Comparing YOLOv10 with Florence-2 for Food Item Detection

Raphael Devenish 25b

Supervised by Daniel Engler and Andreas Jozsa

26th September 2024

Gymnasium Interlaken

Figure 0: Title page graphic

1. Table of Contents

2. Abstract

This paper compares YOLOv10 and Florence-2 for detecting and classifying food items in storage images. The YOLOv10m, YOLOv10x, and Florence-2 base models were fine-tuned on a dataset of food images with 25 classes. The YOLO models were trained with a batch size of 16 for 50 epochs, while the Florence-2 model was trained with a batch size of 10 for 10 epochs. This study demonstrates that YOLO models outperform Florence-2 in both speed and accuracy within the tested conditions. Despite challenges with the Florence-2 model, the results confirm the feasibility of real-time image detection for classifying food items in complex environments. Future research should focus on larger datasets and advanced techniques to improve performance and applicability in practical scenarios.

3. Introduction

As the global population continues to grow, the challenge of feeding more people with fewer resources becomes increasingly critical. One promising approach to address this issue is by reducing food waste through innovative technologies. For instance, using images of food items to suggest recipes can help ensure that perishable goods are utilized efficiently before they spoil. The original objective was exactly that. Building an application that can recognize ingredients and search through a large recipe database and suggest possible meals. This was deemed to be unfeasible for the scope of this paper and replaced with the current objective (Section 4).

As rental prices increase, the space that the checkout section occupies in a store becomes more and more expensive. A more efficient checkout system, which works with automatic grocery detection, may reduce these costs while increasing the customer satisfaction.

In this context, using machine learning (ML) to recognize food items, both in domestic settings (e.g. refrigerators) and in commercial environments (e.g. shopping baskets) holds potential. Vision models, a specific subset of ML, are particularly well-suited for this task. These models are designed to identify objects from images, making them useful in these applications.

Several vision models are available today. They have varying levels of accuracy and efficiency. So how can one decide which to use? The approach used was choosing models that are popular, open source, and new. YOLOv10 was released in May 2024 and the official pytorch implementation is licensed under the AGPL-3.0 license, making it open source [1], [2]. The YOLO models have had great success and are very popular. Florence 2 is also open source with an MIT license [3]. It was released in June 2024 and is gaining in popularity [4] . This study focuses on comparing the two models to determine their effectiveness in detecting and classifying food items. By fine-tuning these models on a dataset comprising 25 different food categories, the study assesses their performance in terms of speed, accuracy, and practicality for real-world applications. The findings from this research could pave the way for advanced AI solutions that enhance food management practices and streamline retail operations.

4. Objective

The objective of this paper is to compare the effectiveness of the YOLOv10 and Florence-2 models in detecting and classifying food items in images of storage spaces. This comparison will focus on both objective and subjective parameters to evaluate the models comprehensively.

In order to evaluate Objective Performance these key metrics were chosen:

- Accuracy: Assess the precision and recall of each model in identifying and classifying food items.
- Speed: Measure the processing time for image classification to determine real-time applicability.
- Robustness: Evaluate performance under various conditions such as different lighting and occlusions.

In order to assess subjective performance the usability and user experience were considered.

5. Methodology

To determine the better vision model for the application, the models must first be trained, than tested on a set of images, which will result in data and metrics wich we can use to compare them. The training process involves setting parameters and using the data set of images to expose the model to the target images to improve its accuracy. My research indicated that the YOLOv10 models are often faster but less accurate in comparison to Florence. It is expected that this process may show how training influences this outcome. It is expected that different settings of the parameters will also impact the accuracy.. expected to be faster bust less accurate than the Florence-2 models.

5.1. Parameters

Training (Fine-tuning) an AI model requires the setting of parameters before initiating training. Key among these are determining the batch size, the number of training epochs, and selecting appropriate pretrained models as starting points.

Our models come with pre trained weights. That is to say that it has already been trained on a large data set. This comparison will fine tune those weights to best suit our use case.

For this study, specific choices were made to optimize the performance of both YOLOv10 and Florence-2 models.

5.1.1. YOLOv10

The pretrained weights utilized for YOLOv10 included yolov10m.pt and yolov10x.pt versions from James Lahm. These pretrained configurations were selected to explore trade-offs between processing time and accuracy, aiming to achieve superior performance in food item detection and classification. [5], [6]

5.1.2. Florence-2

The Florence-2 model started with base pretrained weights, which incorporate 0.23 billion parameters (can be thought of as visual cues the model uses to make an identification of a target image). This may seem like a lot, but is quite small in comparison to most other models. This aspect emphasizes the efficiency of the architecture. Demonstrated as it maintains competitive performance compared to larger models.So for the accuracy this model demonstrates it is actually rather small. This choice was made to evaluate its effectiveness in food recognition tasks within constrained computational resources. The florence-2 model is also not just a object recognition model like the YOLO models. It is a vision foundation model that can also caption images, propose regions for objects and recognize optical characters. This makes it much more divers in its use cases compared to the YOLO models. [7]

5.1.3. Batch Size

Another parameter that must be set is the batch size. The batch size determines the number of samples that are propagated through the network at once. The following are the factors that were considered when choosing the batch size:

Firstly, Available Hardware. The larger the batch size the more computational resources are required. Batch size is limited by the GPUs' memory. It is set to values in proportion to `8`, `16`, `32`, and `64`. For the YOLO model a batch size of 16 was chosen and 10 for the Florence-2 model. [8]

Secondly, the effect of a smaller batch sizes is that it leads to noisier gradient estimates. This can help the model converge to better solutions by escaping from local minima. "Many algorithms use gradient descent because they need to converge upon a parameter value that produces the least error for a certain task." [9]. However, too small a batch size might lead to unstable training, less accurate gradient estimation, and longer training times. [8]

Thirdly the effects of larger batch sizes are improved training times and more efficient recourse use. As it can benefit from vectorized operations and parallelism, leading to more efficient computation. However, after a certain point, the benefits of increasing batch size usually diminish. [8]

5.1.4. Epochs

The number of epochs is another crucial parameter in the training process. It determines how many times the learning algorithm will iterate over the entire dataset. Each iteration over the complete dataset is known as an epoch. While more epochs could potentially improve the model, they also increase the training time. Limiting the number of epochs can help prevent overfitting by a method called early stopping [10]. Overfitting occurs when the model becomes too adapted to the specific images of the training set while loosing its ability to recognize new images. The choice of 50 and 10 epochs is a strategic one. It represents a balance between training time and model performance.

