Myotube Detection Model

Roberto Angel Rillo Calva[1 [A01642022]], Jacob Valdenegro Monzón[2,3 [A01640992]], Carlos Dhali Tejeda Tapia[3 [A00344820]], and Enrique Mora Navarro[4 [A01635459]]

Instituto Tecnologico y de Estudios Superiores de Monterrey, México.

Abstract. The accurate detection and quantification of myotubes in microscopic images is essential for studying the effects of variables, such as blood alcohol concentration, on their formation and development. Manual labeling of myotubes is labor-intensive and prone to inconsistencies, motivating the development of an automated solution. This paper presents a YOLOv11-based object detection model designed to identify and count myotubes in microscopic images efficiently.
The methodology involved collecting a dataset of myotube images from both public sources and custom experiments, followed by manual annotation using the *Labelme* tool. Labels were created using polygon annotations for precise alignment with the irregular shapes of myotubes. These annotations were converted into the YOLO format for training. YOLOv11 was selected because of its support for oriented bounding boxes (OBB), which proved crucial for accurately detecting the varying shapes and orientations of myotubes. Model training was performed using a custom split of the dataset, and performance was evaluated using metrics such as precision, recall, and F1 score, with a focus on balancing detection accuracy and minimizing false positives and negatives.
The refinement of the model was driven by the analysis of precision and recall trends. If performance indicated potential for improvement, training epochs were increased or data augmentation techniques were adjusted to enhance model generalization. The final model demonstrated reliable detection of myotubes across unseen images, offering a significant improvement in efficiency and consistency over manual annotation. This work highlights the potential of modern object detection techniques in automating complex biological image analysis tasks, providing a foundation for future studies on myotube behavior under various experimental conditions.

Keywords: YOLO · Myotubes · Model.

1 Introduction

1.1 Myotubes

Myotubes are multinucleated structures formed during the differentiation of myoblasts, serving as a fundamental step in the development of muscle tissue. Understanding the mechanisms and factors that influence myotube formation and

development is critical for advancing research in muscle regeneration and the effects of various substances on muscle health. One such factor under investigation is the concentration of alcohol in the bloodstream, as it has been hypothesized to disrupt or alter myotube formation.

The primary objective of this study is to determine whether varying levels of alcohol concentration impact the morphology and development of myotubes. To achieve this, a detailed analysis of myotube characteristics in cell culture images is required. However, a significant challenge lies in the need to manually label images to identify and quantify myotubes. This labor-intensive process not only slows down research but also introduces potential variability in the analysis.

To address this issue, we propose a myotube segmentation model. By automating the identification and segmentation of myotubes in microscopy images, this model aims to streamline the analysis process, ensuring consistency and significantly reducing the time required for data processing. This study presents the development and validation of the segmentation model, highlighting its potential to facilitate large-scale analysis of the effects of alcohol on myotube formation.

2 Methodology

In our study, we developed a model focused on object detection to address the challenge of identifying and quantifying myotubes in microscopy images. While image segmentation techniques offer pixel-level precision and enable detailed shape analysis, our primary objective was centered on detecting and counting the myotubes rather than analyzing their intricate shapes.

To achieve this, we implemented an object detection model that generates bounding boxes around each identified myotube. These bounding boxes provide sufficient granularity for our purpose, allowing us to determine the number of myotubes in an image accurately. By focusing on object detection, we streamlined the computational requirements and ensured that the model delivers consistent results for this specific task.

This approach aligns with our overarching goal: to develop a reliable tool that automates the process of myotube detection, facilitating large-scale analysis of factors such as the impact of alcohol concentration on their formation and development.

2.1 Data Collection

For data collection, we were granted access to a NAS, where the project partner uploaded microscopy images corresponding to findings collected during a specific time interval. These images were intended to observe the development of myotubes. However, the large volume of images, poor folder organization, and variability in the stages of myotube formation made the process challenging. Many folders did not contain relevant images or lacked myotube-related findings. Consequently, an initial manual data review was conducted to identify dates with a higher likelihood of containing useful images. Subsequently, a

Python script was developed to automate the process by downloading specific folders from the NAS to Google Drive, filtering those containing relevant images and discarding empty or irrelevant ones.

2.2 Image Labeling

After collecting a sufficient number of images for our study, we proceeded to label them in preparation for training the model. For this task, we required a robust and efficient labeling tool. Initially, we evaluated several options; however, we ultimately selected Labelme due to its unique advantages.

