IEEE CTSoc 2020 Remote Internship Project

Object detection for E-waste in the Indian context

Under the Guidance of:

Prabindh Sundareson

Worked on by: Risha Dassi

Table of Contents:

I. Introduction to problem

E-waste, or electronic waste, encompasses electrical and electronic equipment that's outdated, unwanted, or broken. E-waste contains a laundry list of chemicals that are harmful to people and the environment, like mercury, lead, beryllium, brominated flame retardants, and cadmium, i.e. stuff that sounds as bad as it is. When electronics are mishandled during disposal, these chemicals end up in our soil, water, and air. Thus the detection of E-waste is an essential step to work on further steps of safely disposing and recycling this waste to minimize its impact on the environment.

The goal of this project is to improve object detection for consumer E-waste in Indian Context.

II. Dataset

A. Data Collection

The datasets were sourced from standard platforms such as kaggle and also manually downloaded from the internet to get a variety of images and also have a dataset that contains images specific to the Indian context. The sources mentioned below were combined to form the dataset used for this project:

- waste_pictures
- Starter: e-waste dataset 93b07fb8-a
- Non-copyrighted images downloaded from google for the Indian context

B. Dataset Annotation

Labelling tool used: labelImg

LabelImg is a graphical image annotation tool. It is written in Python and uses Qt for its graphical interface. Annotations are saved as XML files in PASCAL VOC format. The instructions to run labelImg for annotations are given in its documentation on Github which is linked above.

The images were individually opened using labelImg and labeled among the classes mentioned below. Some images had objects of multiple classes within the same image and these were added to make the dataset as realistic as possible.

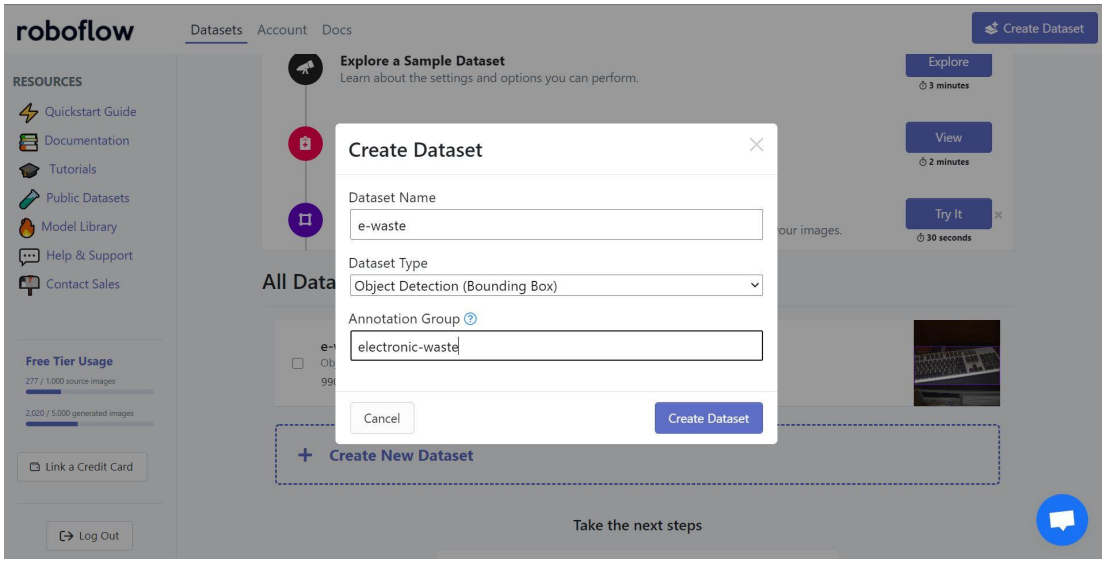
Classes: Battery, Bulb, Keyboard, Laptop, Monitor, Mobile Phone, Mouse

C. Data Preparation

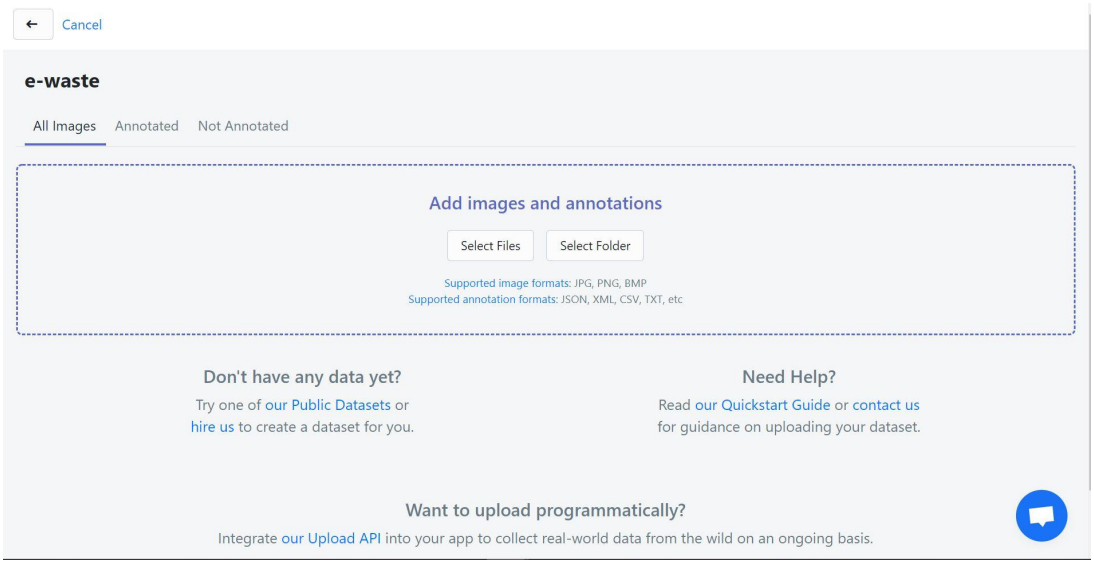
To obtain the final dataset for training in TFRecord format the platform used is Roboflow. Image processing and augmentation are also done here.

