Custom Object Detector using Tensorflow for ZED Mini

Kamali Balusamy
Robotics Engineering Master Course
University of Genova
Email: s4849474@studenti.unige.it

Sathish Kumar Subramani
Robotics Engineering Master Course
University of Genova
Email: s4847560@studenti.unige.it

***Abstract*—- In recent years, deep learning has been used in image classification, object tracking, action recognition and scene labeling. A noticeable growth is observed in the field of computer vision research. In computer vision, object detection is the task of classifying and localizing the objects to detect the same. Deep Neural Networks begin to be used in the last ten years in computer vision, and now they are the state-of-the-art for object detection. In this project we propose and trained a MobileNet neural network which is a lightweight DNNs performs the detection of various objects like (mouse, monitor, keyboard and CPU) in real-time by building the custom object detector and converting it into onnx format to be reusable in any application with using ZED mini performs with good accuracy.**

I. Introduction

The applications and widespread use of machine learning algorithms have made a serious change in the way we perceive computer vision problems. With the introduction of deep learning into the field of image classification, the real-time object detection have faced a huge impact. In recent times, object detection and pose estimation have gained notable attention in the context of robotic vision applications.

Object detection is a computer vision technique in which a software system can detect, locate, and trace the object [1]. The special attribute about object detection is that it identifies the class of object (person, cup, table, chair, etc.) and their location-specific coordinates in real time. The location is identified by drawing a bounding box around the object. The ability to locate the object inside an image defines the performance of the algorithm used for detection [2].

II. Hardware and software

A. Hardware

1) ZED Mini and Laptop: The ZED Mini is a stereo camera that provides high definition images and accurate measure of the environment depth.The baseline of the camera is 63mm, which is similar to the human average eye distance. It has been designed for the most challenging applications, including autonomous vehicle control, mobile mapping, aerial mapping, security, and surveillance.

The hardware which we used for the process is ASUS Laptop has a i5 processor with NVDIA graphic card and Zed mini camera which is a stereo camera which has a maximum resolution of 2560x720 pixels and a FoV of 90(HxV).The ZED SDK provides a depth map obtained through stereo triangulation. Its depth range is from 0.1 mt to 20 mt.

B. Software

1) Tensorflow: TensorFlow is an end-to-end open source platform for machine learning. It was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization to conduct machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well. For custom Object Detection, using Tensorflow is a computer vision technique. As the name suggests, it helps us in detecting, locating, and tracing an object from an image or a video.

2) Anaconda: Anaconda is a Python distribution that makes it easy to install Python and a number of its most often used libraries in a flexible way on a Windows or Linux machine [5].

3) Jupyter Notebook: The Jupyter Notebook is an open-source web application that allows to create and share documents that integrate live code, equations, computational output, visualizations, and other multimedia resources, along with explanatory text in a single document.

We have installed Tensorflow api, git cloned package called labellmg and dependencies.

4) CUDA and cuDNN: CUDA is a parallel computing platform and programming model developed by Nvidia for general computing on its own GPUs (graphics processing units). Tensorflow with GPU support requires a cuDNN version of at least 7.2 [6].

5) GitBash: Git Bash is an application for Microsoft Windows environments which provides an emulation layer for a Git command line experience. We used it for Windows provides a BASH emulation used to run Git from the command line.

6) Protocol Buffer: Protocol Buffers is a free and open-source cross-platform data format used to serialize structured data.

7) MobileNet: MobileNet is a type of convolutional neural network. They are based on a streamlined architecture that uses depth wise separable convolutions to build lightweight deep neural networks that can have low latency. Mobilenet SSD is an object detection model that computes the bounding

box and category of an object from an input image. This Single Shot Detector (SSD) object detection model uses Mobilenet as backbone and can achieve fast object detection .In SSD, after extracting the features using an arbitrary backbone, the bounding boxes are calculated at each resolution while reducing the resolution with Extra Feature Layers [7]. MobilenetSSD will concatenate the output of the six levels of resolution and calculate a total of 3000 bounding boxes, and finally, filter out bounding boxes using non-maximum suppression (nms).

