### **Project Summary**

### This project is an attempt to develop Wild Animal Detection (night vision) using modern computer vision technology.It involved the inclusion of **Yolov5**. The results of the project are expressed in this report.

[**Overview of Object Detection**](https://blog.roboflow.com/the-ultimate-guide-to-object-detection/)

Object detection is a computer vision solution that identifies instances of objects in visual media. Object detection programs draw a bounding box around an instance of a detected object, paired with a label to represent the contents of the box. For example, a person in an image might be labeled "person" and a car might be labeled "vehicle".

**About the YOLOv5 Model**

YOLOv5 is a recent release of the YOLO family of models. YOLO was initially introduced as the first object detection model that combined bounding box prediction and object classification into a single end to end differentiable network. YOLOv5 is the first of the YOLO models to be written in the PyTorch framework and it is much more lightweight and easy to use.

**Flow of project**



**1. Collecting Our Training Images**

Data collection is the process of gathering data relevant to your project's goals and objectives. You eventually obtain a dataset, which is essentially your collection of data, all set to be trained and fed into a model.

**Sources used for Data Collection-**

* Roboflow
* Ena24
* Generated Data (using bing\_image\_downloader)

**2. Data Augmentation**

**Data augmentation** is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset, so that we have more images to learn and train our model.

Here, we have used **Augmentor** for Data Augmentation



**3. Data Cleaning**

Data cleaning is the process of fixing or removing incorrect, irrelevant, incorrectly formatted, or incomplete data within a dataset.

**4. Annotating Our Training Images**

Data Annotation is the human activity of tagging content such as text, photos, and videos so that machine learning models can recognize them and use them to generate predictions.

To train our object detector, we need to supervise its learning with bounding box annotations. We draw a box around each object that we want the detector to see and label each box with the object class that we would like the detector to predict.

There are many labeling tools like, [CVAT](https://blog.roboflow.ai/getting-started-with-cvat/), [LabelImg](https://blog.roboflow.ai/getting-started-with-labelimg-for-labeling-object-detection-data/), Filestage etc . Here in this project we have used, **LabelImg**for annotations.

As you are drawing your bounding boxes, be sure to follow best practices:

* Label all the way around the object in question
* Label included objects entirely
* Avoid too much space around the object.
* Annotated file and training image should be store in same directory

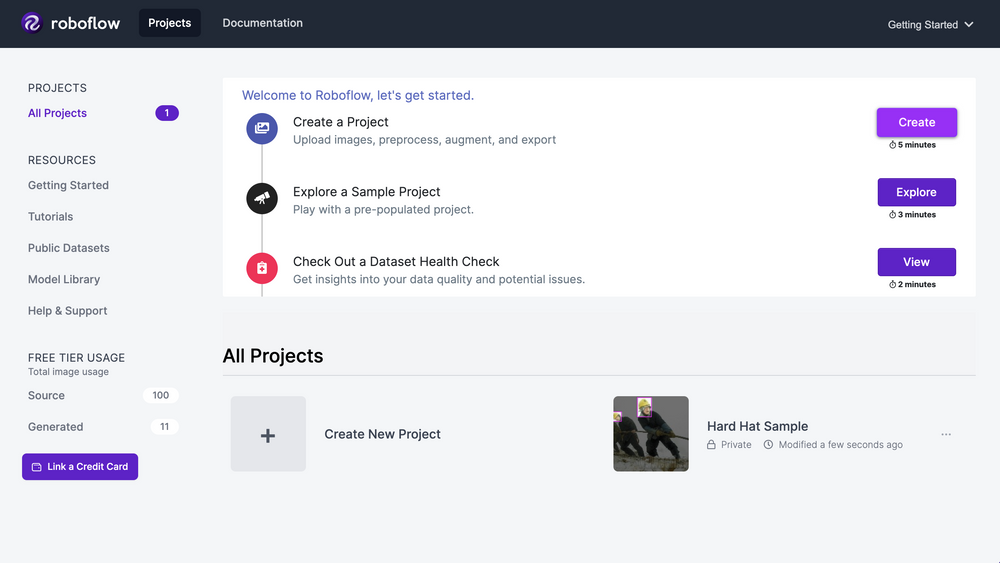
Now we have a folder with our input images and their corresponding annotation.

