CSE4019 – Image Processing

Project Report

Document Verification

By

19BCE1454	Shailja
19BCE1402	Aditya
19BCE1642	Pankaj

B. Tech Computer Science and Engineering

Submitted to

GEETHA S

School of Computer Science and Engineering

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

April 2022

DECLARATION

I hereby declare that the report titled **Document Verification** submitted by me to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of **Dr. GEETHA S**, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my professor, Dr. GEETHA S, for contributing their valuable time and efforts in helping me out with this project. Their suggestions and feedback have helped me a lot in improving the quality of the project.

I would also like to thank my friends and family for their constant encouragement and support throughout the project.

Lastly, I would like to thank all our supporters who have motivated me to fulfill this project before the timeline.

CONTENTS

1. Introduction

Showing any kind of proof of identity is vital to complete any kind of legal formality or verification process. No matter what industry a customer is trying to become a part of, be it banking, insurance, healthcare, technology, travel, education, or any other online service, the customer is asked and obligated to present an Identity proof to verify a customer is who they say they are. Verifying customer identities can reduce the risk of identity theft.
Verifying identities may sound great, but as a business, you'll have to spend money, time, and HR to put secure document verification solutions in place. Extra tough verification process reduces the overall customer experience. While most customers don't like the extra tough verification process, not having a verification process reduces the trust in the brand and can cause businesses to lose more customers during the onboarding process.
To find the balance between these situations, AI-based digital document verification solutions are viable, effective, and hassle-proof.

1.1 Objective and goal of the project

The objective of this project is to attempt to inject a bit of automation in the process of document verification which in real life is a pretty cumbersome process.Just to set the wheels of the system in motion,the document we will be looking at is our college ID Card.The larger objective of this system is to be able to verify official documents such as Aadhar Card,Pan Card etc.The objective of this system is to not only to verify a document but to verify the CORRECT document .

1.2 Problem Statement

We all have come across the official procedure for renewing a passport in India. We are required to upload a soft copy of the documents and then show the same documents in the form of a hard copy to the passport office. The same procedure is more or less followed for getting KYC(Know your customer) done. All this is a pretty cumbersome process. In our project, we wanted to inject a bit of automation into this process. Since most verifications in India require the Aadhar card, our system was originally targeted towards the Aadhar Card. Due to time and computational constraints, we decided to create a system which can first verify VIT Chennai ID Cards and then maybe as an expansion can then verify official government documents like PAN,Aadhar Card

1.3 Challenges

One problem we faced that the object detection model couldn't take multiple evaluation images as input and we had to manually feed different images. As an extension of the problem just mentioned, we had to delete the verified image after one iteration of the system so that for the subsequent iterations, the directory where the verified ID cards are stored is empty.

2. Literature Survey

Author	Name	Methodology	Result
CRISTIAN WISULTSCHEW 1 , GABRIEL MUJICA 1 , (Member, IEEE), JOSE MANUEL LANZA-GUTIERREZ 2 , AND JORGE PORTILLA 1 ,	3D-LIDAR Based Object Detection and Tracking on the Edge of IoT for Railway Level Crossing	An edge IoT HW platform implementation capable of detecting and tracking objects in a railway level crossing scenario is proposed. The response of the system has to be calculated and sent from the proposed IoT platform to the train, so as to trigger a warning action to avoid a possible collision. The system uses a low-resolution 3D 16-channel LIDAR as a sensor that provides an accurate point cloud map with a large amount of data. The element used to process the information is a custom embedded edge platform with low computing resources and low-power consumption. This processing element is located as close as possible to the sensor, where data is generated to improve latency, privacy, and avoid bandwidth limitations, compared to performing processing in the cloud. Additionally,	In this work, real-time and accurate object detection and tracking implementation using raw 3D point cloud data provided by a low-resolution LIDAR is presented. High accuracy in the detected object spatial location is achieved by using this complex sensor. Besides, other improvements are obtained compared to object detection methods based on RGB cameras, since LIDARs work fine in the absence of light and are more robust in adverse weather conditions.

		lightweight object detection and tracking algorithm is proposed in this work to process a large amount of information provided by the LIDAR, allowing to reach real-time specifications.	
Diandian Zhang,1 Yan Liu,1 Zhuowei Wang,2 and Depei Wang	OCR with the Deep CNN Model for Ligature Script-Based Languages like Manchu	In the given work, deep CNN model is used to recognize the text and then build a Manchu recognition system. ,deep CNN model uses four convolution layers to mine different image features. In the Manchu recognition system, the sliding window method is used to identify the same characters in the database	Traditional CNNs require input images of a consistent size. Manchu is a phonetic text, and its word length is not fixed. Preprocessing to ensure the uniform size is therefore required before recognition and classification, and such preprocessing reduces the recognition rate. To reduce the effect of image normalization preprocessing on the recognition rate, this paper improves the traditional CNN and constructs a new non segmented Manchu word recognition network model: deep CNN
XIN ZHANG 1 , LIANGXIU HAN 1 , MARK ROBINSON2 , AND ANTHONY GALLAGHER3	A Gans-Based Deep Learning Framework for Automatic Subsurface Object Recognition From Ground Penetrating Radar Data	In the given work ,a Generative Adversarial Nets (GANs)-based deep learning framework, which generates new training data to address the scarcity of GPR data, automatically learns features and detects subsurface objects (via hyperbola) through an end-to-end solution is proposed	The proposed approach has been evaluated with real data and has been compared with the state-of-the-art deep learning methods for object detection (i.e. Faster-RCNN, Cascade R-CNN, SSD and YOLO V2). The experimental results show that the proposed method outperforms the existing methods and achieved high accuracy of 97% for the mAP. Meanwhile, our proposed model shows good generalizability by a

			cross-validation on independent datasets
QIUYING HUANG, ZHANCHUAN CAI , (Senior Member, IEEE), AND TING LAN	A Single Neural Network for Mixed Style License Plate Detection and Recognition	This article proposes a single neural network called ALPRNet for detection and recognition of mixed style LPs. In ALPRNet, two fully convolutional one-stage object detectors are used to detect and classify LPs and characters simultaneously, which are followed by an assembly module to output the LP strings. ALPRNet treats LP and character equally, object detectors directly output bounding boxes of LPs and characters with corresponding labels, so they avoid the recurrent neural network (RNN) branches of optical character recognition (OCR) of the existing recognition approaches	In the experiments, ALPRNet achieves 98.21% accuracy rate on the HZM multi-style dataset, and the results on the datasets with single LP style also show that the proposed network achieves state-of-the-art recognition accuracy
HASSANIN M. AL-BARHAMTOSHY 1 , (Fellow, IEEE), KAMAL M. JAMBI 2 , SHERIF M. ABDOU3 , AND MOHSEN A. RASHWAN4	Arabic Documents Information Retrieval for Printed, Handwritten, and Calligraphy Image	This paper presents a new computational backend model that supports Arabic document information retrieval (ADIR) as a dataset and OCR servicesthe proposed work can provide accessing different methods of document layout analysis with a platform where they can share and handle such methods (services) without any setup requirements. One of the used datasets composed from 16,800 Arabic letters written by 60 writers. Each writer wrote each letter from	This paper combined two approaches, minimize OCR errors and acquire text inquiry for IR. The two approaches are tested, evaluated and judged tested in three different experimental domains. Some difficulties such as understanding the concept and the meaning of scanned images and related definitions

