Table Recognition in Images: Extracting Information and Saving it to an Excel Table

1. Abstract

In this project, I developed a system for detecting tables in images and converting their content into structured Excel files. I used the Ultralytics library and YOLO models to automate the process of table detection and content extraction from a variety of image types. To build the foundation of the system, I leveraged pre-trained YOLO models and a dataset from Roboflow.

To enhance the model's accuracy, I systematically adjusted hyperparameters, including batch size, image size, and the number of training epochs. I evaluated the system's performance using metrics such as F1 score, mean average precision (mAP), precision, recall, and intersection over union (IoU). By analyzing these metrics, I fine-tuned the system to ensure it achieved high accuracy and reliability across diverse scenarios.

2. Dataset Analysis

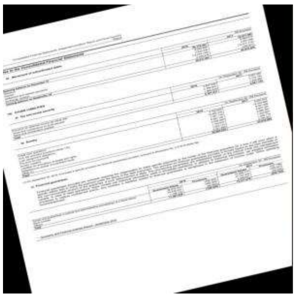
For this project, I used a dataset from Roboflow Universe, which consisted of 827 images and their corresponding text files. The images featured tables with varying characteristics, such as different languages, colours, fonts, and font sizes. The tables were annotated in the YOLOv9 format, providing bounding box information for each detected table.

To prepare the dataset for model training, I applied several preprocessing and augmentation techniques. I resized all images to 640x640 pixels, stretching them to fit this resolution. Additionally, I performed data augmentation by generating three additional versions of each image, applying random rotations within a range of -15° to +15°.

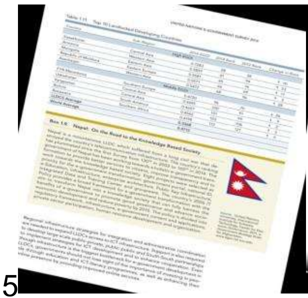
I divided the dataset into three subsets: a training set, a test set, and a validation set. The training set contained 726 images and their corresponding annotation files. Each annotation included a column specifying the table type class (0 for bordered tables, 1 for borderless tables, and 2 for cell tables), along with image size coefficients. The test set comprised 37 images and annotation files, while the validation set included 64 images and annotation files.

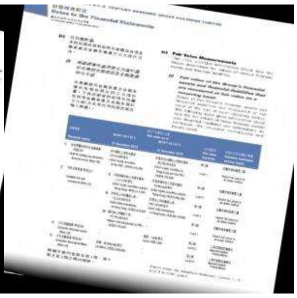
Here are some sample images from the Roboflow dataset:

