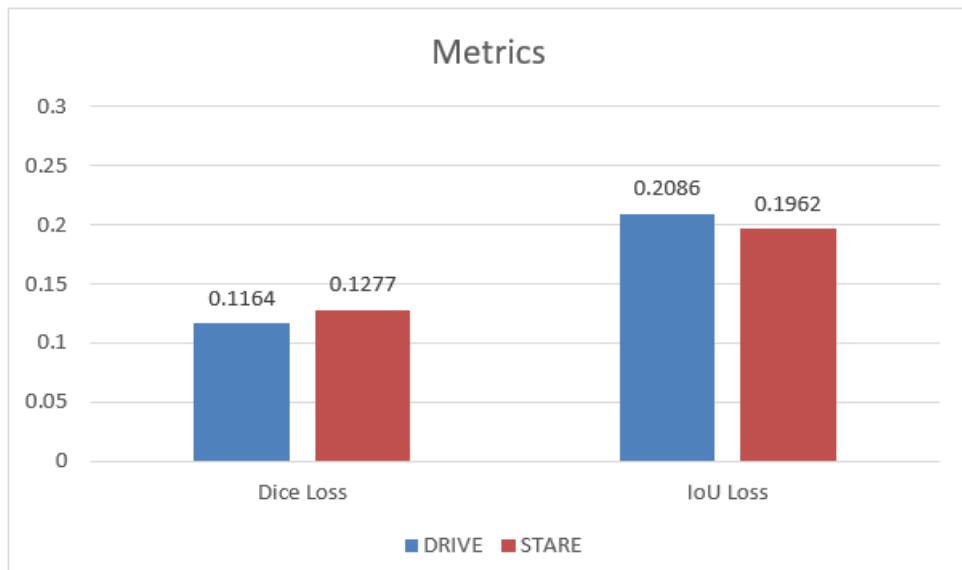


Fig. 6.3.1: Accuracy Comparison with SOTA

6.4 Comparison of LinkNet with state-of-the-art

Table 6.4.1: Comparison with SOTA Architecture

Architecture	Metric
UNETs	0.9790 (AUC ROC)
Lattice NN with Dendrite Processing	0.81 (F1 Score)
Multi-level CNN with Conditional Random Fields	0.9523 (Accuracy)
Multi-scale Line Detection	0.9326 (Accuracy)
CLAHE	0.9477 (Accuracy)
Modified SUSAN edge detector	0.9633 (Accuracy)
Improved LinkNet	0.8836 (Dice Score)

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

Effective segmentation of retinal image can be extensively used in medical imaging, particularly in ophthalmology, for diagnosis, monitoring, and treatment of retinal diseases. In our research, we have tried to improve the segmentation accuracy by improving upon LinkNet, which will help in better diagnosis of various eye ailments. This work can be further improved by collection of a binary classification dataset with classes healthy and infected, and a binary classification model that would classify the output segmented mask into one of the aforementioned classes.

Accurate segmentation can help doctors identify and monitor the progression of retinal diseases such as diabetic retinopathy, glaucoma, and age-related macular degeneration. Moreover, segmentation is a critical step in the development of automated screening systems that can assist ophthalmologists in early detection of such diseases.

In recent years, several segmentation methods based on deep learning have been proposed, including UNETs, CNNs, and SVMs. However, these techniques suffer from the problem of losing crucial spatial information during the feature extraction stage as a result of multiple downsampling. This loss of information can result in inaccurate segmentation, making it challenging for ophthalmologists to diagnose and treat retinal diseases.

In this context, our research aims to improve the accuracy of retinal image segmentation by building upon the existing LinkNet architecture. LinkNet is a light-weight segmentation model that has shown promising results in various medical imaging tasks, including retinal vessel segmentation. Our proposed improvements include modifications to the encoder and decoder blocks of LinkNet, which will help retain important spatial information and enhance the overall accuracy of the model.

However, improving segmentation accuracy is not the only challenge in medical imaging. There is also a need to classify the segmented regions into various classes, such as healthy and infected, to assist in the diagnosis and treatment of various eye conditions. In this regard, we propose to collect a binary classification dataset that includes healthy and infected retinal images and train a binary classification model to classify the output segmented mask into one of the two classes.

In conclusion, improving the accuracy of retinal image segmentation can have a significant impact on the diagnosis and treatment of various eye diseases. Our proposed improvements to LinkNet, coupled with a binary classification model, can provide a reliable and efficient automated screening system that can assist ophthalmologists in the early detection and diagnosis of retinal diseases.

