Automatic ID-Card Detection Using Pretrained Faster R-CNN

Course : Computer Vision and Pattern Recognition

Section : A Semester : Summer (21-22)

Submitted By : Group [D]

Group Members :

Student ID	Name
20-41848-1	Anindra Das Bivas
20-42273-1	Amit Podder
20-42546-1	MD. Mohituzzaman
18-38839-3	MD. Miftahul Alam
19-40151-1	Tanha Reja

Submitted To :

DR. HOSSAIN MD SHAKHAWAT

Assistant Professor , Computer Science

shakhawat@aiub.edu

I. Introduction

The project proposes the use of a deep learning model for automatically detecting an ID card. The project was based on object detection task which was performed with the help of transfer learning.

A. Problem statement and motivation

The main problem that was addressed was the manual ID card detection at any workplace or educational institution require more labor and can also be time consuming and errors are made as well since detecting ID card at distance requires very precise vision to distinguish between different type of ID cards . Thus, it requires automation. Since, detecting ID card at distance can be a difficult task therefore use of computer vision techniques was proposed in this project. The main motivation of the project was to automate the task of manual ID card detection and provide an easy and sustainable solution that could replace the manual identification that requires a physical people in place.

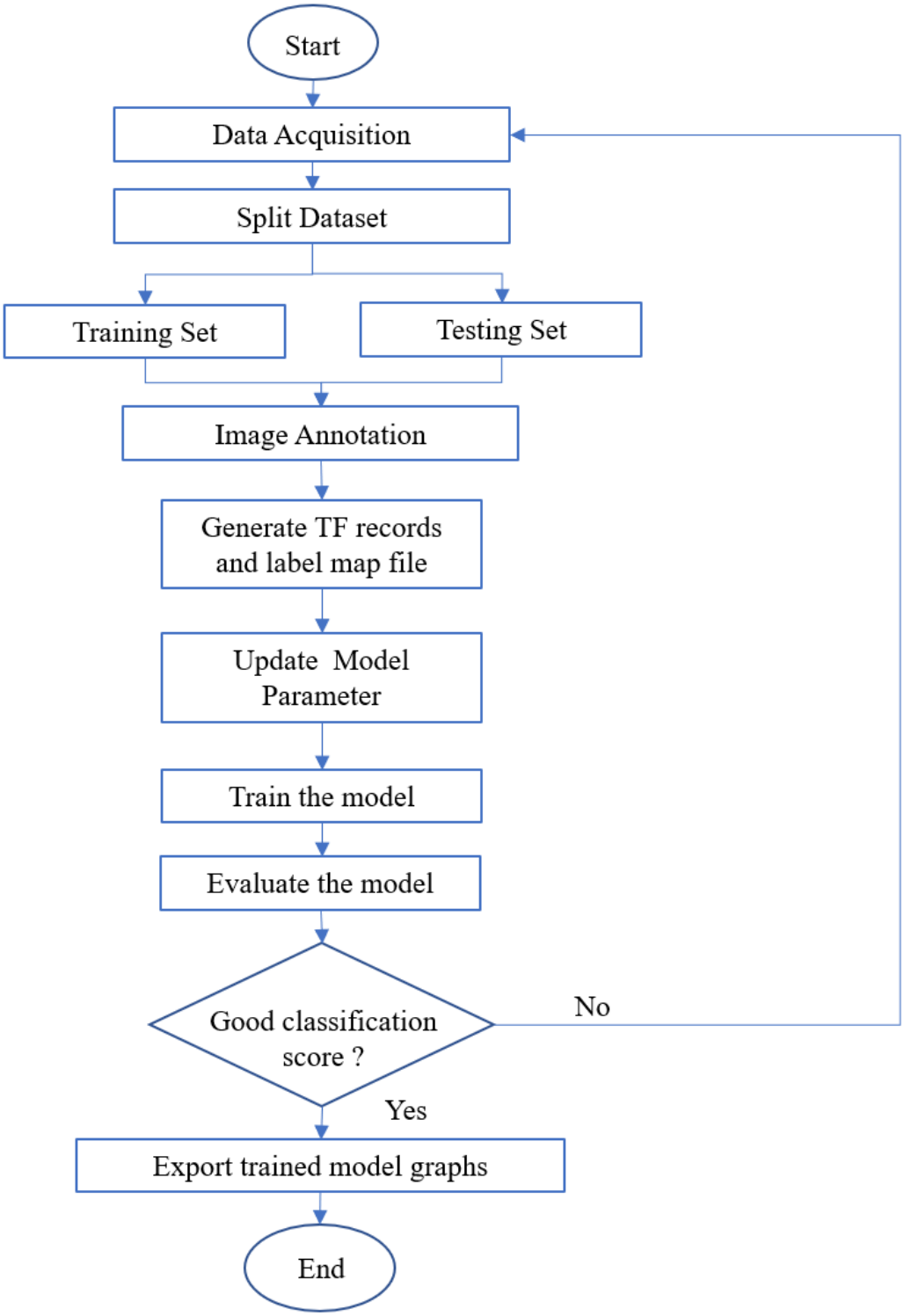
B. Proposed solution

The solution that the project proposes is the use of computer vision techniques to automate this task . The project will utilize a deep learning model called Faster R-CNN which will be trained on the required ID card dataset. The model will then be exported to utilize it on the required task of automatic ID card detection.