		Alif to Ya 10 times in two forms. The forms were scanned at 300 DPI resolution and are segmented in two sets: training set with 13,440 letters for 48 images per class label, and testing set with 3,360 letters to 120 images per class label Convolutional neural network (CNN) is used and adapted for Arabic handwritten letters classification	
FAHAD ASHIQ 1 , MUHAMMAD ASIF 1 , MAAZ BIN AHMAD2 , SADIA ZAFAR1 , KHALID MASOOD1 , TOQEER MAHMOOD 3 , MUHAMMAD TARIQ MAHMOOD 4 , (Senior Member, IEEE), AND IK HYUN LEE	CNN-Based Object Recognition and Tracking System to Assist Visually Impaired People	The application uses MobileNet architecture due to its low computational complexity to run on low-power end devices. To assess the efficacy of the proposed system, six pilot studies have been performed that reflected satisfactory results. For object detection and recognition, a deep Convolution Neural Network (CNN) model is employed with an accuracy of 83.3%, whereas the dataset contains more than 1000 categories	This paper presented a smart and intelligent system for VIPs to assist them in mobility and ensure their safety. The proposed system is based on the day-to-day requirements of VIPs. It assists them in visualizing the environment and providing a sense of the surroundings. They can recognize objects around them and sense the natural environment using CNN-based low-power Mobile-Net architecture. Moreover, a web-based application is developed to ensure the safety of VIPs.
ALEKSANDAR JEVREMOVIC1 , MLADEN VEINOVIC 1 , MILAN CABARKAPA2 , MARKO KRSTIC2 , (Member, IEEE), IVAN CHORBEV 3 , IVICA DIMITROVSKI3 , NUNO GARCIA 4 , NUNO POMBO4 , (Senior Member, IEEE), AND MILOS	Keeping Children Safe Online With Limited Resources: Analyzing What is Seen and Heard	A highly modular framework of analyzing content in its final form at the user interface, or Human Computer Interaction (HCI) layer, as it appears before the child: on the screen and through the speakers is proposed. Our approach is to produce Children's Agents for Secure and Privacy	The framework is a comprehensive, modular method of identifying objectionable online content. It performs well given limited hardware resources and a limited number of categories of harmful content to detect. Expanding its capability to detect every possible type of objectionable content

STOJMENOVIC 1		Enhanced Reaction (CASPER), which analyzes screen captures and audio signals in real time in order to make a decision based on all of the information at its disposal, with limited hardware capabilities. We employ a collection of deep learning techniques for image, audio and text processing in order to categorize visual content as pornographic or neutral, and textual content as cyberbullying or neutral.	for children would entail extensive use of the GPU in most circumstances, but is not outside the scope of capabilities of today's state of the art machine learning software.
TAYYAB NASIR 1 , MUHAMMAD KAMRAN MALIK 2 , AND KHURRAM SHAHZAD 3	MMU-OCR-21: Towards End-to-End Urdu Text Recognition Using Deep Learning	This paper has proposed a very large Multi-level and Multi-script Urdu corpus (MMU-OCR-21). It is the largest-ever Urdu corpus of printed text that is effectively suitable to work with deep learning techniques. In total, the corpus is composed of over 602,472 images, including text-line and word images in three prominent fonts, and their respective ground truth. Also, we have performed experiments using multiple state-of-the-art deep learning techniques for text-line and word level images.	The study concludes that most of the deep learning models used for experimentation generalized well on our developed corpus and they were able to achieve remarkable CER and WER scores. We also analyzed the performance of our models for individual fonts to establish that the models were effectively generalized for all the fonts
Hae Gwang Park, Jong Pil Yun , Min Young Kim , Member, IEEE, and Seung Hyun Jeong	Multichannel Object Detection for Detecting Suspected Trees With Pine Wilt Disease Using Multispectral Drone Imagery	a multichannel convolutional neural network (CNN) based object detection was used to detect suspected trees of pine wilt disease after	The proposed model detects suspected trees of PWD after training based on multispectral aerial photography in RGB, green, red, NIR, and red edge bands

		acquiring aerial photographs through a rotorcraft drone equipped with a multispectral camera. The acquired multispectral aerial photographs consist of RGB, green, red, NIR, and red edge spectral bands per shooting point. The aerial photographs for each band performed image calibration to correct radiation distortion, image alignment to correct the distance error of the lenses of a multispectral camera, and image enhancement to edge enhancement to highlight the features of objects in the image. After that, a large amount of data obtained through data augmentation were put into multichannel CNN-based object detection for training and test. A	acquired through a rotary-wing drone equipped with a multispectral camera. In addition, in order to improve detection accuracy, image preprocessing was performed through image calibration, image alignment, and image enhancement. As a result of the study, the multichannel CNN-based object detection model using RGB, NIR, red edge, and NDRE index was the best, and the final detection performance was mAP 86.63%
SHASHA LI 1 , YONGJUN LI 1 , YAO LI 1 , MENGJUN LI 1 , AND XIAORONG XU 2	YOLO-FIRI: Improved YOLOv5 for Infrared Image Object Detection	a region-free object detector named YOLO-FIR for infrared (IR) images with YOLOv5 core by compressing channels, optimizing parameters, etc is proposed. An improved infrared image object detection network, YOLO-FIRI, is further developed. Specifically, while designing the feature extraction network, the cross-stage-partial-connections (CSP) module in the shallow layer is expanded and iterated to maximize the use of shallow features. In addition, an improved attention	The mAP of YOLO-FIRI is increased by approximately 37% on the infrared images of KAIST, the detection time is reduced by approximately 62%, the network parameters are reduced by more than 89%, and the weight size is reduced by more than 93%. Compared with YOLO-FIR, the mAP of YOLO-FIRI reaches 98.3% and increases approximately 13% on KAIST. The AP for the bicycle class of YOLO-FIRI on FLIR also reaches 85%,

		module is introduced in residual blocks to focus on objects and suppress background. Moreover, multiscale detection is added to improve small object detection accuracy	which is an increase of 15%.

3 Requirements Specification

3.1 Hardware Requirements

1. Laptop with any Operating System(Windows,MacOS etc)
2. 8GB Ram
3. NVIDIA GTX 650+ Graphic Card

3.2 Software Requirements

1)TensorFlow GPU

2)NVIDIA CUDA

3)NVIDIA CUDA Deep Neural Network Library(cuDNN)

4)Visual Studio Build Tools(atleast 2014 version)

Note:-The versions of CUDA and cuDNN have to be compatible with tensorflow GPU for the object detection api to work properly

4 Implementation of System

METHODOLOGY

STEP 1: Dataset

STEP 2: Extracting Region of Interest

STEP 3: Training a Object Detection Model on ID Card

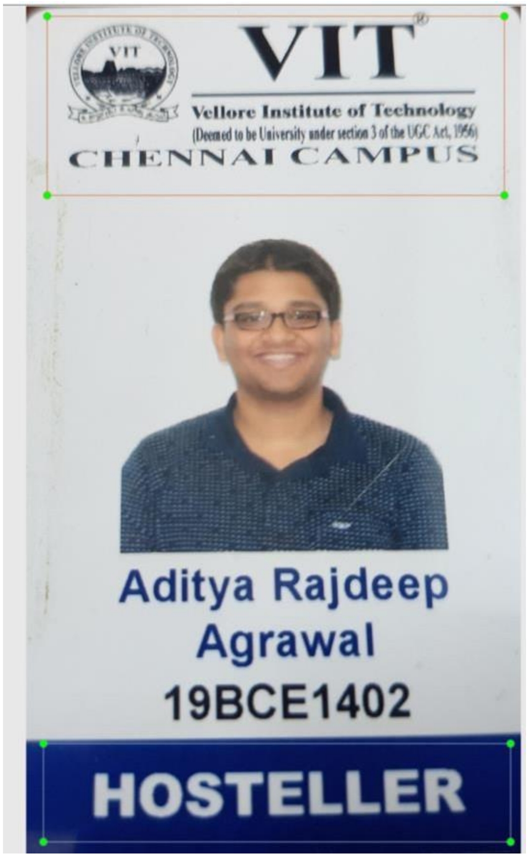
STEP 4: Extracting Data using OCR

STEP 5: Validation of Data extracted using OCR

STEP 1: Dataset

STEP 2: Extracting Region of Interest

After extracting the Region of Interest, feature extraction, and object recognition technology will be applied, and features specific to the ID card will be detected and marked.



