Overview: -

People glancing at an image can instantly recognize what the objects are and where they are located within the image. Precise evaluations of the nature of the things present are aided by a person's knowledge and rapid object detection skills. Researchers are always working on technologies that mimic the quick and accurate object detection of the human visual system.

In computer vision, object detection—which involves locating and identifying items of interest in images or videos—is essential. Among the popular methods for object detection is the You Only Look Once version 8 (YOLO8n) model. YOLO8n improves speed and accuracy in object identification tasks by using cutting-edge deep neural networks to recognize things in real-time. Through the addition of sophisticated features and optimizations, YOLO8n outperforms it. The eighth iteration is represented by the "8" in YOLO8n, which also represents ongoing enhancements and modifications to the model architecture. A significant improvement is the increased class set for object identification, which lets users identify a wider variety of things than the 80 classes included in the default models. The YOLO8n model is intended to perform well in situations where accuracy and quickness are essential. Because of its advanced algorithms, which make it an excellent option for real-time object identification applications, it can process and interpret visual data with efficiency. Furthermore flexible, YOLO8n may be tailored to a variety of use cases by enabling users to train their own custom models to recognize things outside of the specified classifications.

YOLO8n is a state-of-the-art object detection system that builds on YOLOv5s's success. Its enhanced features, more class choices, and real-time processing make it an effective tool for a variety of computer vision applications where precise and quick object recognition is crucial.

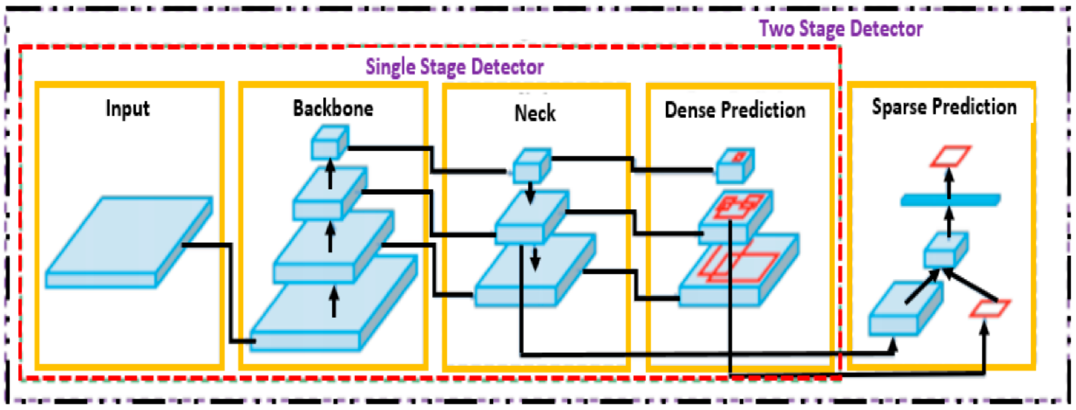


Figure: Model Architecture

In this case, I aim to train and deploy a YOLOv8 model to recognize one classes of objects - Koala - in an image. The process of training an object detection model involves collecting a dataset of labeled images, preparing the data for training, selecting and configuring the YOLOv8 architecture, training the model on the dataset, and evaluating its performance using metrics such as precision, recall, and F1-score. Once the model has been trained, it can be deployed to recognize objects in new images. The ability to detect cats and dogs in an image can have many practical applications, such as in animal monitoring, veterinary medicine, and pet care. By using YOLOv8 to detect these animals, we can accurately and efficiently identify them in a variety of settings, making it an important tool for many industries.

This task will walk I through creating a custom model for the detection and I have completed the entire process by following the subsequent steps:

Data Acquisition and Preparation:

- **Data Acquisition:**
  The initial dataset was collected, resulting in a pool of images. This dataset was then refined by discarding noisy and irrelevant data, leading to a curated set of 191 images.

- **Annotation:**
  Using the single class label "Koala," a third-party web application known as Roboflow was used to annotate the curated dataset in order to find and recognize the koala objects in the images.

- **Data Augmentation:**
  Auto Orientation: The images were automatically oriented for consistency in orientation.
  Resize: To provide standardized input dimensions for the model, every image was scaled to 640 pixels.

- **Image Augmentation:**
  To enhance the dataset and improve model generalization:
  - ✓ **Rotate Transformation:** Rotations between 90 and -90 degrees were applied to the images.
  - ✓ **Brightness Adjustment:** Between -15% and 15%, brightness was arbitrarily changed.
  - ✓ **Saturation Adjustment:** The range of saturation adjustments was randomized from -20% to 20%.

