Image Recognition of Chemical Molecules

Alp OZCAN, Alp ATES, Betul ASLAN, and Melis KARADAG

Supervisor: Stefan DUFFNER
Institut National des sciences appliquées de Lyon

Abstract. This project presents an automated system for recognizing and analyzing chemical reaction diagrams from scientific publications, focusing on water treatment studies. The system processes diagrams through a sequential pipeline that includes preprocessing and arrow detection. A U-Net model, trained with a large amount of data, identifies arrow tips in chemical reaction diagrams and a series of modules which gives us the whole arrow. The integration of these components enables accurate mapping of reaction pathways. The goal is to automate the analyzing process.

Keywords: Chemical reaction diagrams, U-Net model, image recognition, machine learning

1 Introduction

1.1 Background

This project is conducted as part of the collaboration between INSA Lyon, LIRIS, and INRAE. Water treatment studies frequently contain reaction schemes that need to be analyzed, which led to the need for advanced tools capable of handling large volumes of data to automate the system. Recognizing this, the project's preliminary objective aimed to integrate image recognition with machine learning techniques, enabling researchers to process chemical information effectively.

Building upon previous work, this project continues the efforts of Aya Talbi El Alami, a LIRIS intern, who focused on integrating the DECIMER library for recognizing chemical structures in reaction schemes. The initial attempt to train a neural network model gave inconclusive results. The remaining objective for the project was to refine and improve this model, integrate it into the processing pipeline, and ultimately deduce the chemical reaction paths from the diagrams. Building on her work, the current phase of the project, supervised by Stefan Duffner (LIRIS), centers on recognition of chemical reaction diagrams using deep learning and image processing techniques.

1.2 Problem Statement

It is important to understand how molecules transform in fields like environmental sciences and chemistry. Chemical reaction schemes show the formation of a new molecule because of various reactions that a molecule goes through. However, analyzing these schemes manually often requires examining many reactions. This takes a lot of time and can lead to mistakes. The aim of this project is to develop a system that can automatically analyze chemical reaction schemes. In this approach, the system recognizes chemical structures from images and traces their reaction sequences. Machine learning and image processing techniques have been utilized in the implementation of the project.

1.3 Overview

1.3.1 Chemical Image Recognition

Chemical image recognition is the process of extracting meaningful chemical information from visual data, such as reaction schemes and molecular diagrams. The traditional method is manually examining visual data, such as reaction diagrams, to extract molecular structures, and reaction pathways. On the other hand, modern methods use deep learning models, such as convolutional neural networks (CNNs), to automate the extraction of molecular structures, bonds, and reaction pathways from images. Tools like DECIMER (Deep Learning for Chemical Image Recognition) simplify this process by converting diagrams into standardized formats like SMILES.

1.3.2 SMILES

The Simplified Molecular Input Line Entry System (SMILES) is a specification in the form of a line notation for describing the structure of chemical species using short ASCII strings. SMILES strings can be imported by most molecule editors for conversion back into two-dimensional drawings or three-dimensional models of the molecules. The original SMILES specification was initiated in the 1980s. It has since been modified and extended. In 2007, an open standard called OpenSMILES was developed in the open-source chemistry community.[1]

SMILES is a text-based format used to represent chemical structures. For example, ethanol is written as C-C-O. This format shows atoms using element symbols, bonds with symbols like - for single bonds and = for double bonds, and branches or cycles using parentheses and numbers. It is widely used because it allows fast searching, easy analysis, and compatibility with machine learning models.

1.3.3 DECIMER

DECIMER is "a deep learning method based on existing show-and-tell deep neural networks, which translates a pure bitmap image of a molecule, as found in publications, into a SMILES."[2]. It uses Convolutional Neural Networks (CNNs) for the extraction of chemical information from diagrams and translates them into standardized format, SMILES. Firstly, image preprocessing removes noise and normalizes images for consistent input to the neural network. Then, the CNN identifies molecular structures, bonds, and functional groups. Finally, these recognized structures are converted into SMILES which is a text-based format. Compared to traditional methods, DECIMER has better results in terms of handling noisy data, and generalization for different types of chemical diagram styles.