In the above image, the green boxes indicate the features based on which the verification of the ID Card will be done

First green box-College Logo

Second green box-Residence Status

STEP 3: Training a Object Detection Model

Training Custom Object Detector

After installing the requisite software and having the ideal hardware configuration, we will proceed with the following steps:

1. How to organise your workspace/training files
2. How to prepare/annotate image datasets
3. How to generate tf records from such datasets
4. How to configure a simple training pipeline
5. How to train a model and monitor it's progress
6. How to export the resulting model and use it to detect objects.

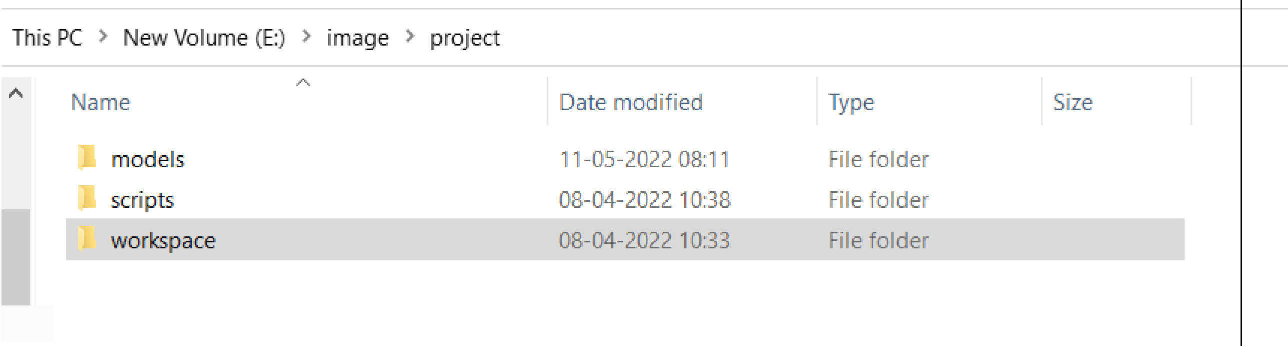
Preparing the Workspace

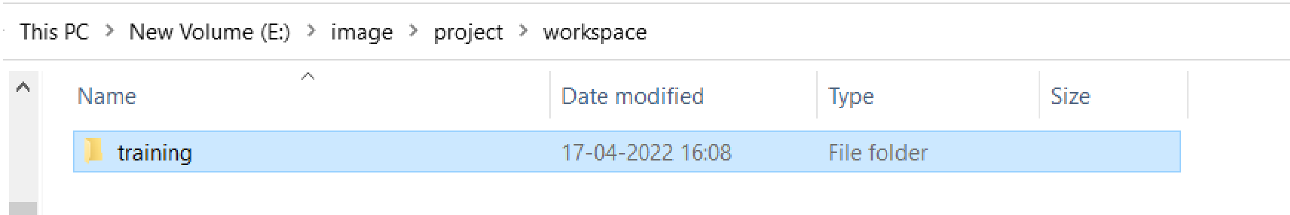
1. We shall begin by creating the following directory structure. The image given below shows the general template. In our case the name of the root directory is project

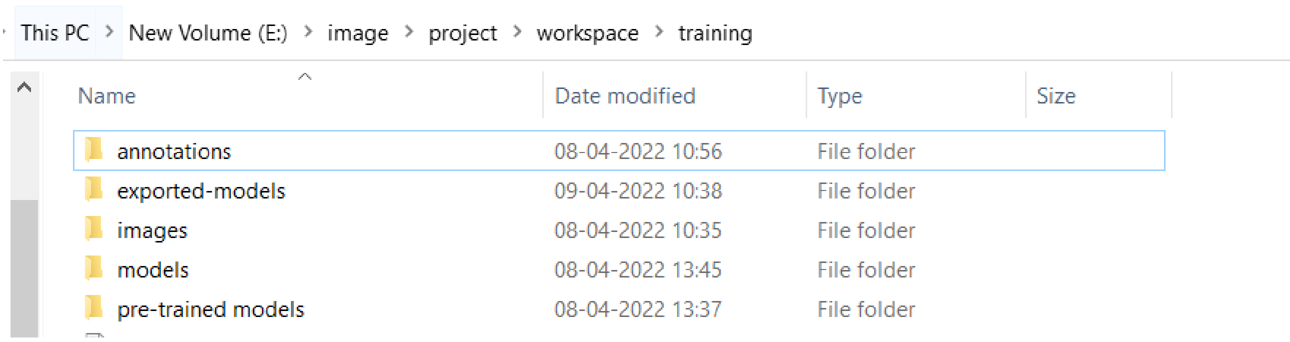
```
2.  TensorFlow/
3.   ├─ addons/ (Optional)
4.   │   └─ labelImg/
5.   └─ models/
6.      ├─ community/
7.      ├─ official/
8.      ├─ orbit/
9.      ├─ research/
10.     └─ ...
```

11. Now create a new folder under `TensorFlow` (project) and call it `workspace`. It is within the `workspace` that we will store all our training set-ups. Now let's go under workspace and create another folder named `training`. Now our directory structure should be as so:

23. The `training_demo` folder shall be our *training folder*, which will contain all files related to our model training. It is advisable to create a separate training folder each time we wish to train on a different dataset. The typical structure for training folders is shown below.



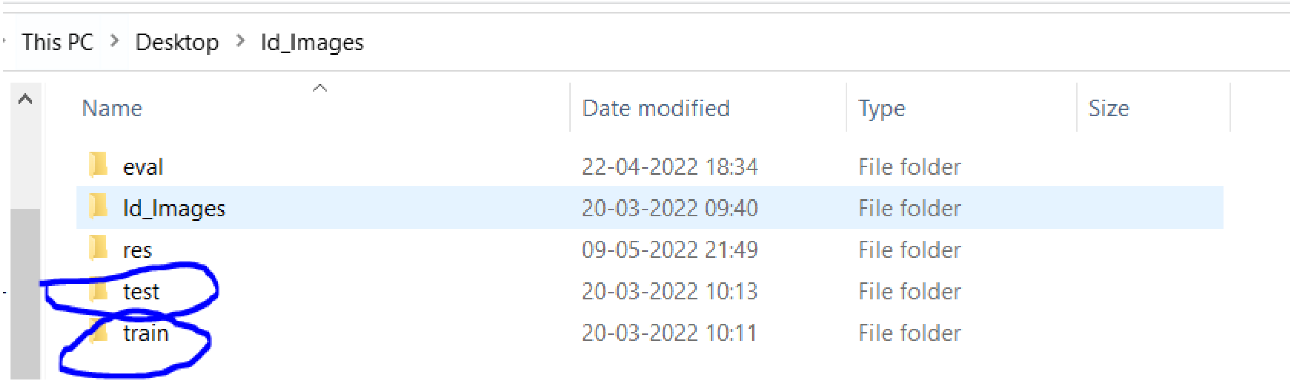




Here's an explanation for each of the folders/filer shown in the above tree:

- `annotations` : This folder will be used to store all `*.csv` files and the respective TensorFlow `*.record` files, which contain the list of annotations for our dataset images.
- `exported-models` : This folder will be used to store exported versions of our trained model(s).
- `images` : In an ideal scenario, This folder contains a copy of all the images in our respective `*.xml` files produced for each one, once `labelImg` is used to annotate objects.
    - : This folder contains a copy of all images, and the respective files, which will be used to train our model.
    - `images/test` : This folder contains a copy of all images, and the respective `*.xml` files, which will be used to test our model.

For our project, since we did not have a lot of images to train,we created the test and training images separately



- `models` : This folder will contain a sub-folder for each of training job. Each subfolder will contain the training pipeline configuration file `*.config` , as well as all files generated during the training and evaluation of our model.
- `pre-trained-models` : This folder will contain the downloaded pre-trained models, which shall be used as a starting checkpoint for our training jobs.
- `README.md` : This is an optional file which provides some general information regarding the training conditions of our model. It is not used by TensorFlow in any way, but it generally helps when you have a few training folders and/or you are revisiting a trained model after some time.

how all the files are generated further down.

Preparing the Dataset

Annotate the Dataset

Install LabelImg

There exist several ways to install `labelImg` . Below are 3 of the most common.