Labelme is a lightweight, open-source annotation tool designed for labeling images, particularly for object detection and segmentation tasks. It supports a wide range of features, including polygonal annotations, which are particularly useful for defining irregular shapes such as myotubes. Additionally, Labelme allows users to export annotations in multiple formats, making it versatile and compatible with various machine learning frameworks.

Another significant factor in choosing Labelme was its ability to run locally, ensuring the confidentiality of our project data—a critical requirement for this study. Its user-friendly interface and straightforward installation process further streamlined our labeling workflow, allowing us to maintain efficiency while achieving high-quality annotations.[1]

For labeling our images, we utilized the polygon tool in Labelme. This tool proved to be highly versatile, enabling us to manually adjust the annotations to closely match the irregular shapes of the myotubes. This flexibility significantly accelerated the labeling process while maintaining a high level of accuracy.

Initially, we labeled all structures we believed to be myotubes, guided by our initial session with Dr. K, where we learned to identify their visual characteristics. As we progressed, Dr. K's feedback and comments helped refine our understanding of what should and should not be considered a myotube. This iterative process greatly improved the quality of the annotations.

However, it is important to acknowledge that the labeling process is not without limitations. Even experts in the field, like Dr. K, occasionally encountered uncertainties regarding the identification of myotubes in certain images. This underscores the abstract and variable nature of myotube morphology, which can present significant challenges in consistent labeling.

As you can see in the example given in Figure 1 we tried to stay as loyal to the myotube shape as we could with the polygon tool, and we tried to catch every single myotube in the image.

2.3 Data Augmentation

During the labeling process, we successfully annotated approximately 200 images. This marked significant progress, as it provided a sufficient starting point for initial training. However, we recognized that 200 images would not be enough to achieve reliable results with our object detection model. Given the time constraints, labeling additional images was not feasible. To address this challenge,

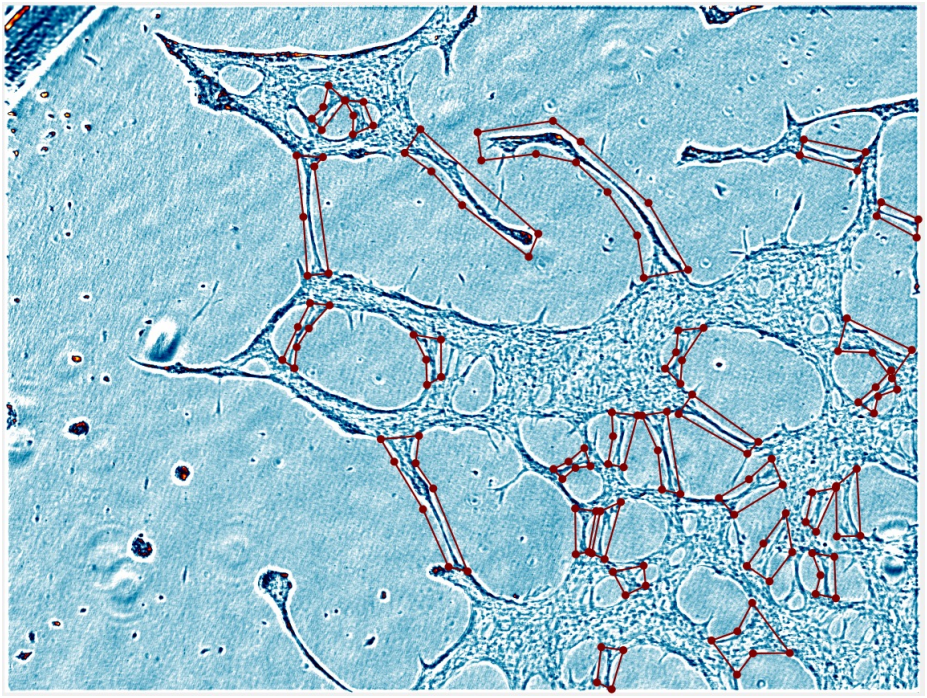

Fig. 1. Example of labeling in an image.

we explored alternative solutions and discovered data augmentation—a powerful technique for artificially expanding datasets by applying transformations to existing images, as illustrated in Figure 2 and 3. For this transformation, we used the OpenCV Python library, which allowed us to efficiently modify image properties and coordinates.

OpenCV (Open Source Computer Vision Library) is an open-source library designed for computer vision and image processing tasks. This Library supports a variety of operations crucial for data augmentation, such as geometric transformations (e.g., rotations, scaling, translations, flips), color adjustments, and noise injection. Its efficient algorithms enable real-time processing and is compatible with multiple programming languages.