  After augmentation, the dataset expanded to a total of 689 images.

- **Data Distribution:**

  Training and validation sets were created from the augmented dataset. The remaining 2% (10 images) of the data were set aside for validation, leaving 98% of the total (660 images) in the training set. The goal of this distribution is to validate the YOLO v8n model using unseen data after it has been trained on a variety of datasets.

  The final Data set is exported from Roboflow as a YoloV8 format.

Setting up to the model training environment:

Initially, I accessed the command prompt (cmd) within the directory where my data is located. Next, I proceeded to install Ultralytics, ensuring that all the necessary requirements were also installed. Subsequently, I structured the annotated and augmented dataset by creating a YAML file that includes my data directory and adheres to the YOLO v8n model specifications. This file ensures that the image file formats, annotations, and class labels are in accordance with the model's expectations.

- **Training and validating the mode, including several steps:**

  After that, I used the following command to start the model training process:

  ```
  yolo detect train data=data.yaml model=yolov8n.yaml pretrained=yolov8n.pt epochs=50 batch=8 imgsz=640
  ```

  The YOLO v8n model configuration file (`yolov8n.yaml`), pre-trained weights (`yolov8n.pt}), the data configuration file (`data.yaml}), and certain training settings, such as the number of epochs (50), batch size (8), and image size (640). By running this command, the YOLO v8n model began to be trained with the goal of optimizing its performance over the designated number of epochs.

- **Training:**

  With 660 images in the training set, the prepared dataset was used to train the YOLO v8n model. In order to precisely identify and locate koala items in the photos, the model's parameters had to be optimized throughout the training phase. The algorithm gained the ability to recognize unique koala characteristics from the annotated data during training. Iterations were used to minimize the loss function while modifying the model's weights to enhance prediction accuracy. In order to provide reliable detection with a low number of false positives and false negatives, the training procedure sought to strike a compromise between accuracy and recall. The model's performance on the particular objective of koala detection was optimized by carefully selecting and fine-tuning the training hyperparameters, which included learning rate, batch size, and epochs.

- **Validation:**

  Ten images comprised the validation set, which was used to evaluate the model's generalization and performance on data that had not been seen before. The model's accuracy, recall, precision, and overall efficacy were assessed by comparing its predictions with the ground truth annotations. The validation procedure revealed information about the robustness of the model and assisted in locating possible overfitting or underfitting problems. To enhance the model's performance on real-world data, the validation findings might be used to inform modifications to the hyperparameters or more training rounds.

Evaluation Metrics:

Mean Average Precision (mAP) and Intersection over Union (IoU) are two common measures that were used to assess the model's performance quantitatively throughout training and validation. These metrics offered a thorough evaluation of the YOLO v8n model's performance in locating and identifying koala items within the supplied photos. With the help of the validation findings, iterative modifications to the model and hyperparameters were performed to make sure the YOLO v8n model could reliably and accurately detect koalas in a variety of settings. Confusion Metrix, Normalized Confusion Metrix, F1 Curve, Labels, Labels Correlogram, Precision Curve, PR Curve and Recall Curve illustrate in bellow:

