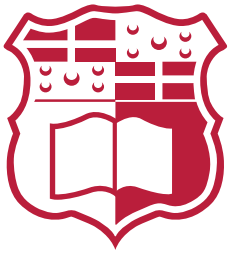


L-Università ta' Malta
Faculty of Information &
Communication Technology

Advanced Computer Vision for AI Group Project

Matthias Bartolo*, Jerome Agius*, Isaac Muscat*

*B.Sc. It (Hons) Artificial Intelligence (Third Year)

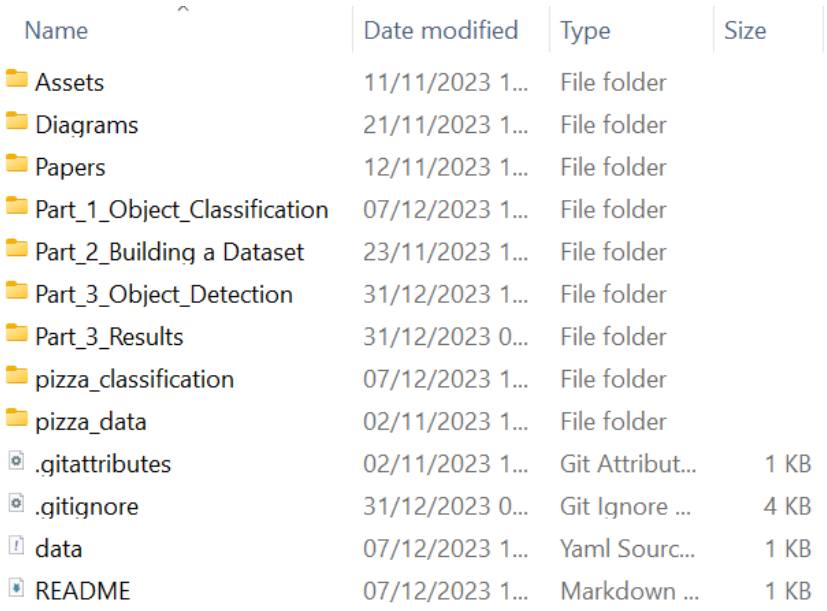
Study Unit Code:	**ARI3129**
Study Unit:	**Advanced Computer Vision for Artificial Intelligence**
Lecturer:	**Dr Dylan Seychell**
Date:	**14th January 2024**
Link:	GitHub Repository

Setup

The code required for this research was run on **Google Colab**; however, the **environment.yml** and the **requirements.txt** files are provided in the case of running the object detectors locally. The main directory contains the following folders:

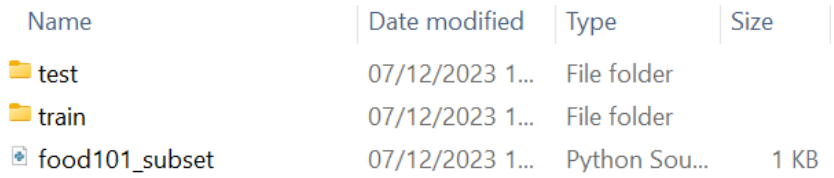
- **Papers** - Directory which contains any paper references throughout the documentation.
- **Part_1_Object_Classification** - Directory which contains all the files related to the Object Classification task.
- **Part_2_Building a Dataset** - Directory which contains all the files related to the creation of an annotated dataset.
- **Part_3_Object_Detection** - Directory which contains all the files related to the Object Detection task.
- **Part_3_Results** - Directory which contains any diagrams or images created for evaluation for the object classification and object detection tasks.
- **pizza_data** - Directory which contains the Kaggle Pizza Data Dataset [1].
- **pizza_classification** - Directory which contains the Kaggle Pizza Classification Dataset [2].

Furthermore, the image datasets [1, 2] used should be organised as can be seen in Setup Figures 1, 2 and 3.

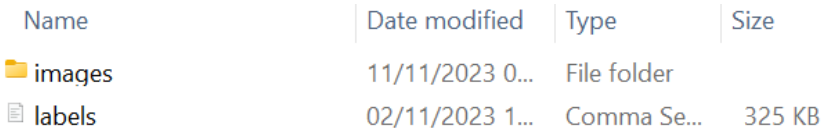


Name	Date modified	Type	Size
Assets	11/11/2023 1...	File folder	
Diagrams	21/11/2023 1...	File folder	
Papers	12/11/2023 1...	File folder	
Part_1_Object_Classification	07/12/2023 1...	File folder	
Part_2_Building a Dataset	23/11/2023 1...	File folder	
Part_3_Object_Detection	31/12/2023 1...	File folder	
Part_3_Results	31/12/2023 0...	File folder	
pizza_classification	07/12/2023 1...	File folder	
pizza_data	02/11/2023 1...	File folder	
.gitattributes	02/11/2023 1...	Git Attribut...	1 KB
.gitignore	31/12/2023 0...	Git Ignore ...	4 KB
data	07/12/2023 1...	Yaml Sourc...	1 KB
README	07/12/2023 1...	Markdown ...	1 KB

Setup Figure 1 Main Directory



Setup Figure 2 pizza_classification Directory



Setup Figure 3 pizza_data Directory

Contents

1 Introduction

Object Classification, **Dataset Creation**, and **Object Detection** stand as pivotal tasks in the field of computer vision. Having a diverse range of applications, such as in the fields of medicine and farming. In this study, the aforementioned techniques were applied to the culinary field, particularly pizza analysis.

The initial task saw the implementation of three pre-trained object classification models aimed at classifying pizza images. The three classification models chosen were **ResNet50**, **VGG16** and **MobileNet**, the main focus of this section was to classify images into two categories: those that contain a pizza and those that do not. This stage established a baseline for the detection of pizza in diverse visual settings and served as a foundation for the following tasks.

The following section required the creation of an annotated dataset, which would then be used to train object-detectors. The images were carefully chosen to include various different types of pizza, with different angles and visual features as well. For the purpose of this task, the **RoboFlow** annotation framework was utilised. Additionally, a set of ingredients were chosen to be the labels in this dataset, and for each image, bounding boxes were placed accordingly.

The final and most important task in this project entailed the training of object-detection models using the annotated dataset. Three different models were implemented: **YOLOv8**, **YOLOv5**, and **RetinaNet**. In addition, **DETR** was also implemented to a rudimentary degree when trying to find viable models to test; however, it is only a basic implementation and will not be used in evaluation. To be able to compare their overall performance, loss and accuracy graphs were then displayed for evaluation purposes.

Through the fusion of all of the aforementioned techniques, a pizza analysis system can be constructed. The process for this system is illustrated in Figure 1.1.