Data augmentation leverages existing data to create new, modified samples that enrich the dataset, improving model optimization and generalization. This process reduces overfitting and enhances the model's ability to generalize to unseen data. By introducing variations to the training data, models are exposed to a broader range of scenarios, ultimately improving their robustness and performance.

The essence of data augmentation lies in transforming pre-existing data rather than creating synthetic data from scratch. Unlike synthetic data, which generates entirely artificial samples, augmented data modifies copies of real-

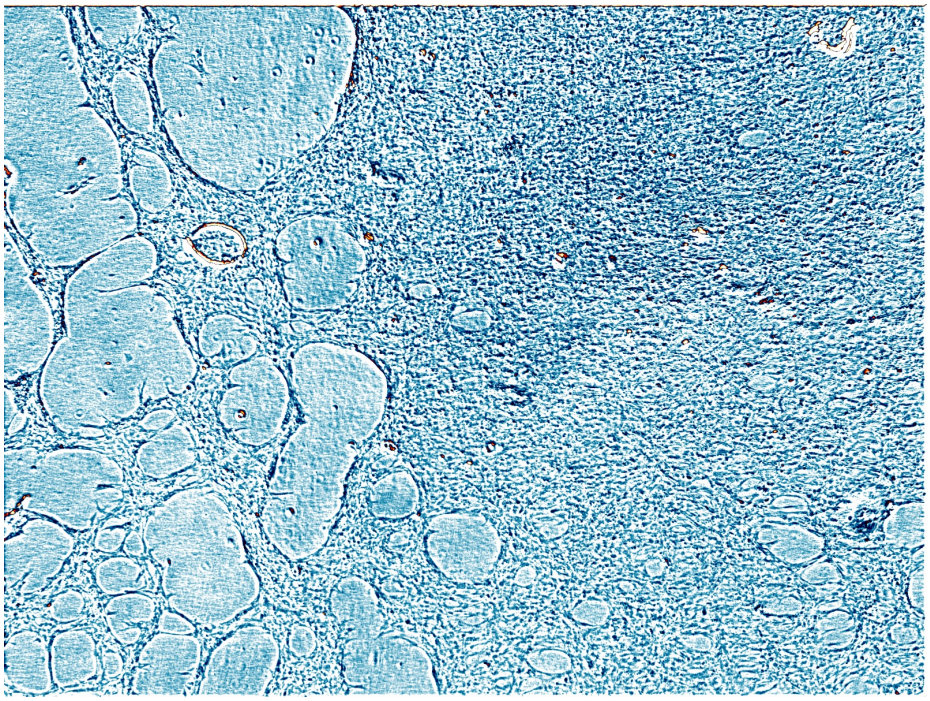

Fig. 2. Original image.

world data, increasing diversity and dataset size without departing from the original distribution. Common transformations include geometric changes (e.g., rotations, flips, cropping) and photometric adjustments (e.g., brightness, contrast, or noise injection).

By applying data augmentation techniques, we tripled the size of our dataset, significantly enhancing its diversity. This augmentation was a critical step, enabling us to train the object detection model more effectively and achieve better results. The expanded dataset included variations that simulated real-world conditions, ensuring our model could handle a wide range of scenarios.[3]

2.4 Dataset Conversion

Our dataset initially consisted of all the images and their corresponding JSON label files, stored together in a single folder. While this format was suitable for annotation and initial inspection, it was not directly compatible with the requirements of the YOLO object detection framework. YOLO requires labels to be in a specific format: each image must have an associated .txt file containing the annotation data, with bounding boxes defined in normalized coordinates. Additionally, the dataset needs to be organized into separate folders for training

