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 ANDTECHNOLOGY
KATTANKULATHUR- 603 203

MAY 2023

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
DEPARTMENT OFCOMPUTATIONAL INTELLIGENCE

SIGNATURE

DR. R. ANNIE UTHRA

PROFESSOR & HEAD
DEPARTMENT OF COMPUTATIONAL INTELLIGENCE
SCHOOL OF COMPUTING
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR, CHENGALPATTU DISTRICT - 603203

SIGNATURE OF INTERNAL EXAMINER

SIGNATURE OF EXTERNAL EXAMINER

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 theUniversity 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.

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**, Asssitant 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]

ABSTRACT

The characteristics of Retinal Blood Vessels helps identify various eye ailments. The proper localization, extraction and segmentation of blood vessels are essential for the treatment of the eye. Manual segmentation of blood vessels may be error-prone and inaccurate, leading to difficulty in further treatment, and causing problems for both operators and ophthalmologists. We present a novel method of semantic segmentation of Retinal Blood Vessels using Linked Networks to account for spatial information that is lost during feature extraction. The implementation of the segmentation technique involves using Residual Networks as a feature extractor and Transpose Convolution and Upsample Blocks for image-to-image translation thereby giving a segmentation mask as an output. The use of Upsample Blocks arises from its ability to give noise-free output while 18 layered Residual Networks using skip connections are used in the feature extraction without the vanishing gradient issues. The main feature of this architecture is the links between the Feature Extractor and the Decoder networks that improve the performance of the network by helping in the recovery of lost spatial information. Training and Validation using the Pytorch framework have been performed on the Digital Retinal Images for Vessel Extraction (DRIVE) Dataset to establish quality results.

TABLE OF CONTENTS

Chapter No.	Title	Page No.

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolutional Neural Networks
ReLU	Rectified Linear Unit
SVM	Support Vector Machine
DRIVE	Digital Retinal Images for Vessel Extraction
IoU	Intersection over Union
ResNet	Residual Networks
CLAHE	Contrast-limited adaptive histogram equalization
RNN	Recurrent Neural Networks
GAN	Generative Adversarial Networks
CV	Computer Vision
SUSAN	Smallest Univalue Segment Assimilating Nucleus
AUC	Area Under Curve
ROC	Receiver Operating Characteristics
ARUNet	Attention Residual UNet
PSPNet	Pyramid Scene Parsing Network
FOV	Field of View
Adam	Adaptive Moment Estimation
SGD	Stochastic Gradient Descent

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 disease pf the eye that damages the optic nerve, causing blindness in human vision.
- Age-related macular degeneration: A condition where the macula, which is responsible for soothing and relaxing vision , begins to deteriorate, leading to loss of central vision.
- Diabetic retinopathy: It is an eye disease arousing from diabetes. This results in damaging the blood vessels in the retina, leading to loss of vision.
- Retinal detachment: A condition where the retina pulls away from its underlying tissue, resulting in vision loss.

A significant role is played by retinal image segmentation in the diagnosis, monitoring, and treatment of various such 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 caused to the blood vessels in the retina, hence allowing doctors to make informed decisions about treatment, such as laser therapy or surgery. Overall, retinal image segmentation is a crucial tool in the detection 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 serious eye ailments like glaucoma, diabetic retinopathy, and macular degeneration.

Pre-detection and diagnosis of these conditions is crucial for effective treatment and management of the diseases. With accurate retinal blood vessel segmentation of the 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 a significant role in the development and evaluation of new treatments for eye ailments. 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 domain of ophthalmology, and its importance in the diagnosis and treatment of various eye ailments cannot be overstated.

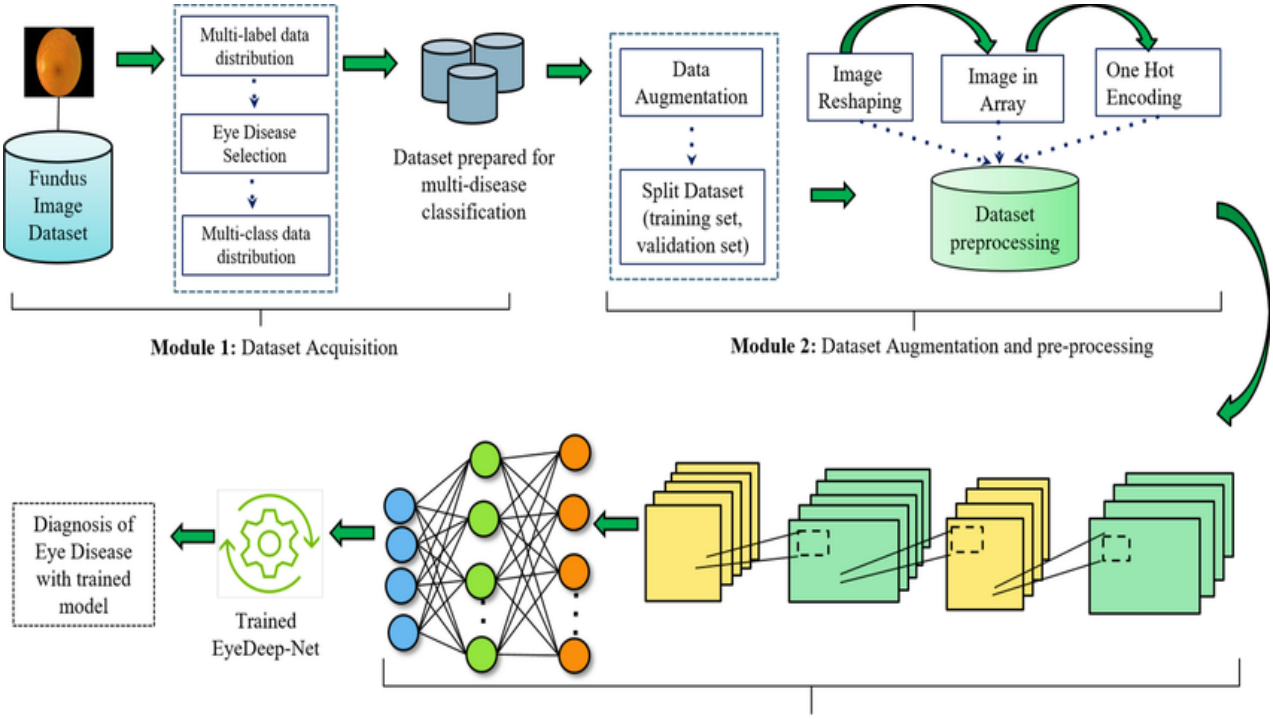


Figure 1.1: Training of a typical ML Model for feature extraction and classification

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 retinal blood vessel segmentation from fundus images has become increasingly popular in this field of medical diagnosis and 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 brought in significant implications for the 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 Blocks, 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 retinal blood vessel segmentation 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 retinal blood vessel segmentation 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 imaging tool that can get visible blood vessels present in the human body. Vascular segmentation of the retina is crucial for identifying many diseases, including diabetic retinopathy 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.

Automated retinal blood vessel segmentation from fundus images has gained favor in the field of medical imaging due to the flourishing of advanced CV (computer Vision) techniques such as deep learning-based methods. These techniques use large datasets to train the network to learn the features necessary for segmentation.

The automatic retinal blood vessel segmentation can provide accurate and reproducible results, enabling better diagnosis, treatment, and monitoring of various eye diseases.

In conclusion, this technique has several advantages over manual segmentation, including accuracy, consistency, and efficiency. It 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

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, natural language processing, deep learning 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.

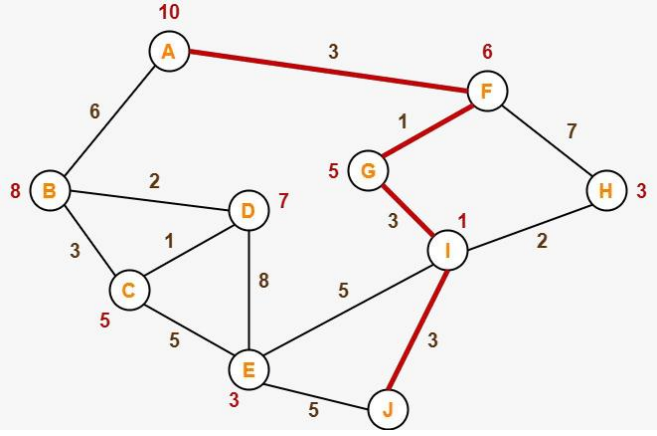


Figure 1.2: A* algorithm using AI

Machine learning (ML) 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 have extensive uses. 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 a few uses of machine learning. It has the ability to change a wide range of sectors, including healthcare, banking, and transportation.

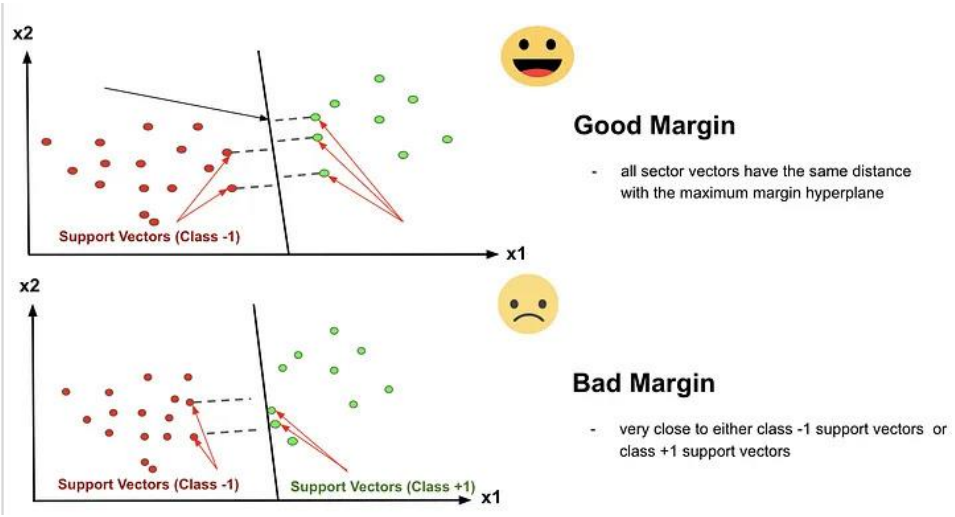


Figure 1.3: Support Vector Machines in ML

Deep learning (DL) 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) are used for image and video processing, recurrent neural networks (RNNs) are used for sequential data like speech and text, and generative adversarial networks (GANs) are used 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.

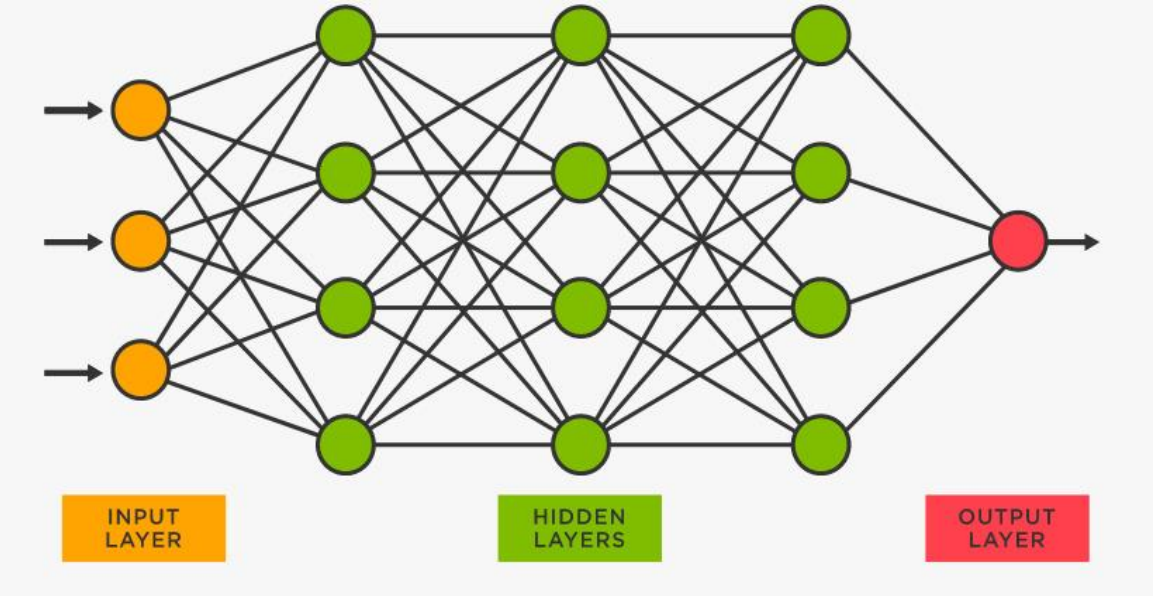


Figure 1.4: A typical 5-layered Neural Network

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.

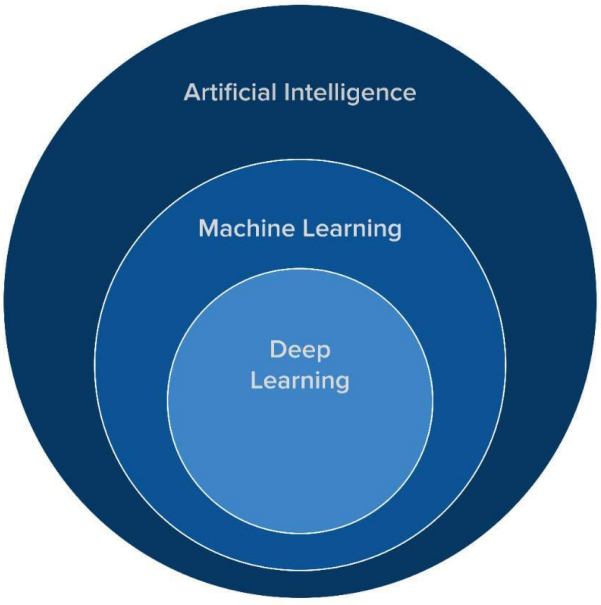


Figure 1.5: Relation between AI, ML and DL

The goal of the AI discipline of **computer vision** is to make the computers able 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, machine learning, pattern recognition 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 categorizing 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 revolutionize a variety of facets of daily life, including robots, autonomous cars, and medical diagnosis and care.

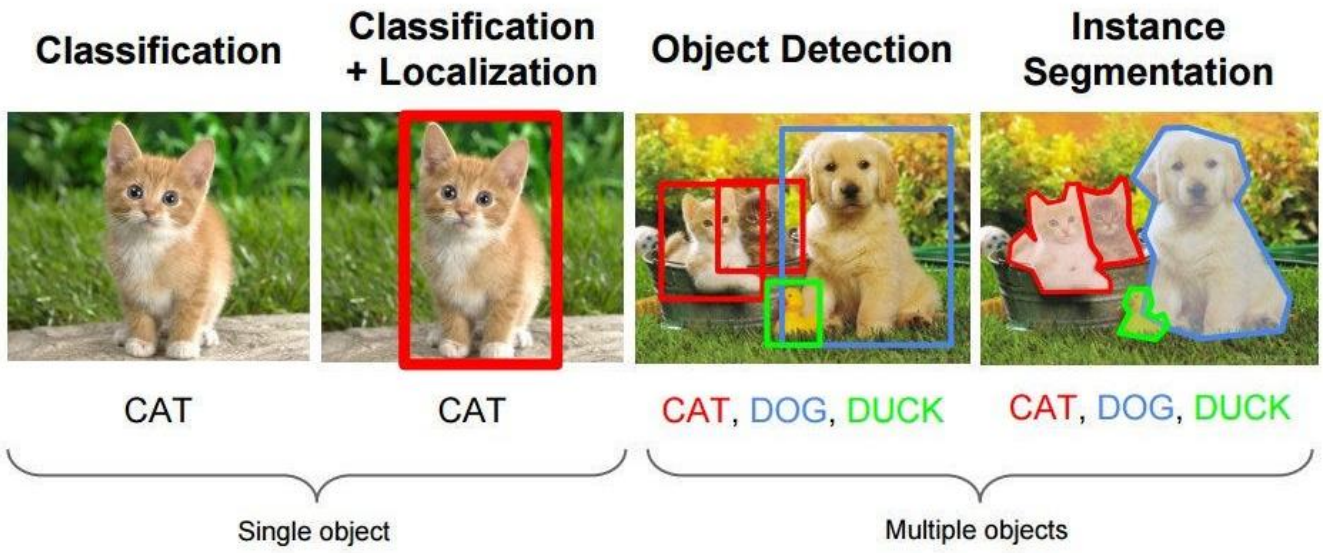


Figure 1.6: Computer Vision Tasks

Segmentation in CV is the act 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 categorizing 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 assigns a class name to each pixel of an image, such as a person, flower, or automobile. It considers several objects of the same class to be one entity.

Instance segmentation, on the other hand, considers numerous objects of the same class as different 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 considers each of the three sheep as different entities and gives them different colored masks.

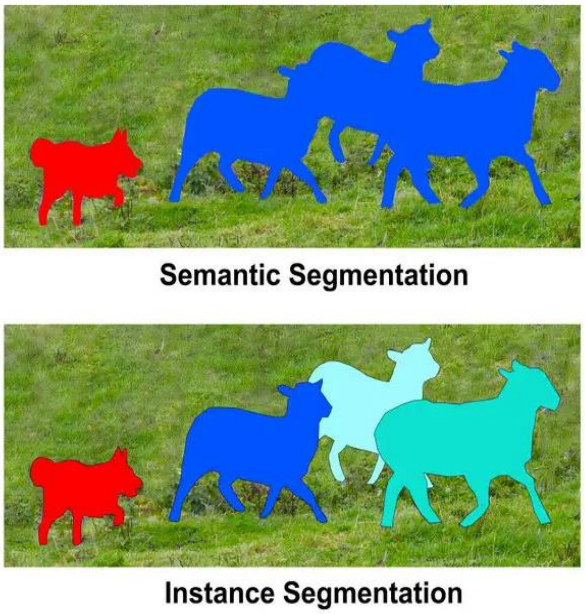


Figure 1.7: Semantic vs Instance Segmentation

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.

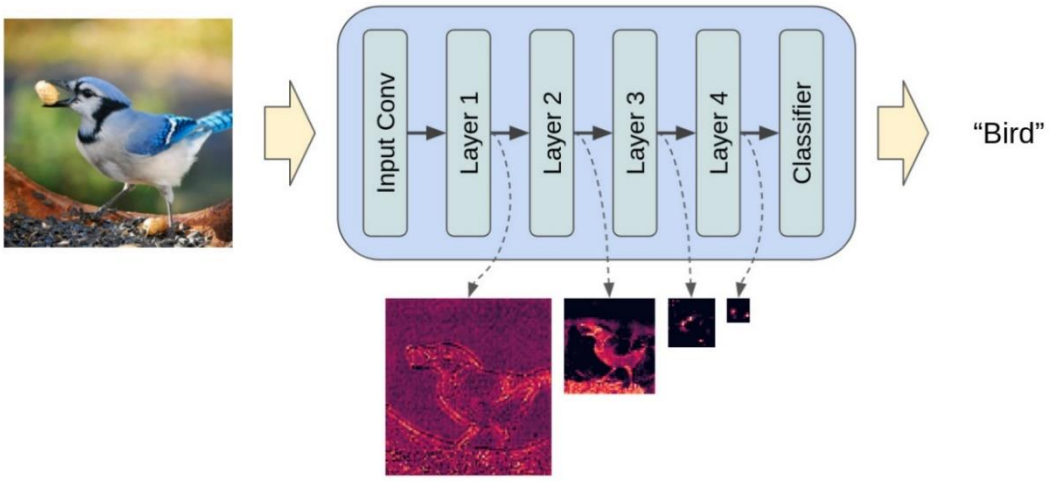


Figure 1.8: Feature Extraction

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 pixel in the expanded picture by computing an average of the weights of the four closest pixels present 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, that transforms a low-resolution image into a high-resolution one, and object detection, which uses upsampling to expand the size of feature maps in a CNN to enhance the recognition of objects.

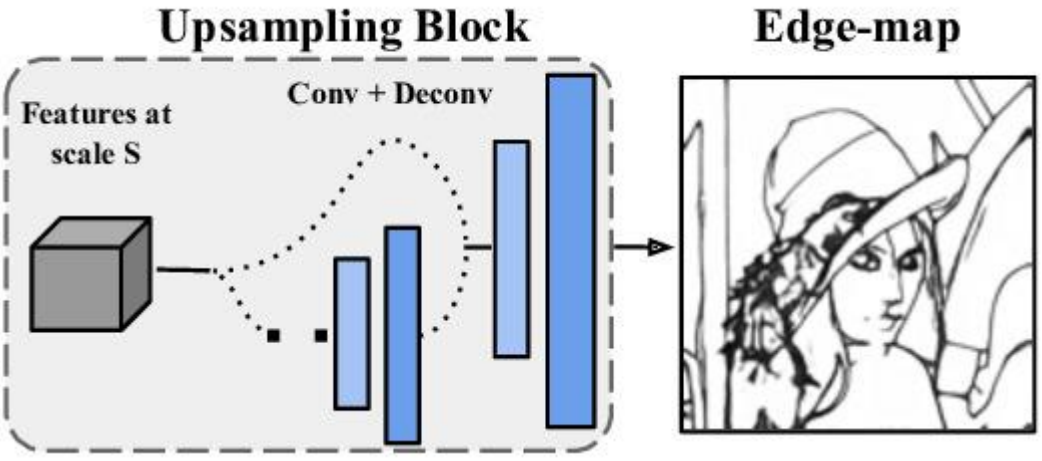


Figure 1.9: Upsample Operation

Skip connections are 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.

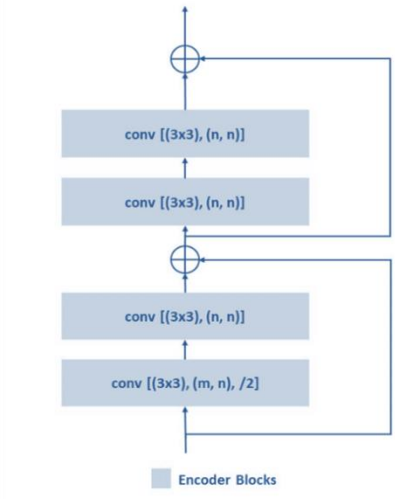


Figure 1.10: Skip Connections

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 Retinal Blood Vessel segmentation. The network makes use of skip connections [6] to link each and every Encoder Block to its corresponding Decoder Blocks, 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, [7] [13] and [15] 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 [7] and [13] 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 top hat 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 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 [7] and [13] 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.

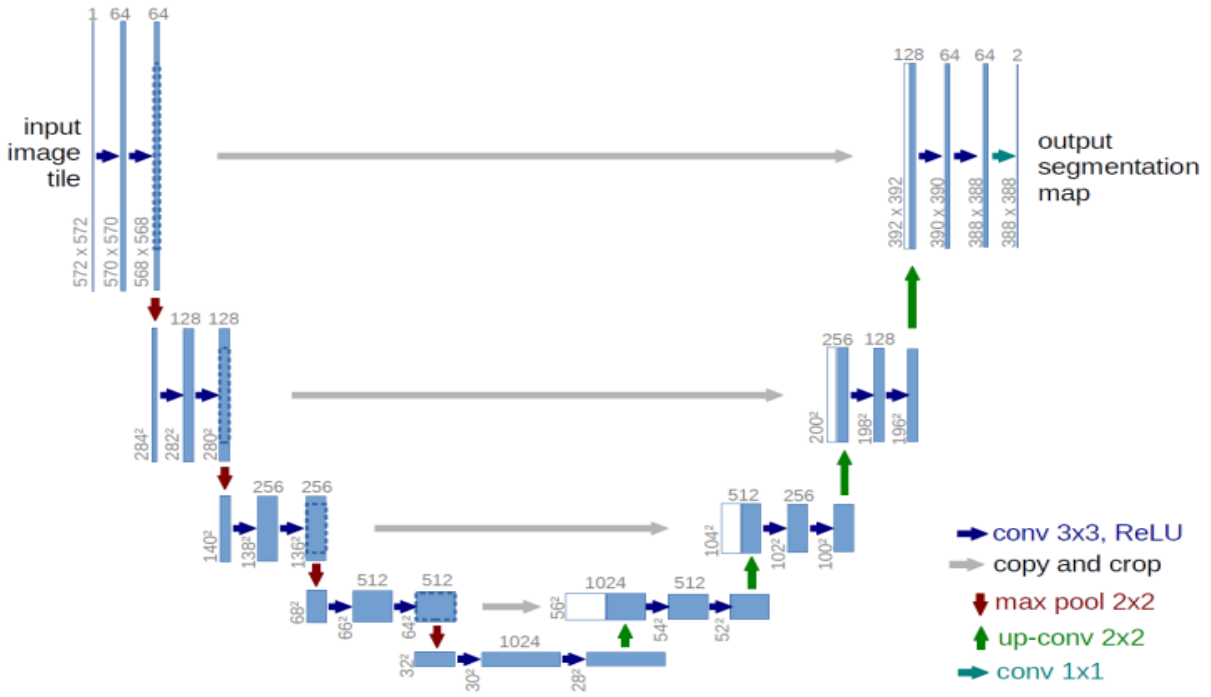


Figure 2.1: U-Net Architecture

By increasing segmentation accuracy and preserving this spatial information, it will be simpler to identify various eye conditions. Transpose Convolutions [4] are used by some networks, particularly UNETs, to accomplish upsampling. It is usual practise to use Transpose Convolution, which produces noisy outputs, for proper handling and to reduce operator fatigue.

The loss of essential spatial information during consecutive downsampling, the feature extraction phase, is therefore a significant problem with existing methods for retinal blood vessel segmentation. The accuracy of the existing models is thus worse than it may be if the spatial information had been preserved.

Some networks, such as UNETs, upsample using transpose convolutions. Noisey outputs are typically produced when Transpose Convolution is used.

Table 2.1: Literature Survey

Paper	Author(s)	Dataset	Method	Metrics
[1]	Ronneberger, O., Fischer et. al.	ImageNet	Data Augmentation	77.5% (Accuracy)
[2]	Yamashita et. al.	STARE Database	Lattice Neural Networks with Dendritic Processing	0.81 (F1 Score)
[3]	Zhang, Y.	Kaggle	SVM	78% (Accuracy)
[4]	Gao et. al.	PASCAL 2012 segmentation dataset, e MSCOCO 2015 detection dataset	pixel transposed convolutional layer (PixelTCL)	73%(Accuracy)
[5]	A. Chaurasia and E. Culurciello	Cityscapes [20] and CamVid [21]	LinkNet without bypass	72.6%(Accuracy)
[6]	A. E. Orhan and X. Pitkow	CIFAR-100	BiasReg	70%(Accuracy)
[8]	J. Staal, M. D. et. al.	DIARETDB1, DRIVE, and STARE	Markov random field (MRF)	68.57%(Accuracy)
[9]	Mendonca et. al.	DRIVE Dataset	Morphological Processing	0.9452 (Accuracy)
[10]	Ibrahim Abdurrazaq et. al.	DRIVE	Mathematical morphology theory and intensity transformation algorithm	TPF value of 0.8214 (True Positive Fraction)
		DRIVE	Morlet transform	0.90% to

[11]	R.Ghaderi et. al.			0.9668% (ROC curve)
[12]	A. OSAREH and B. SHADGAR	Custom Dataset	Octave UNet	95.24% (Accuracy)
[14]	Xinge You et. al.	DRIVE and STARE	Vessel segmentation	0.9434 on DRIVE And 0.9497 on STARE
[15]	Jiaqi Guo et. al.	CHASE	Intent recommendation	40.4% (Question Match)
[16]	Fraz et. al.	DRIVE, STARE, CHASE	Gabor Filter	0.9480, 0.9534, 0.9469 respectively (Accuracy)
[17]	Nguyen et. al.	STARE Dataset	Multi-Scale Line Detection	0.9326 (Accuracy)
[18]	Yu Qian Zhao et. al.	DRIVE and STARE databases	CLAHE, 2D Gabor wavelet	94.77% on the DRIVE database and 95.09% on the STARE database (Accuracy)
[19]	G Azzopardi et. al.	DRIVE	B-COSFIRE fD Gabor waveletilter	0.9442 (Accuracy)
[20]	Roychowdhury et. al.	DRIVE	Unsupervised Iterative Segmentation with tophat reconstruction and global thresholding	0.9494 (Accuracy)

[21]	Saining Xie, Zhuowen Tu	DRIVE	HED	0.9435 (Accuracy)
[22]	Huazhu Fu et. al.	DRIVE, STARE and CHASE_DB1	Multi-level CNN with Conditional Random Fields	0.9523 (Accuracy)
[23]	Wang Xiancheng et. al.	DRIVE Dataset	U-Net with specific data augmentations	0.9790 (AUC ROC)
[24]	Xiuqin Pan et. al.	DRIVE Dataset	Improved U-Net	96.50% (Segmentation Accuracy)
[25]	Maja Braovic et. al.	DRIVE and STARE Dataset	Modified SUSAN edge detector and shape analysis	96.33% and 96.10% (Accuracy) respectively
[26]	Qiangguo Jin et. al.	DRIVE and STARE	Deformable-ConvNet	Accuracy of 0.9628/0.9690
[27]	Srijarko Roy et. al.	BRATS2020	ARUnet, LinkNet3D, PSPNet3D	ARUNet: 0.8500 LinkNet3D: 0.8041 PSPNet3D: 0.8087

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Model Architecture

In its Feature Extractor, the LinkNet design, as shown in Figure 3.1, employs Residual Blocks based on the ResNet18 architecture. Strided convolutions were employed in the encoder to facilitate the linking procedure.