- Train set



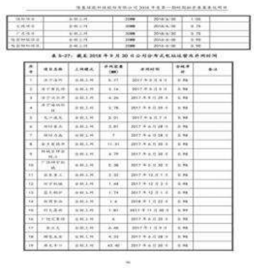
1.5

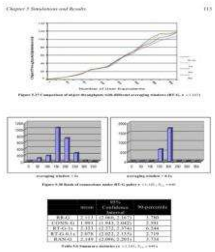




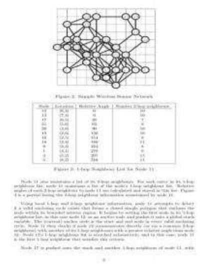
1.5

- Validation set





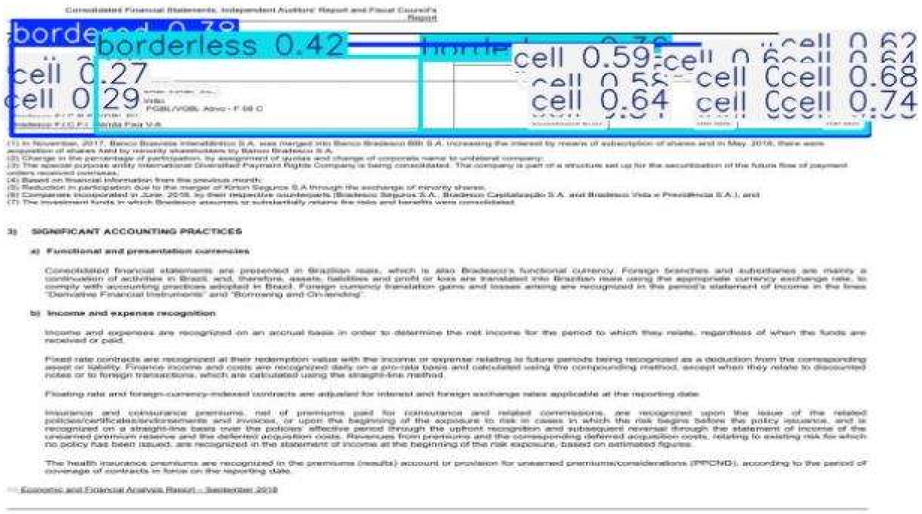
- Test set



3. Model Selection

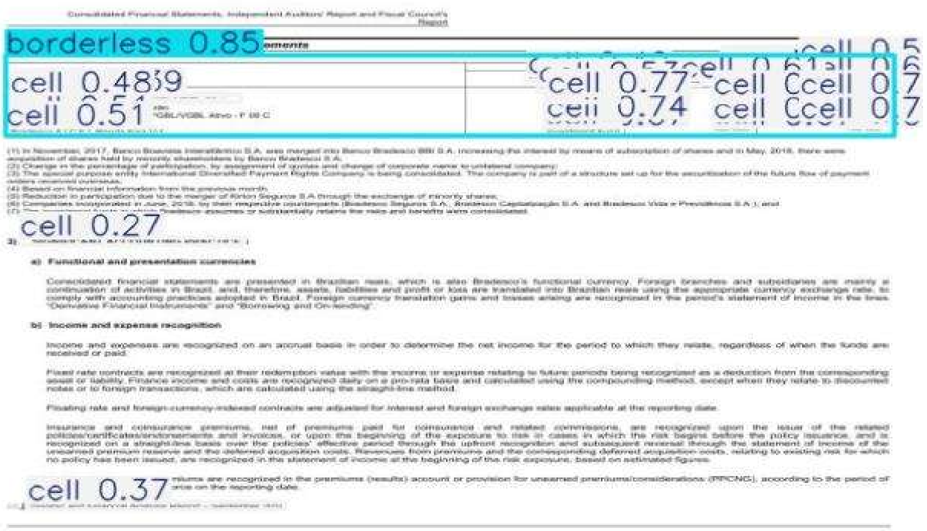
Experiment 00

In my first experiment, I chose the YOLOv5n.pt model. I trained it for 50 epochs with an image size of 640 pixels and a batch size of 32. After the training, I used the predict method to detect the table in the provided image. Below are the results of the table detection.



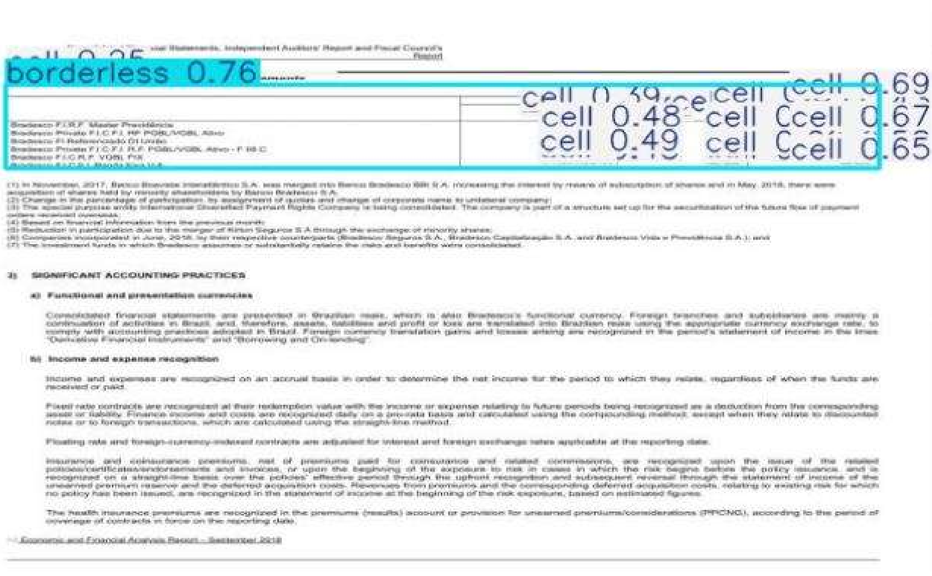
Experiment 01

In my next experiment, I chose the YOLOv5n.pt model. I trained it for 150 epochs with an image size of 640 pixels and a batch size of 32. Below are the results of the table detection.



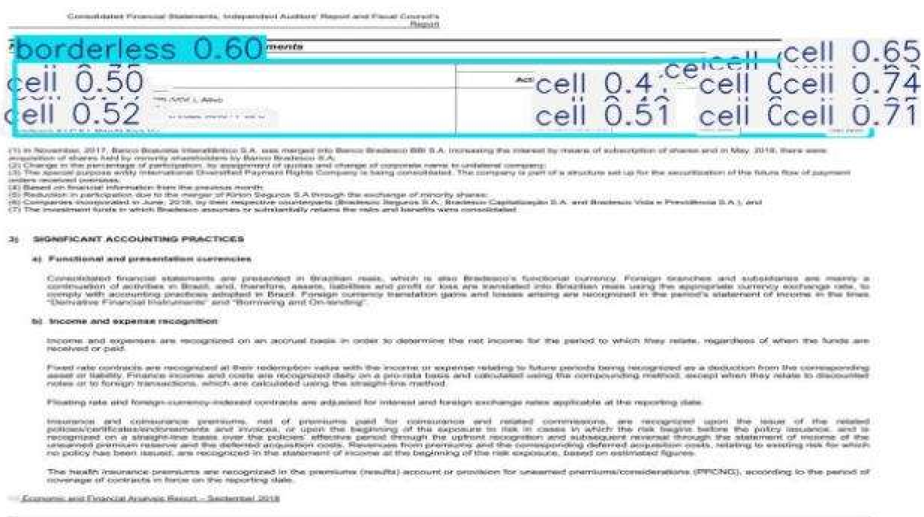
Experiment 02

In my next experiment, I chose the YOLOv8n.pt model. I trained it for 50 epochs with an image size of 640 pixels and a batch size of 32. Below are the results of the table detection.



Experiment 03

In my next experiment, I chose the YOLOv8n.pt model. I trained it for 150 epochs with an image size of 640 pixels and a batch size of 32. Below are the results of the table detection.