Using PIP (Recommended)

1. Open a new *Terminal* window and activate the *tensorflow_gpu* environment (if you have not done so already)
2. Run the following command to install `labelImg`:

```
pip install labelImg
```

1. `labelImg` can then be run as follows:

```
labelImg
# or
labelImg [IMAGE_PATH] [PRE-DEFINED CLASS FILE]
```

Annotate Images

- Once you have collected all the images to be used to test your model (ideally more than 100 per class) but in our case we gave around 40 images,

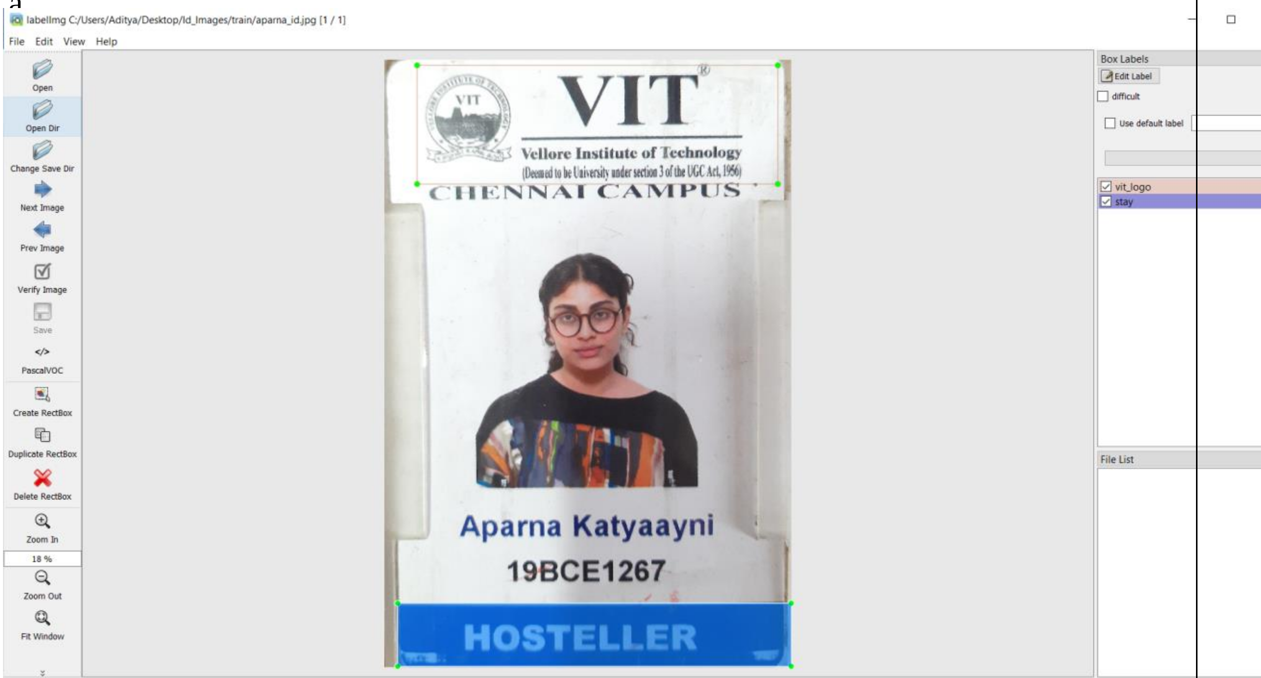
- Open a new *Terminal* window.

- N e x t g o a b e a by typing only labelimg in the command prompt

  - A File Explorer Dialog windows should open, which points to the spot where training images are stored.
    In our case it is the C:\Users\Aditya\Desktop\Id_Images\train folder

  - Press the "Select Folder" button, to start annotating your images.

Once open, you should see a window similar to the one below:



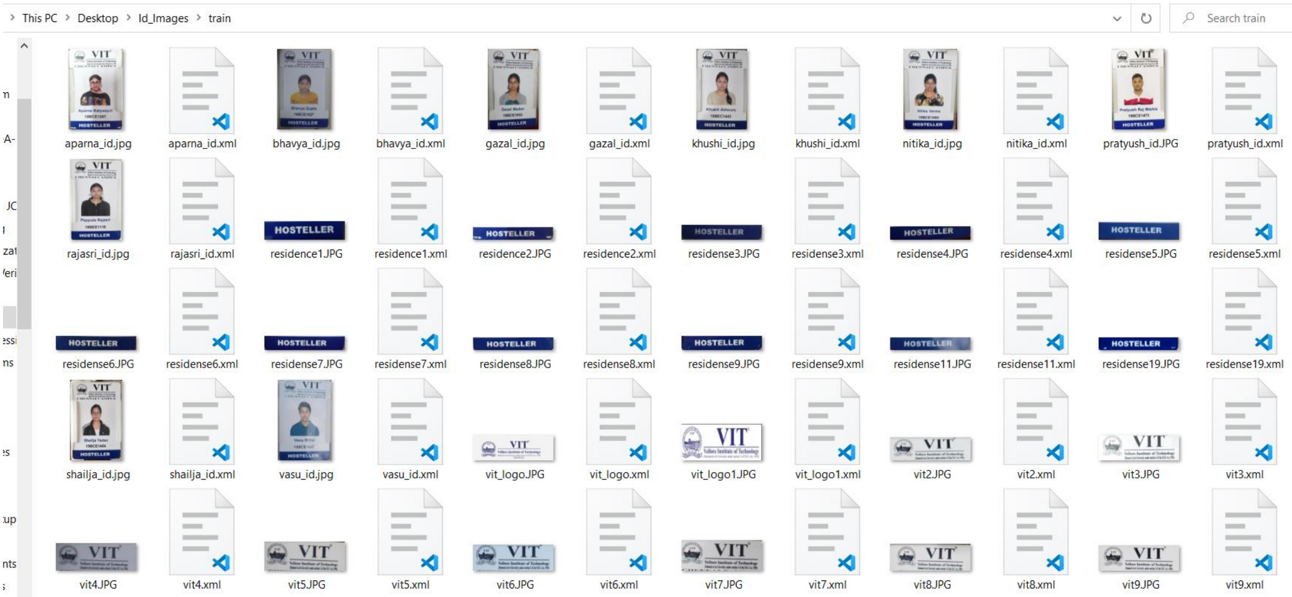
m g

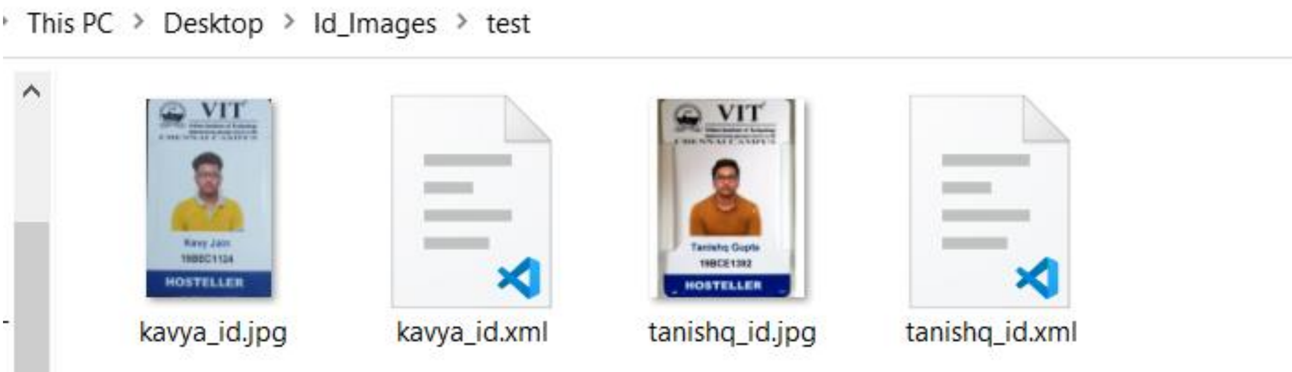
Partition the Dataset

Once you have finished annotating your image dataset, it is a general convention to use only part of it for training, and the rest is used for evaluation purposes (e.g. as discussed in Evaluating the Model (Optional)).

