COMP9900: InfoTech Project Report

P14 - Real Time Retinal Vessels Segmenter

Team: Superlu

Jeremy Lu(Scrum Master): z5536953
Email: z5536953@ad.unsw.edu.au

Chengming Hu(Developers): z5533821
Email: z5533821@ad.unsw.edu.au

Ziyang Wang(Project owner): z5531376
Email: z5531376@ad.unsw.edu.au

Zhendong Zhao(Developers): z5542437
Email: z5542437@ad.unsw.edu.au

Cong Du(Developers): z5504217
Email: z5504217@ad.unsw.edu.au

Yiyu Chen(Developers): z5581750
Email: z5581750@ad.unsw.edu.au

Submission date: 16 Nov 2024

Contents

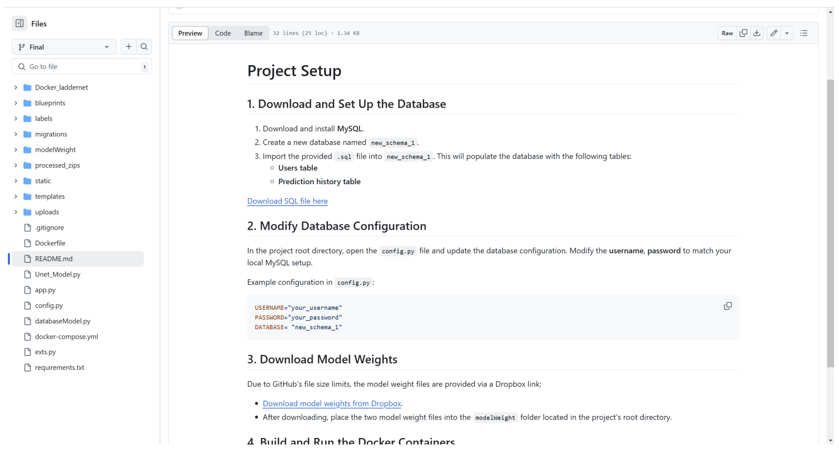
1 Installation Manual

Project Repository:
The project has been uploaded to GitHub. You can find the repository at the following link:

- **GitHub Repository:** Git Link

README File:
The detailed installation manual is available in the `README.md` file located in the root directory of the repository.
Required Files:
You will need to download the necessary files to set up the project. These files are available at the following link:

- **Download Files:** Dropbox Link



Project Setup

1. Download and Set Up the Database

1. Download and install **MySQL**.
2. Create a new database named `new_schema_1`.
3. Import the provided `.sql` file into `new_schema_1`. This will populate the database with the following tables:
   - **Users table**
   - **Prediction history table**

Download SQL file here

2. Modify Database Configuration

In the project root directory, open the `config.py` file and update the database configuration. Modify the **username**, **password** to match your local MySQL setup.

Example configuration in `config.py`:

```
USERNAME="your_username"
PASSWORD="your_password"
DATABASE= "new_schema_1"
```

3. Download Model Weights

Due to GitHub's file size limits, the model weight files are provided via a Dropbox link:

- Download model weights from Dropbox.
- After downloading, place the two model weight files into the `modelWeight` folder located in the project's root directory.

4. Build and Run the Docker Containers

Figure 1: README

Below is a brief overview of the installation steps for the project:

1. **Install the Database:** Install MySQL and create a new database named `new_schema_1`. Import the provided `.sql` file into this database.

2. **Modify Configuration File:** In the project root directory, open the `config.py` file and update the username and password fields to match your local MySQL configuration.

3. **Download Model Weights:** Download two model weight files from the provided link and place them in the `modelWeight` folder in the project root directory.

4. **Run Docker Commands:** Navigate to the project root directory and execute the following commands to build and start the application using Docker:

```
docker-compose build
docker-compose up
```

2 System Architecture Diagram

2.1 Project Technology

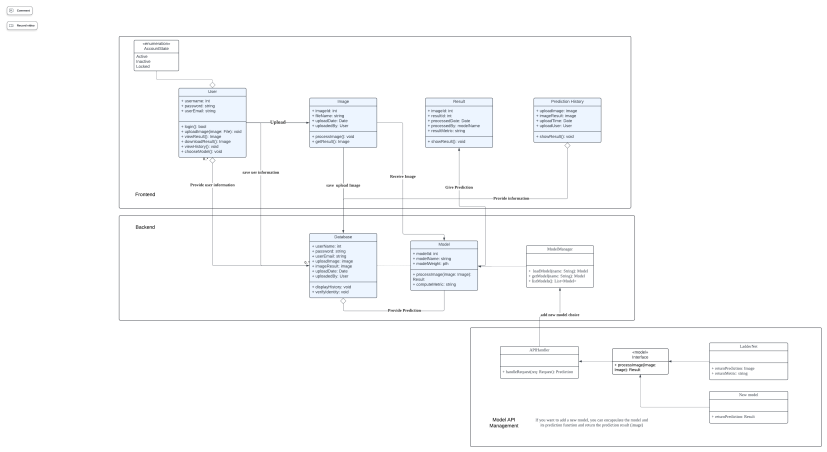


Figure 2: UML Class figure

We use Flask as the backend framework to build our web application, leveraging Flask-SQLAlchemy and Flask-Migrate for database interaction and migration.
For the database, we adopt MySQL, which contains two tables: **User** (for user registration and login) and **History** (to record user-uploaded files and model prediction results). We use pymysql for database connection and SQLAlchemy as the ORM.

In terms of deep learning models, we utilize mature libraries such as PyTorch and torchvision. We have also developed our custom model, AggreUNet. Additionally, we leverage Docker to provide an endpoint for inference using a new model, Laddernet, via its URL.
The frontend is implemented using HTML, handling image and ZIP file uploads and providing the processed files for download.