C. Methodology

In order to train custom Object Detector with the ZED Mini we evaluated some methodology as follows,

Object detection, regardless of whether performed via deep learning or other computer vision techniques, builds on image classification, and seeks to localize exactly where in the image each object appears.

While we were performing object detection, given an input image, we wish to obtain:

- A list of bounding boxes, or the (x, y)-coordinates for each object in an image
- The class label associated with each bounding box
- The probability/confidence score associated with each bounding box and class label.

The neural network is a very simple feedforward network which takes the flattened image as input, and predicts the parameters of the bounding box.

The First option is to choose RCNN. This network transforms the object detection problem into the classification problem and greatly improves the accuracy but It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image. It cannot be implemented in real-time as it takes around 47 seconds for each test image [8]. However, when analyzing this option, we found that it is also not feasible for us because we were working to detect the object in real-time.

The second option was to choose Fast RCNN for performing the object detection Fast R-CNN drastically improves the training and detection time from R-CNN. It also improves Mean Average Precision (mAP) marginally as compared to R-CNN. Most of the time taken by Fast R-CNN during detection is a selective search region proposal generation algorithm resulting in a huge amount of computational time [9]. However, this option is likely unfeasible.

The third option was to use the Faster RCNN. Even though this network is faster than fast RCNN and no longer depends on the region selection algorithm, the training process is complex and slow [10]. Hence this neural network is not viable for our task.

The other option is Mobile net which is a type of convolutional neural network and popular algorithm in object detection and it's generally faster than Faster RCNN. Since it is lighter package and low in latency this option seemed the most appropriate and feasible to follow for the object detection [11].

our goal was to build the neural network and convert it into ONNX format which can be integrated with the ZED mini. For doing this we preferred to use TensorFlow than pytorch because TensorFlow has several options from which we can able to choose. It is very powerful and mature deep learning library with strong visualization capabilities and several options to use for high-level model development.

1) Why Python?: Python offers concise and readable code. While complex algorithms and versatile workflows stand behind machine learning. Python's simplicity allows to write reliable systems. So Python for our project is a great choice, as this language is very flexible for us to code.

III. Implementation

Our Implementation has been split into various phases.

SECTION 1: After the installation process of the software we have created the virtual environment and

SECTION 2: Collected the various images and labeled them. During labeling, we process our data and add labels to help our data to learn. We have created an imaginary box around an object That box is a reference point to the object and can be used in outlining the object's X and Y coordinate. after this process we have

SECTION 3: Trained TensorFlow Object Detection Models using TensorFlow api

SECTION 4: Detected various objects from images and the webcam.

SECTION 5: Freezing TFOD and converting it into ONNX format

A. Installation and setup

Starting from the installation, we looked for creating the new virtual environment and activated it with the pip dependencies, CUDA, and cuDNN which gives the GPU-based acceleration when training the TensorFlow object detection model[12]. and we have installed the ipi kernel which associates the virtual environment with the Jupyter notebook. The version we used for the TensorFlow is tensorflow gpu-2.7.0, CUDA 11.2, and cuDNN 8.1 which is a specific combination for the TensorFlow [13].

B. Collect and Label Images

At first, we have collected the datasets (Downstairs and upstairs steps) which consist of 1000 images from one of the git repositories. But due to some improper image clarity and unlabeled images we faced some difficulties during training the images. Hence we have decided to make the datasets consist of four labels(mouse, monitor, keyboard, and CPU), 25 images of each label. which is 100 images and we augmented as 600 in total under different angles and different lighting conditions on our own so that our model learns to generalize better and perform not only on our training data but also in the real world thus the model is considered to be as generalized. The ratio between the training and the test is 8:2. we had downloaded the various images from google and started labeling it by navigating our image folder and creating a bounding box

around all of the collected images and saving it with the annotations which is an xml file for each image with the git package tzutalian/labellmg [14]. Image annotation is used to label the features we need our system to recognize.