Typically, the ratio is 9:1, i.e. 90% of the images are used for training and the rest 10% is maintained for testing, but you can chose whatever ratio suits your needs.

Once you have decided how you will be splitting your dataset, copy all training images, together with their corresponding `*.xml` files, and place them inside the `training_demo/images/train` folder. Similarly, copy all testing images, with their `*.xml` files, and paste them inside training_demo/images/test .





Download the requisite scripts and put it in the E:\image\project\workspace\training folder.

To run the scripts cd into the above given directory and type python [Script_name]

Create Label Map

TensorFlow requires a label map, which namely maps each of the used labels to an integer values. This label map is used both by the training and detection processes.

Below we show an example label map (e.g label_map.pbtxt), assuming that our dataset containes 2 labels

```
item {
   id: 1
   name: 'vit_logo'
}

item {
   id: 2
   name: 'stay'
}
```

Label map files have the extention .pbtxt and should be placed inside the training_/annotations folder.

Create TensorFlow Records

Now that we have generated our annotations and split our dataset into the desired training and testing subsets, it is time to convert our annotations into the so called TFRecord format.

Convert `*.xml` to `*.record`

To do this we can write a simple script that iterates through all `*.xml` files in the `training_demo/images/train` and `training_demo/images/test` folders, and generates a `*.record` file for each of the two. This script comes with the tensorflow models we earlier cloned from github

- Install the `pandas` package:
  - *Pip install pandas*
- Finally, `cd` into `training/scripts/preprocessing` and run:
  - *# Create train data:*
  - python generate_tfrecord.py -x [PATH_TO_IMAGES_FOLDER]/train -l [PATH_TO_ANNOTATIONS_FOLDER]/label_map.pbtxt -o [PATH_TO_ANNOTATIONS_FOLDER]/train.record

-
- *# Create test data:*
- python generate_tfrecord.py -x [PATH_TO_IMAGES_FOLDER]/test -l [PATH_TO_ANNOTATIONS_FOLDER]/label_map.pbtxt -o [PATH_TO_ANNOTATIONS_FOLDER]/test.record
-
- *# For example*
- *# python generate_tfrecord.py -x C:/Users/sglvladi/Documents/Tensorflow/workspace/training_demo/images/train -l C:/Users/sglvladi/Documents/Tensorflow/workspace/training_demo/annotations/label_map.pbtxt -o C:/Users/sglvladi/Documents/Tensorflow/workspace/training_demo/annotations/train.record*
- *# python generate_tfrecord.py -x C:/Users/sglvladi/Documents/Tensorflow/workspace/training_demo/images/test -l C:/Users/sglvladi/Documents/Tensorflow2/workspace/training_demo/annotations/label_map.pbtxt -o C:/Users/sglvladi/Documents/Tensorflow/workspace/training_demo/annotations/test.record*

Once the above is done, there should be 2 new files under the `training_demo/annotations` folder, named `test.record` and `train.record`, respectively.

Configuring a Training Job

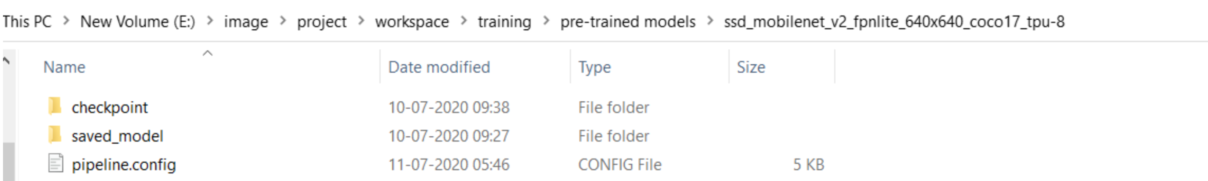
For the purposes of this tutorial we will not be creating a training job from scratch, but rather we will reuse one of the pre-trained models provided by TensorFlow. If you would like to train an entirely new model, you can have a look at TensorFlow's tutorial.

The model we shall be using in our examples is the SSD ResNet50 V1 FPN 640x640 model, since it provides a relatively good trade-off between performance and speed. However, there exist a number of other models you can use, all of which are listed in TensorFlow 2 Detection Model Zoo.

Download Pre-Trained Model

To begin with, we need to download the latest pre-trained network for the model we wish to use. This can be done by simply clicking on the name of the desired model in the table found in TensorFlow 2 Detection Model Zoo. Clicking on the name of your model should initiate a download for a `*.tar.gz` file.

Once the `*.tar.gz` file has been downloaded, open it using a decompression program of your choice (e.g. 7zip, WinZIP, etc.). Next, open the `*.tar` folder that you see when the compressed folder is opened, and extract its contents inside the folder `training/pre-trained-models`. Since we downloaded the ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8model, our `training` directory should now look as follows:



Note that the above process can be repeated for all other pre-trained models you wish to

Configure the Training Pipeline

Now that we have downloaded and extracted our pre-trained model, let's create a directory for our training job. Under the training_demo/models create a new directory

named my_model and copy the training_demo/pre-trained- model/pipeline.config

Our training/models directory should now look like this:

```
training/
├─ ...
├─ models/
│   └─ my_model/
│       └─ pipeline.config
└─ ...
```

Now, let's have a look at the changes that we shall need to apply to the newly added pipeline.config file
 (highlighted in yellow):

```
model {
 ssd {
   num_classes: 1 # Set this to the number of different label classes
   image_resizer {
     fixed_shape_resizer {
       height: 640
       width: 640
     }
   }
   feature_extractor {
     type: "ssd_resnet50_v1_fpn_keras"
     depth_multiplier: 1.0
     min_depth: 16
     conv_hyperparams {
       regularizer {
```

```
      l2_regularizer {
        weight: 0.00039999998989515007
      }
    }
    initializer {
      truncated_normal_initializer {
        mean: 0.0
        stddev: 0.029999999329447746
      }
    }
    activation: RELU_6
    batch_norm {
      decay: 0.996999979019165
      scale: true
      epsilon: 0.0010000000474974513
    }
  }
  override_base_feature_extractor_hyperparams: true
  fpn {
    min_level: 3
    max_level: 7
  }
}
box_coder {
  faster_rcnn_box_coder {
    y_scale: 10.0
    x_scale: 10.0
    height_scale: 5.0
    width_scale: 5.0
  }
}
matcher {
  argmax_matcher {
    matched_threshold: 0.5
    unmatched_threshold: 0.5
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 0.00039999998989515007
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.996999979019165
        scale: true
        epsilon: 0.0010000000474974513
      }
    }
    depth: 256
```

```
      num_layers_before_predictor: 4
      kernel_size: 3
      class_prediction_bias_init: -4.599999904632568
    }
  }
  anchor_generator {
    multiscale_anchor_generator {
      min_level: 3
      max_level: 7
      anchor_scale: 4.0
      aspect_ratios: 1.0
      aspect_ratios: 2.0
      aspect_ratios: 0.5
      scales_per_octave: 2
    }
  }
  post_processing {
    batch_non_max_suppression {
      score_threshold: 9.99999993922529e-09
      iou_threshold: 0.6000000238418579
      max_detections_per_class: 100
      max_total_detections: 100
      use_static_shapes: false
    }
    score_converter: SIGMOID
  }
  normalize_loss_by_num_matches: true
  loss {
    localization_loss {
      weighted_smooth_l1 {
      }
    }
    classification_loss {
      weighted_sigmoid_focal {
        gamma: 2.0
        alpha: 0.25
      }
    }
    classification_weight: 1.0
    localization_weight: 1.0
  }
  encode_background_as_zeros: true
  normalize_loc_loss_by_codesize: true
  inplace_batchnorm_update: true
  freeze_batchnorm: false

}
train_config {
  batch_size: 6 # Increase/Decrease this value depending on the available memory (Higher values require more memory and vice-versa)
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
```

```
149      learning_rate {
150        cosine_decay_learning_rate {
151          learning_rate_base: 0.03999999910593033
152          total_steps: 3000
153          warmup_learning_rate: 0.013333000242710114
154          warmup_steps: 1000
155        }
156      }
157      momentum_optimizer_value: 0.8999999761581421
158    }
159 use_moving_average: false
160 }
161 fine_tune_checkpoint: "pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt-0" # Path to checkpoint of pre-trained model
162  num_steps: 25000
163  startup_delay_steps: 0.0
164  replicas_to_aggregate: 8
165  max_number_of_boxes: 100
166  unpad_groundtruth_tensors: false
167  fine_tune_checkpoint_type: "detection" # Set this to "detection" since we want to be training the full detection model
168  use_bfloat16: false # Set this to false if you are not training on a TPU
169  fine_tune_checkpoint_version: V2
170}
171train_input_reader {
172  label_map_path: "annotations/label_map.pbtxt" # Path to label map file
173  tf_record_input_reader {
174    input_path: "annotations/train.record" # Path to training TFRecord file
175  }
176}
177eval_config {
178  metrics_set: "coco_detection_metrics"
179  use_moving_averages: false
180}
181eval_input_reader {
182  label_map_path: "annotations/label_map.pbtxt" # Path to label map file
183  shuffle: false
184  num_epochs: 1
185  tf_record_input_reader {
186    input_path: "annotations/test.record" # Path to testing TFRecord
187  }
188}
```