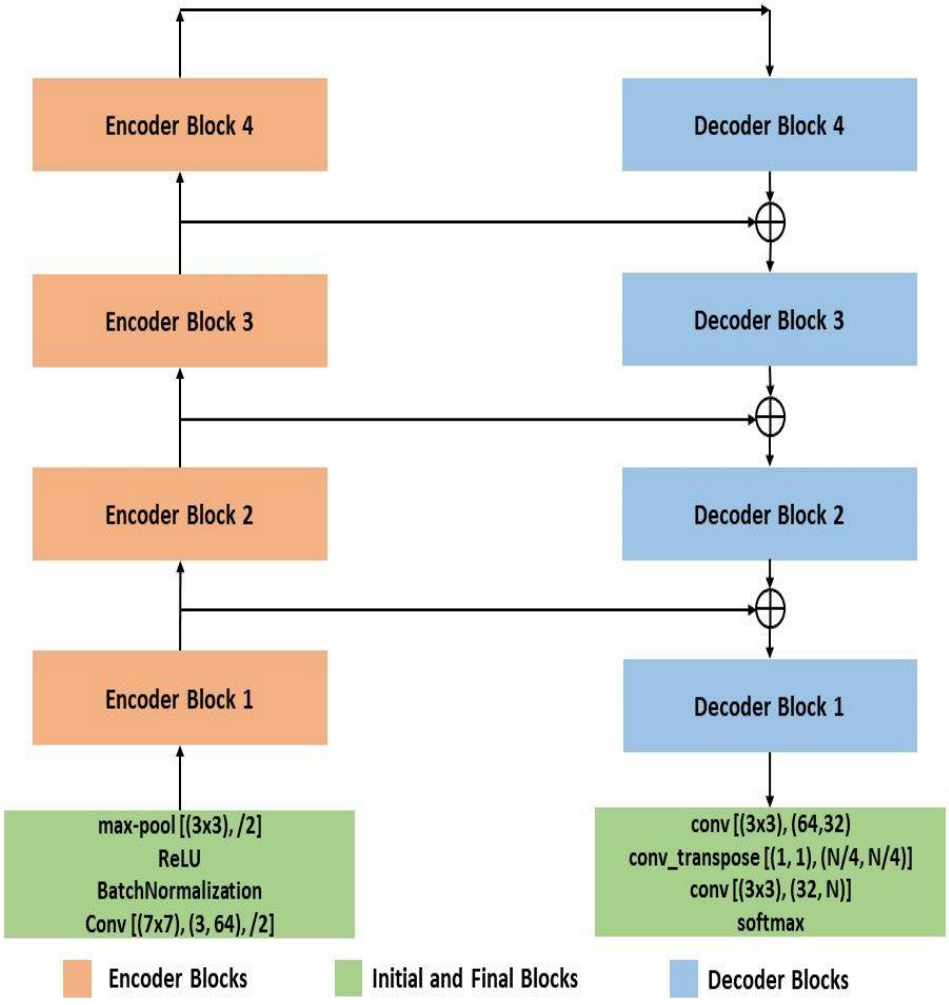


Figure 3.1: LinkNet Architecture

3.1.1 MODULE DESCRIPTIONS

LinkNet is a deep learning architecture proposed by Chaurasia and Culurciello [5] in 2017 for semantic segmentation in computer vision tasks. It is built on an encoder-decoder architecture, with the encoder consisting of a sequence of convolutional and max-pooling layers that lower the dimensions of the picture provided a input while extracting increasingly abstract characteristics. To restore the original spatial dimensions of the input picture and optimize the segmentation output, the decoder employs upsampling and skip connections.

The inclusion of residual connections and a "bottleneck" layer minimises the amount of parameters and computation required by the decoder while boosting its capacity to learn more discriminative features, which is the major novelty of LinkNet. As a consequence, the segmentation model becomes more efficient and precise.

3.1.1.1 INITIAL BLOCK

The Initial Block performs ‘conv’ on the input, with a (7x7) filter, and stride of 2. Batch Normalization and ReLU activation function follows to introduce non-linearity, and a (3x3) kernel spatial max-pooling.

3.1.1.2 FEATURE EXTRACTOR (RESIDUAL BLOCKS)

The Residual Blocks for the primary feature extraction follow the Initial Block of the Feature Extractor. The remaining blocks that follow the Initial Block are utilised to extract features. Each block performs three convolutional operations, beginning with a strided convolution and ending with a (3x3) filter and skip-connections. They are known as Encoder Blocks. Figure 3.2 depicts each layer of the Residual Block in detail. As mentioned before, each Residual Block has a convolution operation accompanied by strides, with a (3x3) kernel and skip connections. The skip connections are in charge of dealing with the Vanishing Gradients problem, which impacts the performance of previously presented models. The network includes additional shortcut pathways with the use of skip connections, so that the loss estimated from the output layer can find an alternate path if they are not backpropagated to the preceding levels.

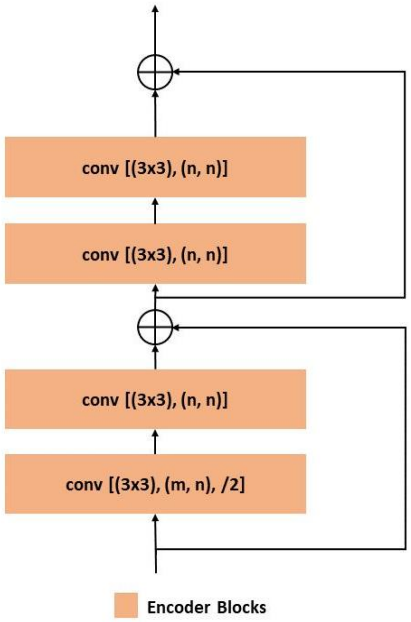


Figure 3.2: Encoder Block with Skip Connections

3.1.1.3 LINKED ARCHITECTURE

Each encoder was linked to each decoder exactly as described in order to reclaim the lost spatial information caused due to numerous downsampling operations in each encoder. Strided convolutions were employed in the encoder to facilitate the linking procedure. Linking encoder blocks to decoder blocks, like in the current LinkNet, has been introduced to recover spatial information lost owing to repeated downsampling. We enabled the linkage by using strided convolutions.

3.1.1.4 DECODER BLOCKS

The Decoder Architecture takes the encoder output as its input. This consists of 4 decoder blocks followed by a final segmentation block for semantic segmentation, as shown in Fig. 3.3.

Each of the decoder blocks has 2 convolutions with (1x1) filter, followed by ‘Upsample’ with a scale factor of 2 in trilinear mode. The Upsample operation is an improvement upon the existing LinkNet, introduced in [27], to counter noisy outputs as a result of transpose convolution. The final segmentation block follows the decoder blocks.

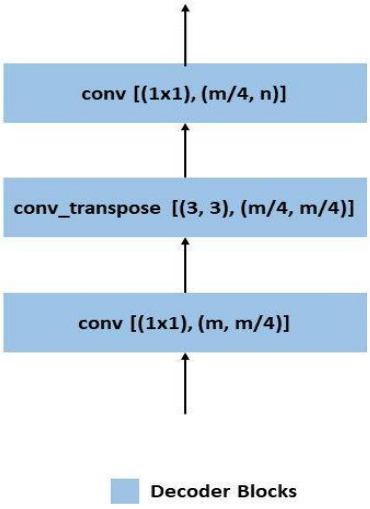


Figure 3.3: Decoder Block

3.1.1.5 SEGMENTATION BLOCK

The primary segmentation work is carried out on the output from Decoder Block 1 by the decoder's final segmentation block. A convolution with a kernel size of (3x3) is applied to the decoder output, and then an upsample operation with a scale factor of 2 in trilinear mode is used to provide a noise-reduced output. Before sending the segmentation mask with pixel probabilities via a softmax layer, another convolution operation with a kernel size of (3x3) is completed.

Hence, The main semantic segmentation is performed on the output from decoder blocks, by the final segmentation block. The decoder output undergoes a convolution operation with a (3x3) filter, followed by Upsampling for noise reduction. The upsampled result undergoes yet another convolution with a (3x3) filter before a Softmax activation is applied.

Table 3.1: LinkModule

Modules	Block	Actions	Out Channel
Encoder	Input-Blocks	Convolution operation and max-pool	64
	Encoder Blocks	Feature Extraction with skip connections	64, 128, 256, 512
Decoder	Upsample	Upscale channels concatenated to the required size	256, 128, 64, 64
	Final Segmentation Blocks	Uses softmax to produce final output mask	1

CHAPTER 4

METHODOLOGY

4.1 Workflow

The initial phase in our model implementation methodology is dataset collecting. Each picture in the dataset was preprocessed after the data was gathered. Augments, standardisation, and normalisation are all part of the preprocessing. Preprocessed pictures are sent into the LinkNet Model for training and validation. The weights have been stored and hosted for use in real time.

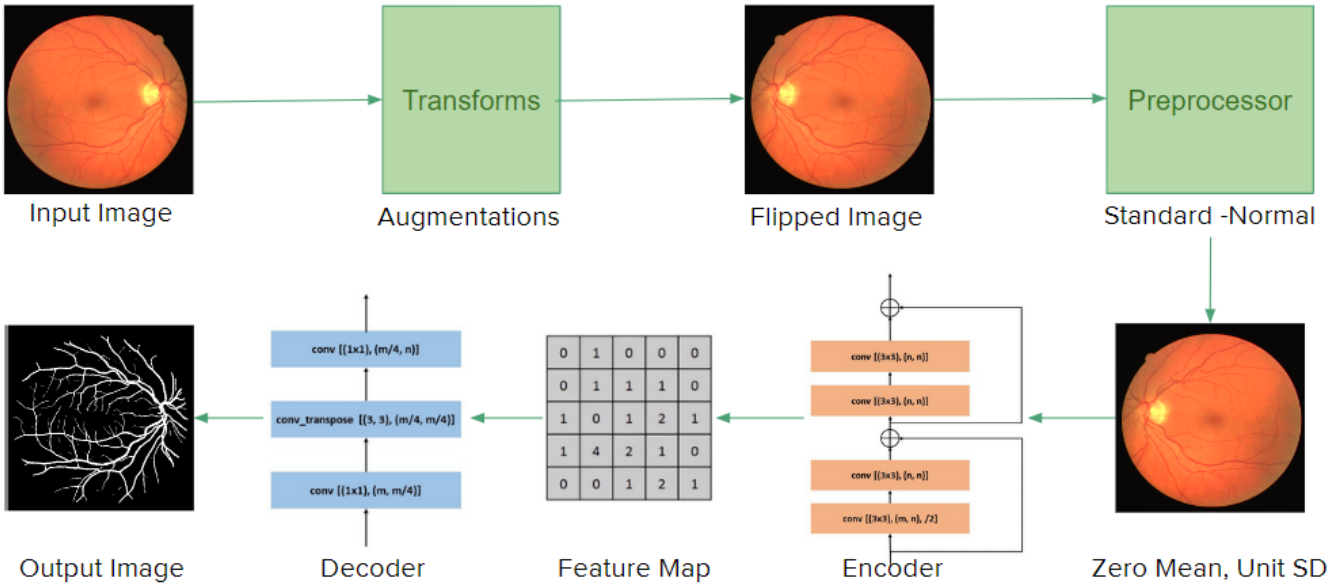


Figure 4.1: Workflow Diagram

4.2 Dataset

For the purpose of segmenting retinal vessels, we employed the DRIVE dataset. It houses a total of 40 JPEG colour fundus photos, including 7 occurrences of the aberrant disease. Each Image is downsized to 512x512 pixels for our model and has three colour channels. The data has been augmented using the argumentation module. Data Augmentation has been accomplished by doing HorizontalFlip, VerticalFlip and Rotate, thereby generating 160 images for training. The use of augmentations and pre-processing procedures like normalization and standardization has

been made. As part of the pre-processing, all images were altered to have zero means and one unit of standard deviation. The model received basic normal Images to learn on.

Out of the total 40 images, twenty were used for the training set and twenty for the testing set. For each image, a circular field of view (FOV) mask with a diameter of roughly 540 pixels is contained in both sets. An ophthalmological professional performed hand segmentation on each image in the practise set. Each picture in the testing set has had two manual segmentations completed by two different observers, with the first observer's segmentation serving as the baseline for performance evaluation. For further enhancement of the training data, augmentation techniques such as horizontal flip and vertical flip were used.

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

Along with the augmentations, pre-processing techniques such as normalisation and standardisation were used; all images have been processed in order to have a zero mean and unit standard deviation, i.e. convert the images to a standard Gaussian distribution as 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. Dice loss is a popular loss function in deep learning, especially for picture segmentation problems. It is a similarity-based loss function that calculates the overlap between the anticipated and true segmentation masks.

Dice loss is defined as follows:

Dice_loss = 1 - (2 * intersection) / (predicted_area + ground_truth_area)

where intersection is the region of overlapping between the predicted mask and the ground truth mask and predicted_area and ground_truth_area are the areas of the predicted and ground truth masks, respectively, and predicted_area and ground_truth_area are the areas of the predicted and ground truth masks.

Because it is sensitive to both FPs (false positives) and FNs (false negatives) the dice loss is a useful loss function. FPs occur when the model wrongly predicts a pixel as being part of the segmentation mask when it is not, and FNs occur when the model incorrectly predicts a pixel as being part of the segmentation mask when it is not. The overlap or resemblance between two sets is calculated using the dice coefficient. We now use Dice Loss as a statistic to assess how well our model is working. The dice coefficient is used to determine if two sets overlap or are similar. For issues including class disparity, this measure is particularly popular. For semantic segmentation, the predicted label and the actual label may be seen as two sets. The Dice coefficient has a value between 0 and 1, with 0 denoting complete overlap and 1 denoting entire overlap. The following formula is used to get the dice coefficient. The dice coefficient is calculated using Equation 4.1.

$$Dice\ Coefficient = 2 \mid T \cap P \mid / \mid T \mid + \mid P \mid \quad \textbf{(4.1)}$$

The dice coefficient is used to calculate the generalized loss function, which is done by subtracting it from 1. The dice coefficient is increased by minimizing this dice loss since a greater value indicates a better overlap. During training, the model may learn to optimise for a segmentation mask that closely fits the ground truth mask while penalising FPs and FNs by using the dice loss as the objective function. This can lead to improved segmentation performance, especially when the class imbalance is severe, or the segmentation mask is sparse.

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. Intersection over Union (IoU) loss is a popular deep learning loss function, notably in object detection and semantic segmentation applications. It is a similarity-based loss function that uses the IoU metric to calculate the overlap between the predicted and ground truth masks.

IoU loss is defined as follows:

IoU_loss = 1 - IoU

where IoU is the ratio of the expected and ground truth masks' intersection to their union: IoU stands for intersection / union.

The intersection and union are determined in the following way:

The overlapping region between the anticipated mask and the ground truth mask is referred to as the intersection. When the predicted mask and the ground truth mask have little overlap or union, the IoU loss penalises the model. This encourages the model to create segmentation masks that are more similar to ground truth masks.

The IoU loss function has an advantage over other loss functions in that it is resistant to class imbalance. This is because it only penalises the model when the anticipated and ground truth masks differ significantly.

Overall, the IoU loss function is a strong and commonly used loss function in deep learning, especially for object detection and semantic segmentation tasks. It is useful for training models to segment pictures properly by assessing the overlapping area between anticipated and ground truth masks using the IoU.

The region that is covered by both the expected mask and the ground truth mask. Similar to the dice loss, it ranges from 0 to 1 , with 0 indicating that there has been no overlap and 1 indicating a total overlap. The Intersection over Union value is calculated as:

$$\textit{Intersection over Union} = |T \cap P| / |T \cup P| \quad \textbf{(4.2)}$$

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 framework was used to code the architecture, and the NVIDIA Tesla T4 GPU and CUDA were integrated combinedly to train it. With a batch size of 4, 80 images were used to train the model. Before being loaded into a data loader object for training, each image underwent pre-processing, was enhanced, normalised, and standardised.

To reduce the dice and IoU loss, Adam optimizer has been used to train the model for 100 iterations at a learning rate of 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

4.5 Sequence Diagram

A sequence diagram is a kind of UML diagram that depicts how items or components in a system interact over time. It depicts the flow of messages or calls between the various components of a system. The following steps are followed in the model, as shown in Fig. 4.3:-

- The client takes a photo and sends the image to the preprocessor.
- The preprocessor performs tasks like standardization, normalization and augmentation.
- The image is passed to the model where feature extraction and semantic segmentation is performed.
- The segmented image is sent as the final output back to the client.

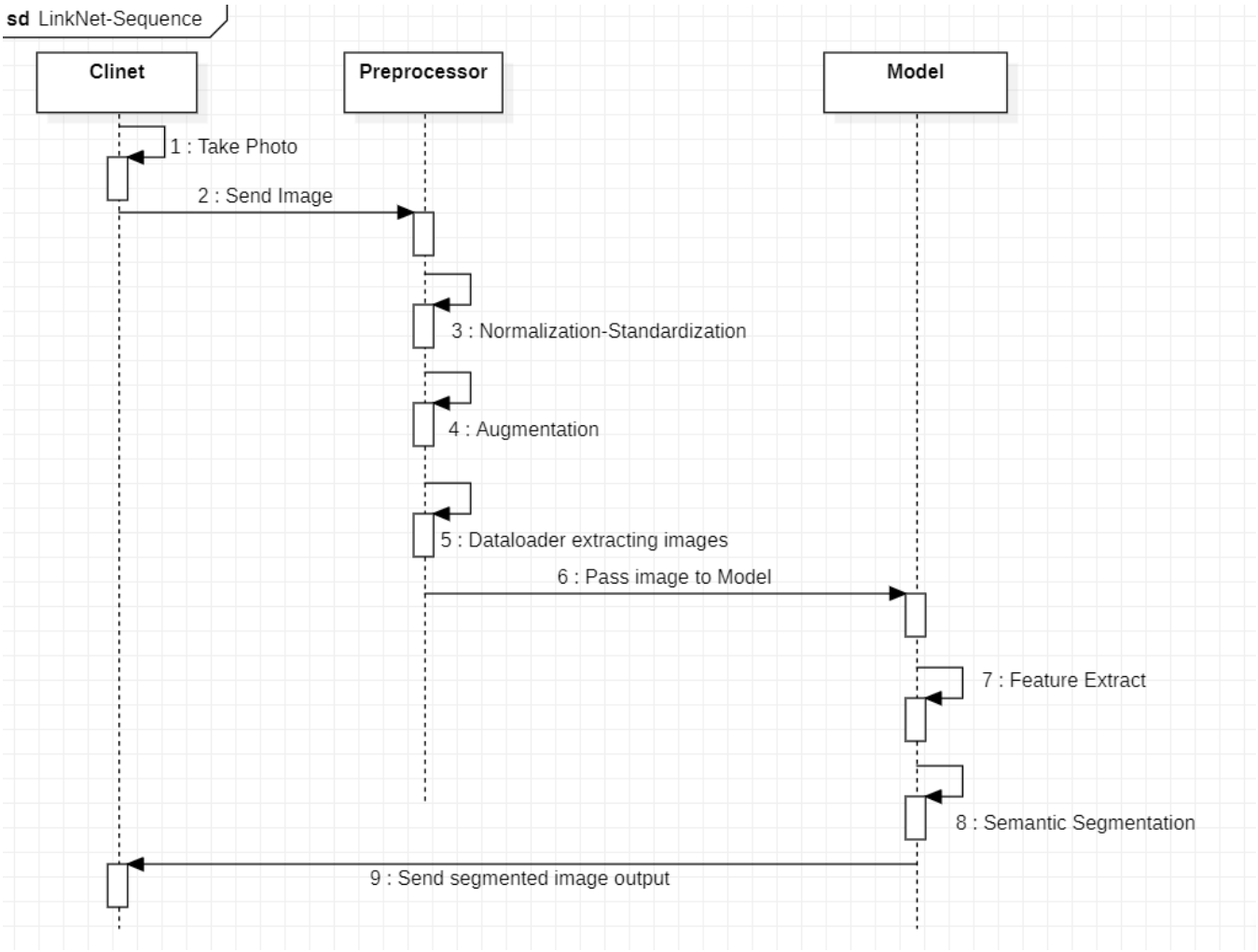


Figure 4.3: Sequence Diagram

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 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 depicts the test results of the input retinal images from DRIVE Dataset and validation results are mentioned in Table 6.1.

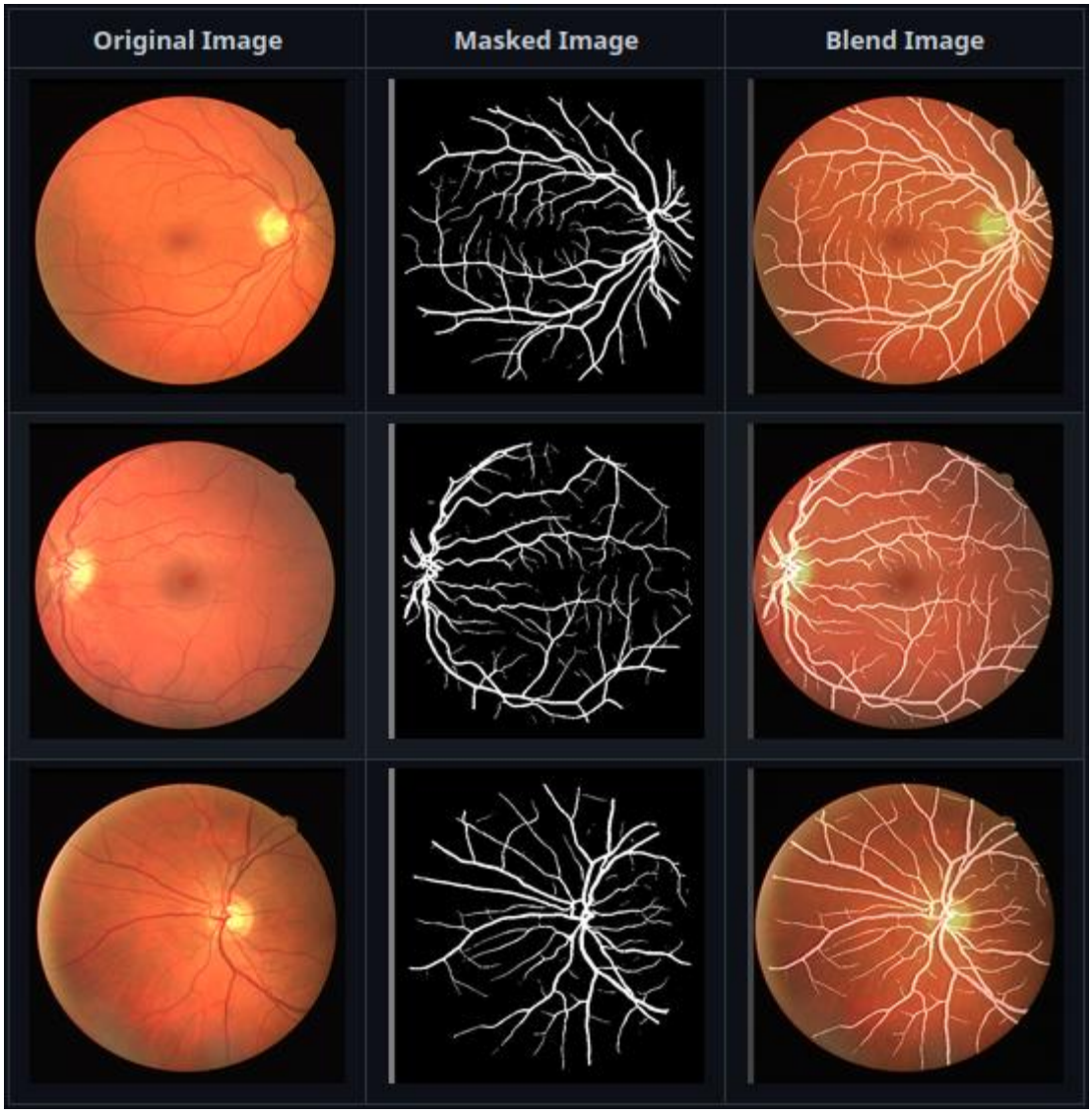


Fig 5.1: Testing Results

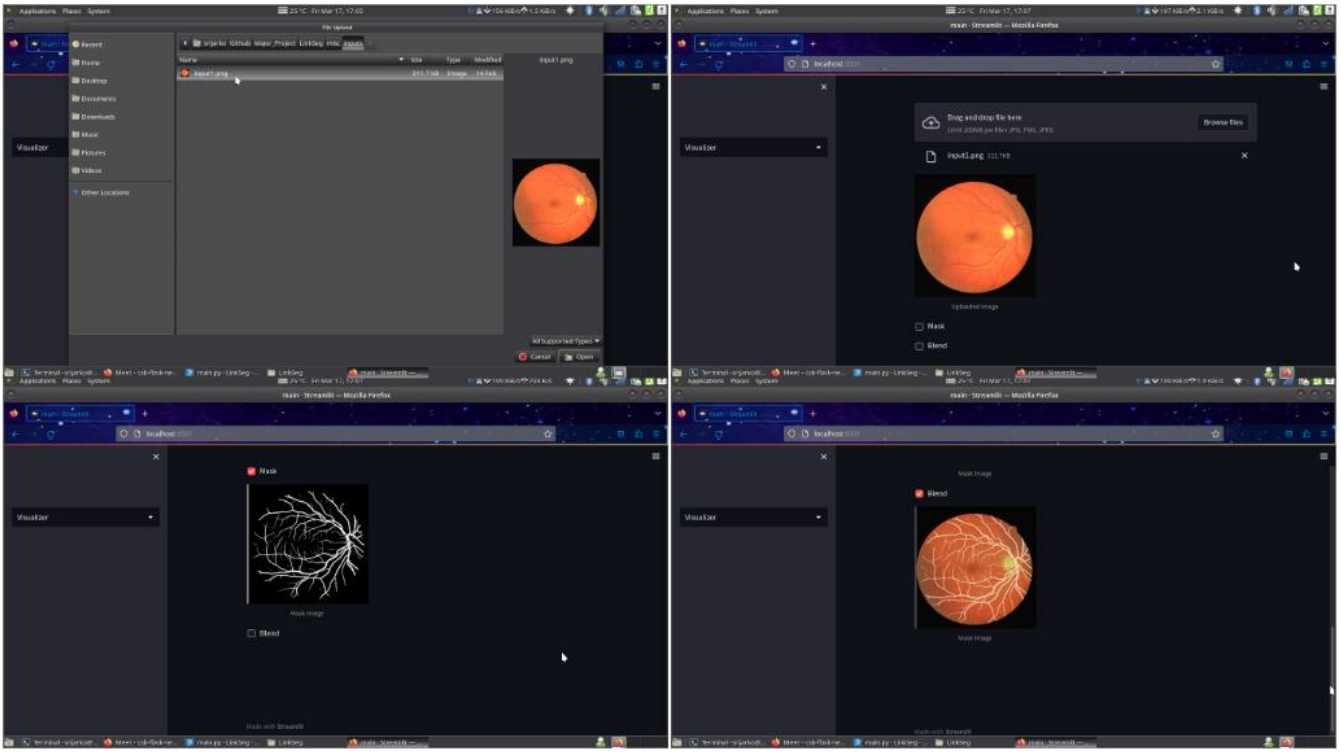


Fig 5.2: Testing Results-2

5.2 Graphs

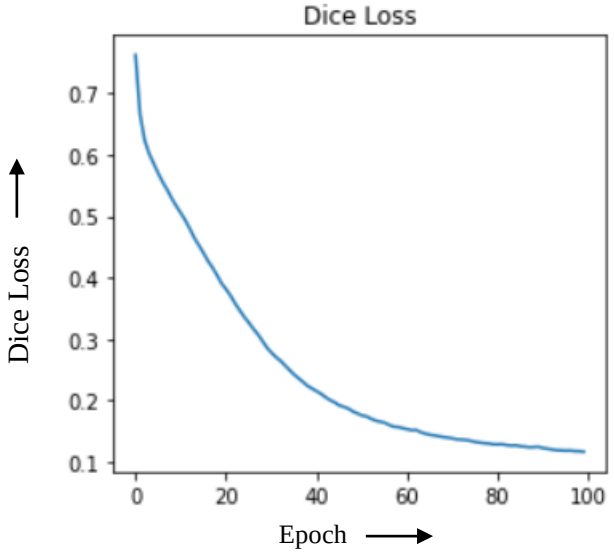


Fig 5.3: Dice Loss

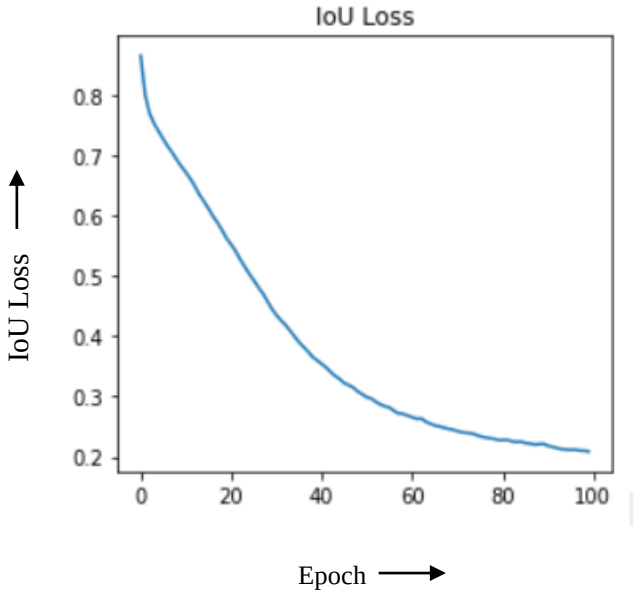


Fig 5.4: IoU Loss

5.3 Tabular Representation of Metrics

Table 5.1: Metrics

METRIC	DRIVE	STARE
Dice Loss	0.1164	0.1277
IoU Loss	0.2086	0.1962

Fig. 5.5: Accuracy Comparison with SOTA

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

Table 5.2: 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 6

CONCLUSION

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 diseases like diabetic retinopathy, age-related macular degeneration and glaucoma. 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 upon 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 more challenging for ophthalmologists to identify 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 diverse 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.