2.2 Module Interaction and Information Flow

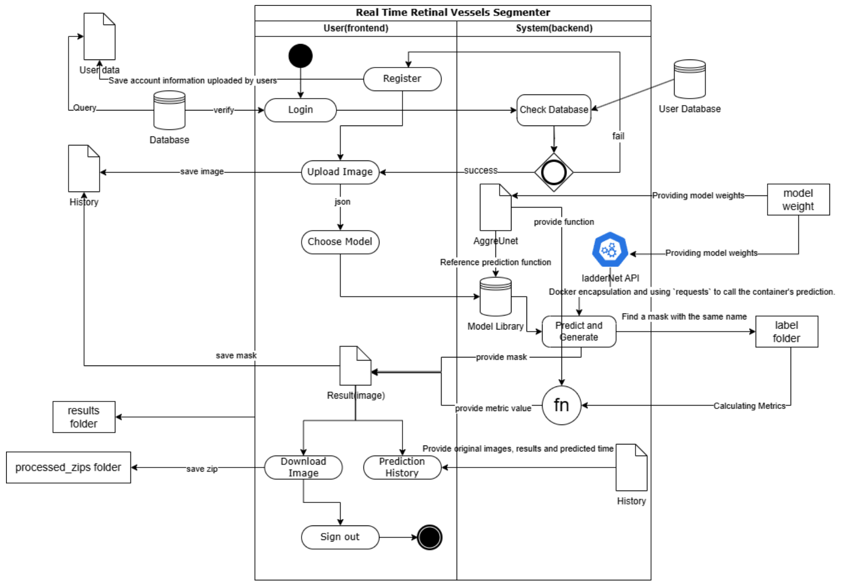


Figure 3: User case

Module Interaction: Users register their accounts through an HTML form, and the account information is stored in the User table of the database, enabling login and registration functionalities. After logging in, users can upload images and ZIP files via an HTML form and select a model for inference. These files and user options are sent to the Flask routes via HTTP requests. Our system includes a custom model, AggreUNet (local inference), which contains the model architecture, prediction, and metric computation functionalities. Flask calls these functions to process the uploaded images, generate corresponding predictions, and compute metric data.
Additionally, we have introduced a new model, Laddernet. For scalability,

this model performs remote inference by requesting another service endpoint through a Docker URL. The encapsulated Docker model processes the user-uploaded images and returns the prediction results. Flask retrieves these results via HTTP requests and evaluates the predictions using the system's metric computation functionality. This approach aims to facilitate the integration of new models into the platform.
Uploaded images are stored in the uploads folder, while the prediction results are saved in either the results folder or the processed_zips folder, depending on the type of file uploaded. These images, along with their predictions, are also recorded in the History table of the database, along with the uploading user and timestamp. This provides data support for users to view their prediction history.

Information Flow: Users upload images or ZIP files via the /upload route. The Flask backend verifies the file and model type, then decides between local inference (AggreUNet) or Docker-based inference (Laddernet). The inference results are saved on the server, and the system computes model performance metrics (Dice, IoU, Accuracy, etc.), which are displayed on the results page. Once the inference is complete, relevant information is stored in the History database table, including the file name, result path, user ID, and timestamp. Users can view the inference result images on the frontend. If a ZIP file was uploaded, users can download a compressed file containing all the processed results.

3 Design Justifications

3.1 Changes in project requirements

Initial design: Initial task was to implement a fast solution for retinal vessel segmentation on datasets like DRIVE, CHASEDB1, STARE etc. Customers request a simple button UI model selection, query image selection, upload 1 or more images store the segmented images in results folder and display them one by one as slide. Prediction function output image for one photo in 1 second. Finally, user wants to show the metrics scores for the images generated on results page.
Changes in sprint requirements: During the project, for further enhancing the user experience, the tutor and our team members suggest adding

functions of registration and login, history management, download prediction result, and add more options for models.

3.2 Iterative design and improvement

Initial design: The initial design emphasizes the completeness of the model. The system only realizes the segmentation function of one AggreUNet model, and the users can only upload one image to get the segmentation result. UI button offers functions of uploading an image and single prediction.
Design limitations: This version of the system does not have user registration and login functions, select model, save the segmentation history, and can only upload one data at a time, which makes the system too monotonous and troublesome to work with.

3.3 Design the iteration process

3.3.1 Sprint1

1. Add user registration and login function
Initial design: In the initial design, it focuses on the function of image segmentation, and the system doesn't have any login and registration function.
Reason: During the meeting, the team found that, for the sake of security protection, of user data, then it has to possess user management functions since this system is going to be operated by different kinds of users.
Change content: Added user registration and login functions and created user tables in the database. Users can register for an account using their username, email, and password, allowing them to log in and have their own account and access permissions.
Effectiveness: The function of registration and login raises the security of system, which allows the users to store the predicted data in the results securely without being stolen by other users.

2. Batch Image Processing and Result Display
Initial design: The preliminary design of the system supports uploading and processing one image alone, and users are allowed to upload images multiple times and can observe the segmentation results only for a single image.

Reason: In customer meetings, batch processing was pointed out as a mandatory requirement because uploading one by one would be very inconvenient for customers when they need to process large-scale datasets. So, in order to save time, we have added batch image processing and incorporated the slideshow display feature in it.
Change content: We've added the folder upload function in the system where users can upload one zip file. The system will batch process these images and show them in a slide show. Besides, it has a download function that can enable the user to pack and download all segmentations.
Effectiveness: Batch processing function raises efficiency in image processing in the system, therefore is suitable for users who need to process a large quantity of model data and simplify user operations.