1.4 Initial Work and Our Approach to Reaction Scheme Recognition

Within the scope of this project on image-based recognition of reaction schemes, initial work done by Aya Talbi El Alami included detection using the U-Net model on arrow tips, while integrating DECIMER to recognize the chemical structure but encountered some challenges in optimizing the U-Net model configuration. Building on this foundation, our approach is based on improving the success of U-Net model by fine tuning the parameters. Also adding advanced line detection technique to provide reaction pathway mapping.

2 Methodology

2.1 Project Pipeline Introduction

Our system consists of a series of interconnected modules to process chemical reaction diagrams. These models work in harmony and ensure an efficient workflow. Each module has a specific task, such as image preprocessing, arrow recognition, etc. Together, these modules take the result of his predecessor and implement it into his module, which ensures achieving accurate and reliable results. The modular design ensures flexibility, scalability, and convenience to easily modify a module without affecting another one. Our pipeline consists of several modules:

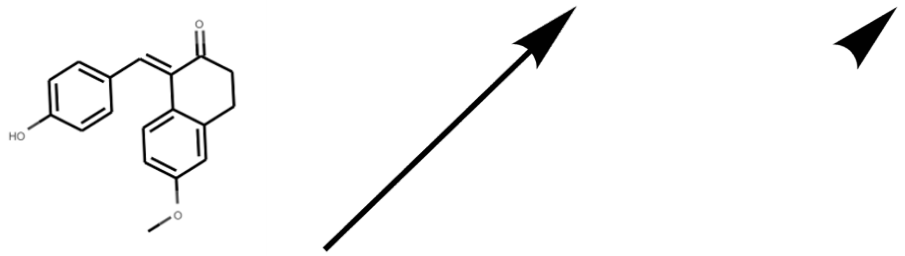
1. Data Collection
2. Model Structure (U-Net)
3. Arrow Detection Pipeline
4. Line Detection
5. Result

2.2 Development Environment

For our project, we used Python 3.10.15 as the main programming language. Our U-Net Model is built with TensorFlow, specifically Keras library, which has comprehensive tools for neural network implementation and training. For image recognition and line detection, we used OpenCV's extensive library. Text detection and removal are handled through the EasyOCR library, while visualization components utilize Matplotlib for result analysis. We also used NumPy for numerical operations.

2.3 Data Collection

The previous researchers formed the dataset used for this project. The data consists of chemical reaction diagrams which were obtained from publicly available research papers, online databases and textbooks from the internet. The data collection process was followed using Python scripts to automate the extraction and segmentation of images from the collected articles resulting in two distinct datasets, one containing chemical structure diagrams and another one with the arrows. At this stage arrow tip masks were manually created by deleting every pixel in the image except for the arrow tips. This extraction and segmentation process created 40000 chemical structure diagrams for the dataset. Additionally, the data was systematically organized and numbered to ensure consistency and facilitate easier handling during preprocessing and training.



(a) Chemical Structure (b) Arrow (c) Arrow Mask

Fig. 1: Example chemical structure diagram, arrow, and arrow mask.

2.4 Model Structure

2.4.1 Model Architecture (U-Net)

U-Net is a convolutional neural network (CNN) first introduced in "U-Net: Convolutional Networks for Biomedical Image Segmentation" paper. The U-Net architecture is particularly effective in tasks requiring precise localization in images in our case detecting the arrow tips.

U-Net structure consists of two symmetric parts. The Contracting Path and the Expansive Path. Between these paths there are connections named Skip Connections.