CHAPTER 7

FUTURE WORKS

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.

According to the statement, there is still room for improvement in the retinal image segmentation process by collecting a binary classification dataset with classes healthy and infected and training a binary classification model that can classify the output segmented mask into one of the aforementioned classes.

At the moment, retinal image segmentation algorithms are largely concerned with finding and segmenting diseased features such as lesions or blood vessels. However, it is necessary to distinguish between healthy and infected retinal regions. This is critical for diagnosing and monitoring disorders like as diabetic retinopathy, which can cause subtle changes in the retina that standard segmentation approaches may overlook.

The initial step would be to collect a binary classification dataset of healthy and diseased retinal pictures.To ensure accuracy and trustworthiness, medical practitioners would need to carefully choose and categorise these photos. Following the collection of the dataset, a binary classification model could be trained using machine learning techniques such as logistic regression, support vector machines, or deep learning models such as convolutional neural networks.The trained binary classification model might then be used to categorise the output segmented mask as healthy or contaminated. This would allow for automatic illness diagnosis based on segmented retinal pictures. Medical personnel might further analyse and interpret the information to identify the severity of the ailment and the best treatment option.

Overall, this method has the potential to increase the accuracy and efficiency of retinal picture segmentation and disease identification. It has the potential to change the way we detect and manage retinal illnesses by allowing for faster and more accurate diagnosis and increasing patient outcomes.

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 retionopathy 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.

APPENDIX 1

Details about the language, programmes, and packages implemented in our project has been provided in this section.

This project was developed using Python, a general-purpose, interactive, interpreted, object-oriented, high-level programming language. It offers readable and transparent code. AI and ML algorithms may help developers create reliable machine intelligent systems when they are implemented in Python. Our project uses the following set of Python packages:

- **Pytorch:** Deep learning models are created and trained using PyTorch, an open-source machine learning library. It is based on the Torch library and was created mostly by Facebook's artificial intelligence research team. Users may create and change computational graphs on-the-fly using the dynamic computational graph architecture that PyTorch offers. As a result, sophisticated neural network topologies can be defined and modified with ease, and you can experiment with various hyperparameters and designs without having to recompile code. Deep learning models are created and trained using PyTorch, an open-source machine learning library.

- **Streamlit:** With the help of the free and open-source Python library Streamlit, you can rapidly and easily create web apps for data science and machine learning projects. You don't need to master HTML, CSS, or JavaScript to build interactive dashboards, data visualisations, and machine learning apps with Streamlit. The user may create user interfaces for your machine learning and data science projects with Streamlit by writing Python code. With the help of the library's straightforward methods, you can easily build widgets that let you interact with your models and data, including sliders, dropdown menus, and text boxes. It is very simple to incorporate data visualisations made with other Python tools, such as Matplotlib, Plotly, and Bokeh, using Streamlit

- **Albumentations:** An open-source Python library called Albumentations is used in applications using computer vision and machine learning to enhance images. It offers a comprehensive selection of picture augmentation methods that may be applied to create fresh training data for deep learning models and enhance their functionality. Numerous image augmentation methods are supported by Albumentations, including random cropping, flipping, rotation, scaling, colour jittering, and many more. These

methods may be mixed and matched in a variety of ways to produce a huge collection of unique and realistic training pictures. The speed of Albumentations is one of its important characteristics. It is appropriate for usage in large-scale machine learning projects since it is made to be extremely efficient and can analyse thousands of photos per second.

- **Pillow:** Popular open-source Python library for working with images is called Pillow. The Python Imaging Library (PIL), from which it was forked, was no longer being actively developed. JPEG, PNG, BMP, TIFF, and GIF are just a few of the image types you may import and edit using Pillow. The library offers a wide range of image processing tools, such as tools for scaling, cropping, rotating, flipping, and changing between different picture modes. A variety of picture improvement methods are also supported by Pillow, including colour space conversions, contrast and brightness modifications, and image filtering. Pillow also lets you create and annotate pictures using a range of tools, including lines, shapes, and text.
- **Numpy:** The open-source Python library known as NumPy (short for Numerical Python) supports multidimensional arrays, matrices, and mathematical functions for numerical computing. It is a core component of the scientific Python ecosystem and is frequently used for scientific computing, machine learning, and data analysis. One can easily execute mathematical operations on arrays and matrices using NumPy's robust array computing functionality. Additionally, it has functions for Fourier transformations, random number generation, and linear algebra, among many others. Python lists are less effective than NumPy arrays in handling huge data sets, while NumPy arrays are faster and use less memory. Because of this, NumPy is especially beneficial for numerical calculations that need to be carried out quickly.
- **Pandas:** Pandas is an open-source package in Python that offers simple data structures and analytical capabilities for handling and analysing data. It is frequently used for data cleansing, transformation, and analysis in data science and data analysis projects. Series and DataFrame are the primary data structures that Pandas offer. A DataFrame is a two-dimensional table-like data structure with columns of possibly distinct kinds, while a Series is a one-dimensional array-like object that may store any data type. You can quickly read and write data from many different data sources, including CSV files, Excel spreadsheets, SQL databases, and more, using Pandas. Additionally, it offers a vast array of data cleaning and manipulation features, including data filtering, grouping, merging, and reshaping.

- **Matplotlib:** A well-liked open-source Python toolkit called Matplotlib is used to develop interactive, animated, and static visualisations in Python. It has several classes and functions, including as line graphs, scatter plots, bar plots, histograms, and more, for representing data in both 2D and 3D. Colours, typefaces, labels, and comments are just a few of the formatting and stylistic choices available in Matplotlib's extensive customization capabilities. It also works nicely with other Python libraries like NumPy, Pandas, and SciPy, making it a potent tool for data analysis and visualisation. Plotting may be done using a variety of Matplotlib APIs, including an object-oriented interface and a state-based interface akin to MATLAB. While the object-oriented interface offers greater flexibility and control, the state-based interface is easier to use and appropriate for straightforward plots.
- **Gdown:** A Python command-line utility called "gdown" may be used to download big files from Google Drive. It can be quickly installed and used from the command line and leverages the Google Drive API to download files from a shared link. Gdown also allows you to download from shared folders using a link and resume stopped downloads. In general, gdown is a helpful tool for quickly downloading big files from Google Drive, especially when the file can't be downloaded straight from the URL.

APPENDIX 2

This section contains the source code and the GUI for the web application.

A 2.1 Code

2.1.1 DATA

augment.py ● | linknet.py | dataset.py | training_utils.py | plotter.py | .gitignore | main.py | {} weights_do

data > augment.py

```python
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):

    ### Split data into train and test set
    train_x = sorted(glob(os.path.join(path, "training", "images", "*.tif")))
    train_y = sorted(glob(os.path.join(path, "training", "1st_manual", "*.gif")))

    test_x = sorted(glob(os.path.join(path, "test", "images", "*.tif")))
    test_y = sorted(glob(os.path.join(path, "test", "mask", "*.gif")))

    return (train_x, train_y), (test_x, test_y)
```

Fig A 2.1: Load Data Function

augment.py | linknet.py | dataset.py | training_utils.py | plotter.py | .gitignore | main.py | {} weights_down

data > augment.py

```python
def augment_data(images, masks, save_path, augment=True):

    ### Augmenting data into Horizontal,Vertical & Rotate

    size = (512, 512)

    for idx, (x, y) in tqdm(enumerate(zip(images, masks)), total=len(images)):
        """ Extracting the name """
        name = x.split("/")[-1].split(".")[0]

        """ Reading image and mask """
        x = cv2.imread(x, cv2.IMREAD_COLOR)
        y = imageio.mimread(y)[0]

        if augment == True:
            aug = HorizontalFlip(p=1.0)
            augmented = aug(image=x, mask=y)
            x1 = augmented["image"]
            y1 = augmented["mask"]

            aug = VerticalFlip(p=1.0)
            augmented = aug(image=x, mask=y)
            x2 = augmented["image"]
            y2 = augmented["mask"]

            aug = Rotate(limit=45, p=1.0)
            augmented = aug(image=x, mask=y)
            x3 = augmented["image"]
            y3 = augmented["mask"]
```

Fig A 2.2: Data Augmentation Function

augment.py | linknet.py | dataset.py | training_utils.py | plotter.py | .gitignore | main.py

data > augment.py

```python
            X = [x, x1, x2, x3]
            Y = [y, y1, y2, y3]

        else:
            X = [x]
            Y = [y]

        index = 0
        for i, m in zip(X, Y):
            i = cv2.resize(i, size)
            m = cv2.resize(m, size)

            tmp_image_name = f"{name}_{index}.png"
            tmp_mask_name = f"{name}_{index}.png"

            image_path = os.path.join(save_path, "image", tmp_image_name)
            mask_path = os.path.join(save_path, "mask", tmp_mask_name)

            cv2.imwrite(image_path, i)
            cv2.imwrite(mask_path, m)

            index += 1

## Usage ##

# if __name__ == "__main__":
#     """ Seeding """
#     np.random.seed(42)

#     """ Load the data """
```

Fig A 2.3: Data Augmentation Function

augment.py | 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 A 2.4: Data Augmentation Function Call

linknet.py | dataset.py | training_utils.py | plotter.py | .gitignore | main.py | weights_downloa

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
```

Fig A 2.5: PyTorch Data Loader

A 2.1.2 MODEL ARCHITECTURE

linknet.py | training_utils.py | plotter.py | .gitignore | main.py | weights_download.json | inference.py | test.py

linkseg > linknet.py

```
        self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size, stride, padding, groups=groups, bias=bias)
        self.bn1 = nn.BatchNorm2d(out_planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_planes, out_planes, kernel_size, 1, padding, groups=groups, bias=bias)
        self.bn2 = nn.BatchNorm2d(out_planes)
        self.downsample = None

        if stride > 1:
            self.downsample = nn.Sequential(nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, bias=False),
                                nn.BatchNorm2d(out_planes),)

    def forward(self, x):

        residual = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            residual = self.downsample(x)

        out += residual
        out = self.relu(out)

        return out
```

Fig A 2.6: Basic Block

linknet.py | training_utils.py | plotter.py | .gitignore | main.py | weights_download.json | inference.py

linkseg > linknet.py

```
        """
        Parameters:

        - in_planes: no. of input channels

        - out_planes: no. of output channels

        - kernel_size: kernel size for conv in each block

        - stride(default: 1): stride to be assigned to each block

        - padding(default: 0): amount of padding to be assigned to each block

        - groups(default: 1): groups to be assigned to each block

        - bias(default: False): boolean bias to be assigned to each block

        """

        super(Encoder, self).__init__()

        self.block1 = BasicBlock(in_planes, out_planes, kernel_size, stride, padding, groups, bias)
        self.block2 = BasicBlock(out_planes, out_planes, kernel_size, 1, padding, groups, bias)

    def forward(self, x):

        x = self.block1(x)
        x = self.block2(x)

        return x
```

Fig. A 2.7: 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, bia
                               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 A 2.8: Decoder Blocks

linknet.py ● | training_utils.py | plotter.py | .gitignore | main.py | {} weights_download.json | infere

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
```

Fig. A 2.9: LinkNet Blocks

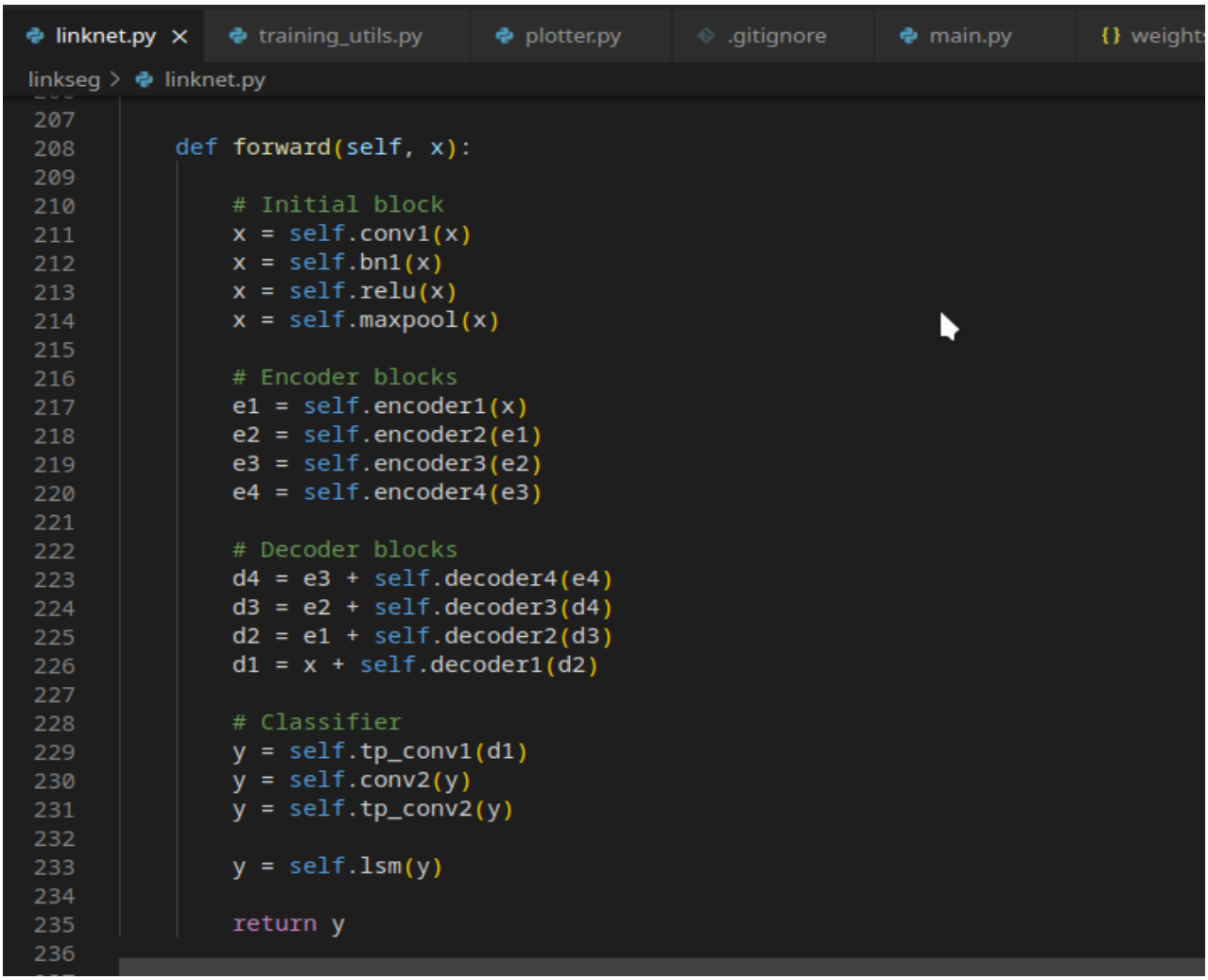


Fig. A 2.10: LinkNet Architecture

A 2.1.3 TRAINING

linkseg > loss.py

```python
import torch
import torch.nn as nn

import warnings
warnings.filterwarnings('ignore')

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. A 2.11: Dice Loss

```python
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 + smooth)

        return iou, 1 - iou
```

Fig A 2.12: 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 A 2.13: 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 A 2.14: Validation Function

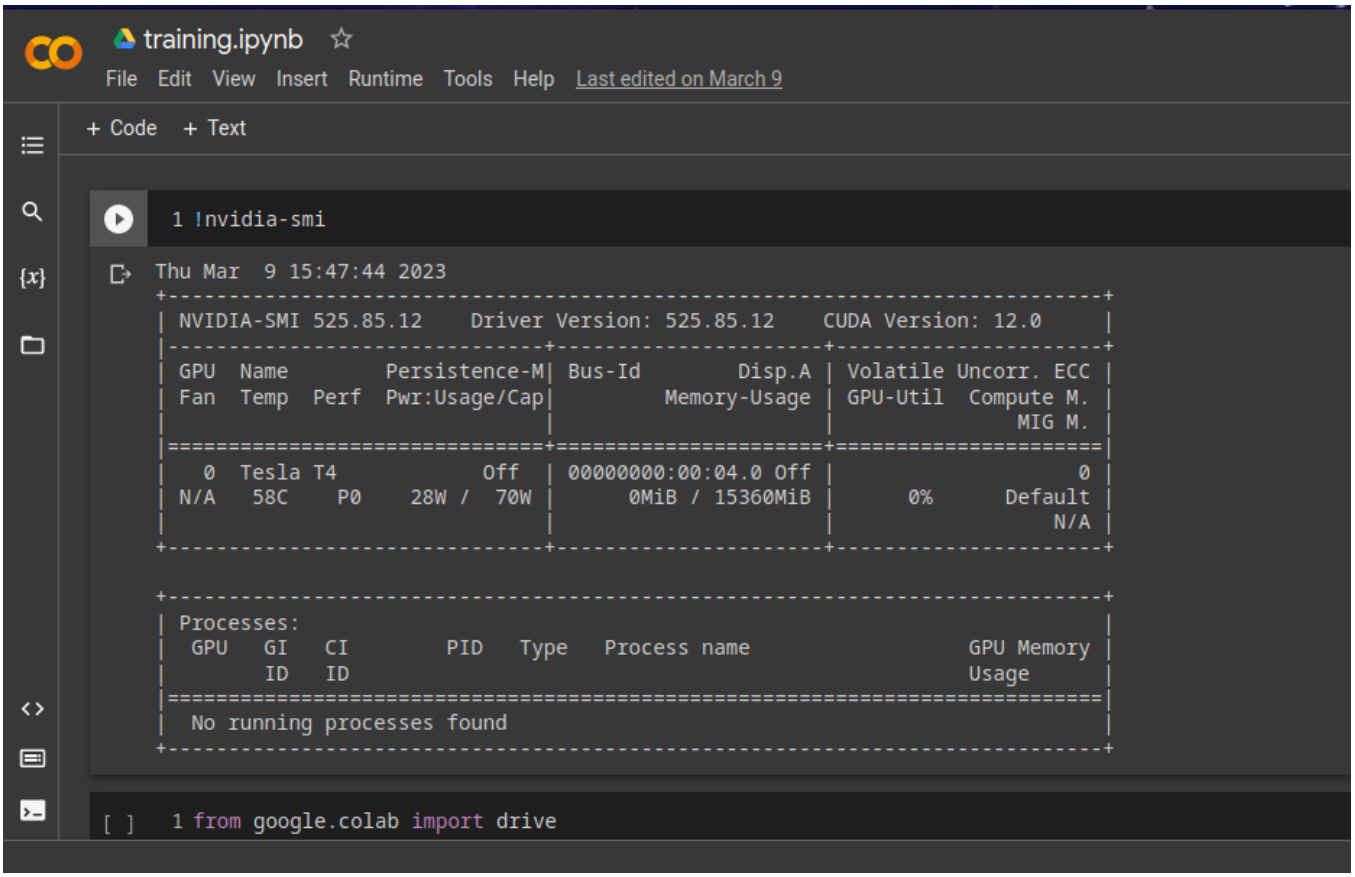


Fig A 2.15: Nvidia Tesla T4

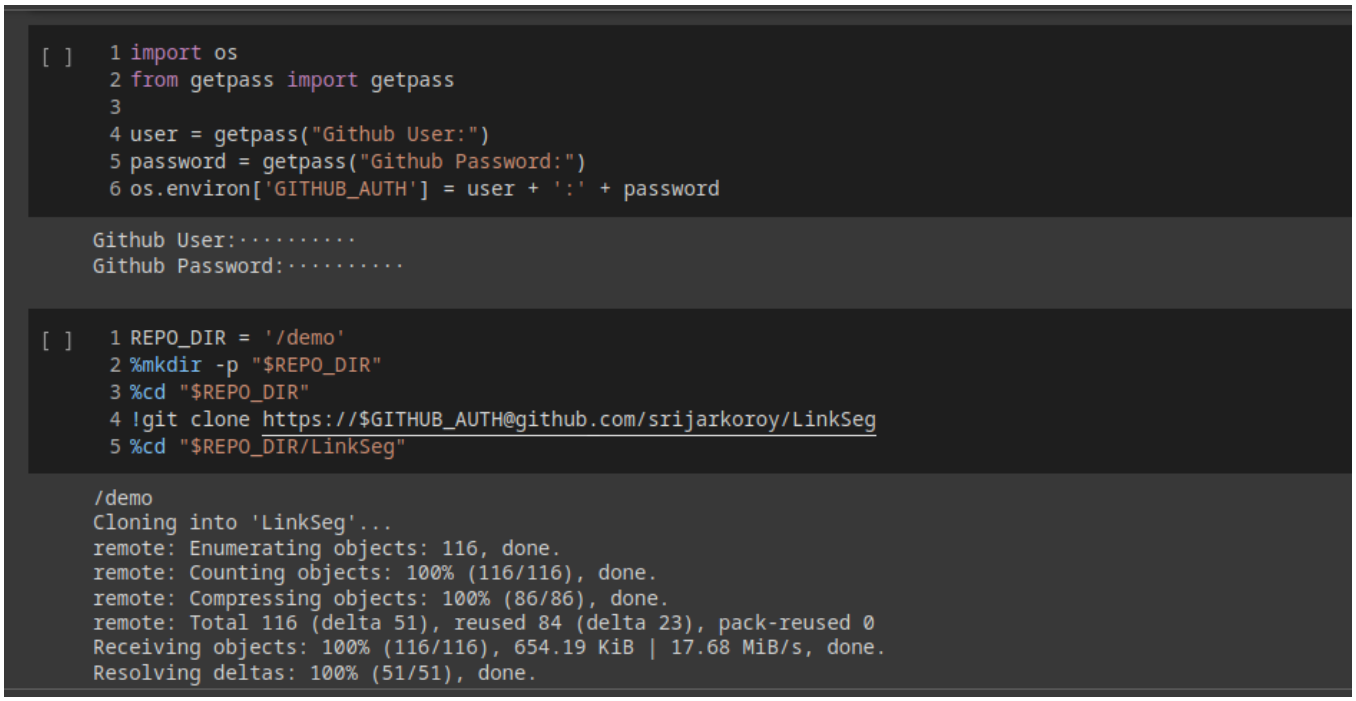


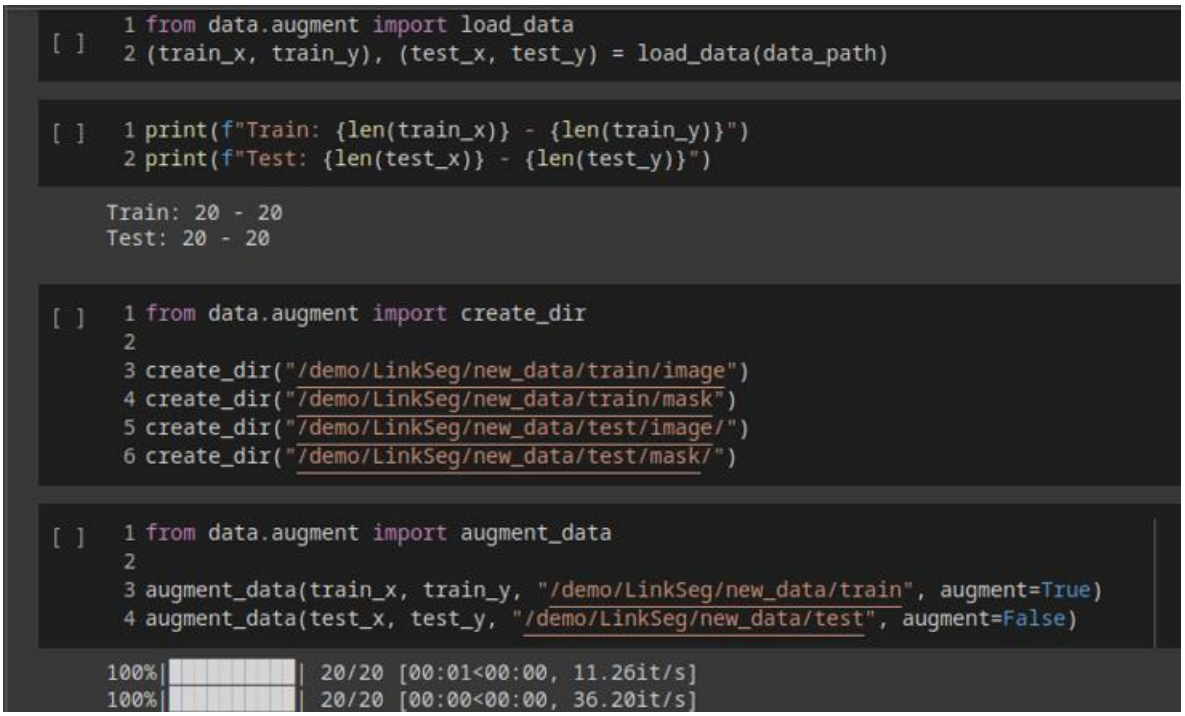
Fig A 2.16: Cloning