II. Methodology

The pretrained model that was used for performing the methods was Faster R-CNN. Faster R-CNN is proposed as a state-of-the-art model in terms of precision accuracy [1]. Since an important part of object detection is region proposal generation Faster R-CNN utilize a separate neural network for producing those region proposals [2]. This layer is called the Region proposal network or RPN. It allows the model training to be quick and also exchange weights between different convolution layer of the model. As the model had higher precision accuracy thus it was selected for the small dataset that was provided during model training since it was able to provide better detection result at small dataset.

The proposed model was built following the required steps below :



A. Data Acquisition

Images were collected using phone camera from university students where ID card was worn on the students. A total of 106 images were collected. Fig 1. shows some sample of the data that were collected.

Fig 1. Sample of the data that was collected

B. Data set build up

From the dataset that was gathered a total 53 images were selected as Training dataset and 19 images for test dataset. For selection of images and the amount of dataset to split no guideline was followed and was build up randomly.

C. Image Annotation

To provide ground truth box and class, the training and test images were manually annotated using bounding box technique and a software called “LabelImg” was utilized to produce the bounding box and its corresponding class data. The data after annotation were saved as xml file for further use. Precautions were taken when placing the class label for the bounding box as it is case sensitive . Fig 2 shows the example of image annotation result.