**5. Create Dataset**

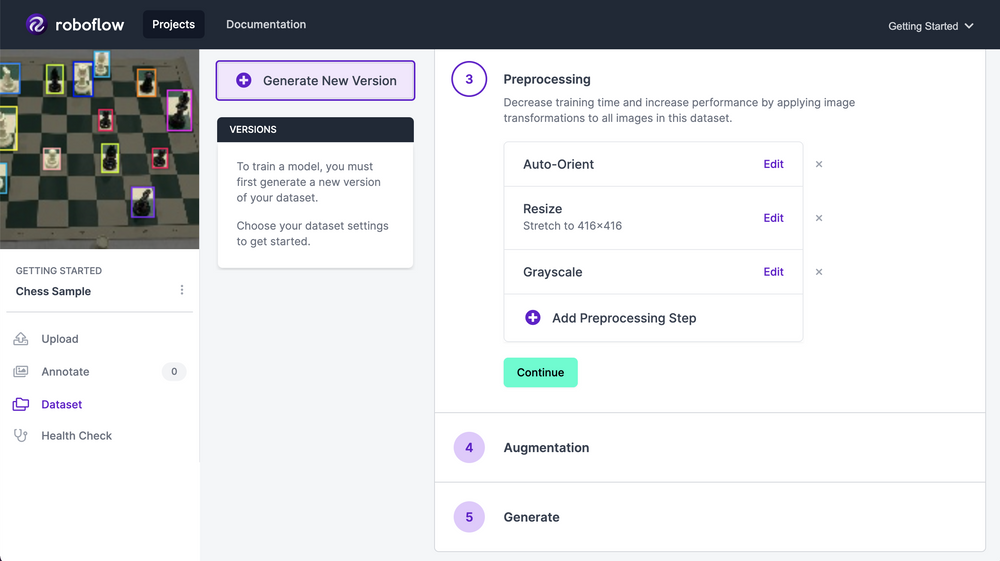
We are creating our dataset using Roboflow. It provides everything you need to turn images into information. We build all the tools necessary to start using computer vision.

**Steps that we performed to create our dataset** : -

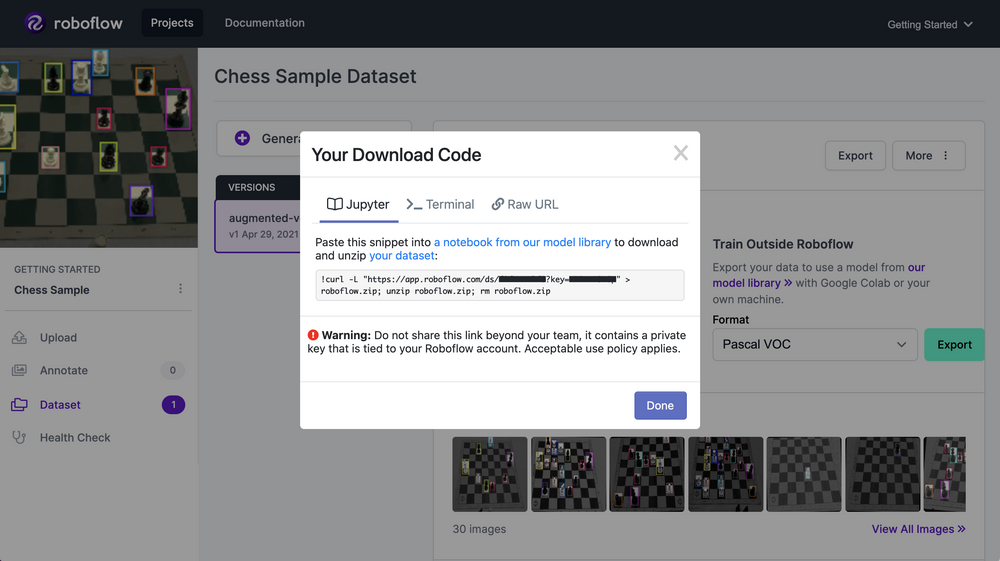
* 1. Create an account on [https://roboflow.com](https://roboflow.com/).
  2. Create a project by specifying the name of your project.



* 1. Under your created project section click “**Create Dataset**.” to continue and upload your dataset folder(contain input images with their annotation) here.
  2. Click “**Finish Uploading**” in the upper right-hand corner! You will be asked to [select a train/test split](https://blog.roboflow.com/train-test-split/) for your images; the default of 70% training, 20% validation, and 10% testing is usually a safe bet.
  3. Continue to [generate your dataset](https://youtu.be/O-ZPxTpb2Yg?t=287&ref=roboflow-blog). This creates a point-in-time snapshot of your images processed in a particular way.