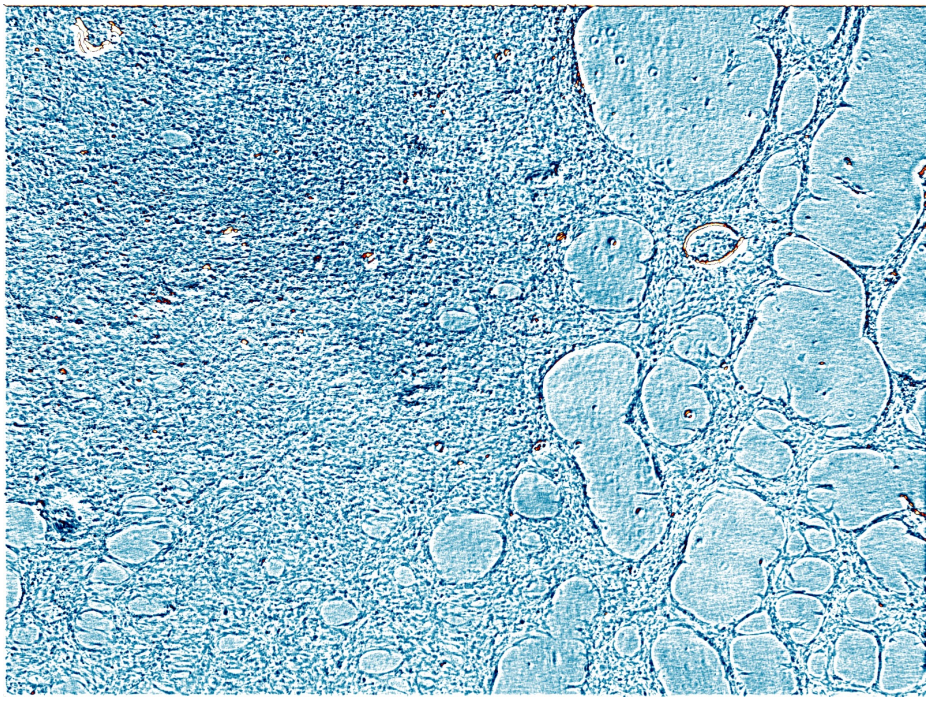

Fig. 3. Horizontal flipped image.

(train) and validation (val) subsets to streamline the training process.

Setting Up Input and Output Directories

The code defines the input directory where the labeled images and their associated JSON annotation files are stored. It also specifies an output directory where the converted YOLO format data will be saved. Within the output directory, subdirectories are automatically created for training and validation datasets, separated into folders for images (images/train, images/val) and labels (labels/train, labels/val).

Defining Dataset Split Ratio

A training-validation split ratio is set, which specifies that 80 percent of the dataset will be used for training, while the remaining 20 percent will be reserved for validation.

Loading and Shuffling JSON Files

The script identifies all JSON files in the input directory and randomly shuffles them to ensure unbiased data distribution. The shuffled files are then divided into training and validation sets based on the predefined ratio.

Normalizing Points for Bounding Boxes

A helper function, normalizePoints, scales the coordinates of bounding box points to a relative scale between 0 and 1, using the dimensions of the image (imageWidth and imageHeight). This is required for YOLO's coordinate format.

Processing Each JSON File

For each annotation file:

The JSON file is opened, and metadata such as image dimensions and bounding box details are read. Bounding boxes defined by four points (quadrilaterals) are normalized and flattened into the format required for YOLO OBB. Each box is associated with a label, represented as 0 for simplicity (assuming a single class).

Saving Labels in YOLO Format

The processed annotations are written to .txt files in the appropriate subdirectory (labels/train or labels/val).

2.5 Model Training

The YOLO Model

In this study, we selected YOLOv11 (You Only Look Once) as the foundation for our object detection solution. YOLO is renowned for its ability to perform real-time object detection with high accuracy and efficiency, processing images in a single forward pass through the network.

Initially, we conducted tests using older versions of YOLO. However, upon discovering that YOLOv11 was the latest iteration and introduced support for Oriented Bounding Boxes (OBB), we decided to transition to this version. OBB is a transformative feature, allowing the model to detect and represent objects with rotated bounding boxes rather than standard rectangular ones. This capability is particularly valuable in applications like ours, where the orientation and shape of myotubes can vary significantly.

The addition of OBB in YOLOv11, combined with its already excellent performance in terms of speed and accuracy (as we can see in the Figure 4 it outperforms the older YOLO models), proved to be a game-changer. It enabled us to achieve a significant performance boost and enhanced the precision of our detection tasks, particularly in cases where myotubes were irregularly shaped or oriented.[2]

We selected YOLOv11 over other available models due to several key advantages:

Quickness: YOLO's real-time detection capabilities make it an ideal choice for projects requiring efficiency in both training and inference.