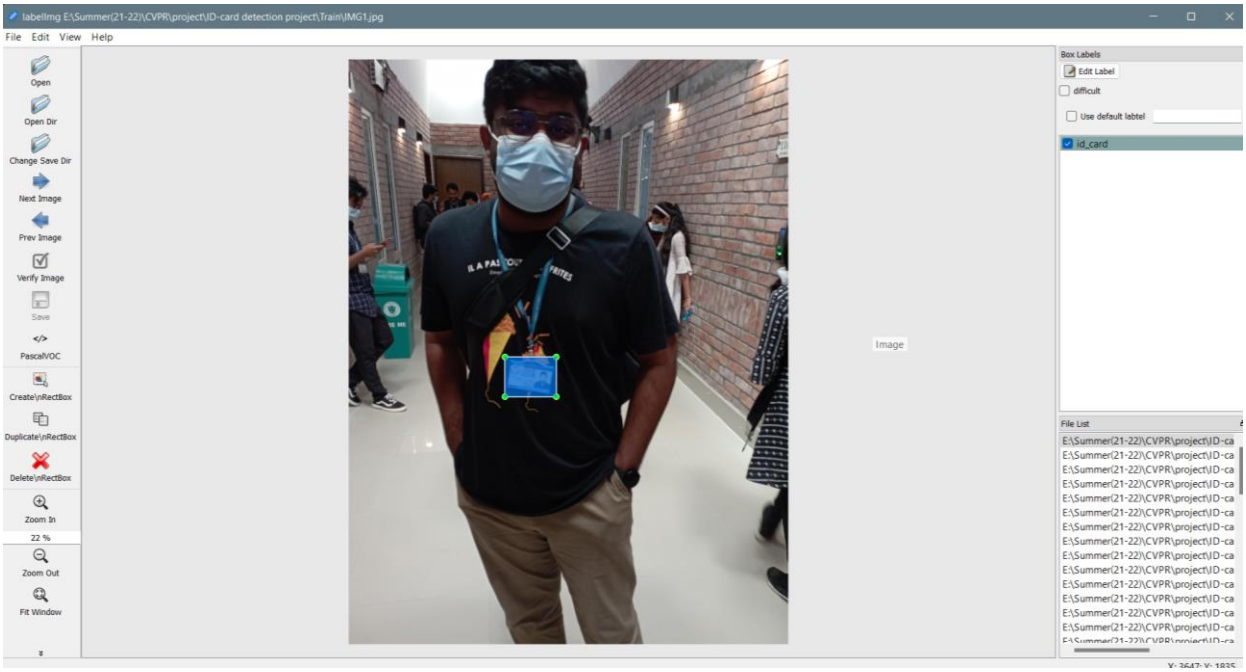


Fig 2. Annotating image with labelimg program

D. Generating TFrecords and labelmap file

The TFrecords file converts the dataset to a binary format file thus making it easier to store and be used during model training, the labelmap file will be utilized for getting the class labels of the different objects in the image. Program was written to generate all the files. The following images will give brief overview of the code that was written to generate the files

The code shown on Fig. 3 was utilized to generate the labelmap file . File reading and writing operation were performed for creating the file

```
def createLabelMap():
    label = {'name' : 'id_card','id':1}
    try:
        if not os.path.exists(paths["annotation_path"]+"\\label_map.pbtxt"):
            f = open(paths["annotation_path"]+"\\label_map.pbtxt","x");
            f.write("item {\n");
            f.write("\tid: {}\n".format(label['id']));
            f.write("\tname: \'{}\'\n".format(label['name']));
            f.write("}\n");
            f.close()
    except:
        print("file already created");
```

Fig. 3 Label map file generation code

Fig. 4 shows the code that was written to generate the record files. The split method will return a list of classes where each class contains a row where the bounding box information as well as class information are placed . Using tensorflow api a record writer instance was created and the instance was called on each class along with image data to create the record files.

```
def generateTFRecord():

    if not os.path.exists(paths['annotation_path']+"\\train.record"):
        w = tf.io.TFRecordWriter(paths['annotation_path']+"\\train.record");
        examples = pd.read_csv(paths['annotation_path']+"\\id_card_labels_train.csv");

        grouped = split(examples,'filename');

        for group in grouped:
            # gives the proto message
            example = tf_example(group,paths['train_img_path'])
            # converts the proto message to binary string
            w.write(example.SerializeToString())

        w.close()
        print("Sucessfully created train record at :", paths['annotation_path']+"\\train.record");
```

Fig. 4 Code for TF record file generation

E. Updating pretrained model parameters

The pretrained model came with its own configuration file for updating the model parameters for building a model trained on the required task and solving the problem needed. Thus, the config file was updated. Table 1. shows the parameters that were updated.

Parameters Name	Description	Previous Value	Updated Value
num_classes	Number of classes to be trained	90	1
batch_size	Number of images to be taken into memory during training	64	4
data_augmentation_options	Parameter for data augmentation	Random horizontal flip	-

optimizer	Parameter for the type of optimizer to use	Momentum optimizer	-
learning_rate	The rate in which model will update its weights for learning	0.04	-
fine_tune_checkpoint	Path for the trained model weights to be initialized	-	Path to the checkpoint : "Pre-trained_model/checkpoint/cktp-0"
fine_tune_checkpoint_type	The way model feature extractor will be initialized for the task	classification	detection

F. Training the model

After the required files were obtained and pipeline config was updated the model was trained . For training google colab was used along with tensorflow object detection api which was used from this repository . The following code will describe the process of training the model.