The Contracting Path

This path uses a combination of convolutional layers and max-pooling operations to extract high level abstract features while reducing spatial dimensions

Skip Connections

A unique feature of U-Net is the skip connections, which bridge the encoder and decoder stages by transferring low-level features from the contracting path directly to the corresponding up-sampling layers. These connections ensure that spatial information lost during down-sampling is effectively recovered, enabling accurate localization.

The Expansive Path

This path functions as decoders restoring the spatial resolution by applying up sampling operations followed by convolutional operations. By combining restored spatial resolution and high-level features through skip connections, the expansive path produces precise outputs.

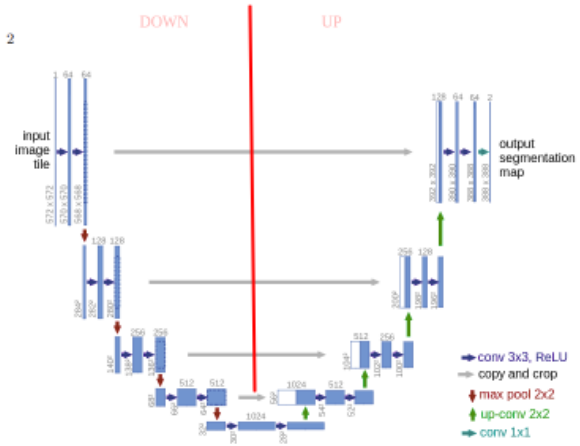


Fig. 2: U-Net Architecture for Image Segmentation. The architecture consists of a contracting path (downsampling) and an expansive path (upsampling).

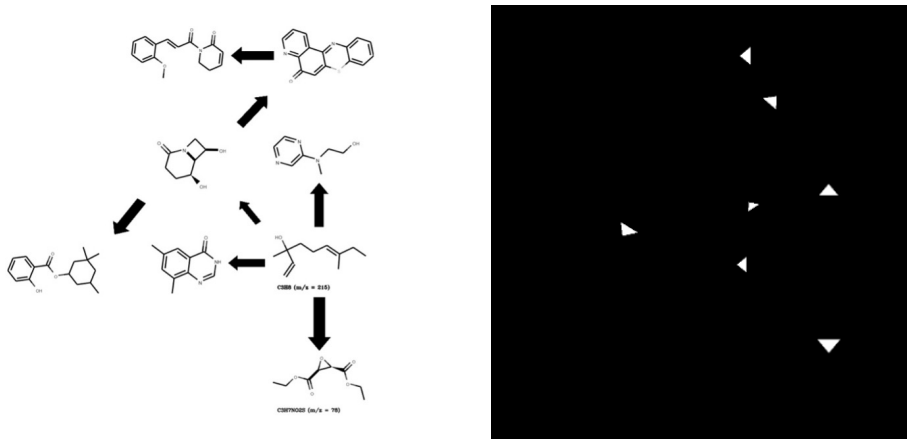
With these advantages of the U-Net architecture, the model was implemented using the TensorFlow and Keras deep learning frameworks, which provide high-level abstractions for building and training neural networks efficiently. Our model is implemented with an input size of 512 x 512 x 1 suitable for greyscale images.

The architecture has a contracting path that reduces spatial dimensions down to 32 x 32 and the expansive path restores the original resolution. The model has 23 convolutional layers in total. Each layer applies filters of size 3 x 3 except for the outout layer which uses 1 x 1 filters for binary segmentation. The number of filters doubles at each downsampling stage in the contracting path, starting at 32 and reaching 512 at the bottleneck.

2.4.2 Model Training

1. Preparing the Dataset

The arrows in the dataset were preprocessed to normalize the values of pixels to [0, 1] range. To ensure the success of the model the dataset was augmented. Scripts were used to automate this generation process. This process is realized by rotating and changing the locations of the extracted arrows. After the augmentation process the dataset contained 2522 arrows with 2522 arrow masks. By using another script, the datasets mentioned previously were used to form training images. The script randomly placed arrows and chemical structure diagrams on new images to simulate a chemical reaction. Simultaneously another dataset was generated which contained only the arrow masks corresponding to these synthetic chemical reactions. These weren't intended to represent valid chemical reactions but were only used to distinguish the arrow tips when there are other components in the image. The interpretation of the reactions is not important in this step. Finally, all images were preprocessed to fit the 512 x 512 input size of the U-Net model, ensuring consistency during training.