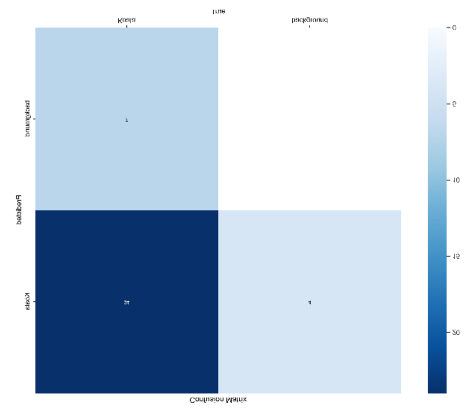


Figure: Confusion Metrix

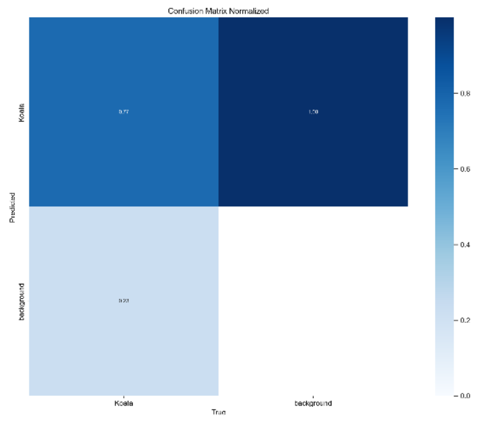


Figure: Normalized Confusion Metrix

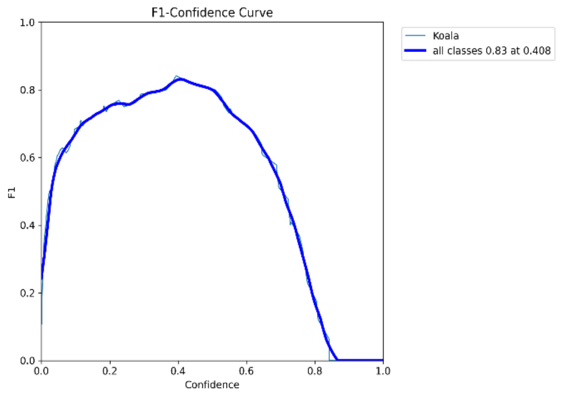


Figure: F1 Curve

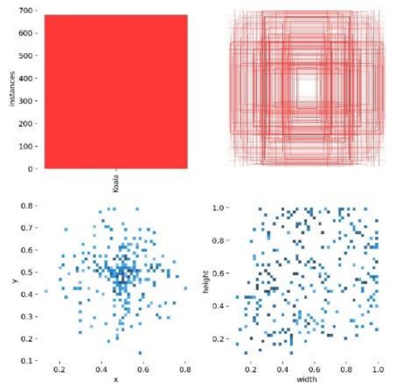


Figure: Labels

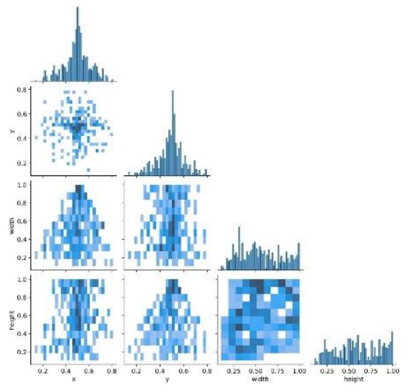


Figure: Labels Correlogram

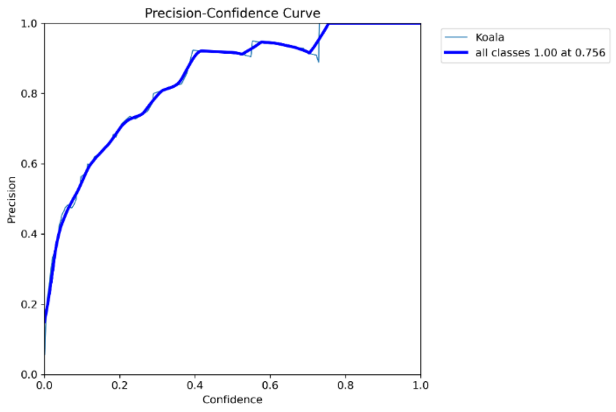


Figure: Precison Curve

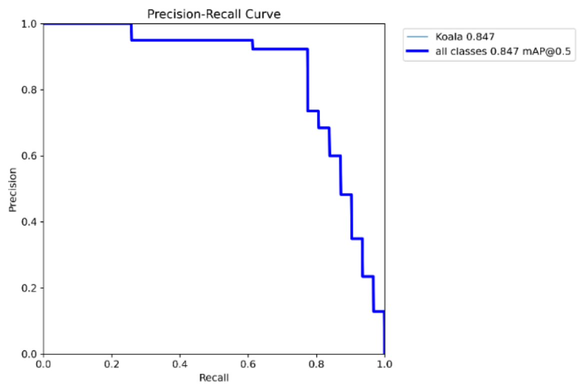


Figure: PR Curve

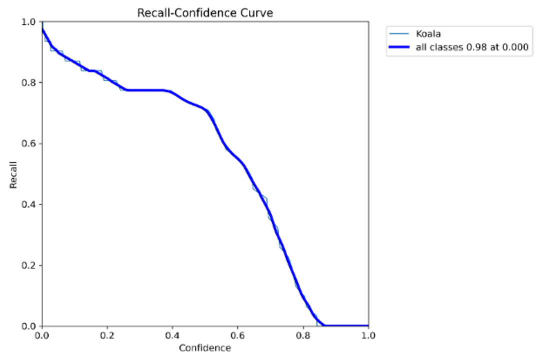


Figure: Recall curve

Detect from unknown image:

And then I used this command to do object identification over pleasant unseen data:

yolo predict model='H:/Koala/Data/runs/detect/train/weights/best.pt' source='C:/Users/ASUS/Downloads/detec'