3.3.2 Sprint 2

1. LadderNet model and supports multiple model selection
Initial design: The initial system designed only one AggreUNet segmentalation model architecture, which was straightforward and did not fulfill the multiple variants that were required by the customer.
Reason: During the meetings with clients and research done about segmentation models, it was very apparent to the team that AggreUNet and LadderNet have their own character with regards to accuracy vs. performance. Though the AggreUNet model performs well in dice scores and can capture more details and small blood vessels, it is weak in noise control and easily affected by noise. The LadderNet model may have shortcomings in handling small blood vessels but does not have noise and the display effect of the main blood vessels is clear. Therefore, the advantages of AggreUNet are higher evaluation metrics, more powerful detail display capabilities, whereas LadderNet has strong resistance to noise and can make the main blood vessels more prominent.
Change content: Added LadderNet model, selected this model for user's prediction in drop down options.
Effectiveness: Users will be able to choose an appropriate model according to their own needs, such as AggreUNet for the higher evaluation metrics and display in detail. And LadderNet is for situations that require emphasis on main blood vessels and noise reduction.

2. Add history function

Initial design: Once the user gets the segmentation result, when going back to the main page, it will not show the previously predicted image results anymore.
Reason: In the group meeting, we realized that a history function will enable users to go back and get results of previous cuts anytime. This will be convenient for customers to check the previous results.
Change content: We have added a history record function in our team, including button for viewing history records and a table for adding history records. And the execution time, model selection, and segmentation results have also been saved into the database that can be viewed by users directly after clicking the history record button.
Effectiveness: View previous segmentation results at any time through historical records. This will make the system from a pure segmentation system into a data management system.

3. Added and then removed image contrast adjustment function
Initial design: In the page of displaying segmentation results, original design only showed the image results of segmentation and did not include the function of image contrast adjustment.
Reason: During the design process, the team suggested adding a contrast adjustment progress bar. Users can adjust the contrast of result image by pulling the progress bar. It was initially thought that this function can improve users' control about the segmentation details. After it actually was implemen-ted and tested, however, we decided that such change in contrast was not high and helped the user very little .
Change content: First, the team introduced a progress bar for contrast adjust-ment; thanks to this, users were able to modify the contrast of the results. In fact, when testing it, it has been found that the image changes very little after such an adjustment. Not to waste space with an ineffective functionality, we have decided to remove the contrast function and recover the display effect of the segmentation results.
Effectiveness: After deleting the contrast adjustment function, the system interface will be more concise and invalid functions are reduced.

3.3.3 Sprint 3

1. Evaluation and Presentation of Metrics' Scores
Initial design: The system only shows the returned result images but does

not own the display function of integrated evaluation indicators.
Reason: In the subsequent meeting with clients, the feedback they provided I performance evaluation indices and have proposed the display in the results of the scores of 5 metrics such as F1 score, recall, miou, dice to clearly present the performance of the segmentation model.
Change content: We have added a calculation function for performance evalu-nation indicators. It displays the F1 score, recall, miou, dice for every segmentation result on the interface. Users can view the performance of the model.
Effectiveness: Based on the function of showing evaluation indicators developed, the system returns scores of metrics for giving scores to return predicted images, by which users can decide whether test results are accurate.

2. Encapsulating databases and systems with Docker
Initial design: The original design separates system deployment and deployment of a database, where users have to configure the connection of databases by themselves for maintaining consistency between database and system environment.
Reason: While the project develops, the team feels that system portability and deployment consistency are key requirements. In this project, we decided to deploy the database and the system into the Docker environment, which may simplify the operation of the system in various environments and reduce configuration issues.
Change content: We encapsulated the entire system and MySQL database into a Docker environment and then used Docker Compose to manage containers. We have packaged all system code, databases, dependency libraries, etc. into Docker images.
Effectiveness: The containerized system has higher portability, allowing users to easily start the system and database with just one click in any Docker supported environment, simplifying the deployment process and avoiding errors.

3.4 Design the iteration process and reasons

3.4.1 Complex Algorithm: Optimization of Lightweight Segmentation Model

AggreUNet model
AggreUNet is a model optimized based on the U-Net architecture, using ResNet as the backbone network for feature extraction. This model introduces the FeatureFuse module for multi-scale feature fusion, enhancing segmentation accuracy and ensuring details are preserved. The FeatureFuse module combines 1x1, 3x3, and dilated convolutions to perform multi-scale feature extraction, enabling the model to handle various details in retinal images.[1] In addition, the ResidualBlock module introduces residual connections, improves gradient flow, stabilizes model training, and enhances segmentation performance while effectively controlling model complexity.

Implementation method: The FeatureFuse module extracts multi-scale features by combining multiple convolution kernels, expanding the model's receptive field without significantly increasing computational complexity. The residual connections in the ResidualBlock module optimize the network structure and support multi-level feature fusion, enabling the model to capture detailed features at various scales. In addition, data augmentation techniques such as random rotation and adding noise have been applied to improve the robustness and accuracy of the model, and experiments have shown that random rotation is particularly effective in improving Dice and IoU scores.
Technical difficulties: The main challenge is to maintain the accuracy of segmentation while controlling the complexity and number of parameters of the model. Small vascular structures are highly sensitive to noise, so the model needs to strike a balance between detailed feature extraction and noise resistance. To this end, we have developed a lightweight convolution operation combination that enables the model to accurately segment target regions at multiple scales while maintaining computational efficiency.
Effectiveness: The optimized AggreUNet model strikes a balance between segmentation accuracy and speed, making it particularly suitable for segmenting small targets and handling complex boundaries. The experimental results showed that the improved model improved the Dice and IoU scores by 3-4 percentage, making AggreUNet an ideal choice for processing complex retinal image segmentation scenarios. Its multi-scale fusion method and anti

noise performance are particularly important for medical diagnostic applications, performing particularly well in scenarios that require high precision and detail preservation.