[](https://youtu.be/O-ZPxTpb2Yg?t=287&ref=roboflow-blog)

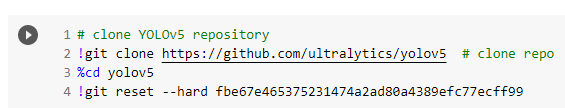
* 1. From here, we can apply any [preprocessing](https://docs.roboflow.com/image-transformations/image-preprocessing?ref=roboflow-blog) and [augmentation](https://docs.roboflow.com/image-transformations/image-augmentation?ref=roboflow-blog) steps that we want to our images. We’ll leave settings as they were: auto-orient and resize 416x416.To create downloaded data, select “**Generate**” at the end of the workflow.
  2. Select Export. Roboflow is now preparing each of your images and annotations for download. Let's use the “Pascal VOC” export format. When you click "Export", your dataset is zipped and you are given several options for using it. When prompted, be sure to select “Show Code Snippet.” This will output a download curl script or jupyter notebook,raw url so you can easily port your data into Colab in the proper format.

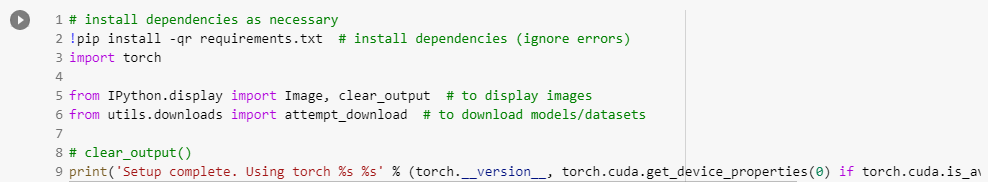


* 1. Now that we have prepared a dataset we are ready to head into the YOLOV5 training code.
  2. Open a notebook on google colab to train YOLOv5.
  3. In Google Colab, you will receive a free GPU. Be sure to File → save a copy in your drive. Then you will be able to edit the code.

**5. YOLO v5 dependencies**

To start off with YOLOv5 we first clone the YOLOv5 repository and install dependencies. This will set up our programming environment to be ready to run object detection training and inference commands.

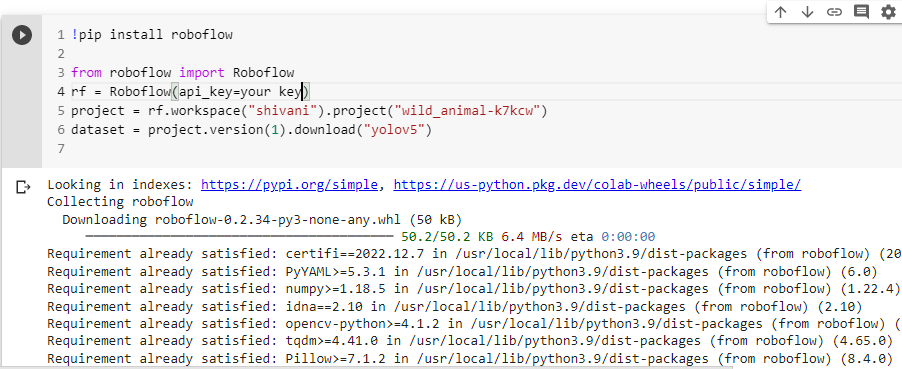




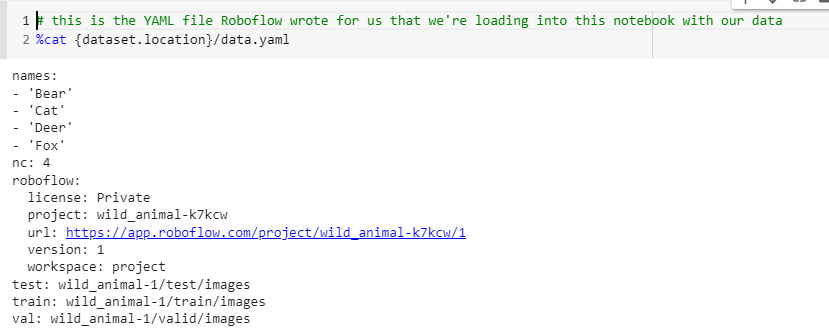
The GPU will allow us to accelerate training time. Colab is also nice in that it comes preinstalled with torch and cuda. If you are attempting this on local, there may be additional steps to take to set up YOLOv5.