The file and folder created locally were uploaded to google drive for using it on colab. The following code was written to mount the drive.

```
from google.colab import drive

drive.mount('/content/drive')
```

Since the object detection api requires protocol buffer files to be compiled the compiler was installed and the required proto buffer files were compiled

```
!apt-get install protobuf-compiler python-pil python-lxml python-tk

!pip install Cython
```

```
!protoc object_detection/protos/*.proto --python_out=.
```

Finally, the object detecting api was installed and it was validated with validation script to check whether it is installed properly.

```
!python setup.py build

!python setup.py install
```

After successfully installing the api the model was trained. For training the following command was written. At different points during training, the model saved checkpoints where the model found good result. The model_dir is where the checkpoints are saved. The num_of_steps flag will provide the number of steps through which the model will be trained. For this model the number of steps was set 5000 and six checkpoints were generated throughout the training.

```
!python API_model/research/object_detection/model_main_tf2.py --
model_dir=Trained_model/my_faster_rcnn \ --
pipeline_config_path=Trained_model/my_faster_rcnn/pipeline.config --alsologtostderr --
num_train_steps=5000
```

G. Evaluating the model

After training the model was evaluated by running the evaluation script which outputted the loss metric and calssification metric and created folder for logging the information. The following command was run for evaluation.

```
!python API_model/research/object_detection/model_main_tf2.py --
model_dir=Trained_model/my_faster_rcnn \ --
pipeline_config_path=Trained_model/my_faster_rcnn/pipeline.config --
checkpoint_dir=Trained_model/my_faster_rcnn --alsologtostderr
```

H. Exporting graphs for utilizing it on real world data

Since the model needs to be utilized for real world use the trained model graph were frozen and exported . This exported graph can be utilized for running inference on the model and can also be converted to different form for usage in web and other hardware. The following command was run to produce the final output graph.

```
!python API_model/research/object_detection/exporter_main_v2.py --
input_type=image_tensor \

--pipeline_config_path=Trained_model/my_faster_rcnn/pipeline.config --
trained_checkpoint_dir=Trained_model/my_faster_rcnn \

--output_directory=new_graph
```

III. Results

After model training the model were evaluated and results were obtained. The metric that was used to evaluate were the model loss value and for classification metric precision and recall was used . The precision and recall metric were calculated instead of accuracy as the dataset was imbalanced. From the precision and recall metric the F-score value was also computed. The following section will describe the graphs of the obtained result.

The graph in Fig. 5 shows the loss value of the model training over the number of training steps. It can be seen that the loss value keeps decreasing over the number of training steps.

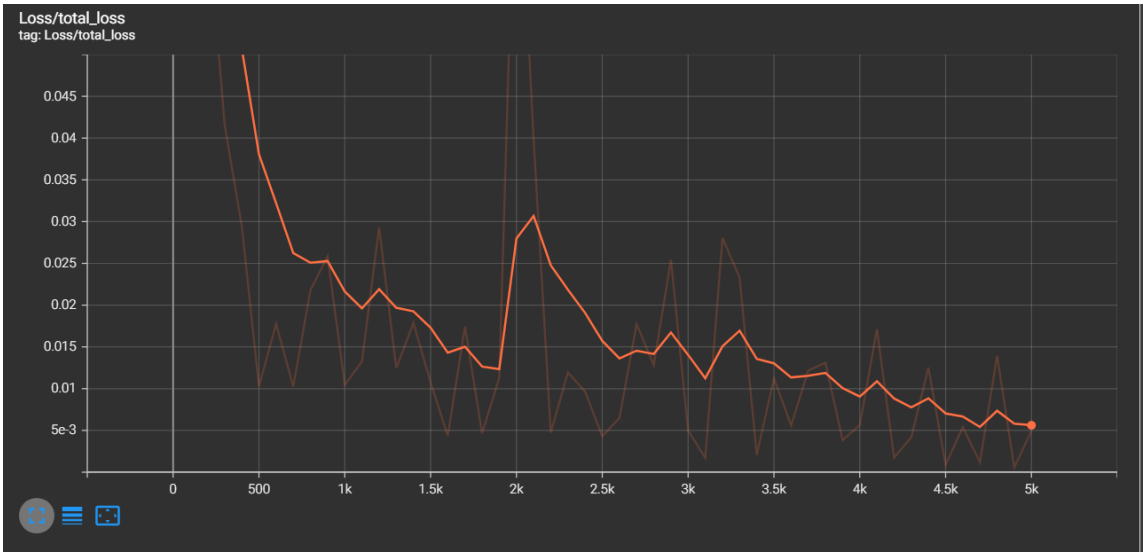


Fig. 5 Total loss curve during training (X-axis : Number of training steps , Y-axis : loss value)

The graph in Fig. 6 shows the learning rate of the model during training. As it can be observed the learning rate increases from start and goes to a stable point from 2k steps.

