Introduction

The datasets used in this project consist of 240 images captured using a smartphone camera, with approximately half taken in city Stirling and the other half taken in city Tashkent. By training the models on separate datasets, and testing them crossing the datasets, we can evaluate their performance in different environmental conditions and assess their generalization capabilities.

The project was developed using the PyTorch library for deep learning, and a pre-trained Object Detection model called "fasterrcnn_resnet50_fpn"[1] was used as a backbone for the models. The pre-trained model was fine-tuned on a custom dataset, which consisted of 240 images of cars and trees taken using a smartphone camera in two different cities.

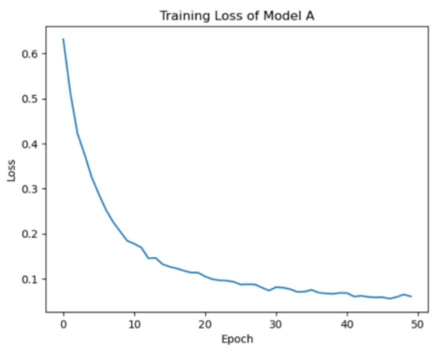
This report will explain the methodology used to create the Object Detection models, including the pre-processing of the datasets and the fine-tuning of the pre-trained model. This report will also present the results of the models' performance, including their Intersection over Union (IoU) scores [8]. To illustrate the models' performance, we will provide example visualizations of object detections on random images from the test set.

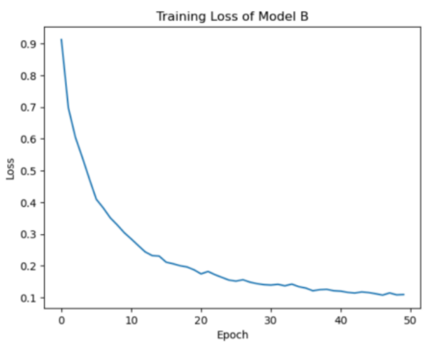
Description of the proposed solution with justifications

We collected 240 images of cars and trees using a smartphone camera in two cities. The images were resized to a width of 416 pixels while preserving their aspect ratio using bulkresizephotos.com [2] tool and labeled using makesense.ai [3] to generate VOC format label files. To load and transform the data during training, we created a custom PyTorch dataset class called "CustomDataset" [4] that reads images and corresponding label files, applies transformations, and returns a dictionary object containing the image, bounding box coordinates, and labels. The dataset has three labels: background, tree, and car, and the background label is mandatory when using the pre-trained "fasterrcnn_resnet50_fpn" model [1]. PyTorch's "DataLoader" [5] class was used to iterate over the custom dataset in batches during training, setting the batch size to 1 since "collate_fn" was not implemented. We implemented functions for training and testing the models, with the former looping through the DataLoader instance to pass images, bounding box coordinates, and labels to the model for loss calculation and gradient update. The number of epochs was set to 50 and the learning rate to 0.005. During testing, the model was set to evaluation mode and predictions of bounding box coordinates and corresponding labels were fed into "SmoothL1Loss" [6] and "cross_entropy" [6] loss functions, respectively. The final IoU value of the model was calculated by averaging the IoU values of each sample in the test dataset.

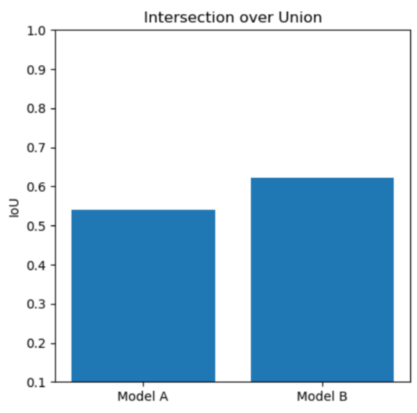
Results

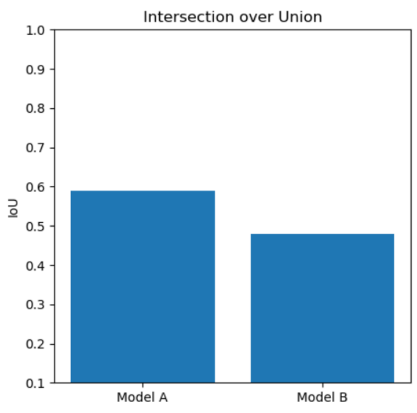
The following graphs illustrate the change in the loss during training of model “A” and model “B”.