5.2. Preparation

There was a lot of preparation that needed to be done before the actual training could begin. This included choosing the models, the specific pretrained weights to fine tune, choosing an appropriate dataset, determining parameters like batch and epoch size and finding a computer that could perform the training.

5.2.1. Research

At first research included searching for similar papers or projects. These helped to determine what tools to use and what results in accuracy to expect. Google scholar was used to find papers that related to anything including food detection and comparisons. The most useful results from this stage in preparation are listed in Section 15.3.

The research also included choosing the specific vision models. The YOLOv10 model was chosen because of its predecessors having widespread adoption and documentation.

YOLOv10, developed by researchers at Tsinghua University (Beijing, China) in May of 2024 is a Real-Time End-to-End Object Detection model. “Object detection is a technique that uses neural networks to localize and classify objects in images.”, this is not to be confused with image classification [11]. [1]

Image classification, also known as image recognition, involves sorting images into predetermined categories. a basic task might be if images contain a stop signs or not. Essentially, image classification attributes a single label to an entire image.

On the other hand, object detection identifies and labels specific objects within an image according to predefined categories. Unlike image classification, which simply distinguishes between images with and without stop signs, object detection pinpoints and classifies every road sign within an image, along with other elements like vehicles and pedestrians.

The End-To-End part of the name refers to a model that does the image detection in a single step.

Traditionally object detection systems consisted of multiple stages, such as feature extraction, object proposal generation, and classification. However, end-to-end object detection systems aim to simplify this pipeline by training a single model to perform all these tasks simultaneously. [12]

In practical applications, real-time object detection can be used in video surveillance, autonomous vehicles, quality control and many other fields.

The Florence-2 model was chosen due to its novel way of detection and its “newness”. Florence-2 is a vision foundation model with a unified, prompt-based representation for a variety of computer vision and vision-language tasks. It was developed by Microsoft in November 2023. The abstract of the Florence-2 paper describes the capabilities of the Florence-2 model as follows: “Florence-2 was designed to take text-prompt as task instructions and generate desirable results in text forms, whether it be captioning, object detection, grounding or segmentation.” [13]

It is referred to as a foundation model because it is designed to be finetuned for various different tasks. It is unified because it handles both images and text prompts in the same way making it more versatile and also simpler.

Florence-2 comes in two different parameter sizes: 0.23B (Base) and 0.77B (Large), making it significantly smaller than other powerful vision models, allowing it to run on devices with limited processing power. [13]

The Florence-2 model is very well suited for generating captions or alt text for images. Screen readers use alternate test to describe pictures included on web pages. “Fewer than 1% of images online contain alt text or image descriptions for people with vision loss, …” [14]. Using an efficient and accurate model like Florence-2 to add these alt texts would greatly increase the accessibility of the internet to visually impaired people.

5.2.2. Model acquisition

Both models are available for download on the Internet. It was a simple process to include the instruction into the training script to download the Yolo model. Importing the Florence model was harder because it involved loading it from the transformers library [15].

5.2.3. Environment

Setting up an environment where these models could be trained was done mostly automatically. The server rentals that were chosen already had pytorch installed and all the other dependencies could just be pip installed. The servers ran on ubuntu and the training files were transferred with scp from the PC to the server. The files were then executed through an ssh connection to the server.

5.2.4. Dataset Preparation

To prepare the dataset it first needed to be chosen. Then it needed to be converted to the correct format for training. For the test dataset, the annotations needed to be drawn manually.

5.2.4.1. Reasons for choosing Freiburg Groceries Dataset

The Freiburg Groceries Dataset was chosen for several reasons. Firstly, its extensive variety of images and realistic settings make it ideal for training a robust object detection model. The dataset’s diversity in perspectives, lighting, and clutter ensures that the model can handle complex real-world scenarios, which is crucial for applications in pantries and refrigerators. Additionally, the inclusion of multiple brands and packaging designs within each class helps the model generalize better to new and unseen items. Importantly, the Freiburg Groceries Dataset has been utilized by other research papers with similar goals of food item detection and classification. This allows for direct comparison of my model’s performance with existing approaches, facilitating a more comprehensive evaluation of the model’s effectiveness. The dataset also provides a challenging benchmark with its cluttered scenes, which are essential for evaluating the model’s performance in detecting multiple objects simultaneously. [16]

5.2.4.2. Dataset Description

The Freiburg Groceries Dataset consists of 5000 images across 25 classes of grocery items, captured in real-world settings such as stores, apartments, and offices in Freiburg, Germany. The images vary in perspective, lighting conditions, and degree of clutter, reflecting typical domestic environments. Each class includes between 97 and 370 images, covering a wide range of brands, flavors, and packaging designs. The dataset also features an additional subset of 74 images depicting 37 cluttered scenes, each containing multiple object classes. These scenes were created to emulate real-world storage conditions and include RGB images, depth images, and point clouds from a Kinect v2 camera. [16]

Figure 1: Some example images for all the classes of the dataset

5.2.4.3. Limitations and Drawbacks

While the Freiburg Groceries Dataset is comprehensive and diverse, it is not without limitations. The dataset predominantly features grocery items found in German stores, which may limit its applicability to other regions with different brands and packaging. Additionally the number of images per class varies, potentially leading to class imbalance issues during model training. The dataset's focus on packaged items may also limit its effectiveness in detecting fresh produce or homemade food items.

5.2.4.4. Preparing the Dataset for Training

The "Groceries Object Detection Dataset" on GitHub, a version of the Freiburg Groceries Dataset annotated in Pascal VOC format, was used. The branch train_eval was used which already split the dataset into training images and validation images. The test set will be a different dataset with multiple objects per image, aligning with the comparison's objective of multi-object detection. Two small python scripts were used to convert the dataset into the respective formats for YOLOv10 and Florence-2.[17]

5.2.4.5. Test Dataset

The test dataset is a subset of the Freiburg Groceries Dataset. The test dataset from [16] only includes the classes that are present in the images (sometimes not all of them or even misspelled ones) which are not enough to run thorough tests on. To label all the 37 images Roboflow was used and all the classes provided by the original dataset were implemented with bounding boxes. Roboflow is an end-to-end computer vision platform that manages datasets, annotates images and trains models. This dataset could then be exported into either the florence-2 architecture or the YOLOv10 architecture simplifying the process since there was no need to the conversion scripts. The test dataset includes images from various real-world settings and covers all 25 classes of grocery items. [16]

5.3. Training Script Development

The training scripts were each developed with help from various tutorials and LLMs (large Language Models, e.g. ChatGPT). Developing for each model were very different experiences. The YOLO scripts were very simple and most was done automatically while the Florence script required writing training and validation loops including logging and saving the models. This will greatly influence the subjective evaluation of working with these models.To increase the efficiency of setting up the training environment and training the model, two different methods were explored. The YOLO model only had one training script, so to set it up it a bash script was written that would automatically download the dataset, convert it into the correct format, and execute the training script. For the florence model a set of python files were built that would do the same thing, but with far more code, due to the complexity of setting up Florence-2. The training and automation scripts are available on GitHub.[18]

5.4. Weights and Biases Integration

Weights and Biases (Wandb) was chosen to log the training process. Wandb is a tool that helps track and visualize machine learning experiments. It logs metrics from the training process like loss and accuracy and can also save model weights.