LadderNet model

A trapezoidal structure is designed to progressively produce the feature down-sampling and upsampling that relies on the strong use of data augmentation to use the available annotated samples more efficiently. Then the features can be extracted and fused gradually at different scales.[3] Initial LadderBlock, LadderBlock, and Final Lad-derBlock are all modules of LadderNet. Each module performs feature extraction and fusion of different resolutions of feature maps and achieves good segmentation performance under the light-weight design of the model. LadderNet fully exploits multiple levels of resolution and has strong detail recognition capabilities and has multiple pairs of encoder-decoder branches, and has skip connections between every pair of adjacent decoder and decoder branches in each level. [5]

Implementation method: The first LadderBlock processes the initial features of the input image, and the LadderBlock realizes the downsampling and upsam-pling in trapezoidal structures to ensure feature interaction between dif-ferent resolutions. The last LadderBlock produces the final segmentation result, so that the multi-scale feature fusion can be completed.

Technical difficulties: The multi-level interaction design of LadderNet should be completed on the premise of ensuring high segmentation accuracy and edge details without increasing excessive computational costs. Through adjustment, the LadderBlock is able to With this, we have a characteristic of feature flow continuity. Additionally, during the interaction between high and low-resolution features, smooth upsampling and downsampling are crucial. We optimized upsampling and downsampling to make appropriate connections of high-level and low-level features.

Effectiveness: Trapezoid structure design made LadderNet not only improve the accuracy of segmentation but also maintain high processing efficiency, this makes the solution applicable to batch images processing.

3.4.2 Tool integration

Docker containerization deployment and service integration.

Implementation method: To guarantee the cross-operating system compatibility of the system and avoid complicating the deployment steps, we use Docker to containerize Flask applications, MySQL databases and segmentation models, and deal with each container's dependency and communication by Docker Compose. Each of the services has been wrapped into a different container for the purpose of making the startup of the system consistent in various system environments.
Technical difficulty: In the multicontainer environment, we need to reasonably configure the environment variable settings and network settings in order to make sure that the communication between modules is stable. In this process, to avoid the conflict in using some particular ports and database connection mistakes, the team did some detailed debugging with respect to the method of connection and ports allocation, which resulted in reaching the goal of stable communications and reasonable container resource allocation.
Effectiveness: Docker containerization deployment provides the system with easy migration to another hosting environment and much faster scaling. Containerized solution greatly improves the maintainability of the system, consistency of the development and production environments, and simplifies the deployment process.

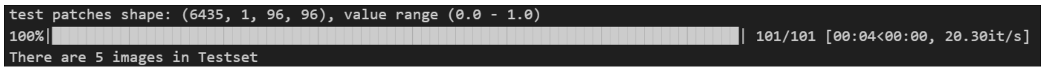
4 User-Driven Evaluation of Solution

Our project, the **Real-Time Retinal Vessels Segmenter**, is designed to address the need for a lightweight, accurate, and real-time retinal vessel segmentation model. Below, we evaluate how the main features of our solution align with the specific user needs derived from the project requirements and user stories.

4.1 Real-Time Efficient Vessel Segmentation

User Need: Medical professionals require retinal vessel segmentation results within the shortest possible time frame to provide instant feedback for diagnosis and monitoring, especially in high-demand healthcare settings where rapid processing is essential.
Solution: We developed a deep learning model capable of segmenting retinal vessels in under one second, achieving the real-time performance goal. This feature significantly enhances operational efficiency and ensures the tool's

usability in high-load medical environments, meeting the user's need for quick response times.

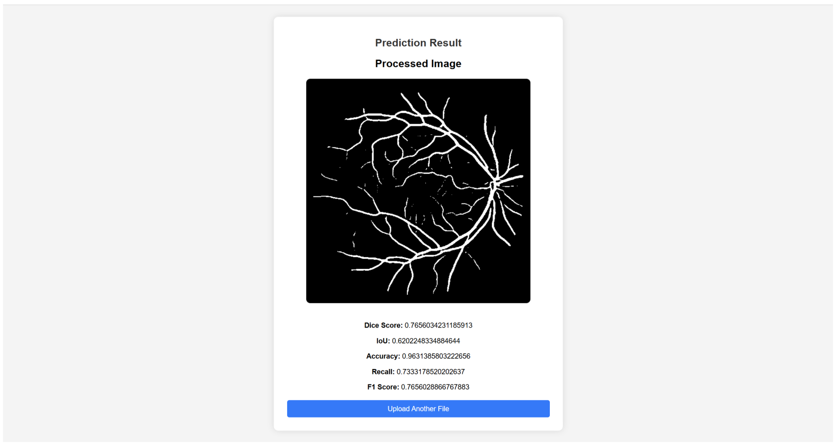


```
test patches shape: (6435, 1, 96, 96), value range (0.0 - 1.0)
100%|██████████████████████████████████████████████████| 101/101 [00:04<00:00, 20.30it/s]
There are 5 images in Testset
```

Figure 4: Real-Time Processing with Test Patches and Progress Bar

4.2 High-Precision Segmentation

User Need: High accuracy in segmenting blood vessels is crucial in medical image analysis, as precise vessel localization and boundary identification directly impact diagnostic accuracy.
Solution: By optimizing neural network architecture and parameters, our model achieves competitive segmentation accuracy, capable of detecting fine vascular structures. Extensive testing on multiple datasets confirms the reliability of the model's accuracy, making it suitable for clinical applications. This high precision fulfills the user's demand for accuracy in segmentation results crucial for medical diagnostics.