It is worth noting here that the changes to lines `178` to `179` above are optional. These should only be used if you installed the COCO evaluation tools, as outlined in the COCO API installation section, and you intend to run evaluation (see Evaluating the Model (Optional)).

Once the above changes have been applied to our config file, go ahead and save it.

Training the Model

Before we begin training our model, let's go and copy the `TensorFlow/models/research/object_detection/model_main_tf2.py` script and paste it straight into our `training_demo` folder. We will need this script in order to train our model.

Now, to initiate a new training job, open a new *Terminal*, `cd` inside the `training_demo` folder and run the following command:

```
python model_main_tf2.py --model_dir=models/my_model-
pipeline_config_path=models/my_model/pipeline.config
```

Once the training process has been initiated, you should see a series of print outs similar to the one below (plus/minus some warnings):

```
...
WARNING:tensorflow:Unresolved object in checkpoint:
(root).model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.4.10.gamma
W0716 05:24:19.105542 1364 util.py:143] Unresolved object in checkpoint:
(root).model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.4.10.gamma
WARNING:tensorflow:Unresolved object in checkpoint:
(root).model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.4.10.beta
W0716 05:24:19.106541 1364 util.py:143] Unresolved object in checkpoint:
(root).model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.4.10.beta
WARNING:tensorflow:Unresolved object in checkpoint:
(root).model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.4.10.moving_mean
W0716 05:24:19.107540 1364 util.py:143] Unresolved object in checkpoint:
(root).model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.4.10.moving_mean
WARNING:tensorflow:Unresolved object in checkpoint:
(root).model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.4.10.moving_variance
W0716 05:24:19.108539 1364 util.py:143] Unresolved object in checkpoint:
(root).model._box_predictor._base_tower_layers_for_heads.class_predictions_with_background.4.10.moving_variance
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore or tf.keras.Model.load_weights) but not
all checkpointed values were used. See above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(...).expect_partial(), to silence these warnings, or use assert_consumed() to make the check
explicit. See https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
W0716 05:24:19.108539 1364 util.py:151] A checkpoint was restored (e.g. tf.train.Checkpoint.restore or
tf.keras.Model.load_weights) but not all checkpointed values were used. See above for specific issues. Use expect_partial()
on the load status object, e.g. tf.train.Checkpoint.restore(...).expect_partial(), to silence these warnings, or use
assert_consumed() to make the check explicit. See https://www.tensorflow.org/guide/checkpoint#loading_mechanics for
details.
WARNING:tensorflow:num_readers has been reduced to 1 to match input file shards.
INFO:tensorflow:Step 100 per-step time 1.153s loss=0.761
I0716 05:26:55.879558  1364 model_lib_v2.py:632] Step 100 per-step time 1.153s loss=0.761
...
```

Important

The output will normally look like it has “frozen”, but DO NOT rush to cancel the process. The training outputs logs only every 100 steps by default, therefore if you wait for a while, you should see a log for the loss at step 100.

The time you should wait can vary greatly, depending on whether you are using a GPU and the chosen value for `batch_size` in the config file, so be patient.

Note

Training times can be affected by a number of factors such as:

- The computational power of you hardware (either CPU or GPU): Obviously, the more powerful your PC is, the faster the training process.
- Whether you are using the TensorFlow CPU or GPU variant: In general, even when compared to the best CPUs, almost any GPU graphics card will yield much faster training and detection speeds. As a matter of fact, when I first started I was running TensorFlow on my *Intel i7-5930k* (6/12 cores @ 4GHz, 32GB RAM) and was getting step times of around *12 sec/step*, after which I installed TensorFlow GPU and training the very same model -using the same dataset and config files- on a *EVGA GTX-770* (1536 CUDA-cores @ 1GHz, 2GB VRAM) I was down to *0.9 sec/step*!!! A 12-fold increase in speed, using a "low/mid-end" graphics card, when compared to a "mid/high-end" CPU.
- The complexity of the objects you are trying to detect: Obviously, if your objective is to track a black ball over a white background, the model will converge to satisfactory levels of detection pretty quickly. If on the other hand, for example, you wish to detect ships in ports, using Pan-Tilt-Zoom cameras, then training will be a much more challenging and time-consuming process, due to the high variability of the shape and size of ships, combined with a highly dynamic background.
- And many, many, many, more….

Evaluating the Model (Optional)

By default, the training process logs some basic measures of training performance. These seem to change depending on the installed version of Tensorflow.

Exporting a Trained Model

Once your training job is complete, you need to extract the newly trained inference graph, which will be later used to perform the object detection. This can be done as follows:

- Copy the `TensorFlow/models/research/object_detection/exporter_main_v2.py` script and paste it straight into your `training` folder.
- Now, open a *Terminal,* `cd` inside your `training` folder, and run the following command:

```
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\models\my_ssd_resnet50_v1_fpn\pipeline.config --trained_checkpoint_dir .\models\my_model\ --
output_directory .\exported-models\my_model
```

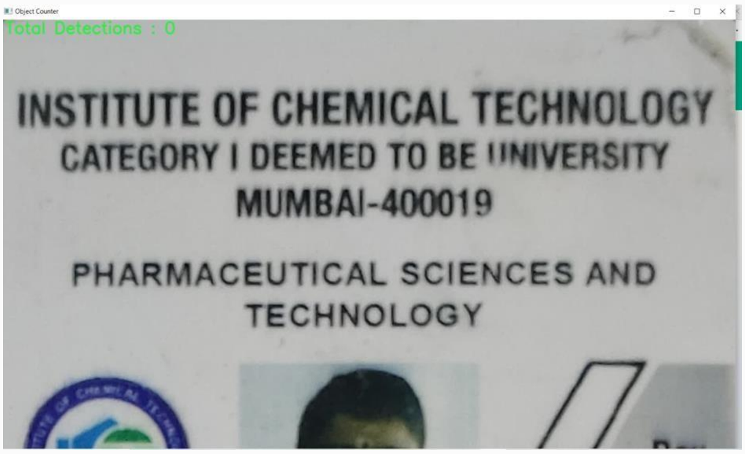
After the above process has completed, you should find a new folder `my_model` under the `training/exported-models` , that has the following structure:

```
training_demo/
├─ ...
├─ exported-models/
│  └─ my_model/
│     ├─ checkpoint/
│     ├─ saved_model/
│     └─ pipeline.config
└─ ...
```

This model can then be used to perform inference.