```
[ ]  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 A 2.17: Unzipping the Data

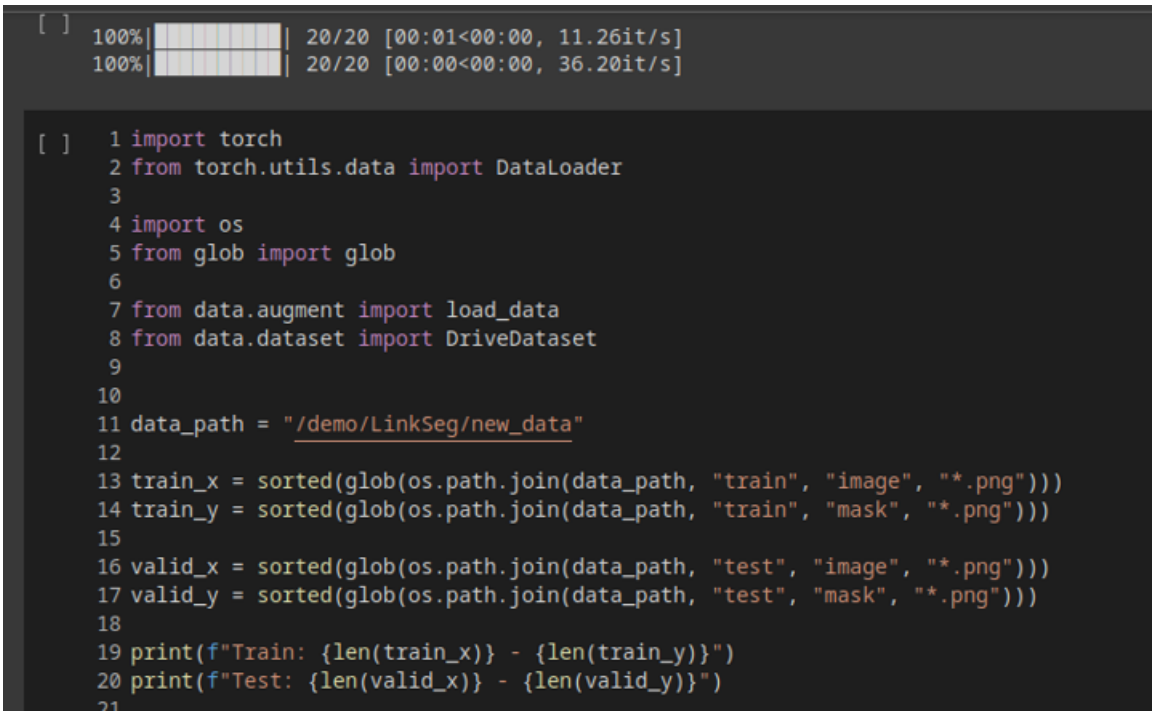
```
[ ]  1 data_path = "/demo/LinkSeg/dataset"
     2 print(data_path)

/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 A 2.18: 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 A 2.19: 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", "*.png")))
    14 train_y = sorted(glob(os.path.join(data_path, "train", "mask", "*.png")))
    15
    16 valid_x = sorted(glob(os.path.join(data_path, "test", "image", "*.png")))
    17 valid_y = sorted(glob(os.path.join(data_path, "test", "mask", "*.png")))
    18
    19 print(f"Train: {len(train_x)} - {len(train_y)}")
    20 print(f"Test: {len(valid_x)} - {len(valid_y)}")
```

Fig A 2.20: Augmented Data

```
[ ]
print(f"Train: {len(train_x)} - {len(train_y)}")
print(f"Test: {len(valid_x)} - {len(valid_y)}")

train_dataset = DriveDataset(train_x, train_y)

train_loader = DataLoader(
        dataset=train_dataset,
        batch_size=4,
        shuffle=True,
        num_workers=2
    )

val_dataset = DriveDataset(valid_x, valid_y)

val_loader = DataLoader(
        dataset=val_dataset,
        batch_size=4,
        shuffle=False,
        num_workers=2
    )
```

```
Train: 80 - 80
Test: 20 - 20
```

Fig A 2.21: 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 A 2.22: 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 A 2.23: Training Function Call

```
  0%|          | 0/100 [00:00<?, ?it/s]Epoch:  0
Shifting the model to cuda!
Train Dice Loss: 0.7625233799219131,  Train IoU Loss: 0.8645183622837067,
Shifting 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
Shifting the model to cuda!
Train Dice Loss: 0.6659743875265122,  Train IoU Loss: 0.7993300944566727,
Shifting 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
Shifting the model to cuda!
Train Dice Loss: 0.6241493344306945,  Train IoU Loss: 0.7683533012866974,
Shifting 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
Shifting the model to cuda!
Train Dice Loss: 0.6021922618150711,  Train IoU Loss: 0.7515868365764617,
Shifting 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
Shifting the model to cuda!
Train Dice Loss: 0.5860089540481568,  Train IoU Loss: 0.7388259142637252,
```

Fig A 2.24: Training

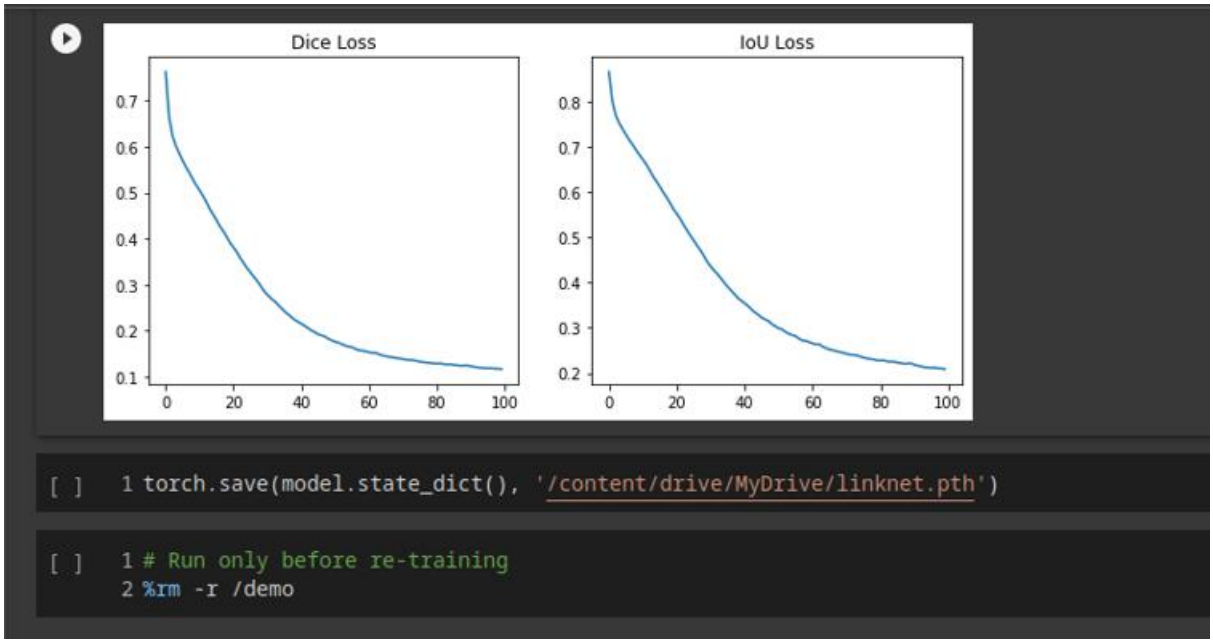


Fig A 2.25: Training Progression

A 2.1.4 TESTING

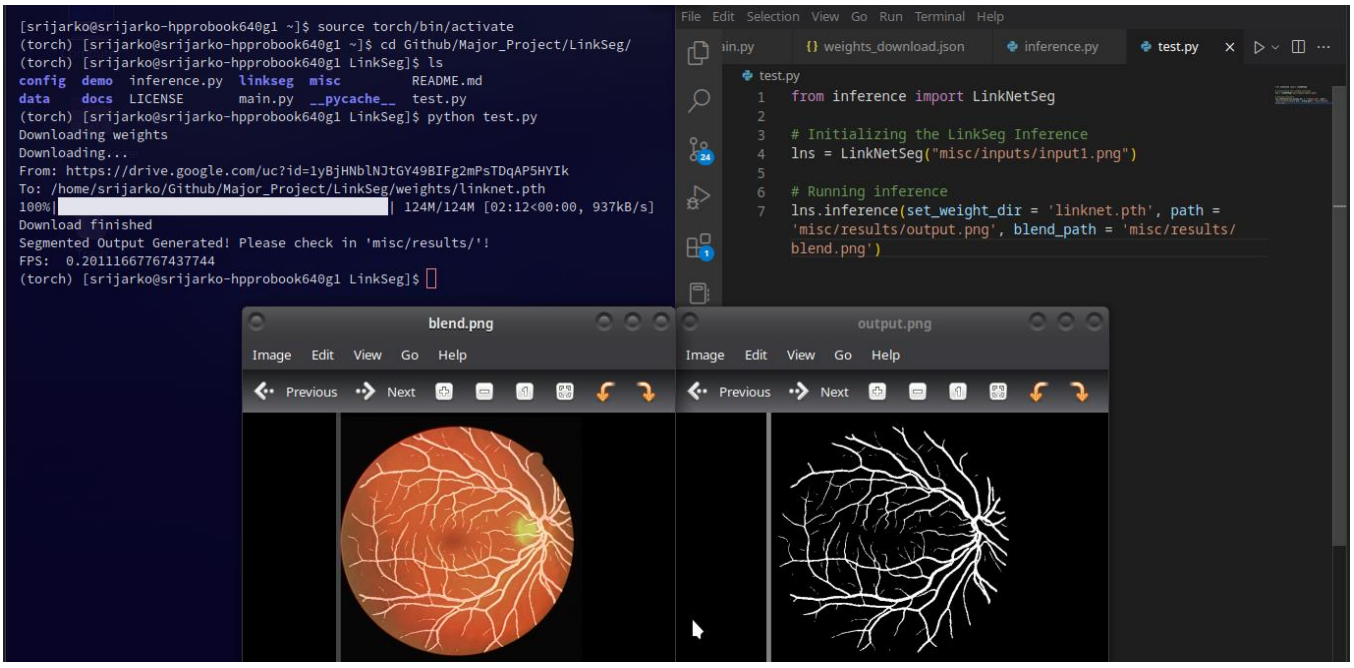


Fig A 2.26: Testing

A 2.2 User Interface

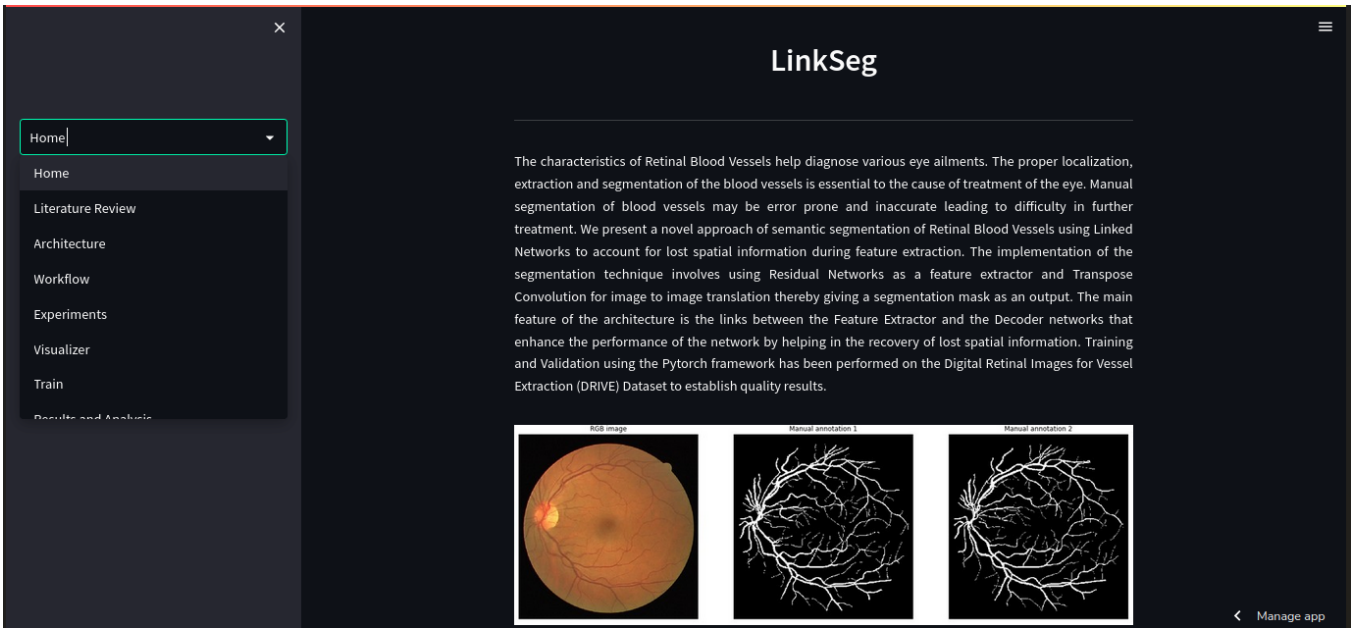


Fig A 2.27: Home Page

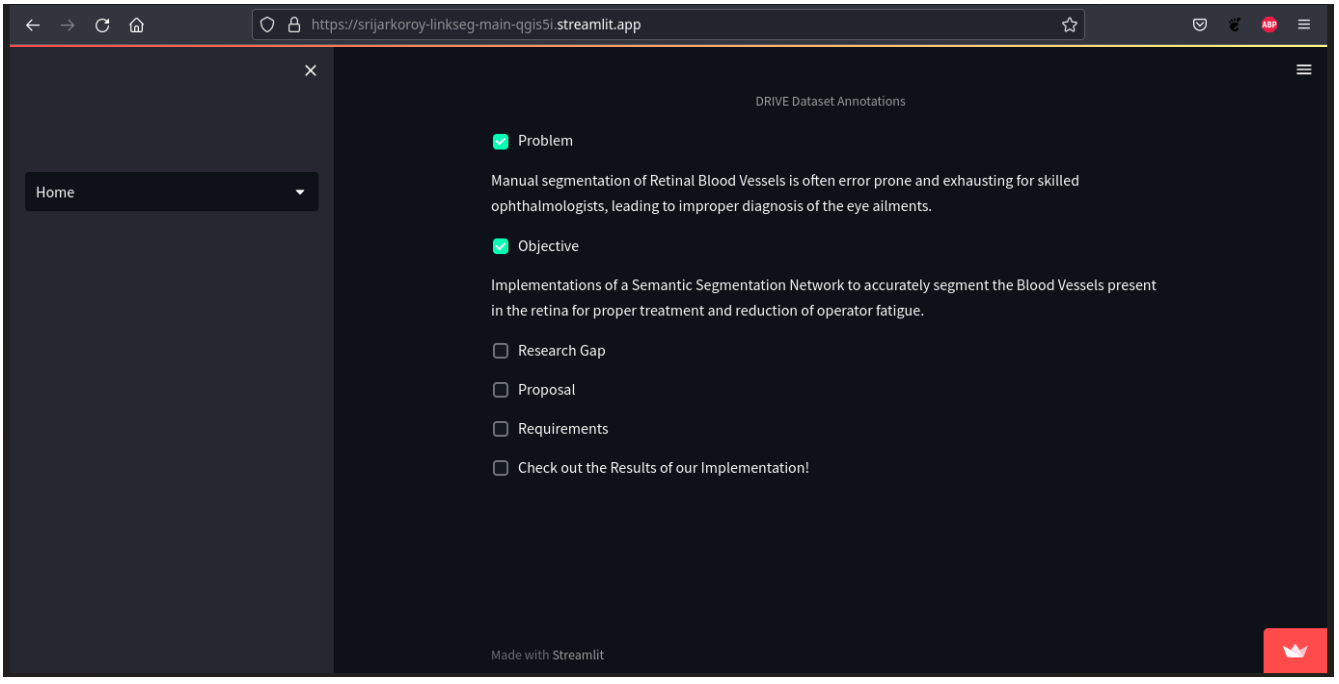


Fig A 2.28: Problem and Objective

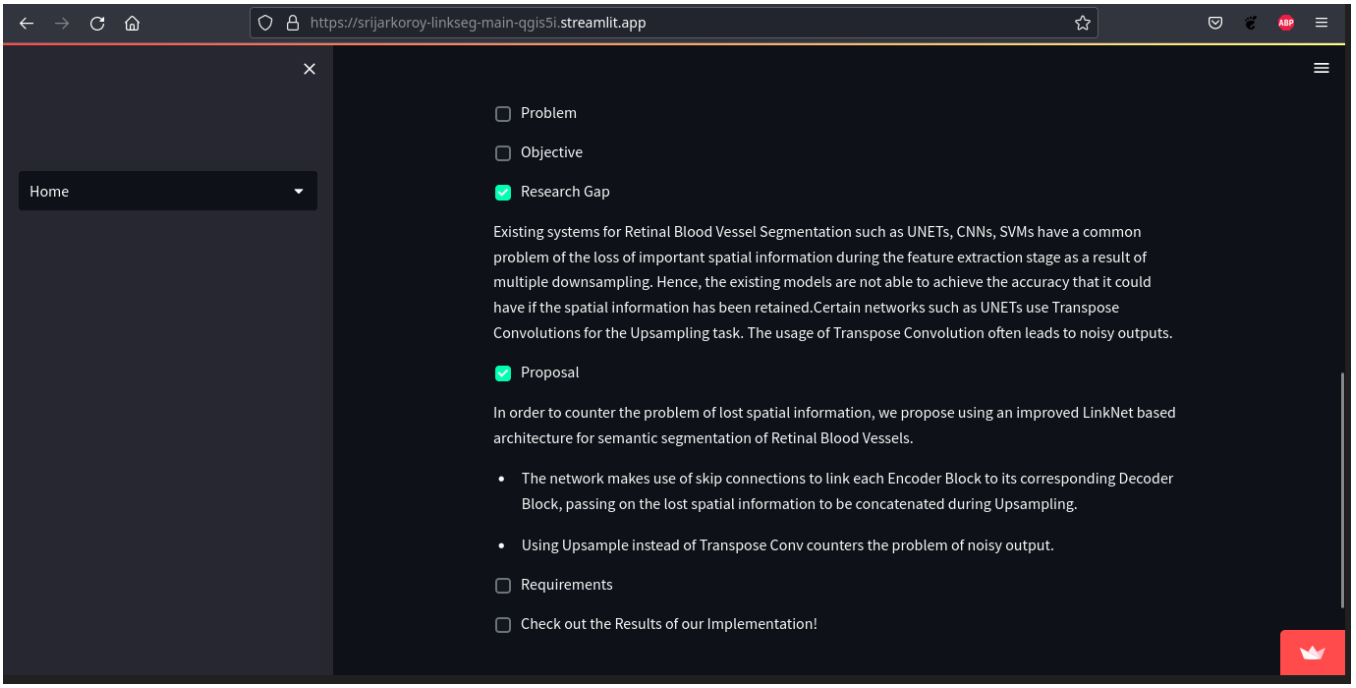


Fig A 2.29: Research Gap and Proposal

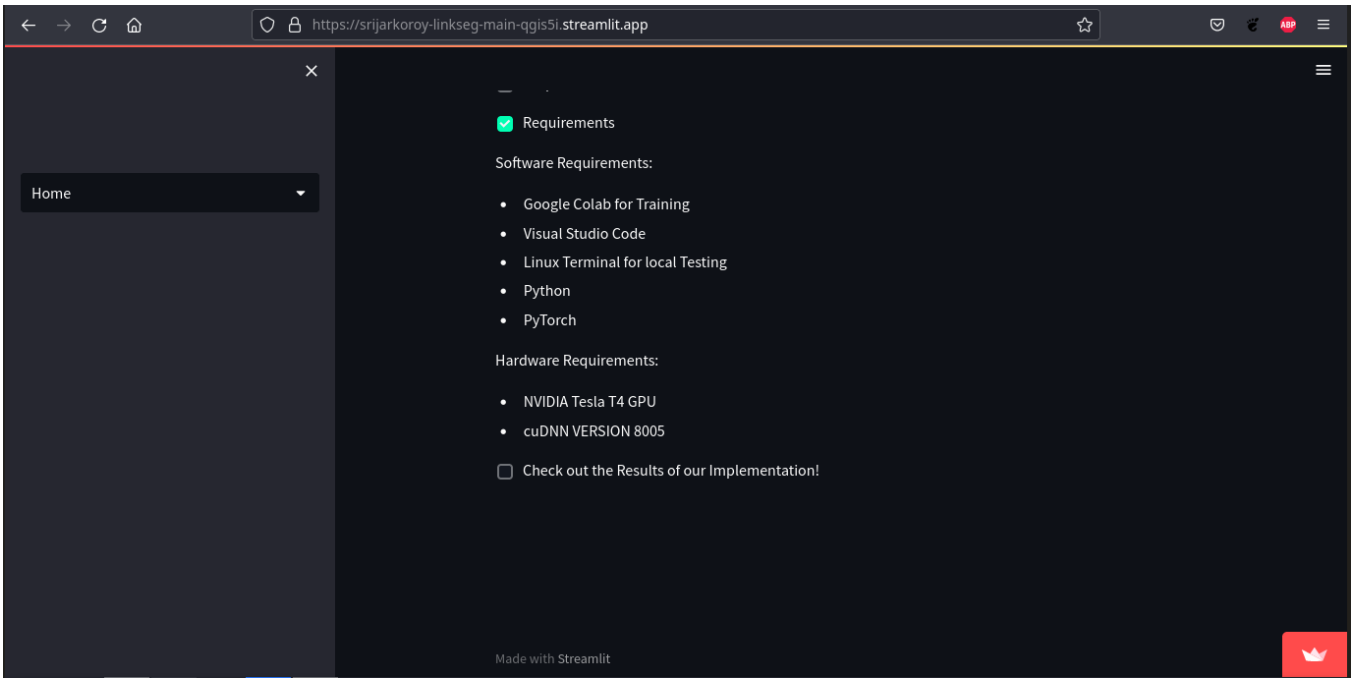


Fig A 2.30: Requirements

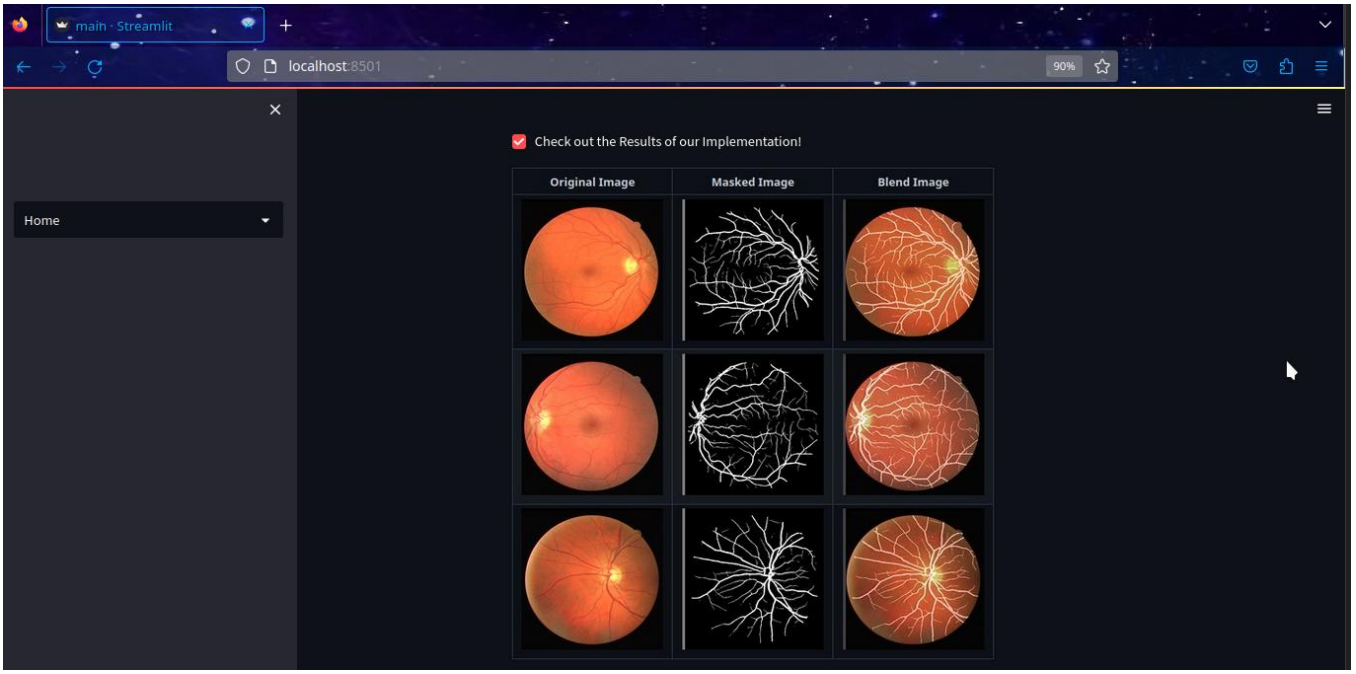


Fig A 2.31: Results

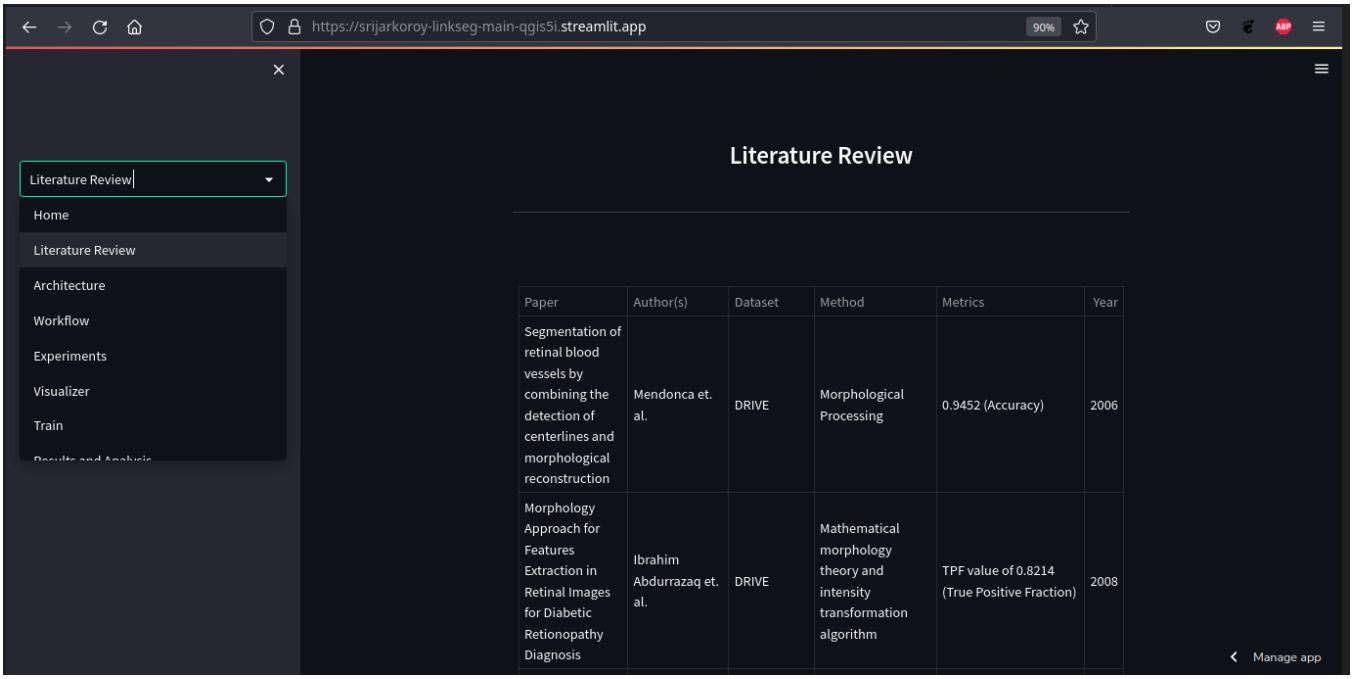
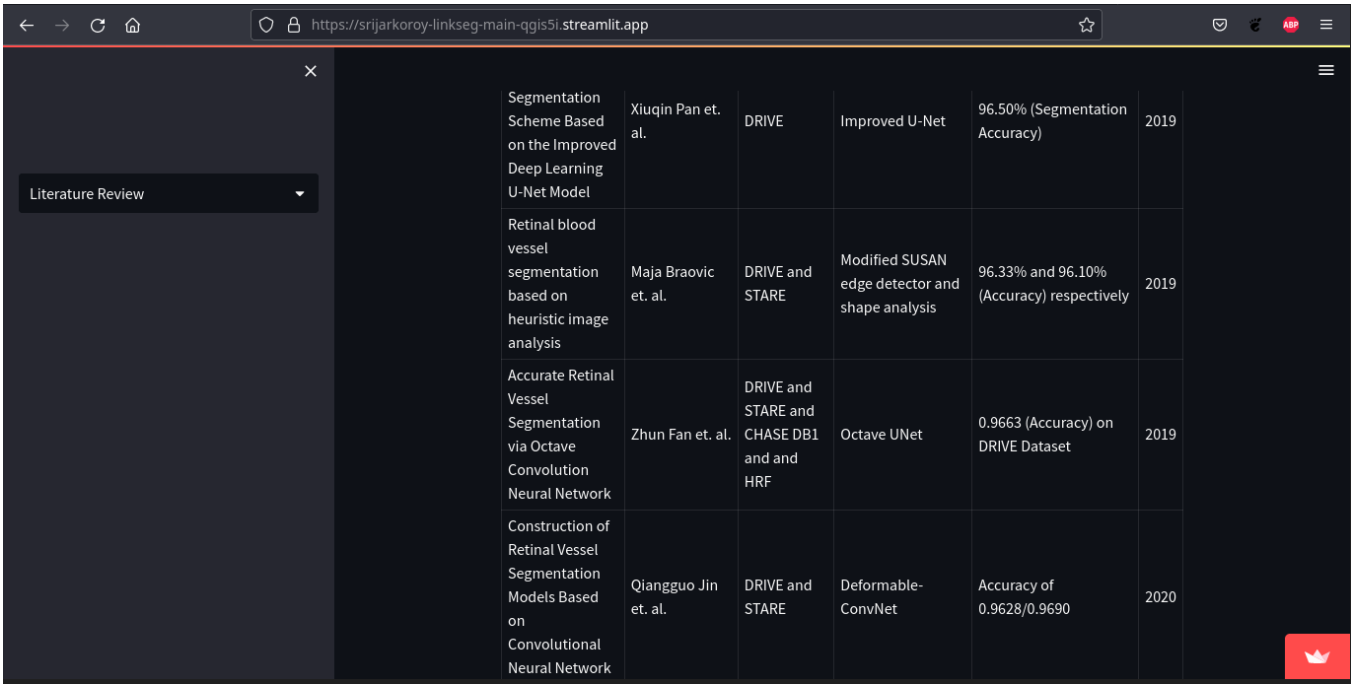