6. Training

To train and evaluate the YOLOv10 and Florence-2 models, the following steps were undertaken:

6.1. Data Preprocessing

The Freiburg Groceries Dataset comes with preprocessing already performed:

- Image Resizing: All images have been down-scaled to a size of 256×256 pixels.
- Padding: Images have been padded with gray borders to maintain consistent dimensions due to different aspect ratios of the cameras used.

6.2. Training Parameters

6.2.1. YOLOv10 Model Training

The YOLOv10 model was trained using two different sets of pretrained weights, specifically the and versions. The training process for each set of weights involved the following parameters:

- Data: The model was trained using the matura.yaml configuration file, which specifies the dataset details.
- Epochs: The training was conducted for 50 epochs.
- Batch Size: A batch size of 16 was used for each training run.
- Pretrained model: Two separate training sessions were conducted:
  1. yolov10m.pt pretrained weights.
  2. yolov10x.pt pretrained weights.

The model's training was managed using the Ultralytics YOLOv10 framework. The training script is available on GitHub [18].

6.2.2. Florence-2 Model Training

The Florence-2 model training process involved the following parameters:

- Epochs: The model was trained for 10 epochs.
- Batch Size: A batch size of 10 was used.
- Optimizer: The AdamW optimizer was employed with the training.
- Data Loaders: Custom data loaders were created for both training and validation datasets. [18]
- Pretrained Model: The Microsoft Florence-2-base pretrained model was used.

The Florence-2 model was trained with a custom script that sets up the training environment by installing all necessary packages, downloading and changing the dataset into the correct structure, train the model on it and afterwards testing the model on the test dataset. All the code is available on GitHub under the setup folder [18].

7. Evaluation

The models performances were evaluated using various metrics to provide a comprehensive understanding of its effectiveness in object detection tasks. These evaluations were facilitated using the Weights and Biases (WandB) platform, which enabled detailed tracking and analysis of the model's training and validation metrics. The metrics used for evaluation and the methods of validation are detailed below.

7.1. Metrics

Mean Average Precision (mAP): The primary metric used for evaluating the object detection performance was the mean Average Precision (mAP). The mAP metric measures the precision and recall of the model across various Intersection over Union (IoU) thresholds. Two specific mAP values were calculated:

$$\text{mAP} = \left(\frac{1}{N}\right) \sum_{k=1}^{k=N} \text{AP}_k$$

$$\text{AP}_k = \text{Ap of class k}$$

$$n = \text{number of classes}$$

Mean Average Precision Formula[19]

- mAP@0.5: This metric calculates the average precision at an IoU threshold of 0.5, which is a more lenient overlap criterion.
- mAP@0.5:0.95: This metric averages the precision over multiple IoU thresholds ranging from 0.5 to 0.95, providing a more comprehensive assessment of the model's performance across different levels of overlap.

7.2. Training Loss Components

The training process was monitored through several loss metrics, which were tracked to understand the model's learning behavior:

- Classification Loss: Measures the error in predicting the class labels of detected objects.
- Box Regression Loss: Measures the error in predicting the bounding box coordinates.
- Objectness Loss: Measures the confidence score for the presence of objects.
- Region Proposal Network (RPN) Loss: Measures the error in generating region proposals for object detection.
- Overall Iteration Loss: The composite loss recorded at each training iteration.
- Overall Epoch Loss: The cumulative loss recorded over each training epoch.

7.3. Validation Set

As mentioned in Section 5.2.4.4 the dataset was split into training and validation sets. It is important to note that the train/val split in the "Groceries Object Detection Dataset" differs from the original Freiburg dataset's split [17], [16]. This discrepancy in data splitting can introduce a slight variance in the model's performance when compared to other studies using the Freiburg dataset. Different data splits mean the model may have been trained and evaluated on different subsets of data, potentially affecting its generalization performance and making direct comparisons with other models less straightforward.

7.4. Implementation with WandB

The evaluation metrics and the entire training process were tracked using the WandB platform, which provided detailed insights and visualizations. The run data, including training and validation metrics, are public and accessible on wandb.ai [20]–[22]. The YOLO model was easier to log since it automatically logs everything needed in comparison to Florence-2 where custom training and validation scripts were needed.

7.5. Test Dataset

To evaluate the performance of the trained models on the test dataset, two distinct scripts were developed which facilitate the evaluation process. These scripts were designed to ensure comprehensive assessment and comparison of model capabilities across various metrics and tasks. Both of these scripts can be found as test-YOLO.py and the setup folder. [18]

7.5.1. Florence-2 Evaluation Script

The evaluation process involved the creation of a dedicated Python script tailored to the specific requirements of the florence-2 model and the custom test dataset. The script leverages the capabilities of PyTorch and Hugging Face's Transformers library to execute the following key steps:

1. Model Loading and Initialization: The scripts begin by loading the pretrained models from the Hugging Face Model Hub using AutoModelForCausalLM and AutoProcessor. These models were fine-tuned on the target task, ensuring they are optimized for generating text outputs based on input sequences and image features.
2. Dataset Handling: The test dataset, previously unseen during training and validation phases, was loaded using custom data loaders configured to efficiently process input data. This ensured that the evaluation scripts could handle varying data formats and sizes, crucial for real-world applicability.
3. Evaluation Metrics Calculation: Metrics such as accuracy, precision, recall, and F1 score were computed using sklearn.metrics. These metrics provided quantitative insights into the models' performance, highlighting their ability to generate accurate and meaningful outputs across different evaluation scenarios.
4. Model Performance Logging: Results from the evaluation were logged using the Weights & Biases (wandb) platform. This logging facilitated real-time monitoring of model performance metrics, enabling rapid iteration and improvement of model configurations based on empirical findings.