STEP 4: Verification of Document

For the purpose of our system, we will be feeding the training and testing images to a pretrained model and configure it accordingly. We will then use the pretrained model to verify whether the given ID card belongs VIT Chennai. The ID cards belonging to VIT Chennai were then stored in the Windows File System. The entire directory consisting of all the verified ID Cards is read by the OCR program.Since the above card does not belong to VIT Chennai, the object detection model could not identify a single feature and therefore the above ID Card was not sent for OCR Verification.



Text Detection and Extraction using OpenCV and OCR

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. OpenCV in python helps to process an image and apply various functions like resizing image, pixel manipulations, object detection, etc. In this article, we will learn how to use contours to detect the text in an image and save it to a text file.

Required Installations:

pip install opencv-python

pip install pytesseract

OpenCV package is used to read an image and perform certain image processing techniques. Python-tesseract is a wrapper for Google's Tesseract-OCR Engine which is used to recognize text from images.

After the necessary imports, a sample image is read using the **imread** function of opencv.

Applying image processing for the image:

The colorspace of the image is first changed and stored in a variable. For color conversion we use the function cv2.cvtColor(input_image, flag). The second parameter flag determines the type of conversion. We can chose among **cv2.COLOR_BGR2GRAY** and **cv2.COLOR_BGR2HSV**. cv2.COLOR_BGR2GRAY helps us to convert an RGB image to gray scale image and cv2.COLOR_BGR2HSV is used to convert an RGB image to HSV (Hue, Saturation, Value) color-space image. Here, we use **cv2.COLOR_BGR2GRAY**. A threshold is applied to the converted image using cv2.threshold function.

There are 3 types of thresholding:

1. Simple Thresholding

2. Adaptive Thresholding

3. Otsu's Binarization

For more information on thresholding, refer Thresholding techniques using OpenCV.
cv2.threshold() has 4 parameters, first parameter being the color-space changed image, followed by the minimum threshold value, the maximum threshold value and the type of thresholding that needs to be applied.

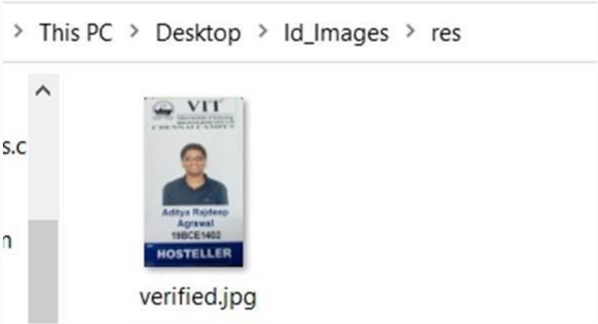
To get a rectangular structure:

cv2.getStructuringElement() is used to define a structural element like elliptical, circular, rectangular etc. Here, we use the rectangular structural element (cv2.MORPH_RECT). cv2.getStructuringElement takes an extra **size of the kernel** parameter. A bigger kernel would make group larger blocks of texts together. After choosing the correct kernel, dilation is applied to the image with cv2.dilate function. Dilation makes the groups of text to be detected more accurately since it **dilates** (expands) a text block.

Finding Contours:

cv2.findContours() is used to find contours in the dilated image. There are three arguments in cv.findContours(): the source image, the contour retrieval mode and the contour approximation method.
This function returns contours and hierarchy. Contours is a python list of all the contours in the image. Each contour is a Numpy array of (x, y) coordinates of boundary points in the object. Contours are typically used to find a white object from a black background. All the above image processing techniques are applied so that the Contours can detect the boundary

edges of the blocks of text of the image. A text file is opened in write mode and flushed. This text file is opened to save the text from the output of the OCR.



The picture shows the directory in which the verified image was stored after the object detection process by the TensorFlow API

Applying OCR:

Loop through each contour and take the x and y coordinates and the width and height using the function cv2.boundingRect(). Then draw a rectangle in the image using the function cv2.rectangle() with the help of obtained x and y coordinates and the width and height. There are 5 parameters in the cv2.rectangle(), the first parameter specifies the input image, followed by the x and y coordinates (starting coordinates of the rectangle), the ending coordinates of the rectangle which is (x+w, y+h), the boundary color for the rectangle in RGB value and the size of the boundary. Now crop the rectangular region and then pass it to the tesseract to extract the text from the image. Then we open the created text file in append mode to append the obtained text and close the file

STEP 5: Extracting Data using OCR

After it is verified that the submitted document is an ID card then information present on ID will be extracted by the means of Optical Character Recognition (OCR). That program extracts the Register No and Residence Status from the text given in the ID Card and compares it with the records in the Database

Registration number :- XXXXXXXXX

Residence Status :- XXXXXXXX

The output will show Done if the image is successfully processed and a black message(won't show anything) if process failed.

```
(image) E:\image\project\workspace\training>python TF-image-object-counting1.py
Loading model...Done! Took 9.066720008850098 seconds
Running inference for C:/Users/Aditya/Desktop/Id_Images/eval/aditya_id.jpg... Image Sent for OCR Verification
Done
```

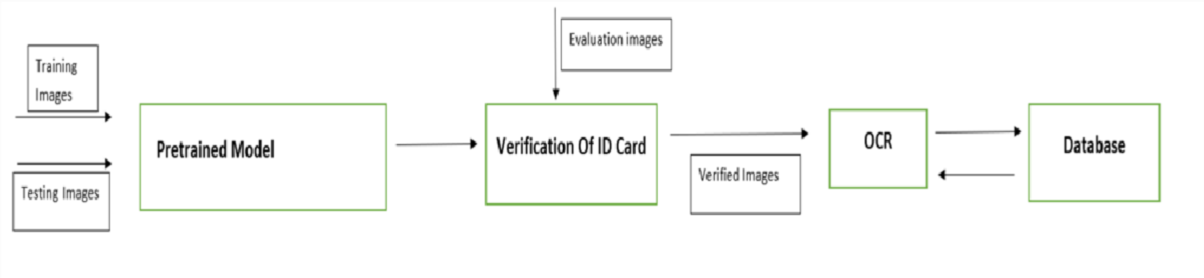
STEP 6: Validation of Data extracted using OCR

This information will be processed and validated by using an existing database. Registration number, Accommodation detail will be verified by comparing it with customer records available in the database.

Since the above card belongs to VIT Chennai, the object detection model identified and therefore the above ID Card was sent for OCR Verification. The OCR verified the Register No and residence status from the database and confirmed the validity of the ID Card.

```
(ocr) E:\ocr>python multiocr.py
Valid Record
```

Proposed System



5. Results and Discussion

The object detection model worked well enough to differentiate between VIT and Non-VIT ID cards. One problem we faced was that the object detection model couldn't take multiple evaluation images as input and we had to manually feed different images. As an extension of the problem just mentioned, we had to delete the verified image after one iteration of the system so that for the subsequent iterations, the directory where the verified ID cards are stored is empty.

6. Conclusion and Future Work

The entire project has been developed from the requirements to a complete system alongside evaluation and testing. The system developed has achieved its aim and objectives. More careful analysis is required on a project intrinsically. The ways used may be combined with others to attain nice results. Completely different ways are enforced within the past in keeping with the literature review. The conclusion to set the parameters of this part of the system based on a very small class size was due to the failures obtained from the recognition part of the system. The size of the image is very important in face recognition as every pixel counts. The algorithm we've used would be analyzed with images of different sizes. The basic idea is, no matter how the image is turned, we should be able to center the vit logo in roughly the same position in the image, thus recognizing the logo and verifying thee document.

In future work, we need to improve detection effectiveness with the help of the interaction among our system, and the user. We will improve this technique by working on different aspects like :

- Can improve security by linking ID Card to Application Number(Number given to student during admission) of student.
- Can use Neural Network for high accuracy.
- Can be use a large dataset.
- Can build on a fully web-based system.