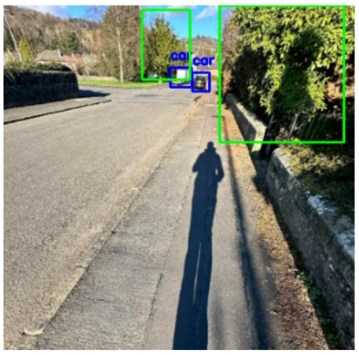
The two bar charts below depict the IoU values of two models on different test datasets. In the first chart, Model "A" was tested on a test dataset from city "A" while Model "B" was tested on a test dataset from city "B". In the second chart, Model "A" was tested on a test dataset from city "B" while Model "B" was tested on a test dataset from city "A".



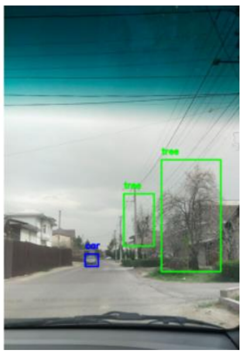


Examples of Model “A” and “B” performances on random images from dataset from city “A”.

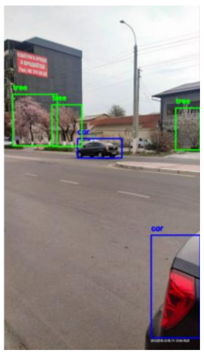
Model “A” performance on image from dataset “A”



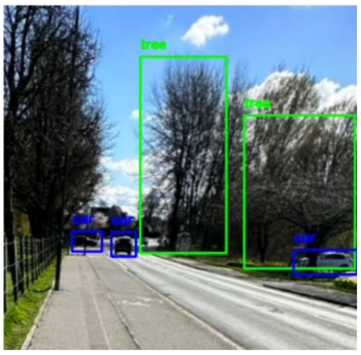
Model “A” performance on image from dataset “B”



Model “B” performance on image from dataset “B”



Model “B” performance on image from dataset “A”



Discussion of the results

According to the results, when models are tested on their corresponding datasets the model “B” performs better, however when datasets are crossed, model “A” has a higher performance, as it has higher IoU value in the second bar chart. Furthermore, the fact that Model A outperforms Model B when tested on images from a different city, despite being trained on images from a different city, suggests that Model A has better generalization ability and is less prone to overfitting. [7]

Conclusion

In conclusion, although the models have shown some promising results with their IoU values being approximately 60% in their best performance, there is still room for improvement. One way to achieve better results is by expanding the dataset, especially when it comes to the images of trees. This is because trees come in various shapes, sizes, and volumes, making it more challenging to accurately detect and classify them compared to cars, which have more consistent shapes.

References:

1. PyTorch’s fasterrcnn_resnet50_fpn documentation: Link
2. bulkresizephotos: Link
3. makesense.ai: Link
4. Creating custom dataset PyTorch documentation: Link
5. Creating DataLoader PyTorch documentation: Link
6. C. Versloot, "How to Use PyTorch Loss Functions," GitHub, 2021. [Online]. Available: Link
7. J. Brownlee, "Overfitting and Underfitting With Machine Learning Algorithms," Machine Learning Mastery. [Online]. Available: Link. [Accessed: Apr. 7, 2023].
8. Stack Overflow, "Calculating percentage of Bounding box overlap, for image detector evaluation": Link