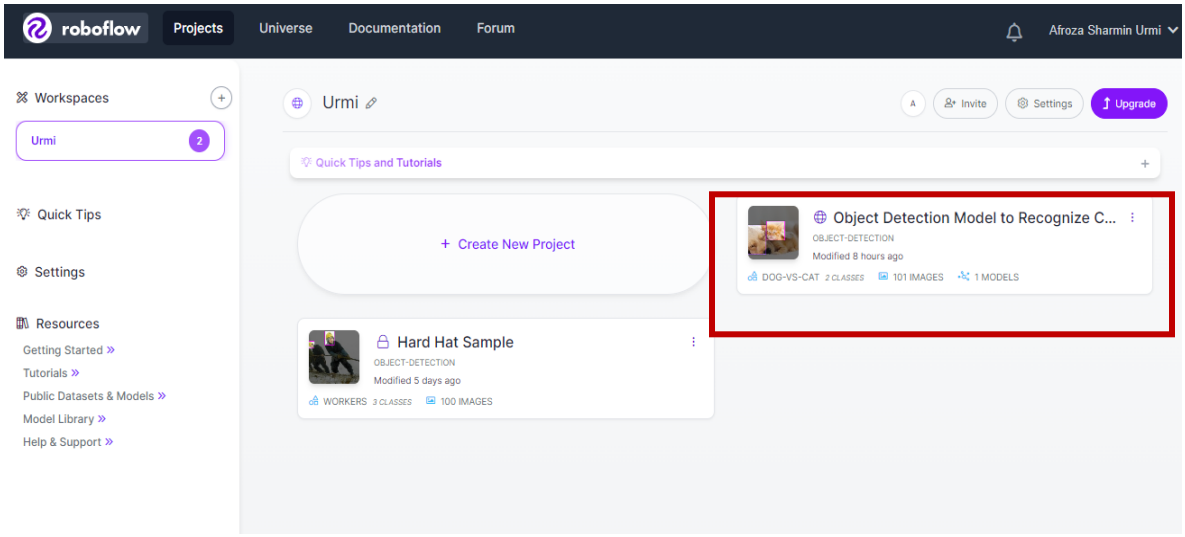
Level-1 ML Engineer Recruitment Test-2023

"A brief description and observations of the given model implementation"

1. Data Annotation and Pre-processing

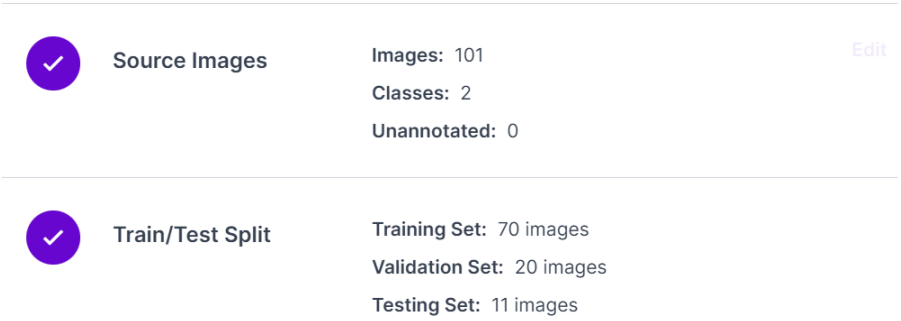
1.1 The given dataset consists of images of cats and dogs, and for computer vision projects, we first need to annotate our data. As mentioned in the task of Annotating images using bounding boxes, we annotate our dataset in "Roboflow" using bounding boxes. The dataset highlighted in red below is annotated for further implementation.