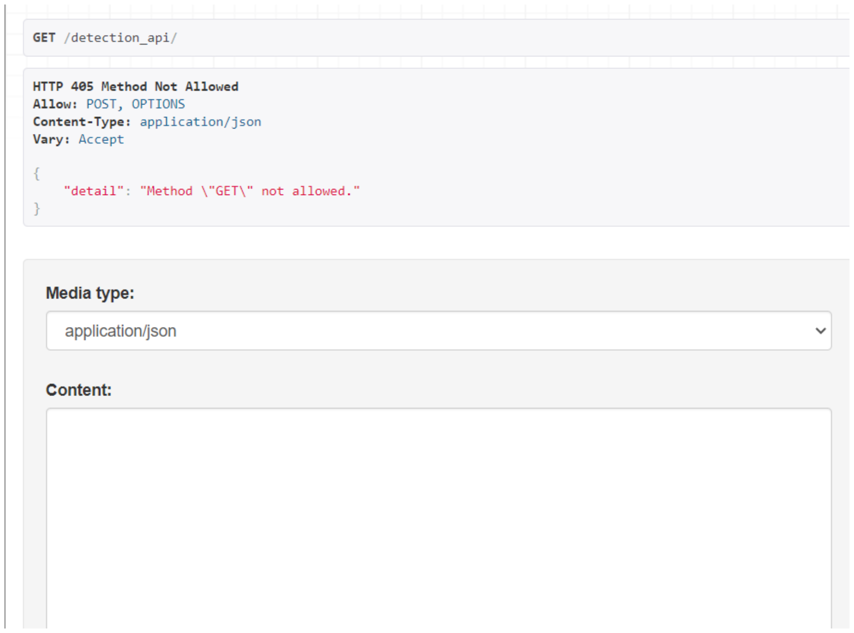
The trained model to predict and identify objects in the source directory ('C:/Users/ASUS/Downloads/detec') may be found at the given location. Through this prediction procedure, the model's generalization and object detection accuracy in previously unknown data were evaluated.

Figure: Detect Unseen Data

API:

A Django project and a Django app must first be created in order to construct a Django REST API for decoding an image to a base64 representation and providing a status class object name. Define a Django model to represent my status class object, including any pertinent fields, after installing the required packages (`django{, `djangorestframework}, and `ultralytics}). Next, implement the decoding logic in a Django view function inside your application. Utilize the serializers provided by the Django REST framework to handle the input and output data types. Decode the image to base64 within my view method, then build the response using the necessary status class object. Set up your URLs so they point to the newly generated view function. Additionally, you might wish to configure any databases or other Django settings that are required. Lastly, to test your API, launch the Django development server. Don't forget to use migrations to build the database tables my model requires. The Django REST framework's capabilities for error handling, request validation, and documentation are used to provide a reliable and thoroughly documented API. Additionally, if authentication and authorization are required for my application, think about safeguarding your API.

Figure: Input Image



```
GET /detection_api/
```

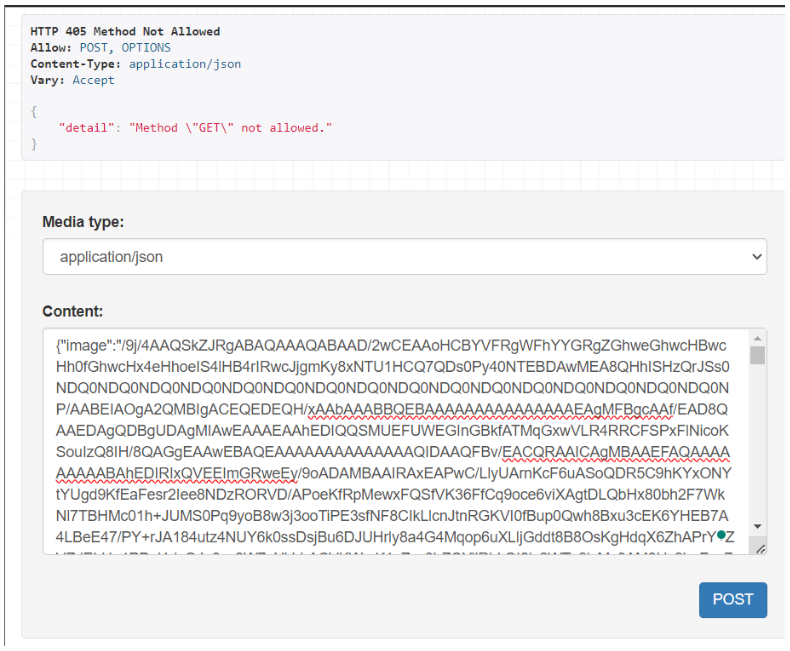
```
HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Method \"GET\" not allowed."
}
```