Fig A 2.32: Literature Review - 1



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 A 2.33: Literature Review - 2

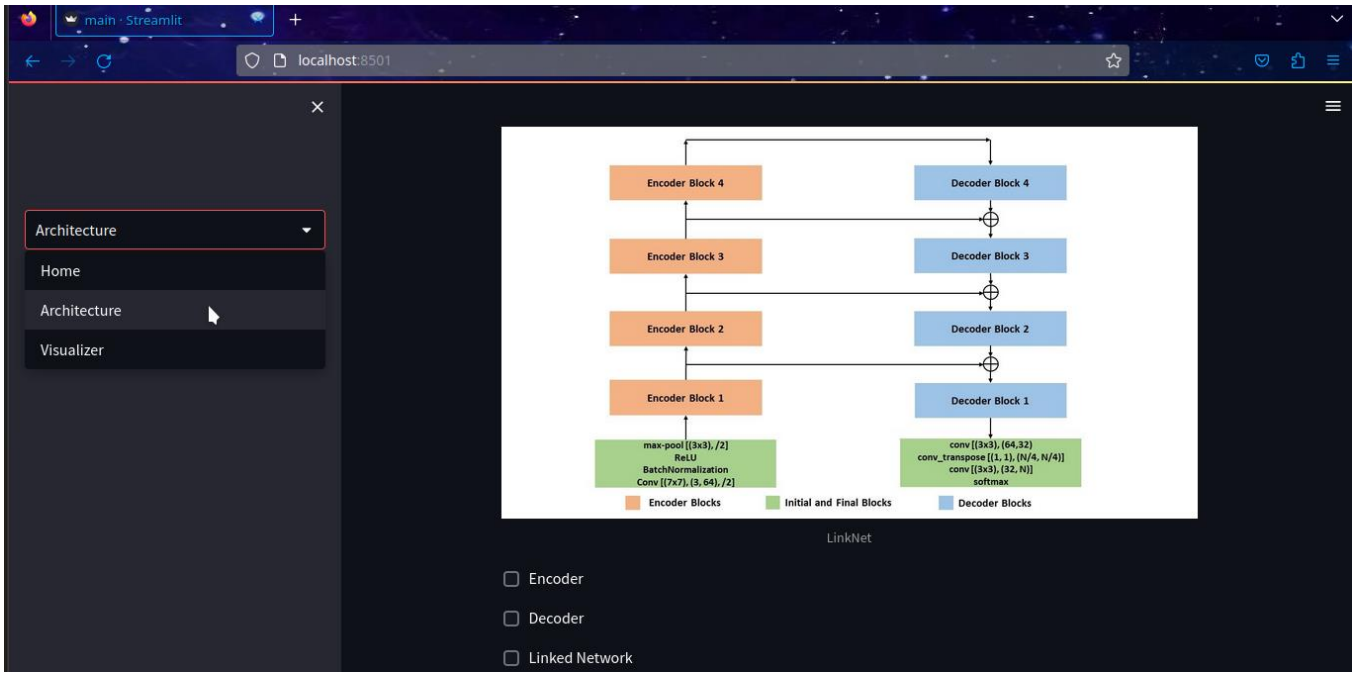


Fig A 2.34: Architecture

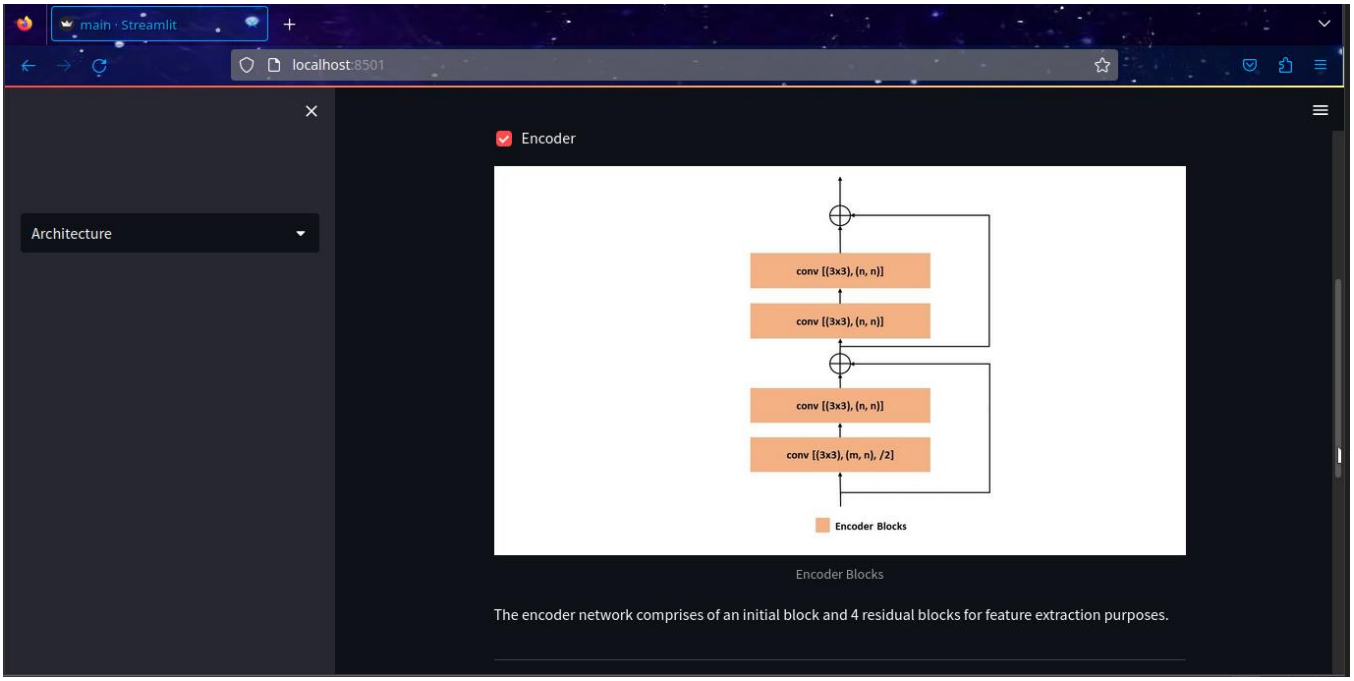


Fig A 2.35: Encoder

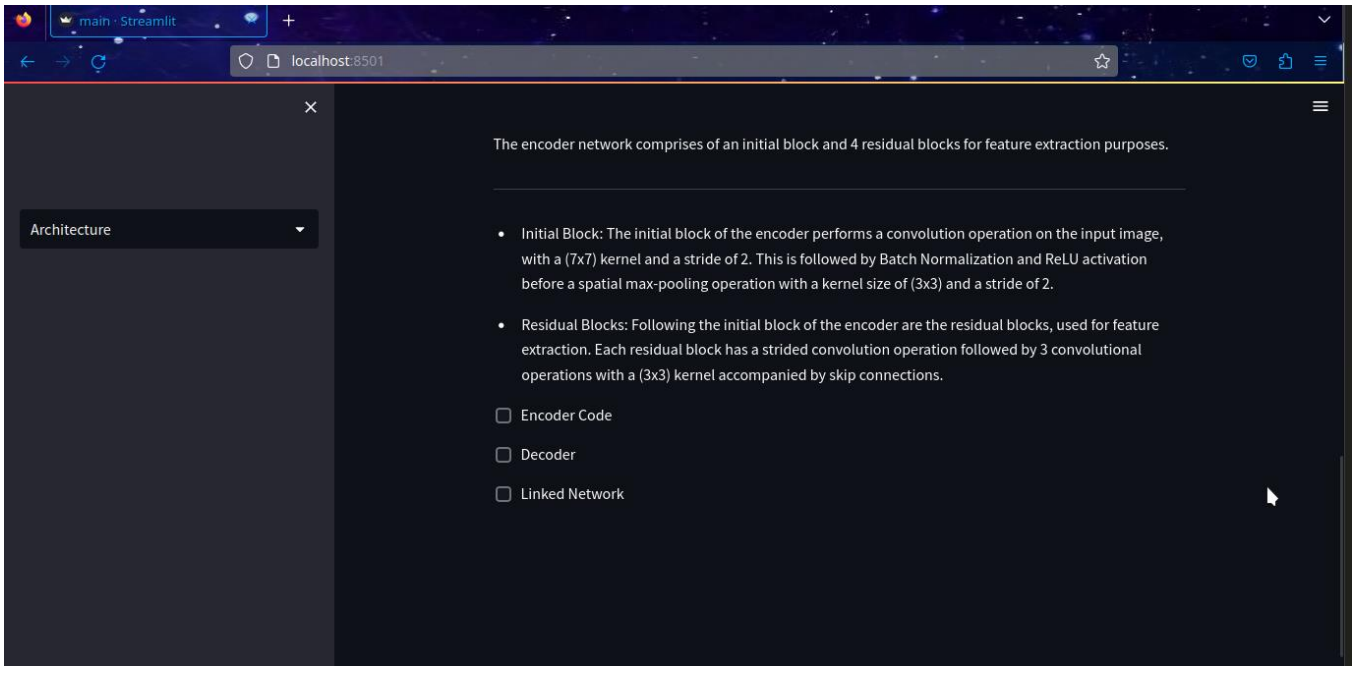


Fig A 2.36: Encoder Theory

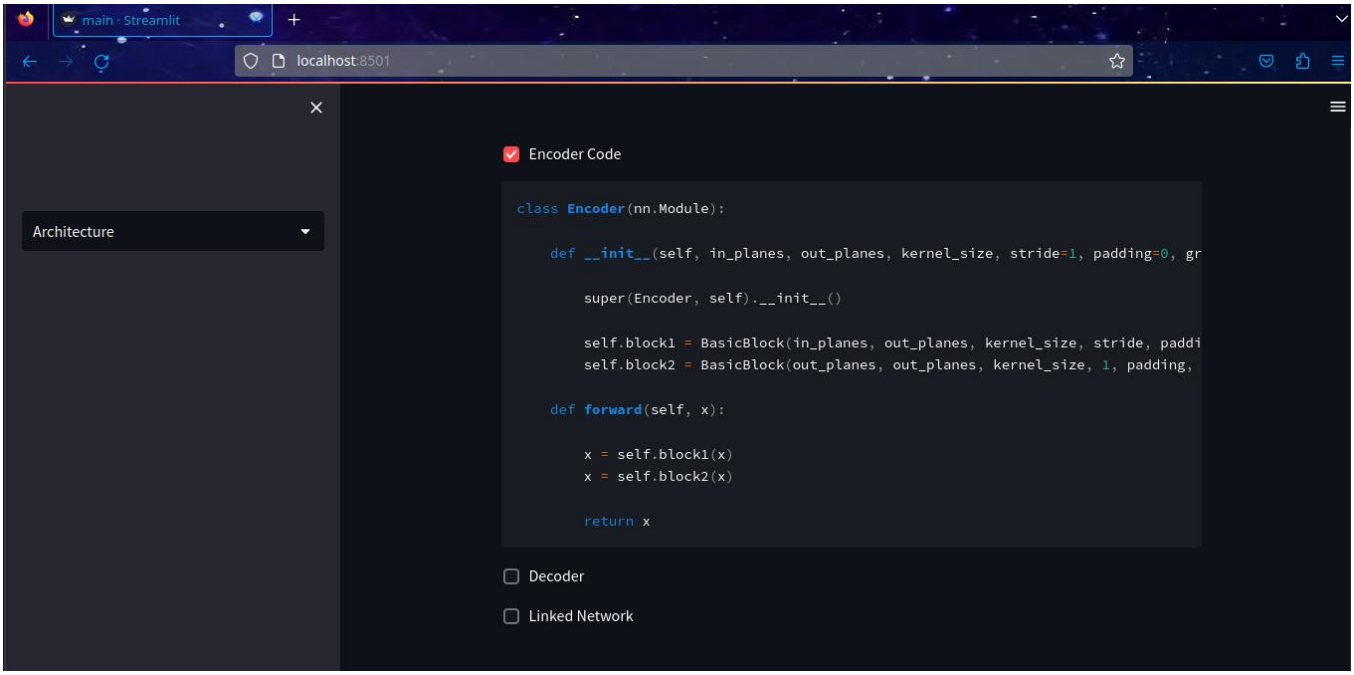


Fig A 2.37: Encoder Code

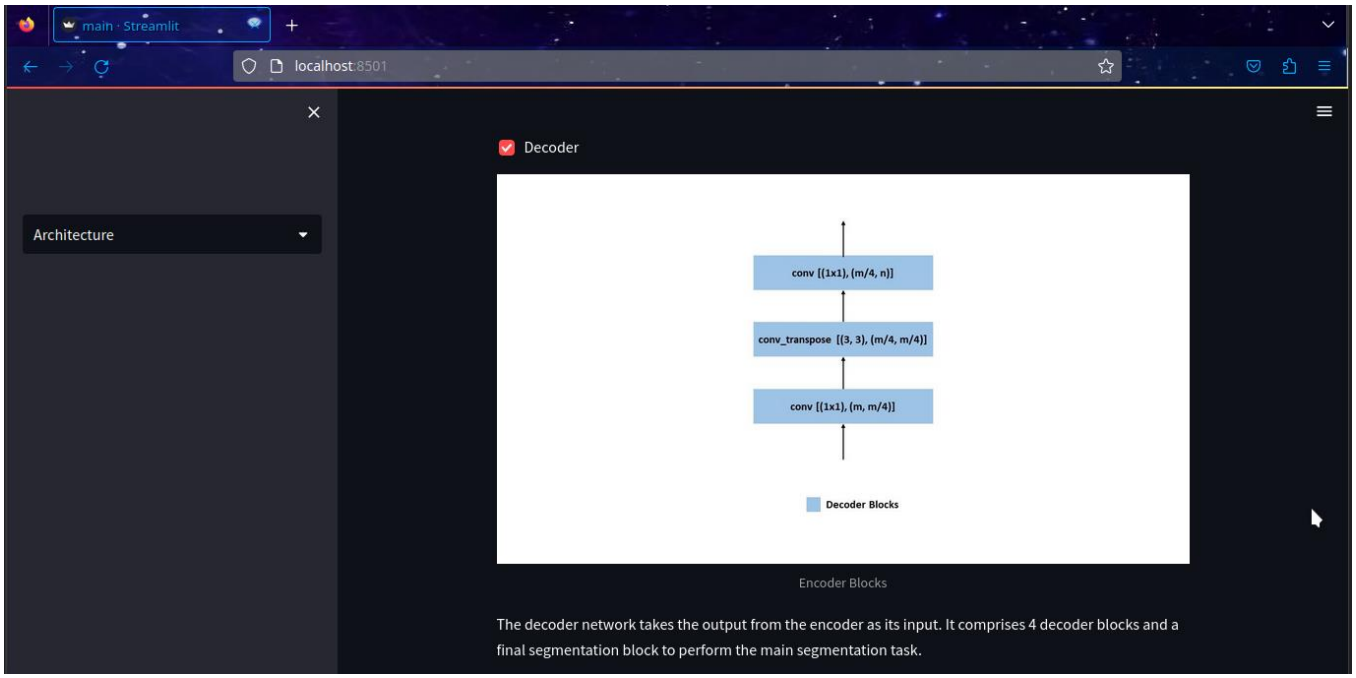


Fig A 2.38: Decoder

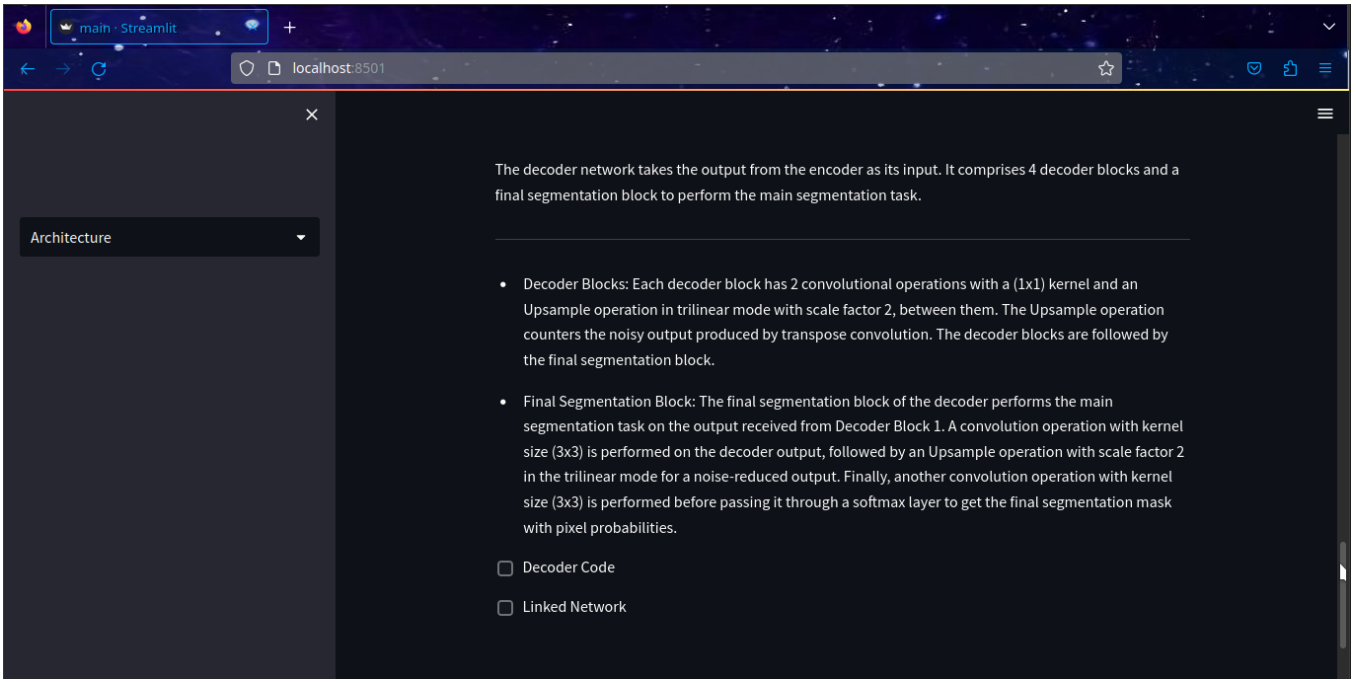


Fig A 2.39: Decoder Theory

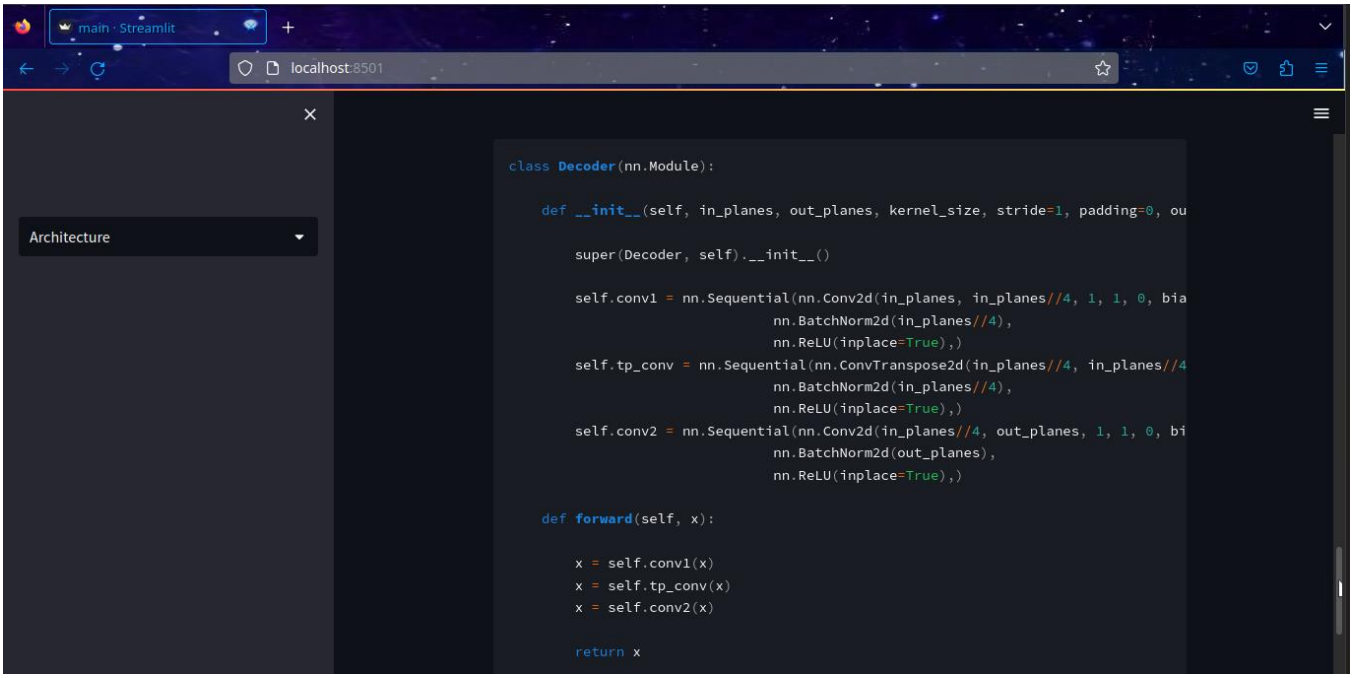