**6. Download Custom YOLO v5 Object Detection Data**

We'll copy and paste that jupyter code snippet that will download our dataset from Roboflow. Use the "**YOLOv5 PyTorch**" export format. Note that the Ultralytics implementation calls for a YAML file defining where your training and test data is. The Roboflow export also writes this format for us.



Change classes name into data.yml file as per annotation id of classes here we have 4 classes Bear is on ‘0’ id,Cat is on ‘1’ id ,Deer is on ‘2’ id and Fox has id number ‘3’ . Once you save data.yml run the following command which will show the directory of train,test,validation dataset and names of your classes which you were changed from indexes to names.



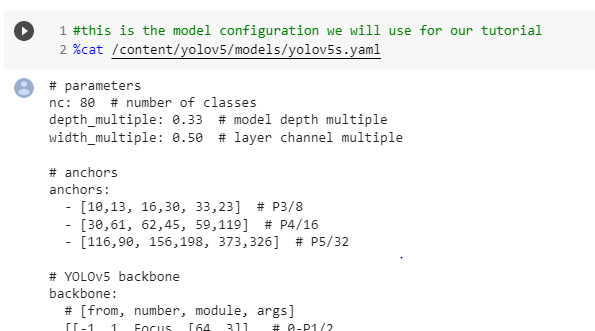
**7. Define YOLO v5 Model Configuration and Architecture**

The configuration file defines the components and policies that your model will use to make predictions based on user input.

Here,we write a model configuration file for our custom object detector. For this project, we chose the smallest, fastest base model of YOLOv5. You have the option to pick from other YOLOv5 models including:

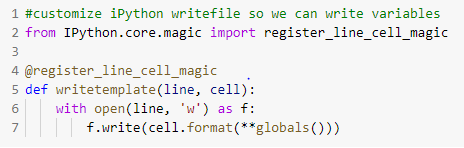
* YOLOv5s
* YOLOv5m
* YOLOv5l
* YOLOv5x

You can also edit the structure of the network in this step, though rarely will you need to do this. Here is the YOLOv5 model configuration file, which we term custom\_yolov5s.yaml

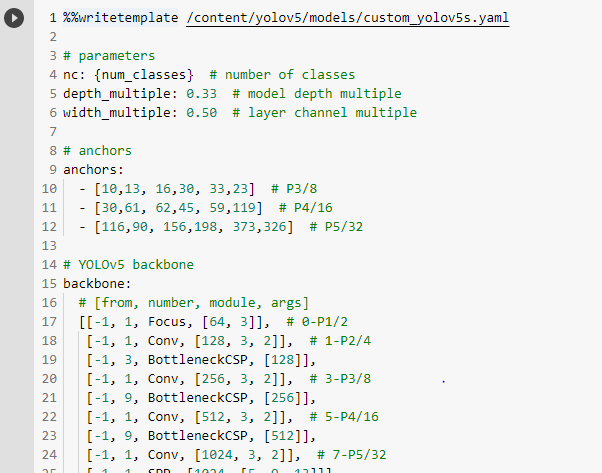


Here ,we just changed the number of classes based on YAML.





Using writetemplate we will create our custom yolov5s.yml with only change in number of classes.