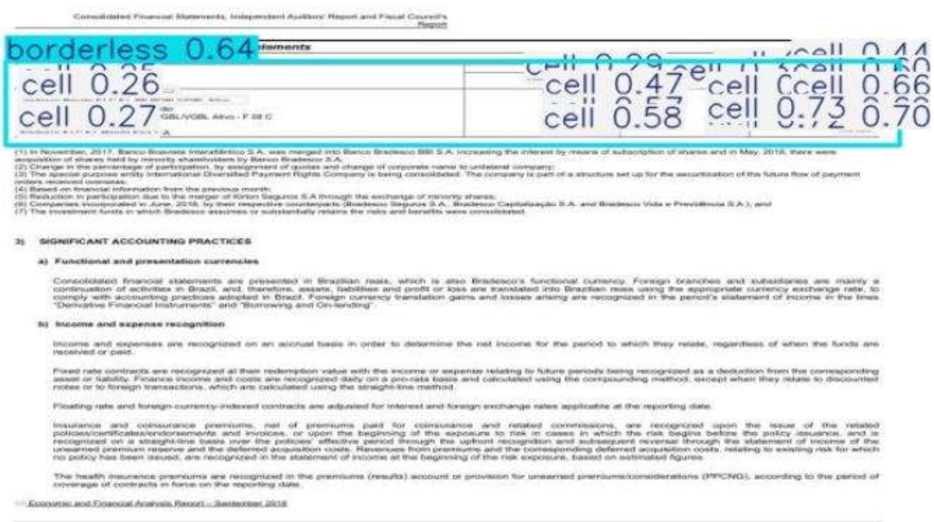


Experiment 04

In my next experiment, I chose the YOLOv8n.pt model. I trained it for 180 epochs with an image size of 416 pixels and a batch size of 32. Below are the results of the table detection.

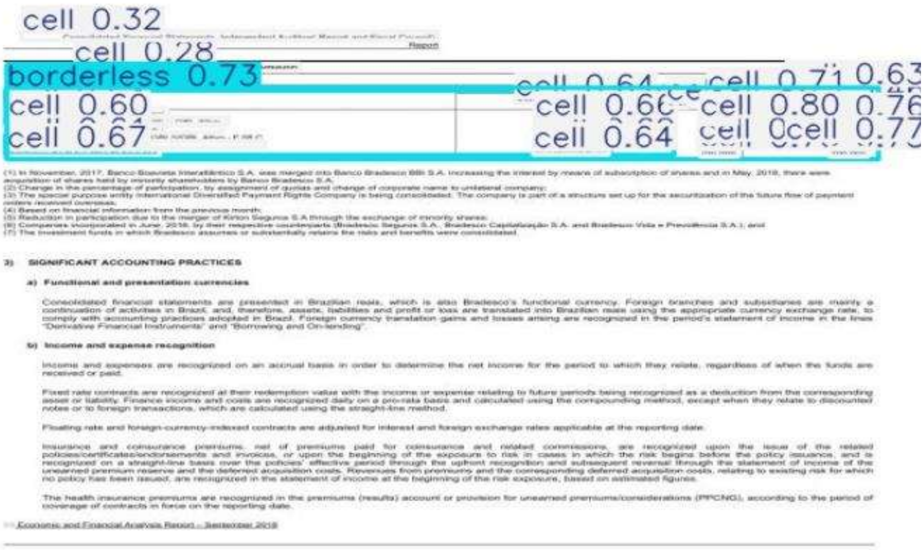
Experiment 05

In my next experiment, I chose the YOLOv8m.pt model. I trained it for 50 epochs with an image size of 640 pixels, a batch size of 16 and optimizer Adam. Below are the results of the table detection.



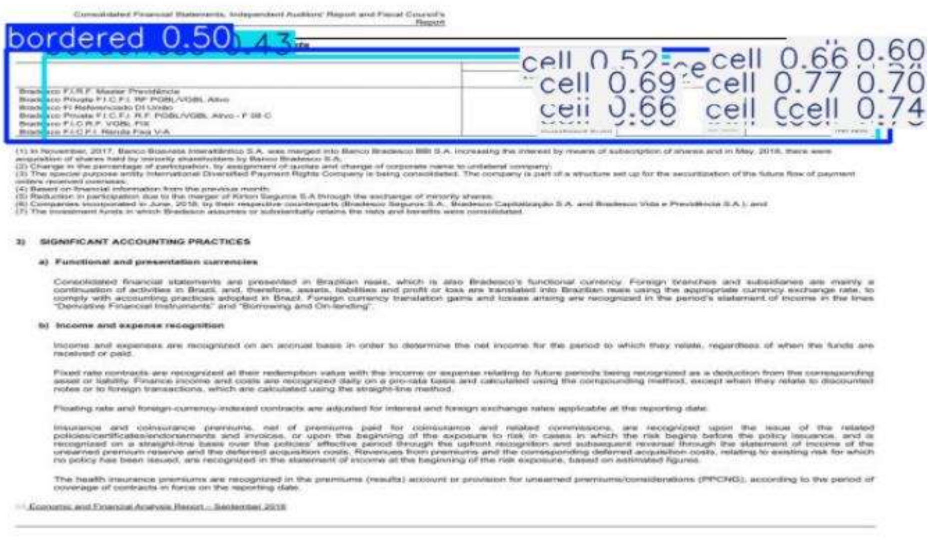
Experiment 06

In my next experiment, I chose the YOLOv8m.pt model. I trained it for 150 epochs with an image size of 640 pixels, a batch size of 16 and optimizer Adam . Below are the results of the table detection.