Fig A 2.40: Decoder Code

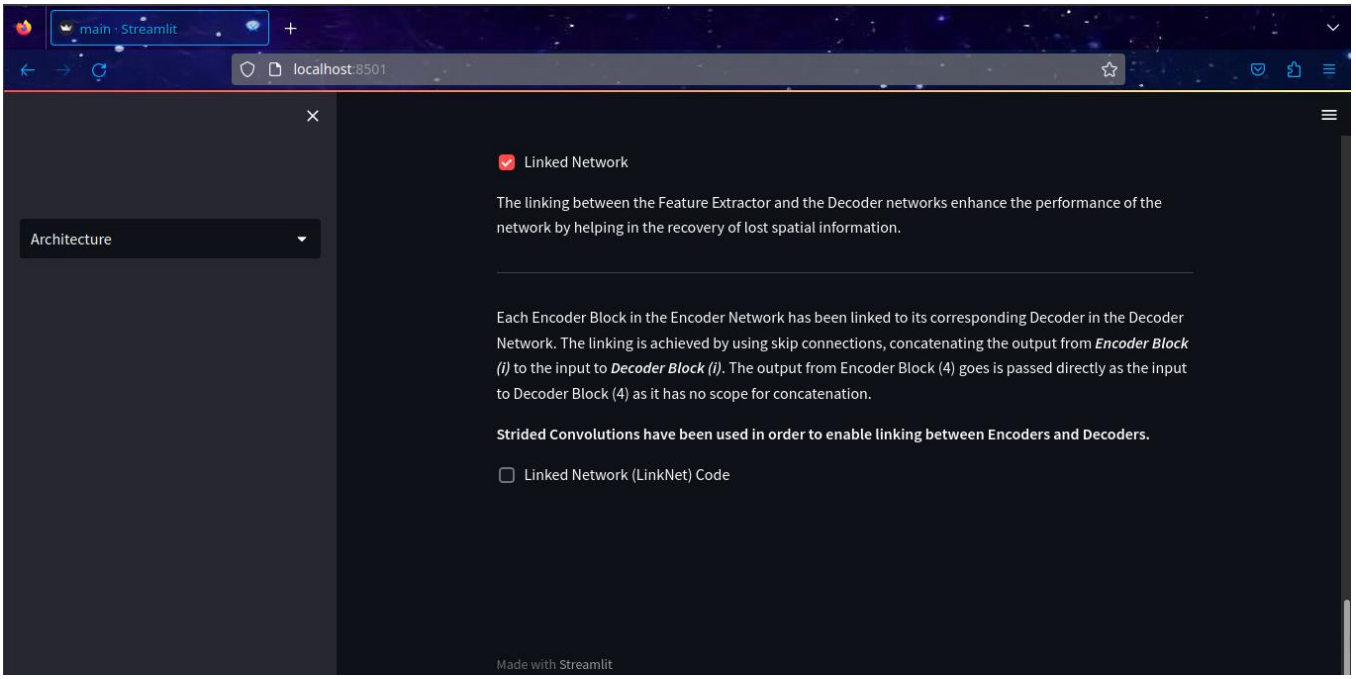


Fig A 2.41: Linked Network

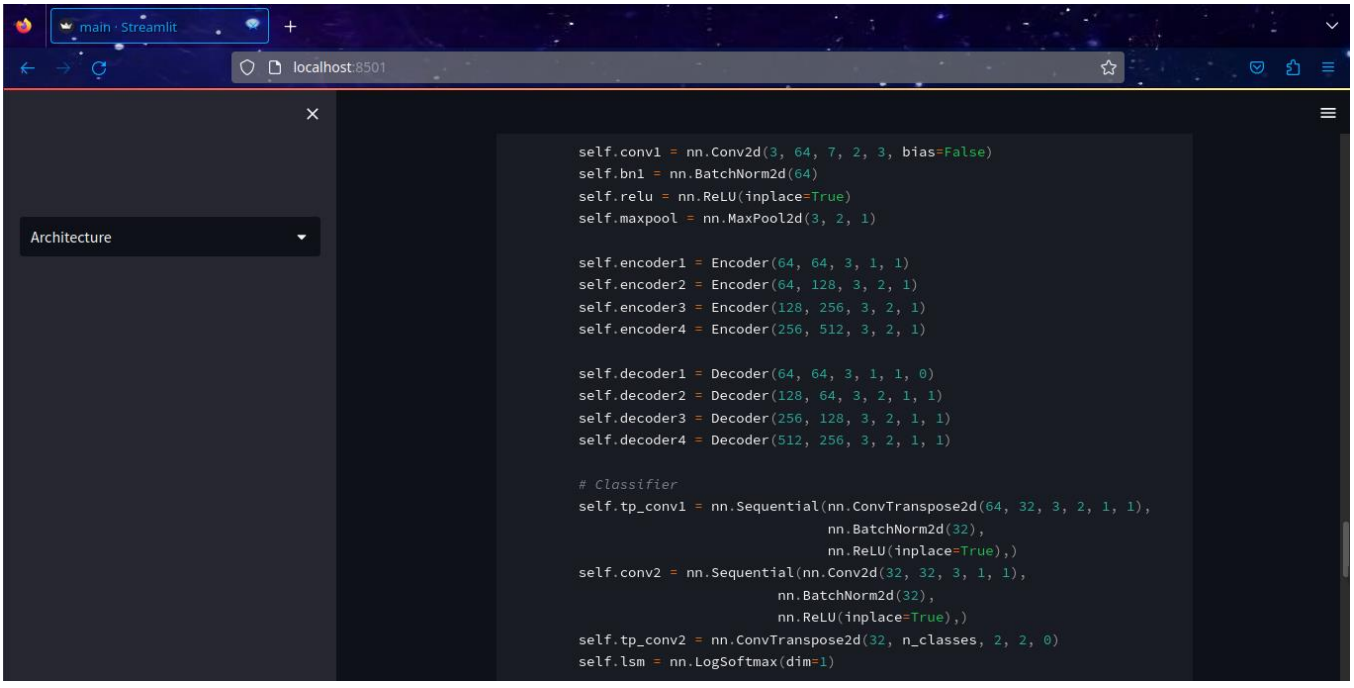


Fig A 2.42: Linked Network Code – 1

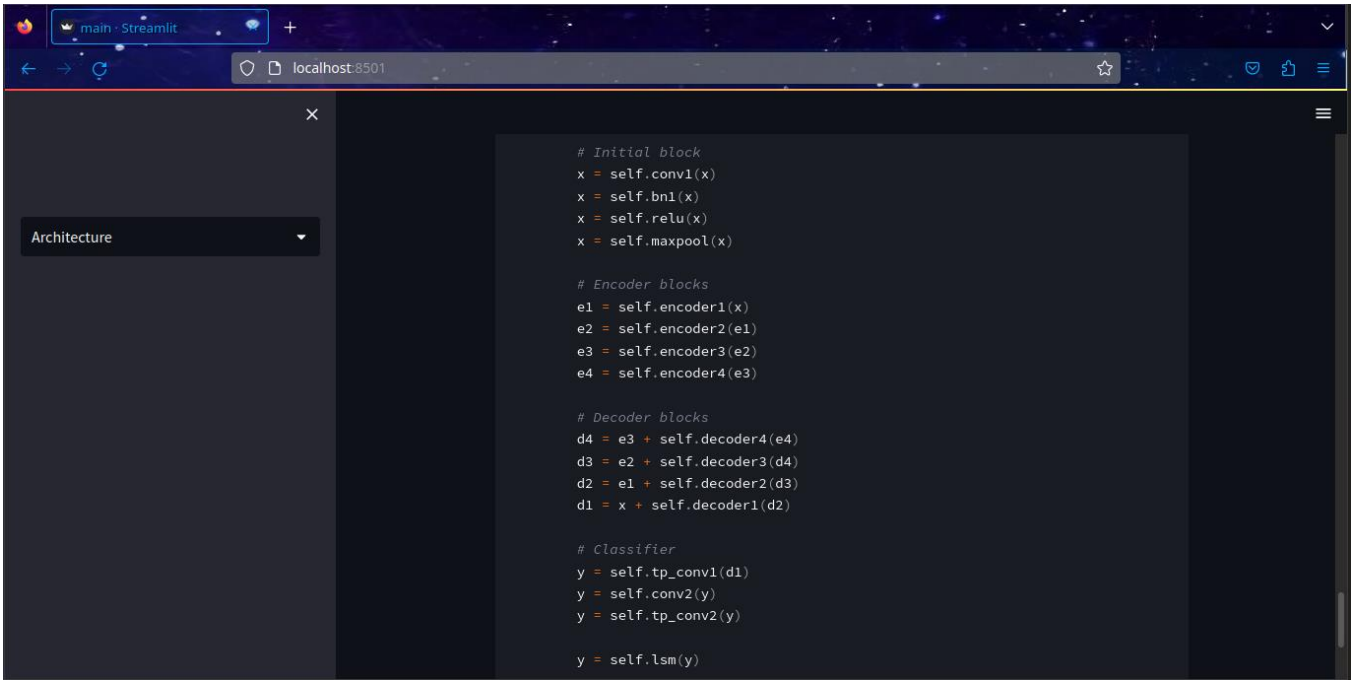


Fig A 2.43: Linked Network Code – 2

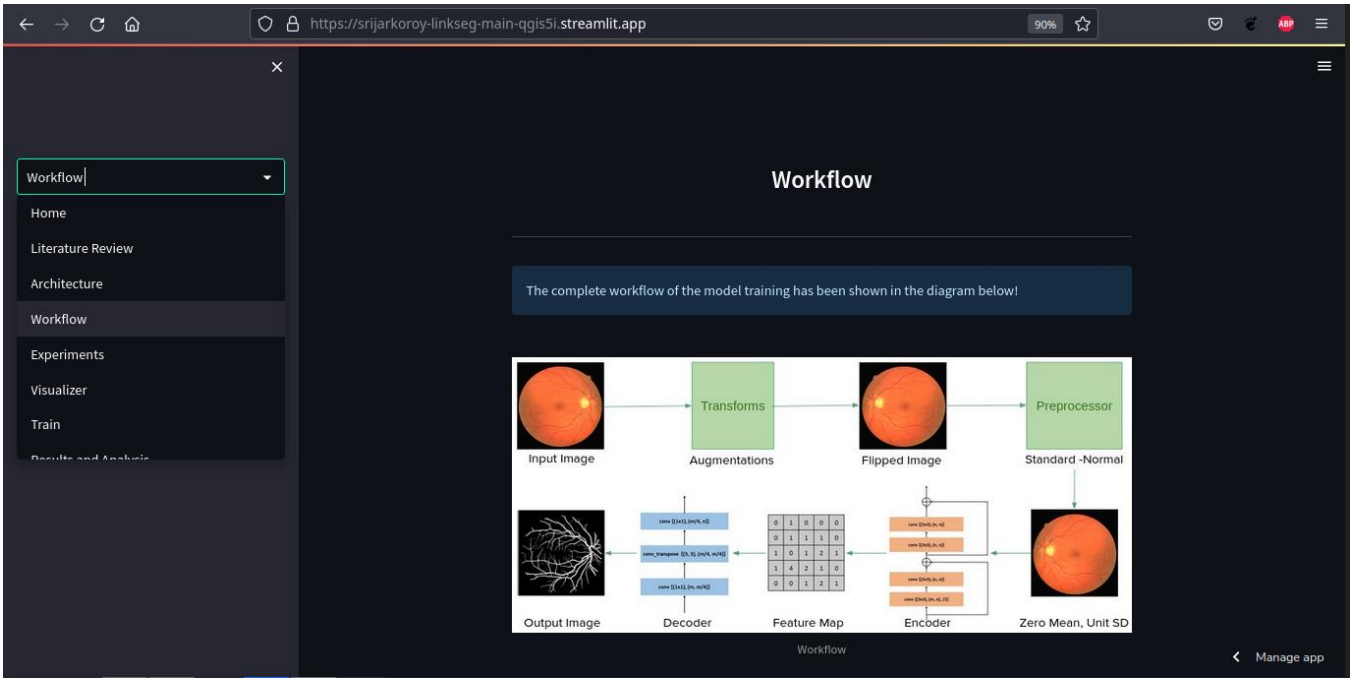


Fig A 2.44: Workflow 1

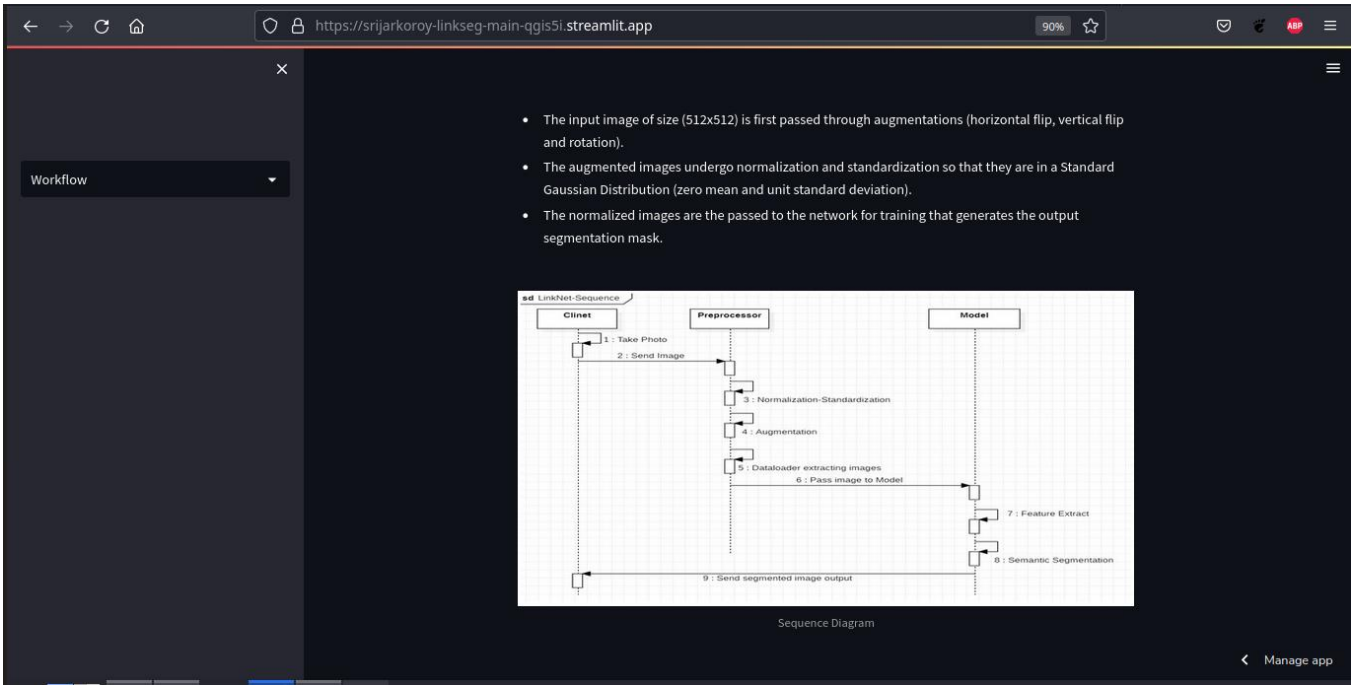


Fig A 2.45: Workflow 2

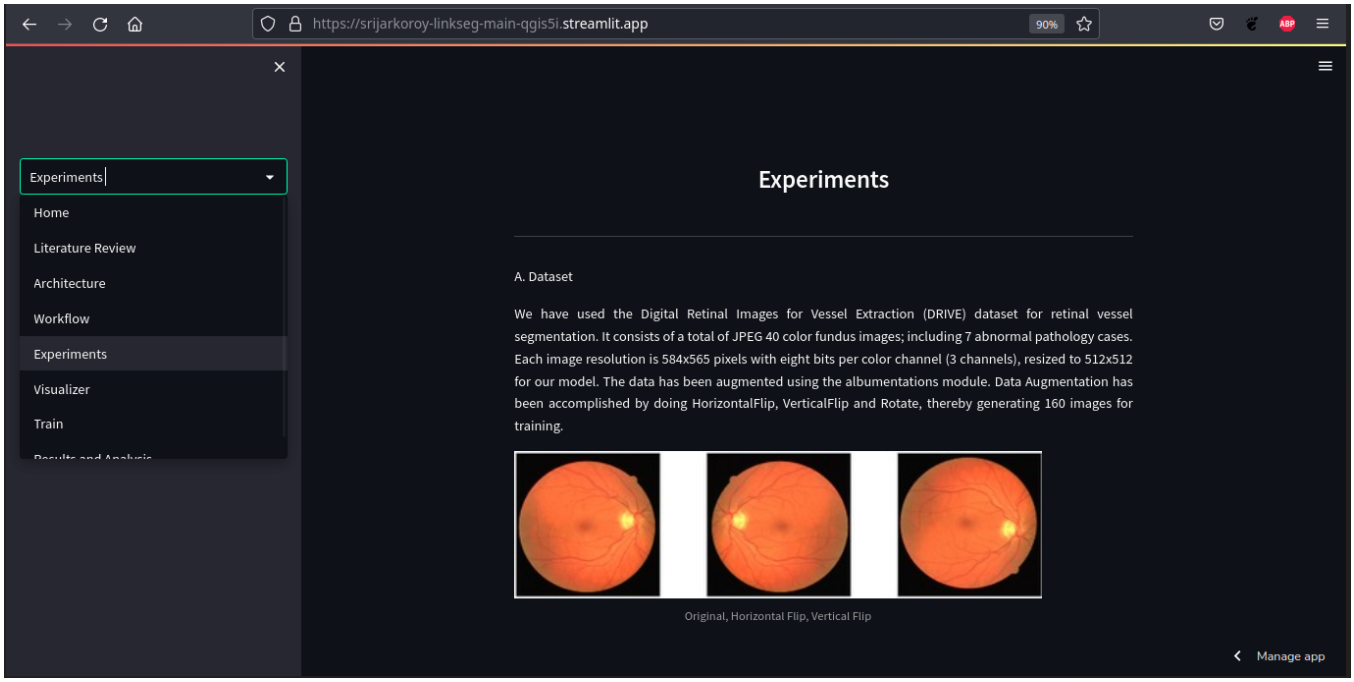


Fig A 2.46: Dataset

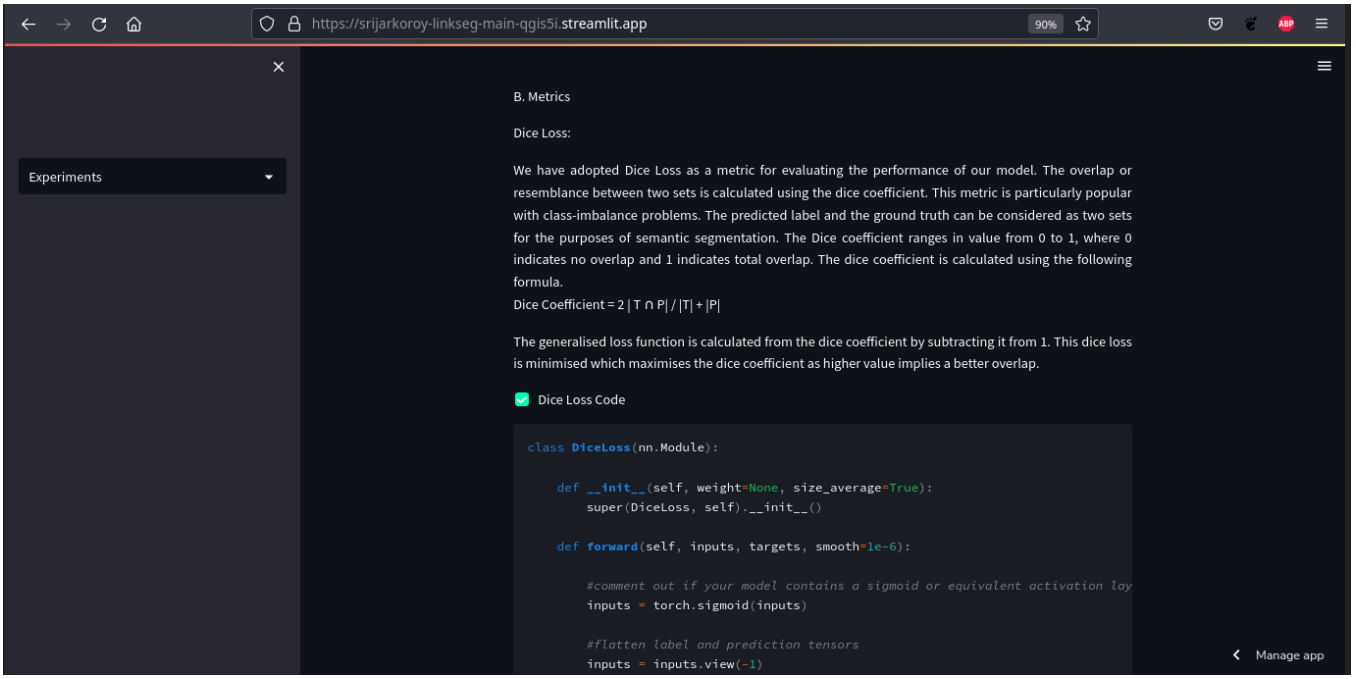


Fig A 2.47: Dice Score Theory

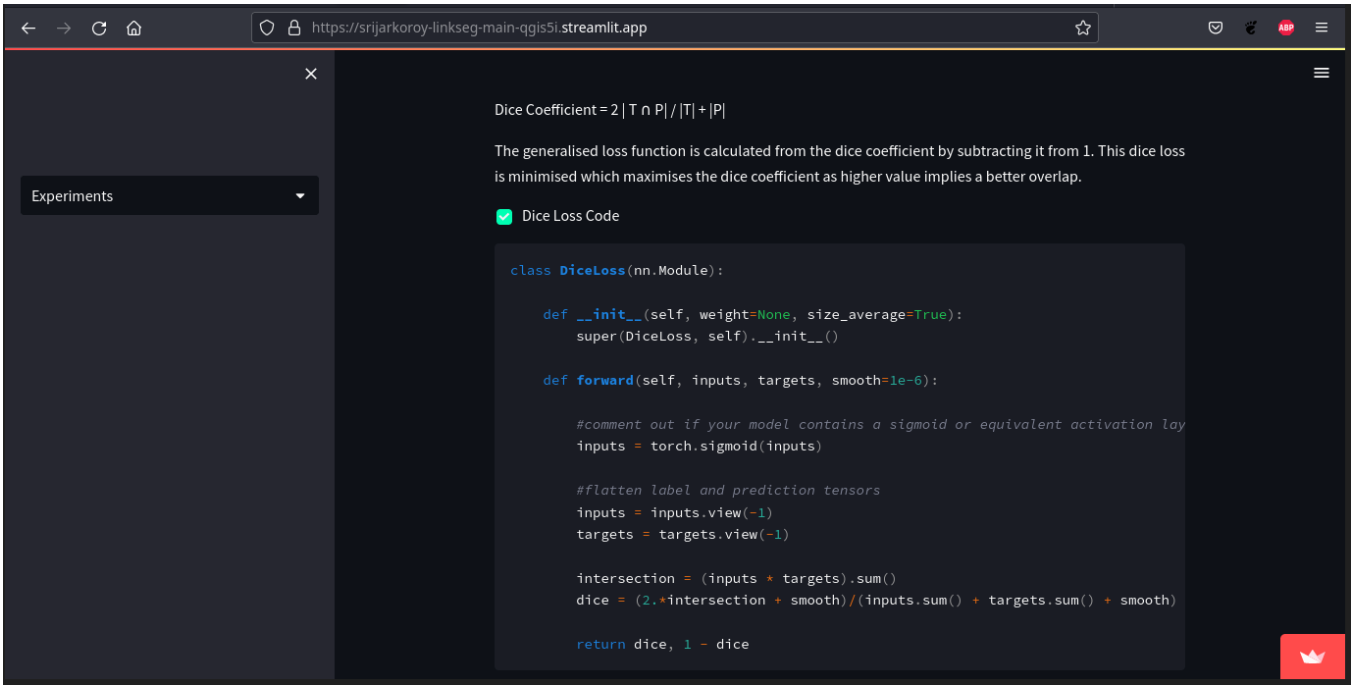