1. Steps

A. After creating an account on Roboflow one can click on the option of creating the dataset and give it a name, select its type and annotation group as shown in the image below.

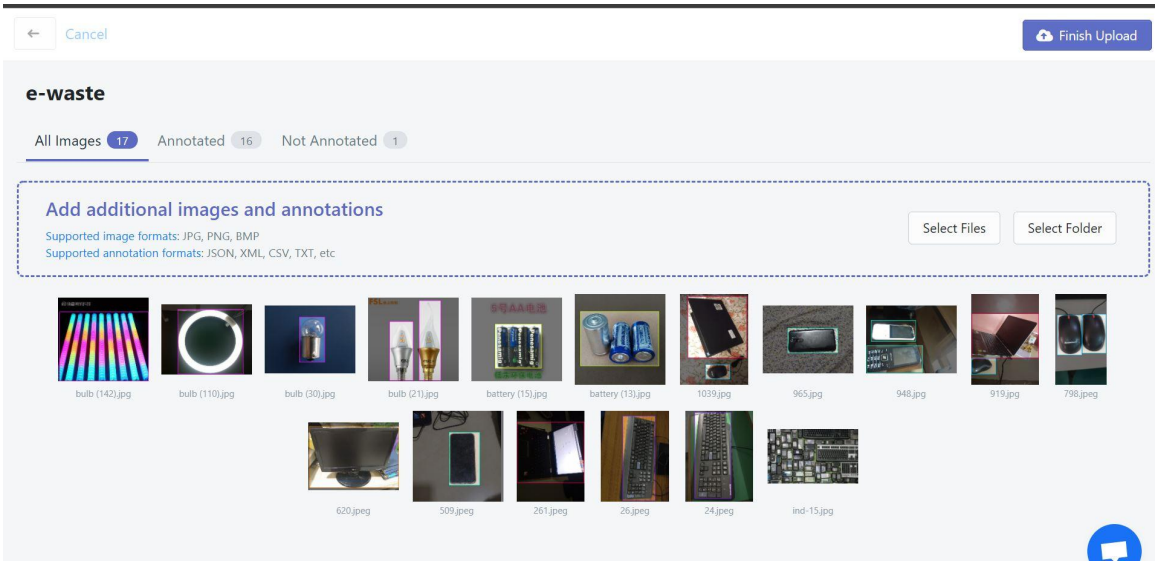


B. Next, you have to upload the images and the annotations that you would have completed in the data collection and annotation phase. You can either upload the folder with all the contents or upload files individually.

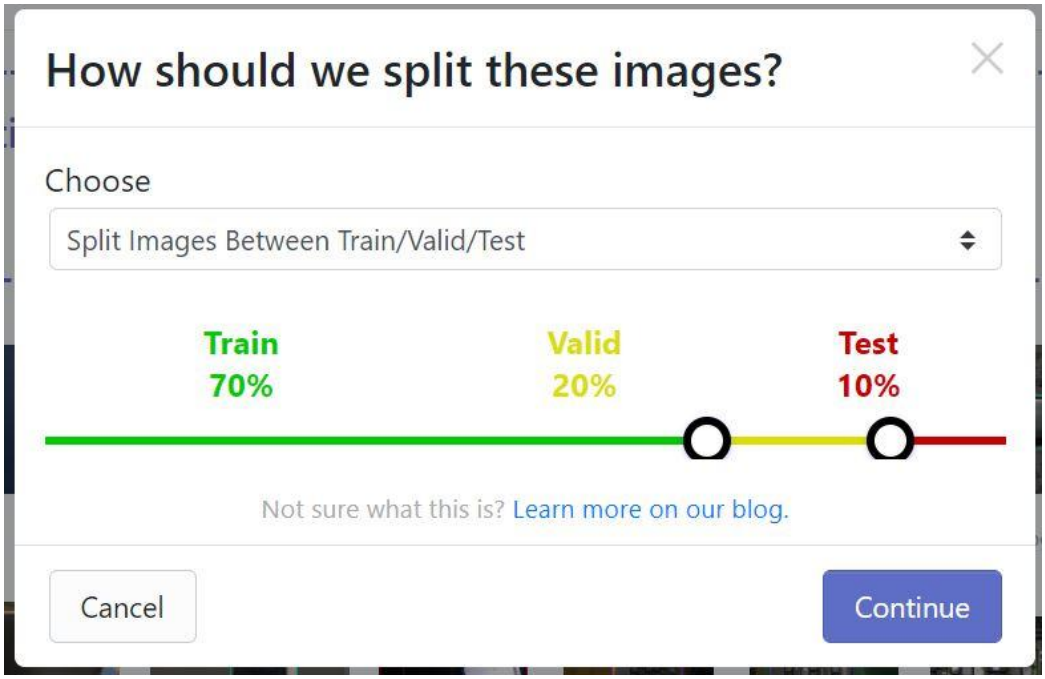


C. Once the images are upload as seen in the sample screenshot below, you can see that Roboflow notifies you if some images are not annotated and you can