7.5.2. YOLOv10 Evaluation Script

The evaluation process involved the creation of a dedicated Python script tailored to the specific requirements of the YOLOv10 model and a custom image dataset. The script leverages the capabilities of OpenCV, Supervision, and the Ultralytics YOLO library to execute the following key steps:

1. Model Loading and Initialization: The script begins by loading the pretrained YOLOv10 model from a specified directory. This model was fine-tuned on the target task, ensuring it is optimized for detecting objects in the given images.
2. Dataset Handling: The images in the test dataset, previously unseen during training and validation phases, are processed one by one. The script reads images from a specified folder, ensuring that it can handle various image formats such as PNG, JPG, and JPEG.
3. Object Detection and Annotation: For each image, the model generates predictions which are then converted into a suitable format for annotation using the Supervision library. The script utilizes bounding box and label annotators to overlay detection results on the original images, providing visual feedback on model performance.
4. Output Generation: Annotated images are saved in an output directory, allowing for easy review and analysis of detection results. This step ensures that the evaluation process produces tangible outputs that can be visually inspected.

8. Implementation

The models were trained as described above. This resulted in finetuned models that satisfied the exceptions.

During testing multiple issues were encountered. These included issues with the dataset, the testing code and finally the logging process. After undertaking a troubleshooting process, these issues were resolved and data could be collected.

Contrary to expectations this provided test results that still didn't make sense. A number of metrics that were collected showed 0 while precision showed 1. In the visualizations, there were no bounding boxes drawn. These results were viewed on wandb [23]. It is not known what the issues with the testing process were. These could, after extensive troubleshooting, not be resolved.

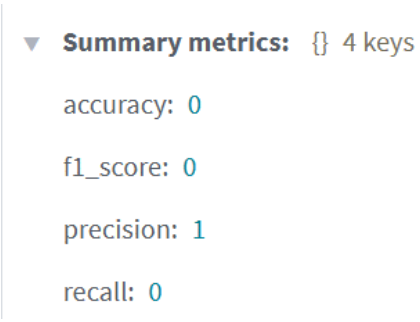

Figure 2: Screenshot of the test data run

The purpose of this study is to compare the models, even if the test data could not be collected, the training data still can be used as a reflection of the models capabilities. To continue with the study, the training data was used to compare the models, instead of the test data. This means that the data used in the conclusion, was not from the test dataset, but from the validation dataset.

Other metrics like the mAP could not be collected in the Florence model due to its more complex training process.

9. Results

The model's performance was evaluated using the detailed metrics and methodologies described in Section 7, providing a comprehensive understanding of its effectiveness in object detection tasks.

9.1. YOLO

Metric	YOLOm	YOLOx
Classification Loss (validation)	0.43766	0.51302
Speed (PyTorch, ms)	5.98	14.028
Box Loss (Validation)	0.5053	0.5042
Classification Loss (Training)	0.22104	0.1517
mAP50-95 (B)	0.8422745530374682	0.8448791072885542
mAP50 (B)	0.9311299999553868	0.927779008686234
Precision (B)	0.9139740838056396	0.9072414324828788
Recall (B)	0.8447696526591085	0.8459360902451885
GFLOPs	64.13	171.267

Table 1: Metrics for the YOLOv10m/x finetuned model

9.2. Florence-2

Metric	Last Epoch (10)	Best Epoch (3)
Training Loss	0.3543624049188698	0.5513969439865403
Validation Loss	0.6115014888346195	0.5074058137834072

Table 2: Metrics for the Florence-2 finetuned model and the 10th and 4th epoch

9.3. YOLO Metrics Analysis

This section focuses on comparing the YOLOv10x and YOLOv10m models in various metrics.

9.3.1. Speed and Size

YOLOv10m demonstrates significant speed superiority, processing images in 5.98 ms compared to YOLOv10x's 14.028 ms. Moreover, YOLOv10m requires fewer parameters (GFLOPs: 64.13) than YOLOv10x (GFLOPs: 171.267), making it a lighter model. These metrics indicate YOLOv10m's suitability for real-time applications.

9.3.2. Accuracy and Precision

Both models exhibit exceptionally high precision, with YOLOv10x at 0.9072 and YOLOv10m at 0.914, highlighting YOLOv10m's slight edge in precision. Recall rates are comparable between the two models.

Figure 3: A training image from YOLOv10x with bounding boxes drawn

9.3.3. Loss and Error Metrics

While both models exhibit striking similarity across various metrics, there are subtle distinctions. Validation dataset classification losses indicate YOLOv10m's superior accuracy in classifying items, whereas box loss metrics show similar performance in drawing bounding boxes. Nearly identical mAP metrics suggest comparable overall performance, although nuanced differences may impact specific applications, such as uses outside of the finetuned field.

9.3.4. Model Complexity

YOLOv10x is more complex with a higher number of parameters and computational requirements, which might be more suitable for scenarios where computational resources are not constrained. YOLOv10m, being less complex and faster, would be more appropriate for deployment in resource-constrained environments or where low latency is critical.

9.3.5. Conclusion

In conclusion, the YOLOv10m model is more than twice as fast as the YOLOv10x model while maintaining at the very least similar accuracy. The speed of the m model comes from its small size. The accuracy results are more surprising since it would be expected that the larger model would be more accurate. This suggests that the pretrained models are both improved for the specific data set areas in the fine tuning process to a similar degree. This may be tested by using a larger dataset in the fine tuning process, to test if the similar accuracy is maintained. Moving forward, we will compare the Florence-2 model exclusively with the YOLOv10m model.

9.4. Florence-2 Metrics Analysis

This section focuses on the results of the training data of the Florence-2 model.

9.4.1. Overfitting

As seen in Section 9.2, the last epoch was far from the best one. This is consistent with a result expected of overfitting since the avg. training loss kept going down while the avg. validation loss was increasing. Overfitting is discussed in more detail in Section 5.1.4.