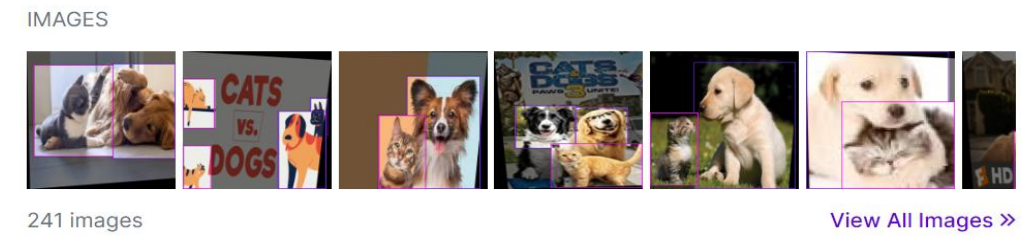
1.2 A total of 102 images were provided for the task, excluding one image that appeared to be of low-quality annotation and would ultimately account for the poor performance. As we can see, there are 101 annotated images with two classes (cat and dog). We split our dataset as below:

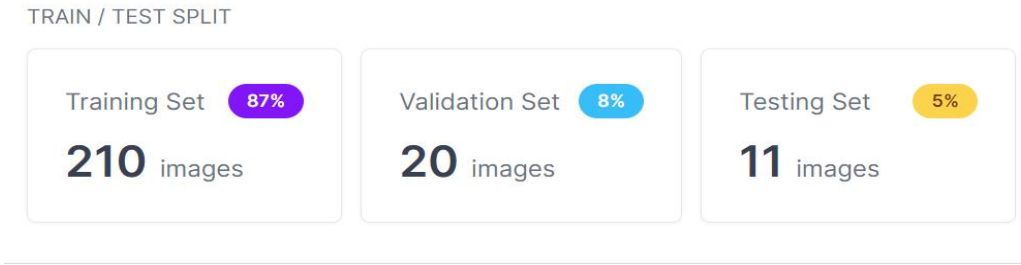
Annotated dataset in Roboflow: https://app.roboflow.com/urmi/object-detection-model-to-recognize-cat-and-dog/generate/preprocessing

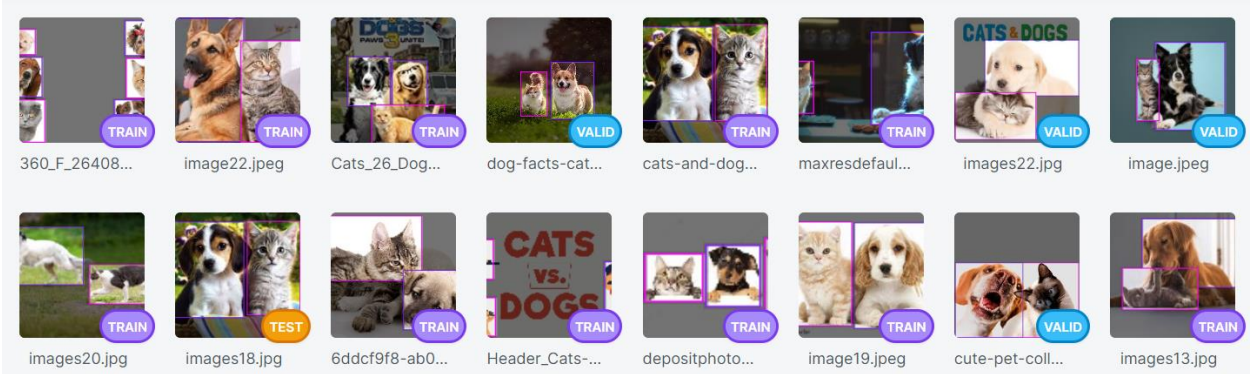
1.3 Some samples of our dataset split in training, testing and validation set below.



1.4 **Pre-processing and Augmentation:** We did some preprocessing like auto-orientation, re-sizing, auto-adjust contrast, etc for enhancing our dataset. Also augmented our dataset using crop, shear, etc in order to get precise performance from our dataset. Below showing a little peak view after preprocessing and augmenting our dataset.