(a) Chemical reaction diagram. (b) Arrow mask representation.

Fig. 3: Comparison of the chemical reaction diagram and the arrow mask representation.

2. Training Configuration

Loss Function

The model was created using the binary cross-entropy loss function. This function is suitable for binary segmentation tasks. It measures the difference between the predicted and the real values.

Mathematically Binary Cross-Entropy (BCE) is defined as:

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

where:

- N is the number of observations.
- y_i is the actual binary label (0 or 1) of the i-th observation.
- p_i is the predicted probability of the i-th observation being in class 1[4] .

Optimizer and Learning Rate

We used Adam optimizer with a learning rate of 0.001. Adam is a well suited optimizer for segmentation tasks due to its adaptive learning capabilities. 0.001 learning rate ensured a balance between convergence speed and model stability. The learning rate value is crucial because higher learning rate might cause the model to overshoot the optimal value while a lower value could result in low convergence or getting stuck in a suboptimal state [5].

Batch Size

During the training process the batch size was set to 32. This value was selected to balance computational efficiency and memory usage as larger batch sizes could exceed GPU memory limits while smaller batch sizes might act oversensitive and might result in noisy gradient updates. With a batch size of 32, the model benefited from stable gradient updates, facilitating smoother and faster convergence.

Epochs

The model was trained for 50 epochs. This value was chosen through trial and error to ensure model was working with low loss while preventing overfitting of the training dataset.

Validation Split

A 10% split of the dataset was reserved to monitor the accuracy of the model's prediction during the training. 10% was considered sufficient to monitor the success of the model without significantly reducing the training dataset.

2.4.3 Model Result

The trained model outputs a list of coordinates where each [x,y] represents the position of a point of the arrow tip: $\left[\left[[x_1, y_1], [x_2, y_2], \ldots, [x_n, y_n]\right]\right]$, where x_i and y_i represent the horizontal and vertical positions of the i-th arrow tip in the image.

This approach is the base for centroid creation, a process detailed in the further stages of the pipeline. The objective was first to evaluate and visualize which points were detected by the model and if it needed re-training or tuning of the parameters.

Visualization with Matplotlib

To evaluate the model's performance we used Matplotlib to visualize the arrow tip points of the output list as red points over the original image. This step provided several benefits:

1. Verification: It allows immediate visual verification making it easier to identify false or missed detections.
2. Threshold Optimization: By observing different outputs from the test images we fine-tuned the threshold to achieve optimal balance between precision and recall.

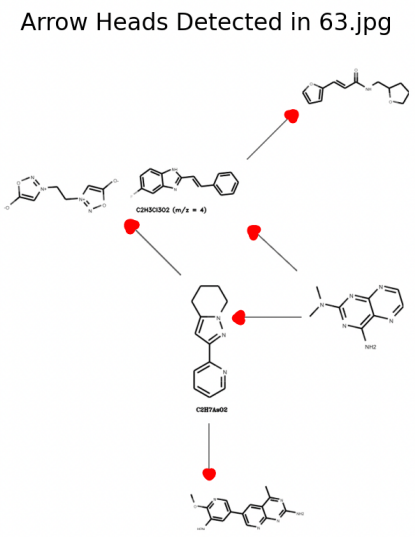
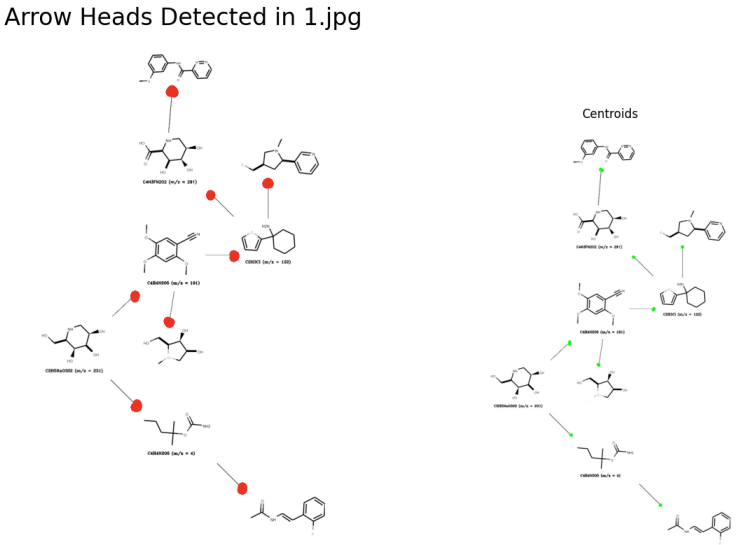