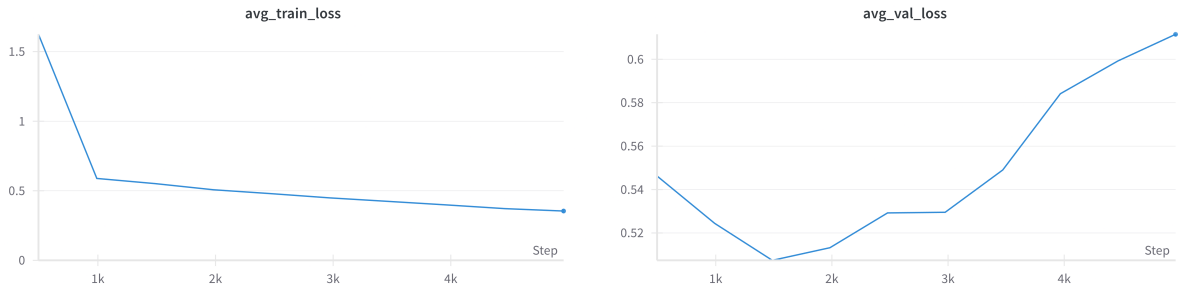

Figure 4: The avg_train_loss keeps going down while the avg_val_loss increases

9.4.2. Missing Data

One might have noticed that the table of data for the Florence-2 model (Section 9.2) is lacking compared to the YOLOv10 models table (Section 9.1). This is a result of incomplete data gathering and logging. The reasons and problems with data gathering in the Florence-2 model are discussed in Section 15.1.3.

9.4.3. Conclusion

The results seem to suggest that the model was trained for too many epochs (overfitting is explained in Section 5.1.4). The amount of data collected from the Florence-2 training was much small than the YOLO models since the logging was not automatic. There were multiple challenges when trying to log Florence-2 with wandb which is explain more in Section 15.1.3.

9.5. YOLOv10m vs. Florence-2 Model Comparison

This section will discuss the differences between the YOLOv10m model and the Florence-2 model.

9.5.1. Loss and Error Metrics

The results show that YOLOv10m and Florence-2 exhibit similar trends in loss metrics in general. However, Florence-2 shows a higher validation loss value compared to YOLOv10m model. This results in the Florence-2 model being slightly slightly less accurate.

Both models perform comparably in drawing bounding boxes and classifying objects. Usually mAP metrics would be used to further compare models but due to difficulties getting accurate data from the Florence-2 model. Due to this, the study will only compare the training and validation loss.

9.5.2. Model Complexity and Deployment

YOLOv10m's lower computational complexity and faster inference make it advantageous for deployment in environments where speed and efficiency are critical factors. In contrast, Florence-2, despite its higher computational demands, demonstrates comparable accuracy and may be more suitable for applications requiring detailed descriptions of the objects recognized.

From personal experience working and setting up the YOLO model was far easier than the Florence-2 model. This is further discussed in Section 15.1.

9.5.3. Limitations of Comparison

It's important to note that this comparison was limited due to the inability to use the test data. The test dataset would have been a simulation of a real world application of the models, since it includes never before seen images and cluttered scene. Comparing YOLOv10m and Florence-2 directly based on the metrics recorded in this paper may not be entirely conclusive without the requirements of the application. This comparison does not include many standard comparison metrics as are well documented, this study is just to compare these models on a very specific application and dataset [24].

9.5.4. Conclusion

In conclusion, YOLOv10m and Florence-2 are both very current and advanced models, but YOLOv10m seems superior to Florence-2 in most ways. Since these are both very new models, this conclusion may very well change with future iterations of the models and datasets, but YOLOv10m is the overall best suited model in this study.

10. Analysis of Results

The following section details nuances of the results found in this paper.

10.1. Comparison with other Methods or Studies

Numerous research papers explore similar topics to this study, albeit with varying objectives. Many of these studies do not primarily focus on object classification or identification. Instead, these studies usually concentrate on assessing the quality and/or condition of food items, to determine if they are spoiled or have damaged packaging. While these goals differ fundamentally, they often leverage similar technologies and methodologies. For projects and papers that share more common ground with the objectives of this study, refer to the resources listed in Section 15.3.

10.2. Limitations

Several limitations were encountered during this comparison:

- Dataset Size: The dataset used to finetune, was relatively small, containing far fewer classes than what would be encountered in a real-world retail environment. This limited the model's ability to generalize to a broader range of products.
- Occlusion and Variability: The models struggled with images showing only partial items (occluded items), a common issue in real-world scenarios like fridge or pantry organization. This issue could potentially be addressed with a multi-camera solution for better item visibility. Another solution would be moving the camera while taking a video, to show multiple perspectives.
- Packaging Variability: Items such as vegetables, eggs, and bread are generally easy to identify due to their consistent appearance across regions. However, packaged items like milk, candy, and canned items vary widely, even within the same country, making them more challenging to identify accurately.
- Data collection issues: There were problems when collecting exact data from the Florence-2 model which leads to a lower quality comparison.
- Computational Resources: Training such models demands considerable computational power. Even for this relatively small paper, renting a server was necessary. Scaling up to a model capable of handling the diverse range of items in a retail setting would require even more resources.
- Accuracy: As with all image detection models, these models show inaccuracies, making them too unreliable for use in a checkout system like "Just Walk Out" by Amazon[25]. In comparison, a simple recipe suggestion app can tolerate occasional mistakes, as the consequences of inaccuracies are less critical.

11. Conclusion

The results of this study indicate that the YOLO models are superior in both speed and accuracy within the context of this specific dataset, training method, and set of training variables. The Florence 2 model may be less suited for this application since it is a model that can do much more than just object recognition (discussed in Section 5.1.2). The YOLO models are built specifically for object detection. This finding is not meant to be generalized beyond these conditions. Although I encountered difficulties in gathering data from the Florence-2 model and comparing the test results, the conclusions drawn here still demonstrate the feasibility of a real-time image detection model capable of accurately classifying food items.

This study does not present entirely new information but rather demonstrates the advancements in machine learning technology. The project on which this paper is based (fridge snap) could only identify food items when they were isolated and under near-optimal conditions. In contrast, these newer models, especially YOLOv10m, offer the potential to analyze images of more complex environments, such as a pantry or refrigerator, enhancing user-friendliness. [26], [27]

The dataset used was relatively small compared to modern standards, highlighting a need for a similar but larger dataset in future research to improve model performance. Further studies could also explore advanced techniques for handling occlusions and presentation variations in food items to enhance the applicability of these models in practical scenarios. This is further discussed in Section 15.1.4. [28]

With the clear superiority of the YOLOv10 model demonstrated, any attempts to build a program utilizing this object detection should strongly consider using it. In terms of popularity and resources supporting it, the YOLOv10 model is supported by its previous versions. This means building with YOLOv10 would be easier than using Florence-2, since it has had more time to mature.