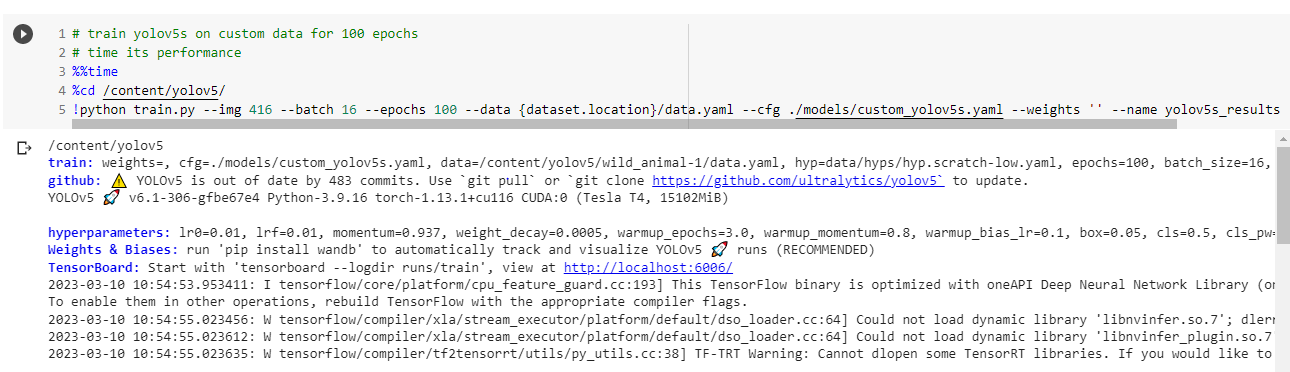


**8. Train a custom YOLO v5 Detector**

With our data.yaml and custom\_yolov5s.yaml files we are ready to train our model.

Here, we are running the training command with the following options:

* **img:** define input image size
* **batch:** determine batch size
* **epochs:** define the number of training epochs. (Note: often, 3000+ are common here!)
* **data:** set the path to our yaml file
* **cfg:** specify our model configuration
* **weights:** specify a custom path to weights. (Note: you can download weights from the Ultralytics Google Drive [folder](https://drive.google.com/open?id=1Drs_Aiu7xx6S-ix95f9kNsA6ueKRpN2J))
* **name:** result names
* **nosave:** only save the final checkpoint
* **cache:** cache images for faster training



**9. Evaluate YOLO v5 performance**

###### 99.5%

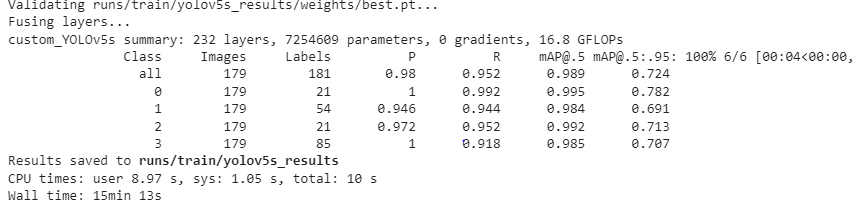
**mAP**

###### 99.3%

**precision**

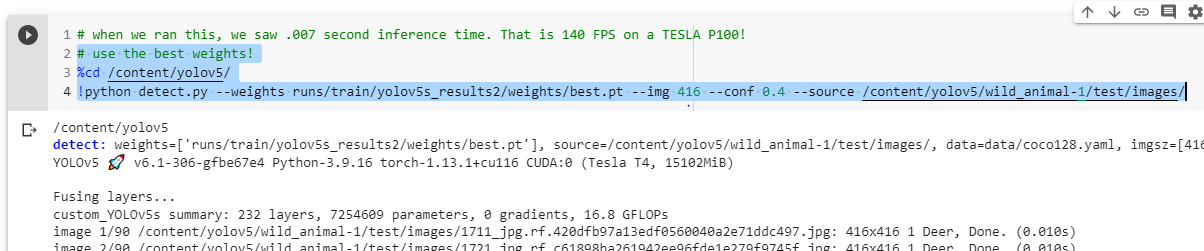
###### 99.3%

**recall**



**10. Run YOLO v5 Inference on test images**

Now we take our trained model and make inference on test images. After training has completed model weights will save in weights/. For inference we invoke those weights along with a conf specifying model confidence (higher confidence required makes less predictions), and an inference source. source can accept a **directory of images, individual images, video files, and also a device's webcam port**.



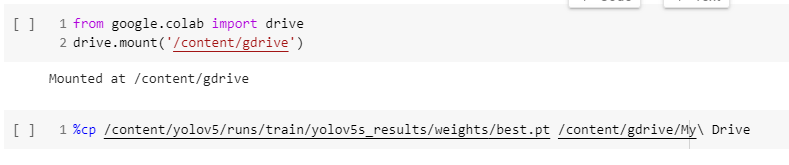
Put source = 1 if you want to open a webcam.

!python detect.py --weights /runs/train/yolov5s\_results2/weights/best.pt --img 416 --conf 0.4 --source 1



**11. Export Saved YOLO v5 Weights for Future Inference**

Now that our custom YOLOv5 object detector has been verified, we might want to take the weights out of Colab for use on a live computer vision task. To do so we import a Google Drive module and send them out.



**Results of model -**