Dice Score: 0.7656034231185913

IoU: 0.6202248334884644

Accuracy: 0.9631385803222656

Recall: 0.7333178520202637

F1 Score: 0.7656028866767883

Figure 5: Metric Results

4.3 Simple and Intuitive User Interface

User Need: Many target users are medical professionals with minimal technical background, so the system should be intuitive, with a low learning curve,

and free from unnecessary technical complexity.
Solution: We designed a straightforward, user-friendly interface that enables users to complete model selection, image loading, and segmentation result visualization with ease. With just a few clicks, users can obtain vessel segmentation results, enhancing diagnostic efficiency.

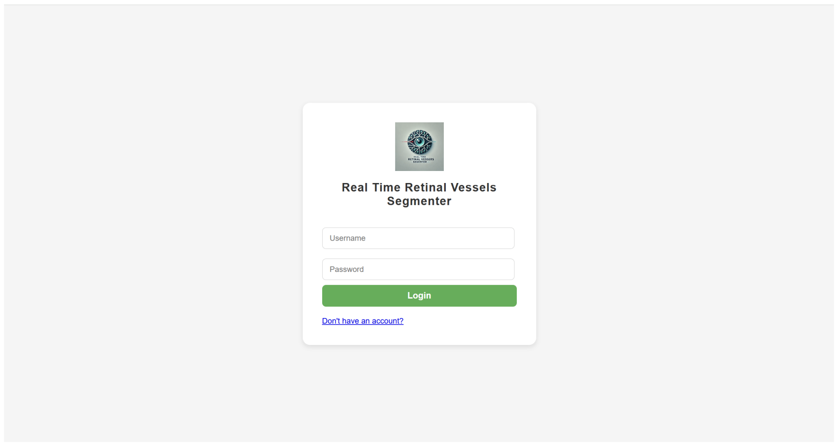


Figure 6: User Interface

4.4 Multi-Model Support and Flexibility

User Need: Retinal images from different patients may vary significantly due to disease type and imaging conditions. Therefore, users would benefit from the ability to switch between different models to ensure the system's applicability across varied imaging scenarios.
Solution: We implemented a model selection feature, allowing users to switch between pre-trained models to address various imaging needs. This flexibility increases the system's applicability, making it suitable for diverse medical contexts, and ensures consistent segmentation accuracy across different scenarios.

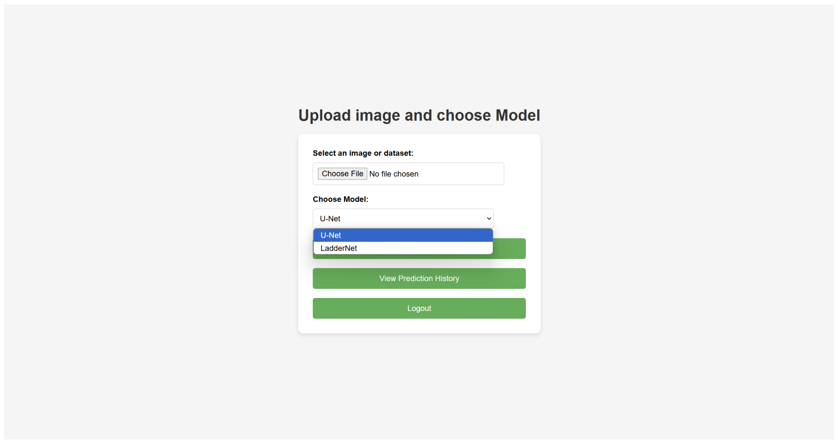
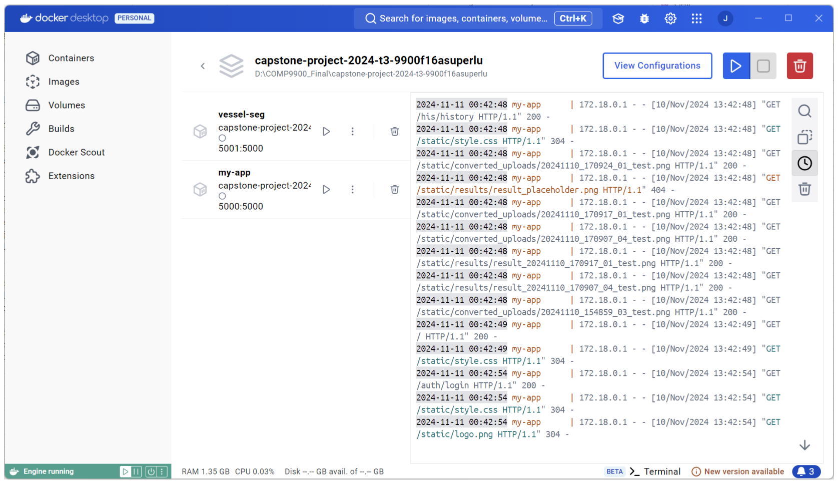


Figure 7: Selecting Different Trained Models for Comparison

4.5 Stability and Reliability with Docker Packaging

User Need: The system must be stable to handle continuous diagnostic tasks and large data volumes without crashes or delays, while also ensuring easy deployment across different environments.
Solution: To enhance stability and compatibility, we packaged the entire system using Docker. This allows the solution to be easily deployed in different environments without compatibility issues. Multiple rounds of stress and long-duration tests validated the system's stability in continuous operation. This stability meets the high reliability requirements necessary for practical use in medical environments.