2. Object Detection Model using Yolov5

Step: 01 (Installing and Importing Libraries and Dataset)

2.1 After annotating the dataset we implemented our model in Google Colab. Here we used Yolov5 for our object Detection Model. As we see below, we clone yolov5 and also install, and import essential prerequisites to build our model.

```
!git clone https://github.com/ultralytics/yolov5
%cd yolov5
%pip install -qr requirements.txt
%pip install -q roboflow

import torch
import os
from  IPython.display import Image, clear_output
print(f"Setup complete. Using torch{torch.__version__}({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'})")
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 15615, done.
remote: Counting objects: 100% (222/222), done.
remote: Compressing objects: 100% (165/165), done.
remote: Total 15615 (delta 106), reused 126 (delta 57), pack-reused 15393
Receiving objects: 100% (15615/15615), 14.65 MiB | 27.32 MiB/s, done.
Resolving deltas: 100% (10634/10634), done.
/content/yolov5
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 184.3/184.3 kB 9.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 6.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.2/56.2 kB 4.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.5/54.5 kB 6.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 7.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.8/67.8 kB 7.3 MB/s eta 0:00:00
  Building wheel for wget (setup.py) ... done
Setup complete. Using torch2.0.0+cu118(CPU)
```

2.2 We already installed Roboflow and below we're importing Roboflow in order to generate the API key of our dataset in Roboflow.

```
from roboflow import Roboflow
rf = Roboflow(api_key="YOUR API KEY HERE")
```

```
upload and label your dataset, and get an API KEY here: https://app.roboflow.com/?model=undefined&ref=undefined
```

2.3 Using the API key from above we get a path like below from "Roboflow", which we define in the cell for further implementation.

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("").project("object-detection-model-to-recognize-cat-and-dog")
dataset = project.version(8).download("yolov5")
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="DPoFLaGRx96aoZgimwaX")
project = rf.workspace("urmi").project("object-detection-model-to-recognize-cat-and-dog")
dataset = project.version(2).download("yolov5")
```

```
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2 to yolov5pytorch: 100% [10831343 / 10831343] bytes
Extracting Dataset Version Zip to /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2 in yolov5pytorch:: 100%|██████████| 494/494 [00:00<00:00, 835.06it
```

Step: 02 (Training Dataset using Yolov5)

2.4 Our image size is 640x640, taking image as 640, total batches are 16 and we ran 100 epochs in order to train our dataset.

```
!python train.py --img 640 --batch 16 --epochs 100 --data {dataset.location}/data.yaml --weights yolov5s.pt --cache --name yolov5s_results
```

```
train: weights=yolov5s.pt, cfg=, data=/content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/data.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=100,
github: up to date with https://github.com/ultralytics/yolov5 ✅
requirements: /content/requirements.txt not found, check failed.
YOLOv5 🚀 v7.0-158-g8211a03 Python-3.10.11 torch-2.0.0+cu118 CPU

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0,
ClearML: run 'pip install clearml' to automatically track, visualize and remotely train YOLOv5 🚀 in ClearML
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5 🚀 runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 75.6MB/s]
Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt to yolov5s.pt...
100% 14.1M/14.1M [00:00<00:00, 297MB/s]

Overriding model.yaml nc=80 with nc=2
```

Locating the dataset, our dataset location is saved in (dataset.location). Specify a path to weights using transfer learning, we choose generic pretrained checkpoint.

```
Transferred 343/349 items from yolov5s.pt
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 60 weight(decay=0.0005), 60 bias
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01), CLAHE(p=0.01, clip_limit=(1, 4.0), tile_grid_size=(8, 8))
train: Scanning /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/train/labels... 210 images, 0 backgrounds, 0 corrupt: 100% 210/210 [00:00<00:00,
train: New cache created: /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/train/labels.cache
train: Caching images (0.2GB ram): 100% 210/210 [00:01<00:00, 173.04it/s]
val: Scanning /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/valid/labels... 20 images, 1 backgrounds, 0 corrupt: 100% 20/20 [00:00<00:00, 171.9
val: New cache created: /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/valid/labels.cache
val: Caching images (0.0GB ram): 100% 20/20 [00:00<00:00, 96.55it/s]