Fig. 4: Arrow heads detected in a chemical reaction diagram. The red dots indicate the detected arrow tips.

2.5 Arrow Detection Pipeline

2.5.1 Centroid Creation By taking the result from U-Net algorithm which consists of a matrix with x and y coordinates of approximate locations of arrows. Then we aimed to transform these matrix points into one single object for coordinates too close to each other to get one single result for every single arrow. After a thorough search at the libraries of Python, we decided to use OpenCV's get connected components function, specifically get connected components with stats, to turn these dots into one specific 'centroid'. This function takes input as our result matrix. It gives us the output as the number of connected components (including background), label matrix where each pixel is located with their respective ID, bounding box of the connected component, the area (in pixels) of the component, and the centroid's center of mass (x,y) coordinates [3]. To check if there is a cluster between actual arrows and our centroids, we showed our image with our centroids marked green (see Fig.1 (b)).



(a) Arrow heads with red dots representing the result matrix of the algorithm.

(b) Arrow heads represented as centroids as green dots.

Fig. 5: Arrow head detection results: (a) algorithm output with red dots, (b) centroid representation with green dots.

By comparing these images, we came to conclusion that our algorithm detects approximately arrows with matrix and centroid coordinates which is important for detecting the lines connecting the arrow tips.

2.6 Line Detection and Connection Analysis

After detecting the arrow tips as centroids in our algorithm, our next step is detecting the lines following them. For detecting lines in an image, there is a lot of variety of algorithms for this objective. After analyzing and testing multiple algorithms, we agreed to use the Probabilistic Hough Line Transform, which is a more efficient implementation of the Hough Line Transformation. The Probabilistic Hough Line Transform gives the extremum points of the detected lines as (x0, y0, x1, y1) [6]. Before applying the transform, we used the Canny edge detector for detecting the edges in our image. This detector provides us with a simplified binary edge map. The binary edge map reduces the search space and makes the algorithm faster and more reliable [7]. The Probabilistic Hough Line Transform takes the binary edge map which is the output of the Canny edge detector, the resolution of the parameter r in pixels, the resolution of the parameter θ in radians, the minimum number of intersections to detect a line, the minimum distance between points that can form a line and maximum gap between two points to be considered in the same line. The function gives us the output as an array of extreme points of the lines [6]. After finding out every line possible in our image, we use another function to detect which function is within the threshold distance of a centroid. We use a recursive function to go through every line in this array to find the lines that intersects with a centroid and add them to a different array. And because every arrow tip has one line that follows him in most of cases, we limit the number of lines intersecting a centroid to 1. By having multiple edges of a line, we use backtracking the lines to reach their start point.