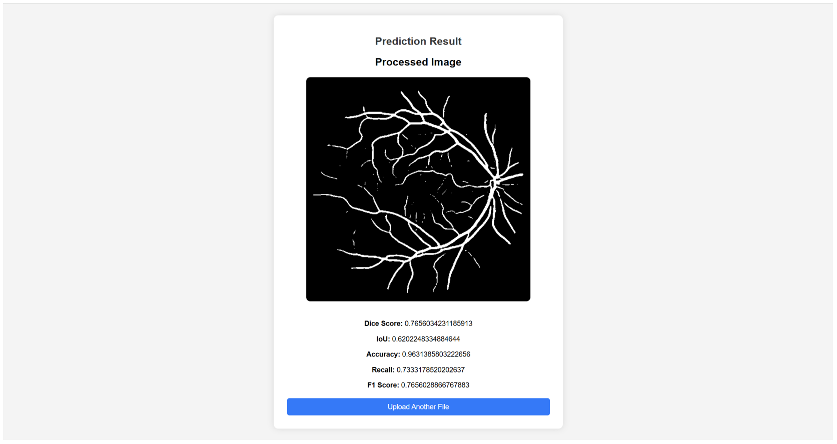


```
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/his/history HTTP/1.1" 200 -
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/static/style.css HTTP/1.1" 304 -
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/static/converted_uploads/20241110_170924_01_test.png HTTP/1.1" 200 -
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/static/results/result_placeholder.png HTTP/1.1" 404 -
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/static/converted_uploads/20241110_170917_01_test.png HTTP/1.1" 200 -
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/static/converted_uploads/20241110_170907_04_test.png HTTP/1.1" 200 -
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/static/results/result_20241110_170917_01_test.png HTTP/1.1" 200 -
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/static/results/result_20241110_170907_04_test.png HTTP/1.1" 200 -
2024-11-11 00:42:48 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:48] "GET
/static/converted_uploads/20241110_154859_03_test.png HTTP/1.1" 200 -
2024-11-11 00:42:49 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:49] "GET
/ HTTP/1.1" 200 -
2024-11-11 00:42:49 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:49] "GET
/static/style.css HTTP/1.1" 304 -
2024-11-11 00:42:54 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:54] "GET
/auth/login HTTP/1.1" 200 -
2024-11-11 00:42:54 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:54] "GET
/static/style.css HTTP/1.1" 304 -
2024-11-11 00:42:54 my-app      | 172.18.0.1 - - [10/Nov/2024 13:42:54] "GET
/static/logo.png HTTP/1.1" 304 -
```

Figure 8: Docker Packaging for System Deployment

4.6 Evaluation Metric Results Display

User Need: Users, particularly clinicians and medical researchers, need to see evaluation metrics (e.g., accuracy, sensitivity, specificity) for each segmentation task to assess the model's performance on individual cases.
Solution: Our system provides an evaluation metrics display, showing relevant performance metrics alongside segmentation results.



Dice Score: 0.7656034231185913

IoU: 0.6202248334884644

Accuracy: 0.9631385803222656

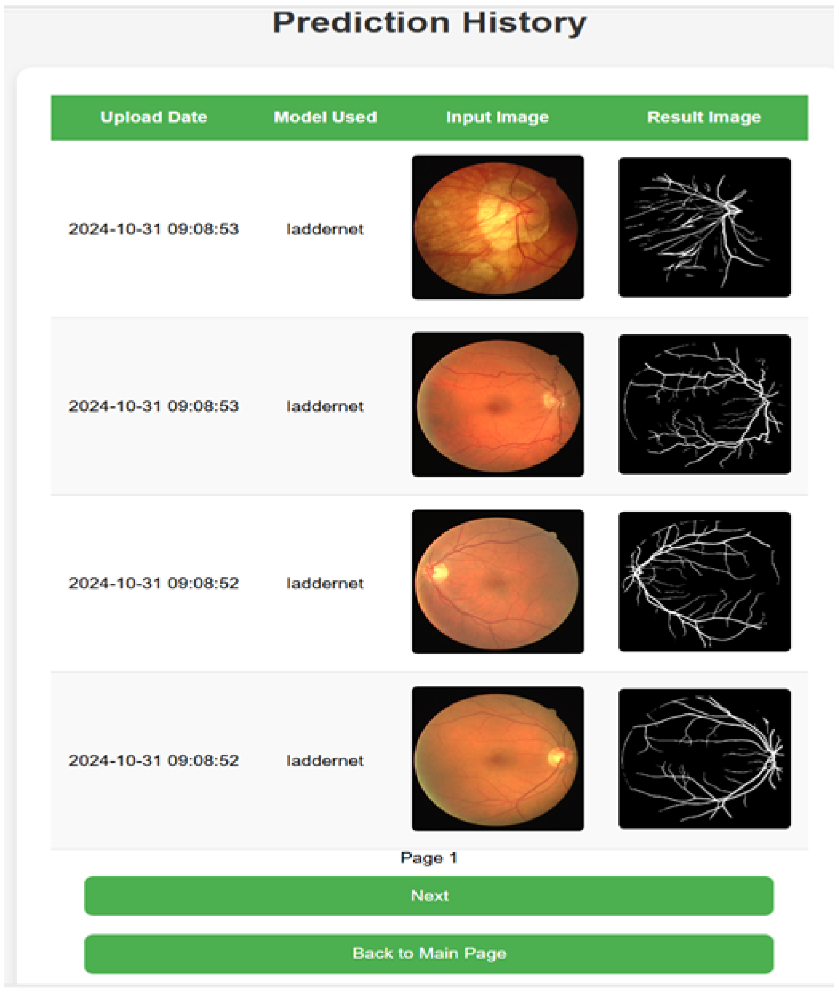
Recall: 0.7333178520202637

F1 Score: 0.7656028866767883

Figure 9: Metric Results Display for Model Evaluation

4.7 Prediction History Tracking

User Need: Medical professionals may need to refer back to previous segmentation results for comparison and historical reference, especially when monitoring changes over time.
Solution: We implemented a prediction history function that stores past segmentation results, allowing users to easily access and review previous predictions. This feature supports continuity in patient monitoring and diagnostic tracking.