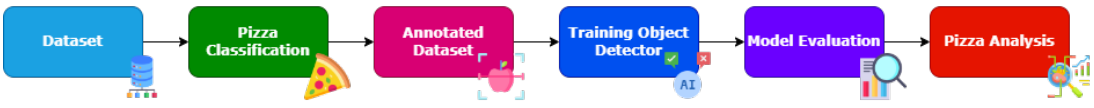


Figure 1.1 Project Pipeline

2 Background on the techniques used

2.1 Part 1: Object Classification

Object classification is a task in machine learning and computer vision where a model detects and labels objects based on a predefined set of classes. Such a system entails the training of a model on a labelled training set. After the training process, the model is then able to assign labels to given images with a certain confidence. A high confidence value signifies that the model is certain with its prediction, whereas a low confidence value denotes that the prediction is unlikely to be accurate. This is why it is common practice in computer vision to add a threshold of confidence to rule out low-confidence classifications. [3]

2.1.1 ImageNet

ImageNet (2009) is a dataset of images widely used for the training and testing of computer vision models. This dataset consists of labelled images in thousands of different categories. Model's can benefit from ImageNet's diverse range of images, enabling them to classify and detect a wider range of objects. In this project, the ImageNet dataset itself was not used; however, the weights of the ImageNet pre-trained models were used. [4, 5]

2.1.2 VGG16

VGG16 (Visual Geometry Group with 16 Layers) is a convolutional neural network architecture released in 2014, mostly known for its effectiveness in image classification tasks while being very simple to implement. This model consists of 16 layers, 13 of which are convolution layers, while the other 3 are fully connected layers. VGG16's main focus is on capturing intricate patterns and features within images. While its design today is slightly outdated compared to more modern and complex architectures such as ResNet50, it is still considered a foundational model for understanding convolutional neural networks in computer vision. [6]

2.1.3 ResNet50

ResNet50 (Residual Network with 50 Layers) is a deep convolutional neural network released in 2015 used for image recognition. The model developed by Microsoft Research resolves issues related to deep network training by implementing residual learning blocks with shortcut connections. This feature allows the network to learn residual functions, thus solving the vanishing gradient problem and easing the training of deep models. ResNet50's model consists of 50 layers, which utilise skip connections to preserve information retrieved from previous layers. This further improves the training accuracy and helps ResNet50 excel at image classification. [7]

2.1.4 MobileNet

MobileNet, released by Google in 2017, is a convolutional neural network architecture designed to function on mobile and edge devices with limited computational power. This model utilises depth-wise separable convolutions to minimise the number of parameters and computations required compared to the traditional convolutional layers. This architecture compromises between model accuracy and computational efficiency, making it well-suited for real-time applications on low-computational-resource devices. [8]

2.2 Part 2: Dataset Creation

The creation of a robust dataset consisting of annotated images necessitates both a significant number of images as well as a substantial amount of time. The first step involves selecting a dataset and choosing a predetermined set of labels before the annotation process can begin. Following this, labelled bounding boxes are then placed appropriately over the image. Since this particular project's main focus was object detection, bounding boxes were used, whereas in other projects tackling image segmentation, polygon annotation would be carried out. To retain a model's accuracy, it is crucial that annotations are consistent and precise. This task can be facilitated by utilising Dataset creation tools such as Roboflow [9].

Founded in 2019, Roboflow serves as a user-friendly platform for dataset creation that comes equipped with useful tools, including annotation, augmentation, and organisation. These features further enable developers to efficiently manage and optimise their datasets.

2.3 Part 3: Object Detection

Object Detection is a computer vision task that involves identifying an instance of an object and its location within images or videos. Object detection, unlike object classification, is not only responsible for identifying the presence of an object within a given image or video, rather it determines the location of an object through the use of a bounding box and assign the appropriate class labels to it. The training process, on the other hand is similar to that of an object classification model, making use of a dataset consisting of images with the respective coordinates and labels of the classes within the image. The model is then trained to recognise patterns and extract features using the given dataset to be able to predict the classes and coordinates of objects within the new data. The Intersection over Union (IoU) metric is utilised to better understand the performance of the object detection model. The higher the IoU value, the more accurate the model is. [10, 11]

2.3.1 RetinaNet

RetinaNet is another object detection model, designed specifically to cater for class imbalance and image localisation. Released in 2017 by Facebook AI Research (FAIR), RetinaNet presented the Focal Loss feature, a type of loss function that assigns higher weights to objects that are difficult to detect, thus emphasising the training on challenging examples. This architecture makes use of a feature pyramid network (a network capable of extracting multi-scale features), allowing the model to be able to deal with features of different sizes. Thanks to its one-stage architecture, RetinaNet is capable of combining both object classification and bounding box regression, allowing it to excel in multi-scale object detection scenarios. [12]

2.3.2 YOLOv5

YOLOv5 (You Only Look Once), is one of the most widely used object detection models available. Developed by Ultralytics in 2020, YOLOv5 improves upon YOLO's model legacy with a streamlined architecture that produces adequate results. Unlike previous iterations, YOLOv5 utilises a more lightweight architecture, allowing more user customisation options as well as deployment across multiple platforms. The model comes in multiple variants, including YOLOv5x and YOLOv5s, which can be applied to different scenarios depending on the computational requirements. The results showcased in the upcoming sections utilise the YOLOv5s model. This architecture is widely adopted model With a focus on simplicity, efficiency, and state-of-the-art performance. [13]

2.3.3 YOLOv8

YOLOv8 (You Only Look Once), is another object detection model that further continues to build upon the advancements of its predecessors. Created in 2023, YOLOv8 excels in efficiently processing images or videos while preserving a high level of accuracy. One of the main new features of YOLOv8 is the implementation of CSPDarknet53 [14] as a backbone, which is widely employed for its ability to extract complex features from multiple diverse images. Other variants of YOLOv8, including YOLOv8-CSP, and YOLOv8-Darknet, have been developed to provide a selection of different models to use based on the available computational resources and the required use case. This architecture has been implemented into multiple object detection systems where real-time processing is crucial. The results showcased in the upcoming sections utilise the YOLOv8s model. [15]

2.3.4 DETR

Another model developed by Facebook AI Research is Detection Transformer (DETR). This model, released in 2020, replaces traditional anchor-based methods in favour of leveraging transformers to predict bounding boxes and object labels in one single end-to-end model. In this architecture, a set-based global loss is introduced to be able to manage multiple objects in an image, making it ideal in complex environments. With this design, DETR has provided noteworthy results, especially in multi-scaled images containing small-sized objects. A diagram depicting the DETR architecture can be seen in Figure 2.1. [16]

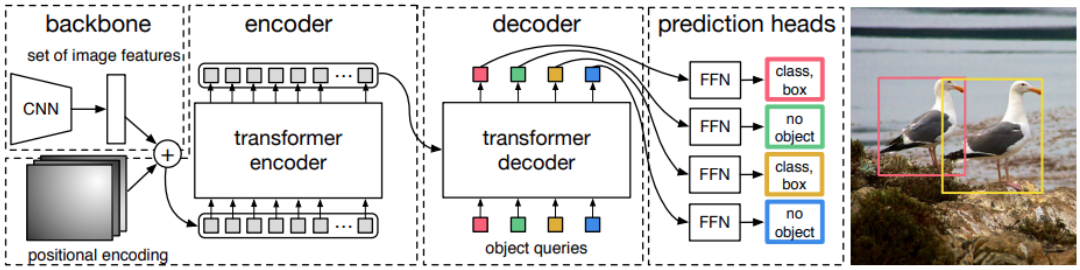


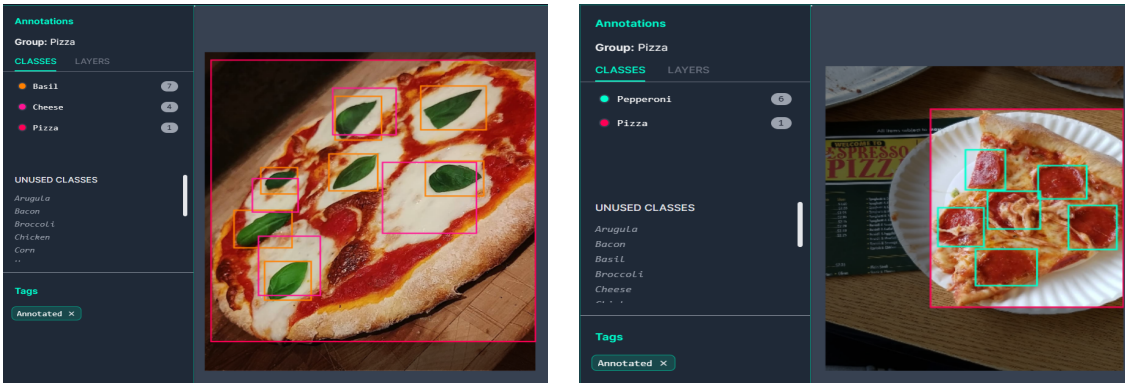
Figure 2.1 DETR Architecture (Source: [16])