C. Training TensorFlow detection model

We used a model named mobilenet which is a specific deep learning architecture. when it comes to training a model using TensorFlow object detection, we have installed all the various dependencies, folder structures and started training the model. Once the setup is done for training the models the Tensorflow Model Zoo consists of the whole heap of different types of models which we can train and normally there is a tradeoff, whereas faster detections will be lower in accuracy and higher accuracy will be in slower detections. Tensorflow Model Zoo is a set of links on a Github page set up by the Object Detection team. The models in the Tensorflow Model Zoo represent different neural network architectures that are used for object detection [15]. The model which we used is SSD MobileNet V2 FPNLite 320×320 because it has average precision and it has a speed of 22 milliseconds which is average accuracy, low latency and fast.

The next process is to head towards the training and detection notebook and the first step we did is to import the operating system in the jupyter notebook and to set up the pre-trained model variables. In this case, we're defined what we want our custom model to be which is the SSD MobileNet, we have set up the name of our pre-trained model which is the name of the file that we download from our TensorFlow model zoo then we defined the pre-trained URL, TF RECORD script name and label map name.

Following this, the further process we did is to create the different paths which make the whole heap easier to work with all of the stuff that we've got available within our working directory so we're able to refer to each one of these reasonably easily whereas the various paths are WORKSPACE PATH, SCRIPTS PATH, APIMODEL PATH, ANNOTATION PATH, IMAGE PATH, MODEL PATH, PRE-TRAINED MODEL PATH, CHECKPOINT PATH, OUTPUT PATH, PROTOC PATH where each of one these represents a different file path that we've got available. When we were stepping into the TensorFlow folder we were having a Label image folder, models, protoc, scripts, and the workspace. The next step is to import wget to pull down our model from the tensorflow model zoo.

The next shell is to detect whether or not we're using Linux or whether or not we're installing windows and it does the two things whereas first it installed protoc which is one of the object detection stuff uses something called protocol buffers and we set this up for our TensorFlow object detection API and we verified our installation in the next shell which is the VERIFICATION SCRIPT which tells whether or not we've got the TensorFlow object detection API installed successfully. Then we have checked the pip list and check whether we have actually installed TensorFlow or not and we run the VERIFICATION SCRIPT again and we got okay which means we have successfully installed the TensorFlow object detection API. In this case, our object detection API is now installed and ready for use.

Then we run the next shell which downloads the model from the TensorFlow model zoo which checks the operating system which we are in and it uses the wget to get it from our pre-trained model URL which we defined previously and it cloned the TensorFlow model zoo and paste it inside a specific folder. We got a checkpoint folder that contains all of the different checkpoints and various files which helps when we freeze our model.

1) Create Label Map: A label map effectively represents a map of all our different labels such as mouse, monitor, keyboard, and CPU. We have successfully run this part without any errors [16].

2) Create TF records: TFRecords are a binary file format for storing data. Using TFRecord helps speed up training for our custom object detection model. TFRecord converts our annotation in our images into a file format which we can use [17].

3) Copy Model Config to Training Folder: In this shell, we have copied the model config. There are various models are there in the TensorFlow model zoo and the pipeline config controls the model actually looks like but we can able to customize this for our specific model which can be done by updating the components of our model thus for updating we have copied the model config.

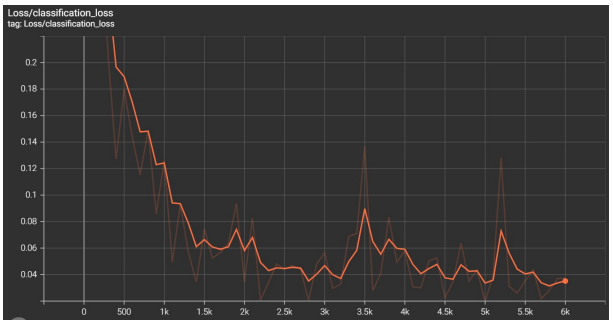
4) Update config For Transfer Learning: This shell was to update the components of the model.

5) Train the Model: We have run this outside the notebook so that it allows us to see the progress as our model is training by navigating the root TensorFlow Object Detection course folder in the command prompt and activating the environment.