Fig A 2.48: Dice Score Code

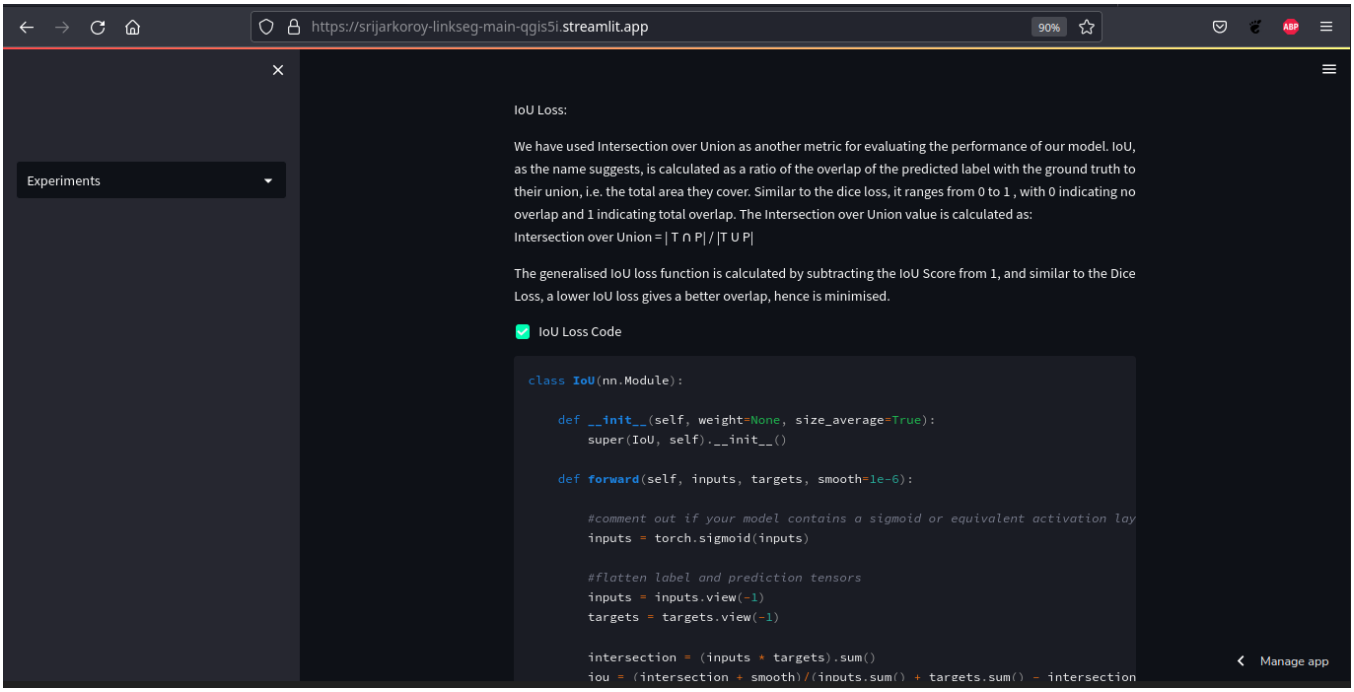


Fig A 2.49: IoU Theory

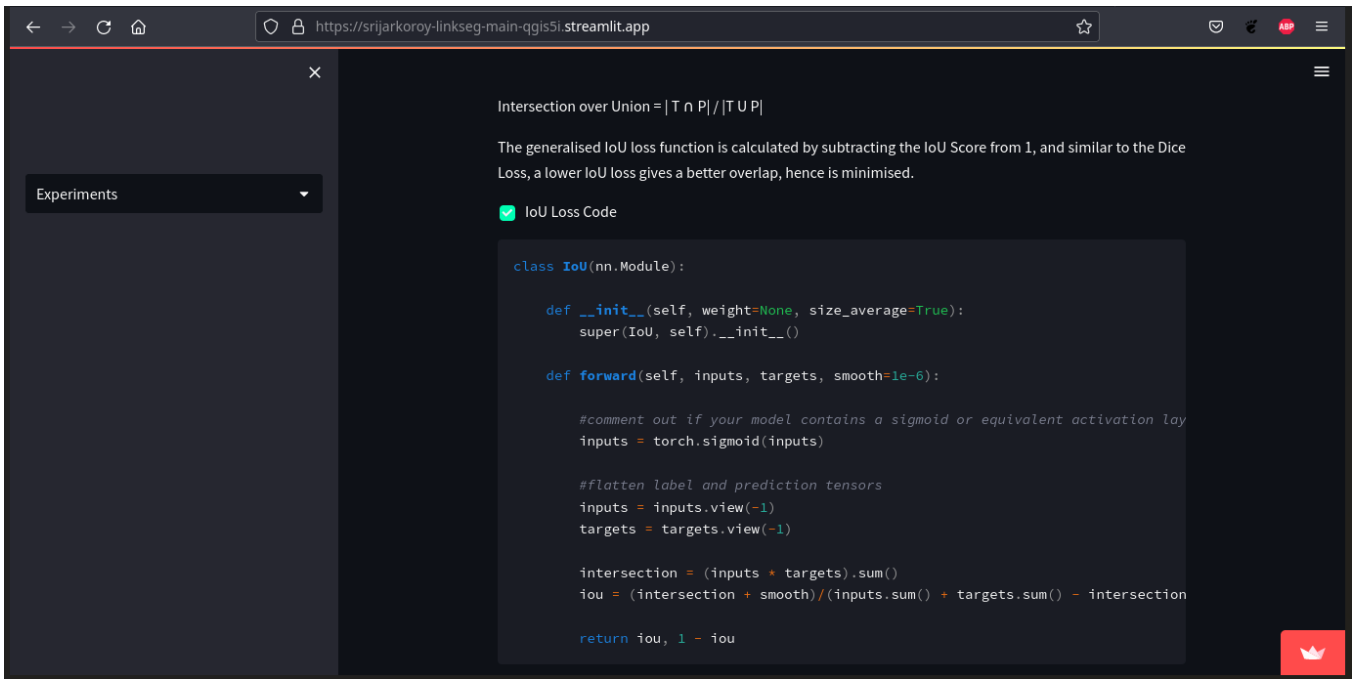


Fig A 2.50: IoU Code

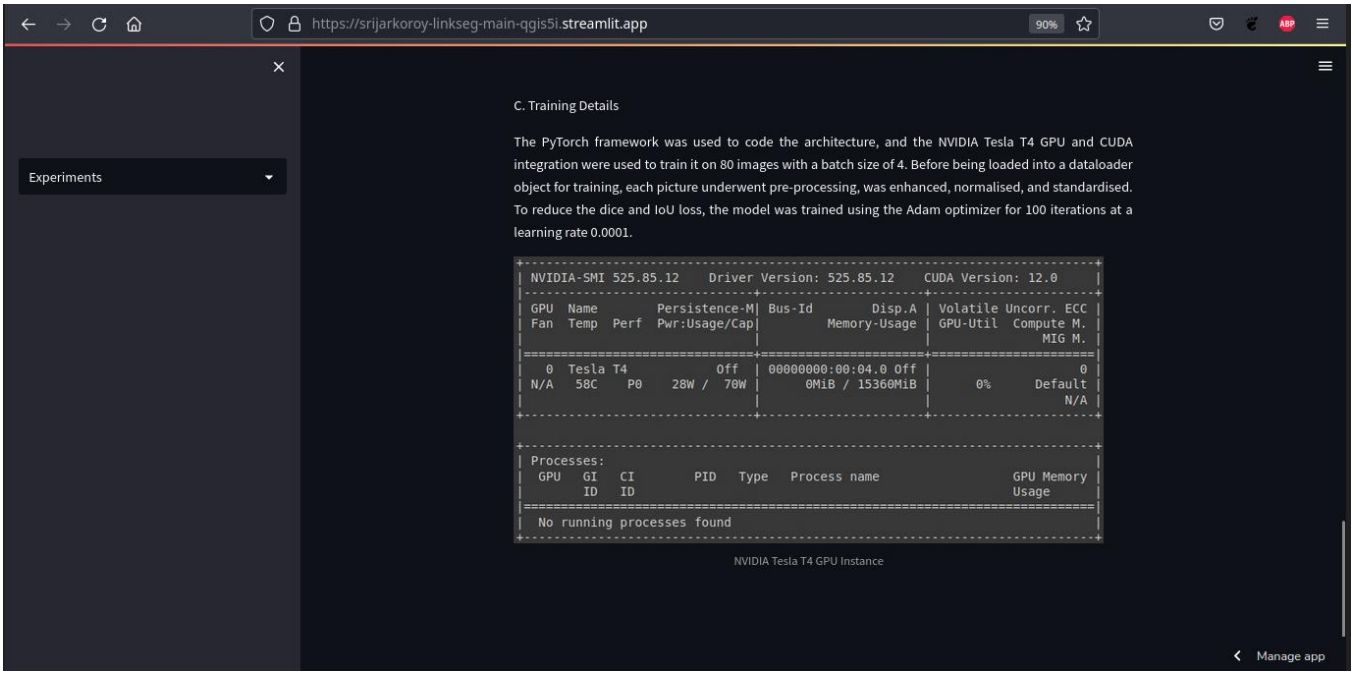


Fig A 2.51: Training Details

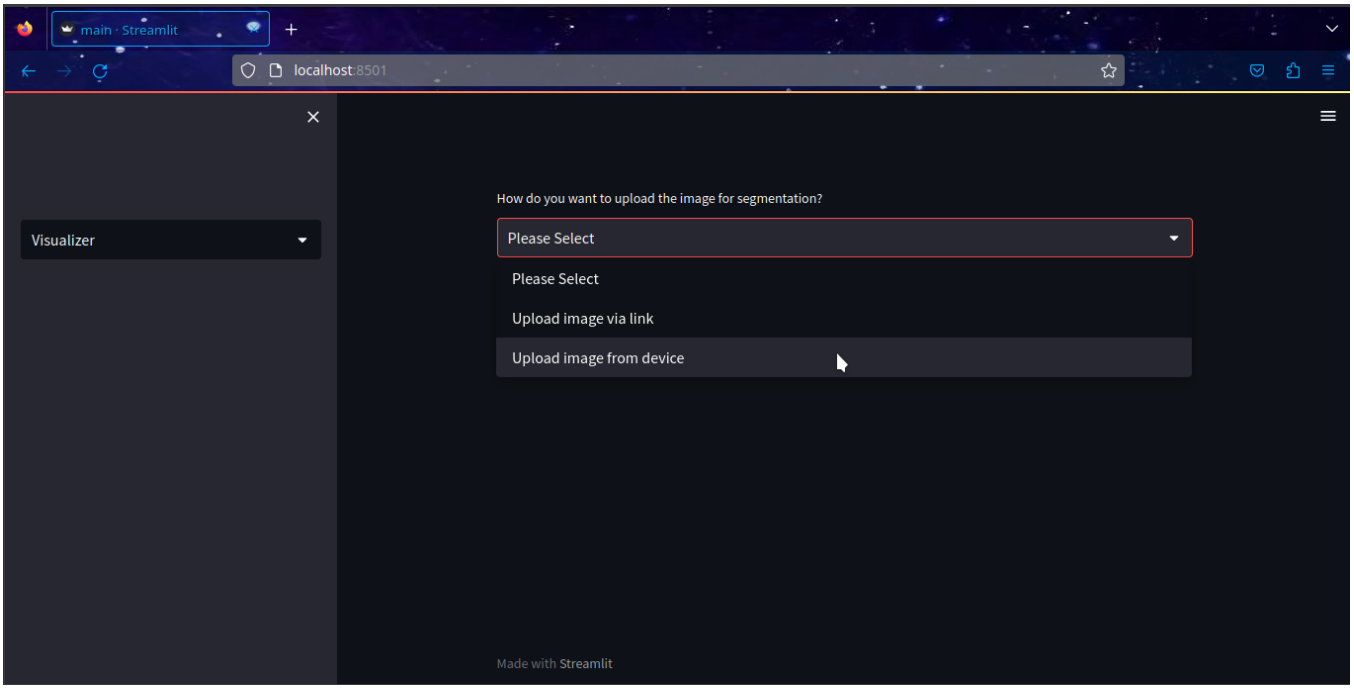


Fig A 2.52: Visualizer

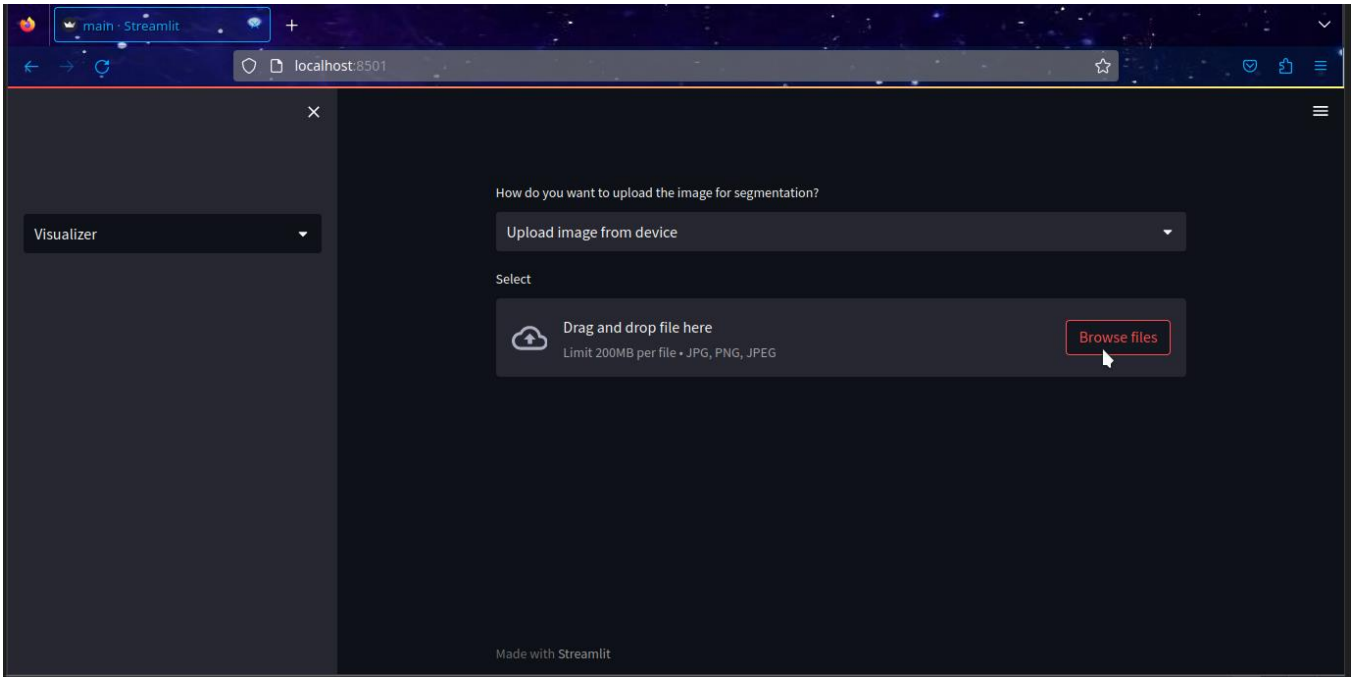


Fig A 2.53: Development

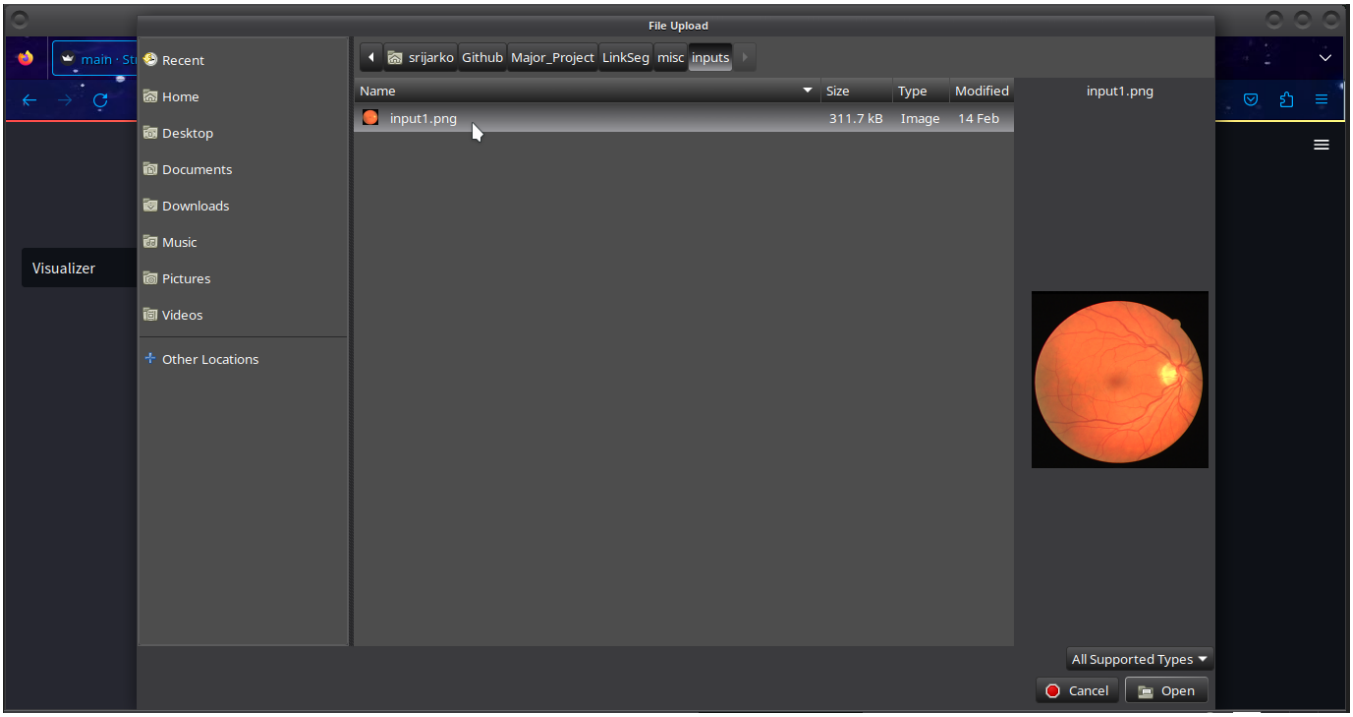


Fig A 2.54: Image upload

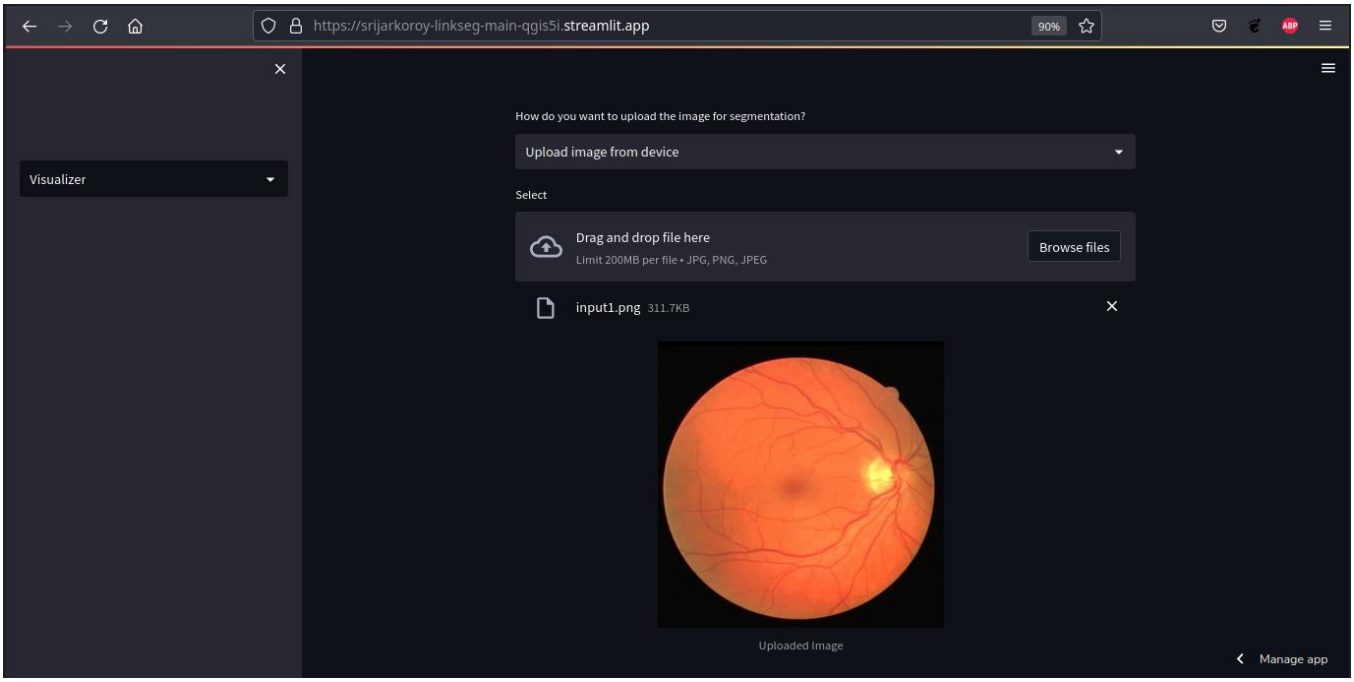


Fig A 2.55: Image uploaded.

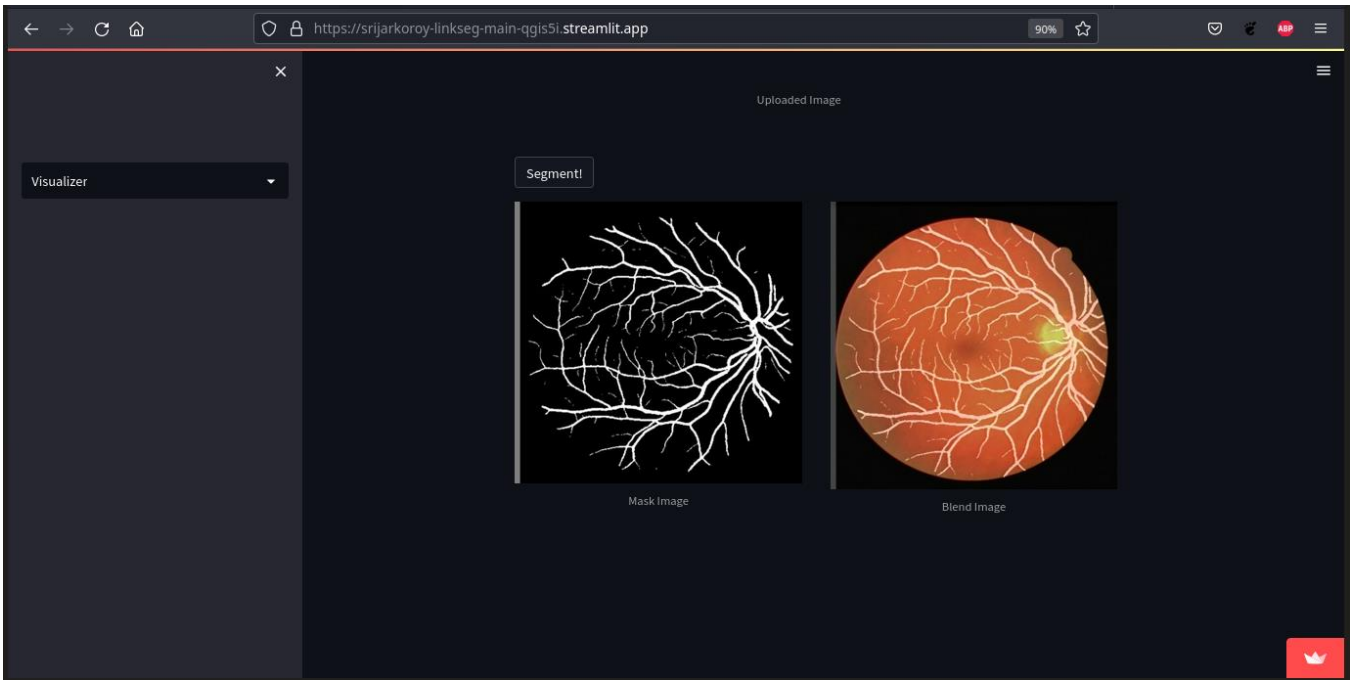


Fig A 2.56: Image segmented.

