The TensorFlow Object Detection API has been used for this project.

Steps to install:

1. Clone Models repository of tensorflow from github **https://github.com/tensorflow/models**
2. Install all the dependencies in your virtual environment using requirements.txt.
3. Install protobuf to extract .proto files to python files for object detection API to work.
   I used Protoc 3.11.4-win64.zip
   Protobuf: https://github.com/protocolbuffers/protobuf/releases/download/v3.11.4/protoc-3.11.4-win64.zip
4. Download and extract it. Save the location of protoc.exe . Run protobuf.py to extract the proto files present in the object detection folder. Save the script in the research folder in models and write the following command in the console.

   ```
   python protobuf.py object_detection/protos C:/Users/Ankita/Desktop/models-master/protobuf/bin/protoc
   ```

5. Add the research and research/slim folders to our PYTHONPATH environment variables .

Run setup.py in the model/research folder.

```
python setup.py build
```

```
python setup.py install
```

Steps to create datasets

1. I transformed the images to suitable resolution for training using script transform_image_resolution.py to speed up learning.
2. Split the transformed images in two folders ‘train’ and ‘test’ in 70:30 ratio. Put these folders in a new folder images in model/research/object_detection
3. For labeling dataset, I used LabelImg tool and saved images in PascalVOC format. It saves a xml file for each image containing label data, ie bounding box info.
4. To create input dataset for training, we need to create TFRecords. Put xml_to_csv.py and generate_tfrecords.py in object detection folder. Use xml_to_csv.py to convert xml files to csv and create two files ‘test_labels.csv’ and ‘train_labels.csv’ in images directory.
   To create train.record and test.record, run the generate_tfrecords.py as

   ```
   python generate_tfrecord.py --csv_input=images/train_labels.csv --image_dir=images/train --output_path=train.record
   ```

```
python generate_tfrecord.py --csv_input=images/test_labels.csv --image_dir=images/test --output_path=test.record
```

Steps before training

1. Create a training folder in object_detection folder
2. Create a 'label map' and save as .pbtxt to map the ids to the names.
3. I used faster_rcnn_inception model. Download link of 'faster_rcnn_inception_v2_coco' model. Save and extract it and get the location of model.ckpt file.
   https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md
   The configuration file 'faster_rcnn_inception_v2_pets.config' for this model, is in the object_detection/samples/configs folder . Copy it to object_detection/training folder .
   Change Fine tune checkpoint in the config file to the model.ckpt file location.

Training the model

1. Put the train.py (which is present in the object_detection/legacy folder) in object_detection folder and run to train the model.

   ```
   python train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/faster_rcnn_inception_v2_pets.config
   ```

2. Run the export_inference_graph.py in object_detection folder to get the frozen_inference_graph.pb and the associated model checkpoint in the inference_graph folder

   ```
   Python export_inference_graph.py --input_type image_tensor --pipeline_config_path training/faster_rcnn_inception_v2_pets.config --trained_checkpoint_prefix training/model.ckpt-(largest number) --output_directory inference_graph.
   ```

3. Make changes to the visualization_utils.py in object_detection/utils folder by adding a function to return bounding box xmin, ymin, xmax and ymax. Crop the image accordingly and extract out the bounding box by
   running the code object_detection_image.ipynb in object_detection folder to use the frozen_inference_graph and labelmap and run on test image.