12. Summary

12.1. Objectives

The project compares YOLOv10 and Florence-2 for detecting and classifying food items in storage images

12.2. Methodology

The YOLOv10m, YOLOv10x, and Florence-2 base models were fine-tuned on a dataset of food images with 25 classes. The YOLO models were trained with a batch size of 16 for 50 epochs, while the Florence-2 model was trained with a batch size of 10 for 10 epochs.

12.3. Conclusion

This study demonstrates that YOLO models outperform Florence-2 in both speed and accuracy within the tested conditions. Despite challenges with the Florence-2 model, the results confirm the feasibility of real-time image detection for classifying food items in complex environments. Future research should focus on larger datasets and advanced techniques to improve performance and applicability in practical scenarios.

13. Bibliography

[1] A. Wang *et al.*, "YOLOv10: Real-Time End-to-End Object Detection." [Online]. Available: https://arxiv.org/abs/2405.14458

[2] T. U. THU-MIG, [Online]. Available: https://github.com/THU-MIG/yolov10?tab=readme-ov-file

[3] P. Skalski, "Florence-2: Open Source Vision Foundation Model by Microsoft," *Roboflow Blog*, Jun. 2024, [Online]. Available: https://blog.roboflow.com/florence-2/

[4] A. Marafioti and M. Noyan, "Fine-tuning Florence-2 - Microsoft's Cutting-edge Vision Language Models," *Huggingface Blog*, Jun. 2024, [Online]. Available: https://huggingface.co/blog/finetune-florence2

[5] A. Wang *et al.*, [Online]. Available: https://huggingface.co/jameslahm/yolov10m

[6] A. Wang *et al.*, [Online]. Available: https://huggingface.co/jameslahm/yolov10x

[7] B. Xiao *et al.*, [Online]. Available: https://huggingface.co/microsoft/Florence-2-base

[8] A. Lheureux, "How to maximize GPU utilization by finding the right batch size." [Online]. Available: https://blog.paperspace.com/how-to-maximize-gpu-utilization-by-finding-the-right-batch-size

[9] A. Pai, [Online]. Available: https://www.machinelearningworks.com/tutorials/gradient-descent

[10] X. Ying, "An Overview of Overfitting and its Solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 22022–22023, Feb. 2019, doi: 10.1088/1742-6596/1168/2/022022.

[11] E. K. Jacob Murel Ph.D., [Online]. Available: https://www.ibm.com/topics/object-detection

[12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 213–229.

[13] B. Xiao *et al.*, "Florence-2: Advancing a Unified Representation for a Variety of Vision Tasks." [Online]. Available: https://arxiv.org/abs/2311.06242

[14] V. Lewis, 2023, [Online]. Available: https://www.perkins.org/resource/how-write-alt-text-and-image-descriptions-visually-impaired/

[15] T. Wolf *et al.*, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[16] P. Jund, N. Abdo, A. Eitel, and W. Burgard, "The Freiburg Groceries Dataset," 2016, [Online]. Available: https://arxiv.org/abs/1611.05799

[17] A. Aleksandrov, “Groceries Object Detection Dataset.” [Online]. Available: https://github.com/aleksandar-aleksandrov/groceries-object-detection-dataset

[18] R. Devenish, [Online]. Available: https://github.com/dreamingdodo/Matura-files

[19] D. Shah, [Online]. Available: https://www.v7labs.com/blog/mean-average-precision

[20] [Online]. Available: https://wandb.ai/gyminterlaken/YOLOv10-final/runs/654yop9h/overview

[21] [Online]. Available: https://wandb.ai/gyminterlaken/YOLOv10/runs/d662x9dq/overview

[22] [Online]. Available: https://wandb.ai/gyminterlaken/groceries-object-detection/runs/hm9vmp7g/overview

[23] [Online]. Available: https://wandb.ai/gyminterlaken/groceries-object-detection-eval/overview

[24] B. Xiao *et al.*, “Florence-2: Advancing a unified representation for a variety of vision tasks,” *arXiv preprint arXiv:2311.06242*, 2023.

[25] [Online]. Available: https://justwalkout.com/

[26] L. Boyd and N. Nnamoko, “FridgeSnap: A software for recipe suggestion based on food image classification,” *Software Impacts*, vol. 18, Nov. 2023, doi: 10.1016/j.simpa.2023.100585.

[27] L. Boyd, N. Nnamoko, and R. Lopes, “Fine-Grained Food Image Recognition: A Study on Optimising Convolutional Neural Networks for Improved Performance,” *Journal of Imaging*, vol. 10, no. 6, 2024, doi: 10.3390/jimaging10060126.

[28] P. Villalobos and A. Ho, “Trends in Training Dataset Sizes.” [Online]. Available: https://epochai.org/blog/trends-in-training-dataset-sizes

[29] M. Ding, B. Xiao, N. Codella, P. Luo, J. Wang, and L. Yuan, “Davit: Dual attention vision transformers,” in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV*, 2022, pp. 74–92.

[30] T. Mansoor, [Online]. Available: https://medium.com/@tahamansoor.liqteq/fine-tuning-florence-2-base-model-on-a-custom-dataset-for-image-captioning-a7861b15fbf8

[31] P. Selvam and J. A. S. Koilraj, “A Deep Learning Framework for Grocery Product Detection and Recognition,” *Research Square*, May 2022, [Online]. Available: https://doi.org/10.21203/rs.3.rs-1431986/v1

[32] R. Bhimani, [Online]. Available: https://github.com/bhimar/GrocerEye

14. Figure References

Fig. 0: The title page was decorated with graphics from iStock. Available: https://www.istockphoto.com/

Fig. 1: Some example images for all the classes in the dataset. Available: http://aisdatasets.informatik.uni-freiburg.de/freiburg_groceries_dataset/

Fig. 2: Screenshot of the test data run. Available: https://wandb.ai/gyminterlaken/groceries-object-detection-eval/overview

Fig. 3: A training image from YOLOv10x with bounding boxes drawn. Available: https://wandb.ai/gyminterlaken/YOLOv10-final/reports/train_batch27001-24-07-14-16-43-09---Vmlldzo4NjY4Njg0

Fig. 4: The avg_train_loss keeps going down while the avg_val_loss increases. Available: https://wandb.ai/gyminterlaken/groceries-object-detection/runs/hm9vmp7g

15. Appendices

15.1. Problems encountered

During the making of this paper, multiple problems arose. The most significant problem faced, the failure to test the models on the test dataset, was never solved, but circumvented by using the data collected whilst training.