7. REFERENCES

[1] CRISTIAN WISULTSCHEW,GABRIEL MUJICA,JOSE MANUEL LANZA-GUTIERREZ,JORGE PORTILLA ,et al. 3D-LIDAR Based Object Detection and Tracking on the Edge of IoT for Railway Level Crossing. March 2021

[2] Diandian Zhang,Yan Liu,Zhuowei Wang,Depei Wang et al. OCR with the Deep CNN Model for Ligature Script-Based Languages like Manchu. June 2021

[3] XIN ZHANG,LIANGXIU HAN,MARK ROBINSON2,ANTHONY GALLAGHER3 et al. A Gans-Based Deep Learning Framework for Automatic Subsurface Object Recognition From Ground Penetrating Radar Data. March 2021

[4] QIUYING HUANG, ZHANCHUAN CAI ,TING LAN et al. A Single Neural Network for Mixed Style License Plate Detection and Recognition. February 2021

[5] HASSANIN M. AL-BARHAMTOSHY,KAMAL M. JAMBI, SHERIF M. ABDOU3 ,MOHSEN A. RASHWAN4 et al. Arabic Documents Information Retrieval for Printed, Handwritten, and Calligraphy Image. April 2021

[6] FAHAD ASHIQ,MUHAMMAD ASIF,MAAZ BIN AHMAD,SADIA ZAFAR,KHALID MASOOD,TOQEER MAHMOOD,MUHAMMAD TARIQ MAHMOOD,IK HYUN LEE et al. CNN-Based Object Recognition and Tracking System to Assist Visually Impaired People. February 2022

[7] ALEKSANDAR JEVREMOVIC,MLADEN VEINOVIC,MILAN CABARKAPA,MARKO KRSTIC, IVAN CHORBEV,IVICA DIMITROVSKI,NUNO GARCIA,NUNO POMBO,MILOS STOJMENOVIC et al. Keeping Children Safe Online With Limited Resources: Analyzing What is Seen and Heard. September 2021

[8] TAYYAB NASIR,MUHAMMAD KAMRAN MALIK,KHURRAM SHAHZAD et al. MMU-OCR-21: Towards End-to-End Urdu Text Recognition Using Deep Learning. September 2021

[9] Hae Gwang Park,Jong Pil Yun,Min Young Kim,Seung Hyun Jeong et al. Multichannel Object Detection for Detecting Suspected Trees With Pine Wilt Disease Using Multispectral Drone Imagery.2021

[10] SHASHA LI,YONGJUN LI,YAO LI,MENGJUN LI,XIAORONG XU. YOLO-FIRI: Improved YOLOv5 for Infrared Image Object Detection. October 2021

APPENDIX :-

Multiocr.py

```
import pytesseract

import cv2

import re

import mysql.connector

import glob

pytesseract.pytesseract.tesseract_cmd = 'E:/Tesseract-OCR/tesseract.exe'

images = []

for image in glob.glob("C:/Users/Aditya/Desktop/Id_Images/res/*.jpg"):

    images.append(cv2.imread(image,1))

db =
mysql.connector.connect(host="localhost",user="aditya",password="root1234",database="company")

for i in images:
```

```
    test = pytesseract.image_to_string(i)
    x = re.search("19B[B-E][B-E][0-9]+",test)
    x1 = re.search("HOSTELLER",test)
    cursor = db.cursor()
    cursor.execute("SELECT * FROM student_details")
    res = cursor.fetchall()
    for record in res:
        if record[1]==x.group() and record[4]==x1.group():
            print("Valid Record")
```

TF-image-object-counting1.py

```
#!/usr/bin/env python
# coding: utf-8
"""
Object Detection (On Image) From TF2 Saved Model
=====================================
"""


import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging (1)
import pathlib
import tensorflow as tf
import cv2
import argparse
```

```
tf.get_logger().setLevel('ERROR')           # Suppress TensorFlow logging (2)


parser = argparse.ArgumentParser()
parser.add_argument('--model', help='Folder that the Saved Model is Located In',
                    default='exported-models/my_mobilenet_model')
parser.add_argument('--labels', help='Where the Labelmap is Located',
                    default='exported-models/my_mobilenet_model/saved_model/label_map.pbtxt')
parser.add_argument('--image', help='Name of the single image to perform detection on',
                    default='images/test/i-1e092ec6eabf47f9b85795a9e069181b.jpg')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected
objects',
                    default=0.5)


args = parser.parse_args()
# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)


# PROVIDE PATH TO IMAGE DIRECTORY
IMAGE_PATHS = 'C:/Users/Aditya/Desktop/Id_Images/eval/manish_id.jpg'




# PROVIDE PATH TO MODEL DIRECTORY
PATH_TO_MODEL_DIR = 'E:/image/project/workspace/training/exported-models/my_model'
```

```
# PROVIDE PATH TO LABEL MAP

PATH_TO_LABELS = 'E:/image/project/workspace/training/annotations/label_map.pbtxt'


# PROVIDE THE MINIMUM CONFIDENCE THRESHOLD

MIN_CONF_THRESH = float(0.3)


# LOAD THE MODEL


import time

from object_detection.utils import label_map_util

from object_detection.utils import visualization_utils as viz_utils


PATH_TO_SAVED_MODEL = PATH_TO_MODEL_DIR + "/saved_model"


print('Loading model...', end='')

start_time = time.time()


# LOAD SAVED MODEL AND BUILD DETECTION FUNCTION

detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)


end_time = time.time()

elapsed_time = end_time - start_time

print('Done! Took {} seconds'.format(elapsed_time))
```

```
# LOAD LABEL MAP DATA FOR PLOTTING

category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                    use_display_name=True)
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')  # Suppress Matplotlib warnings

def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.
    Puts image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
    (height, width, channels), where channels=3 for RGB.
    Args:
      path: the file path to the image
    Returns:
      uint8 numpy array with shape (img_height, img_width, 3)
    """
    return np.array(Image.open(path))




print('Running inference for {}... '.format(IMAGE_PATHS), end='')
```

```
image = cv2.imread(IMAGE_PATHS)

image_rgb = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

imH, imW, _ = image.shape

image_expanded = np.expand_dims(image_rgb, axis=0)


# The input needs to be a tensor, convert it using `tf.convert_to_tensor`.

input_tensor = tf.convert_to_tensor(image)

# The model expects a batch of images, so add an axis with `tf.newaxis`.

input_tensor = input_tensor[tf.newaxis, ...]


# input_tensor = np.expand_dims(image_np, 0)

detections = detect_fn(input_tensor)


# All outputs are batches tensors.

# Convert to numpy arrays, and take index [0] to remove the batch dimension.

# We're only interested in the first num_detections.

num_detections = int(detections.pop('num_detections'))

 detections = {key: value[0, :num_detections].numpy()

              for key, value in detections.items()}

detections['num_detections'] = num_detections


# detection_classes should be ints.

detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

scores = detections['detection_scores']

boxes = detections['detection_boxes']
```

```
classes = detections['detection_classes']

count = 0

for i in range(len(scores)):

    if ((scores[i] > MIN_CONF_THRESH) and (scores[i] <= 1.0)):

        #increase count

        count += 1

        # Get bounding box coordinates and draw box

        # Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()

        ymin = int(max(1,(boxes[i][0] * imH)))

        xmin = int(max(1,(boxes[i][1] * imW)))

        ymax = int(min(imH,(boxes[i][2] * imH)))

        xmax = int(min(imW,(boxes[i][3] * imW)))



        cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

        # Draw label

        object_name = category_index[int(classes[i])]['name'] # Look up object name from "labels" array using class index

        label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example: 'person: 72%'

        labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size

        label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too close to top of window

        cv2.rectangle(image, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in

        cv2.putText(image, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text
```

```
if count==2:
    cv2.imwrite('C:/Users/Aditya/Desktop/Id_Images/res/verified.jpg',image)
    print("Image Sent for OCR Verification")


cv2.putText (image,'Total Detections : ' +
str(count),(10,25),cv2.FONT_HERSHEY_SIMPLEX,1,(70,235,52),2,cv2.LINE_AA)
print('Done')
# DISPLAYS OUTPUT IMAGE
cv2.imshow('Object Counter', image)
# CLOSES WINDOW ONCE KEY IS PRESSED
cv2.waitKey(0)
# CLEANUP
cv2.destroyAllWindows()
```