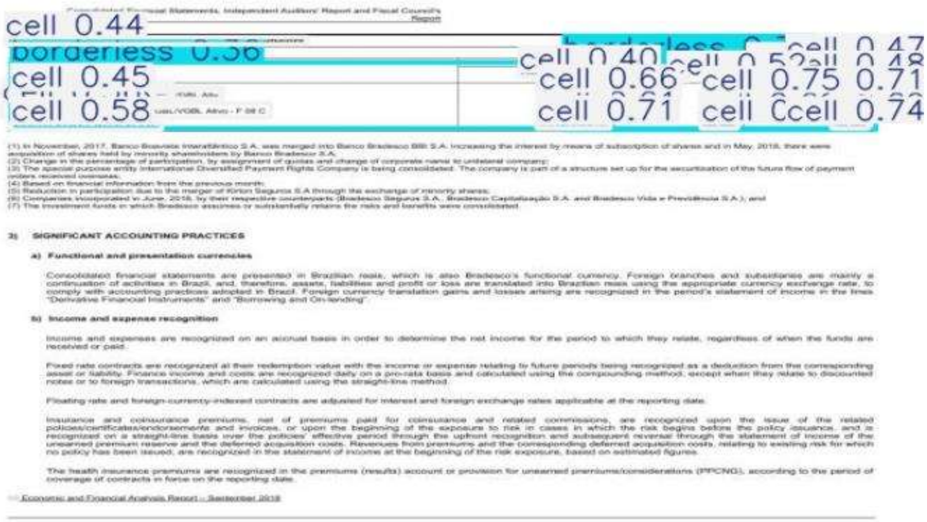
Experiment 07

In my next experiment, I chose the YOLO11n.pt model. I trained it for 150 epochs with an image size of 640 pixels, a batch size of 8 and optimizer Adam . Below are the results of the table detection.



Experiment 08

In my next experiment, I chose the YOLO11s.pt model. I trained it for 150 epochs with an image size of 640 pixels, a batch size of 16 and optimizer Adam . Below are the results of the table detection.



Experiment 09

In my next experiment, I chose the YOLOv8n.pt model. I trained it for 150 epochs with an image size of 640 pixels, a batch size of 16 and optimizer SGD . Below are the results of the table detection.

I created a table summarizing the results of all the experiments.

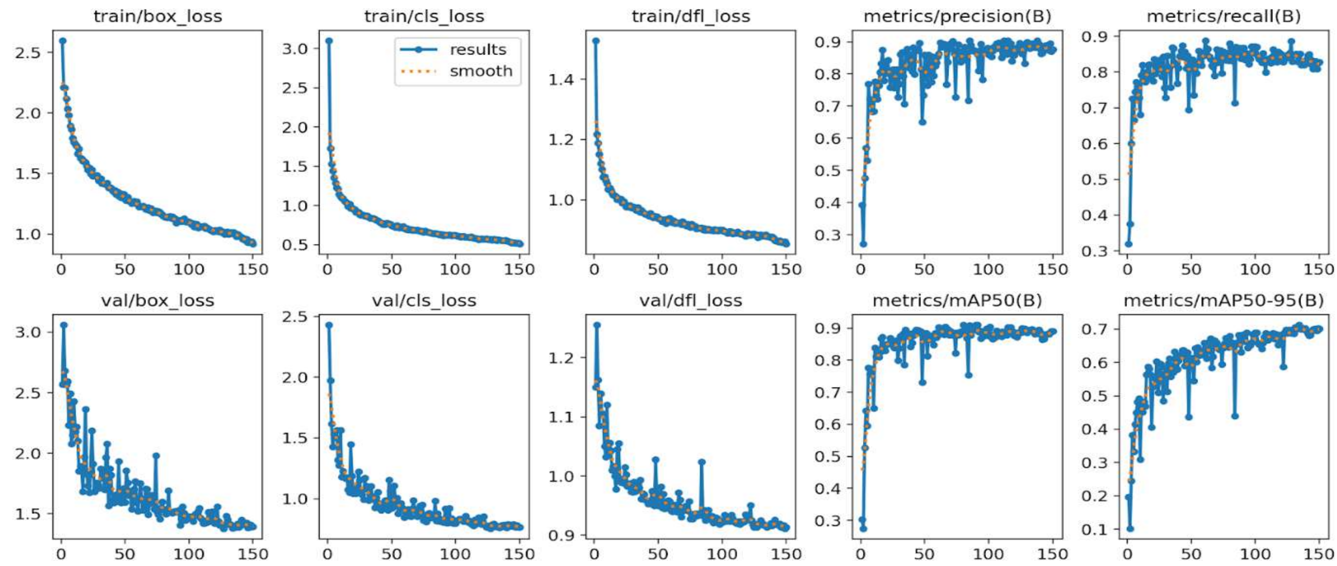
Experiment	Model	Epochs	Image Size	Batch Size	Optimizer	Precision	Recall	mAP50	mAP50-95	Fitness	Preprocess	Inference	Loss	Postprocess
Experiment 00	YOLOv5n.pt	50	640	32	-	0.8794	0.8298	0.8978	0.6498	0.6746	0.31	2.3	0.00027	1.1998
Experiment 01	YOLOv5n.pt	150	640	32	-	0.8794	0.8298	0.8978	0.6498	0.6746	0.31	2.3	0.00027	1.1998
Experiment 02	YOLOv8n.pt	50	640	32	-	0.861	0.85	0.88	0.66	0.68	0.35	2.27	0.00026	2.49
Experiment 03	YOLOv8n.pt	150	640	32	-	0.86	0.86	0.91	0.72	0.74	0.35	2.11	0.00023	1.16
Experiment 04	YOLOv8n.pt	180	416	32	-	0.867	0.81	0.869	0.865	0.7	0.174	0.85	0.00027	1.03
Experiment 05	YOLOv8m.pt	50	640	16	Adam	0.846	0.782	0.856	0.546	0.577	0.2772	10.0226	0.00063	3.8094
Experiment 06	YOLOv8m.pt	150	640	16	Adam	0.852	0.833	0.905	0.656	0.681	0.2719	10.0773	0.00057	4.5999
Experiment 07	YOLO11n.pt	150	640	8	Adam	0.847	0.876	0.910	0.651	0.677	0.3211	5.0640	0.00101	3.8546
Experiment 08	YOLO11s.pt	150	640	16	Adam	0.871	0.827	0.887	0.646	0.670	0.2558	4.8511	0.00066	4.0199
Experiment 09	YOLO8n.pt	150	640	16	SGD	0.894	0.850	0.897	0.712	0.731	0.2226	2.0156	0.00051	2.6220