AutoAnchor: 2.58 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset ✅
Plotting labels to runs/train/yolov5s_results/labels.jpg...
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/train/yolov5s_results
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       0/99         0G     0.1078    0.03999    0.02538         16        640: 100% 14/14 [05:18<00:00, 22.78s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95:   0% 0/1 [00:00<?, ?it/s]WARNING ⚠ NMS time limit 1.500s exceeded
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:09<00:00,  9.93s/it]
                   all         20         37     0.0051      0.162    0.00314    0.00103

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       1/99         0G    0.08741    0.04138    0.02574          9        640: 100% 14/14 [04:57<00:00, 21.25s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95:   0% 0/1 [00:00<?, ?it/s]WARNING ⚠ NMS time limit 1.500s exceeded
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:12<00:00, 12.60s/it]
                   all         20         37    0.00476       0.27    0.00705    0.00174
```

.

.

```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      97/99         0G    0.02085    0.02359   0.003545          8        640: 100% 14/14 [04:57<00:00, 21.23s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:09<00:00,  9.23s/it]
                   all         20         37      0.766      0.674       0.75      0.403

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      98/99         0G    0.02514    0.02446   0.004931         12        640: 100% 14/14 [04:57<00:00, 21.26s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:08<00:00,  8.42s/it]
                   all         20         37      0.673      0.728      0.769      0.406

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      99/99         0G    0.02234    0.02152    0.00396          7        640: 100% 14/14 [04:58<00:00, 21.32s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:09<00:00,  9.58s/it]
                   all         20         37      0.823      0.648      0.765      0.401
```

.

.

```
Validating runs/train/yolov5s_results/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:08<00:00,  8.88s/it]
                   all         20         37       0.87      0.835      0.851      0.452
                   cat         20         18      0.913      0.722      0.848      0.433
                   dog         20         19      0.827      0.947      0.855      0.471
Results saved to runs/train/yolov5s_results
```

So, we ran our model using 100 epochs and finally got the score or model summary: where we can see all the layers, parameters, performance measurement scores, and model object, class, and box losses.

Red box is the path where our model results are saved, if we want to use it later must make sure we are providing the path as it is!

Step: 03 (Evaluate YOLOv5 Detector Performance)

2.5 Training loss and performance metrics are saved to Tensorboard and to a logfile. As shown below dashboard (a complete overview of our trained models). Tensorboard showing our model's performance measurement score graphs (Precision, recall scores in increasing manner). Also showing losses like class, object, box losses (are in decreasing manner). More **analysis and visualization** can be found in the Tensorboard in the .ipynb file.

Step: 04 (Run Inference with Trained Weights)

2.6 Running inference with a pretrained checkpoints on context of test/images folder downloaded from Roboflow so that we can move on test model. Inference results are saved in below red marked box, this path is going to use during testing our model.

```
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 640 --conf 0.1 --source {dataset.location}/test/images

detect: weights=['runs/train/yolov5s_results/weights/best.pt'], source=/content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images, data=da
requirements: /content/requirements.txt not found, check failed.
YOLOv5 🚀 v7.0-158-g8211a03 Python-3.10.11 torch-2.0.0+cu118 CPU

Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
image 1/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/497333_jpg.rf.4a5cfdd604455500b16156fb9e4126a4.jpg: 640x640 1 cat, 1
image 2/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/black-white-mixedbreed-cat-happy-border-collie-dog-panting-blue-back
image 3/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/cat-and-dog_jpg.rf.b8f08e949970b659464b3a2e7b8e3cdb.jpg: 640x640 1 c
image 4/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/cat_runs_away_from-dog-768x576_jpg.rf.03f1d378e3c9fedc81daf5d165763
image 5/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/dog-and-cat-cover_jpg.rf.978f6e15d5b9b43fae7d48f7b12b6df2.jpg: 640x6
image 6/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/image23_jpeg.rf.d03ead7de6759c594e5bbc717aea37c5.jpg: 640x640 1 cat
image 7/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/image6_jpeg.rf.7f0d8da183ffece8f0ad89a2de0b9e6a.jpg: 640x640 1 dog,
image 8/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/images16_jpg.rf.cf988ff99f6c8a7138da2128b044ff1b.jpg: 640x640 2 cats
image 9/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/images18_jpg.rf.1acd09dbba4c07fab1518877a7cfb1b1.jpg: 640x640 1 cat
image 10/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/images27_jpg.rf.619ff47d3a4ebdcfb840ef336b8e0fe6.jpg: 640x640 2 cat
image 11/11 /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/test/images/n-cats-a-20181226_jpg.rf.b51dc2c3f6e9d54f13366a82e9c98f35.jpg: 640x
Speed: 2.0ms pre-process, 511.4ms inference, 1.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/exp4
```

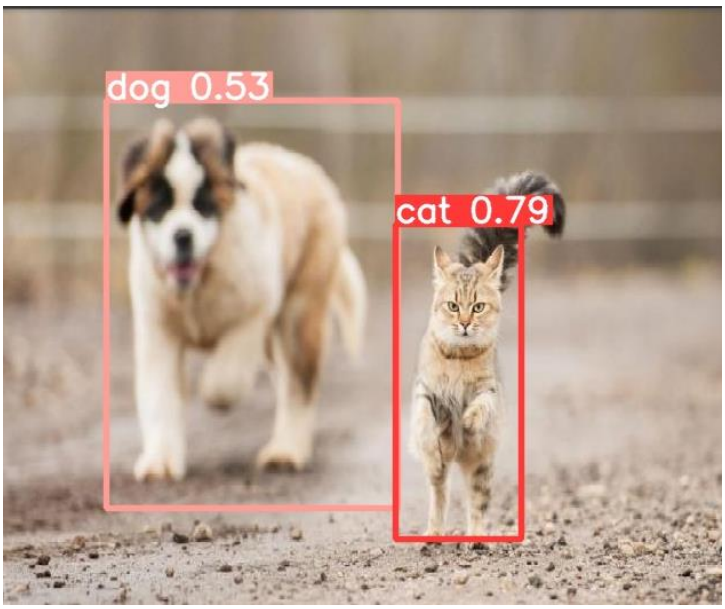
Step: 05 (Test Model Using Test Dataset)

2.7 Before moving into the testing phase, we must be careful about the path above. Importing globe to define the path and uploading the test images. Now we'll use above-saved result in the path to test our model using test dataset as below:

```
import glob
from IPython.display import Image, display

for imageName in glob.glob('runs/detect/exp4/*.jpg'):
    display(Image(filename = imageName))
    print("\n")
```

2.8 The test image below shows that the confidence score for detecting a dog is .53 and for a cat is 0.79 in a bounding box which is established as a pretty good model. Since the image of the dog is quite blurry, it shows a poorer score than the cat.



Step: 06 (Model Validation)

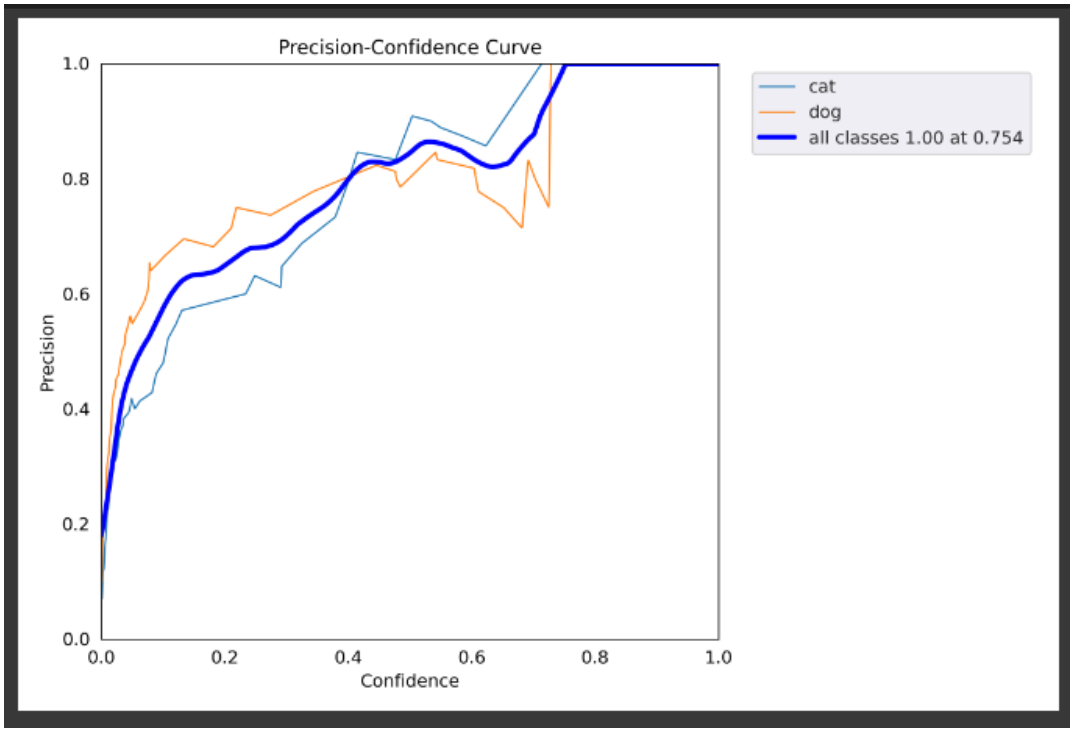
2.9 Model Validation is shown below, where including the Class of an image, and total instances. Performance measurement scores and Our model Performance measures are getting from precision, and recall scores. Also, Model class, object, and bounding box loss analysis (which we can also find in the Tensorboard dashboard above). Model validation results are saved in the below marked path. Our model provides good performance measurement scores as shown in the below screenshot.

```
!python val.py --weights runs/train/yolov5s_results/weights/best.pt --data /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/data.yaml  --img 640 --i

val: data=/content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/data.yaml, weights=['runs/train/yolov5s_results/weights/best.pt'], batch_size=32, imgsz=64
requirements: /content/requirements.txt not found, check failed.
YOLOv5 🚀 v7.0-162-gc3e4e94 Python-3.10.11 torch-2.0.0+cu118 CPU

Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning /content/datasets/Object-Detection-Model-to-Recognize-Cat-and-Dog--2/valid/labels.cache... 20 images, 1 backgrounds, 0 corrupt: 100% 20/20 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:13<00:00, 13.24s/it]
                   all         20         37      0.876      0.835      0.849      0.456
                   cat         20         18       0.92      0.722      0.841      0.439
                   dog         20         19      0.832      0.947      0.858      0.474
Speed: 10.4ms pre-process, 620.9ms inference, 0.8ms NMS per image at shape (32, 3, 640, 640)
Results saved to runs/val/exp
```

2.10 Traditional precision-confidence curve showing below to give an overview of the result of detecting cat and dog.



Step: 07 (Test model performance using random image from internet)

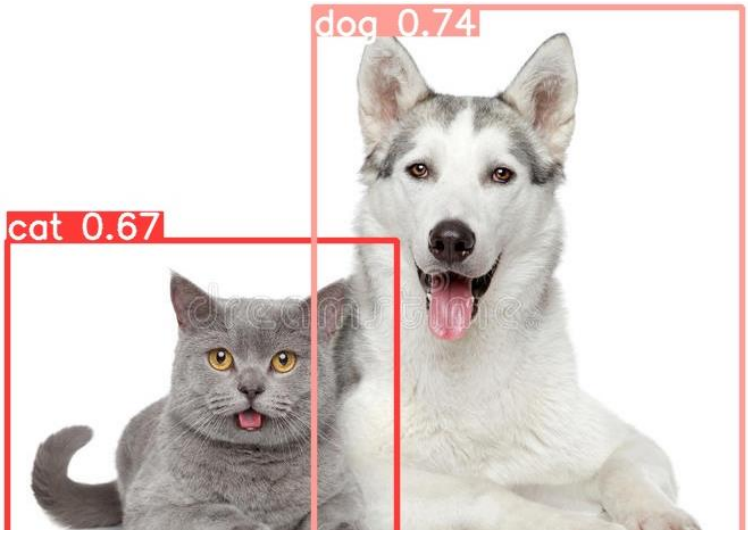
2.11 We can upload random images and copy path in the source to test our model performance randomly.

