VISUAL POLLUTION EFFICIENTDET

A project submitted to Smartathon challenge for theme 1:
Detection and evaluation of the following elements on street imagery taken from moving vehicles

Team members:
Reem Hejazi 219410002@psu.edu.sa
Nour Alakhras 220410351@psu.edu.sa
Ayat Abodayeh 220410035@psu.edu.sa

Project Overview

To shed light on the significance of adopting environment management solutions, our project has utilized a convolutional neural network (CNN) in order to automate the classification of visual pollutants. By doing so, we have fine-tuned a pre-trained EfficientDet D0 model, implemented within the TensorFlow Object Detection API, which we retrained on raw sensor camera inputs (9,966 images) as perceived by multiple vehicles in a restricted geographic area in KSA. We opted for the EfficientDet model since it's an effective and scalable multi-object detection model based on BiFPN that works with a high speed and COCO mAP without compromising on either of them. The architecture is shown in Figure 1. We first converted the csv file to a tfrecord file using our TfRecordConverter. We trained our model after configuration using the CPU of two devices (both with an i7 core and 16 GB RAM, but one of them had the Nvidia Geforce GTX 1050 GPU). After exporting the trained and evaluated model, we ran the inference Python program we created in order to apply non-max suppression. We chose the results of the model that had a better accuracy.

We have faced many challenges in our project's implementation since it is our first time confronting convolutional neural networks. First, we struggled to train our model using the imperfect dataset, which was significantly skewed, since we used a CPU and had limited computational power. We tried training our model via GPU instead but were not able to utilize gradient check-pointing; hence, the GPU's limited RAM could not handle training the large dataset. Furthermore, our model had many bounding boxes despite using the pre-trained EfficientDet. We were able to eliminate this issue by building our own non-max suppression code which successfully used intersection over union and minimum score threshold in order to increase the model's mean average precision and reduce the number of bounding boxes by optimally selecting the most appropriate bounding boxes. Moreover, we struggled with the installation of the required tools since there were several issues with regards to the compatibility with our devices and the pre-installed Python libraries we had. We resolved these issues by testing multiple commands on the terminal according to each of the errors that we have faced. As a result of the limited resources and time we had, the mean average precision was lower than what we had expected.

To make our solution more scalable, we suggest augmenting the training data set in order to increase the model's precision. We also recommend training our model on powerful GPUs for more enhanced speed and efficiency. In terms of environment management, we suggest increasing the amount of classes as there are more pollutants that can pose harm to society. Upon utilizing our model, technological experts can extend it to applications that upon detecting these pollutants will alert local organizations to concentrate their efforts towards cleaning and renovating the affected areas. Finally, this model can be implemented on pictures from different areas around the world in order to detect these pollutants and execute the necessary plans to eradicate them.

The open source software we have used in our model includes the following: TensorFlow Object Detection API, Efficient Detection D0 512x512 from the TensorFlow 2 Detection Model Zoo (pretrained on a COCO 2017 dataset), COCO pytools, and Anaconda where we set up our TensorFlow virtual environment.

If we had more time and better resources, we would have first used gradient check pointing in order to enhance the precision of our model. We would also have used more powerful devices that could significantly decrease the amount of time it took to train the model which, in turn, would have given us the opportunity to fine-tune a better version of EfficientDet. Finally, we would have modified the training dataset since it suffered from severe imbalance and could have been better if it was inclusive of more classes that target other pollutants around the kingdom.