Media type:

application/json

Content:



```
HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Method \"GET\" not allowed."
}
```

Media type:

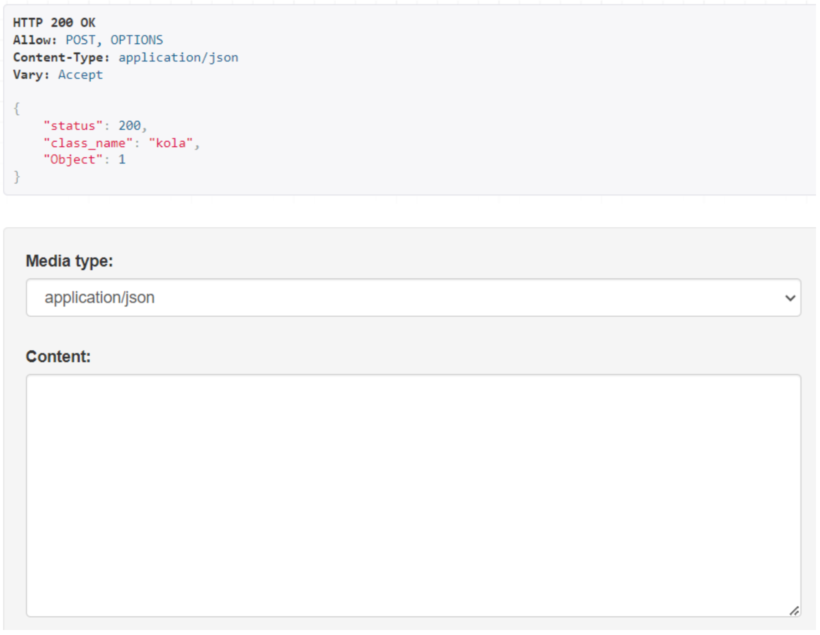
application/json

Content:

{"image":"/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAoHCBYVFRgWFhYYGRgZGhweGhwcHBwc
Hh0fGhwcHx4eHhoelS4lHB4rIRwcJjgmKy8xNTU1HCQ7QDs0Py40NTEBDAwMEA8QHhlSHzQrJSs0
NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0NDQ0N
P/AABEIAOgA2QMBIgACEQEDEQH/xAAbAAABBQEBAAAAAAAAAAAAAAAEAgMFBgcAAf/EAD8Q
AAEDAgQDBgUDAgMIAwEAAAEAAhEDIQQSMUEFUWEGInGBkfATMqGxwVLR4RRCFSPxFlNicoK
SoulzQ8IH/8QAGgEAAwEBAQEAAAAAAAAAAAAAAAQIDAAQFBv/EACQRAAICAgMBAAEFAQAAAA
AAAAABAhEDIRIxQVEElmGRweEy/9oADAMBAAIRAxEAPwC/LlyUArnKcF6uASoQDR5C9hKYxONY
tYUgd9KfEaFesr2lee8NDzRORVD/APoeKfRpMewxFQSfVK36FfCq9oce6viXAgtDLQbHx80bh2F7Wk
Nl7TBHMc01h+JUMS0Pq9yoB8w3j3ooTiPE3sfNF8ClkLlcnJtnRGKVI0fBup0Qwh8Bxu3cEK6YHEB7A
4LBeE47/PY+rJA184utz4NUY6k0ssDsjBu6DJUHrly8a4G4Mqop6uXLIjGddt8B8OsKgHdqX6ZhAPrY●Z

POST

Figure:- Input String



```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "status": 200,
    "class_name": "kola",
    "Object": 1
}
```

Media type:

application/json

Content:

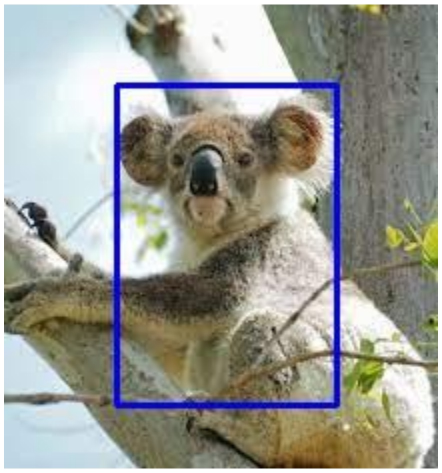


Figure: Detected Object