Based on the analysis of the table, **experiment 09** emerges as the best-performing model.

1. **Precision (0.894)**: The highest precision among all experiments, indicating the model is excellent at minimizing false positives.
2. **Recall (0.850)**: A high recall score, showing the model effectively identifies true positives.
3. **mAP50 (0.897)**: This metric is one of the best in the table, suggesting strong overall detection performance.
4. **mAP50-95 (0.712)**: While not the absolute highest, it balances well with precision and recall.
5. **Fitness (0.731)**: This score reflects an optimized balance of metrics, and it's the highest among all experiments.
6. **Efficiency**: Preprocessing, inference, loss, and postprocessing times are competitive and efficient compared to others, with particularly low inference time (2.0156 seconds).

While some experiments have slightly higher individual scores in certain metrics, experiment 09 provides the best combination of accuracy and efficiency. This balance makes it the optimal choice for detecting tables effectively and efficiently.

4. Final Model Analysis



The image presents various training and validation metrics over 150 epochs for the model

Training Loss (train/box_loss, train/cls_loss, train/dfl_loss):

- The box_loss (bounding box loss) and cls_loss (classification loss) are decreasing steadily over the epochs, which suggests that the model is learning to predict more accurate bounding boxes and classifying objects better.
- The dfl_loss (distribution focal loss) also shows a similar decreasing trend, indicating that the model's ability to handle hard examples and focus on difficult predictions is improving.

Validation Loss (val/box_loss, val/cls_loss, val/dfl_loss):

- val/box_loss and val/cls_loss show a similar trend to the training losses but with more fluctuations, which is common due to the validation set's noise.
- val/dfl_loss follows a similar pattern to the training set and decreases, suggesting that the model generalizes well to the validation data.
- There are occasional spikes in both training and validation loss, which might indicate occasional overfitting or noisy validation data.

Metrics (Precision, Recall, mAP50, mAP50-95):

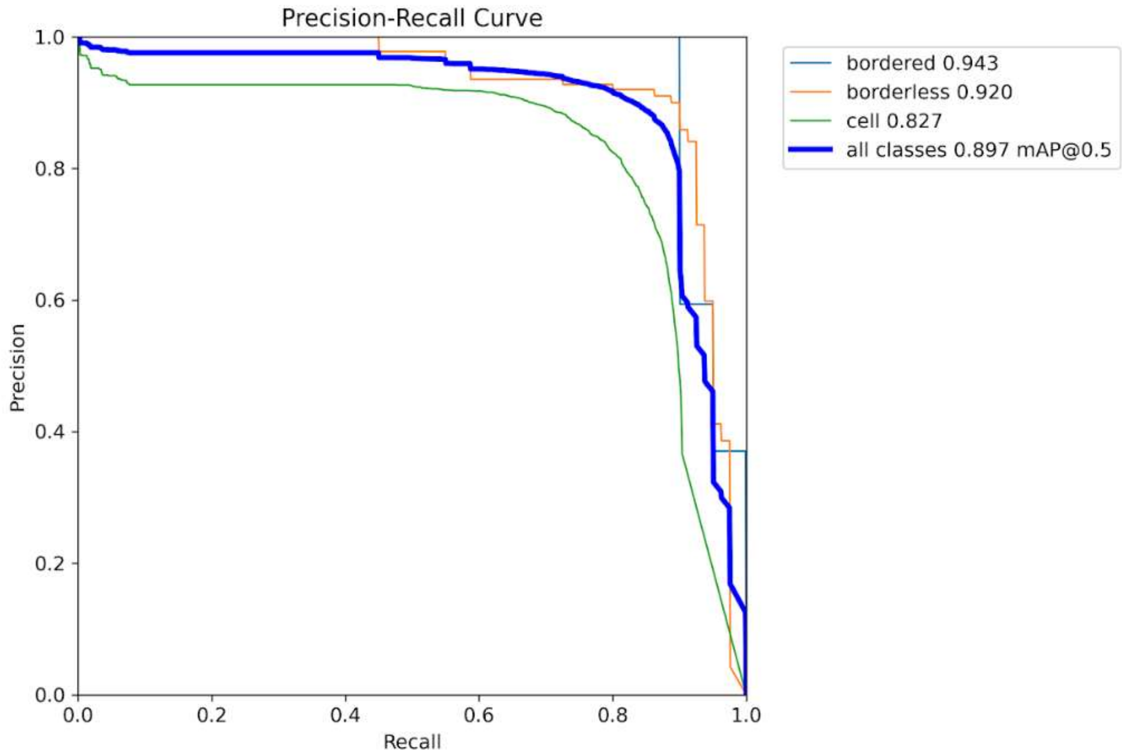
- Precision and Recall are both increasing steadily, with precision reaching a higher plateau, suggesting that the model is getting better at identifying and classifying relevant objects.
- mAP50 (Mean Average Precision at 50% IOU) also increases, indicating that the model's overall accuracy and robustness in detecting tables (or other objects) improves over time.
- mAP50-95 (mean average precision averaged across multiple IOU thresholds) is increasing similarly, meaning that the model is capable of detecting objects with high confidence at different thresholds, ensuring more reliable and precise detections.