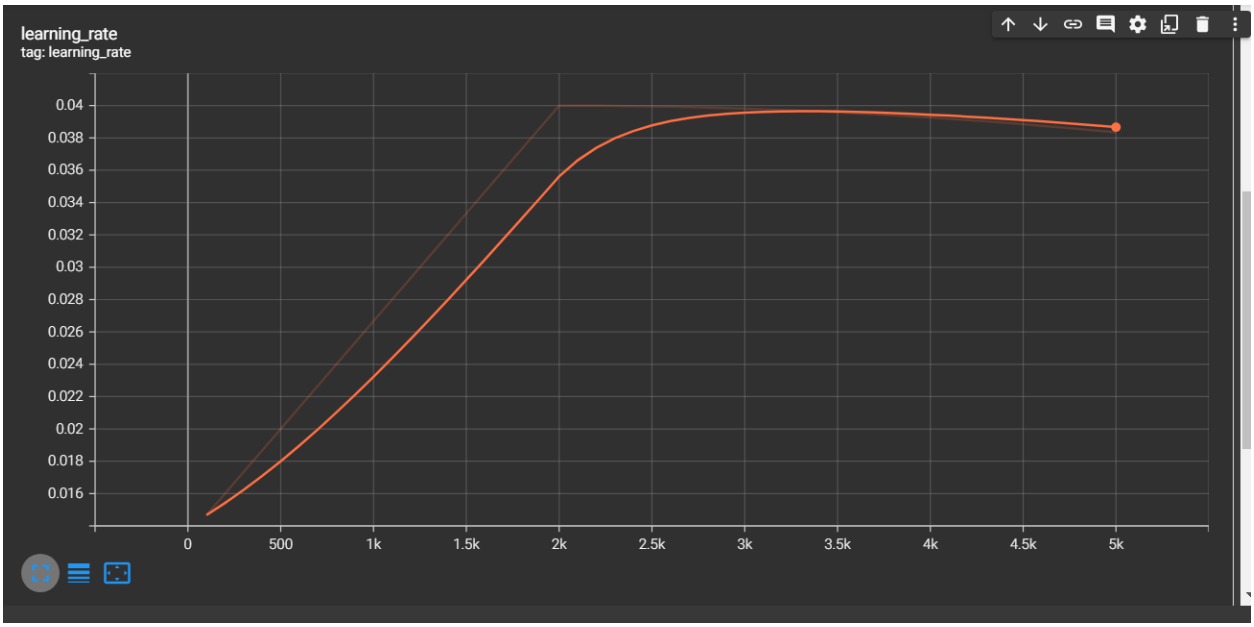


Fig. 6 Learning rate curve during training (X-axis : Number of training steps , Y-axis : learning rate)

The graph in Fig. 7 shows the average precision metric value after training of 5k steps . It can be observed that the precision value obtained was around 0.836 after 5k steps.