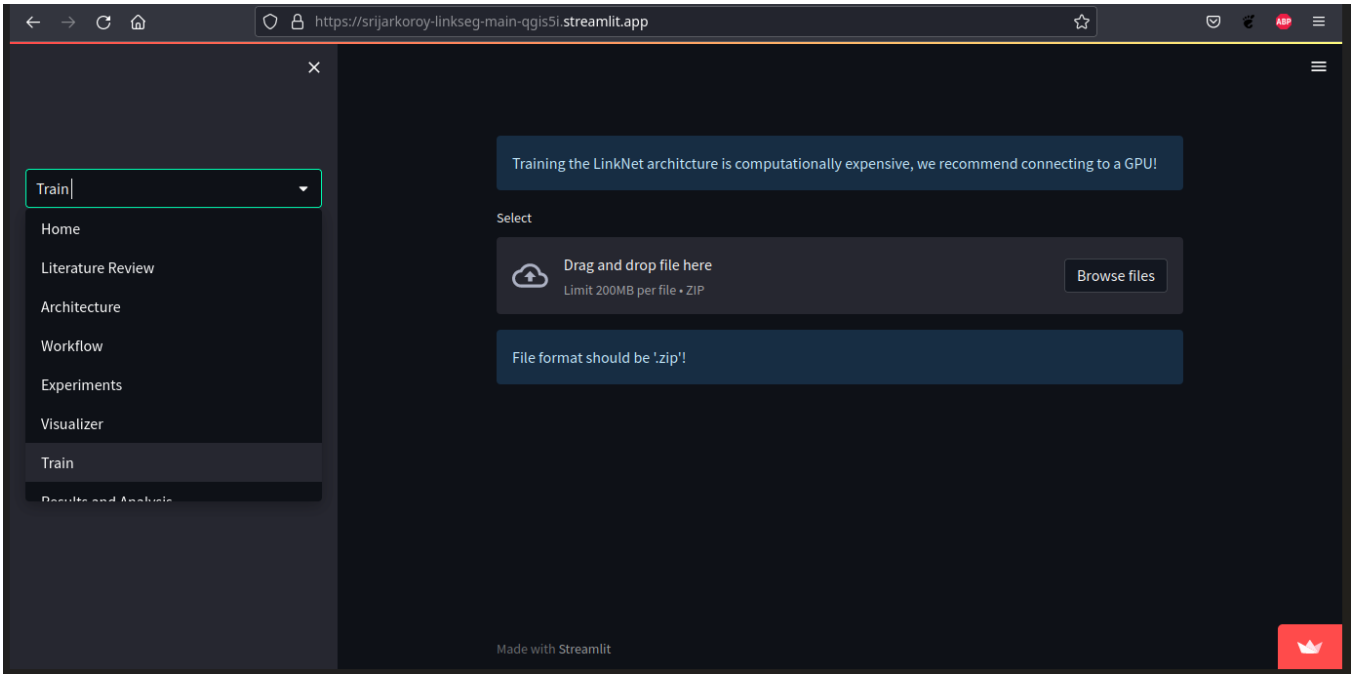


Fig A 2.57: Training Module in UI

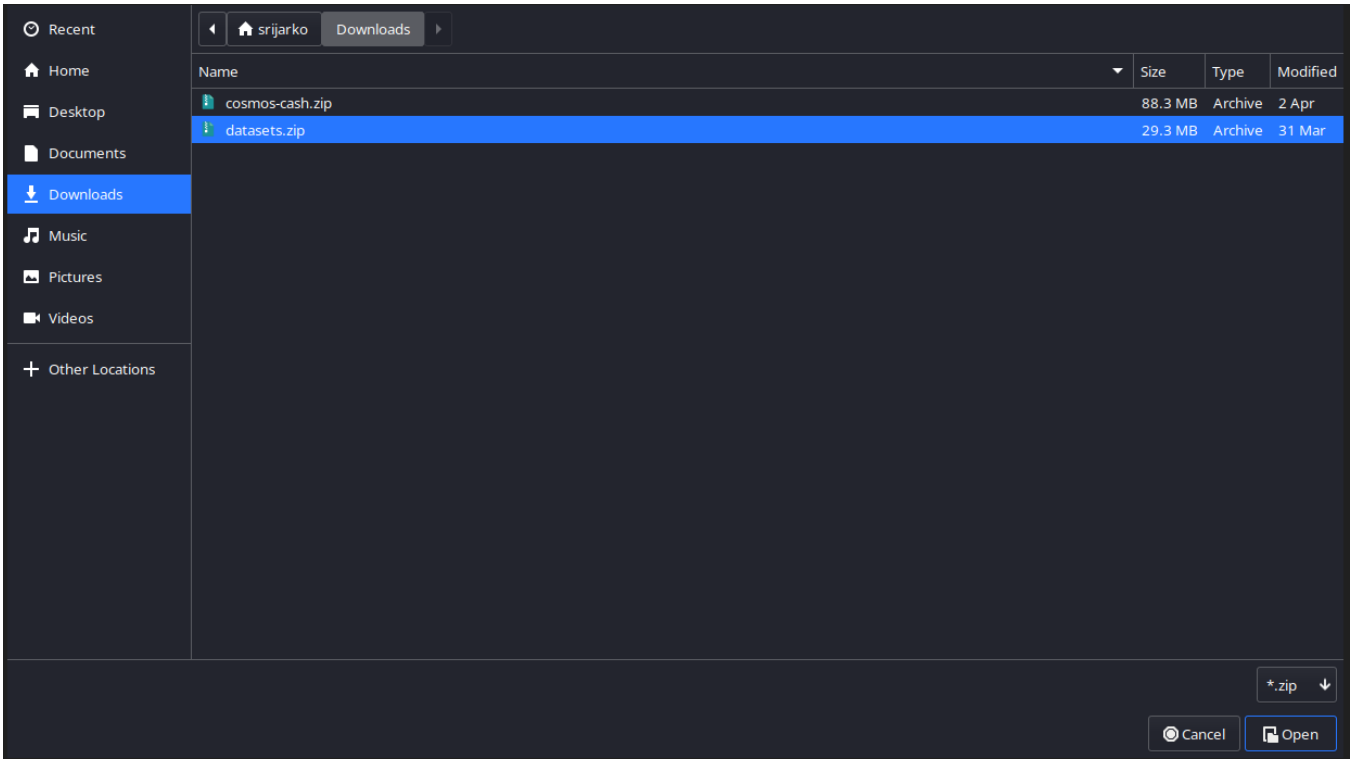


Fig A 2.58: Dataset Upload

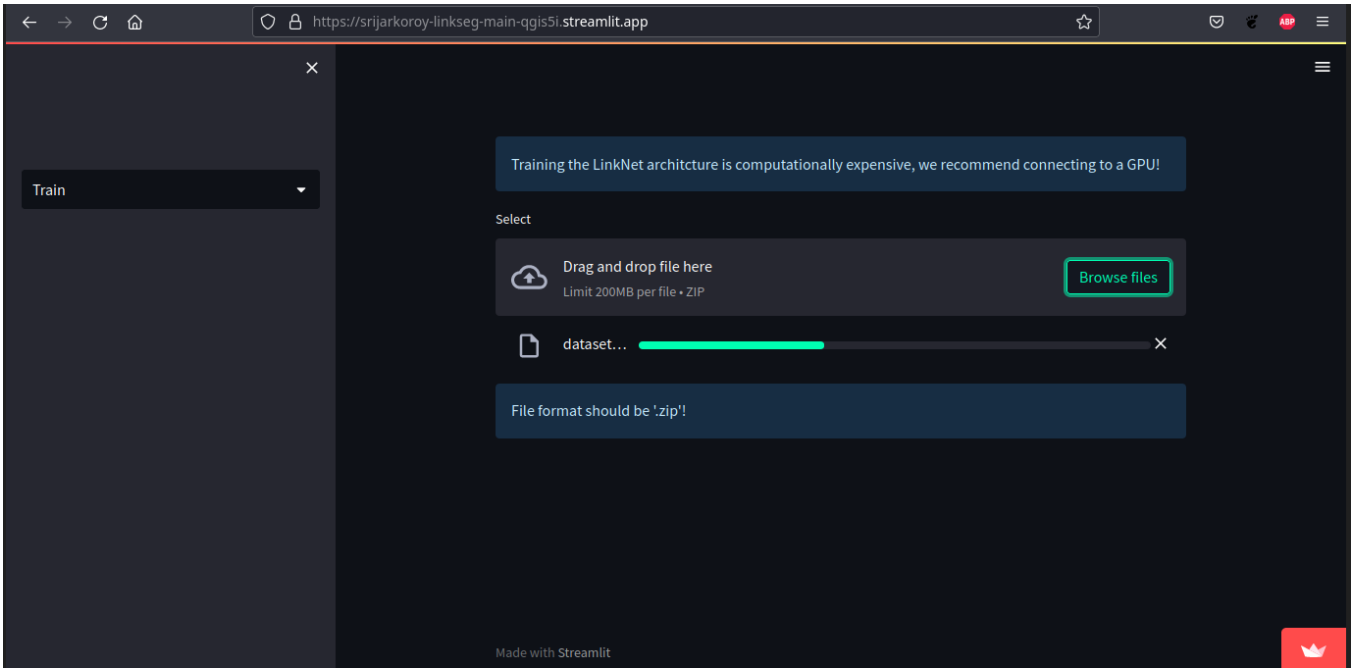


Fig A 2.59: Dataset Uploading

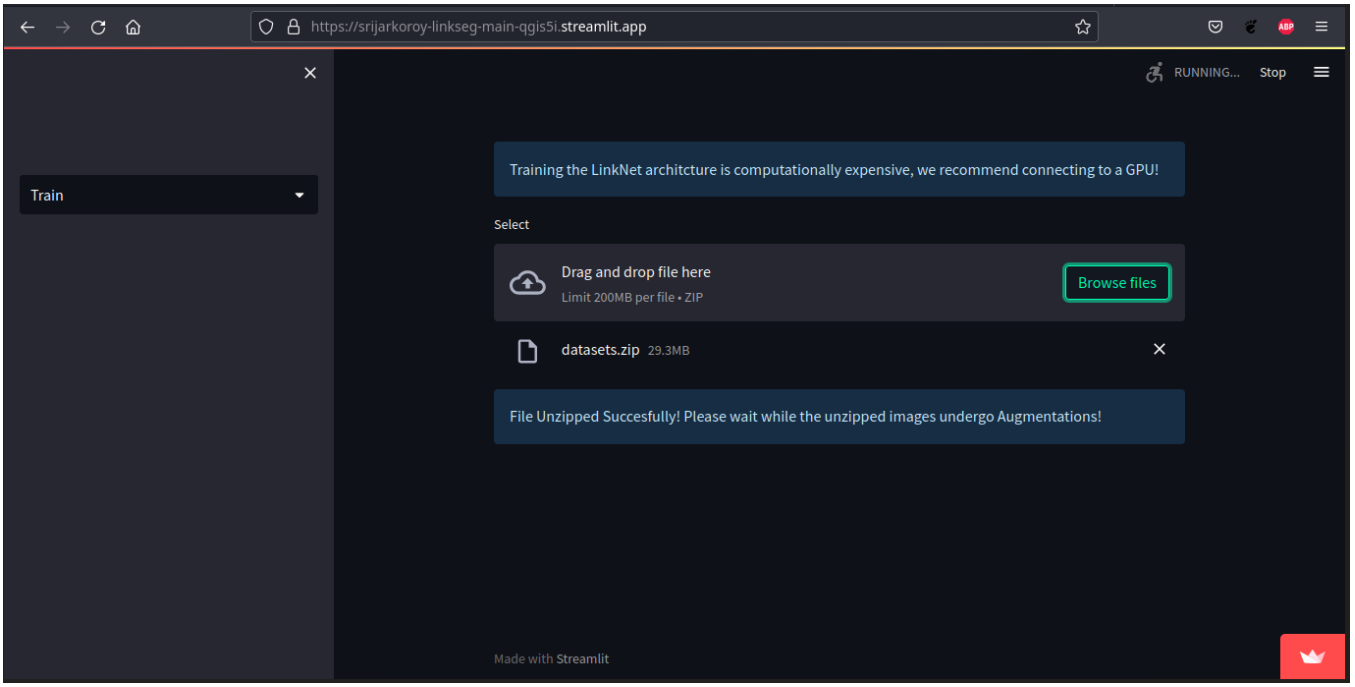


Fig A 2.60: Dataset Uploaded

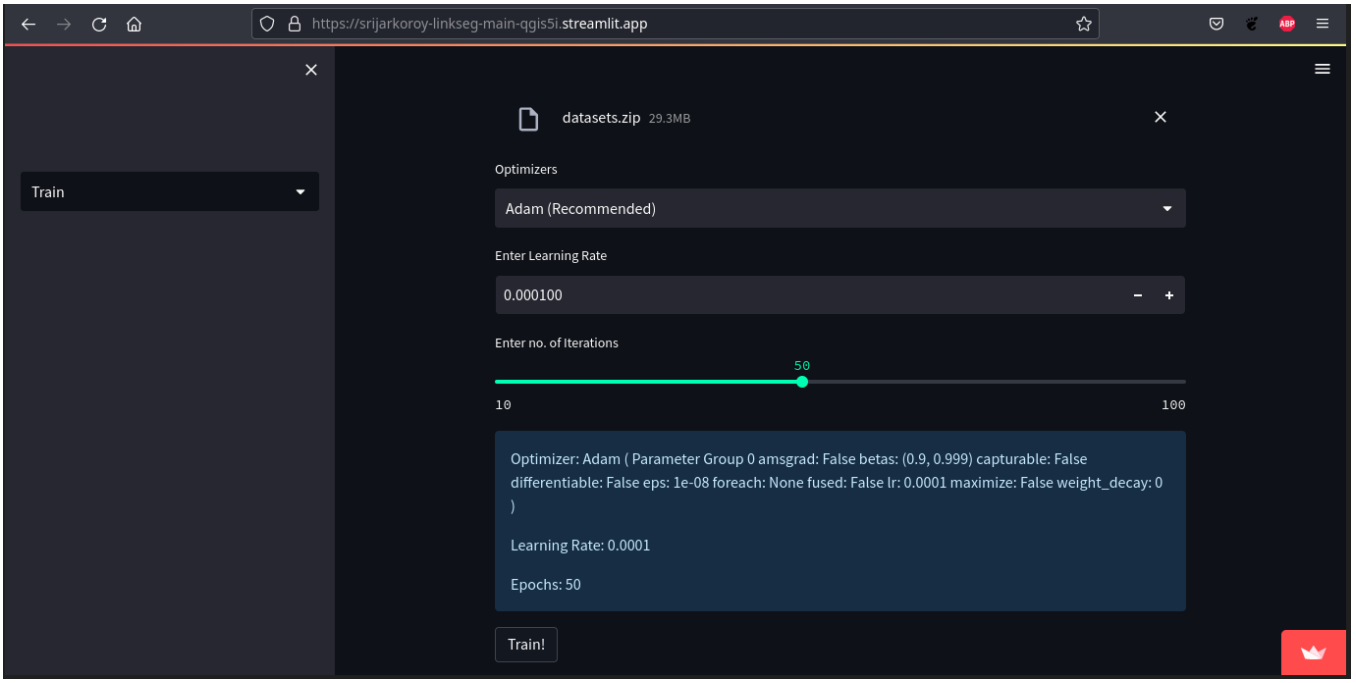


Fig A 2.61: Training Hyperparametes

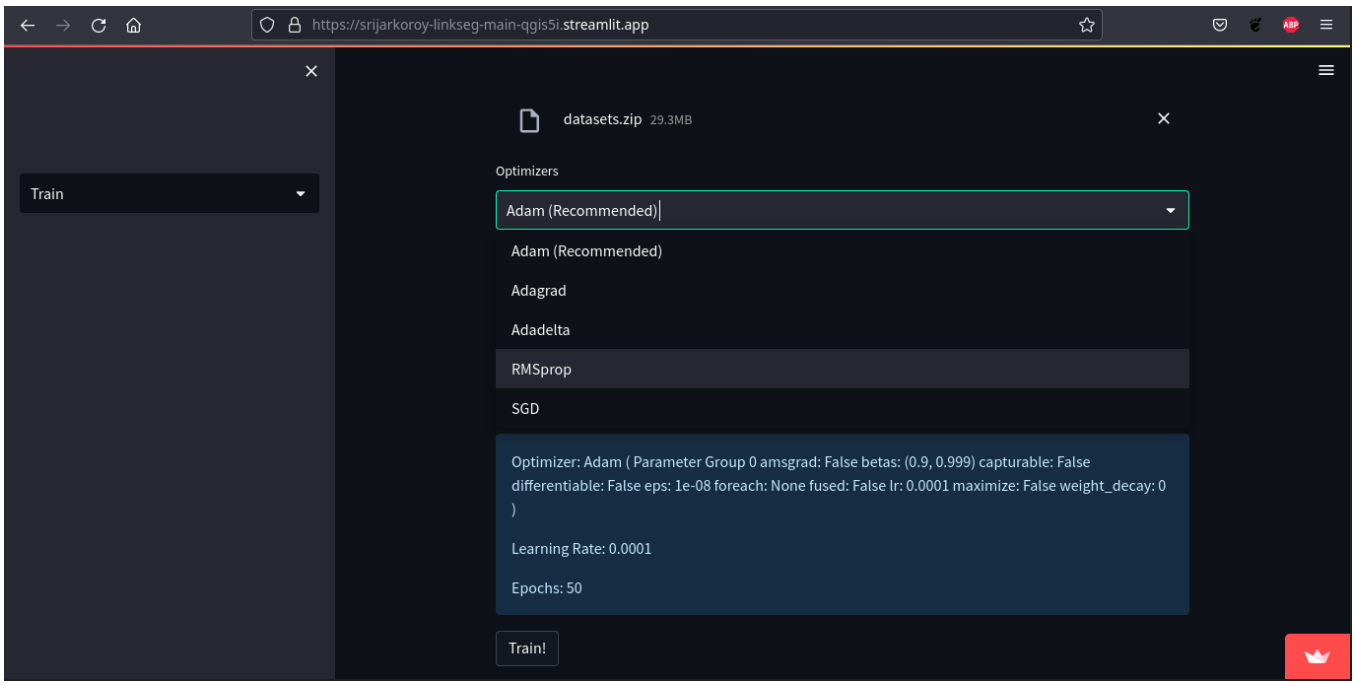


Fig A 2.62: Optimizer

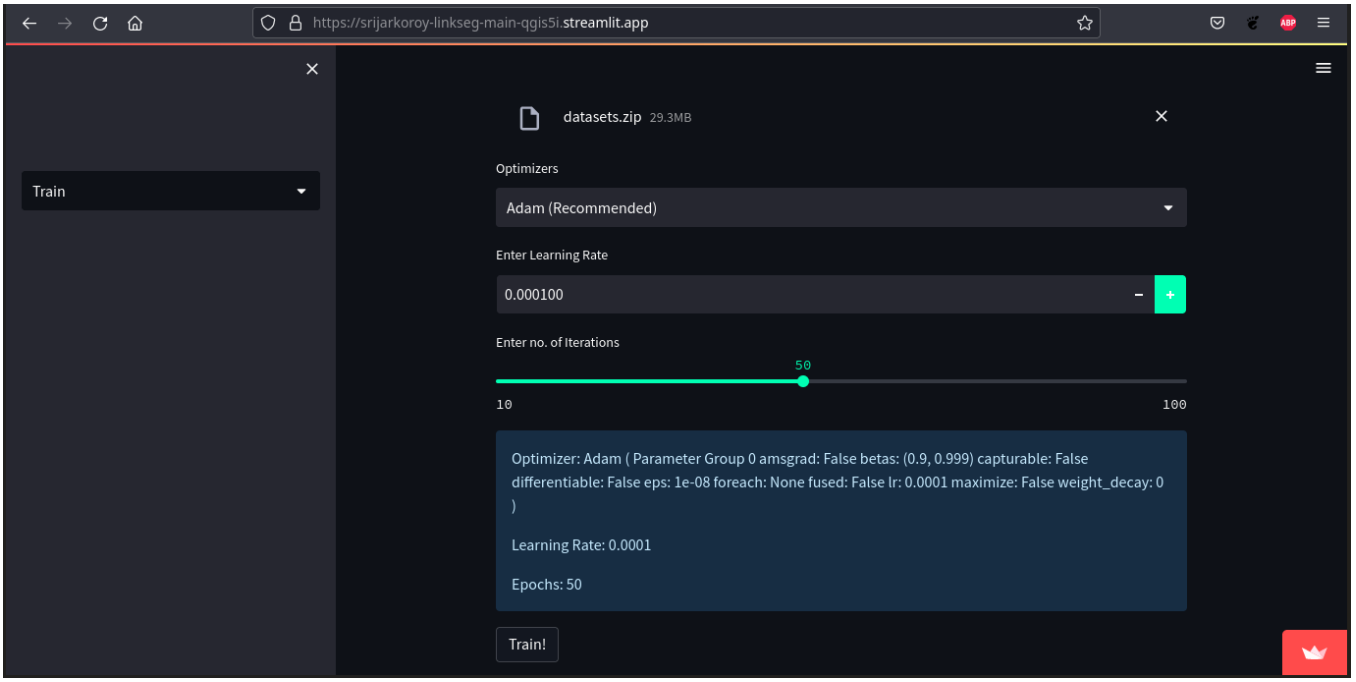


Fig A 2.63: Learning Rate

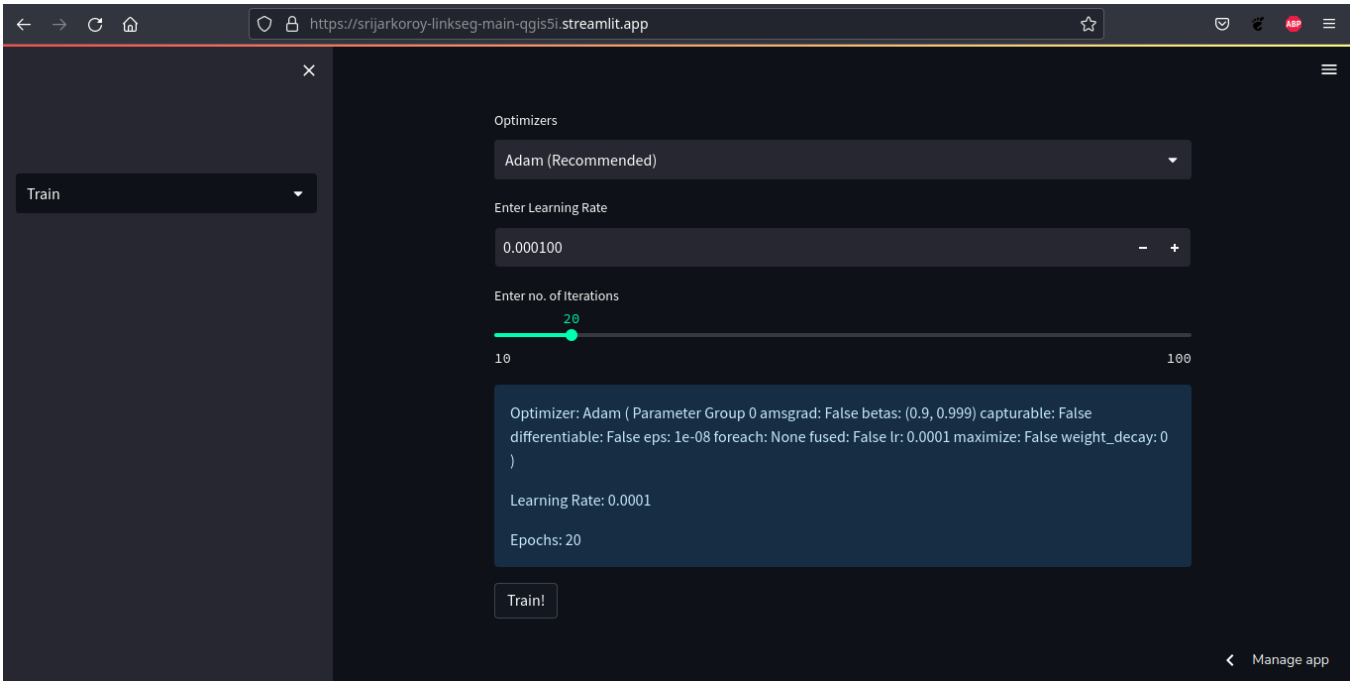


Fig A 2.64: Epochs

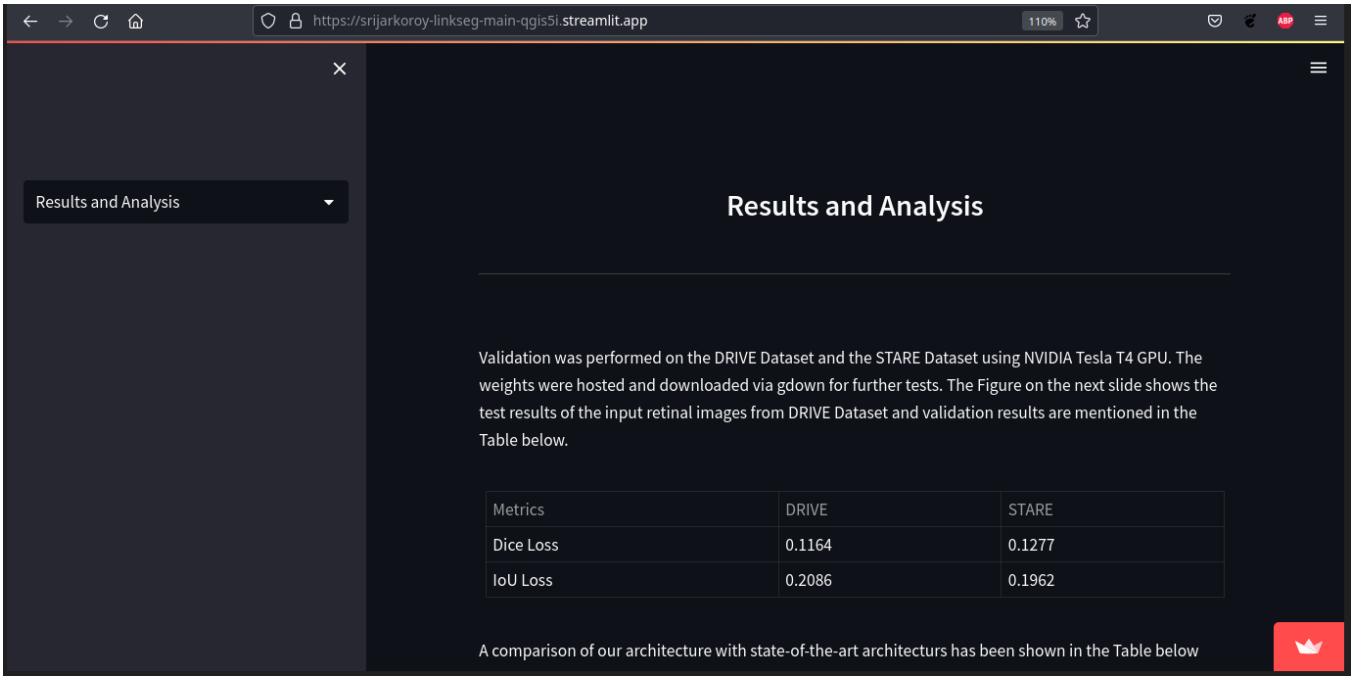


Fig A 2.65: Result Analysis

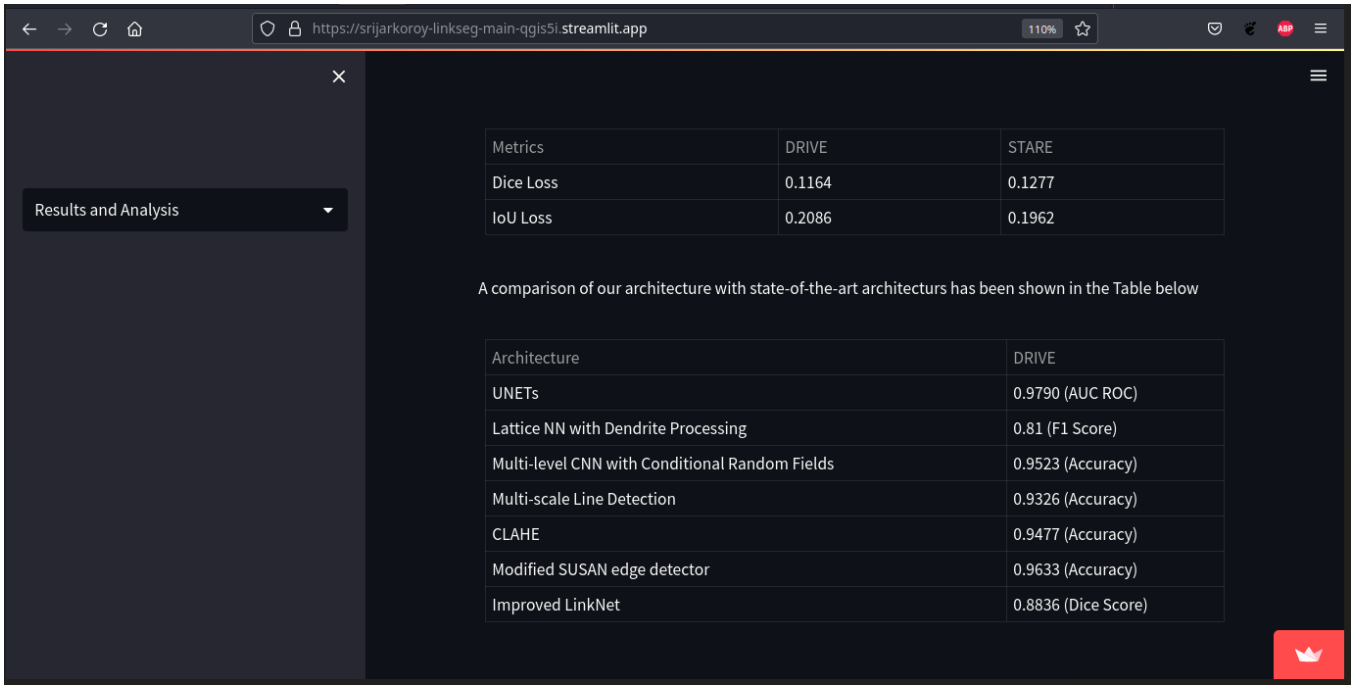


Fig A 2.66: SOTA

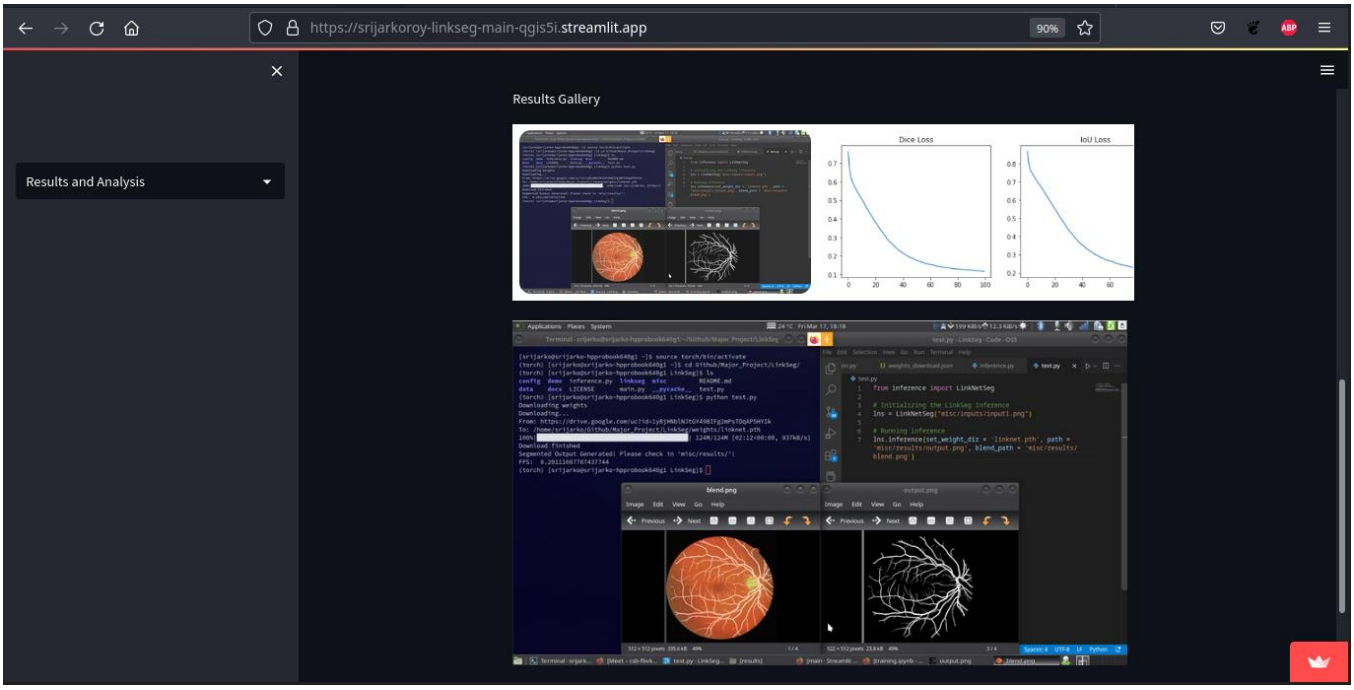


Fig A 2.67: Gallery

Fig A 2.68: Gallery – 2

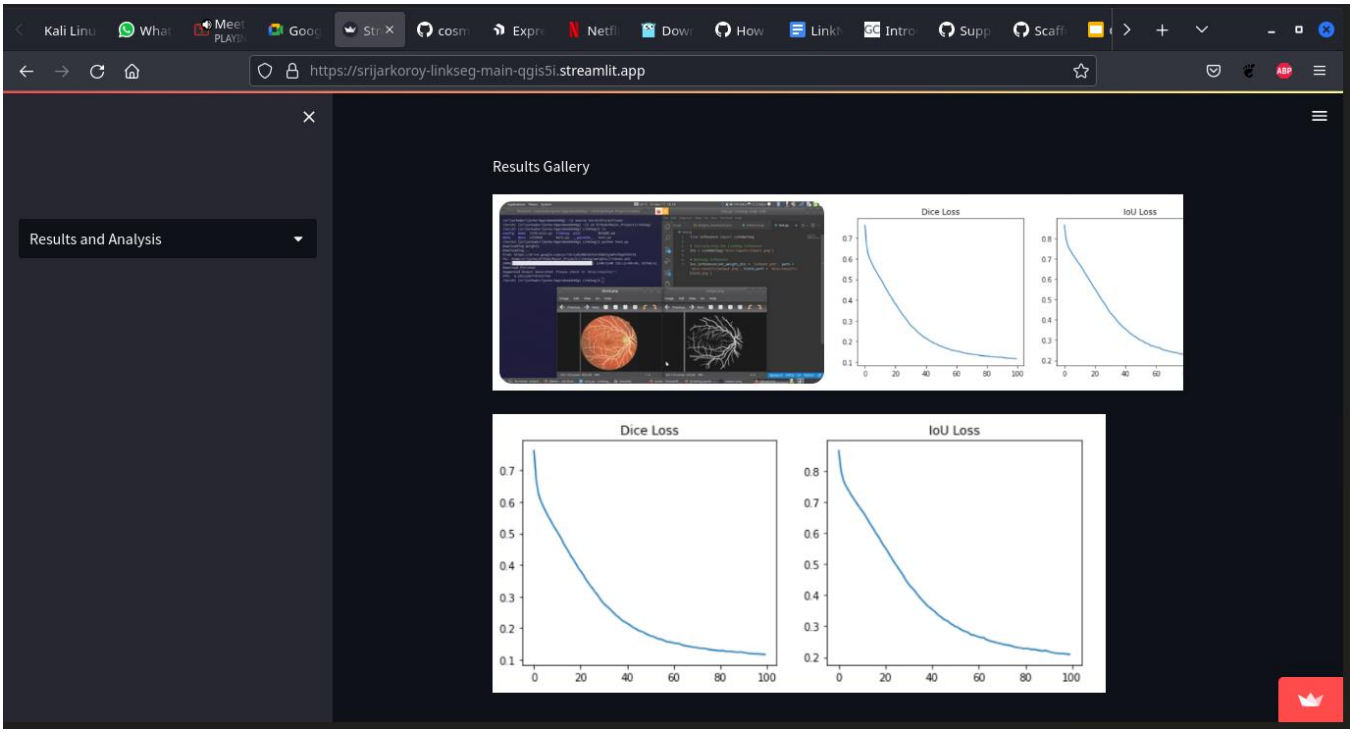


Fig A 2.69: Gallery – 2

PAPER PUBLICATION STATUS

Acceptance Notification - IEEE 3rd CONIT 2023 External Inbox ×

Microsoft CMT <email@msr-cmt.org> Fri, Apr 21, 8:08 PM
to me

Dear Ankit Mathur

Paper ID / Submission ID : 412

Title : An improved LinkNet-based approach for Retinal Image Segmentation

Greeting from3rd CONIT 2023

We are pleased to inform you that your paper has been accepted for the Oral Presentation and publication as a full paper for the- "IEEE 2023 3rd International Conference for Intelligent Technologies (CONIT), Hubballi, Karnataka, India with following reviewers' comment.

All accepted and presented papers will be submitted to IEEE Xplore for the further publication.

Note:
All of Accepted and Presented Papers of CONIT series has been Published by IEEE Xplore and indexed by Scopus and other Reputed Indexing partners of IEEE. - http://inconf.in/index.php/publications/

You should finish the registration before deadline, or you will be deemed to withdraw your paper:

Complete the Registration Process (The last date of payment Registration is 26 APRIL 2023)

Payment Links

IEEE 3rd CONIT 2023 - IEEE PDF eXpress and E Copyright Information (Activated) Inbox ×

Microsoft CMT <email@msr-cmt.org> Thu, 27 Apr, 14:37 (8 days ago)
to me

Dear Ankit Mathur

Paper / Submission ID :412

Thank you for registering your paper, we received your payment. E copyright is activated kindly submit that also.

The link of to submit camera ready and E-copyright is available on your CMT login.

PDF Instructions for your Authors

2023 3rd International Conference on Intelligent Technologies (CONIT)

₹8,800.00

Paid Successfully

Payment Id	pay_Lhf6rkk1IX00pA
Method	UPI ankitmathur34@okhdfcbank
Paid On	24th Apr, 2023 20:00:18 PM IST
Email	ankitmathur34@gmail.com
Mobile Number	+917063528293

PLAGIARISM REPORT

LinkNet Report 087_088

ORIGINALITY REPORT

7%	3%	3%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Berlin School of Business and Innovation Student Paper	1%
2	www.researchgate.net Internet Source	1%
3	Submitted to Houston Community College Student Paper	1%
4	Submitted to Post Graduate Institute of Medical Education and Research Student Paper	<1%
5	Zhenwei Li, Mengli Jia, Xiaoli Yang, Mengying Xu. "Blood Vessel Segmentation of Retinal	<1%