2.7 Integration of Components

After creating all modules, we put all these modules in a sequential format to create the pipeline. We created a new function `get_result` which forms the pipeline with every output being the input of the next one. We ensured to validate each output before being passed to the next component to ensure data integrity. The pipeline starts with the preprocessing and text removal of the image for getting better and more accurate results. Then we implement arrow detection algorithm using U-Net model, which outputs a binary mask indicating arrow tip's (x, y) coordinates. This binary mask is then processed by the arrow detection pipeline to give us the centroids with their center of mass' (x, y) coordinates. The line detection module then processes these components to establish connections between arrow tips and lines, creating a complete representation of the chemical reaction pathways. Finally, the result is given both as an image for interpreting the result and an array of arrow tip's (x, y) coordinates.

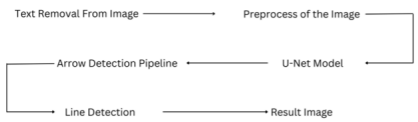
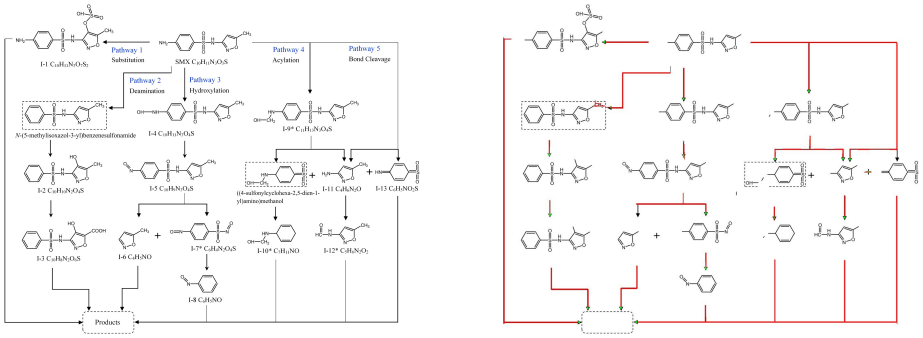


Fig. 6: Image representation of the algorithm's pipeline

3 Results and Discussions

3.1 Model Performance

Our U-Net model demonstrated strong performance in detecting arrow tips from real-world chemical reaction diagrams, including complex cases with multiple molecular structures and reaction pathways. As shown in the figure, the model successfully identified arrow tips with high precision, highlighted as green circles. These detected tips align accurately with the actual arrow tips in the original diagram, and the corresponding reaction lines are marked in red. To further enhance the model's accuracy, we implemented a preprocessing step to remove text elements from the diagrams. This preprocessing significantly improved the detection performance by reducing noise and ensuring cleaner input for the model. As a result, the model was able to achieve more consistent and reliable predictions across diverse chemical diagrams.



(a) Chemical reaction diagram.

(b) The result with centroids and intersecting lines.

Fig. 7: Comparison of the chemical reaction image and the result image.

3.2 Challenges Encountered

While developing the modules for our pipeline, we faced several challenges.

1. **Hardware Limitations:**
   For training the U-Net model, the computer we trained the algorithm had some insufficient hardware to support effective training. This limitation forced us to reduce the epoch count and batch size, which impacted the training process. Additionally, the training duration was longer than anticipated, further restricting our ability to fine-tune the model.

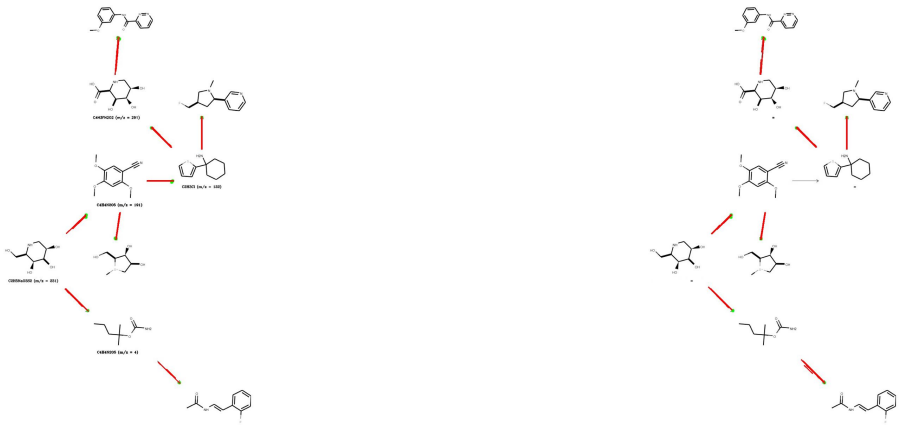
2. **Line Detection Algorithm:**
   We experimented with various line detection algorithms, each with its unique strengths and weaknesses. These algorithms produced outputs in different formats and required diverse configurations. Through extensive testing, we optimized the settings for each algorithm and ultimately integrated the one that produced the least errors into our pipeline. However, even with the optimal settings, some cases yielded incorrect results, and we had to prioritize minimizing errors overall.