REFERENCES

- [1] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI

2015. Lecture Notes in Computer Science(), vol 9351. Springer, Cham.
- [2] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. *Insights Imaging* 9, 611–629(2018).
- [3] Zhang, Y. (2012). Support Vector Machine Classification Algorithm and Its Application. In: Liu, C., Wang, L., Yang, A. (eds) Information Computing and Applications. ICICA 2012. Communications in Computer and Information Science, vol 308. Springer, Berlin, Heidelberg.
- [4] Gao, Hongyang & Yuan, Hao & Wang, Zhengyang & Ji, Shuiwang. (2019). Pixel Transposed Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. PP. 1-1. 10.1109/TPAMI.2019.2893965.
- [5] A. Chaurasia and E. Culurciello, “Linknet: Exploiting encoder representations for efficient semantic segmentation,” pp. 1–4, 2017.
- [6] A. E. Orhan and X. Pitkow, “Skip connections eliminate singularities,” 2017.
- [7] <https://drive.grand-challenge.org/DRIVE/>
- [8] J. Staal, M. D. Abramoff, M. Niemeijer, M. A. Viergever and B. van Ginneken, "Ridge-based vessel segmentation in color images of the retina," in *IEEE Transactions on Medical Imaging*, vol. 23, no. 4, pp. 501-509, April 2004, doi: 10.1109/TMI.2004.825627.
- [9] A. M. Mendonca and A. Campilho, "Segmentation of retinal blood vessels by combining the detection of centerlines and morphological reconstruction," in *IEEE Transactions on Medical Imaging*, vol. 25, no. 9, pp. 1200-1213, Sept. 2006, doi: 10.1109/TMI.2006.879955.
- [10] I. Abdurrazaq, S. Hati and C. Eswaran, "Morphology approach for features extraction in retinal images for diabetic retinopathy diagnosis," 2008 International Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia, 2008, pp. 1373-1377, doi: 10.1109/ICCCE.2008.4580830.
- [11] R. Ghaderi, H. Hassanpour and M. Shahiri, "Retinal vessel segmentation using the 2-D Morlet wavelet and neural network," 2007 International Conference on Intelligent and Advanced Systems, Kuala Lumpur, Malaysia, 2007, pp. 1251-1255, doi: 10.1109/ICIAS.2007.4658584.
- [12] Osareh, Alireza & Shadgar, Bita. (2009). Automatic blood vessel segmentation in color images of retina. *Iranian Journal of Science and Technology. Transaction B: Engineering*. 33.
- [13] <https://cecas.clemson.edu/~ahoover/stare/>
- [14] You, Xinge & Peng, Qinmu & Yuan, Yuan & Cheung, Yiu-ming & Lei, Jiajia. (2011). Segmentation of retinal blood vessels using the radial projection and semi-supervised

- approach. Pattern Recognition. 44. 2314-2324. 10.1016/j.patcog.2011.01.007.
- [15] Jiaqi Guo, Ziliang Si, Yu Wang, Qian Liu, Ming Fan, Jian-Guang Lou, Zijiang Yang, and Ting Liu. 2021. Chase: A Large-Scale and Pragmatic Chinese Dataset for Cross-Database Context-Dependent Text-to-SQL. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 2316–2331, Online. Association for Computational Linguistics.
- [16] M. M. Fraz et al., "An Ensemble Classification-Based Approach Applied to Retinal Blood Vessel Segmentation," in IEEE Transactions on Biomedical Engineering, vol. 59, no. 9, pp. 2538-2548, Sept. 2012, doi: 10.1109/TBME.2012.2205687.
- [17] Nguyen, Uyen & Bhuiyan, Alauddin & Park, Laurence & Ramamohanarao, Kotagiri. (2012). An effective retinal blood vessel segmentation method using multi-scale line detection. Pattern Recognition. 46. 703-. 10.1016/j.patcog.2012.08.009.
- [18] Yu Qian Zhao, Xiao Hong Wang, Xiao Fang Wang, Frank Y. Shih, Retinal vessels segmentation based on level set and region growing, Pattern Recognition, Volume 47, Issue 7, 2014, Pages 2437-2446, ISSN 0031-3203, <https://doi.org/10.1016/j.patcog.2014.01.006>.
- [19] Azzopardi, G., Strisciuglio, N., Vento, M., & Petkov, N. (2015). Trainable COSFIRE filters for vessel delineation with application to retinal images. Medical Image Analysis, 19(1), 46-57.
- [20] S. Roychowdhury, D. D. Koozekanani and K. K. Parhi, "Iterative Vessel Segmentation of Fundus Images," in IEEE Transactions on Biomedical Engineering, vol. 62, no. 7, pp. 1738-1749, July 2015, doi: 10.1109/TBME.2015.2403295.
- [21] Xie, S., & Tu, Z. (2015). Holistically-Nested Edge Detection. ArXiv. <https://doi.org/10.48550/arXiv.1504.06375>.
- [22] H. Fu, Y. Xu, S. Lin, D. W. K. Wong and J. Liu, "DeepVessel: Retinal vessel segmentation via deep learning and conditional random field", Proc. MICCAI, pp. 132-139, 2016.
- [23] Xiancheng, Wang, et al. "Retina blood vessel segmentation using a U-net based Convolutional neural network." Procedia Computer Science: International Conference on Data Science (ICDS 2018). 2018.
- [24] P. Xiuqin, Q. Zhang, H. Zhang and S. Li, "A Fundus Retinal Vessels Segmentation Scheme Based on the Improved Deep Learning U-Net Model," in IEEE Access, vol. 7, pp. 122634-122643, 2019, doi: 10.1109/ACCESS.2019.2935138.
- [25] Braovic, Maja & Stipanicev, Darko & Šerić, Ljiljana. (2018). Retinal blood vessel segmentation based on heuristic image analysis. Computer Science and Information Systems. 16. 14-14. 10.2298/CSIS180220014B.

- [26] Jin, Q., Chen, Q., Meng, Z. et al. Construction of Retinal Vessel Segmentation Models Based on Convolutional Neural Network. *Neural Process Lett* 52,1005–1022(2020).
- [27] P. Ravi, S. Roy, I. Dutta and K. Kottursamy, "Attention Mechanism, Linked Networks, and Pyramid Pooling Enabled 3D Biomedical Image Segmentation," 2022 IEEE/ACIS 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Taichung, Taiwan, 2022, pp. 91-96, doi: 10.1109/SNPD54884.2022.10051771.