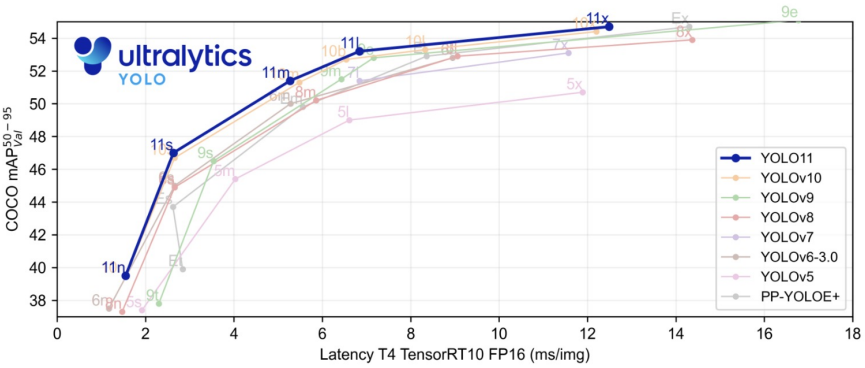

Fig. 4. Yolo Models Performance Test.

Ease of Use: The model's well-documented implementation and compatibility with various platforms simplify setup and training.

Popularity: As one of the most widely used object detection models, YOLO has an active community, offering extensive resources and support for troubleshooting and optimization.

Training Process

To train the YOLO model, we followed a systematic workflow:

Local Installation:
We installed YOLOv11 locally on our system, ensuring full control over the training environment. This setup allowed us to customize the training process and leverage local computational resources efficiently.

Dataset Preparation:
Our labeled dataset, divided into training, validation, and test subsets, was organized and placed in the designated folder structure within the YOLO environment. This structure ensures seamless compatibility with the model's input requirements.

Training Execution:
Using Anaconda, we executed a custom training script to initiate the process. This script utilized the YOLO framework to read the dataset, adjust model hyperparameters, and start the training loop. During training, the model optimized its weights to detect and count myotubes accurately.

The training process culminated in generating the model's best.pt file, representing the weights and configuration of the model after achieving optimal performance on the validation set. This trained model was subsequently used for inference on new images to evaluate its real-world applicability and accuracy.

2.6 Inference and Detection

After completing the training process, we utilized the best.pt file, which contains the optimized weights and configuration of the YOLOv11 model, to perform inference. For this phase, we tested the model on a set of images that were not included in the training or validation datasets.

This approach allowed us to evaluate the model's ability to generalize and accurately detect myotubes in previously unseen data.

By employing these unseen images, we ensured that the evaluation reflected the real-world applicability of the model, reinforcing its reliability and effectiveness for automating the myotube segmentation process.

2.7 Performance evaluation

When training a model using YOLO, the framework generates a *results folder* that provides a comprehensive set of evaluation metrics. These include **precision**, **recall**, **mean Average Precision at 50% Intersection over Union (mAP50)**, **box loss**, and a **confusion matrix**, among others. While these metrics offer a thorough analysis of model performance, we focused on the ones most relevant to our objective: understanding how effectively the model recognized myotubes.

Selected Metrics

Precision Precision measures the proportion of correctly identified myotubes among all the objects detected by the model.

$$\textbf{Precision} = \frac{\textbf{True Positives}}{\textbf{True Positives} + \textbf{False Positives}} \tag{1}$$

In our context, high precision indicates that the model minimizes false positives, i.e., it rarely misidentifies non-mytoubes as myotubes. This is crucial to ensure that the detected myotubes are reliable and accurate for further analysis.

Recall Recall assesses the proportion of actual myotubes correctly identified by the model.

$$\textbf{Recall} = \frac{\textbf{True Positives}}{\textbf{True Positives} + \textbf{False Negatives}} \tag{2}$$

In the context of myotubes, high recall ensures that the model is capturing as many true myotubes as possible, minimizing the risk of missing myotubes in the images.

F1-Score The F1-score provides a single metric that balances precision and recall, offering a comprehensive view of the model's overall performance.

$$\textbf{F1-Score} = 2 \times \frac{\textbf{Precision} \times \textbf{Recall}}{\textbf{Precision} + \textbf{Recall}} \tag{3}$$

For our project, the F1-score was chosen as the primary metric to summarize the model's performance. A high F1-score indicates that the model achieves a good balance between correctly identifying myotubes and avoiding false detections. This makes it an ideal metric to evaluate the model's effectiveness in detecting and quantifying myotubes in images.

By focusing on these metrics, we tailored our evaluation to align with the project's specific goals: ensuring accurate detection of myotubes and avoiding both missed detections and false positives. The precision and recall metrics allowed us to assess these aspects independently, while the F1-score provided a concise measure of the model's overall reliability.

2.8 Model Refinement

After evaluating the metrics of the model, we focused primarily on the **precision** and **recall** graphs to guide the refinement process. These graphs provided critical insights into the model's learning behavior and its capacity to detect myotubes effectively.