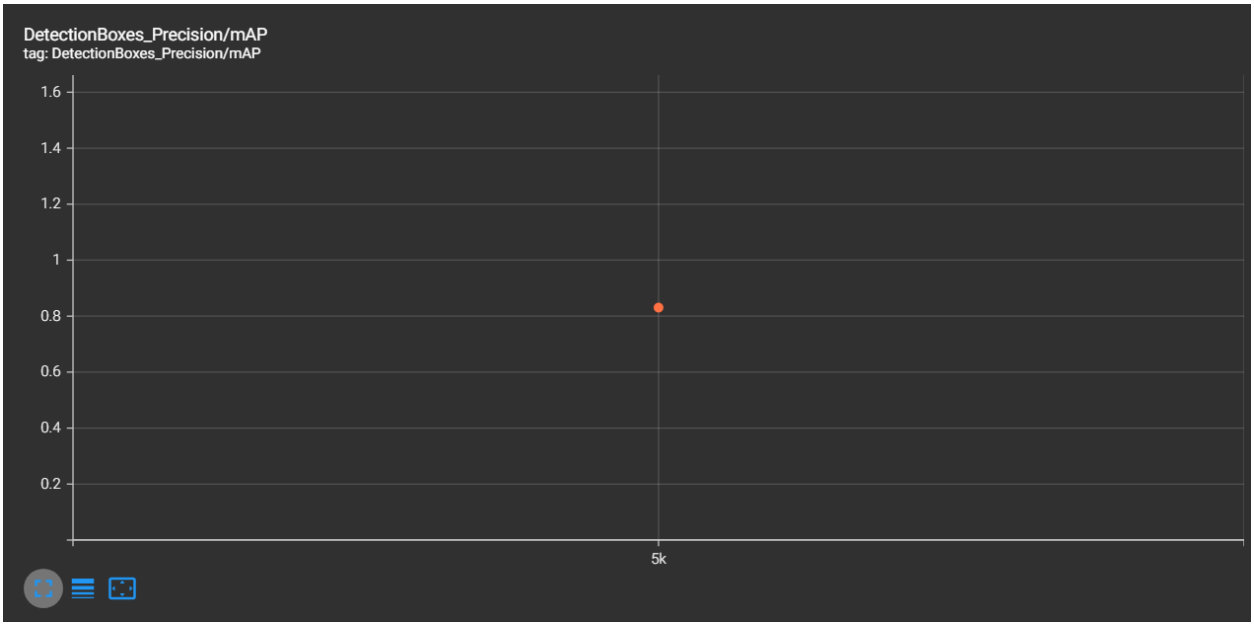


Fig. 7 Avg. Precision value plot after 5k step

The graph in Fig. 8 shows the average recall value after 5k training steps. The average recall value obtained was 0.852 after 5k steps

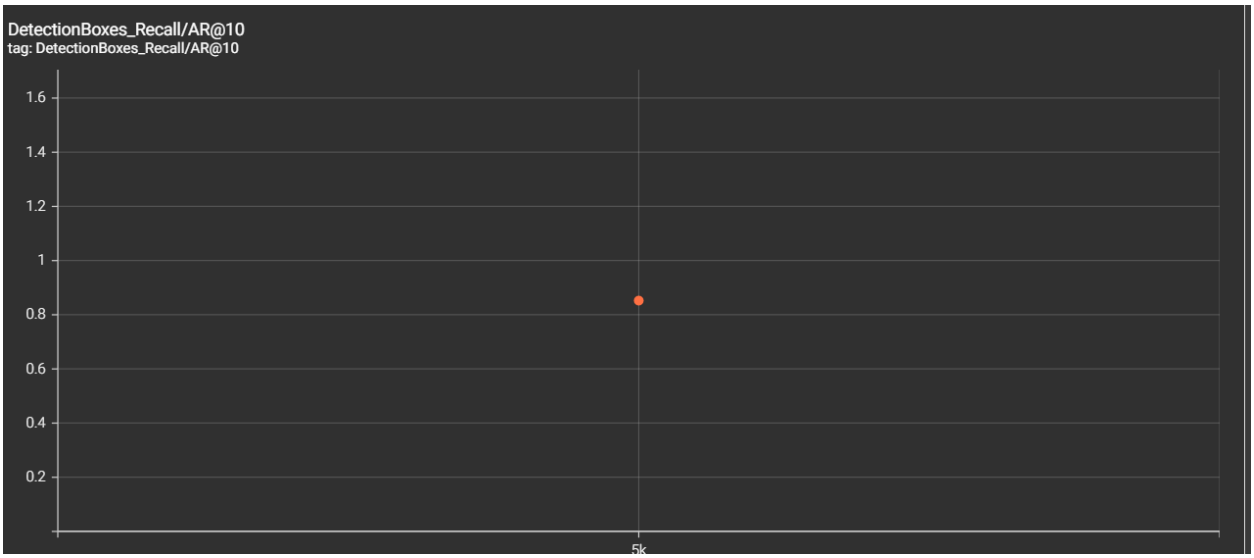


Fig. 8 Avg. Recall value plot after 5k step

From the precision and recall metric obtained the final F-score was calculated . The value obtained from F-score was 0.84.