Overfitting:

- The plots show smooth decreases in both training and validation losses, with some occasional irregularities. This is a good sign, but the gaps between training and validation metrics might suggest a slight overfitting, especially in the mAP50-95 plot where the gap starts to widen.

Summary:

- Overall, the model seems to be learning effectively, as evidenced by the steady improvement in both precision and recall metrics, as well as mAP scores. The model appears to be overfitting slightly in the final epochs, but the performance is still strong.



The image shows a Precision-Recall (PR) Curve for a model tested on multiple classes, with each class showing different performance.

Precision-Recall Curve Overview:

- The curve represents the trade-off between precision (the proportion of true positive results among all predicted positives) and recall (the proportion of true positives identified among all actual positives).
- A higher precision and recall indicate a better model, particularly in imbalanced datasets where high recall can be important.

Classes:

- Bordered (blue): The precision-recall curve for the bordered class has the highest precision and recall, with a mAP@0.5 of 0.943, suggesting that the model is performing very well in detecting bordered objects. This is the most reliable class for this model.
- Borderless (orange): The precision-recall curve for the borderless class shows slightly lower performance, with a mAP@0.5 of 0.920. This suggests that while the model still

performs well, it is slightly less confident in detecting objects in this class compared to the bordered class.

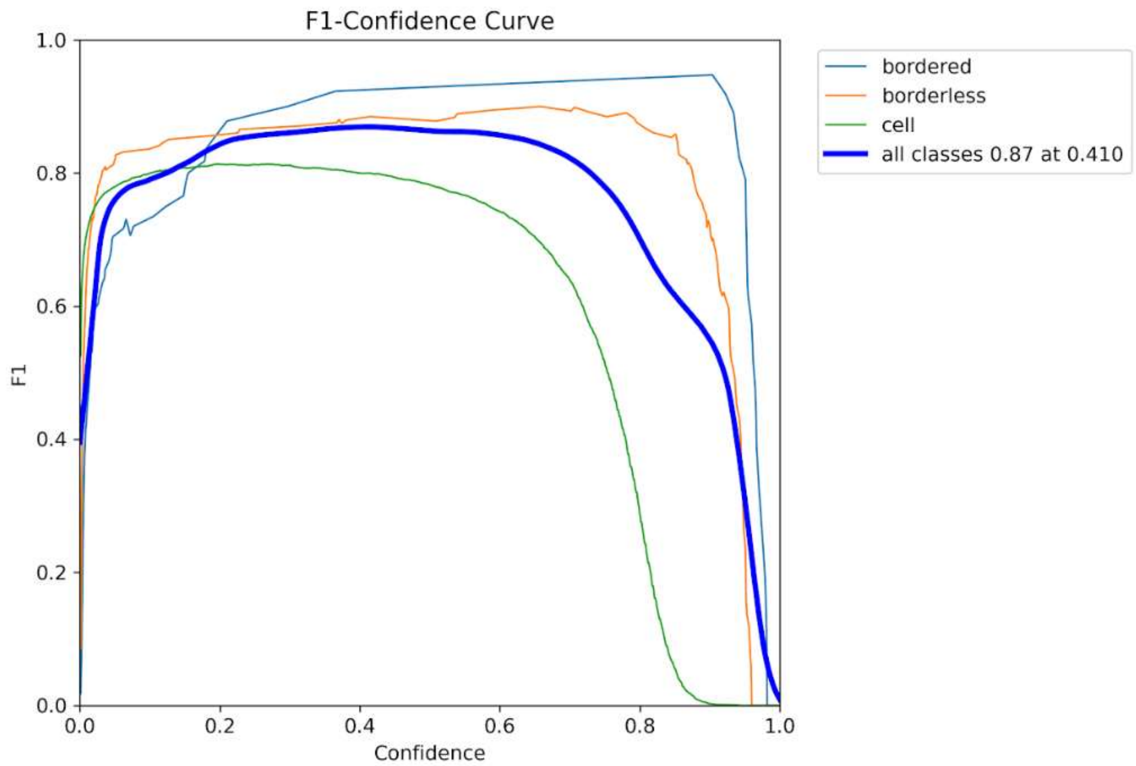
- Cell (green): This class has the lowest precision and recall, with a mAP@0.5 of 0.827. The curve for the "cell" class drops more significantly, which means there are more false positives or missed detections in this category.
- All Classes (blue): The mAP for all classes combined is 0.897, which indicates a generally strong performance across all classes, although it's affected by the lower performance in the "cell" class. The combined PR curve shows the trade-off of precision and recall across the different categories, with a clear drop-off towards the recall side.

Key Insights:

- The bordered and borderless classes perform much better in terms of precision and recall compared to the cell class. This suggests that the model might need improvement on detecting and classifying cells more accurately, which could be due to the dataset or model architecture.
- The all-classes curve indicates that the model is overall decent but has room for improvement, especially for the "cell" class. Fine-tuning or addressing the imbalance between classes could improve overall performance.
- The plot shows that for all classes, as recall increases, precision drops, which is a normal characteristic of precision-recall curves. However, the model still manages a good balance overall, especially for the most important classes.