If, after a 100- or 200-epoch training, the precision and recall graphs indicated upward trends as we can see in Figure 5 (suggesting the model was still improving), we adjusted the hyperparameter for the number of epochs. By increasing the number of epochs, we allowed the model to train further and monitored how the metrics evolved over the extended training.

On the other hand, if the precision and recall graphs appeared stable or began to decline, this suggested the model had reached its performance plateau or was starting to overfit. In such cases, we explored alternative strategies, such as modifying the data augmentation process. For instance, we experimented with different transformations applied to the training dataset to introduce more variability and potentially enhance the model's generalization capabilities.

This iterative process of fine-tuning hyperparameters and exploring data augmentation techniques was repeated until we achieved our final model. The resulting model exhibited an optimal balance between precision and recall, effectively meeting the requirements of our project.

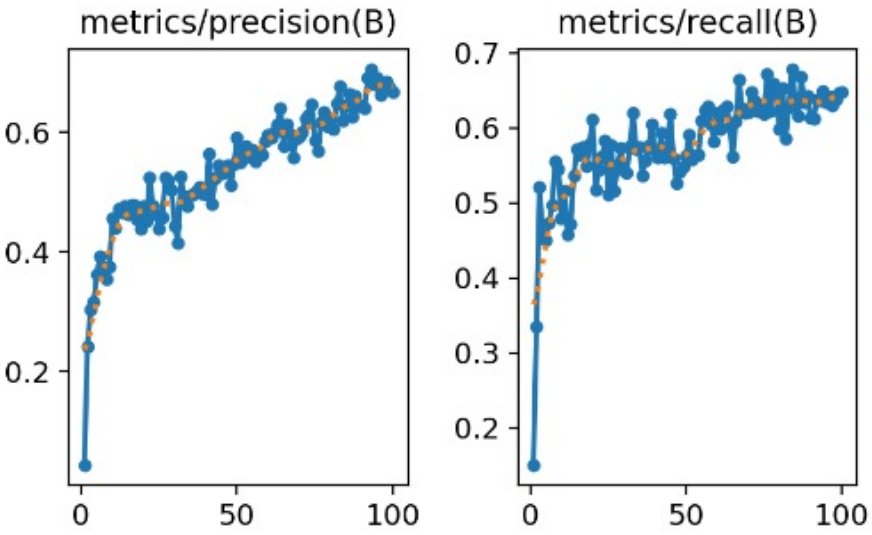

Fig. 5. Model Training Graph Tendency.

3 Experiments

This section describes the experiments performed with two different versions of the YOLO model for myotube detection. Each experiment included training the model with a labeled dataset using LabelMe and evaluating performance metrics such as precision, recall, and F1-score.

3.1 YOLOv5

Since we had no previous experience using YOLO, we decided to use version 5 for initial training. This model was trained with 100 epochs using the prepared dataset. Although the detections were functional, problems were observed with the bounding boxes, since they were very large compared to the myotubes or in some cases they were overlapped, which limited the accuracy of the predictions. The results are summarized in Table 1.

Table 1. Results of the experiment with YOLOv5

Metrics	**Precision**	**Recall**	**F1-score**
Value	0.45	0.45	0.45

A test image with the detections generated by this model is presented in Figure 6.

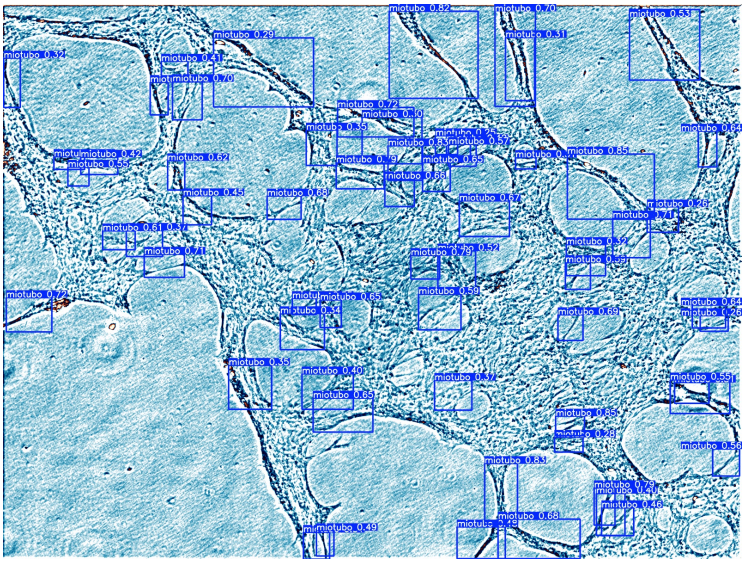

Fig. 6. Example of detections with YOLOv5.