3 Data Preparation

To carry out object detection, an annotated dataset was required. This was achieved by utilising a pizza dataset acquired through Kaggle [1]. The chosen dataset consists of roughly 9000 pizza images captured from diverse angles and under varying visual conditions. To align with the specific goals of this research, approximately 1500 images were randomly chosen for annotation. Furthermore, these images were annotated in accordance with the ingredient list below:

1. Arugula
2. Bacon
3. Basil
4. Broccoli
5. Cheese
6. Chicken
7. Corn
8. Ham
9. Mushroom
10. Olives
11. Onion
12. Pepperoni
13. Peppers
14. Pineapple
15. Pizza
16. Tomatoes

The ingredient labels chosen were selected due to their frequent appearance in several pizzas. Examples of the manual annotation process can be seen in Figure 3.1.



Annotation Example 1 Annotation Example 2

Figure 3.1 Bounding Box Annotation Examples

Certain ingredients, such as sardines and nutella, were omitted in favour of streamlining the annotation process and enhancing system performance. Once the dataset was procured and labels chosen, annotation was carried out via the use of RoboFlow [9], a user-friendly dataset creation tool. RoboFlow provided several tools, including label management, bounding box creation, image sorting, and management. Throughout the annotation process, a recurring problem was the mislabelling of ingredients in blurry images. As such, these were omitted in favour of preserving system performance.

Once the annotation process concluded, the dataset was divided into three sets: the training, validation, and testing sets. To create these subsets, a split ratio of 60%, 20%, and 20% respectively, was utilised. The following data-augmentation techniques were then used on the training set:

- **Rotation:** Between -20° and +20°
- **Blur:** Up to 7.75 pixels
- **Bounding Box Blur:** Up to 2.5 pixels
- **Bounding Box Noise:** Up to 5% of pixels

These augmentations were carried out only on the training set, which effectively tripled its size, resulting in a final count of 2544 images. The validation and testing sets comprised of 284 and 283 images, respectively, as can be seen in Figure 3.2. Following data augmentation, the dataset[1] was exported into several formats via Roboflow according to the requirements of the object detection models. This comprehensive data preparation process laid the foundation for the subsequent training and evaluation phases of the object detection model.

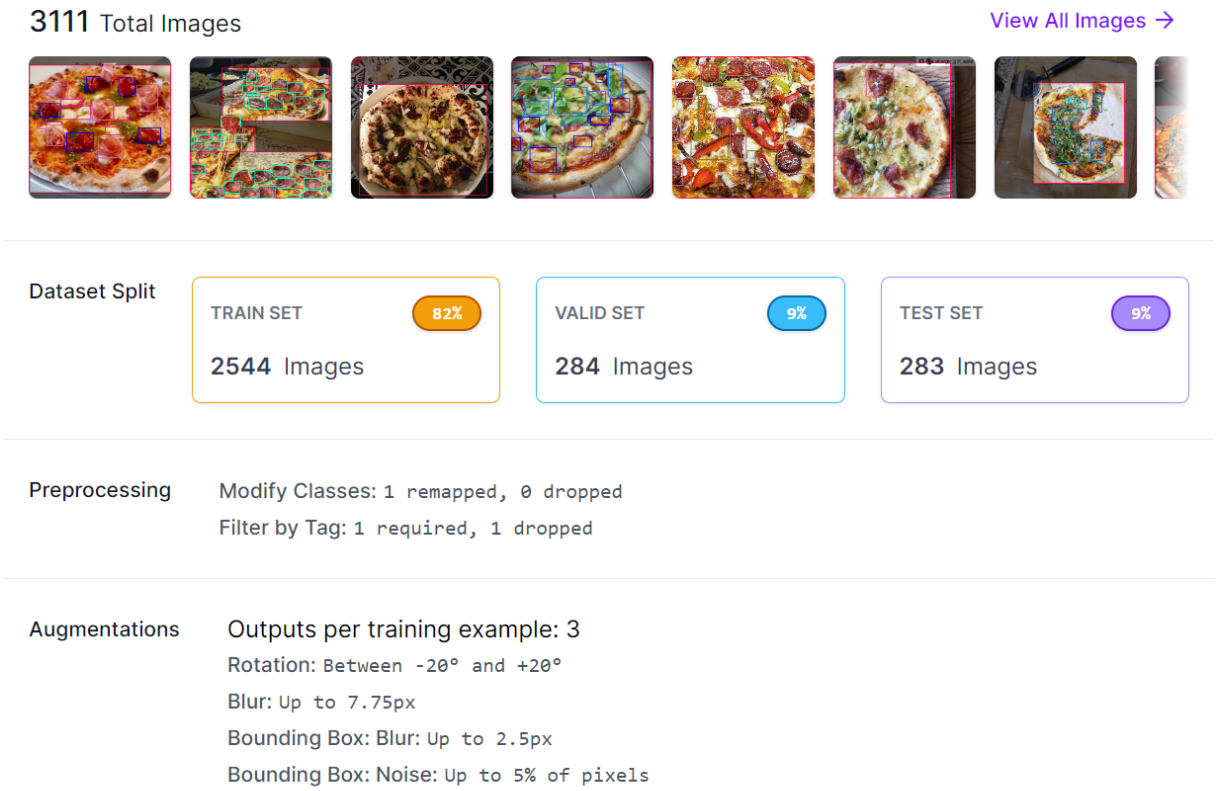


Figure 3.2 RoboFlow Dataset Information Dashboard

[1] *The created dataset can be accessed through: https://app.roboflow.com/advanced-computer-vision-assignment/pizza-object-detector/overview*

Using Roboflow, some interesting graphs were generated. The graph depicted in Figure 3.3 shows the class imbalance in relation to some of the labels. Bacon for instance was under-represented with only 192 examples, whereas Pepperoni was over-represented with nearly 5000 examples.