3. **U-Net Model Misdetections:**
   One significant issue with the U-Net model was the misdetection of arrow tips. Although the algorithm correctly identified most arrow tips, it occasionally mistook parts of chemical formulas for arrow tips. This misdetection disrupted the rest of the pipeline, leading to incorrect outputs.

4. **Centroid Creation Module:**
   In the centroid creation module, centroids rarely appeared in random locations within the image. These wrongly placed centroids led to the detection of unnecessary lines and incorrect chemical pathways, complicating the results further.

5. **Dataset Limitations:**
   The dataset used for training and validation did not cover every possible variation in chemical diagrams. This limitation caused the model to struggle with certain edge cases, such as diagrams with atypical arrow styles, highly stylized reaction pathways, or unconventional layouts. Augmenting the dataset with more examples of these cases would likely improve model performance.

6. **Text Removal Challenges:**
   During preprocessing, removing text elements from the diagrams occasionally led to the removal or reducing the resolution of certain arrow. This loss sometimes caused the algorithm to misinterpret the reaction pathways or fail to detect certain lines. Although the preprocessing step improved arrow tip detection overall, it introduced an additional layer of complexity.



(a) The result with text labels preserved.

(b) The result after removing text labels.

Fig. 8: Comparison of the results before and after text removal.

3.3 Future Improvements

Our project is currently focused on detecting the starting and ending molecules in chemical reaction diagrams. While we have made progress, the work is still ongoing, and there are several planned improvements for the future. In the next stages, we aim to further optimize the algorithm to improve its accuracy and efficiency. Additionally, we plan to integrate DECIMER into the pipeline to accurately identify the input and output molecules in the reactions. This integration will enable us to fully automate the process and provide more comprehensive results for analyzing chemical reaction pathways. By enhancing the algorithm and adding DECIMER, we hope to make the system more reliable and useful for researchers.

References

1. D. Weininger, "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules," J. Chem. Inf. Comput. Sci., vol. 28, no. 1, pp. 31-36 (1988). https://doi.org/10.1021/ci00057a005
2. Rajan, K., Zielesny, A., Steinbeck, C.: DECIMER: towards deep learning for chemical image recognition. J Cheminform. 12, 65 (2020). https://doi.org/10.1186/s13321-020-00469-w.
3. G. R. Bradski and A. Kaehler, "Contours and connected components", in *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, Inc., Sebastopol, CA, pp. 234–236 (2008).
4. GeeksforGeeks: Binary Cross-Entropy / Log Loss for Binary Classification. `https://www.geeksforgeeks.org/binary-cross-entropy-log-loss-for-binary-classification/`. Accessed: January 24, 2025.
5. Jason Brownlee: Adam Optimization Algorithm for Deep Learning. `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`. Accessed: January 24, 2025.
6. G. R. Bradski and A. Kaehler, "Hough Line Transform", in *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, Inc., Sebastopol, CA, pp. 156–158 (2008).
7. G. R. Bradski and A. Kaehler, "Canny Edge Detection", in *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, Inc., Sebastopol, CA, pp. 149–153 (2008).