For more information on our work: https://github.com/rem2718/visual_pollution_efficientDet

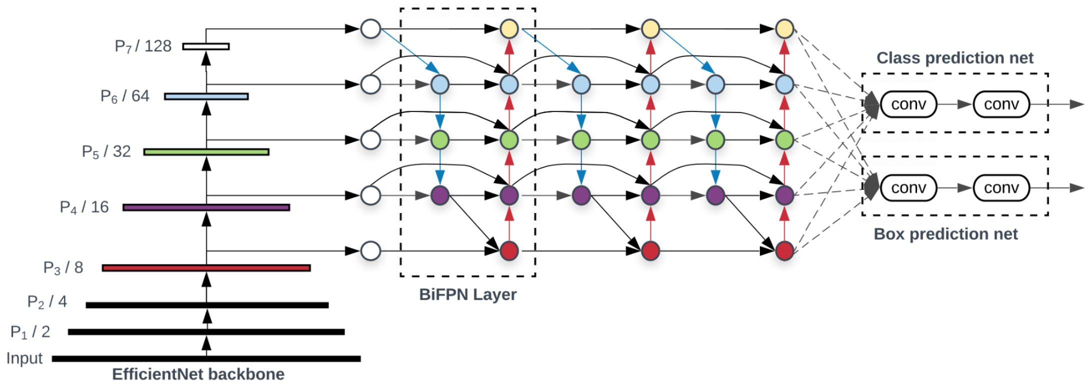
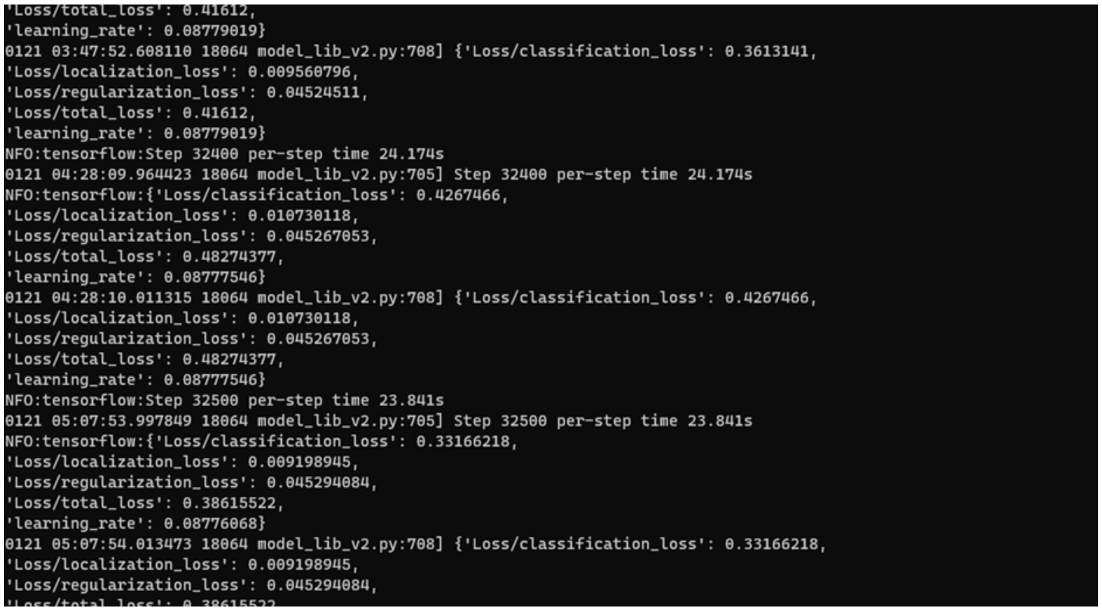