3.2 YOLOv11 with Oriented Bounding Boxes (OBB)

In the second experiment, we decided to test YOLOv11 (OBB), which allows the Bounding Boxes to be adjusted to the rotation of the myotubes and helped solve the problem we had using the previous version. This model was trained again with 100 epochs, achieving significantly better results compared to YOLOv5, as shown in Table 2.

Table 2. Results of the experiment with YOLOv11 OBB

Metrics	Precision	Recall	F1-score
Value	0.70	0.63	0.67

An example of detection performed by this model is presented in Figure 7, where the Bounding Boxes aligned with the myotubes can be observed.

4 Results

Our myotube detection automation model was highly successful in labeling. The integration of **Oriented Bounding Box (OBB)** and **YOLOv11** significantly enhanced our labeling process, achieving precise and efficient classification. Within our latest and most robust model iteration, precision reached **90% (0.9)** and recall achieved **87% (0.87)**, marking substantial progress from our latest model.

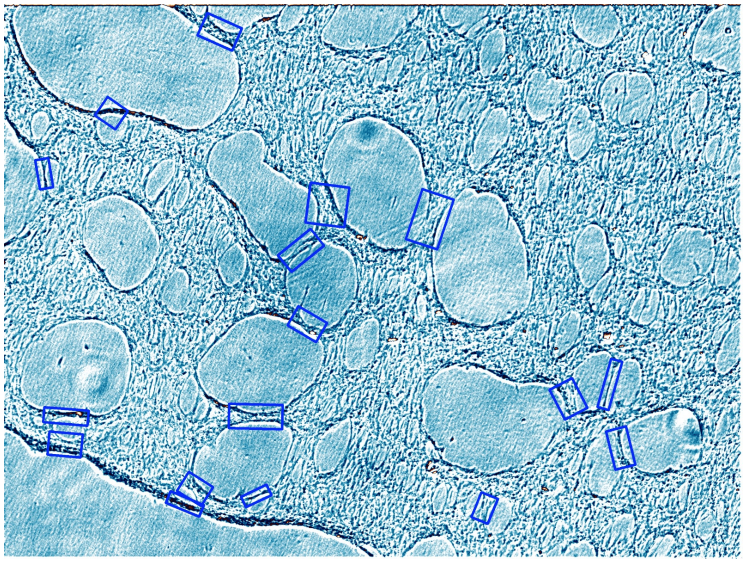

Fig. 7. Example of detections with YOLOv11 OBB.

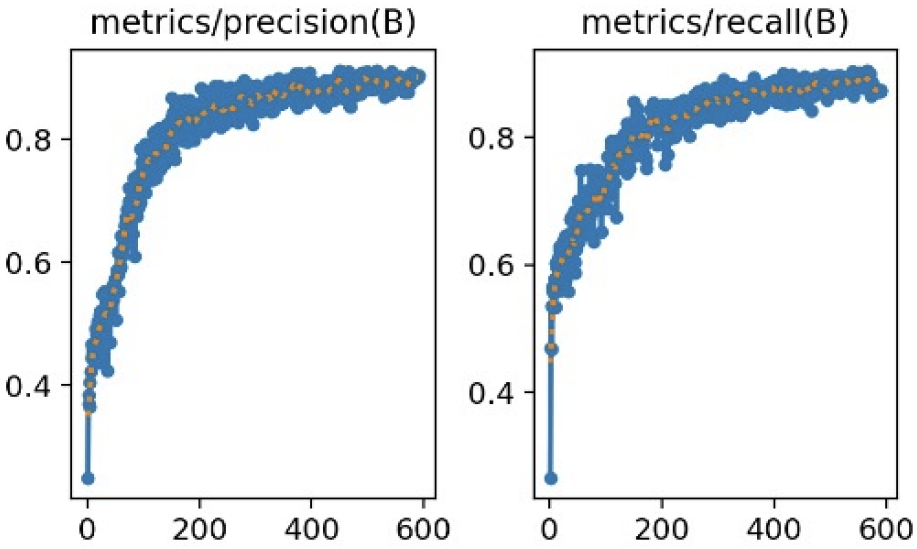

Fig. 8. Precision and Recall Metrics Chart

4.1 Model Growth

The image below demonstrates the evolution of our models in efficiency and precision. Starting with **YOLOv5**, we observed steady improvement as we transi-

tioned to **YOLOv11**. The most significant gains were realized when we combined **data augmentation techniques** with **YOLOv11**, resulting in a substantial boost in both precision and recall.