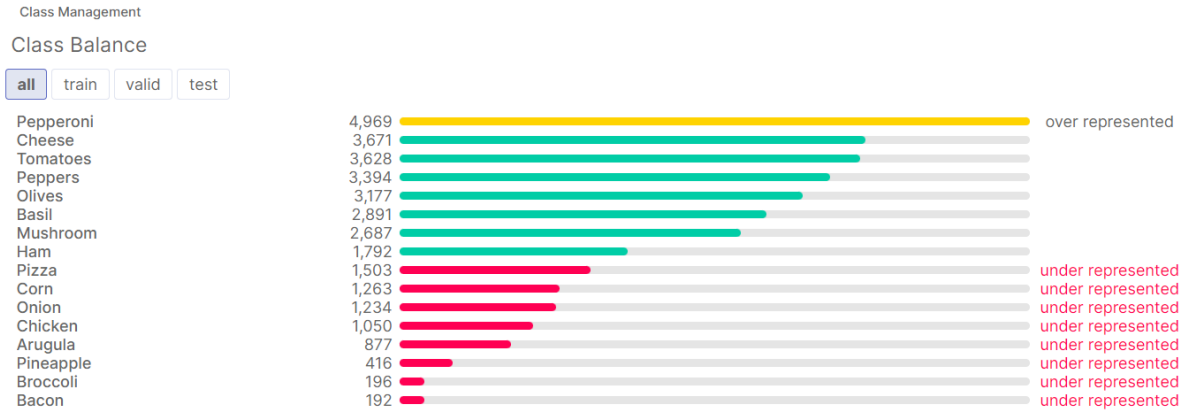


Figure 3.3 RoboFlow Class Imbalance Graph

Figure 3.4 depicts the annotation heat map of bounding boxes, providing that generally the ingredients are placed around the centre of the image.

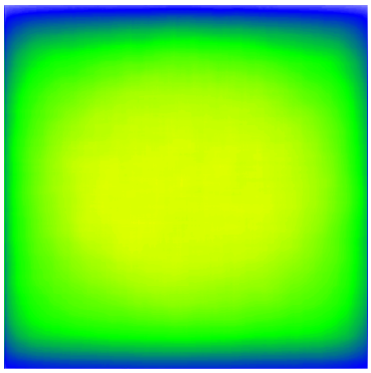


Figure 3.4 RoboFlow Bounding Box Annotation Heatmap

Roboflow also provided a histogram depicting the number of objects within an image, showing that most pizzas had between 2 and 16 images. This can be seen in Figure 3.5.

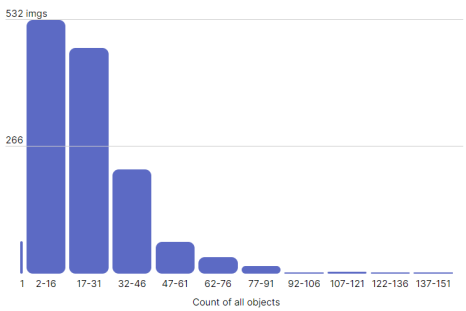


Figure 3.5 RoboFlow Bounding Box Histogram

4 Implementation

4.1 Part 1: Object Classification

In the first section, three classification models were selected to be implemented, these being:

- ResNet50
- VGG16
- MobileNet

The models chosen above were selected due to the fact that, apart from being some of the most popular classification models in the industry, they were not particularly computationally heavy and also offered a diverse range of architectures.

To facilitate comparison between the three aforementioned architectures, the dataset retrieved from Kaggle [2] was used. Prior to feeding the images to the models, they were resized to 224x224. This resizing was performed for two main reasons, these being that certain models take an input size of 224x224, such as the VGG16 [6]. In addition, the aforementioned architectures were chosen because they had the same input size, thus ensuring a fair and equal evaluation.

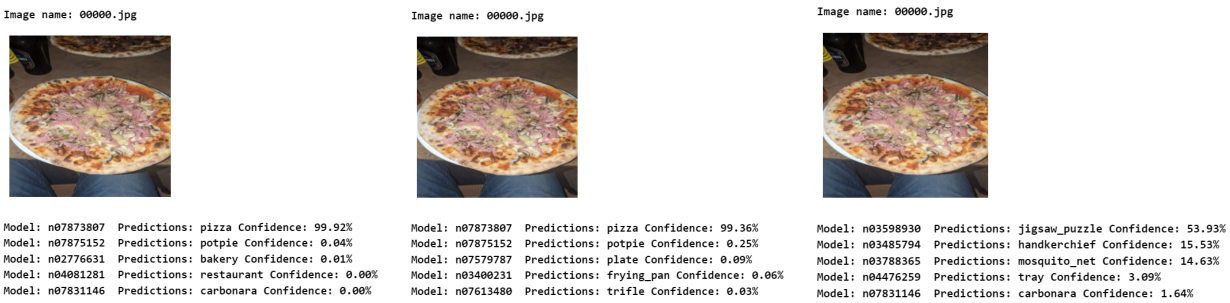


Figure 4.1 Predictions for Pizza Object Classification from several pre-trained models.

After appending all the required images to a list, the classification process could begin. The pre-trained weights for each model were retrieved from ImageNet. Each model is then given a set of images, and the predict function is called on them. The returned list consists of tuples comprising the label identifier, the label name, and the associated confidence of the image containing that label. For each image, the *top five* labels per image were displayed, ranked by confidence in descending order. The results were then saved into JSON files according to the model used. These models, although pre-trained to classify various labels, could only classify a pizza and not the toppings, as none of them were on the list of known labels. An example of the displayed results after classification can be seen in Figure 4.1.

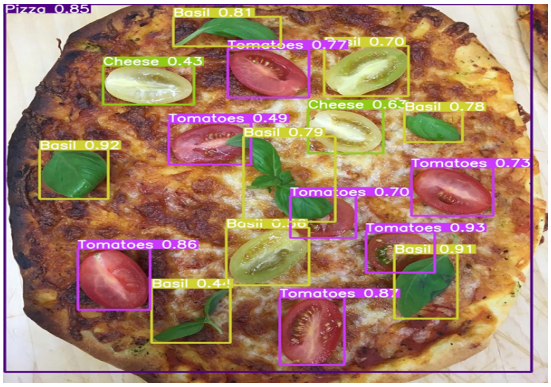
4.2 Part 3: Object Detection

The final part of this project focused on the implementation of three object detection models. The models chosen were:

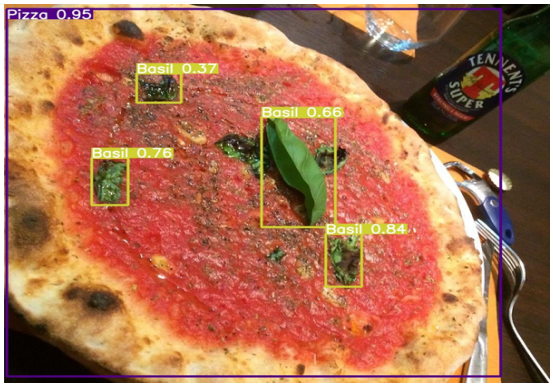
- YOLOv5
- YOLOv8
- RetinaNet

These models were chosen as they are not only popular but well-established, having a wide range of resources and guides explaining how to utilise and train them. Moreover, our exploration aimed to discern potential differences among YOLO, a model specialised in swift object detection with minimal processing time; RetinaNet, tailored for optimal performance in complex, multi-scale object detection environments. DETR, a groundbreaking detection transformer that dispenses with traditional anchor-based methods, was also implemented to a lesser degree and thus will be discussed in this section.

The initial step for all models involved downloading the dataset in the prescribed format. YOLOv8 and YOLOv5 adopted the YOLOv8 PyTorch TXT and YOLOv5 PyTorch TXT annotation formats, while RetinaNet employed the RetinaNet Keras CSV format, and DETR utilised the COCO JSON format for annotations.



YOLOv8 Object Detection Example 1



YOLOv8 Object Detection Example 2

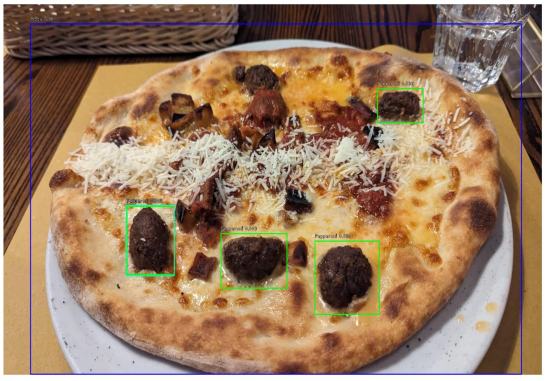
Figure 4.2 YOLOv8 Object Detection Examples

Following this, the training process could begin. For YOLOv8, a YOLO object was first imported, and the hyperparameters, such as the batch size and epoch numbers, were predefined. The model was assigned a dropout rate of 0.5 so that it could learn more generalised features about the ingredients rather than specific features exclusive to the images themselves. Furthermore, early stopping with a patience of 15 was applied to the model to avoid overfitting to the training set. After the training function was called, the validation set and testing set were used to check the overall performance of the model trained. The results were then displayed to the

users, an example of which can be seen in Figure 4.2. A similar process was carried out on YOLOv5, the main difference being that the relevant git repository was cloned as no library was available to facilitate its implementation. The following processes followed similarly to YOLOv8, where, after training, the model was validated and tested. Once the testing was performed, an example of the resultant detected objects was displayed, as can be seen in Figure 4.3.

Figure 4.3 YOLOv5 Object Detection Examples

RetinaNet followed a slightly different approach, the first of which involved updating files within the RetinaNet package itself. Following this, the dataset was downloaded from Roboflow and the classes and files were setup to prepare for the training process. Moreover, the pre-trained weights are downloaded from RetinaNet's Github page [17], and the model is then trained on the dataset using the default hyper-parameters. Testing is then performed on the model to evaluate the its performance. An example of the objects detected in the test set images can be seen in Figure 4.4.



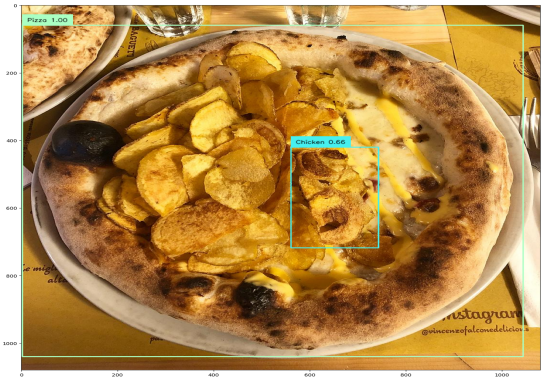
RetinaNet Object Detection Example 1

RetinaNet Object Detection Example 2

Figure 4.4 RetinaNet Object Detection Examples

The final of the four object detectors implemented was DETR, which first involved downloading the dataset from Roboflow. The dataset is then converted into a DataLoader format to then be able to be read and trained on by the DETR model. The DETR model was then initialised and the required settings were set for the training process to begin. The model was then tested and evaluated, an example of the output can be seen in Figure 4.5.

DETR Object Detection Ground Truth



DETR Object Detection Example

Figure 4.5 DETR Object Detection Examples

5 Evaluation

5.1 Metrics

Metrics can be defined as measurements with certain mathematical properties. These metrics are used to measure the performance of models during evaluation.

Accuracy is the metric that describes how a model performs across all classes, which is useful when all classes are given equal importance. This can be seen in Equation 5.1.

$$\textbf{Accuracy} = \frac{\text{True Positive (TP)} + \text{True Negative (TN)}}{\text{Total Number of Samples}} \tag{5.1}$$

Precision is the metric that defines how many items retrieved by the model were relevant. This can be seen in Equation 5.2.

$$\textbf{Precision} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Positive (FP)}} \tag{5.2}$$

Recall is the metric that describes how many of the relevant items were succesfully retrieved. This can be seen in Equation 5.3.

$$\textbf{Recall} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Negative (FN)}} \tag{5.3}$$

The F1-Score is the metric that serves as a harmonic mean between precision and recall. This can be seen in Equation 5.4.

$$\textbf{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

A confusion matrix is a tool used to assess the quality of predictions against an expected ground truth. This can be seen in Equation 5.5.

$$\textbf{Confusion Matrix} = \begin{array}{c|cc} & \textbf{Predicted Positive} & \textbf{Predicted Negative} \\ \hline \textbf{Actual Positive} & \text{True Positive (TP)} & \text{False Negative (FN)} \\ \textbf{Actual Negative} & \text{False Positive (FP)} & \text{True Negative (TN)} \end{array} \tag{5.5}$$

The Intersection over Union (IoU) can be defined as the ratio of the overlapping area ($b \cap g$) of ground truth (g) and predicted area (b) to the total area ($b \cup g$). This can be seen in Equation 5.6.

$$\textbf{Intersection over Union (b, g)} = \frac{\text{area}(b \cap g)}{\text{area}(b \cup g)} \tag{5.6}$$

The Average Precision (AP) is found by working out the average overall precision across each distinct recall level. This can be seen in Equation 5.7, where P denotes the precision for a particular item whilst rel denotes the relevance [0, 1] of the item in question.

$$\textbf{Average Precision} = \frac{1}{n} \sum_{k=1}^{n} \text{P}(k) \cdot \text{rel}(k) \tag{5.7}$$

The Mean Average Precision (mAP) is the product of the precision and recall of the detected bounding boxes. This can be seen in Equation 5.8.

$$\textbf{Mean Average Precision} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{5.8}$$

5.2 Part 1: Object Classification

As an experimental setup, the aforementioned models were evaluated using the Pizza Classification dataset found on Kaggle [2]. Moreover, this dataset was chosen as it provided the necessary tools to assess the quality of the predictions for the loaded classification models, as it had both pizza and non-pizza labels. The experiment was conducted on the training set for the aforementioned dataset, which had roughly 800 pizza images and 800 non-pizza images. In addition, the above metrics were used to compare the results of the different models.

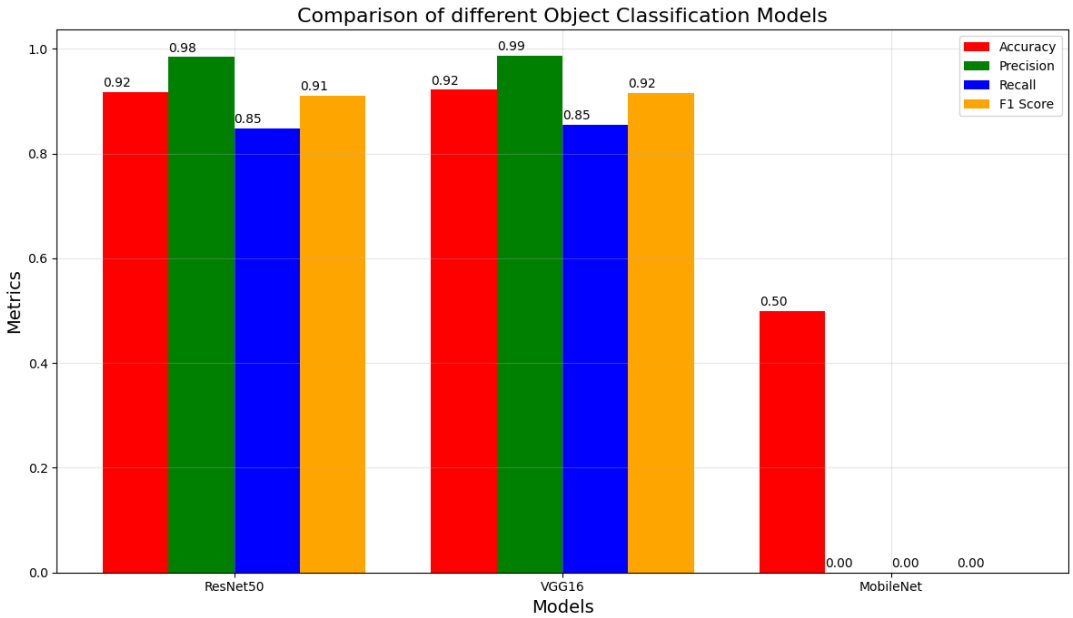
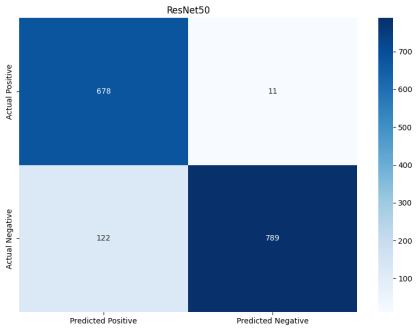
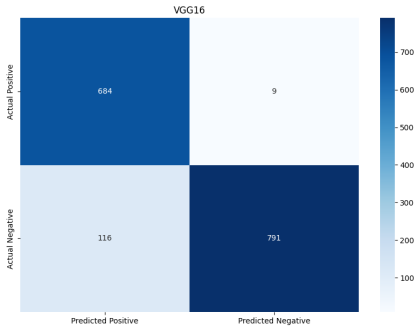


Figure 5.1 Bar graph depicting the metrics variations across the chosen Object Classification Models.

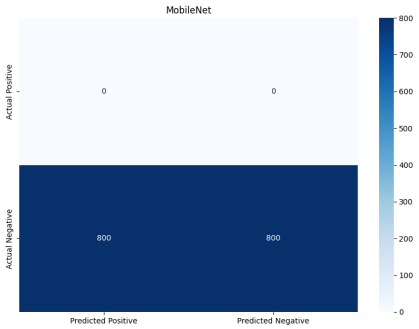
Firstly, the confusion matrices, as can be seen in Figure 5.2, show that for this particular experiment, both ResNet50 and VGG16 performed well. This was proven by both models, which correctly categorised the images with a high success rate. Contrary to this, MobileNet performed relatively poorly, as it managed to only predict correctly all non-pizza items, whereas pizzas were labelled incorrectly or went undetected, leading to the conclusion that MobileNet is not trained to recognise the Pizza label. In Figure 5.1, the prior conclusions drawn from the confusion matrices are further reinstated. ResNet50 and VGG16 performed well, achieving a testing accuracy of 92%, whereas MobileNet performed rather poorly, attaining a testing accuracy of 50%. Overall, VGG16 performed the best, as it obtained a Precision and F1-Score slightly better than ResNet50.



ResNet50



VGG16

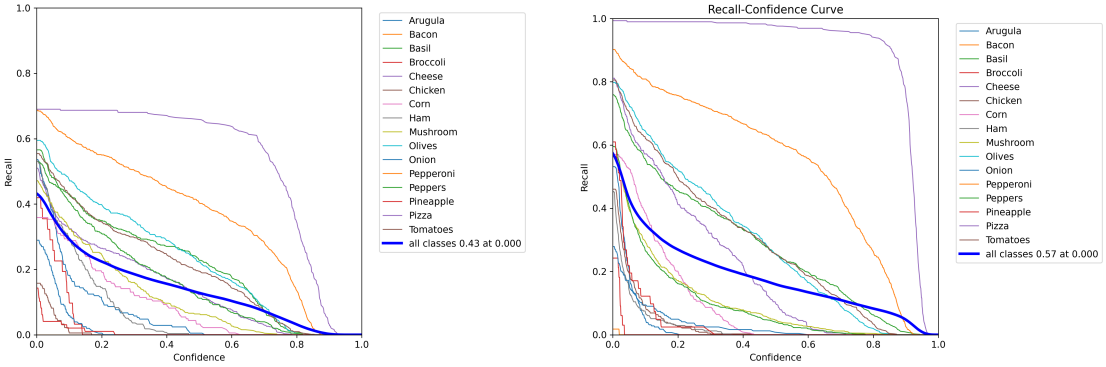


MobileNet

Figure 5.2 Confusion Matrices for the different Object Classification Models.

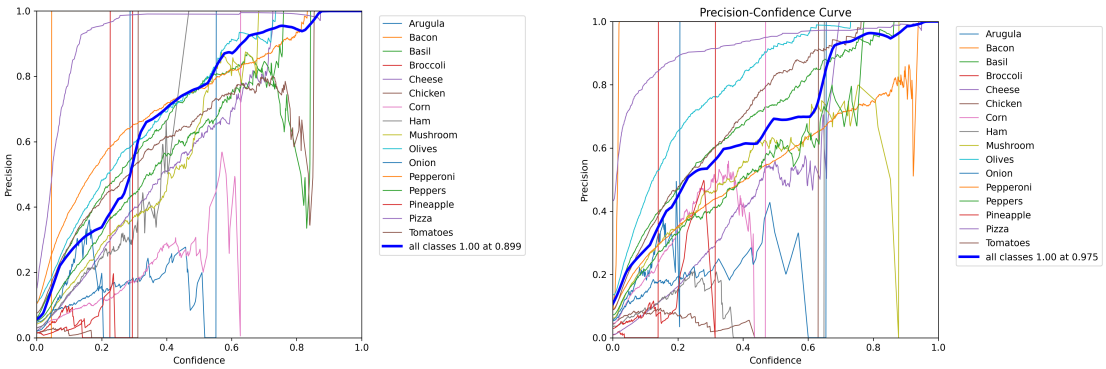
5.3 Part 3: Object Detection

Once all four models were tested, graphs were generated to be able to evaluate and compare each model's overall performance. To perform such evaluation, Precision-Confidence, Recall-Confidence, Precision-Recall along with the F1 curves and Confusion Matrices were displayed. The values for these graphs were calculated using the Equations 5.1, 5.2, 5.3, 5.4, 5.8, 5.6, 5.7 and 5.8. These graphs can be seen in Table 5.1 and Figures 5.3, 5.4, 5.5, 5.6, 5.7 and 5.8.



YOLOv5 Recall Curve YOLOv8 Recall Curve

Figure 5.3 Recall Curves

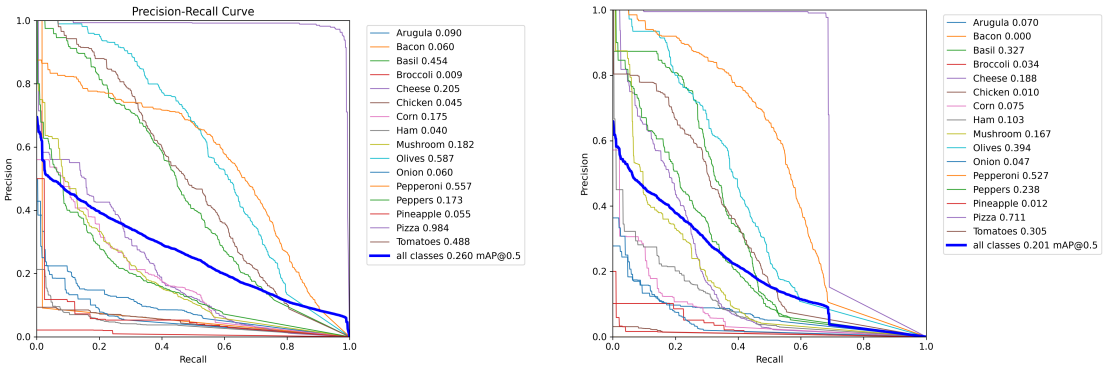


YOLOv5 Precision Curve YOLOv8 Precision Curve

Figure 5.4 Precision Curves

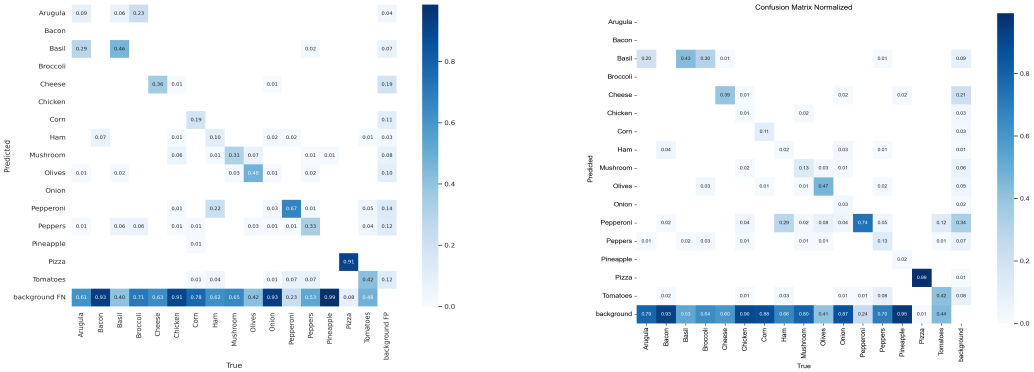
The above diagrams, specifically the YOLO precision, recall and precision-recall graphs can be used to determine the differences in the iteration of YOLO models. In the recall curves YOLOv8 offers better results; this can be noted due to the area under the curve being larger for most ingredients. This can once again be concluded in the precision curves. However, in the precision-recall curves, since YOLOv5 has more consistent gradient reduction in addition to having greater precision the greater the recall, it may seem that YOLOv5 performs slightly better. The reason this could have

occurred is due to the fact that YOLOv8 made use of early stopping, which might have affected the overall model capabilities. The absence of early stopping or increased patience could have resulted in a more accurate comparison, showcasing the discrepancy between the two YOLO instances.



YOLOv5 Precision Recall Curve YOLOv8 Precision Recall Curve

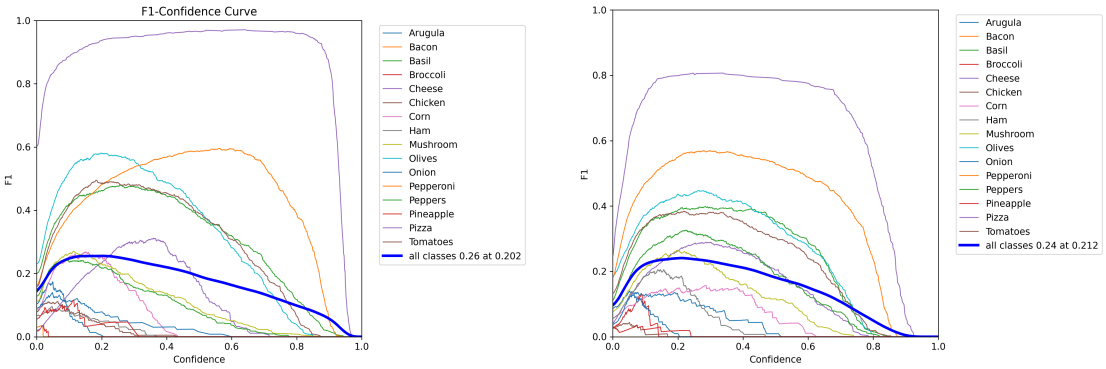
Figure 5.5 Precision Recall Curves



YOLOv5 Confusion Matrix YOLOv8 Confusion Matrix

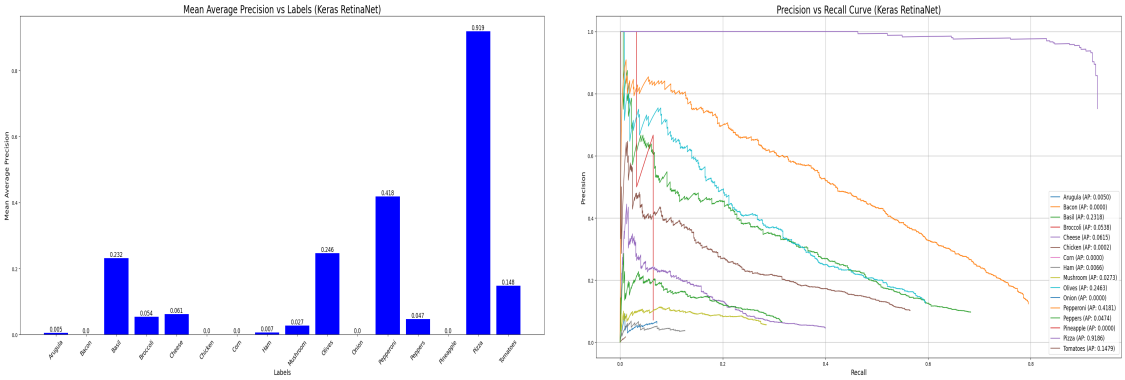
Figure 5.6 Confusion Matrices

In comparison to these two models, RetinaNet produced worse results, with most ingredients having low precision whilst having high recall. This is most likely because RetinaNet's architecture may be deemed a bit outdated compared to the more modern YOLO architectures, although it still produces acceptable outcomes. Hence, the best-performing model was YOLOv8 despite YOLOv5 having a slightly better precision-recall curve. This is proven when comparing the other curves and considering the early stopping functionality mentioned prior. Whilst DETR was implemented, no graphs were depicted; however, a table shown in Figure 5.1 denotes the results of the Average Precision and Recall values.