Figure 1: *EfficientDet architecture using EfficientNet as the backbone architecture, BiFPN as the feature network, and shared class/ box prediction network*



```
'Loss/total_loss': 0.41612,
'learning_rate': 0.08779019}
0121 03:47:52.608110 18064 model_lib_v2.py:708] {'Loss/classification_loss': 0.3613141,
'Loss/localization_loss': 0.009560796,
'Loss/regularization_loss': 0.04524511,
'Loss/total_loss': 0.41612,
'learning_rate': 0.08779019}
NFO:tensorflow:Step 32400 per-step time 24.174s
0121 04:28:09.964423 18064 model_lib_v2.py:705] Step 32400 per-step time 24.174s
NFO:tensorflow:{'Loss/classification_loss': 0.4267466,
'Loss/localization_loss': 0.010730118,
'Loss/regularization_loss': 0.045267053,
'Loss/total_loss': 0.48274377,
'learning_rate': 0.08777546}
0121 04:28:10.011315 18064 model_lib_v2.py:708] {'Loss/classification_loss': 0.4267466,
'Loss/localization_loss': 0.010730118,
'Loss/regularization_loss': 0.045267053,
'Loss/total_loss': 0.48274377,
'learning_rate': 0.08777546}
NFO:tensorflow:Step 32500 per-step time 23.841s
0121 05:07:53.997849 18064 model_lib_v2.py:705] Step 32500 per-step time 23.841s
NFO:tensorflow:{'Loss/classification_loss': 0.33166218,
'Loss/localization_loss': 0.009198945,
'Loss/regularization_loss': 0.045294084,
'Loss/total_loss': 0.38615522,
'learning_rate': 0.08776068}
0121 05:07:54.013473 18064 model_lib_v2.py:708] {'Loss/classification_loss': 0.33166218,
'Loss/localization_loss': 0.009198945,
'Loss/regularization_loss': 0.045294084,
'Loss/total_loss': 0.38615522
```

Figure 2: *Train logs*

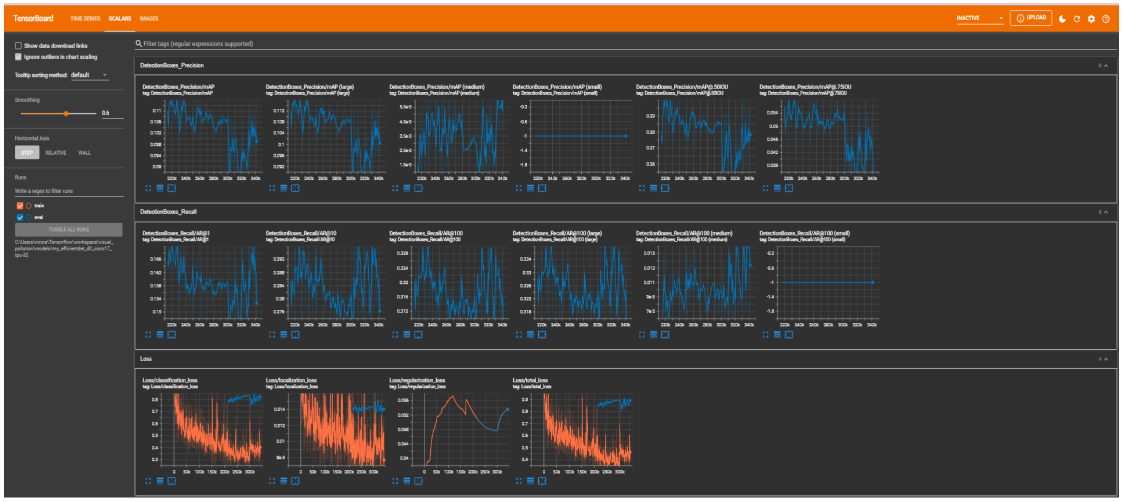

TensorBoard
TIME SERIES
SCALARS
IMAGES
INACTIVE
UPLOAD
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train
eval
TOGGLE ALL RUNS
C:\Users\noora\Tensorflow\workspace\visual_pollution\models\my_efficientdet_d0_coco17_tpu-32
Filter tags (regular expressions supported)
DetectionBoxes_Precision
DetectionBoxes_Precision/mAP
DetectionBoxes_Precision/mAP (large)
DetectionBoxes_Precision/mAP (medium)
DetectionBoxes_Precision/mAP (small)
DetectionBoxes_Precision/mAP@.50IOU
DetectionBoxes_Precision/mAP@.75IOU
DetectionBoxes_Recall
DetectionBoxes_Recall/AR@1
DetectionBoxes_Recall/AR@10
DetectionBoxes_Recall/AR@100
DetectionBoxes_Recall/AR@100 (large)
DetectionBoxes_Recall/AR@100 (medium)
DetectionBoxes_Recall/AR@100 (small)
Loss
Loss/classification_loss
Loss/localization_loss
Loss/regularization_loss
Loss/total_loss