15.1.1. Test Dataset Interference

The models performed well in training and validation. During the testing phase, I encountered significant obstacles in the ability of the models to analyze images effectively. They consistently failed to process the test images correctly. Despite numerous attempts to rectify the issues, the output remained consistently broken and unreliable. This stark contrast with the reported ease of others using similar setups has left me uncertain about including possibly flawed validation and training data in my final paper.

The lack of comprehensive documentation on the encountered errors and the attempted fixes has significantly hindered my ability to clearly articulate these challenges. Furthermore, the absence of discernible error codes compounded the frustration, making it particularly difficult to troubleshoot effectively.

15.1.2. DaViT Model Issues

During the testing and evaluation phase of the models, I encountered a critical issue related to model compatibility. Specifically, when attempting to load the model designed for the DaViT (Data-efficient Vision Transformer) architecture, an AssertionError was raised, stating that only DaViT models are supported for this checkpoint. [29], [30]

Upon further investigation, it was discovered that the root cause of this issue lay in the missing `vision_config` parameter within the `config.json` file associated with the model. This parameter is crucial for specifying the model type as “davit”, ensuring compatibility with the DaViT architecture during model initialization. [30]

To address this, the `config.json` file was updated to include the required `vision_config` parameter with `"model_type": "davit"`. This adjustment was essential to align the model's architecture expectations with its configuration, thereby resolving the loading error and enabling a evaluation of the model. [30]

15.1.3. Logging Data with Weights & Biases (WandB)

Logging data with WandB in the Florence-2 model was challenging because all parameters needed to be calculated and collected manually in the training script, which often caused issues. Despite providing useful graphs, the process was generally cumbersome and prone to producing nonsense graphs.

15.1.4. Dataset Issues

The dataset I used was small, inaccurate, and sometimes contained typos (e.g., "fosh" instead of "fish"). A more professional, larger, and internationally applicable dataset would be beneficial. For example, the RPC (Retail Product Checkout) dataset is comprehensive but is in Chinese and focuses on specific Chinese products. The main problem is the infinite variations in packaging and labeling, making it more practical to read and interpret the packaging directly rather than using predefined classes.

15.1.5. Documentation Challenges

There is a significant lack of good documentation explaining critical parameters and their functions. Often, similar features do not work well together, and rapid advancements in the field mean information quickly becomes outdated. Up-to-date information is typically found only in research papers, which require substantial prior knowledge to understand. This is problematic for comparing new models, as there is no standard way to compare them. Additionally, many guides rely on proprietary tools, limiting accessibility. There is a wealth of basic AI information available, but there is no middle ground for studying the topic without a formal education.

15.1.6. AI Hype and Misuse

The current AI hype leads to its application in solving problems where it is not the best solution. AI solutions are often resource-intensive and difficult to generalize, giving the field a bad reputation. Despite its potential, the misuse of AI for unsuitable problems undermines its credibility and utility.

15.1.7. Language Barriers

Datasets like the RPC are in Chinese, limiting their accessibility. While I can work in English, most cutting-edge AI information and resources are primarily available in English. This language barrier reinforces a "western" cultural bias in technology, as the leading AI companies and startups are predominantly based on the west coast of the United States.

15.1.8. Formats and Standards

The fast-moving AI space means standards quickly become outdated. This makes learning and information gathering difficult and complicates the use of large datasets, which are often created over many years and require frequent updates. Tutorials and scripts quickly become obsolete, further contributing to the lack of high-quality, current information.

15.1.9. Computational Requirements

Training a decent model requires either a powerful GPU (typically from a server) or a significant amount of time. The high cost of NVIDIA GPUs can prevent many interested individuals from experimenting with AI, as renting a GPU can cost upwards of 2 Dollars per hour. Considering that training can take days and often requires multiple attempts to optimize settings, the financial barrier is substantial. This exacerbates the concentration of power and the selection bias in computer engineering toward individuals from first-world countries.

15.1.10. Complexity of Choices

Setting up a training script involves numerous decisions, including selecting the dataset, batch size, epochs, validation/test/train distribution, model, and more. This high level of complexity creates a significant barrier to entry, making it challenging for newcomers to get started and produce meaningful results.

15.2. Future Work

Building on the findings of this study and addressing the encountered challenges, future work will focus on several key areas:

15.2.1. Creation of a Large and Diverse Dataset

Enhancing the dataset size and diversity is crucial. Future efforts will involve creating a comprehensive dataset of food items encompassing various storage conditions, lighting variations, and a wide range of food categories. Accurate labeling and annotations will be prioritized to bolster model training and evaluation. Problems with the current datasets are in Section 15.1.4.

15.2.2. Development of an Advanced Image Detection Application

An innovative application integrating QR code and label text recognition will be developed. This application aims to enhance food item detection and classification accuracy by cross-referencing detected items with product databases. This approach will support functionalities like expiry date tracking and nutritional information retrieval, improving overall usability. A version of this already exists and has achieved significant results [31].

15.2.3. Addressing Data Quality and Generalization Issues

Mitigating data quality issues and ensuring model generalization are ongoing challenges. Future work will explore advanced data augmentation techniques to simulate diverse storage conditions and minimize overfitting. Techniques such as synthetic data generation, domain adaptation, and transfer learning will be investigated to enhance model robustness.

15.2.4. Handling Occlusions and Variations in Food Presentation

To address occlusion challenges, future research will consider deploying multiple cameras in store shelves, refrigerators, or cabinets. Strategically positioned cameras will minimize occlusions, allowing for more comprehensive food item detection and classification. Techniques like instance segmentation and attention mechanisms will be explored to improve detection accuracy for partially visible items and items presented in varying orientations.

15.2.5. Integration with Practical Applications

The developed models and applications will undergo rigorous testing and integration into practical systems, such as recipe suggestion platforms and automated checkout systems. User feedback will guide model refinements, enhancing performance in real-world scenarios and increasing user satisfaction.

15.2.6. Exploring Advanced Techniques for Model Enhancement

Advanced methodologies like ensemble learning, self-supervised learning, and reinforcement learning will be investigated to further elevate model performance. These techniques offer potential improvements in detection accuracy, efficiency, and adaptability to new and unseen data.

15.3. Related Work

Similar studies and projects that focus on food item detection.