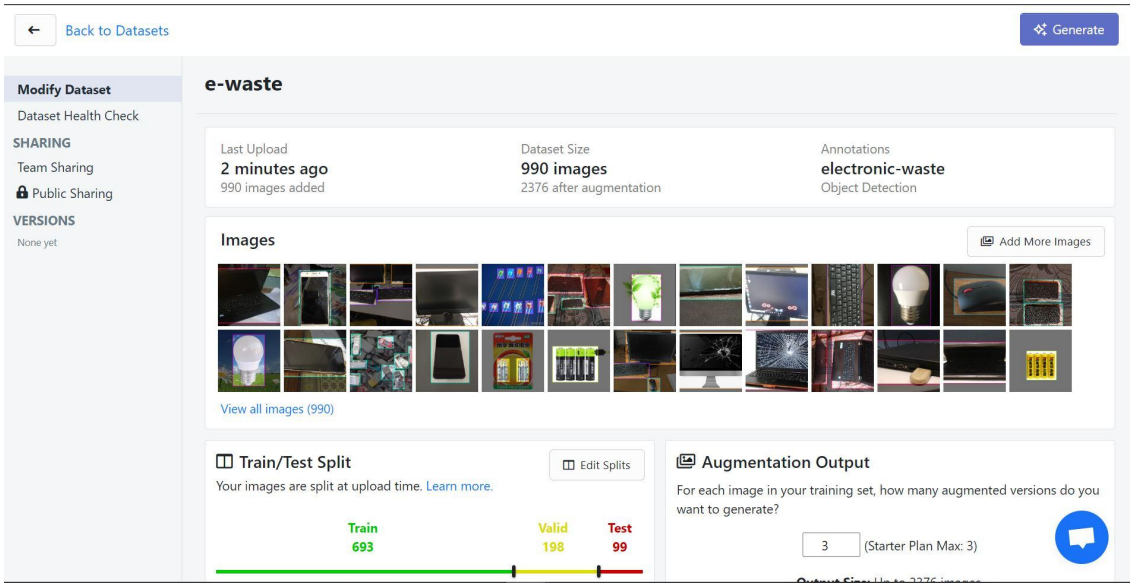
add the annotation for those images or remove the image from the dataset. Click on `Finish Upload` once you are satisfied with the images in the dataset.



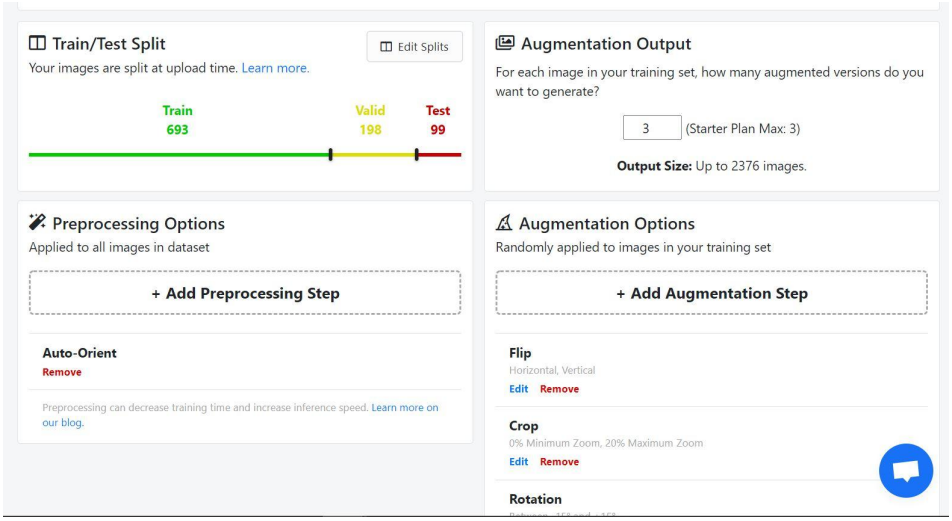
D. Then you will be prompted to select how you wish to split the dataset. It is preferable to choose the Train: Valid: Test split since it can minimize the effects of data discrepancies and help us understand the characteristics of the model better.

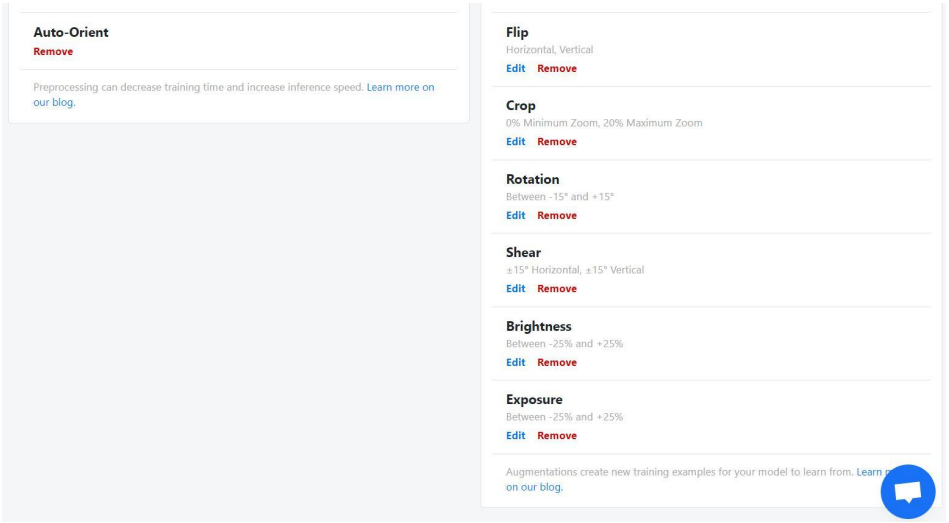


E. The image below shows the dataset uploaded for the model trained.
   a. The train: valid: test split of the dataset is 636: 128: 91
   b. Size of the dataset before augmentation: 990 images
   c. Size of the dataset after augmentation: 2376 images