Figure 3: *Model training performance summary via TensorBoard*

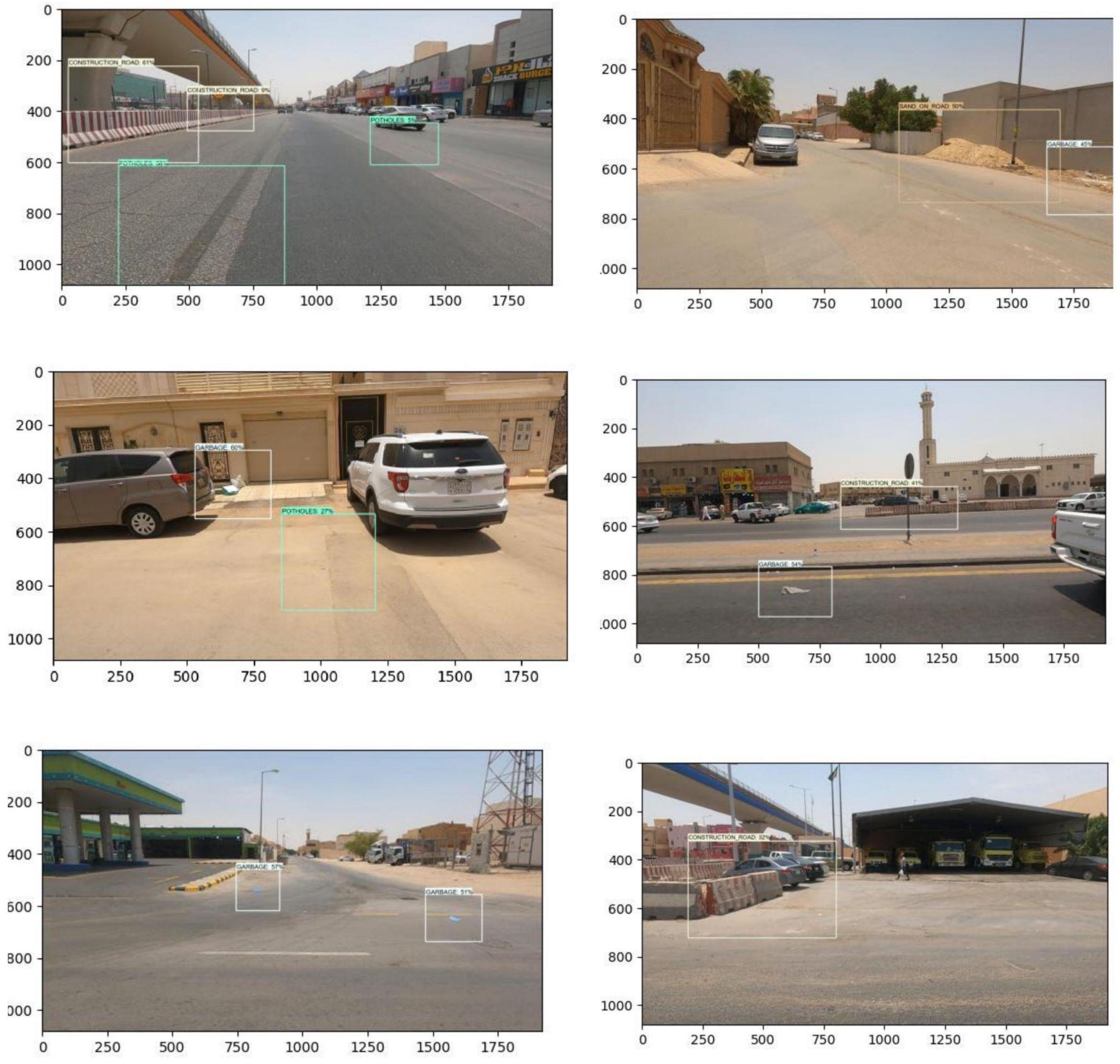


Figure 4: *Output of test images based on our model*