15.3.1. Grocer Eye

Abstract

> This project is an investigation into real time object detection for food sorting technologies to assist food banks during the Covid-19 pandemic. I trained a YOLOv3 model, pretrained on ImageNet, on the Frieburg grocery dataset that was annotated with object detection labels. By training for 6000 iterations and 13 hours on a Google Colab GPU, the model was able to achieve 84.59% mAP and 70.10% IOU on the test set. I demonstrate the effectiveness of the model on images from the test set and inference from a natural video. Though the model performs well on the test set, it does not seem to be effective enough to deploy for real time object detection at this time. I discuss the challenges and possible extensions of this work. [32]

15.3.2. FridgeSnap

Abstract (shortened)

> Computer vision plays a crucial role in reducing food waste by identifying individual food items. Existing datasets often categorize foods broadly, lacking granularity needed for precise identification in mobile applications. This study develops a model for fine-grained food recognition, integrating it into the FridgeSnap app for users to photograph and identify foods, and receive recipe suggestions. Evaluating seven convolutional neural network architectures on a custom dataset of 41,949 images across 20 classes, DenseNet emerged as the top performer. Parameter tuning further enhanced its accuracy and generalization, with the optimized model achieving significant improvements over the baseline. Integrated into FridgeSnap, this technology aims to combat food waste effectively. [27]. The application: [26].

15.3.3. A Deep Learning Framework for Grocery Product Detection and Recognition

Abstract (shortened)

> This study focuses on detecting and recognizing retail products on and off shelves in grocery stores by identifying label texts. The proposed framework comprises three modules: (a) Retail product detection using YOLOv5, (b) Enhanced text detection with the "TextSnake" algorithm and WHBBR, and (c) Text recognition using "SCATTER". YOLOv5 accurately detects products in both scenarios, while WHBBR improves text detection for regular and irregular text. Experimental results demonstrate the framework's effectiveness in extracting detailed product information like names, brands, prices, and expiration dates from retail environments. [31]

15.4. Replicating the Results

To replicate the results of this study either download the setup folder and execute main with the following parameters or follow the instruction below to train the YOLO models [18].

```
python main.py --mode train --batch_size 10
```

The models were trained on a server running Ubuntu with PyTorch 2.2.0. Firstly, the dataset needs to be downloaded from the GitHub repository.

```
clone -b train-val https://github.com/aleksandar-aleksandrov/groceries-object-detection-dataset.git
```

The dataset needs to be converted either into the YOLO format format. As detailed in Section 15.5, a Python script was used for this. The script handled the annotation conversion and ensured compatibility with the training format.

To ensure that all dependencies are installed, including PyTorch and any required libraries. There is a `requirements.txt` file that lists all the requirements [18].

To train the model training code was developed which is available on GitHub under the name `train-YOLO.py` [18].

15.5. Pascal VOC to YOLO v10

To convert the Pascal VOC annotations to the YOLO format, a Python script was developed and employed. The conversion process is detailed as follows:

1. XML Parsing:

The XML annotation files are parsed using the xml.etree.ElementTree module. This allows the extraction of essential data, such as object classes and bounding box coordinates.

1. Image Size Extraction: The dimensions of the image (width and height) are retrieved from the XML file to facilitate the normalization of bounding box coordinates.
2. TXT File Creation: For each XML file, a corresponding TXT file is created with the same base name. This TXT file will store the converted annotations in the YOLO format.
3. Object Annotations Conversion: For each object annotated in the XML file:
   - The class name is extracted and checked against a predefined list of classes.
   - The bounding box coordinates are converted from the XML format to the YOLO format. In the YOLO format, the bounding box is represented by the normalized coordinates of its center (x, y) and its width and height, relative to the image dimensions.
   - These converted annotations are then written to the TXT file in the format: class_id x_center y_center width height.
4. XML File Removal: To maintain a clean dataset directory, the original XML files are deleted after their conversion to TXT files.

For further details on the YOLO annotation format, you can refer to this article: https://roboflow.com/formats/yolov10-pytorch-txt.

The code used for this conversion is available on GitHub under the name “Pascal-VOC-to-YOLO.py”. [18]

15.6. Pascal VOC to Florence-2

To convert the Pascal VOC annotations to the Florence-2 format, a second Python script was developed and employed. The conversion process is detailed as follows:

1. Directory Management:

The script moves image and annotation files from their respective class subdirectories into a single destination directory. After moving the files, the original subdirectories are removed to maintain a clean structure.

1. XML Parsing and Image Size Normalization:
   - The script parses the Pascal VOC XML files using the xml.etree.ElementTree module, extracting essential information such as object classes and bounding box coordinates.
   - The dimensions of the images are normalized to a standard size of 224x224 pixels to standardize the annotations.
2. Florence-2 Annotation Format Conversion:
   - For each XML file, a corresponding JSONL entry is created. This entry includes the image name and a suffix constructed from the normalized bounding box coordinates and class names.
   - The bounding box coordinates are normalized and scaled, then converted into a specific format required by Florence-2. The coordinates are scaled up by a factor of 1000 for precision.
3. Writing Annotations to JSONL File:
   - The converted annotations are written to a JSONL file, with each line representing the annotations for one image.

The code used for this conversion is available on GitHub under the name “Pascal-VOC-to-Florence .py”. [18]

15.7. Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich diese schriftliche Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen und bei der Verwendung von KI-/LLM-Tools entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass gemäss Artikel 8 Absatz 2 Mittelschuldirektionsverordnung vom 16. Juni 2017 (BSG 433.121.1; MiSDV) bei fehlbarem Verhalten, insbesondere bei der Nutzung unerlaubter Hilfsmittel, die ganze Maturaarbeit als nicht bestanden erklärt wird. Weiter ist mir bekannt, dass der Entzug des aufgrund einer solchen Arbeit verliehenen Titels erfolgen kann. Ich nehme zur Kenntnis, dass zur Kontrolle der Einhaltung der Selbständigkeitserklärung und der Regelungen betreffend Plagiate meine Arbeit mit Hilfe einer Software (Plagiatserkennungstool) geprüft werden kann. Ich nehme zur Kenntnis, dass meine Arbeit zu diesem Zweck vervielfältigt und dauerhaft und anonymisiert in einer geschlossenen Datenbank gespeichert werden kann und diese zur Überprüfung von Arbeiten Dritter verwendet oder hierzu zur Verfügung gestellt werden kann.

Klasse: 25b

Name: Devenish

Vornamen: Raphael Keith Alberto

Datum: 26.09.2024

Unterschrift: