ARI3129 - Advanced Computer Vision for AI

By Kyle Demicoli and Nicholas Vella

Introduction

In recent years, the field of computer vision has witnessed significant progress, propelled by the development and utilisation of deep learning models in various applications. This paper will delve into the meticulous evaluation and comparison of pre-trained models within the Keras Applications API [1], dataset development and object detector training.

The first task involves exploring three trained architectures VGG16 [2], VGG19 [3] and ResNet50 [4] with ImageNet weights. The goal is to understand how these models perform in classifying pizza images with their complexities. This initial investigation helps us gain insights into the strengths and limitations of each model guiding us towards exploration.

The second task involves creating a curated dataset specifically designed for training custom object detectors. Alongside ImageNet pizza images we incorporate sources. Apply meticulous annotation and augmentation processes to enhance the datasets quality. This task focuses on considerations such as planning, format, configuration and content choices that lay the groundwork for developing object detection models.

The crux of the project unfolds in task three which involves training and evaluating two object detection methods specifically designed for identifying pizza and its toppings. Using TensorFlow PyTorch the models are carefully examined, resulting in a comparison of their performance. The paper also incorporates analytics to reveal insights into the detection process, such as listing the toppings and other relevant information obtained from analysing the images.

Background on the techniques used

Task 1:

Keras Applications API [1]:

The Keras Applications API [1] from the Keras deep learning library provides a set of pre-trained models for diverse computer vision tasks. It makes it easier to apply cutting-edge neural networks, allowing users to benefit from advanced models with little effort. Keras Applications allows you to apply transfer learning on specific applications using models that have been pre-trained on large datasets such as ImageNet.

VGG16 [2] and VGG19 [3]:

VGG16 [2] and VGG19 [3] were introduced by the Visual Geometry Group (VGG) from Oxford University to be known for its simplicity in design. VGG16 and VGG 19 contain 16 layers of weight each, having a small receptive field (3x3). These models are characterized by their simplicity and performance in image classification assignment. VGG architectures played an important role in gaining insights into deep convolutional neural networks and have found popularity among the researchers of computer vision.

ResNet50 [4]:

The ResNet50 [4], from the family of Residual Network (ResNet), solves this problem- training very deep networks. ResNet50 as introduced by Microsoft Research utilises skip-type connections for residual learning to train very deep neural networks. The residual connections solve the issue of vanishing gradient, making it possible for one to train networks with hundreds of layers. The 50-layer ResNet is the standard for a wide range of computer vision applications because it works well and can easily be trained.

These models found in Keras Applications API act as strong instruments for different goals including image recognition, object detection and feature extraction etc. Researchers and practitioners often treat them as a starting point for building customised deep learning models.

VGG16, VGG19 and ResNet50 were used in Task 1 for evaluation and comparison. From the evaluation of Task 1, we can conclude that VGG16 is the most accurate, while ResNet50 is the least reliable from the three models.

Task 3:

YOLO:

The YOLO architecture [7] is a popular object detection model, well-known for its efficiency and speed. YOLO stands for (You only look once) which accurately describes how this algorithm functions. Traditional object detection frameworks typically require more than one step in the process: first, the image must be scanned multiple times to identify different regions of interest. Each region must then be classified to ascertain whether it contains an object of interest. This method may require a lot of time and computational resources.

By only examining the image once, YOLO, on the other hand, streamlines this procedure. Here's how it functions:

- YOLO processes the entire image using a single neural network. This network creates a grid of cells out of the image and simultaneously predicts bounding boxes and probabilities for each cell.

- The algorithm predicts every bounding box and its class probability in a single pass. This differs from other approaches that might need multiple passes in order to identify objects.

- YOLO is faster than conventional two-step models by nature because it examines the entire image during training and testing. It is thus especially well-suited for tasks involving the real-time detection of objects.
- By examining the entire image at once, YOLO considers the image's global context in addition to the local areas that are taken into account for object detection. Accurate object identification can be aided by this, particularly in complex scenes.

A Closer Look:

YOLOv5:

YOLOv5 [9] was released on the 6th of January 2020. It combines the ideas of YOLO with CNN, emphasising accuracy, speed, and simplicity. YOLOv5, which is renowned for its real-time processing ability, provides a balance between accuracy and speed. Because it is based on the PyTorch framework, it is simple to use and deploy. Its object detection capabilities are almost at the cutting edge.

YOLOv7:

YOLOv7 [10] was released on July 6th, 2022. CNN is combined with YOLO techniques in YOLOv7, just like in YOLOv5. Its goal is to increase speed and accuracy over the previous models. When it comes to object detection tasks, YOLOv7 is an excellent choice because it can achieve higher throughput on specific hardware than previous YOLO versions.

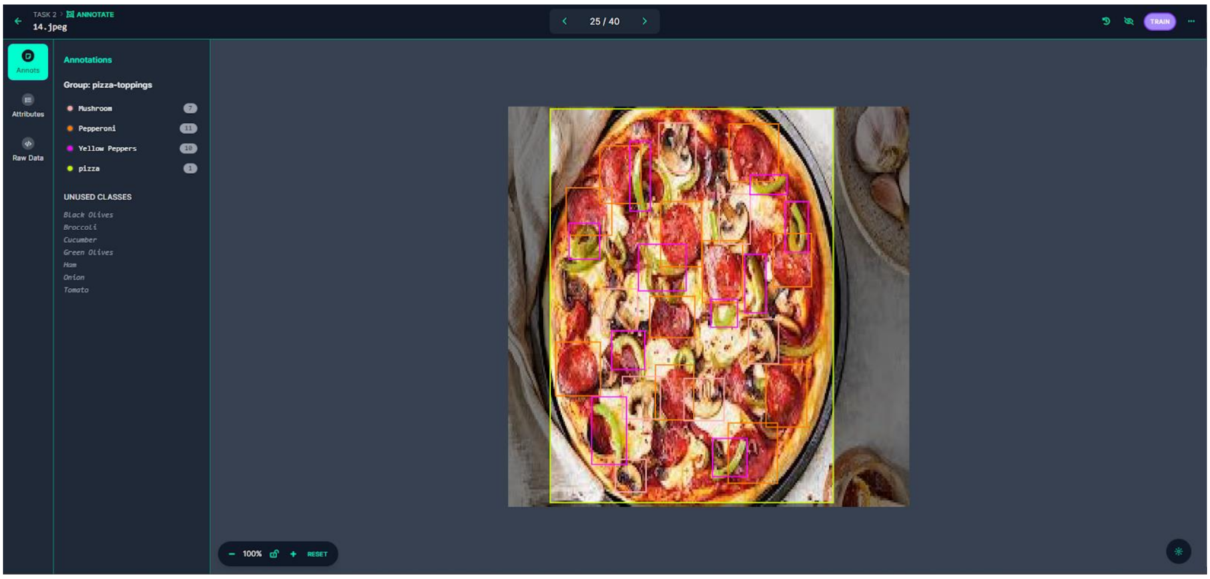
YOLOv8:

Lastly, YOLOv8 [11] was introduced on the 10th of January 2023. In terms of the architecture used, YOLOv8 does away with the need for manually defined anchor boxes by introducing an anchor-free design. In addition to making training easier, this improves the model's capacity to identify objects with different sizes and aspect ratios. Additionally, it includes multi-scale prediction, which enables the model to successfully represent objects at various scales. Thanks to architectural improvements, YOLOv8 has proven to be faster and more accurate than both YOLOv5 and YOLOv7. Because of its optimization for embedded platforms, such as the RTX 4070 Ti and NVIDIA Jetson AGX Orin, performance is dependable and effective.

Data Preparation

A significant stage in the design of strong and effective object detection models is curation and preparation for this dataset. In this project, an inclusive dataset comprising various pizza images was compiled from a GitHub repository [5]. This selection was motivated by the fact that so many pizza composition and toppings are included in this dataset. The need for diversity in the dataset was considered vital to a comprehensive assessment of object detection models because they are designed to function effectively in different real-life situations.

The Roboflow [6] platform helped orchestrate the annotation and organisation of the dataset. While well-known for its convenient interface and robust annotation functions, Roboflow [6] aided in the normalisation of these images into specific classes based on diverse pizza toppings. Pictorially, the annotated classes were Black Olives, Broccoli, Cucumber, Green Olives, Ham, Mushroom, Onion, Pepperoni, Tomato, Yellow Peppers and Pizza.



Annotation Process

Subsequently, the dataset underwent a judicious partitioning into three subsets: training, testing, and validation. The arrangement was deliberately set at 81%, 11%, and 8% to ensure enough data for the model learning while also leaving other sets distinct for rigorous testing and validation.

During the preprocessing stage in Roboflow [6], a series of data augmentation techniques was carefully used to strengthen the model resilience and generalisation capacity.Each training example was subjected to a triad of horizontal and vertical flips, minimal and maximal zoom crops, and rotations within a range of -15° to +15°. Additionally, adjustments to hue and exposure, ranging from -25° to +25° and -10% to +10%, respectively, were introduced.

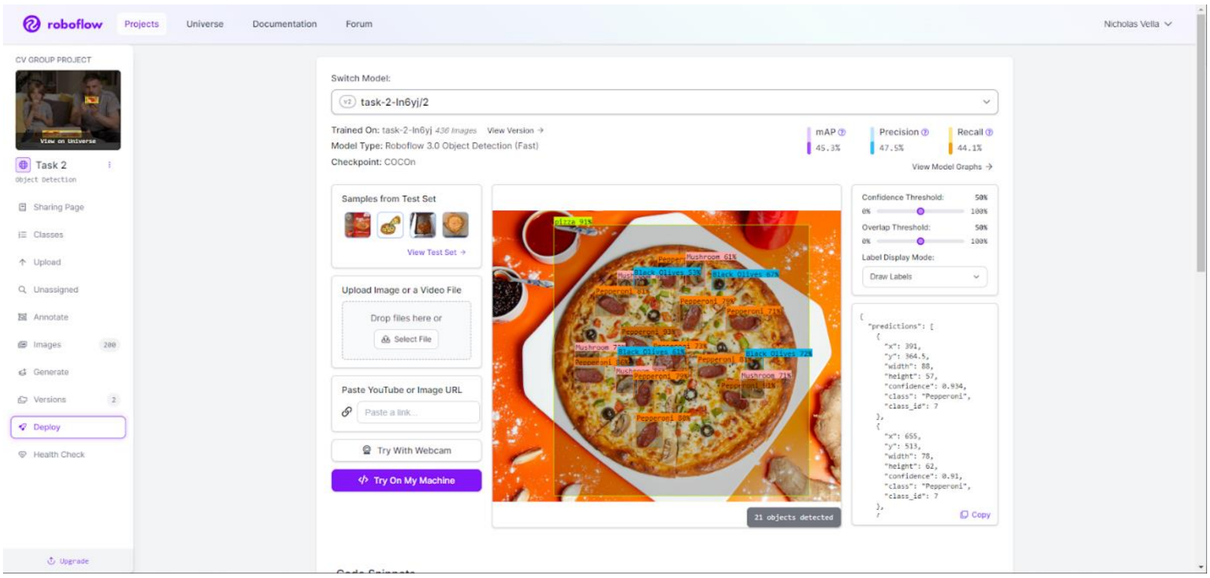
These augmentations were carefully tailored together to introduce variation into the dataset, mimicking different opinions and lighting conditions that models could see in practice use. Through leveraging this augmented dataset of meticulously curated, balanced-class samples with variations added to it, the foundation for training and evaluating optimised models targeted at solving nuanced tasks such as pizza recognition can be established firmly.

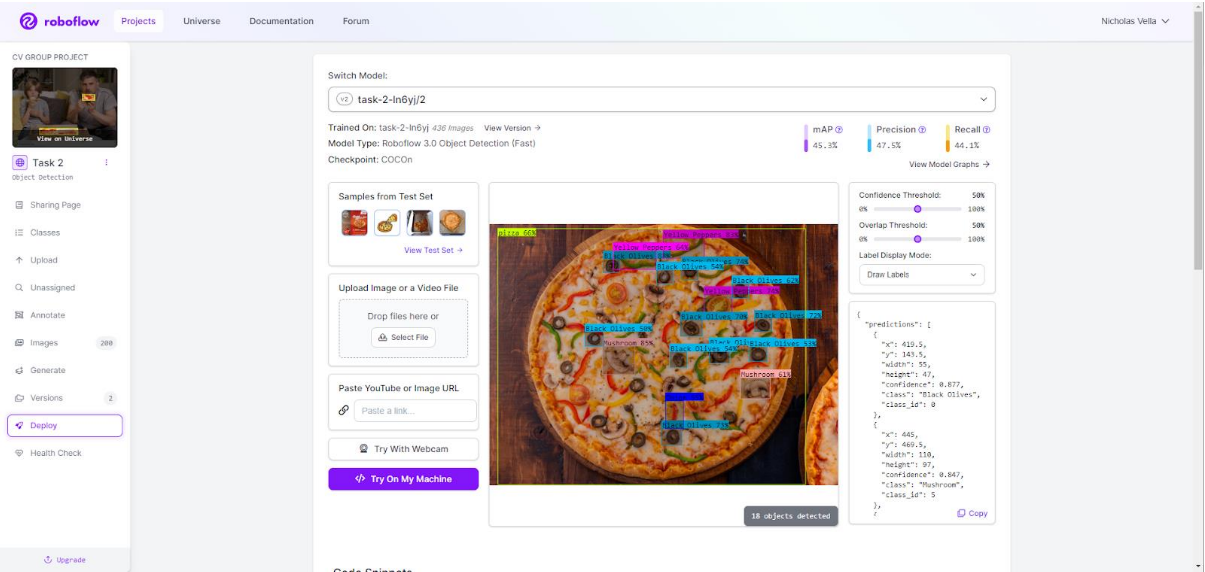
After a careful process of dataset preparation and annotation, the model training within in Roboflow [6] platform was another important validation step. This procedure evaluated annotation accuracy and the appropriateness of a dataset for subsequent object detection tasks. Using the Roboflow [6] infrastructure, testing was also merged with annotation-based workflows to form a unified development cycle. The iterative testing verified the accurate detection of various pizza toppings, claiming that the dataset constructed was usable and credible.

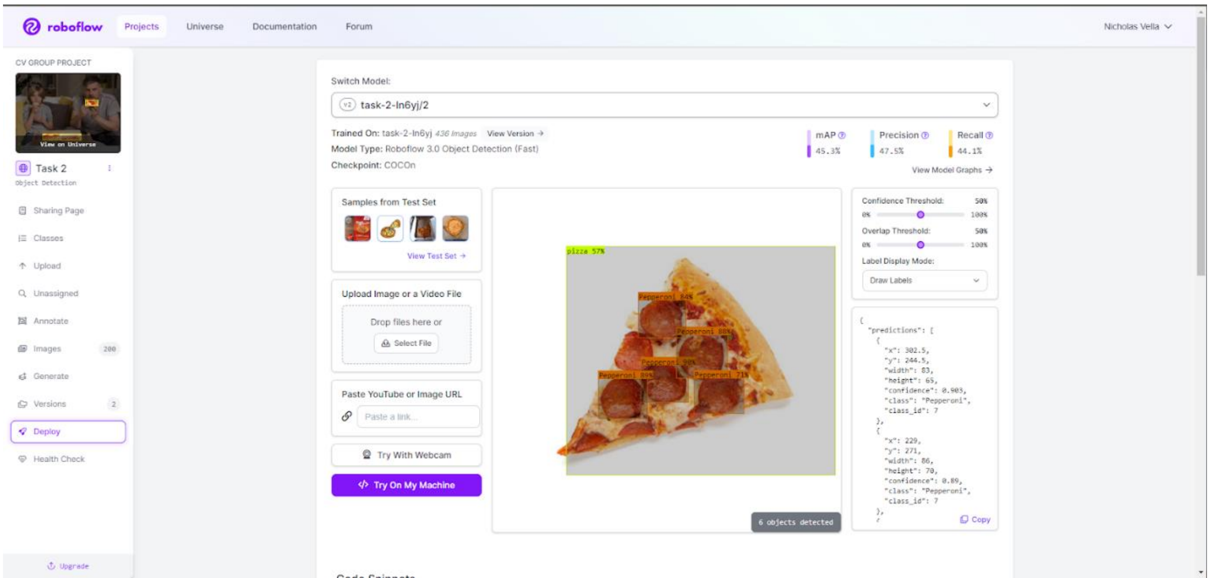
Try on mobile

This is the QR code for testing the Roboflow [6] model with the prepared dataset.

These are some examples of the trained Roboflow model.







Implementation of the object detectors

For this section of the task, intensive research was done to find 3 computer vision model architectures that would be ideal for the detection of pizzas and their toppings. The idea was to use three different architectures mainly:

- Faster R-CNN: In the field of computer vision, Faster R-CNN (Region-based Convolutional Neural Networks) is a well-liked and significant model. It offers better performance in terms of accuracy and speed compared to earlier models such as R-CNN and Fast R-CNN. Faster R-CNN presents a region proposal network that allows almost cost-free region proposals by sharing full-image convolutional features with the detection network. A fully convolutional network, or RPN, predicts object bounds and objectness scores at each position simultaneously. The model has two phases of operation. The RPN carries out the first step, which suggests potential object bounding boxes. These suggestions are used by the Fast R-CNN detector in the second stage to further classify and improve the bounding boxes.

- SSD: Another well-known architecture in the field of computer vision object detection is the Single Shot MultiBox Detector (SSD). SSD completes both tasks in a single pass, in contrast to two-stage models like R-CNN, where one stage proposes regions and the other classifies them. SSDs become faster and more efficient as a result. In order to help the model predict the presence of objects, SSD presents the idea of default boxes, which are a collection of fixed-size boxes. There are several default boxes with varying aspect ratios used for each feature map cell. Finally, SSD is based on common CNN architectures for feature extraction, such as VGG16 (explained in the Background section). Convolutional layers are added to the model in later layers, especially for object detection.

- YOLO: (explained in the Background section)

However, finding models that are up to date was a difficult task, so we had to settle for 3 variations of the YOLO architecture.

- YOLOv5 [9]:
- YOLOv7-tiny [10]:
- YOLOv8 [11]:

The following are the instructions carried out to train such models:

1. Cloning the GitHub repository.

```
git clone https://github.com/WongKinYiu/yolov7.git
cd yolov7
```

   - The first line clones the github repository and places the folder in C:/Users/*username/yolov5/7/8

- o The second line simply moves the command prompt to the correct directory.

2. Installing all necessary Python dependencies.

```
pip install -r requirements.txt
```

- o The above command installs all necessary dependencies for the model to be able to work correctly. In most cases, the cloned repository had a specific requirements.txt file that listed all the information. I rare cases where this was not provided, it was essential to follow the readme.md file to check and install all libraries used.

3. Modifying or creating files to match our database.

```
/yolov7
      /datasets
            /dataset_name
                  /images
                        /train
                              (all training images here)
                        /val
                              (all val images here)
                  /labels
                        /train
                              (all train annotations.txt here)
                        /val
                              (all val annotations.txt here)
```

- o The above shows the correct structure that the files had to follow in order for the model to be trained correctly.
- o Each annotation.txt file stores annotations for a specific image in a list, each annotation having the following format:

```
<class_id> <x_center_norm> <y_center_norm> <width_norm> <height_norm>
```

- o Where:
  - <class_id> shows the specific pizza/topping in the image.
  - <x_center_norm> shows the x-coord of the centre of the bounding box.
  - <x_center_norm> shows the y-coord of the centre of the bounding box.

- <width_norm> holds the width of the bounding box.
- <height_norm> holds the height of the bounding box.

4. Implementing/modifying a configuration file.

```
train: C:\Users\path\to\training\images\file
val: C:\Users\path\to\val\images\file

# Number of classes
nc: 10

# Class names
names: ['pizza', 'Mushroom', 'Pepperoni',
'Yellow Peppers', 'Black Olives', 'Onion',
'Ham', 'Tomato', 'Broccoli', 'Green Olives']
```

- This is a very simple implementation of a configuration file, which was enough to train the model. The content of the configuration file is very self explanatory.
    - train: holding the path to training images.
    - val: holding the path to validation images.
    - nc: holding the number of classes to be detected.
    - names: holding a list of names of all the classes.

5. Training the model.

```
python train.py --img 640 --batch 16 --epochs 50 -
-data pizza_toppings.yaml --weights yolov7-tiny.pt
```

- The above code was used to run the train.py file, which handles the training of the model. The variables passed after holding specific information according to the exact training we want to produce are as follows.
    - --img: holds the size of the images in the dataset
    - --batch: holds the batch size we want.
    - --epochs: holds the number of epochs we want the training to run for.
    - --data: holds the specific config file we want to use.
    - --weights: holds the weights that are to be used according to which version of YOLO is being used.

Evaluation of the object detectors

Different implementations had to be implemented as although similar, all models used had slightly different data handling methods.

The below are the steps taken to evaluate the models:

Load the models:

To load the models, the weights that were produced after training the model were used. Several weight files were produced, one for each checkpoint. The best.pt weight file was used, which was supposed to contain the best weights produced during training. The path to these trained weights was specified when loading the model with the torch.hub.load method.

Data Processing:

Class labels and object bounding boxes are retrieved from the text files. Here, normalization is done. These converts normalized coordinated (concerning image dimensions) to absolute pixel coordinates for accurate bounding box positioning.

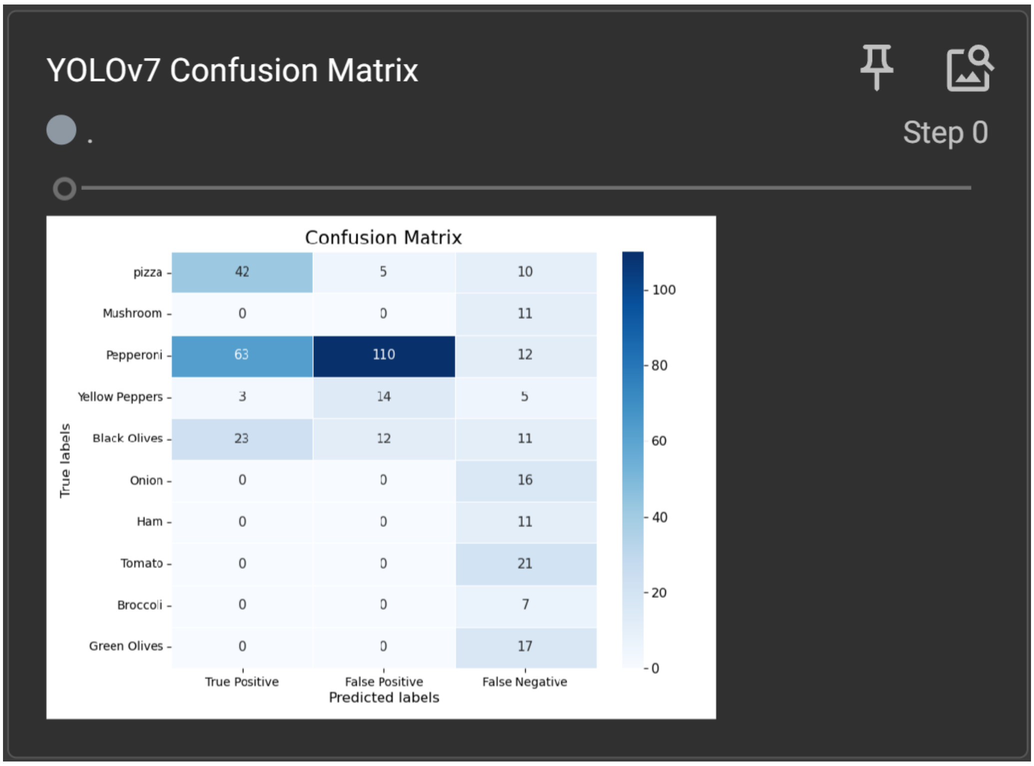
Model Predictions:

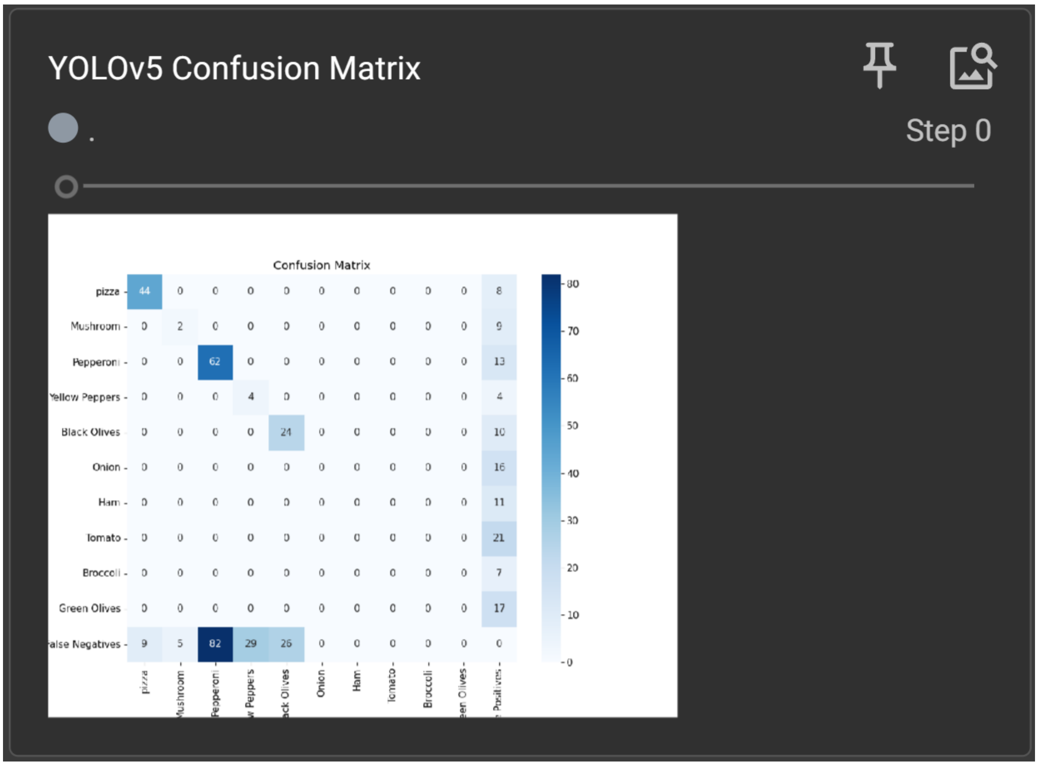
The model predicts object classes and bounding boxes for every image in the testing dataset. This would provide class IDs and bounding box coordinates for each prediction.

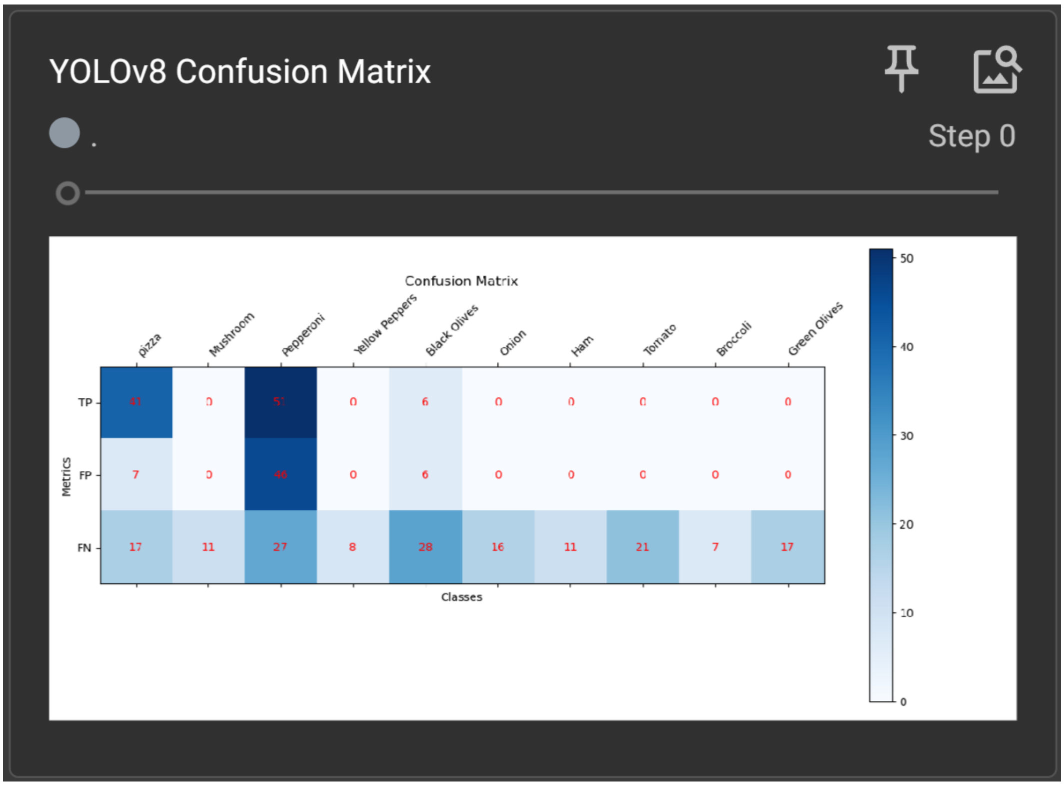
Evaluation Metrics:

This section starts by using the Intersection over Intersections (IoU). This function calculates the bounding box overlap between the actual and predicted boundaries. The next step is the use the IoU threshold and class labels to determine matches. This step is essential to distinguish between accurate predictions (True Positives) and inaccurate predictions (False Positives/Negatives). With this information, it was now possible to calculate some evaluation metrics and figure out how our model was performing.

- Confusion Matrix: In the fields of statistics and machine learning, a confusion matrix is an essential tool for illustrating how well a classification algorithm is working. It is especially helpful for figuring out how well a model is doing in terms of classifying instances correctly and incorrectly. In multi-class problems (like ours), the instances in a predicted class are represented by each column of the matrix, and the instances in an actual class are represented by each row. The number of accurate classifications for each class is displayed by the diagonal elements, while misclassifications are indicated by the off-diagonal elements.


YOLOv7 Confusion Matrix
Step 0
Confusion Matrix
pizza
Mushroom
Pepperoni
Yellow Peppers
Black Olives
Onion
Ham
Tomato
Broccoli
Green Olives
True labels
True Positive
False Positive
False Negative
Predicted labels

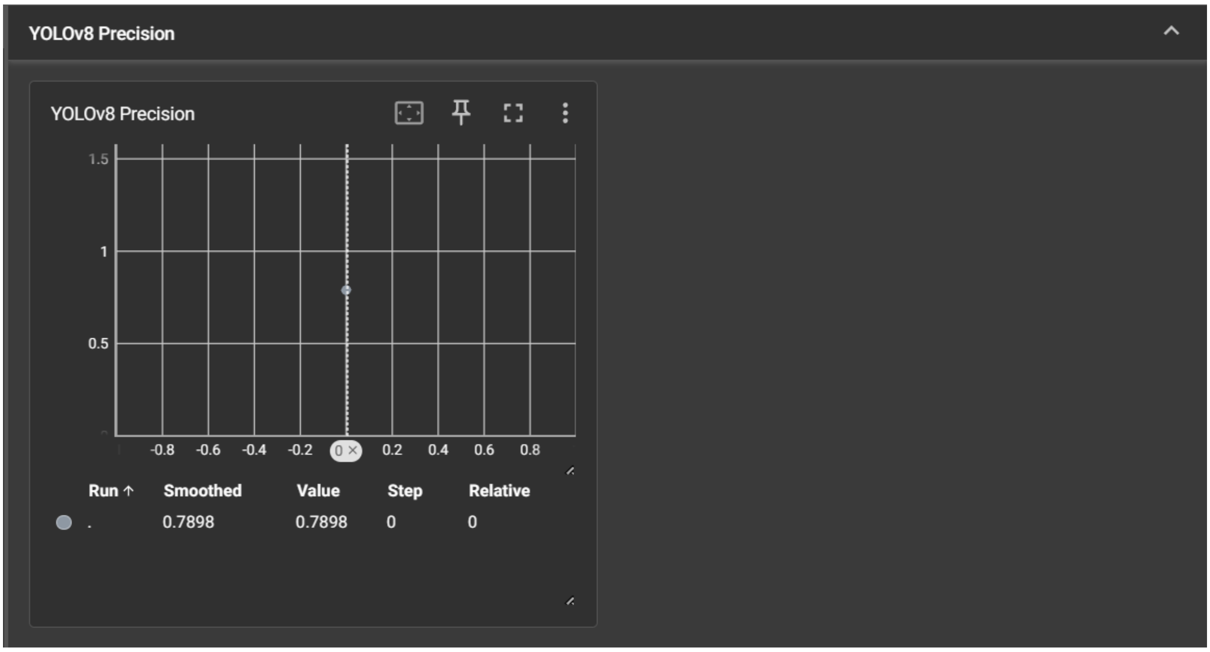

YOLOv5 Confusion Matrix
Step 0
Confusion Matrix
pizza
Mushroom
Pepperoni
Yellow Peppers
Black Olives
Onion
Ham
Tomato
Broccoli
Green Olives

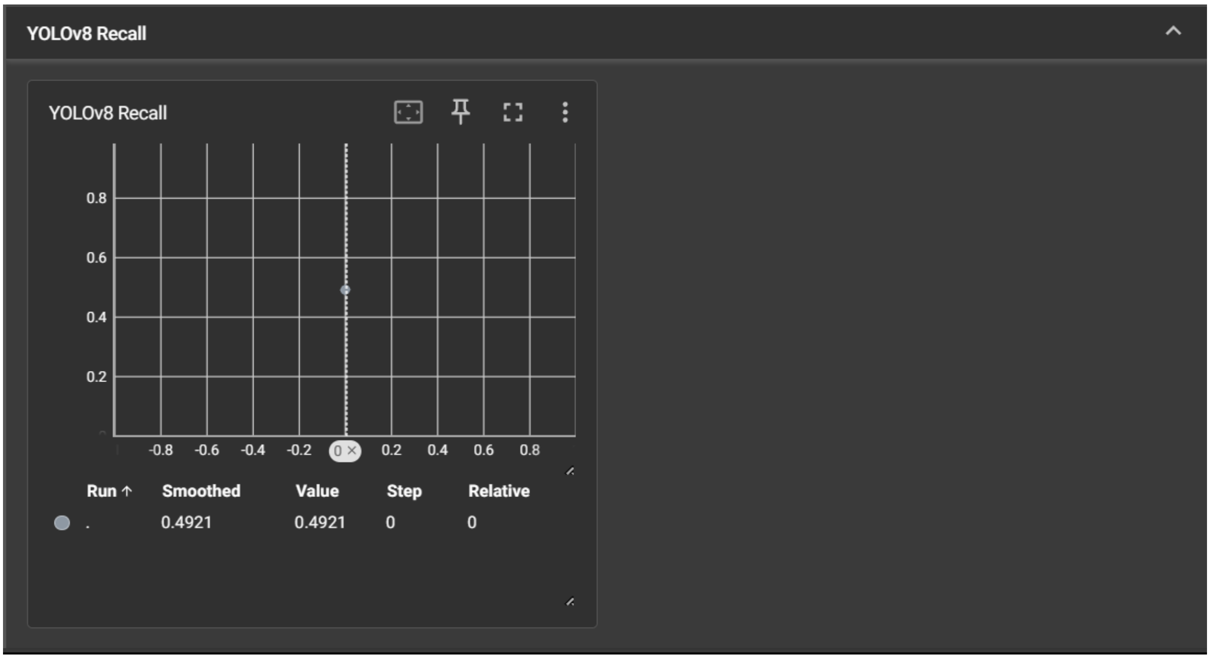


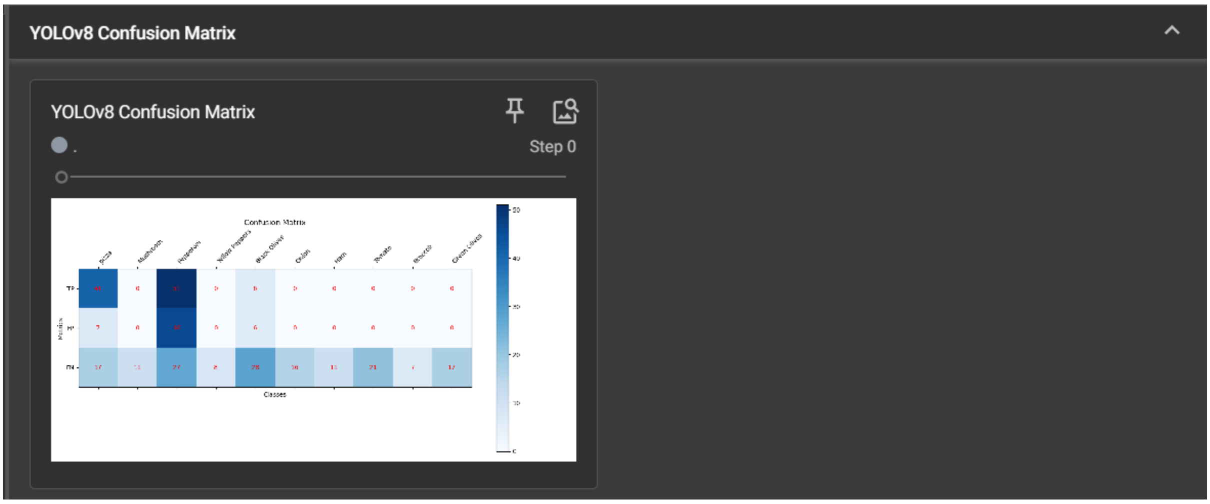
- Precision: Precision [8] is a metric for positive prediction accuracy. It is the percentage of pertinent instances found in the instances that were retrieved. A high precision means that the model produced a significant proportion of relevant results compared to irrelevant ones.

- Recall: Recall [8], which is a substitute for Sensitivity or True Positive Rate, quantifies a model's capacity to locate each and every pertinent case in a dataset. A high recall indicates that the majority of pertinent results were retrieved by the model.

The calculated metrics were all logged to tensorboard for a better, more organised comparison of the models as can be seen in the following page.


YOLOv8 Precision
YOLOv8 Precision
1.5
1
0.5
-0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8
Run Smoothed Value Step Relative
. 0.7898 0.7898 0 0


YOLOv8 Recall
YOLOv8 Recall
0.8
0.6
0.4
0.2
-0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8
Run Smoothed Value Step Relative
. 0.4921 0.4921 0 0


YOLOv8 Confusion Matrix
YOLOv8 Confusion Matrix
Step 0
Confusion Matrix
Classes

Comparing metrics:

- Precision:

```
YOLOv8: Precision = 0.789
YOLOv7: Precision = 0.482
YOLOv5: Precision = 0.2162
```

Notes:

When compared to the other two models, YOLOv8 has the highest precision, meaning that it makes more accurate positive predictions. This suggests that YOLOv8 has a higher chance of being accurate when predicting an object.

Compared to YOLOv8 and YOLOv5, YOLOv7 makes more positive predictions with moderate precision, but fewer accurate ones.

The fact that YOLOv5 has the lowest precision indicates that a sizable percentage of its positive predictions are false positives.

- Recall:

```
YOLOv8: Recall = 0.492
YOLOv7: Recall = 0.5198
YOLOv5: Recall = 0.5333
```

Notes:

Even with its lowest precision, YOLOv5 has the highest recall. This suggests that although it causes more false positive errors, it is more accurate at detecting the majority of positive cases.

Recall for YOLOv7 is marginally lower than that of YOLOv5, indicating that it misses more real positives than YOLOv5 but fewer than YOLOv8.

YOLOv8 has the lowest recall despite having the highest precision. As a result, there are more missed actual positives (false negatives) and more cautious positive predictions.

Overall Comments:

Recall is frequently decreased by increased precision and vice versa. This is referred to as the trade-off between precision and recall. This is evident in our evaluation which makes a lot of sense and is a good sign that everything is working as it should be. In conclusion, every iteration of the YOLO model demonstrates proficiency in various facets of object detection functionality.

Which model to use should be determined by the particular requirements of the application, including whether it's more crucial to guarantee that no real positive is overlooked or to reduce false positives.

Analytics:

After the detection, analytics were also given. The part was implemented in a way that the image is outputted with bounding boxes around the area that were predicted as pizzas or toppings. Each bounding box was also given a label to show what that object was predicted as.

The following shows the output of the algorithm explained above.

References and List of Resources Used

[1] Keras, “Applications,” Keras, 2024. [Online]. Available: https://keras.io/api/applications/. [Accessed: 14-Jan-2024].

[2] Keras, “VGG16 Function,” Keras, 2024. [Online]. Available: https://keras.io/api/applications/vgg/#vgg16-function. [Accessed: 14-Jan-2024].

[3] Keras, “VGG19 Function,” Keras, 2024. [Online]. Available: https://keras.io/api/applications/vgg/#vgg19-function. [Accessed: 14-Jan-2024].

[4] Keras, “ResNet50 Function,” Keras, 2024. [Online]. Available: https://keras.io/api/applications/resnet/#resnet50-function. [Accessed: 14-Jan-2024].

[5] U. Dave, “PizzaDetector,” GitHub, 2024. [Online]. Available: https://github.com/utsavDave97/PizzaDetector. [Accessed: 14-Jan-2024].

[6] Roboflow, “Roboflow,” Roboflow, 2024. [Online]. Available: https://roboflow.com/. [Accessed: 14-Jan-2024].

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[8] Purva Huilgol, Analytics Vidhya. Available: https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/

[9] Ultralytics yolov5. [Online]. Available: https://github.com/ultralytics/yolov5

[10] WongKinYiu yolov7. [Online]. Available: https://github.com/WongKinYiu/yolov7

[11] Ultralytics yolov8. [Online]. Available: https://github.com/ultralytics/ultralytics

Distribution of work:

Nicholas Vella: Task 1, Task 2, AI Journal and Documentation

Kyle Demicoli: Task 3 and Documentation

Generative AI Journal

Introduction

The generative AI model selected for this project is ChatGPT, an advanced language model built by OpenAI on the GPT-3.5 architecture [1][2]. Modern natural language processing models like ChatGPT are well known for their ability to produce text that is coherent and appropriate for the given situation, much like a human [2]. The selection of ChatGPT was made because of its exceptional capacity to comprehend and produce text in response to various cues [2]. This makes it an excellent choice for a variety of uses, such as communication, content production, and problem-solving [3][4][5][6].

The choice to use ChatGPT was spurred by its in-depth training on a wide range of online text, which endowed it with a substantial body of data as of the January 2022 cutoff date [7][8][9][10]. This model has proven to be adept at comprehending and reacting to complex inquiries, enabling lively and interactive engagement [2]. Its adaptability fits very well with the goals of the project as it may be used for anything from text completion to code debugging.

In the following sections of this paper, we discuss ethical considerations one must note when using ChatGPT, the approaches used, the difficulties faced, and the results obtained by using ChatGPT within the parameters of the given project.

Ethical Considerations

Misinformation, Privacy, and Security: Since ChatGPT can produce writing that seems human, there are privacy and security risks as private user information might be accidentally shared or utilized improperly [11][12][13]. Moreover, ChatGPT may be used to produce deepfakes or other false material, escalating worries about the veracity and integrity of digital content [11][12][13]. Strong data protection policies and procedures are needed to address these issues and stop technology abuse [11][12][13].

ChatGPT 3.5 emphasizes a dedication to protecting data privacy and security by using careful design principles [11][12][13]. Strict procedures have been put in place by OpenAI to reduce the dangers of illegal access and possible data breaches [11][12][13]. User interactions with ChatGPT are carefully anonymised, with all personally identifiable information carefully deleted, even while the model is trained on a wide corpus of online language [11][12][13]. OpenAI has a code of ethics and is always improving the model to make it more in line with current privacy best practices [11][12][13]. This guarantee emphasizes how comfortable users are interacting with ChatGPT and how much they value the strong security measures in place to protect the privacy and security of their communications with the model [11][12][13][14].

Incident and Regulatory Response: Notably, in March 2023, Italy's data protection regulator, the Italian Data Protection Authority, GPDP, banned ChatGPT and launched an investigation [15]. This was following an issue when ChatGPT was shut down for a while due to a bug that allowed users to see other users' chat history and conversations.

Bias:

Cultural and Linguistic Bias: ChatGPT may be biased toward particular cultures, languages, or viewpoints that are more frequently expressed online because it was trained mostly on data from the internet [16][17]. As a result, the information that the AI model produces may not fairly represent the variety of human experiences or linguistic expressions [16][17].

Gender and Racial Bias: Due to biases in the training data, ChatGPT may unintentionally reinforce racial and gender prejudices [17][18]. For instance, the model might reinforce preexisting preconceptions by linking particular occupations or professions to particular genders or ethnicities [17][18].

Bias in information Recommendations: ChatGPT may display biases in recommendation systems by giving preference to information that conforms to a user's preexisting beliefs or preferences [19][20][21]. This could lead to polarization and filter bubbles [21].

Methodology

The incorporation of the ChatGPT AI model, into this project primarily played a role in focusing on tasks such, as debugging explaining errors and gathering information. The techniques utilized were designed to make the most of ChatGPT's natural language processing abilities to enhance the results of the project.

1. Debugging and Error Explanation:

ChatGPT played a part in the debugging process by helping to identify and understand errors, in the project. By using prompts related to error messages or unexpected behaviours the system generated responses that offered insights, into potential issues. This made the debugging process more efficient. Helped resolve coding problems quickly.

2. Information Retrieval and Foundation Building:

The model was used to gather information, about the project's area. By creating prompts that asked for explanations or definitions ChatGPT helped develop a comprehension of concepts. This process of retrieving information went beyond programming questions. Included broader contextual inquiries that were pertinent, to the project goals.

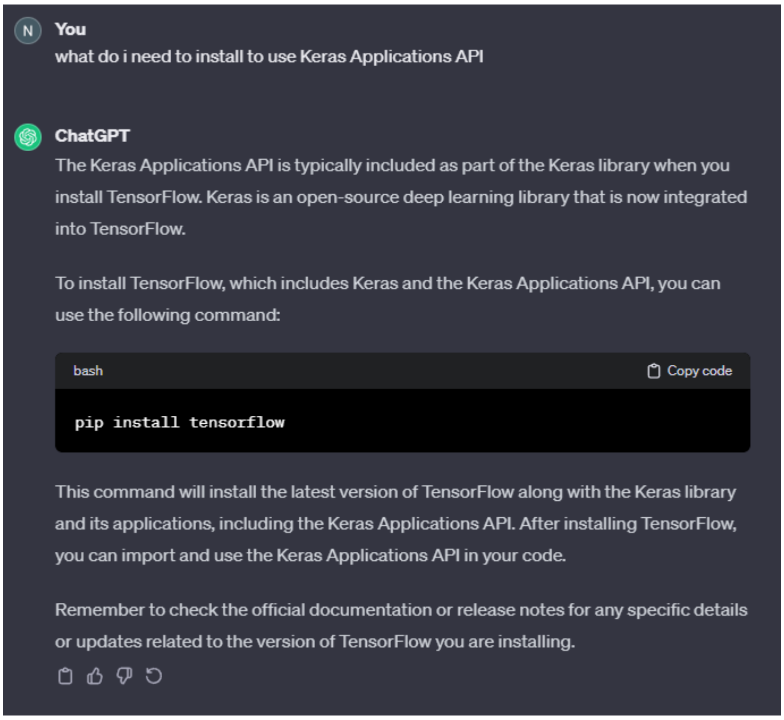
3. Textual Review and Documentation Enhancement:

Notably, the documentation for the project was crafted before the intervention of generative AI. ChatGPT was then utilized to enhance the text, ensuring a meticulous review process that minimized the chance of overlooking any aspect. The model's ability to offer suggestions to improve language and enhance coherence was utilized to improve the quality of the documentation.

While the main focus wasn't on generating content or coming up with ideas, working with ChatGPT in a manner allowing for continuous enhancements, in project-related outputs. By integrating the model into the debugging and information retrieval stages and utilizing it for reviewing documentation the project team leveraged AI's generative power to boost efficiency, knowledge acquisition and overall project material quality.

Prompts and Responses

In this section we list the prompts that were used in ChatGPT. In this section we will provide the responses generated by the AI and will discuss how they helped improve the project. Each prompt and its corresponding response will highlight the collaboration, between input and AI contributions. These prompts played a role, in debugging, gathering information and refining the project.



This was the first prompt, to see what is needed to be installed to uses the Keras Applications API. This response expedited the research process by not only specifying the library needed but how to install it.

Mismatch: True Label: 147.jpeg, Predicted Label: pizza
Mismatch: True Label: 148.jpeg, Predicted Label: crayfish
Mismatch: True Label: 149.jpeg, Predicted Label: rotisserie
Mismatch: True Label: pizza 1.jpg, Predicted Label: pizza
Mismatch: True Label: pizza 2.jpg, Predicted Label: pizza
Mismatch: True Label: pizza 3.jpg, Predicted Label: pomegranate
Mismatch: True Label: pizza 4.jpg, Predicted Label: pizza
Mismatch: True Label: 142 (1).jpeg, Predicted Label: jeep
Mismatch: True Label: 143.jpeg, Predicted Label: breastplate
Mismatch: True Label: 144.jpeg, Predicted Label: jackfruit
Mismatch: True Label: 145.jpeg, Predicted Label: home_theater
Mismatch: True Label: 146.jpeg, Predicted Label: restaurant
Mismatch: True Label: 147.jpeg, Predicted Label: pizza
Mismatch: True Label: 148.jpeg, Predicted Label: candle
Mismatch: True Label: 149.jpeg, Predicted Label: bakery
Mismatch: True Label: pizza 1.jpg, Predicted Label: pizza
Mismatch: True Label: pizza 2.jpg, Predicted Label: pizza
Mismatch: True Label: pizza 3.jpg, Predicted Label: pomegranate
Mismatch: True Label: pizza 4.jpg, Predicted Label: potpie
VGG16 Accuracy: 0.00%
VGG19 Accuracy: 0.00%
ResNet50 Accuracy: 0.00% why there are mistmatch in predicted label pizza

ChatGPT
The issue with mismatches in predicted labels for "pizza" may be due to the fact that the true labels and predicted labels have slight differences in formatting. To address this, you can further modify the custom accuracy function to handle cases where the predicted label contains the true label or vice versa, considering both substrings and avoiding unnecessary replacements.

This was another prompt for Task 1. In this prompt the AI was asked about an unexpected output, the Predicted Label was correct, but it was still identified as a mismatch. The AI suggested that the True Label might not be initialised as "pizza", which was right. Therefore, was very helpful in fixing the logical error in my code.

You
I want to compare 3 pre-trained models in the Keras Applications API. which models do you suggest
‹ 2 / 2 ›

ChatGPT
Certainly! Here's a brief description for the specified models:

1. **VGG16:**
   - **Pros:** VGG16, or the 16-layer Visual Geometry Group model, is known for its simplicity and uniform architecture. It consists of convolutional layers with small 3×3 filters and max-pooling layers. VGG16 performs well on a variety of image-related tasks.
   - **Cons:** The model is relatively large in terms of parameters, which may result in higher computational requirements.
2. **VGG19:**
   - **Pros:** VGG19 is an extension of VGG16, with an additional three convolutional layers. This can provide slightly improved performance on certain tasks. Like VGG16, it is easy to understand and implement.
   - **Cons:** Similar to VGG16, VGG19's drawback lies in its larger size and computational demands.
3. **ResNet50:**
   - **Pros:** ResNet50, or the 50-layer Residual Network, introduced the concept of residual connections, addressing the vanishing gradient problem in deep networks. It excels in training deep architectures and has achieved state-of-the-art results on various benchmarks.
   - **Cons:** While more efficient than VGG models, ResNet50 is still relatively large and may require substantial computational resources.

For this next prompt, the AI was asked to suggest 3 pre-trained models from the Keras Applications API. This was helpful in identifying the 3 best pre-trained models and gave a short description of all models.

```
File ~\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:84, in
_check_targets(y_true, y_pred)
  57 def _check_targets(y_true, y_pred):
  58   """Check that y_true and y_pred belong to the same classification task.
  59
  60   This converts multiclass or binary types to a common shape, and raises a
 (...)
  82   y_pred : array or indicator matrix
  83   """
---> 84   check_consistent_length(y_true, y_pred)
  85   type_true = type_of_target(y_true, input_name="y_true")
  86   type_pred = type_of_target(y_pred, input_name="y_pred")

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:409, in
check_consistent_length(*arrays)
  407 uniques = np.unique(lengths)
  408 if len(uniques) > 1:
--> 409   raise ValueError(
  410     "Found input variables with inconsistent numbers of samples: %r"
  411     % [int(l) for l in lengths]
  412   )

ValueError: Found input variables with inconsistent numbers of samples: [12, 60]
```

ChatGPT

It appears that there is an inconsistency in the number of samples between your true labels and the predictions made by the models. This could be caused by not correctly updating the true_labels list while iterating over the image paths for each model.

For this prompt, the run time error was pasted in ChatGPT, it explained the error and suggested was needs to be charged. This helped in solving this error.

You

I need to train a model using a custom dataset, to detect pizzas and their toppings. Can you suggest what architectures would be good?

ChatGPT

Certainly! Training a model to detect pizzas and their toppings is a computer vision task, and there are several architectures and approaches you can consider. Here are some suggestions:

1. **Convolutional Neural Networks (CNNs)**:
   CNNs are the go-to choice for most computer vision tasks. You can start with a standard CNN architecture like VGG16, ResNet, or Inception, and fine-tune it for your specific task. These architectures have proven to be effective for image classification and object detection tasks.
2. **YOLO (You Only Look Once)**:
   YOLO is an object detection architecture that is efficient and well-suited for real-time applications. YOLO divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell. YOLOv4 and YOLOv5 are popular versions of the YOLO architecture.
3. **Faster R-CNN**:
   Faster R-CNN is another widely used object detection architecture that combines CNNs with region proposal networks (RPNs) for precise object localization. It's a good choice when you need high accuracy.
4. **Single Shot MultiBox Detector (SSD)**:
   SSD is a real-time object detection architecture that combines multiple bounding box predictions at different scales in a single pass. It's known for its speed and accuracy.

This prompt was regarding Task 3, ChatGPT was prompted to suggest what architecture should be used for the project's purpose. ChatGPT gave valuable information on 4 architectures that expedited the research process.

You

When using a YOLO model architecture what structure should the dataset be in?

ChatGPT

When using the YOLO (You Only Look Once) model architecture for object detection, your dataset should be structured in a specific way to train the model effectively. YOLO typically expects a dataset in the format of Pascal VOC or COCO, but the structure can vary depending on the specific implementation or framework you are using. Here are the general requirements for structuring your dataset for YOLO:

1. Image Directory:
   - Create a directory where you store all your images. Each image in this directory should contain the objects you want to detect.
2. Annotation Files:
   - YOLO requires annotation files for each image in your dataset. These annotation files contain information about the objects in the image, such as their class labels and bounding box coordinates. The format of the annotation files depends on the YOLO version and implementation you are using. The most common formats are:
     - Darknet Format (YOLOv2, YOLOv3): Each annotation file should have the same name as its corresponding image file but with a .txt extension, and it should contain lines for each object in the image, specifying the object's class index and bounding box coordinates normalized to the image's width and height.

```php
<class_index> <x_center> <y_center> <width> <height>
```

     - COCO Format (YOLOv4 and later): COCO-style annotation files are in JSON format, containing information about images, annotations, and categories. Each annotation should include the class label, bounding box coordinates, and other relevant information.
3. Classes File:
   - You need a file that lists all the class names used in your dataset. This file is usually a plain text file with one class name per line. YOLO uses class indices to identify objects in the annotation files, so the order of class names in this file should match the class indices used in your annotation files.

```python
class1
class2
class3
...
```

4. Directory Structure:
   - Here is a common directory structure for a YOLO dataset:

```scss
├── dataset_directory/
    ├── images/
    |   ├── image1.jpg
    |   ├── image2.jpg
    |   ├── ...
    ├── annotations/
    |   ├── image1.txt (or image1.json)
    |   ├── image2.txt (or image2.json)
    |   ├── ...
    ├── classes.names
```

For this prompt, ChatGPT was asked for the correct structure the dataset should be when using the YOLO model architecture. This was helpful as the suggested structure was implemented and now it is working.

You

What data does a configuration file (.yaml) contain when talking about object detection tasks?

ChatGPT

A configuration file in YAML format for an object detection task typically contains various settings and parameters required to train, evaluate, or run an object detection model. These configuration files are often used with popular object detection frameworks like TensorFlow Object Detection API, Detectron2, YOLO (You Only Look Once), and others. The specific contents of the YAML file may vary depending on the framework and the project, but here are some common elements you might find in such a configuration file:

1. Model Architecture:
   - Model architecture and backbone (e.g., ResNet, MobileNet, etc.).
   - Number of classes to detect.
   - Input image size and preprocessing options.
2. Dataset:
   - Paths to training, validation, and test datasets.
   - Annotations file format (e.g., COCO, Pascal VOC).
   - Class labels and their corresponding IDs.
   - Data augmentation options (e.g., random cropping, flipping).
3. Training Parameters:
   - Learning rate, learning rate schedule (e.g., step-wise, exponential decay).
   - Batch size.
   - Number of training steps or epochs.
   - Weight decay and regularization settings.
   - Optimizer type (e.g., SGD, Adam).
4. Loss Function:
   - Type of loss function (e.g., cross-entropy, focal loss).
   - Loss function parameters (e.g., alpha, gamma for focal loss).
5. Anchor Boxes (if applicable):
   - Anchor box sizes and aspect ratios for anchor-based detectors like Faster R-CNN.
6. Model Hyperparameters:
   - Dropout rates, normalization methods (e.g., BatchNorm), and other architectural settings.
7. Evaluation:
   - Evaluation metric(s) (e.g., mAP - mean Average Precision).
   - Thresholds for detection confidence.
8. Checkpoints and Logging:
   - Paths to save model checkpoints during training.
   - Logging settings for monitoring training progress.
9. Inference:
   - Confidence threshold for inference.
   - Path to the trained model checkpoint for inference.
10. GPU and Distributed Training:
    - GPU configuration and settings for multi-GPU training (if applicable).
11. Other Custom Settings:
    - Any other project-specific settings or custom components.

Here's a simplified example of what a YAML configuration file might look like:

```yaml
model:
  type: "FasterRCNN"
  backbone: "ResNet50"
  num_classes: 20

train:
  dataset:
    train_data: "data/train_dataset.tfrecord"
    val_data: "data/validation_dataset.tfrecord"
    class_labels: "data/class_labels.txt"
    image_size: [640, 640]
  optimizer:
    type: "Adam"
    learning_rate: 0.001
  batch_size: 16
  num_epochs: 1000

evaluation:
  metric: "mAP"
  confidence_threshold: 0.5

logging:
  log_dir: "logs/"
```

This is just a basic example, and the actual content and structure of the YAML file can be more complex depending on the specific object detection framework and project requirements.

This prompt was about the data configuration file. ChatGPT gave an informative response that helped in structuring the code.

These are some of the useful prompts and responses used in this project.

Improvements and Contributions

The implementation of ChatGPT, in the project resulted in contributions improving aspects of the work. In this section, we will explore the areas where generative AI played a role, in enhancing project outcomes.

5.1. Debugging Precision and Efficiency:

ChatGPT has greatly improved the process of debugging by providing insights, into identified errors. By asking questions, about error messages or unexpected behaviours the model offered detailed explanations that helped quickly resolve coding issues. This not only expedited the debugging phase but also bolstered the overall efficiency of the development cycle.

5.2. Comprehensive Knowledge Acquisition and Tool Recommendations:

ChatGPT showcased its expertise in providing, in-depth insights in the quest for information. It proved instrumental in identifying the pre-trained models offered within the Keras Applications API streamlining the process of selecting a most suitable models that should be used for comparison (In task 1). Additionally, it went a step further by offering recommendations on libraries and guidance, on installation procedures thereby aiding the decision-making process of the project team. This wealth of knowledge extended beyond programming queries. Encompassed broader contextual inquiries aligned with the project objectives.

5.3. Textual Refinement and Documentation Enhancement:

The use of ChatGPT, in the documentation stage demonstrated its capability to enhance the quality of written materials. Even though the documentation was initially created without the assistance of AI, ChatGPT played a role in reviewing and giving advice on how to refine the text. The model's capacity to propose enhancements polishes the language. Improve coherence greatly enhanced the quality of the documentation. This iterative approach highlighted how valuable the model is, in tuning project outputs.

Individual Reflection

My interaction, with AI especially using ChatGPT in this project has been a transformative experience that has significantly impacted my understanding of how AI contributes to academic pursuits.

One of the aspects of integrating generative AI was observing its ability to speed up the process of identifying and resolving errors. ChatGPTs assistance, in recognizing and interpreting coding issues was invaluable. The model's aptitude for comprehending prompts related to programming problems provided insights, which ultimately expedited the resolution of challenges. This enhanced efficiency did not only save time but also improved the overall development workflow.

During the documentation phase, we utilized ChatGPT to refine the text and observed how it significantly improved the quality of the content. The iterative process of leveraging the model to enhance language, coherence and overall document structure showcased its potential to elevate written materials. Collaborating with AI augmentation we were able to produce documentation that not only met but exceeded our projects standards.

After my experience with ChatGPT, my perspective on integrating AI into academic projects underwent a significant transformation. I no longer see it simply as a means to automate tasks, but as an interactive partner that can elevate various elements of the project process. The insights gleaned from this encounter have sparked a deep interest in delving into the potential of generative AI for academic research. With the continuous advancement of AI, the possibilities for innovative use in academic inquiry are endless.

References and List of Resources Used

[1] OpenGenus IQ, “GPT-3.5 model architecture,” 2024. [Online]. Available: https://iq.opengenus.org/gpt-3-5-model/.

[2] OpenAI, “Introducing ChatGPT,” 2024. [Online]. Available: https://openai.com/blog/chatgpt/.

[3] AIMultiple, “50 ChatGPT Use Cases with Real-Life Examples in 2024,” 2024. [Online]. Available: https://research.aimultiple.com/chatgpt-use-cases/.

[4] Interesting Engineering, “ChatGPT: 30 incredible ways to use the AI-powered chatbot,” 2024. [Online]. Available: https://interestingengineering.com/lists/chatgpt-30-incredible-ways-to-use.

[5] MUO, “16 Things You Can Do With ChatGPT,” 2024. [Online]. Available: https://www.makeuseof.com/things-you-can-do-with-chatgpt/.

[6] Upwork, “How To Use ChatGPT: A Beginner’s Guide,” 2024. [Online]. Available: https://www.upwork.com/resources/how-to-use-chatgpt.

[7] GPT Blogs, “ChatGPT: How Much Data Is Used in the Training Process?,” 2024. [Online]. Available: https://gptblogs.com/chatgpt-how-much-data-is-used-in-the-training-process.

[8] Ars Technica, “You can now train ChatGPT on your own documents via API,” 2023. [Online]. Available: https://arstechnica.com/information-technology/2023/08/you-can-now-train-chatgpt-on-your-own-documents-via-api/.

[9] Writesonic, “How To Train ChatGPT On Your Data & Build Custom AI Chatbot,” 2024. [Online]. Available: https://writesonic.com/blog/how-to-train-chatgpt-own-data.

[10] ZeroShotDataAug, “Generating and Augmenting Training Data with ChatGPT,” 2024. [Online]. Available: https://arxiv.org/abs/2304.14334.

[11] Harvard Business Review, “The New Risks ChatGPT Poses to Cybersecurity,” 2023. [Online]. Available: https://hbr.org/2023/04/the-new-risks-chatgpt-poses-to-cybersecurity.

[12] arXiv.org, “Unveiling Security, Privacy, and Ethical Concerns of ChatGPT,” 2023. [Online]. Available: https://arxiv.org/pdf/2307.14192.pdf.

[13] Proofpoint, “What Is ChatGPT - How it Works & Security Risks,” 2024. [Online]. Available: https://www.proofpoint.com/us/threat-reference/chatgpt.

[14] Tech.co, “Does ChatGPT Save My Data? OpenAI’s Privacy Policy Explained,” 2024. [Online]. Available: https://tech.co/news/does-chatgpt-save-my-data.

[15] The Regulatory Review, “Chatting About ChatGPT Regulation,” 2023. [Online]. Available: https://www.theregreview.org/2023/11/08/yin-chatting-about-chatgpt-regulation.

[16] Cado Security, “Experimenting with ChatGPT for Incident Response,” 2024. [Online]. Available: https://www.cadosecurity.com/experimenting-with-chatgpt-for-incident-response.

[17] Medium, “The Hidden Cultural Bias of ChatGPT: Would It Be Your Friend?,” 2024. [Online]. Available: https://medium.com/@diceinrete/the-hidden-cultural-bias-of-chatgpt-would-it-be-your-friend-fe550651ede5.

[18] Axios, “The English-first bias of ChatGPT and similar AI services is a growing …,” 2023. [Online]. Available: https://www.axios.com/2023/09/08/ai-language-gap-chatgpt.

[19] medRxiv, “ChatGPT Exhibits Gender and Racial Biases in Acute Coronary Syndrome …,” 2023. [Online]. Available: https://www.medrxiv.org/content/10.1101/2023.11.14.23298525v1.

[20] “ChatGPT perpetuates racial and gender biases,” AuntMinnie. [Online]. Available: https://www.auntminnie.com/imaging-informatics/artificial-intelligence/article/15660882/chatgpt-perpetuates-racial-and-gender-biases.

[21] “Uncovering The Different Types Of ChatGPT Bias,” Forbes. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2023/03/31/uncovering-the-different-types-of-chatgpt-bias/.