Conclusion:

- The model performs excellently on the bordered and borderless classes but requires optimization for the cell class. Depending on the application, more data or a tailored approach for the "cell" class could help improve the precision and recall further, boosting the overall mAP.



This image shows an F1-Confidence Curve for a model tested on multiple classes. Here's an analysis based on the plot:

<u>F1-Confidence Curve Overview:</u>

- The F1 score is the harmonic mean of precision and recall, providing a balance between the two metrics. The F1-Confidence Curve shows the F1 score at different confidence thresholds (ranging from 0 to 1).
- As the confidence threshold increases, the model becomes more selective, predicting fewer positives and focusing on higher-confidence predictions.

<u>Classes:</u>

- Bordered (blue): The bordered class has the highest F1 score at lower confidence thresholds, but it drops more sharply as the confidence increases compared to the other classes. It performs well across the confidence range, reaching a strong F1 score, especially in the middle of the curve. This suggests that the model is reliable in detecting this class even with moderate confidence.
- Borderless (orange): The borderless class has a more gradual F1 curve. The F1 score starts lower and increases slowly compared to the bordered class. This indicates that the model is less confident in detecting borderless objects but still performs decently. The model achieves an F1 score comparable to the bordered class, but only at a higher confidence threshold.
- Cell (green): The cell class has the weakest performance, with a much lower F1 score across the confidence spectrum. This suggests that the model struggles to identify this class accurately. The F1 score remains relatively low even at high confidence thresholds, which could be a result of either poor data quality, class imbalance, or insufficient model focus on this class.

- All Classes (blue): The F1 score for all classes combined shows a good balance, with a peak of 0.87. This indicates that the model performs reasonably well across all classes at the optimal confidence threshold of 0.410. However, the model's performance is more heavily influenced by the higher-performing bordered and borderless classes, with the lower performance in the "cell" class pulling down the overall F1 score.

  Key Insights:

- Bordered and Borderless Classes: These classes have good F1 scores, especially in the range of medium to high confidence, indicating the model can effectively identify these objects with relatively high certainty.
- Cell Class: This class has the poorest performance in terms of F1 score, indicating that further model tuning or more data might be needed to improve its detection. The curve for this class never exceeds an F1 score of 0.6, suggesting significant room for improvement.
- All Classes: The overall F1 score of 0.87 at 0.410 confidence threshold indicates that, despite struggles with the "cell" class, the model generally balances precision and recall well. The key to improving the model lies in improving performance on the "cell" class.

  Conclusion:

- The bordered and borderless classes perform strongly, but the model's performance on the cell class needs improvement. The cell class's performance could be addressed by providing more examples of this class, tuning the model, or using techniques such as data augmentation or oversampling for better detection.
- Overall, the model is solid, but improving the detection of the cell class will likely boost its F1 score across all classes.

5. EasyOCR

EasyOCR is a popular Python library for extracting text from images. It simplifies the process of Optical Character Recognition (OCR) by using advanced deep learning techniques. With its support for over 80 languages and straightforward implementation, EasyOCR is widely used for text recognition in images like scanned documents, photographs, and screenshots.

To use EasyOCR, you simply install the library, create a reader object for the desired languages, and process your image to extract the text. It provides results in the form of text, bounding box coordinates, and confidence scores, making it easy to integrate into applications that require text analysis or automation. EasyOCR is particularly useful in tasks like document digitization, license plate recognition, or data extraction from forms.

The result returned by EasyOCR is a list of tuples, where each tuple contains:

- Bounding box coordinates of the detected text (as a list of four points: top-left, top-right, bottom-right, and bottom-left).
- Text that was detected.
- Confidence score representing the model's certainty about the detected text.

Performance Considerations

- Image Quality: The accuracy of OCR depends heavily on the quality of the input image. Clean, high-resolution images yield better results.
- Preprocessing: If the image contains noise or poor contrast, applying preprocessing techniques like resizing, thresholding, or grayscale conversion can improve OCR accuracy.
- Model Optimizations: EasyOCR's underlying models can be fine-tuned or trained on custom data for more specific use cases.

6. OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source library aimed at real-time computer vision and machine learning. It contains over 2,500 optimized algorithms, which can be used for a variety of tasks like detecting and recognizing faces, identifying objects, tracking camera movements, and much more.

The cv2 module in Python is OpenCV's wrapper that allows for seamless integration of OpenCV with Python applications.

Key Features of OpenCV

- Image and Video I/O: OpenCV supports reading and writing images and videos in various formats.
- Image Processing: Includes resizing, blurring, sharpening, thresholding, edge detection, etc.
- Object Detection: Detect faces, objects, and text in images.
- Camera Access: Allows capturing live video from a camera.
- Feature Detection and Matching: Detect and match features such as corners, edges, and contours.
- Machine Learning: Contains algorithms for clustering, regression, and classification tasks.
- 3D Reconstruction: Tools for constructing 3D models from multiple images.