Then we have pasted the command which runs the model training script which gets into the object detection folder and runs the script in it which is the modelmain tf2 after passing through the number of arguments we were passing through our model directory where the pipeline.config file is and we were passing through the actual pipeline config and passing through the number of training steps that we want to run. We have received the last parameter which is the number of training steps is 2000 and we changed it into 5000 then to 12000 where the model is fully trained with low loss.

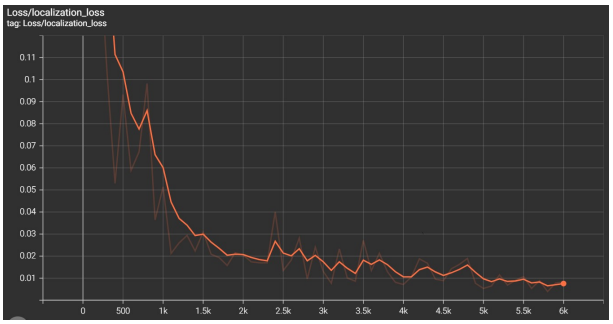
Due to over-fitting, the data has been increased from 80 to 100 and we augmented the data to 600 images in total and the number of training steps is 3000 for the augmented data. We achieved the intended result.

6) Evaluate the model: Once we have trained the model we paste the code in the command prompt for analyzing the performance metrics for the model, precision, recall, loss and we got the decent precision, performance, recall and loss then we evaluated the models with the Tensor board were we saw our results in the form of a graph [18].



Classification Loss Train

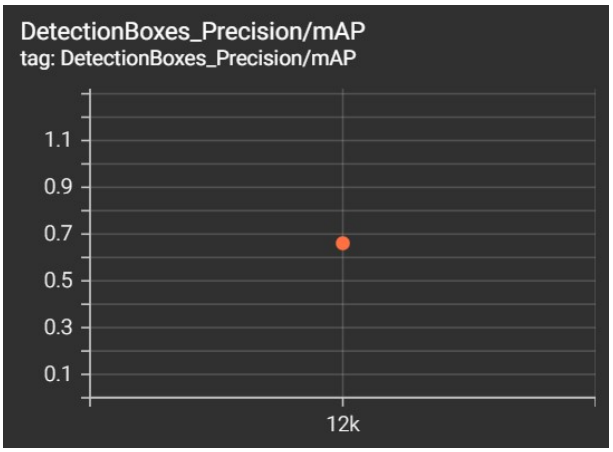
The above mentioned loss is the classification loss in the training. It starts from high and dropping low.



Localization Loss Train

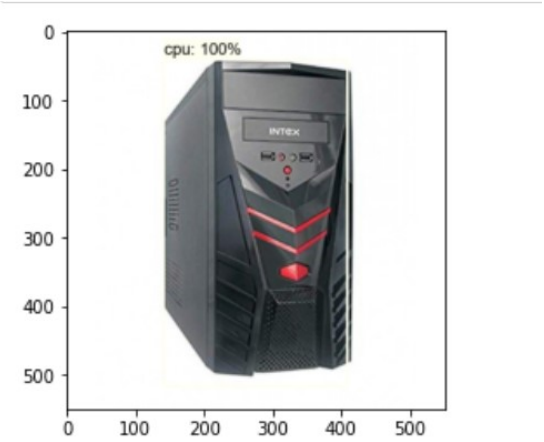
The above mentioned loss is the Localization loss in the training. It starts from high and dropping low.

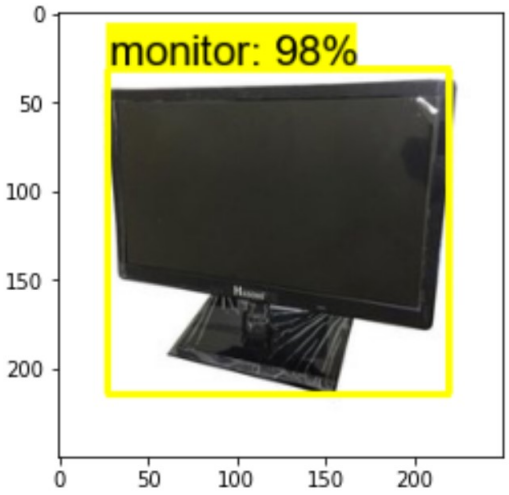
Then we evaluated the evaluation graph with the Tensorboard we got our mean average precision as seen in the below graphical image.



Evaluate Graph

7) Detecting Object: At first we had detected the object with the image by setting up the category index and we have grabbed the test image in the notebook. After running the code the shell runs the object detection on the particular image which is the monitor and the CPU and printed the result with the greater accuracy.





These are the results which we have achieved when we detect the object with the images with 100 percent accuracy for CPU and 98 percent Accuracy for the Monitor and also we tried it for the different images.

Even though MobileNet is very lightweight and low in latency, For some images, the precision and the accuracy are average as the object detected below.

Low precision in object

8) Real Time Detections from your Webcam: After running this shell we can able to detect the various objects like mouse, keyboard etc.with greater accuracy in real time with the webcam.

9) Freezing and Conversion: For saving a slice of our model for use next time we freeze our graph. So we freezed our graph [19] and converted it into ONNX format so that it can be integrated with ZED mini [20].

10) Converting it into ONNX Format: Exporting a TensorFlow neural network to ONNX takes a bit longer but it is still straightforward. For this we have installed the tf2onnx and pip install onnxruntime.

After the installation using the git repository we have converted the tensorflow in ONNX format which is feasible to adapt with the ZED mini [21].

IV. Conclusion

In this project, we have proposed and trained a MobileNet neural network that performs the detection of various objects like (mouse, monitor, keyboard, and CPU) in real-time by building the custom object detector and converting it into ONNX format to be reusable in any application with using ZED mini.

After the installation process of the various software and dependencies, we have collected the various images and labeled them. The process we did is trained TensorFlow Object Detection Models using TensorFlow API and detected various objects from images and the webcam and evaluated the model. The speed of the neural network is 19-20 ms(milli second). The process is to Freeze the TFOD and converted it into ONNX format. We have achieved 0.6 mAP (mean average precision). The loss is very low after 2000 steps. The fluctuation is very low and the low loss is stable.

The neural network which we have chosen is MobileNet which is very lightweight and low in latency. Even though it has many advantages to use, the precision and the accuracy are average. our objective was to build a lightweight faster object detection model. So the limitation did not have a huge effect on the final objective.

References

[1] Object detection with Deep Learning https://machinelearningmastery.com/object-recognition-with-deep-learning/
[2] attribute about object detection:A Review. https://arxiv.org/pdf/1807.05511.pdf
[3] https://www.stereolabs.com/zed-mini/
[4] https://www.tensorflow.org/hub/tutorials/object_detection
[5] https://anaconda.org/conda-forge/tf_object_detection
[6] Explanation of CUDA and cuDNN https://deeplizard.com/learn/video/6stDhEA0wFQ
[7] MobileNethttps://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d
[8] http://d2l.ai/chapter_computer-vision/rcnn.html RCNN Object Detection Algorithms
[9] https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e Fast RCNN Object Detection Algorithms
[10] https://arxiv.org/abs/1506.01497 Faster R-CNN
[11] https://medium.com/@techmayank2000/object-detection-using-ssd-mobilenetv2-using-tensorflow-api-can-detect-any-single-class-from-31a31bbd0691 Object Detection using SSD Mobilenet
[12] https://developer.nvidia.com/cuda-downloads
[13] https://www.tensorflow.org/install/pip TensorFlow Installation
[14] https://github.com/tzutalin/labelImggit link for Labelling Images
[15] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md Tensorflow model zoo
[16] https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html
[17] https://www.tensorflow.org/tutorials/load_data/tfrecordTF record
[18] https://www.tensorflow.org/tensorboardEvaluating the model by TensorBoard
[19] https://cv-tricks.com/how-to/freeze-tensorflow-models/#:~:text=Freezing%20is%20the%20process%20to,This%20contains%20the%20complete%20graph. Freezing the model
[20] https://github.com/onnx/tensorflow-onnx
[21] https://github.com/onnx/tensorflow-onnx/blob/master/tutorials/ConvertingSSDMobilenetToONNX.ipynb convert into onnx