```
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 640 --conf 0.1 --source /content/cat-dog-together-white-background-28862899.jpg
```

Before running images, we have to make sure about giving the path where results are saved.

```
for imageName in glob.glob('runs/detect/exp3/*.jpg'):
    display(Image(filename = imageName))
    print("\n")
```

As below, we can see our model did pretty well on detecting cat and dog on random image (performance score can vary based on environment and set up). As we can see, confidence score is 0.74 for dog and 0.67 for cat.



Step: 08 (Save the Model)

2.12 We are saving our model so that we can build API in streamlit using .pt file for common usages of our model. Convenient User Interface to understand our work and implemented model.

In the next step, we are saving and downloading .pt file and yolov5 model. We're going to use .pt file for making API using streamlit in the VSCode.

3. Python API framework to Recognize Cat and Dog

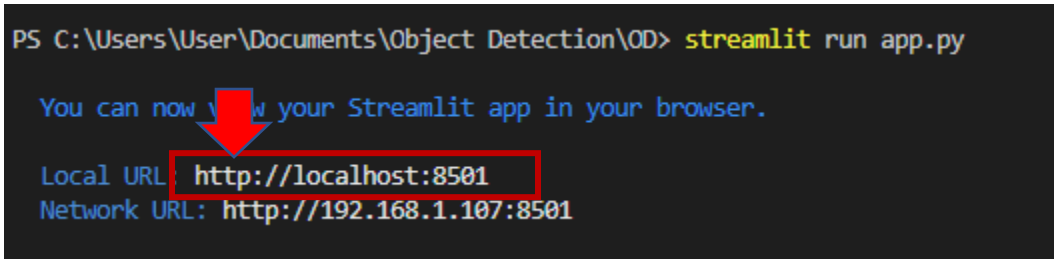
3.1 For API we made a folder called **ObjectDetection** and where we saved .pt file which was downloaded from our previously saved model. Now in **VSCode** we installed **streamlit** and other required python packages to build our API and import those libraries in the **app.py** file as below:

```
import streamlit as st
import torch
import numpy as np
from PIL import Image
```

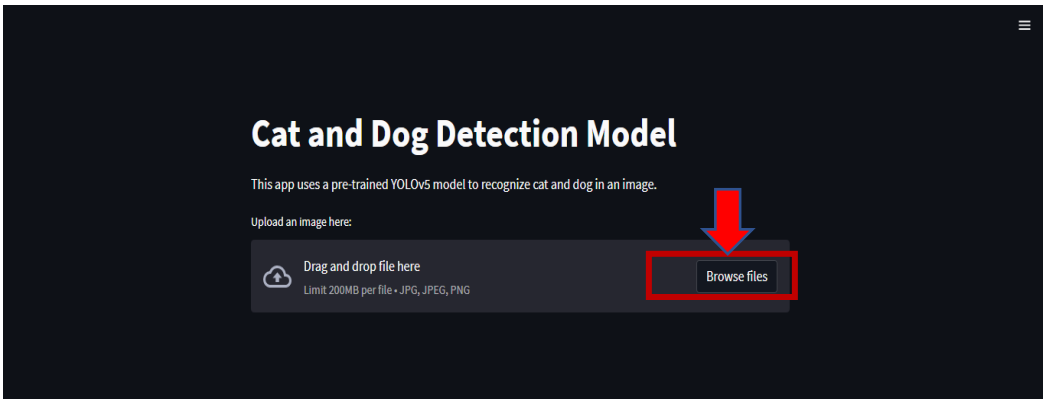
We define .pt file path in the model for recall model implementation. The .pt file is saved in the same folder where **app.py** file was created.