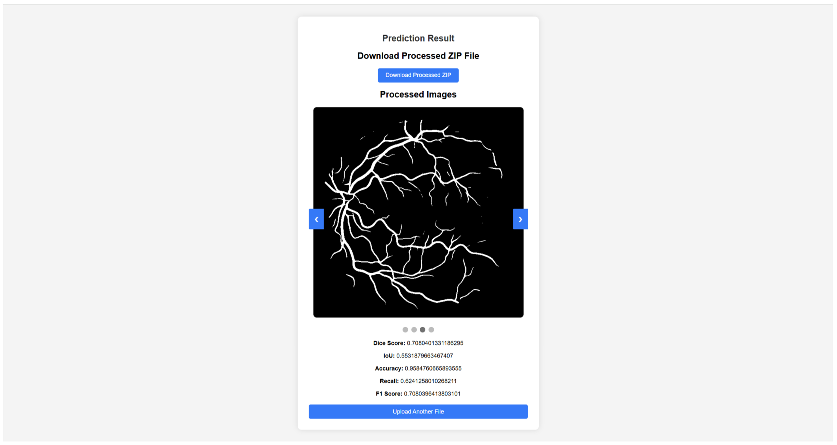

Prediction History

Upload Date	Model Used	Input Image	Result Image
2024-10-31 09:08:53	laddernet		
2024-10-31 09:08:53	laddernet		
2024-10-31 09:08:52	laddernet		
2024-10-31 09:08:52	laddernet		

Figure 10: Prediction History for Reference and Comparison

4.8 Multiple Results Presentation and Downloading

User Need: Users may wish to view and compare multiple segmentation results at once and download them for documentation or further analysis.
Solution: The system includes a feature to display multiple segmentation results simultaneously and offers download functionality. This meets the users' need for comparative analysis and facilitates the recording of segmentation outcomes.



Dice Score: 0.7080401331186295

IoU: 0.5531879663467407

Accuracy: 0.9584760665893555

Recall: 0.6241258010268211

F1 Score: 0.7080396413803101

Figure 11: Multiple Results Presentation and Downloading

4.9 Dataset Upload Function

User Need: Medical institutions may want to test the system with their specific datasets to assess its performance in unique contexts.
Solution: We implemented a dataset upload function, allowing users to upload custom datasets and test the segmentation model on new data. This enhances the system's adaptability and allows it to meet various institutional needs.

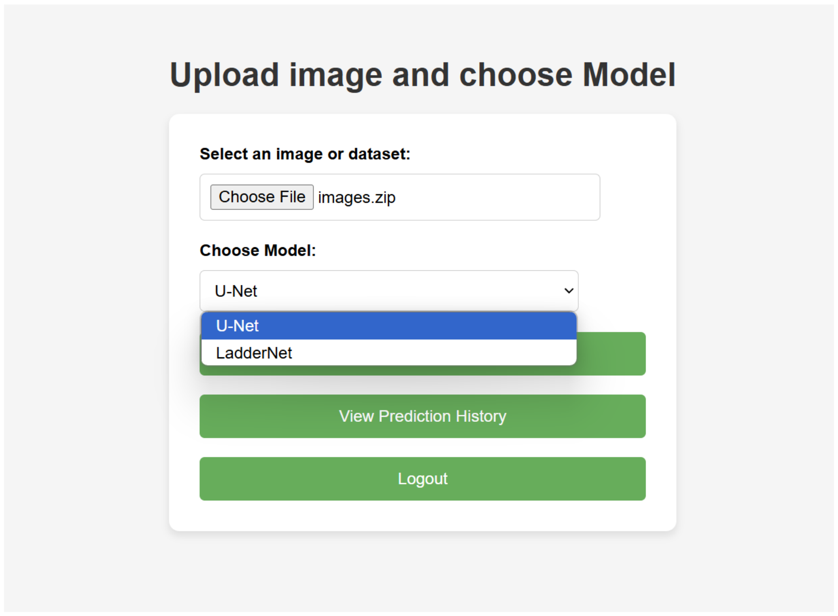


Figure 12: Dataset Upload Function for Custom Testing

4.10 Login and Registration Functions

User Need: Access control is essential for user-specific data storage, data privacy, and personalized settings, especially in a healthcare environment.
Solution: We implemented a secure login and registration feature to manage user access, ensuring that data is kept secure and user-specific settings are maintained. This feature supports data privacy requirements and allows each user to maintain their settings and history.