5 Conclusions

Our findings highlight the necessity of labeling additional images to further train and enhance the robustness of our model. To achieve this, **800 additional images** should be labeled and incorporated with the **data augmentation techniques** employed in our latest model.

Close collaboration between domain experts and technical teams is crucial to ensuring accurate labeling and identifying potential errors early. Supervised labeling sessions will minimize mistakes and maintain the integrity of the dataset.

To streamline model testing, we will deliver a user-friendly **UI** that simplifies and accelerates the testing process. Additionally, implementing comprehensive observability practices will allow us to monitor **key data quality metrics**, ensuring the sustained growth of the model and mitigating risks of hallucinations or inaccuracies.

As a team, we firmly believe that with the appropriate knowledge and techniques, **image classification presents a tremendous business opportunity** for computer scientists and industry professionals alike. Our experience applying computer vision and AI techniques to real-world challenges in **biology** has been particularly rewarding, showcasing the immense potential of this technology in addressing complex, impactful problems.

References

1. Labelme. Private, Flexible AI Dataset Creation with Offline App | Labelme.
2. Shaoni Mukherjee. What's new in YOLOV11 Transforming Object Detection Once Again Part 1 | DigitalOcean, 10 2024.
3. Jacob Murel, PhD and Eda Kavlakoglu. Data-augmentation, 8 2024.

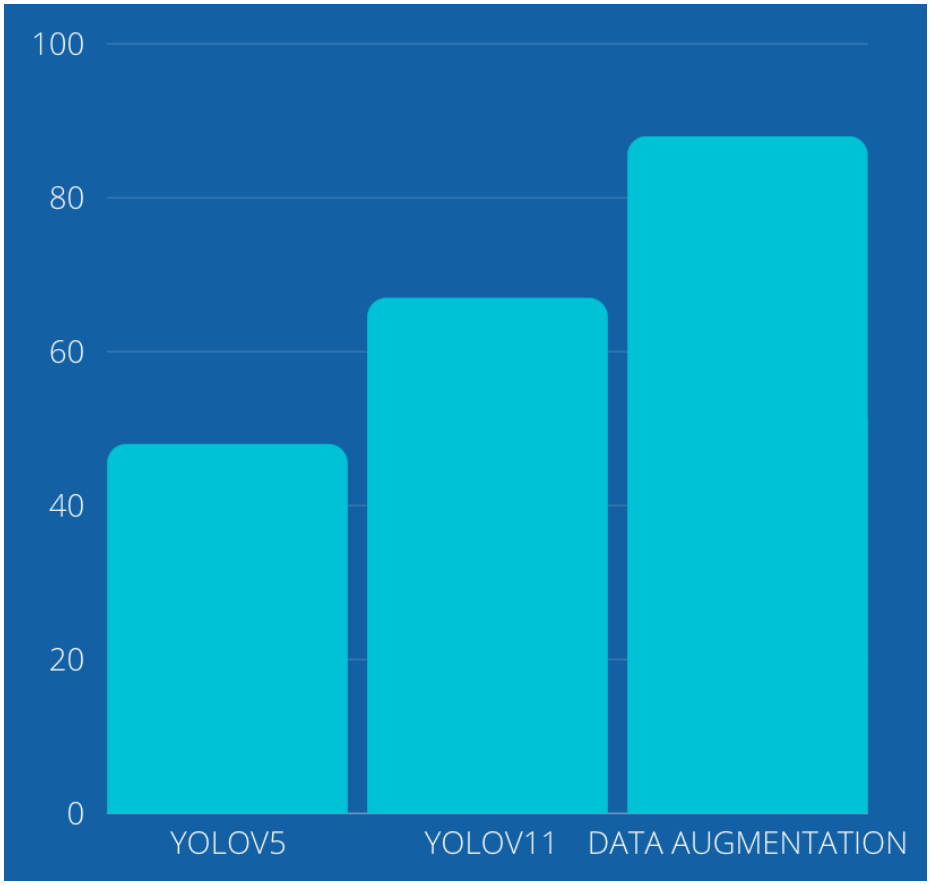

Fig. 9. Model Efficiency and Precision Growth Chart