```
# Load YOLOv5 and our pre-traied model
model = torch.hub.load('ultralytics/yolov5', 'custom', path='C:\\Users\\User\\Documents\\Object Detection\\ObjectDetection\\best.pt')
```

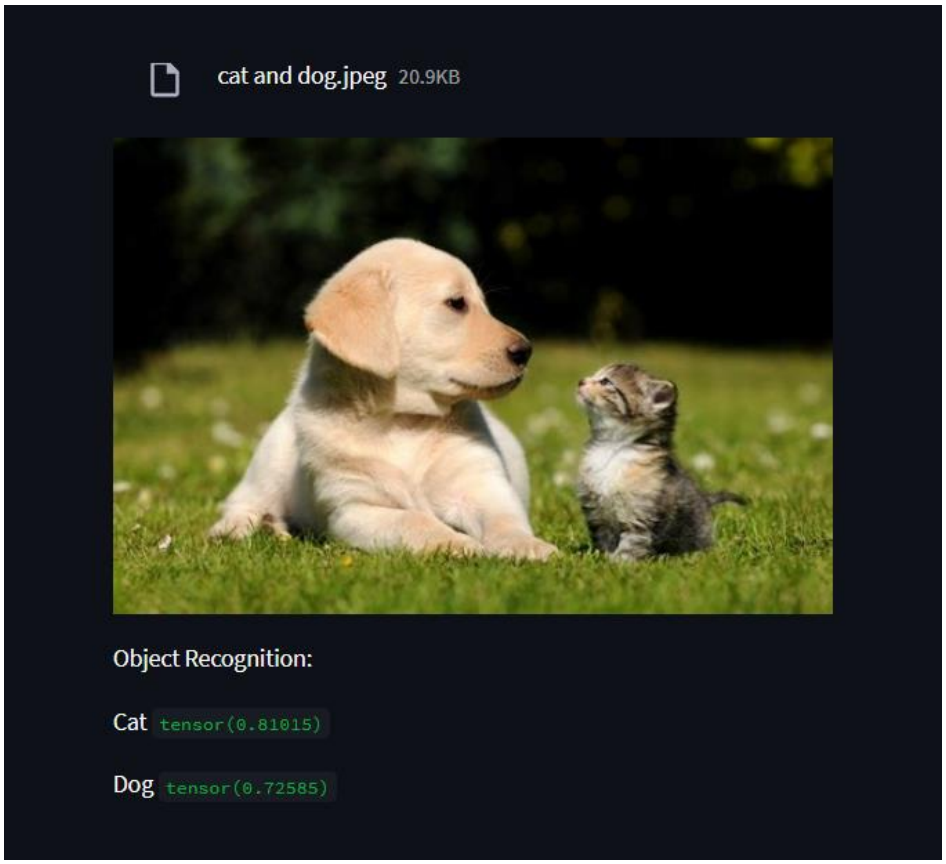
3.2 After defining the function, streamlit app, creating uploader using st.file uploader and initializing type, loading images and performing object detection, displaying a list of images using bounding box and confidence score in the app.py file. Run the command streamlit run app.py, which loaded us in:



3.3 Above "local URL" going to shift us to the browser showing "Cat and Dog Detection Model" and ask to browse images. And then detect the image and confidence score as below:



We need to click on browse files icon and provide random cat and dog image.



The above prediction shows the confidence score of Cat is 0.73 and Dog is 0.81. So, we established that our model has a good performance in object detection.

Notes:

1. In the Dataset many of the images are poor quality, we tried to preprocess and augment the image as much as possible. This may result in poor performance.

2. Roboflow annotated dataset link is defined in the cell.

3. Now if we want to test a random image, we must upload and provide a path to the specific test image.

4. Annotated Dataset is exported from Roboflow.

5. I'm sharing files in both formats (google drive share link and GitHub repository link), in case one might get corrupted or show an error.