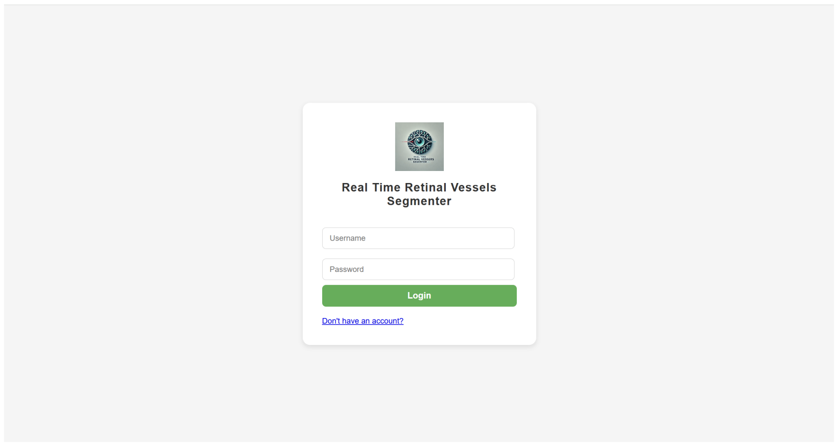


Figure 13: Login Function

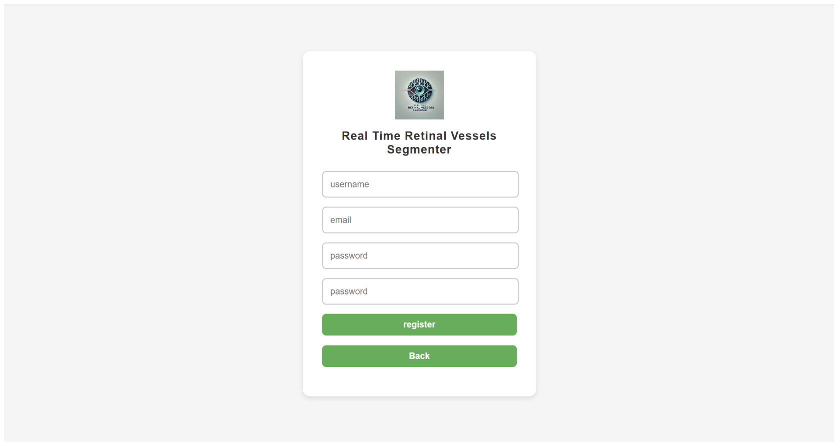


Figure 14: Registration Function

5 Limitations and Future Work

5.1 Web related

Network Communication Delay: Docker-based remote inference relies on network communication for requests and responses. Network instability or insufficient bandwidth may cause delays, impacting the user experience.
Concurrent Request Handling: Flask operates in a single-threaded mode

by default, which is inefficient for handling a large number of concurrent requests. When multiple users upload files and request inference simultaneously, server congestion and increased response times may occur.
File Transfer Efficiency: Uploading large files (e.g., high-resolution images or ZIP files containing multiple images) takes considerable time and consumes the server's I/O bandwidth.
Future Work:

- Introduce asynchronous task queues (e.g., Celery) to improve concurrency performance.
- Employ high-performance web servers (e.g., Nginx) to enhance request handling capabilities.

5.2 Model related

Dataset Limitation: During training, we utilized the DRIVE and STARE datasets, but the sample size of each dataset remains relatively small. This limitation may affect the model's generalization capability. Improving the model's generalization can effectively enhance its predictive performance on different types of retinal images.
Future Work: Due to the limited availability of public medical image datasets, we can perform more data augmentation. During our model research, we encountered a data augmentation method that can split the dataset into more high-resolution files, thereby improving the model's ability to capture details. However, this method generates a large number of data files and requires significant computational resources, posing a high demand on local computing power. Therefore, in the future, we can leverage server resources to perform such data augmentation. Additionally, incorporating unsupervised learning methods, such as GAN-based adversarial generation mechanisms [2], can further enhance the model's segmentation performance.
Inference Performance and Prediction Time: Local inference relies entirely on the server's hardware resources. When processing high-resolution images or a large number of files, insufficient computational resources can lead to slower inference speeds. Remote inference, on the other hand, depends on the hardware environment of the container. If the model has a large number of parameters or high computational demands, prediction time may increase, thereby affecting the user experience.

Future Work: Using high-performance GPUs and optimizing hardware configurations are the most straightforward and effective methods. Additionally, by introducing deep supervision mechanisms and pruning the model[4], the number of model parameters can be further reduced, thus improving inference speed and enabling real-time segmentation.
Model accuracy: The model has basic blood vessel segmentation capabilities, but its accuracy still needs further improvement. When handling more demanding medical tasks such as quantifying vessel curvature and diameter, the model's performance is not precise enough.
Future Work: Try using more complex deep learning models (Transformer-based models) to capture finer features. Adopt loss functions suitable for class imbalance and detail capturing (such as Dice Loss or Tversky Loss) to focus more on the segmentation of small blood vessels. Additionally, in cases where annotated data is limited, leverage weakly supervised or self-supervised learning techniques to enhance the model's ability to learn detailed features.

References

[1] Ahmed AL Qurri and Mohamed Almekkawy. "Improved UNet with Attention for Medical Image Segmentation". In: *Sensors* 23.20 (2023). ISSN: 1424-8220. DOI: `10.3390/s23208589`. URL: `https://www.mdpi.com/1424-8220/23/20/8589`.

[2] Shijie Cheng et al. "SUGAN: A Stable U-Net Based Generative Adversarial Network". In: *Sensors* 23.17 (2023). ISSN: 1424-8220. DOI: `10.3390/s23177338`. URL: `https://www.mdpi.com/1424-8220/23/17/7338`.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* 2015. arXiv: `1505.04597 [cs.CV]`. URL: `https://arxiv.org/abs/1505.04597`.

[4] Sunil Vadera and Salem Ameen. *Methods for Pruning Deep Neural Networks.* 2021. arXiv: `2011.00241 [cs.LG]`. URL: `https://arxiv.org/abs/2011.00241`.

[5] Juntang Zhuang. *LadderNet: Multi-path networks based on U-Net for medical image segmentation.* 2019. arXiv: `1810.07810 [cs.CV]`. URL: `https://arxiv.org/abs/1810.07810`.