- **F-score** = $\frac{2 * precision * recall}{precision + recall} = \frac{2 * 0.831 * 0.852}{0.831 + 0.852} = 0.841$

The final model after exporting inference was ran on the model using some test images . Fig. 9 shows a sample of the images.

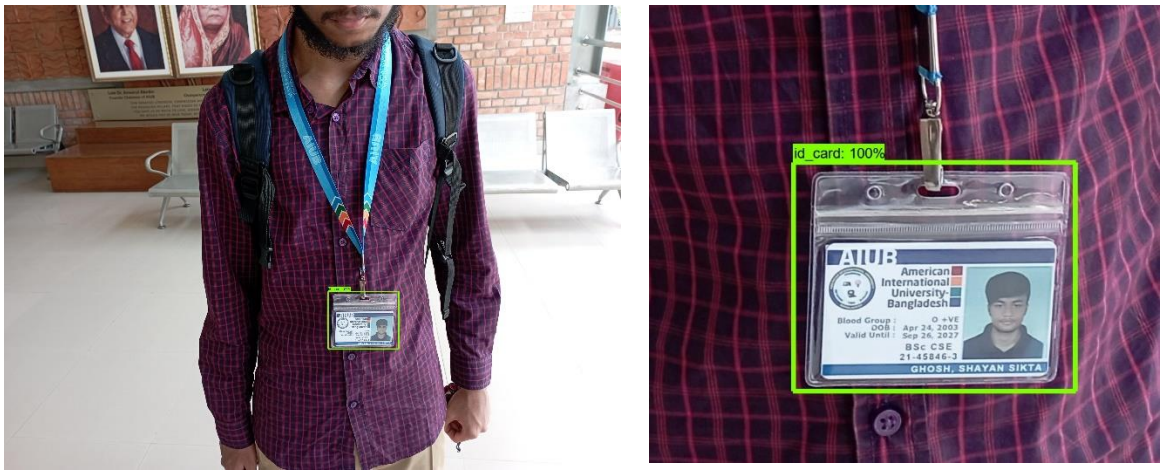


Fig. 9 Example of detection result on test images

IV. Discussion And Conclusion

From the obtained results it was observed that the model gave an overall high score. From the test image for detecting the object the model showed it was also able to detect the object in the images correctly. However, the current model does have some limitation and needs to be improved. The current limitation and future improvement are discussed below.

A. Current Limitation

The limitation of the current model include :

1. The model is only able to detect whether ID card is in the image or not and cannot distinguish different types of ID card so users can put a fake id card and it will detect it.
2. The training graph shown was very unstable.
3. Faster R-CNN tends to be slow for real time detection.
4. Training on small dataset.

B. Future improvement

Possible future improvements that can be done :

1. Training the model on two or more different types of ID card to distinguish between different ID cards.
2. Training the model to extract information from ID card and saving it on database for automatic attendance.
3. Training on models such as YOLO or ssd that give better performance for realtime detection.
4. Using more data augmentation technique for obtaining larger dataset.
5. Two hyperparameter can be tuned which are learning rate and the optimizer so that the training is more stable and give better optimal trained weights

Even though the current model has some limitation the current work will act as a starting point and guideline for improving it in the future. It is expected that the current work after applying the improvements it will be able to be used wide scale for automatically detecting ID card and replace the manual labor needed for this task.

References

[1] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2017. Available: 10.1109/tpami.2016.2577031.

[2] A. Fawzy Gad, "Faster R-CNN Explained for Object Detection Tasks | Paperspace Blog", *Paperspace Blog*, 2019. [Online]. Available: https://blog.paperspace.com/faster-r-cnn-explained-object-detection/. [Accessed: 09- Aug- 2022].