YOLOv5 F1 Curve YOLOv8 F1 Curve

Figure 5.7 F1 Curves



RetinaNet Mean Average Precision Curve RetinaNet Precision Recall Curve

Figure 5.8 RetinaNet Curves

Metric	IoU Thresholds	Area	AP or AR
Avg Precision	0.50:0.95	all	0.140
Avg Precision	0.50	all	0.269
Avg Precision	0.75	all	0.112
Avg Precision	0.50:0.95	small	0.036
Avg Precision	0.50:0.95	medium	0.080
Avg Precision	0.50:0.95	large	0.156
Avg Recall	0.50:0.95	all	0.077
Avg Recall	0.50:0.95	all	0.174
Avg Recall	0.50:0.95	all	0.213
Avg Recall	0.50:0.95	small	0.043
Avg Recall	0.50:0.95	medium	0.149
Avg Recall	0.50:0.95	large	0.224

Table 5.1 DETR Intersection over Union (IoU) and Average Precision/Recall (AP/AR) Metrics

6 Conclusion

From the above evaluation, in terms of image classifiers, VGG16 performed the best, obtaining the highest precision and F1-Score from all the other models implemented. Given the custom annotated pizza dataset, from the four models implemented, the best performing one was YOLOv8, retrieving the best curves in comparison to other models. In conclusion, this study explores the performance of multiple models in relation to image classification and object detection related tasks using the creation of an annotated dataset. Whilst some models were proven to perform better on this custom annotated dataset, it is also important to note that results may have differed if another dataset was chosen. Furthermore, in relation to the object detection models, performance may have varied if a larger or smaller list of pre-defined labels was chosen.

Appendix A Roboflow Deployment

In addition to the models mentioned throughout the paper, Roboflow provided its own model (Roboflow 3.0) which made use of the aforementioned labelled dataset to facilitate training. The dataset can be found in [18] whilst the resultant model can be used through ones phone via the QR-code presented in Figure A.2 or via the website[1] deployment dashboard as depicted in A.1.

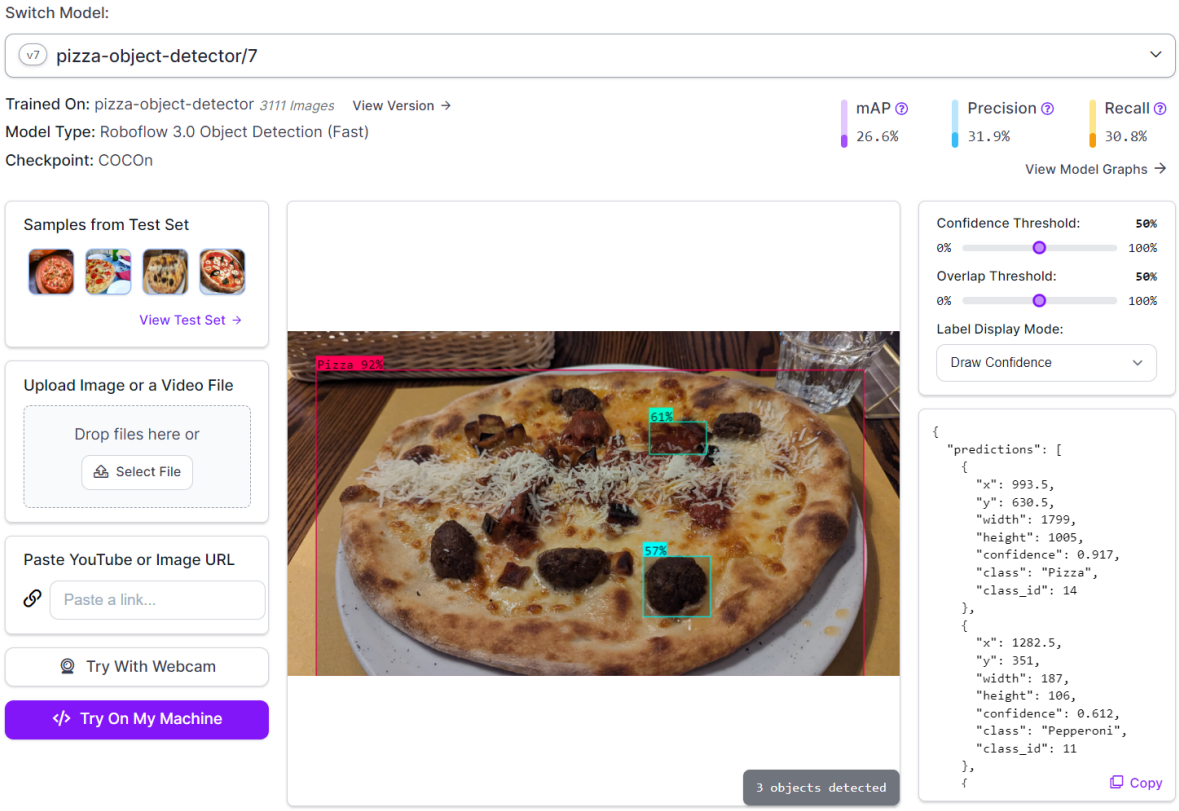


Figure A.1 RoboFlow Deployment Information Dashboard

Try on mobile

Figure A.2 Roboflow 3.0 QR Code

[1] https://app.roboflow.com/advanced-computer-vision-assignment/pizza-object-detector/deploy/7

References

[1] M. Bryant, *Pizza images with topping labels*, https://www.kaggle.com/datasets/michaelbryantds/pizza-images-with-topping-labels/, Jun. 2019.

[2] Project_SHS, *Pizza images with topping labels*, https://www.kaggle.com/datasets/projectshs/pizza-classification-data/, Dec. 2022.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[5] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. arXiv: 1409.0575. [Online]. Available: http://arxiv.org/abs/1409.0575.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[8] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. arXiv: 1704.04861. [Online]. Available: http://arxiv.org/abs/1704.04861.

[9] S. Alexandrova, Z. Tatlock, and M. Cakmak, "Roboflow: A flow-based visual programming language for mobile manipulation tasks.," in *ICRA*, IEEE, 2015, pp. 5537–5544, ISBN: 978-1-4799-6923-4. [Online]. Available: http://dblp.uni-trier.de/db/conf/icra/icra2015.html#AlexandrovaTC15.

[10] A. B. Amjoud and M. Amrouch, "Object detection using deep learning, cnns and vision transformers: A review," *IEEE Access*, vol. 11, pp. 35 479–35 516, 2023. DOI: 10.1109/ACCESS.2023.3266093.

[11] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, *Object detection with deep learning: A review*, 2019. arXiv: 1807.05511 [cs.CV].

[12] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal loss for dense object detection*, 2018. arXiv: 1708.02002 [cs.CV].

[13] G. Jocher, *YOLOv5 by Ultralytics*, version 7.0, May 2020. DOI: 10.5281/zenodo.3908559. [Online]. Available: https://github.com/ultralytics/yolov5.

[14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, 2020. arXiv: 2004.10934 [cs.CV].

[15] G. Jocher, A. Chaurasia, and J. Qiu, *YOLOv8 by Ultralytics*, version 8.0.0, Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

[16] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, *End-to-end object detection with transformers*, 2020. arXiv: 2005.12872 [cs.CV].

[17] Fizyr, *Keras retinanet*, https://github.com/fizyr/keras-retinanet, Mar. 2023.

[18] M. Bartolo, J. Agius, and I. Muscat, *Pizza object detector dataset*, https://universe.roboflow.com/advanced-computer-vision-assignment/pizza-object-detector, Open Source Dataset, visited on 2023-12-31, Nov. 2023. [Online]. Available: https://universe.roboflow.com/advanced-computer-vision-assignment/pizza-object-detector.

Plagiarism Form

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published" (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I/ We* , the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is ~~my~~/ our* work, except where acknowledged and referenced.

I/ We* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

StudentName	Signature
Isaac Muscat	I.Muscat
StudentName	Signature
Jerome Agius	J.Agius
StudentName	Signature
Matthias Bartolo	MBartolo
StudentName	Signature

ARI3129	Advanced Computer Vision for AI Group Project
CourseCode	Title of work submitted

14th January 2024
Date