7. Code logic

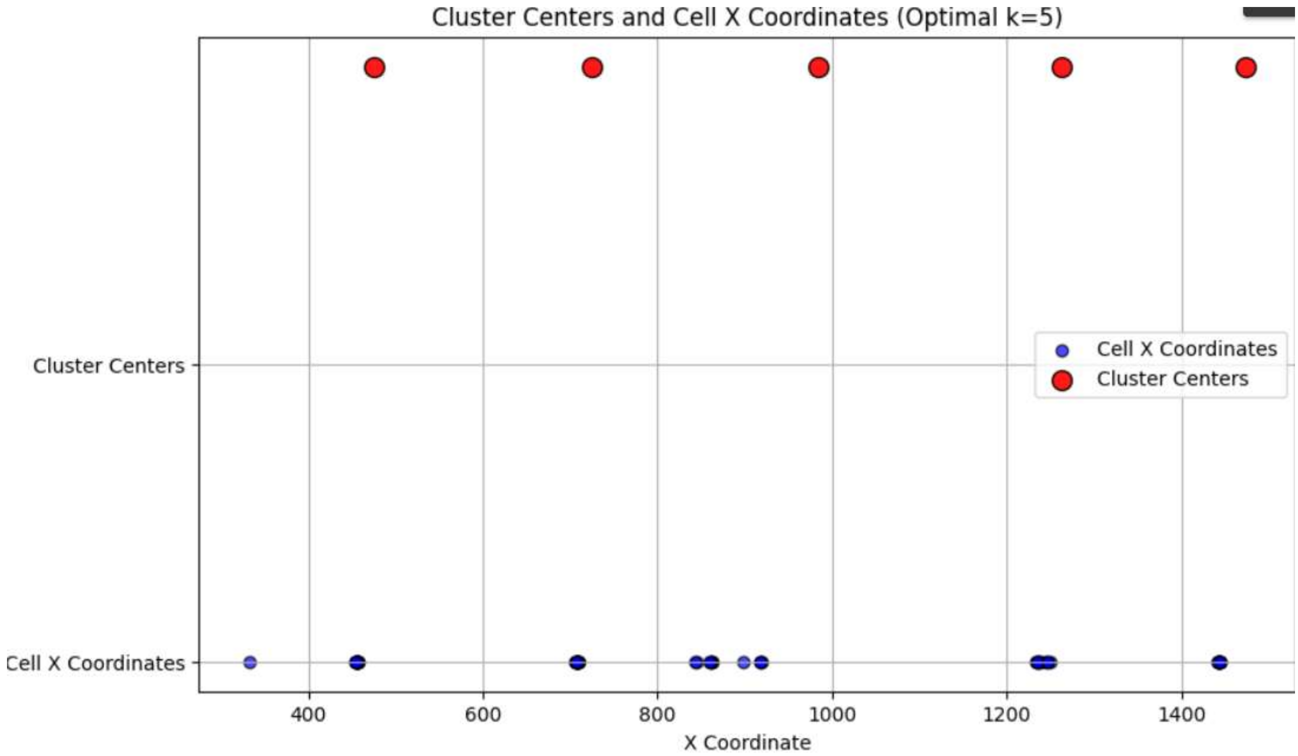
First we identify the columns in the table. We use the x-coordinates of the bounding boxes of the cells inside a table to perform clustering, with each cluster representing a column. The function assigns a cluster label to each bounding box based on its x-coordinate, grouping together cells that belong to the same column.

After clustering, we associate the OCR text extracted from the table cells with the appropriate columns. If the clustering is successful, the function matches each OCR result (text) to its corresponding column, based on the cluster label assigned to that cell's bounding box.

Once the columns are detected and the OCR results are matched, we save the structured data into an Excel file. The data is organized such that each column corresponds to a detected cluster, with the OCR text placed in the correct column based on the clustering results.

Helper Functions

- find_elbow_point
    - The purpose is to identify the "elbow point" in the clustering loss curve (where adding more clusters doesn't significantly reduce loss).
    - It does that by using the KneeLocator from the kneed library.Takes the range of possible clusters (2 to 20) and the associated losses (inertia values) from KMeans.Determines the optimal number of clusters (knee).
- find_columns_with_kmeans
    - The purpose is to detect columns in the table by using the midpoints of bounding boxes and KMeans clustering.
    - It does that by extracting the x-coordinates of bounding box midpoints (x_midpoints). Initialising potential cluster centers evenly across the table width.Than it Iteratively runs KMeans clustering for different values of k (from 2 to 20) and computes loss (inertia) for each k.Finds the optimal k using the elbow point method (find_elbow_point).Than reruns KMeans again with the optimal k to compute final cluster centers.Than we visualize the results.



In the image we see that the blue dots are the x-coordinates of the table cells and the red dots are the detected column centers. In the end it returns cluster labels for each bounding box.

- Match_ocr_to_columns
    - The purpose is to Match OCR results to detected table columns based on their cluster labels.
    - We create a dictionary where each key corresponds to a column index.We Iterate through OCR results and assign text to the column corresponding to its cluster label.
- Save_columns_as_table
    - The purpose is to save the detected columns as an Excel file in tabular format.

- We calculate the maximum number of rows in any column.We normalize the lengths of columns by appending empty strings ("").Than we convert the data to a Pandas DataFrame and export the DataFrame to an Excel file.

Main Script

1. Input Data
   - bounding_boxes: List of bounding box coordinates (in [x_min, y_min, x_max, y_max] format) for table cells.
   - table_width: Width of the table derived from its bounding box.
   - table_height: Height of the table derived from its bounding box.

2. Steps:
   - Column Detection: Calls find_columns_with_kmeans to detect columns and assign cluster labels to bounding boxes.
   - Match OCR Results: If clustering is successful, matches OCR text to corresponding columns using match_ocr_to_columns.
   - Save to Excel: Saves the detected columns as a structured table in an Excel file using save_columns_as_table.