F. To perform preprocessing steps or augmentation on the uploaded dataset, you can click on `Add Preprocessing Step` and `Add Augmentation Step` respectively. The interactive GUI assists you in selecting both and you can also alter parameters for augmentation steps like "shear", "crop", "rotation" etc. The augmentation steps selected by us are:
    a. Flip
    b. Crop (min zoom- 0% ; max zoom- 20%)
    c. Rotation (between -15° and 15°)
    d. Shear (Horizontal: +15° ; Vertical: +15°)
    e. Brightness (-25% to +25%)
    f. Exposure (-25% to +25%)

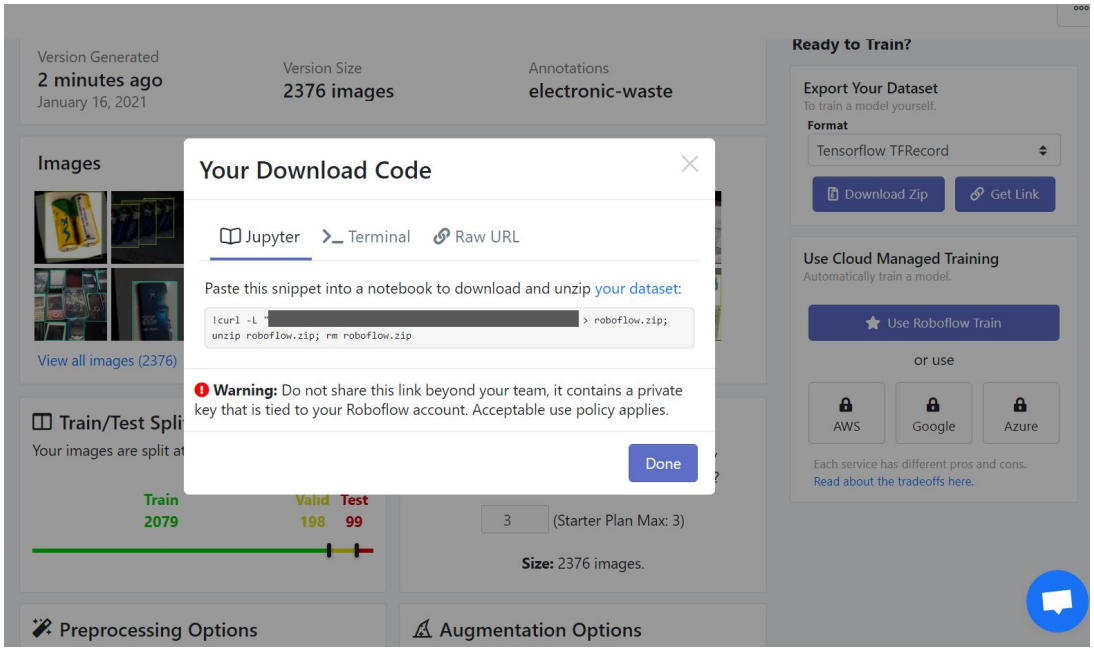




Once you have added the augmentations and pre-processing steps you need,

click on `Generate Dataset` to generate the final dataset that will be used for training.

G. Finally, you will be given the option to either download the zip version of the code in the chosen export format (here the chosen format is TFRecord) or copy the link for the Jupyter notebook which is used in this implementation to load the training dataset for the model.



The final details of the data are:

- Train/Test split- 693 : 198 : 99 (train : valid : test)
- Preprocessing steps: Auto-Orient
- Augmentation: Flip (Horizontal, Vertical); Rotation; Shear; Brightness; Exposure; Brightness; Crop
- Size of the dataset before augmentation: 990 images
- Size of the dataset after augmentation: 2376 images

[Processing is done via Roboflow]

The final dataset used can be found here

III. Implementation

A. Phase 1: Few Shot Training

Few-shot learning (FSL), is a type of machine learning problem where the training dataset contains limited information.

A common practice for machine learning applications is to feed as much data as the model can take to enable the model to predict better. However, few-shot learning aims to build accurate machine learning models with less training data. As the dimension of input data is a factor that determines resource costs (e.g. time costs, computational costs etc.), one can reduce data analysis/machine learning (ML) costs by using few-shot learning.

Code can be found here

Steps:

1. Clone the tensorflow models repository if it doesn't already exist.
2. Install the Object Detection API and import necessary libraries.
3. Load images [option of annotating and labelling] and visualize.
4. Load pipeline config and build a detection model.
5. Select variables in top layers to fine-tune.
6. Load test images and run inference with the new model.

B. Phase 2: TF2.0 Object Detection (Roboflow)

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models.

Tensorflow provides a collection of detection models pre-trained on the COCO 2017 dataset. These models can be useful for out-of-the-box inference. The model used for our implementations is `EfficientDet D0 512x512` which is found with other EfficientDet models in the Roboflow notebook here.

Steps

1. Install TensorFlow2 Object Detection Dependencies.
2. Download Custom TensorFlow2 Object Detection Dataset.
3. Write Custom TensorFlow2 Object Detection Training Configuration.
4. Train Custom TensorFlow2 Object Detection Model.
5. Export Custom TensorFlow2 Object Detection Weights.
6. Use Trained TensorFlow2 Object Detection For Inference on Test Images.

Note:

- Add your own dataset link generated on Roboflow in the notebook while training or upload your dataset in .TFRecord format.

```
#Downloading data from Roboflow

%cd /content

!curl -L "DATASET LINK HERE" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

- Update the names of the TFRecord names from "cells" and "cells_label_map" to your files

```
test_record_fname = '/content/test/filename.tfrecord'

train_record_fname = '/content/train/filename.tfrecord'

label_map_pbtxt_fname = '/content/train/filename_label_map.pbtxt'
```

C. Phase 3: Optimizations in TF2.0 Roboflow Implementation

- **`momentum_optimizer`**: Momentum or Stochastic Gradient Descent with momentum is a method that helps accelerate gradient vectors in the right directions, thus leading to faster converging. It is one of the most popular optimization algorithms and many state-of-the-art models are trained using it.

- **`adam_optimizer`**: Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

- Replacing Momentum with Adam optimizer in **`/content/models/research/deploy/pipeline_file.config`** after we write the custom config file:

    - Default optimizer in config file:

```
optimizer {
    momentum_optimizer: {
        learning_rate: {
            cosine_decay_learning_rate {
            learning_rate_base: 8e-2
            total_steps: 300000
            warmup_learning_rate: .001
            warmup_steps: 2500
            }
        }
        momentum_optimizer_value: 0.9
        }
        use_moving_average: false
    }
    max_number_of_boxes: 100
    unpad_groundtruth_tensors: false
}
```

- Changing it to ADAM:

```
    optimizer {
    # momentum_optimizer {
        adam_optimizer: {
        learning_rate: {
            manual_step_learning_rate {
            initial_learning_rate: .0002
            schedule {
                step: 4500
                learning_rate: .0001
            }
            schedule {
                step: 7000
                learning_rate: .00008
            }
            schedule {
                step: 10000
                learning_rate: .00004
            }
            }
        }
        # momentum_optimizer_value: 0.9
        }
        use_moving_average: false
    }
}
```

IV. Results

Instead of using only the gradient of the current step to guide the search, momentum accumulates the gradient of the past steps to determine the direction to go. While momentum accelerates our search in the direction of minima, RMSProp impedes our search in the direction of oscillations.

Adam or Adaptive Moment Optimization algorithm combines the heuristics of both Momentum and RMSProp.

While both the models were able to detect objects from images of electronic devices that comprise of E-waste, in terms of efficiency the models compared as follows:

A. Loss

After running the model for 25000 epochs for momentum_optimizer and adam_optimizer we get the loss value as 0.254 and 0.227 for the different models obtained as seen in the images below.

Adam

```
I0118 02:41:29.437662 140147324303232 model_lib_v2.py:651] Step 24400 per-step time 1.230s loss=0.184
INFO:tensorflow:Step 24500 per-step time 1.267s loss=0.175
I0118 02:43:31.182879 140147324303232 model_lib_v2.py:651] Step 24500 per-step time 1.267s loss=0.175
INFO:tensorflow:Step 24600 per-step time 1.207s loss=0.145
I0118 02:45:31.846111 140147324303232 model_lib_v2.py:651] Step 24600 per-step time 1.207s loss=0.145
INFO:tensorflow:Step 24700 per-step time 1.285s loss=0.162
I0118 02:47:33.315750 140147324303232 model_lib_v2.py:651] Step 24700 per-step time 1.285s loss=0.162
INFO:tensorflow:Step 24800 per-step time 1.237s loss=0.180
I0118 02:49:35.407914 140147324303232 model_lib_v2.py:651] Step 24800 per-step time 1.237s loss=0.180
INFO:tensorflow:Step 24900 per-step time 1.221s loss=0.227
I0118 02:51:35.839375 140147324303232 model_lib_v2.py:651] Step 24900 per-step time 1.221s loss=0.227
```

Momentum

```
I0118 02:48:46.386409 140660891088768 model_lib_v2.py:651] Step 24600 per-step time 1.345s loss=0.187
INFO:tensorflow:Step 24700 per-step time 1.254s loss=0.244
I0118 02:50:49.340247 140660891088768 model_lib_v2.py:651] Step 24700 per-step time 1.254s loss=0.244
INFO:tensorflow:Step 24800 per-step time 1.211s loss=0.224
I0118 02:52:50.985520 140660891088768 model_lib_v2.py:651] Step 24800 per-step time 1.211s loss=0.224
INFO:tensorflow:Step 24900 per-step time 1.178s loss=0.186
I0118 02:54:53.302254 140660891088768 model_lib_v2.py:651] Step 24900 per-step time 1.178s loss=0.186
INFO:tensorflow:Step 25000 per-step time 1.183s loss=0.254
I0118 02:56:56.244026 140660891088768 model_lib_v2.py:651] Step 25000 per-step time 1.183s loss=0.254
```

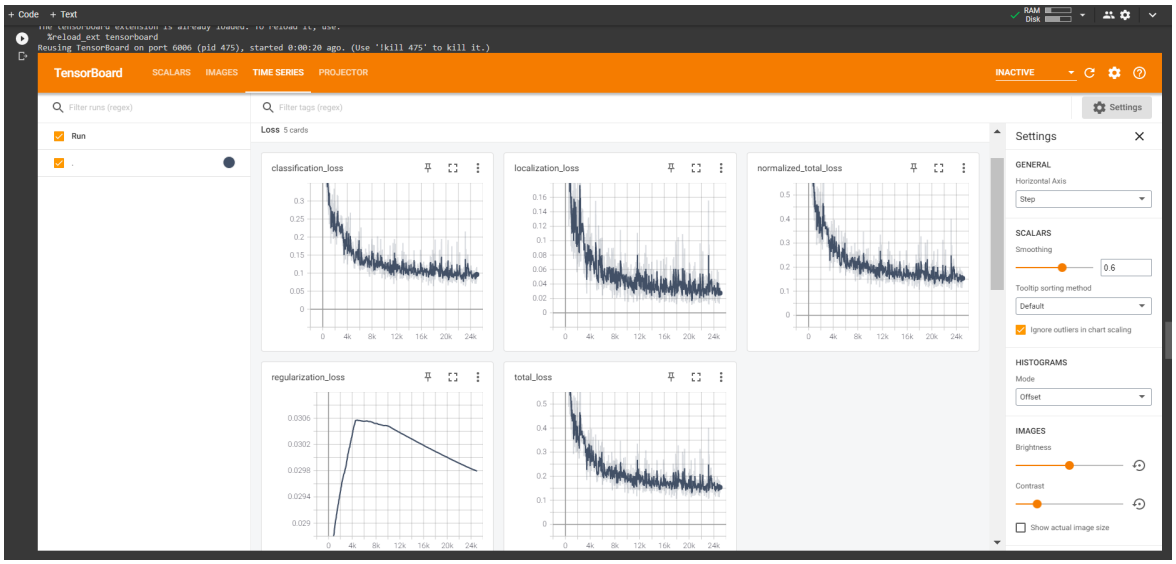
As seen in the images above, Adam optimizer is able to achieve better efficiency with a loss value of 0.227 as compared to momentum's loss value of 0.254 and thus in future iterations/improvements, using the Adam optimizer would be preferable.

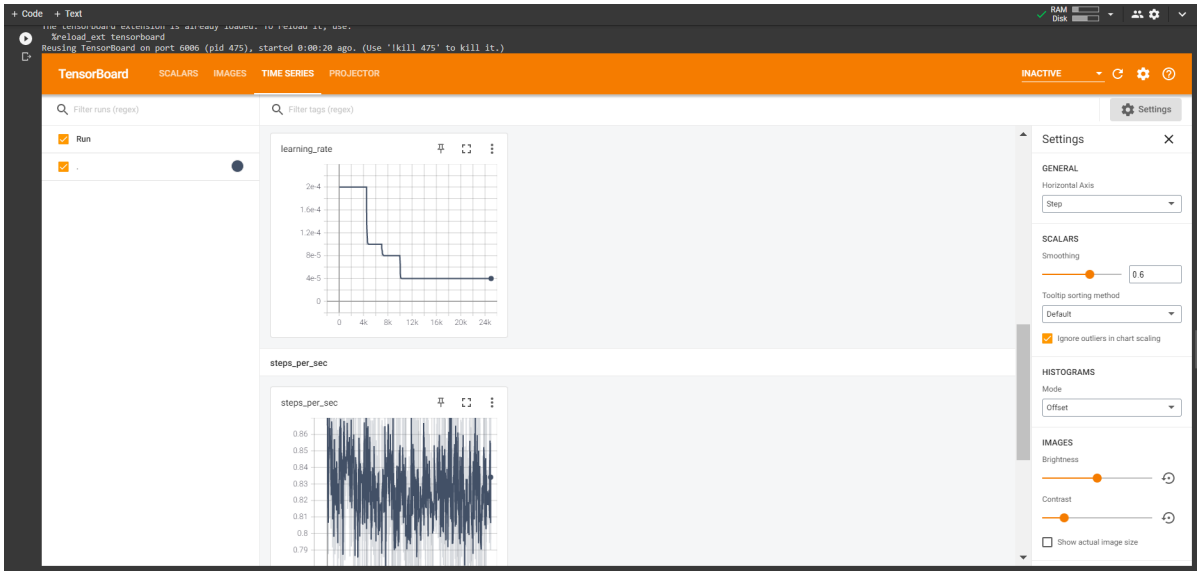
B. Tensorflow Board

TensorBoard is a tool for providing the measurements and visualizations needed during the machine learning workflow. It enables tracking experiment metrics like loss and accuracy, visualizing the model graph, projecting embeddings to a lower dimensional space, and much more.

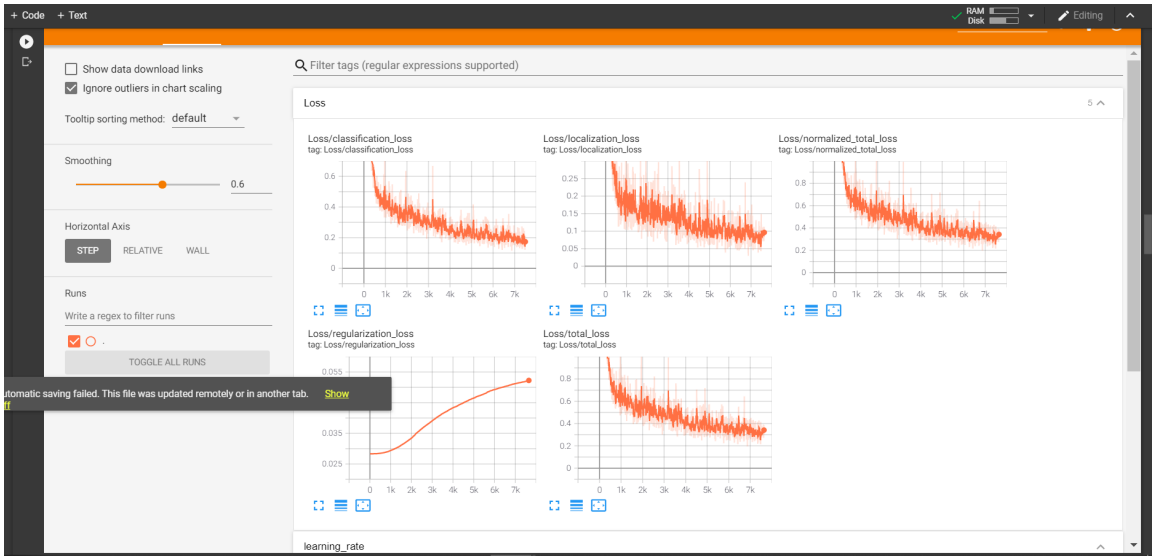
The screenshots below show that Adam optimizer has an adaptive learning rate which helps it reach the minima much more efficiently in comparison to momentum which has the same learning rate throughout after one point.

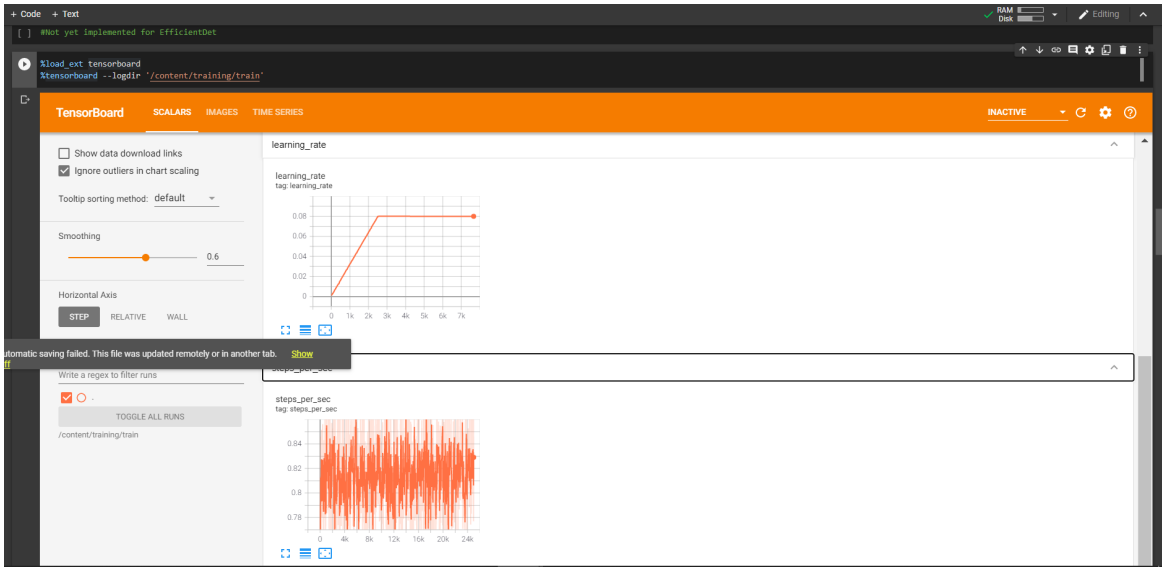
Adam





Momentum

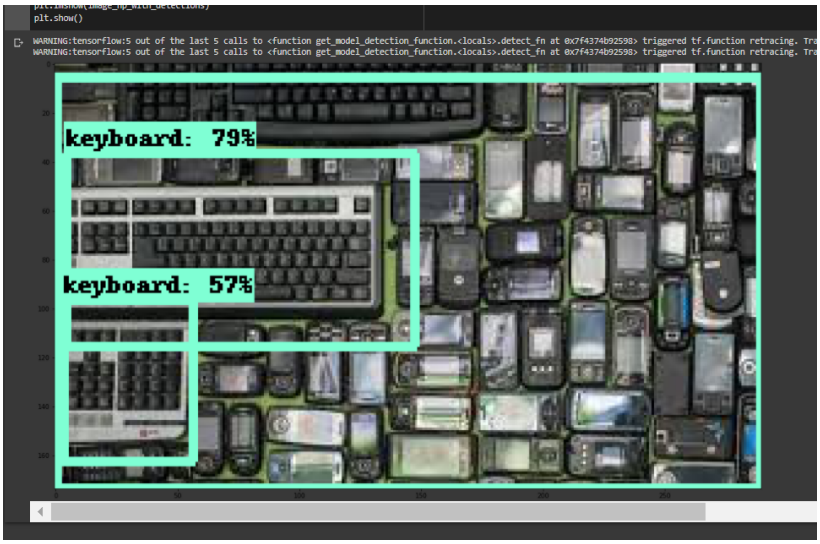


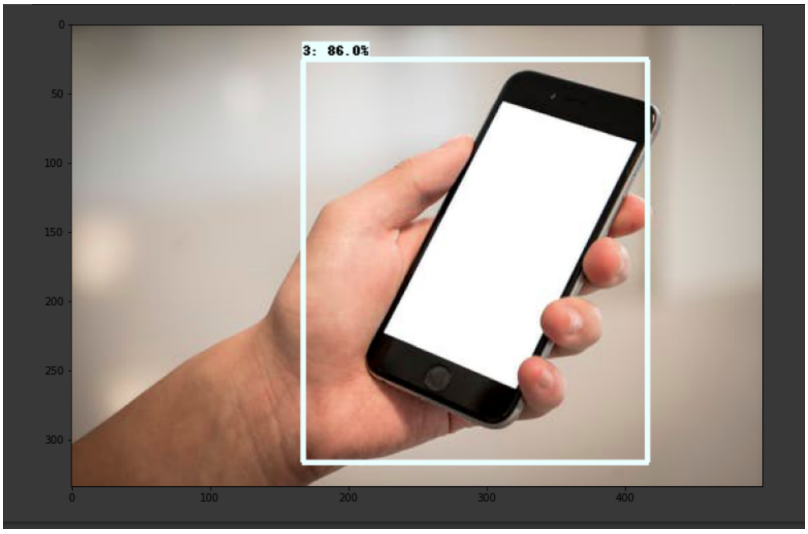


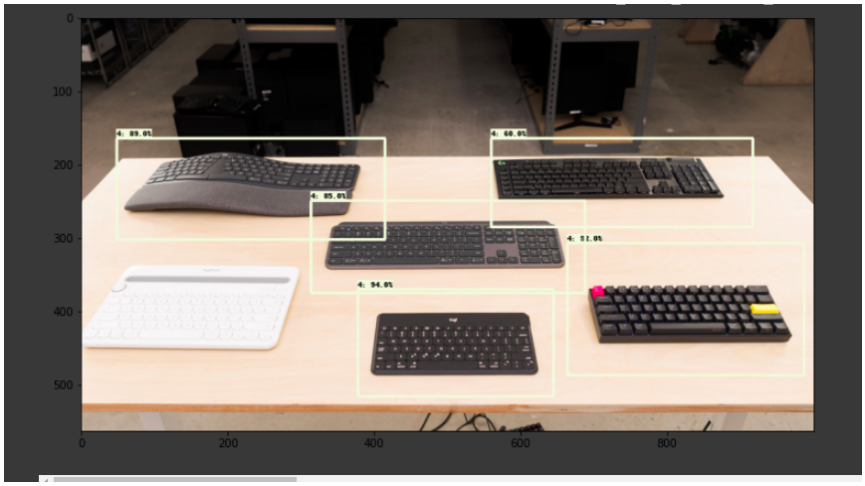
V. Object Detection Result Images

The model shows significant improvement in detecting multiple objects within the same image while using Adam instead of Momentum optimizer. Not only does the confidence of detection increase but also the model is able to detect objects in a realistic scenario which is what the aim of the project is.

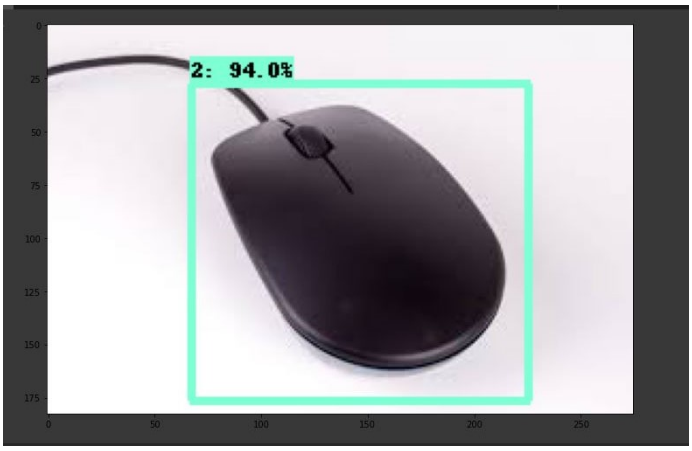
Adam

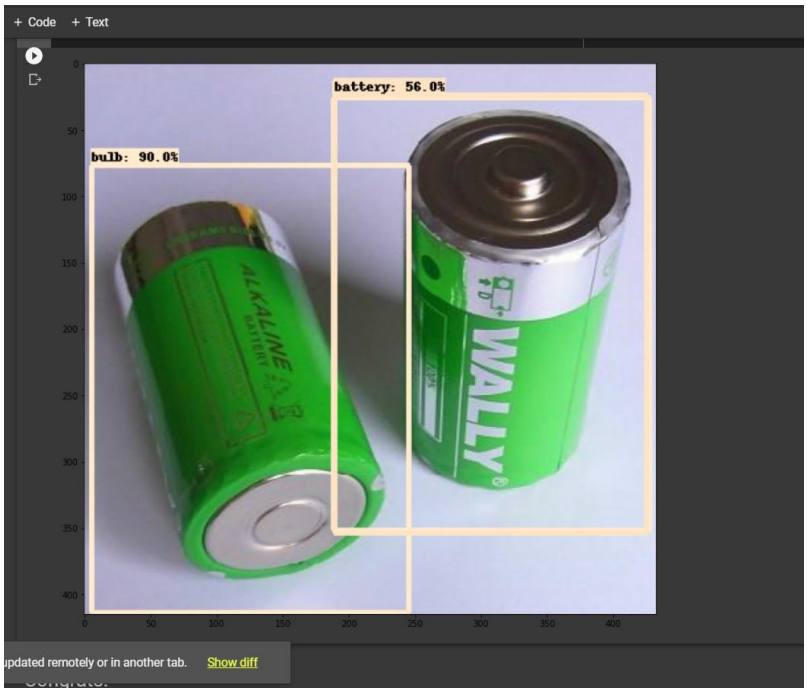

plt.show()
WARNING:tensorflow:5 out of the last 5 calls to <function get_model_detection_function.<locals>.detect_fn at 0x7f4374b92598> triggered tf.function retracing.
WARNING:tensorflow:5 out of the last 5 calls to <function get_model_detection_function.<locals>.detect_fn at 0x7f4374b92598> triggered tf.function retracing.
keyboard: 79%
keyboard: 57%


3: 86.0%
0
50
100
150
200
250
300
0
100
200
300
400



Momentum





VI. Conclusion and Future Scope

The project was able to achieve its goal of being able to detect E-waste in an Indian context and also compared the performance of two optimizers (Adam and Momentum) for the object detection model to see which of these is better.
The results obtained clearly show that Adam is the better of the two and can be used in future work on the project or in any other similar project.

The project can be extended in the future to cover non electronic waste as well and the object detection models can be used to aid in automated segregation of waste with the appropriate technology.

VII. Resources

1. https://ewh.ieee.org/r10/bangalore/ces/
2. https://recyclecoach.com/residents/blog/an-intro-to-e-waste-why-its-a-problem
3. https://colab.research.google.com/drive/1sLqFKVV94wm-lglFq_0kGo2ciM0kecWD
4. https://www.tensorflow.org/guide/tensor
5. https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f
6. https://www.tensorflow.org/guide/basic_training_loops
7. https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52
8. https://research.aimultiple.com/few-shot-learning/
9. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tf2.md
10. https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d
11. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning
12. https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/